POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Systems and Methods for Big and Unstructured Data Project

Author(s): **Niccolò Bindi**

Matteo Forlivesi

Lorenzo Franzè

Aleksandro Aliaj

Renato Gallicola

Group Number: **36**

Academic Year: 2022-2023

Contents

I First delivery **4**

1 Description of the problem **6**

2 Assumptions **7**

3 ER Description **8**

4 Dataset Description **11**

5 Data pre-processing and parsing **14**

6 Data Upload **16**
6.1 Assertions and constraints 17
6.2 Node upload 17
6.3 Attribute addition 20
6.4 Relationship addition 21

7 Graph Diagram **23**

II Second delivery **41**

8 Data shaping and document structure **43**
8.1 Document structure 43
8.2 Data description 44
8.3 Example of documents 45

9 Data extraction and preprocessing **47**

10 Data upload **52**

11 Alternative database generation **55**

12 Queries **57**

III Third delivery **82**

13 Description of the structure of the dataset in pyspark **84**
13.1 Dataset structure 84
13.2 Data description 85

14 Python script **86**
14.1 Script used to transform the data in the pyspark format 86

15 Queries **90**

Part I

First delivery

1 | Description of the problem

The aim of this project is to work on a dataset that contains records of various scientific publications in the field of Computer Science. The publications in the dataset include books, articles, thesis and others.

The first step is to get the dataset from its publisher (dblp.org) and study the documentation that comes with it to understand its structure. This poses many challenges as the data can be inconsistent at times, so we need to make some assumptions (better explained in the next paragraph). The results would then be graphically represented with a ER model.

Following this first phase, we want to better explore this dataset with the tools that we know:

- Graph DB (Neo4j)
- lxml (Python library)
- draw.io
- StAX (Java library)
- Document DB (MongoDB)

In the first part of the project, we will concentrate on Neo4j. To do this, first we need to convert the dataset from the XML format to CSV, as the former is not supported by Neo4j. Then we import the whole dataset. Now that we are all set, we can make queries of different complexity to gain insight of the data, while also keeping an eye on performance.

2 | Assumptions

We have found a lot of conventions and micro-syntax rules during our data exploration on DBLP records, for these reasons we can define the following assumptions on:

- **Record Attributes**
  - *key*: composed of three parts. The first can be "conf" (conference or workshop papers) or "journals" (articles published in journals, transactions, magazines or newsletters) and the second designates the conference series or periodical the papers appeared in. The last part can be alphanumerical characters, for this you cannot make assumptions about the last part but neither in general because it can contain controversial information also in the previous parts since the world of publications does not form a hierarchy.
- **Author**
  - Due to the lack of DOI's presence in all records we cannot use it as a primary key.
  - Due to the lack of ORCID's presence in all records we cannot use it as a primary key.
  - If people change names a new profile is created, and they are identified as a new person but with queries on ids like ORCID you can still unite them.
- **Title**
  - Must exist
- **Years**
  - The year of a publication Is always present with a number to be interpreted according to the Gregorian calendar
- **Crossref**
  - The crossref field in the inproceedings record (paper) contains the key of the proceedings record (volume). The proceedings records (and the crossref fields) are missing for a lot of legacy inproceedings records.

3 | ER Description

We want to design an ER model for storing relevant research in the Computer Science field, focusing on the different type of released articles and the people working on them.

Every **publication** can be divided into seven main categories according to the BibTex standard:

- **article** – An article from a journal or magazine.
- **proceedings** – The proceedings volume of a conference or workshop.
- **inproceedings** – A paper in a conference or workshop proceedings.
- **book** – An authored monograph or an edited collection of articles.
- **incollection** – A part or chapter in a monograph.
- **phdthesis** – A PhD thesis.
- **mastersthesis** – A Master's thesis.

Many inproceedings belong to one proceeding and the same goes for many incollection that belong to one book.

They both have a many to one relationship (possible thanks to the crossref field in the data of inproceedings/incollections).

Every single publication presents a series of common characteristics (not all compulsory):

- Key: it is the unique key of a record.
- Mdate: date of last modification compatible with the ISO standard 8601 YYYY-MM-DD
- Title: it is the only compulsory attribute of every record of these kinds.
- DOI: digital object identifier, it is a unique key associated with physical and digital objects that is present in almost every record.
- Author: it can contain one or multiple names stated in order of importance for the paper
  - **NOTE**: if the author is unknown the attribute is not present.
- Pages: it tells the starting and ending page of a record. The first page number is compulsory, but the end page can be omitted if not known. If the paper is only one page long only the starting page should be noted.

- **Year**: it contains a four-digit number representing the publication year. It might differ from the collection year.
- **Url**: it represents the position inside the table of contents, it might not be present in monographs or proceedings. It is usually a local address, but it can also be global.Up to 2
  - **NOTE**: in www publications if present represents an external link to the authors site
- **EE**: It represents the position of the electronic edition, usually it is a global address.Up to 2
- **Note**

Article can also contain:

- *Journal*: title of the journal in which the article appears
- *Volume*: number of the volume
- *Number*: number of the release specific to the volume

Proceedings can also contain:

- *Editor*: name and surname
- *Publisher*: name of a publishing company
- *ISBN*: international standard book number (unique identifier)
- *Series*
- *Volume*
- *Booktitle*: shortened version of a book title or a convention name

Inproceedings can also contain:

- *Booktitle*

Books can also contain:

- *Publisher*
- *Series*
- *ISBN*
- *School*

Phdthesis Mastersthesis can also contain:

- *School*

Incollections can also contain:

- *Booktitle*

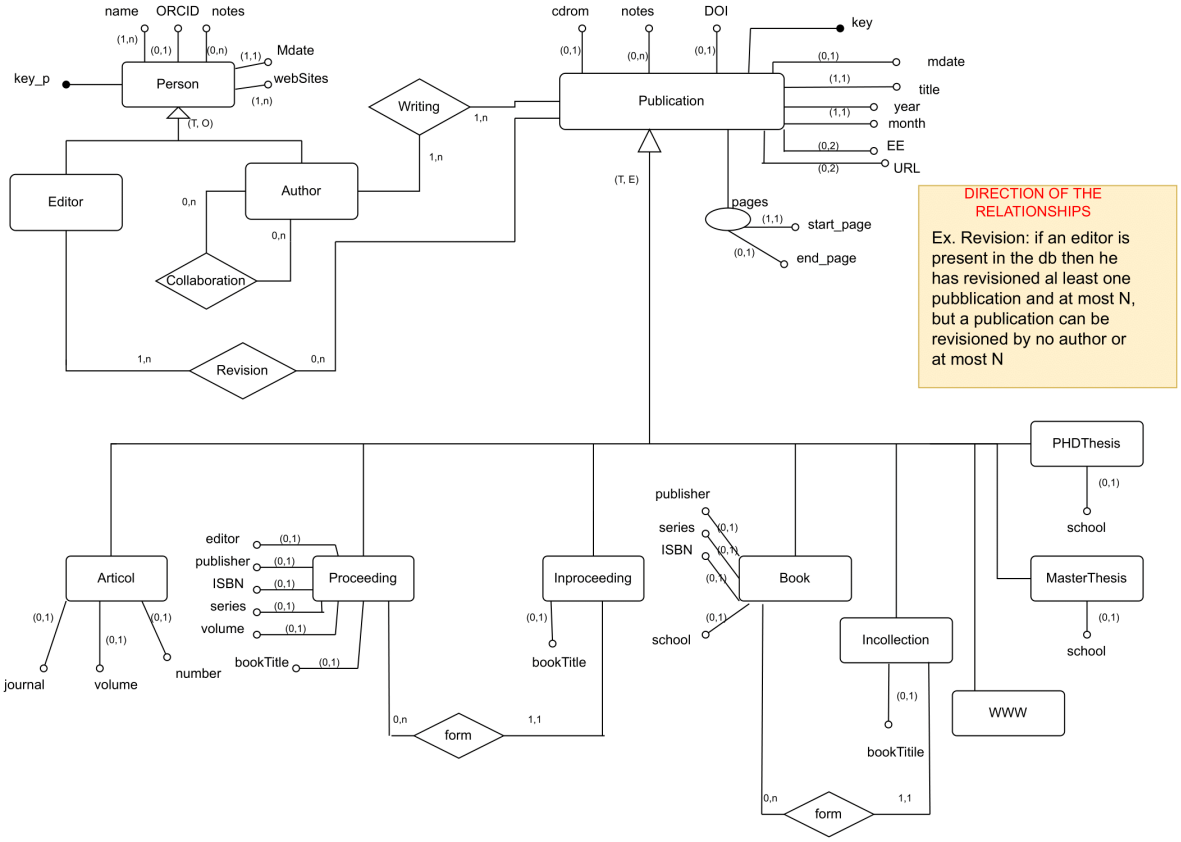
People are another focal point of the data. They are divided into authors and editors but one can be both at the same time. Authors can collaborate with each other to make publications of any sort. People are identified by a unique key present in the database.

Each person is linked to a publication by one of their names in the author tag or editor tag when present.

Each person has a list of personal websites.

Every person has usually the following traits:

- *ORCID*: a unique id of every author (independently of any name change), unfortunately not available for everyone so not fit to be a key.
- *Name*: a list of their names and surnames (due to synonyms of the name or name changes) with eventually a progressive number code starting at 0001 to differentiate between different homonyms (e.g. Marco Brambilla 0001 is different from Marco Brambilla 0007). Any name change will result in a additional name being added to this list.
- *Mdate*: date of last modification of the profile on the dblp database.
- *Notes*: a list of various infos like affiliations and awards.



4 | Dataset Description

- **Article**

	Author	Editor	Key	Mdate	Title	Pages
e.g.	Frank Manola		tr/gte/TR-0263-08-94-165	25/10/2019	An Evaluation of Object-Oriented DBMS Developments: 1994 Edition.	

	Year	Journal	Volume	Number	Month
e.g.	1994	GTE Laboratories Incorporated	TR-0263-08-94-165		August

	Url	EE	Cdrom	Note
e.g.	db/journals/gtelab/index.html#TR-0263-08-94-165			

- **Proceedings**

	Author	Editor	Key	Mdate	Title	Pages
e.g.		Ngoc Thanh Nguyen	journals/tcci/2014-14	14/05/2019	Transactions on Computational Collective Intelligence XIV	

	Year	Month	Url	EE	Cdrom	Note	Publisher
e.g.	2014		db/journals/tcci/tcci14.html				Springer

	ISBN	Series	Volume	Booktitle
e.g.		Lecture Notes in Computer Science	8615	Trans. Computational Collective Intelligence

- **Inproceedings**

	Author	Editor	Key	Mdate	Title	Pages
e.g.			journals/lncs/BollingerLP91	20/05/2017	The LILOG Inference Engine.	402-427

	Year	Month	Url	EE	Cdrom
e.g.	1991		db/journals/lncs/lncs546.html#BollingerLP91	https://doi.org/10.1007/3-540-54594-8_72	

	Note	Booktitle	Crossref
e.g.		Text Understanding in LILOG	djournals/lncs/1991-546

- **Book**

	Author	Editor	Key	Mdate	Title	Pages
e.g.	Peter A. Gloor		books/daglib/0068017	17/07/2021	Synchronisation in verteilten Systemen	I-XII, 1-227

	Year	Month	Url	EE	Cdrom	Note	Publisher
e.g.	1989			https://d-nb.info/890776466			Teubner

	ISBN	Series	School
e.g.	978-3-519-02494-1	Leitfäden der angewandten Informatik	University of Zurich, Zürich, Switzerland

- **Incollection**

	Author	Editor	Key	Mdate	Title	Pages
e.g.	David Naccache		reference/crypt/Naccache11i	12/07/2017	Naccache-Stern Higher Residues Cryptosystem.	829

	Year	Month	Url	EE	Cdrom
e.g.	2011		db/reference/crypt/crypt2011.html#Naccache11i	https://doi.org/10.1007/978-1-4419-5906-5_893	

	Note	Booktitle	Crossref
e.g.		Encyclopedia of Cryptography and Security (2nd Ed.)	reference/crypt/2011

- **Phdthesis**

	Author	Editor	Key	Mdate	Title	Pages
e.g.	Samuel Larsen		phd/ndltd/Larsen06	04/05/2022	Compilation techniques for short-vector instructions.	

	Year	Month	Url	EE
e.g.	2006			https://hdl.handle.net/1721.1/37890

	Cdrom	Note	School
e.g.		ndltd.org (oai:dspace.mit.edu:1721.1/37890)	Massachusetts Institute of Technology, Cambridge, MA, USA

- **Mastersthesis**

	Author	Editor	Key	Mdate	Title	Pages
e.g.	Oliver Hoffmann 0002		ms/Hoffmann2008	12/03/2020	Regelbasierte Extraktion und asymmetrische Fusion bibliographischer Informationen.	

	Year	Month	Url	EE
e.g.	2009			http://dblp.uni-trier.de/papers/DiplomarbeitOliverHoffmann.pdf

	Cdrom	Note	School
e.g.		Diplomarbeit, Universität Trier, FB IV, DBIS/DBLP	University of Trier

5 | Data pre-processing and parsing

After analyzing the structure of the dataset, we had to pre-process it in order to be able to import it in Neo4j. In fact, while the original dataset is in the XML format, Neo4j only accept CSV as input. So we wrote some simple Python scripts in order to convert the XML to CSV.

We made one script for every type of publication in the database (7 in total, as described in the former paragraph).

Every script is made up of mainly two parts: the first one is a parser that scan through the entire XML tree and finds all the elements with a certain tag (eg. articles). Then for every element found, the code looks for the tags that *may* be contained in a certain publication and stores the values that are found.

```
import lxml.etree as et
import csv

parser = et.XMLParser(attribute_defaults=True)
tree = et.parse("dblp.xml", parser)
root = tree.getroot()
```

Figure 5.1: Setting up the parser and finding the root of the XML file

The script does not assume that every attribute that a publication can have is actually present (apart from the tag "key" which is mandatory for every publication). If one attribute is missing, the script stores just an empty string.

The second part of the script uses a standard library provided by Python to work with CSV files. First it opens (or create) a new CSV file and writes the heading of the table.

It then takes all the attributes stored for a certain publication and writes them in a new line of the CSV file.

For the "author" and "editor" tag, we had to create a nested loop in order to write one line in the CSV for every author/editor of a certain publication (because a single publication can have more than one author or editor).

After repeating these steps for every type of publication (and adapting the attributes parsed to the structure of every type) we have a set of CSV files ready to be imported in Neo4j.

```
i = 0
for element in root.iter("article"):
    if i > 10000:
        break

    if element.attrib.get("key") is not None:
        key = element.attrib.get("key")
    else:
        key = ""
    if element.attrib.get("mdate") is not None:
        mdate = element.attrib.get("mdate")
    else:
        mdate = ""
    if element.find("title") is not None:
        title = element.find("title").text
    else:
        title = ""

    if element.find("year") is not None:
        year = element.find("year").text
    else:
        year = ""
    if element.find("url") is not None:
        url = element.find("url").text
    else:
        url = ""
    if element.find("note") is not None:
        note = element.find("note").text
```

Figure 5.2: Parsing the attributes of 10000 articles from the dataset.

```
csvfile = open('articles.csv', 'w', encoding='utf-8')
writer = csv.writer(csvfile)

writer.writerow(
    ["author", "editor", "key", "mdate", "title", "pages", "year", "journal",
    "volume", "number", "month", "url", "ee", "cdrom", "note"])
```

Figure 5.3: Opening the communication with the CSV file.

```
    for author in element.iterchildren(tag="author"):
        if author is not None:
            name = author.text
        if element.find("editor") is not None:
            for editor in element.iterchildren(tag="editor"):
                editorName = editor.text
                csv_line = [name, editorName, key, mdate, title, pages, year,
                            journal, volume, number, month, url, ee, cdrom,
                            note]
                writer.writerow(csv_line)
        else:
            editorName = ""
            csv_line = [name, editorName, key, mdate, title, pages, year,
                        journal, volume, number, month, url, ee, cdrom, note]
            writer.writerow(csv_line)
```

Figure 5.4: Parsing the "author" and "editor" tags, then writing a line in the CSV.

6 | Data Upload

The methodology we applied to upload data was a mechanical and strict set of passages that allowed us to stay true to the graph description we wanted to achieve as shown in the next chapter.

Given the CSV files obtained with the XML parsing we noticed we could not create a table that allowed us to get rows already containing the relationship information: we used Cypher to import the nodes and then we created the relationships we wanted.

Due to the enormity of that file (dblp.xml was about 3.5 GB uncompressed) we opted to upload 10000 entities of the following categories:

- **Books** from books.csv
- **Incollections** from incollections.csv
- **Articles** from articles.csv
- **Proceedings** from proceedings.csv
- **Inproceedings** from inproceedings.csv
- **Master's thesis** from mastersthesis.csv
- **Phd thesis** from phdthesis.csv

We followed a step by step process that involved:

- creating a node;
- filling them up with the attributes where they are available;
- connecting all the nodes with the relationships.

As an example we report the addition to the database of the rows of the document *proceedings.csv*.

The columns in that document are:

- editor;
- key;
- mdate;
- title;

- year;
- url;
- ee;
- publisher;
- series;
- volume;
- booktitle.

6.1. Assertions and constraints

Before importing any data,to make sure that the data import from the csv is coherent (it could have rows not conform to the graph structure that we want to impose) we make a set of assertion on existence and most importantly on uniqueness.

We impose that all the fields considered keys for the nodes we want to create are unique:

- *key* must be unique in Proceeding nodes;
- *editor* must be unique for Editor nodes because we identify people by name;
- *year* must be converted to an integer and unique being the key of node Year;
- *publisher* must be unique being key of nodes Publisher;
- *series* must be unique being key of nodes Serie.

We also want to impose that the attributes *mdate* and *title* are always present in every node Proceeding.

A similar procedure has been done before importing any of the seven csv's adapting the constraints to the fields present.

6.2. Node upload

First we upload the nodes,that is the entites that have connection with each other. From the csv we can upload:

- *editor* as Editor nodes;
- *key* as Proceeding nodes (as it is the key of them);
- *year* as a Year node;
- *publisher* as a Publisher node;
- *series* as a Serie node.

To make sure that the nodes fill up correctly we checked that each row had effectively present the key attribute for that node by using the construct "**WITH** row.*neededKey* as *alias* **WHERE** *alias* **is not null**"

Figure 6.1: Adding Editor Nodes

Since we knew that, in the xml file, proceedings always contained the attributes *mdate* and *title* because of documentation we initialized the Proceeding nodes with those attributes too.

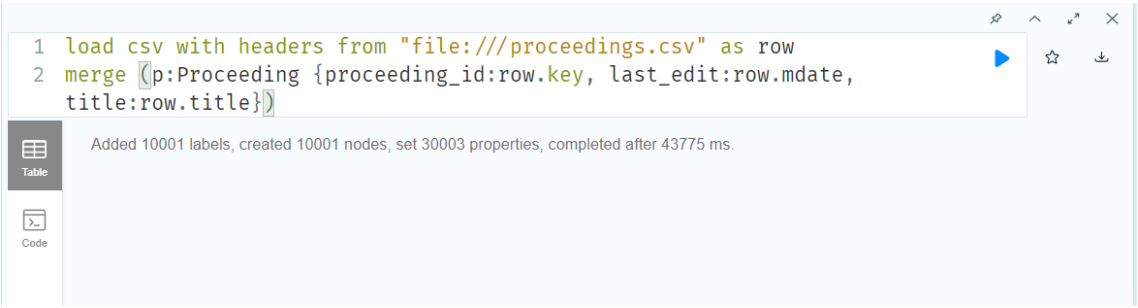


Figure 6.2: Adding Proceeding nodes

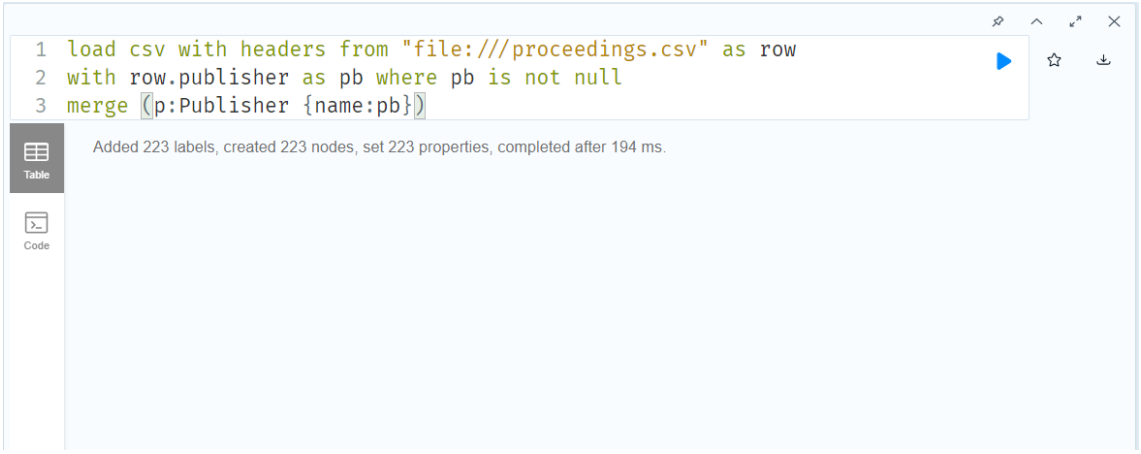


Figure 6.3: Adding Publisher nodes

Figure 6.4: Adding Year nodes

Figure 6.5: Adding Series nodes

6.3. Attribute addition

Secondly we make sure that all the attributes are added to the Proceeding nodes if and only if they exist in the csv.

We do that by cycling through the csv row by row and when the key attribute in the csv matched the key attribute in the node we check if the property is not empty in the csv row: in that case we add it.

Consequently we end up with nodes of the same type with different attributes (only key,mdate and title are common through all)

Figure 6.6: Setting the pages attribute

The same can be done for adding the volume, url and ee attributes.

6.4. Relationship addition

Finally we can build some relations between the nodes we just created and the ones already in the database because of previous uploads.

All the relationships present the same algorithm for upload:

- match node of type 1 that has his key property on a row of the csv;
- match node of type 2 that has his key property in the same row;
- merge a relation between the two with the eventual addition of a property on it.

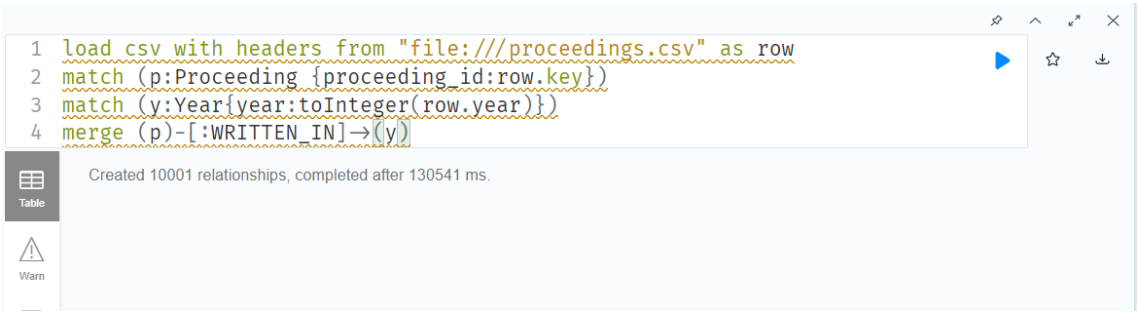


Figure 6.7: Relation WRITTEN-IN

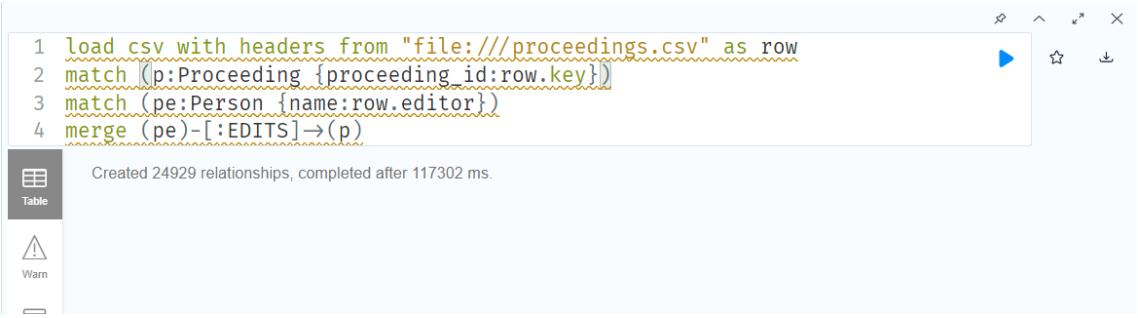


Figure 6.8: Relation EDITS

Figure 6.9: Relation PUBLISHES

Figure 6.10: Relation PART-OF-SERIES

7 | Graph Diagram

Nodes

- **Article**
- **Proceedings**
- **Inproceedings**
- **Book**
- **Incollection**
- **Phdthesis**
- **Mastersthesis**
- **Journal**: it is a collection of Articles
- **Series**: it is a collection of Books of Proceedings
- **Publisher**
- **School**: the school that has
- **Person**: they can be either an author or an editor
- **Year**: the year when a publication was written

Links

- **WRITTEN_IN**: relationship between a Publication and a Year that indicates when the publication was written
- **WRITES**: relationship between a Person and a Publication that links an author to their works
- **PART_OF_BOOK**: relationship that links an Incollection to the Book in which it is included
- **PART_OF_PROCEEDING**: relationship that links an Inproceeding to the Proceeding in which it is included

- **PART_OF_SERIES**: relationship between a Book or a Proceeding and the Series they appear in
- **APPEARS_IN**: relationship between an Article and the Journal it appears in. It has two attributes: Volume and Number, representing the journal's volume and number in which the article appears in, respectively.
- **AFFILIATION**: relationship between a Phdthesis, Mastersthesis or Book and the School that has collaborated on their realisation
- **PUBLISHES**: relationship between a Publisher and a Book or Proceeding that it has published
- **EDITS**: relationship between a Person and a Proceeding, showing that a Person is the editor of a Proceeding

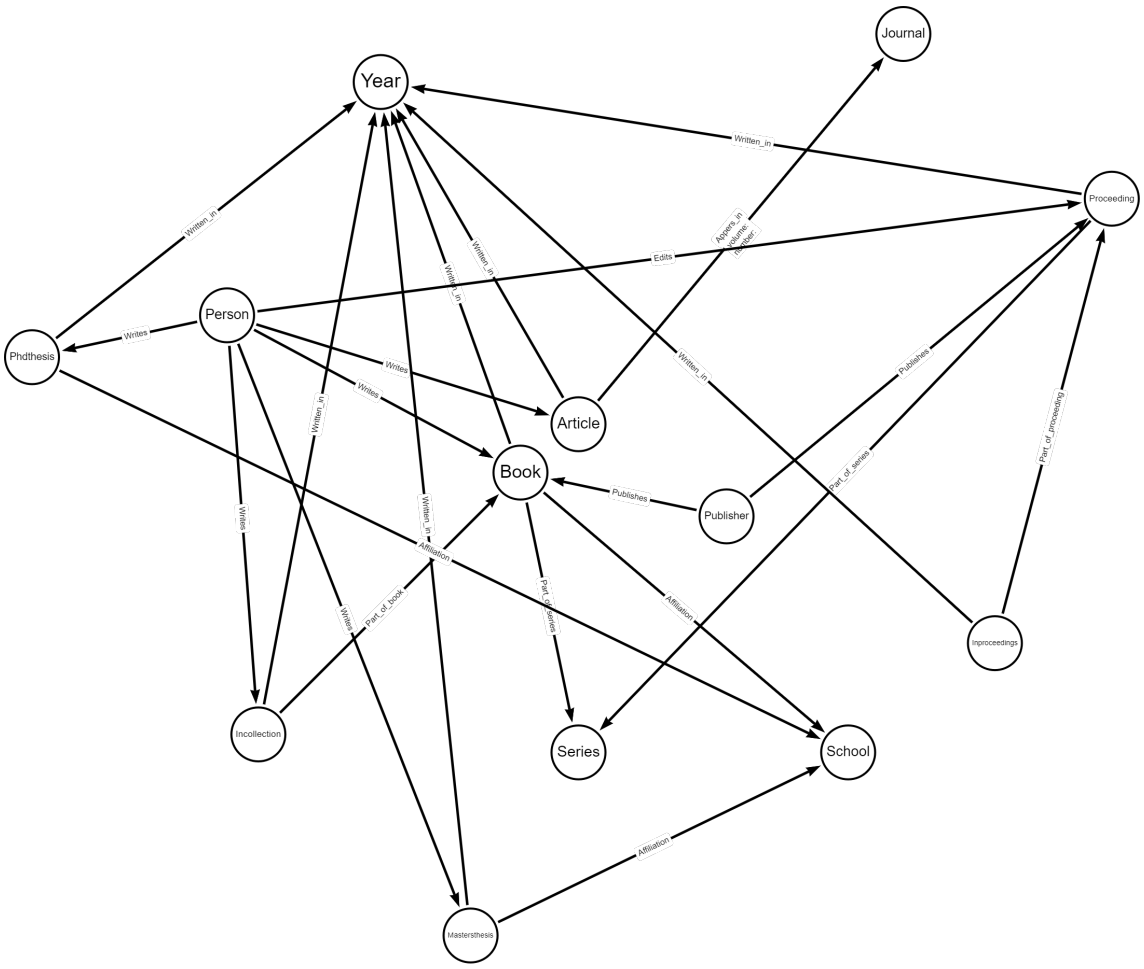


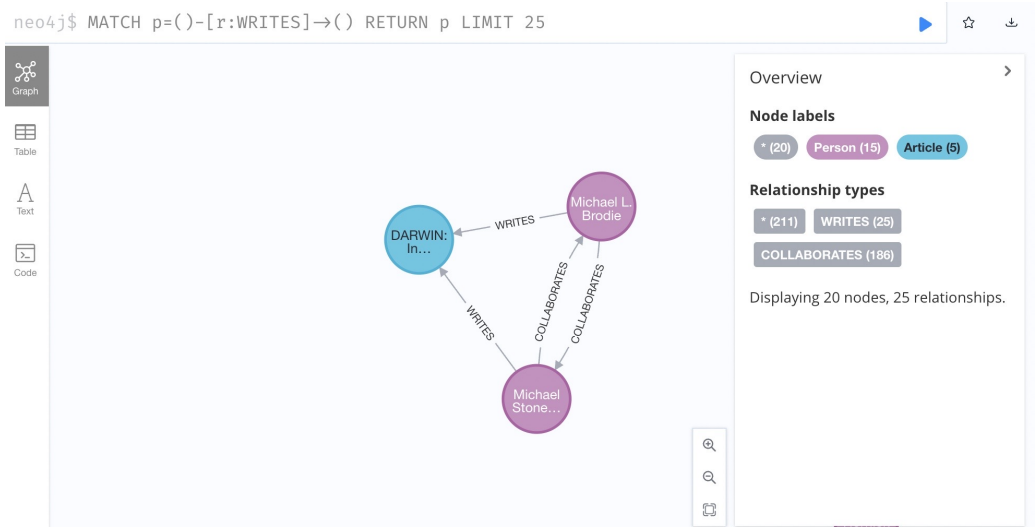
Figure 7.1: Picture of the diagram showing the nodes and how their are connected to each other

Queries

- **QUERY #1**

CREATE Relationship "COLLABORATES": is a relationship between 2 people who worked on the same Article (the same can be done with other types of publications or more generally by omitting the label Article). We added the property "name", which is the title of the article, to distinguish the collaboration of the same authors in different works.

Results: the query generates the right collaborations without creating self-rings



- **QUERY #2**

Tag existing articles by era (WRITE QUERY): "recent" are works written after the 2000s, "old" are works written between the years 1975 and 2000 and "ancient" are works written before 1975s. Using the library apoc we added the attribute 'era' and then we ordered the articles by last_edit. We can see that the articles were all edited after the 2000s but written also before the 1975s:

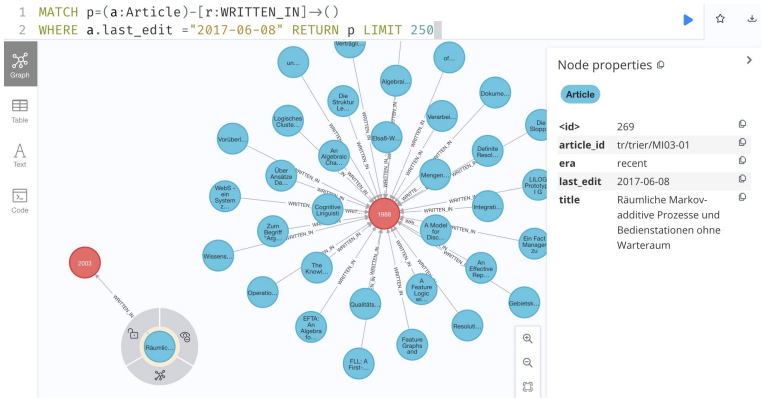
```
MATCH p=(a:Article)-[r:WRITTEN_IN]→(y:Year)
CALL apoc.do.case([
    y.year ⩾ 2000, 'SET a.era = "recent"
    RETURN a',
    y.year < 2000 AND y.year ⩾ 1975, 'SET a.era = "old"
    RETURN a'],
    'SET a.era = "ancient" RETURN a', {a:a}) YIELD value
RETURN value.a.title AS Title, value.a.last_edit AS Edited,
value.a.era AS Era ORDER BY Edited
```

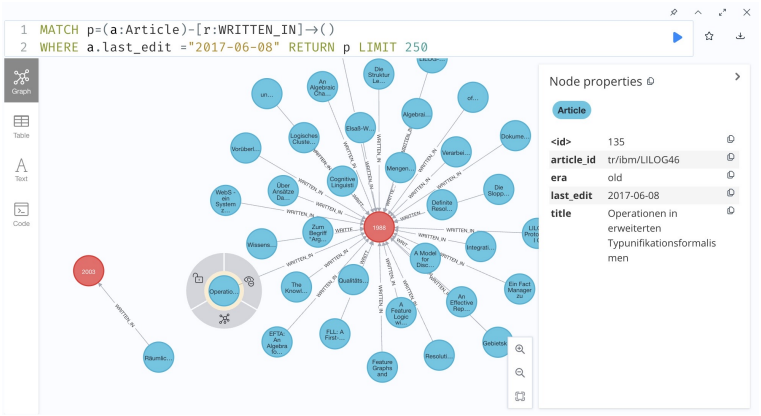
Table | Text | Code

"Title"	"Edited"	"Era"
"Common Subexpression Identification in General Algebraic Systems."	"2002-01-03"	"ancient"
"Muffin: A Distributed Database Machine"	"2017-06-08"	"old"
"Die Repräsentation räumlichen Wissens und die Behandlung von Einbettungsproblemen mit Quadtreedepiktionen"	"2017-06-08"	"old"
"Algebraical Optimization of FTA-Expressions"	"2017-06-08"	"old"
"Wissensrepräsentation und Maschinelles Lernen"	"2017-06-08"	"old"
"An Algebraic Characterization of STUF"	"2017-06-08"	"old"
"A Combined Symbolic-Empirical Apprach for the Automatic Translation of Compounds"	"2017-06-08"	"old"
"Zur Systemarchitektur von LILOG"	"2017-06-08"	"old"
"Mengenorientierte Auswertung von Anfragen in der Logikprogrammiersprache PROLOG"	"2017-06-08"	"old"

Results: we can see that, for example, the articles whit the attribute last_edit = "2017-06-08" may have been written in different "eras". The article with <id> = 135 is 'old' because is written in 1988, instead the other one <id> = 269 is 'recent' because is written in 2003.

```
MATCH p=(a:Article)-[r:WRITTEN_IN]→()
WHERE a.last_edit ="2017-06-08" RETURN p LIMIT 250
```





- **QUERY #3**

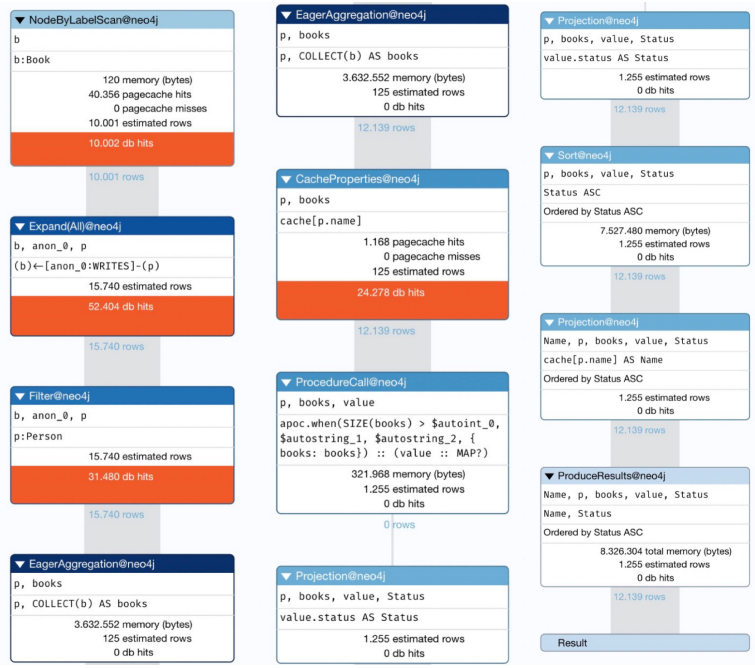
Classify people based on the number of written books: we have done a count on the books and when the size is greater than 5 the query returns "expert writer" as the status for the person, otherwise the person is "upcoming".

```
PROFILE
MATCH (p:Person)-[:WRITES]→(b:Book)
WITH DISTINCT p, COLLECT(b) AS books
CALL apoc.when(
    SIZE(books)>5, 'RETURN "Expert writer" AS status',
    'RETURN "Upcoming" AS status',
    {books:books}) YIELD value
RETURN p.name AS Name, value.status AS Status ORDER BY Status
```

	Name	Status
102	"C. J. Date 0001"	"Expert writer"
103	"Friedrich L. Bauer"	"Expert writer"
104	"Brian Carper"	"Upcoming"
105	"Christophe Grand"	"Upcoming"
106	"Chas Emerick"	"Upcoming"
107	"Louis Baker"	"Upcoming"

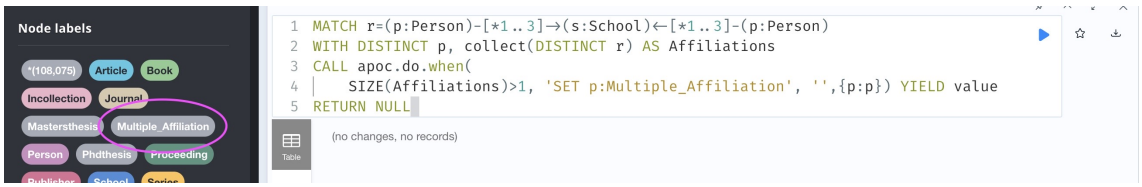
Started streaming 12139 records after 555 ms and completed after 559 ms, displaying first 1000 rows.

Results: Brian Carper is the first who appears that has written less than 6 different books, found with the following operations.

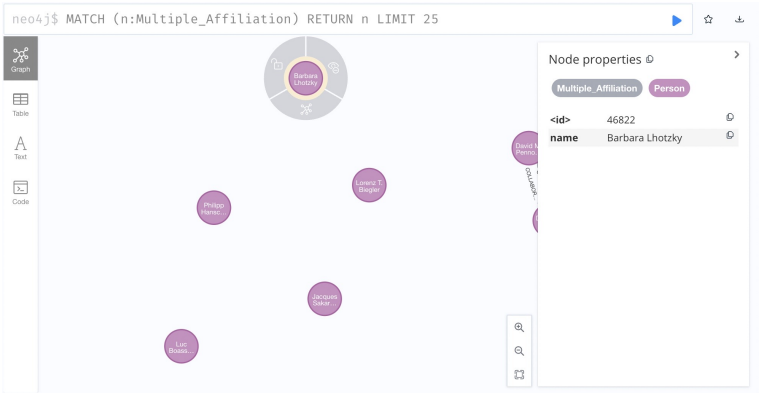


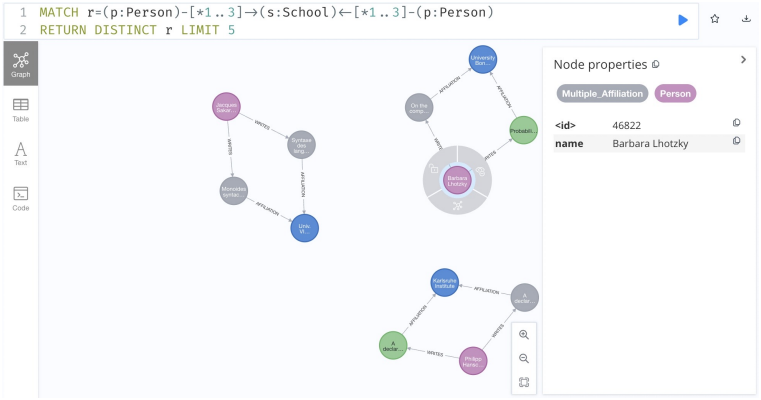
- **QUERY #4**

create new node label "MULTIPLE_AFFILIATION": after this query we have got the new label "MULTIPLE_AFFILIATION" to make sure that if we select a person we can see this new label attached to him, that indicates people affiliated at least one time with the same school.



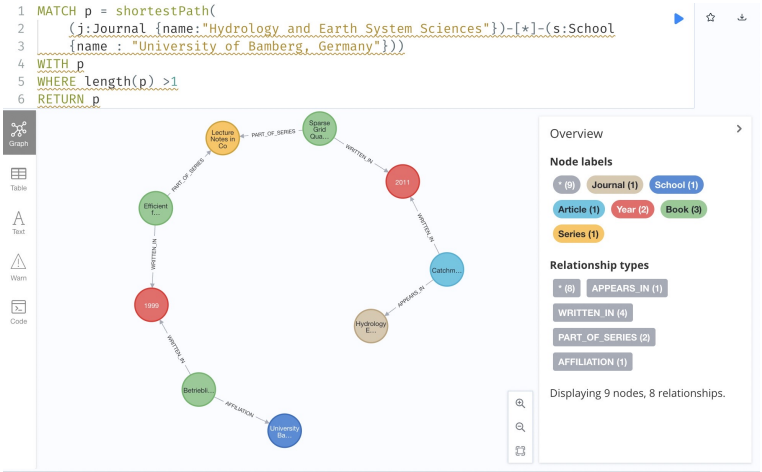
Results: we can see that Barbara Lhotzky, who is affiliated 2 times with the same University but with different works appears if we look for Multiple_Affiliation nodes.



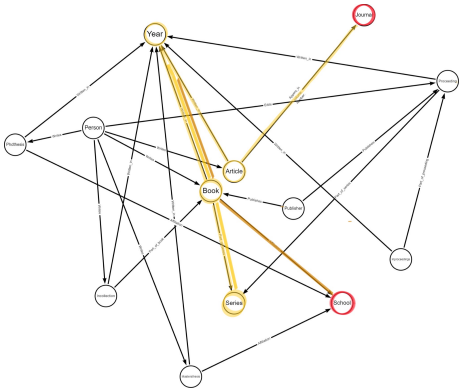


- **QUERY #5**

  Shortest Path: with this query we wanted to find the shortest path between a Journal and a School using the Fast Algorithm (not guaranteed to return a result, even if exists).



Results: 9 nodes, 8 relationships. we have schematized the solution within the entire database.



- **QUERY #6**

Recommendation query: find People that "Frank Manola" hasn't yet worked with, but his collaborators have. We found someone who can introduce Frank to his potential collaborators. We matched "Frank Manola" collaborators and then people who collaborated with his collaborators. We checked that co-Collaborators didn't work with Frank and in the end we ordered the results counting the number of times the co-Collaborator is found.

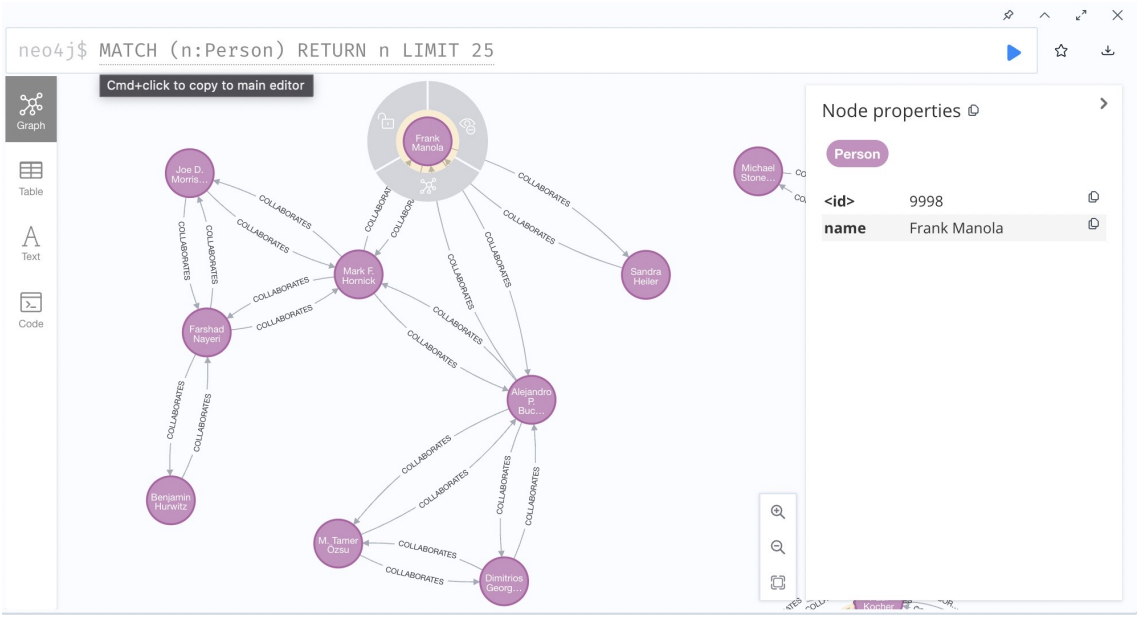
```
1 PROFILE
2 MATCH (p:Person {name:"Frank Manola"})-[:WRITES]→(m)←[:WRITES]-(collaborators),
3 (collaborators)-[:WRITES]→(m2)←[:WRITES]-(coCollaborators)
4 WHERE NOT (p)-[:WRITES]→()←[:WRITES]-(coCollaborators) AND p <> coCollaborators
5 RETURN coCollaborators.name AS Recommended, count(*) AS Strenght ORDER BY Strenght
  DESC
```

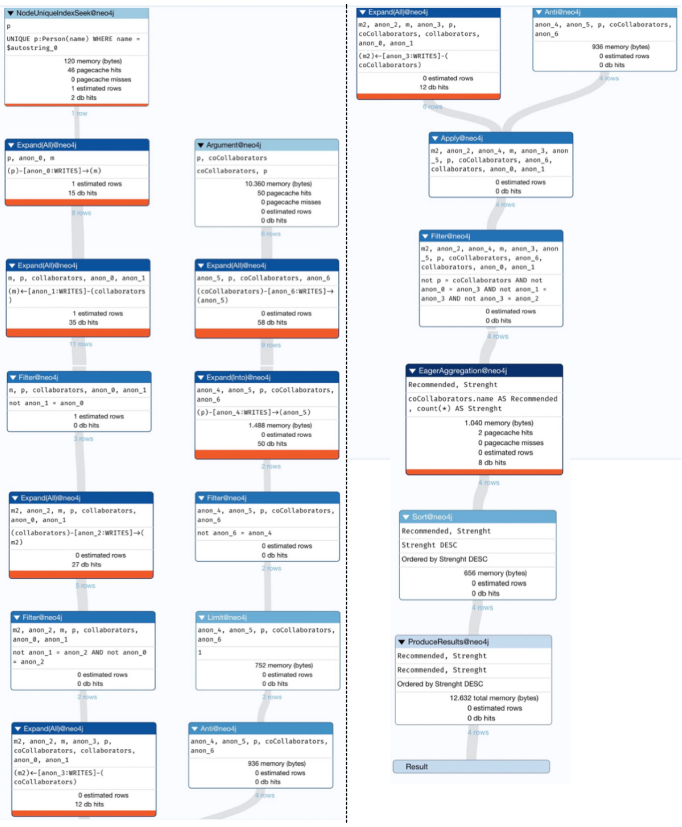
Table | Text | Plan | Code

	Recommended	Strenght
1	"Dimitrios Georgakopoulos"	1
2	"M. Tamer Özsu"	1
3	"Farshad Nayeri"	1
4	"Joe D. Morrison"	1

Started streaming 4 records after 1 ms and completed after 2 ms.

Results: All the recommended are found 1 time, it can be seen also with the following graph, where everyone collaborates with each other only once.





- **QUERY #7**

  Most productive writer of the decade: whit this query we want to extract the top 5 writers, with the name starting with 'c', who did more publications between the years 2000 and 2010.

```
1  PROFILE
2  MATCH (p:Person)-[:WRITES]→(m), (m)-[:WRITTEN_IN]→(y:Year)
3  WHERE p.name STARTS WITH 'C' AND  y.year > 2000 AND y.year < 2010
4  RETURN
5    p.name AS name,
6    count(DISTINCT m) AS counter
7  ORDER BY counter DESC LIMIT 5
```

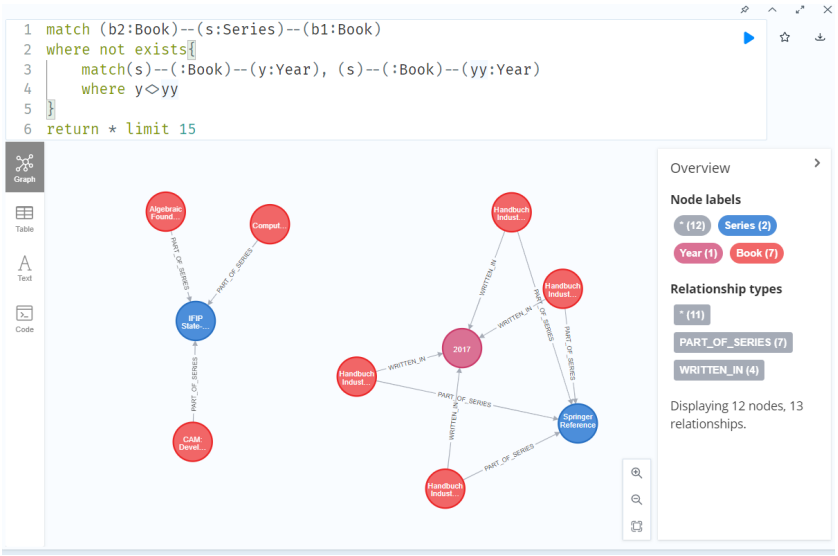
	name	counter
1	"Carlisle Adams"	24
2	"Christoph Meinel"	17
3	"Christophe De Cannière"	13
4	"Constantine Stephanidis"	13
5	"Caroline Fontaine"	11

Started streaming 5 records after 73 ms and completed after 180 ms.

Results: we found the best 5 writers ordered by the number of works that they have done, with the following complete set of operations provided to perform the query.

- **QUERY #8**

  The query searches for series with at least 2 books and selects those whose books are written in the same year. A "not exists" condition is used to verify that there aren't 2 books written in different years in that series. Some controls have been avoided (ex. b1 <> b2) since they are in the same match, and in the second match two books are necessarily different if they have different publication's year.

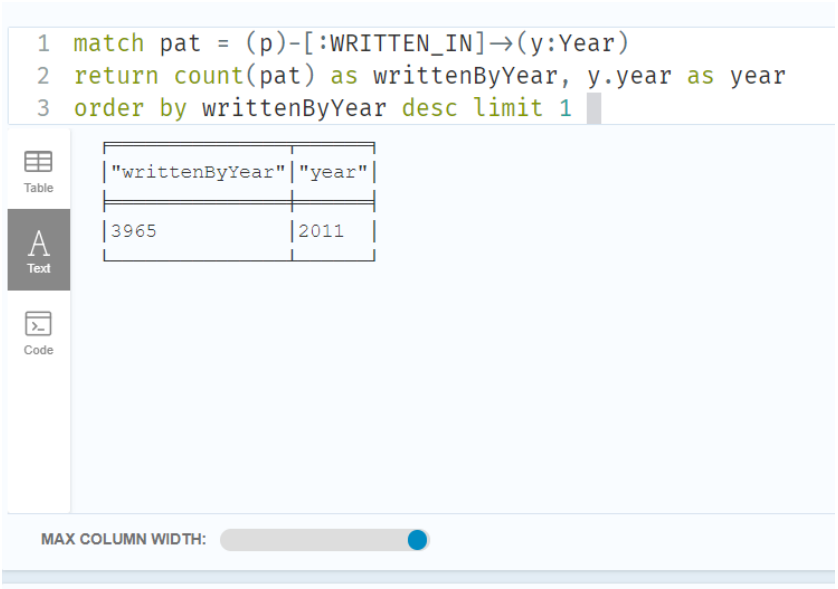


  Complexity: the query needs to find first a series with more than 2 books and then to verify the condition for all the books: this is the most expensive operation.

- **QUERY #9**

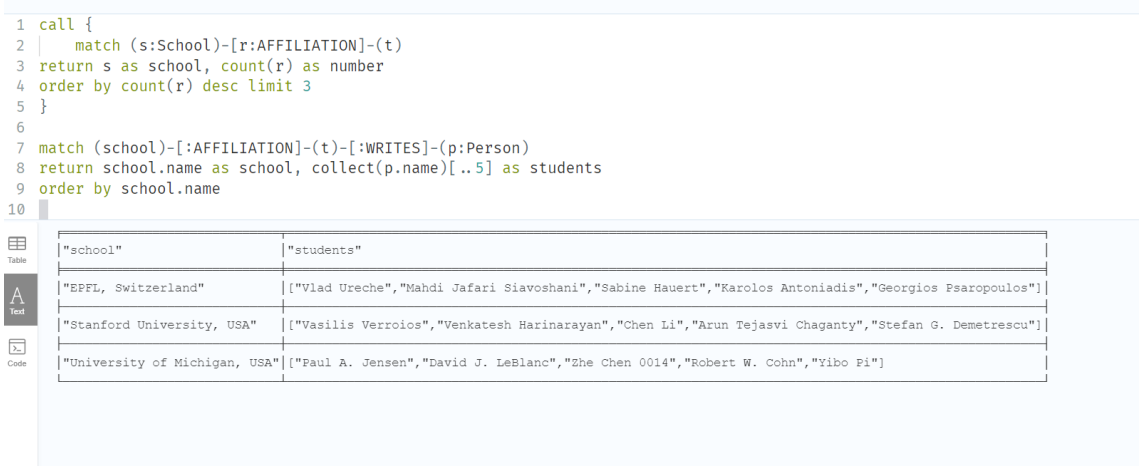
  The query asks for the year with the highest number of publications and the number of publications is shown also in the result

  Complexity: All the nodes "Year" are retrieved by the database since the result is grouped by year property, then the relations that enter in a "Year" node are counted.



- **QUERY #10**

  The query asks for the first 5 students who wrote a publication (phd thesis or master thesis) in a school with the highest number of publications of which is affiliated. The number of schools chosen is limited to 3 and the result for each school shows a list of names. The first call is used to get the schools.



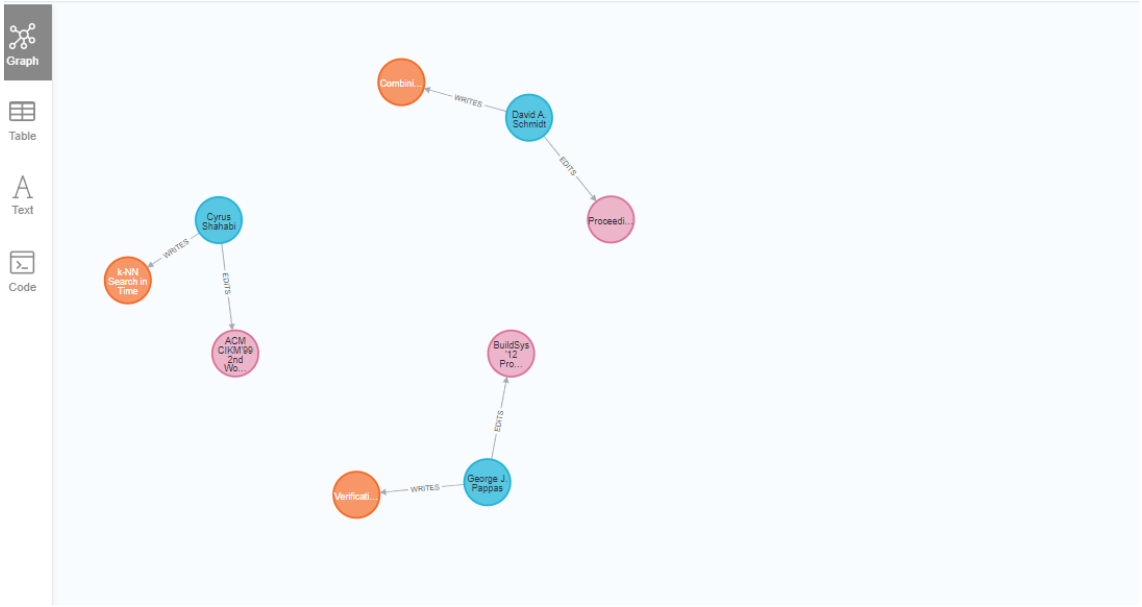
  Complexity: The most expensive part of the query is the first call because for each school in the db the number of affiliations is counted. The second part needs to retrieve all the people linked to these schools although only 5 names are conserved.

- **QUERY #11**

  The query return 3 people who before the 2016 were the only editors of a publication (in this way the edits was necessarily made by that person and not by someone else), then

the person has never make an edit after the 2016, instead after 2016 has wrote at least a book.

```
match (p:Person)-[:EDITS]-(pub), (p)-[:WRITES]-(pub2)-[:WRITTEN_IN]-(y:Year)
with toInteger(left(pub.last_edit,4)) as date, p, pub
where date<2016 and not exists{
    match (p1)-[:EDITS]-(pub)
    where p<>p1
    } and not exists{
        match(p)-[:EDITS]-(pub1)
        where toInteger(left(pub1.last_edit,4))>=2016
    } and y.year>=2016
return * limit 3
```



Complexity: the query searches for a person who has edited and who has written at least 2 different publications. In the where clause some checks using "not exists" are made but the publication and the person have already been retrieved so they are the starting point for the research in the db to check the condition.

- **QUERY #12**

  The aim of the procedure is to specialize some people: who has written an article. A new node labeled "Journalist" is created, then a link between the person and the new node is created: the relation IS_JOURNALIST, the new node is linked to all the articles he has written and finally the old relation: WRITES is deleted.

  Complexity: As shown a new node is created for each person who wrote at least an article.

- **QUERY #13**

  The procedure creates a new relation between a person and a journal if he has written an article that appears in that journal: relation JOURNALIST_IN is created. If "create"

```
1 match (p:Person)-[r:WRITES]-(a:Article)
2 merge (j:Journalist{name: p.name})
3 merge (p)-[:IS_JOURNALIST]→(j)
4 merge (j)-[:WRITES]→(a)
5 delete r
```

Table

Code

Added 17256 labels, created 17256 nodes, set 17256 properties, deleted 27934 relationships, created 45190 relationships, completed after 175324 ms.

Added 17256 labels, created 17256 nodes, set 17256 properties, deleted 27934 relationships, created 45190 relationships, completed after 175324 ms.

was used in place of "merge" there would be multiple repeated links between the same person and the same journal if he wrote multiple articles for that journal. In the relation the attribute last_year is used to specify the most recent year a person wrote an article for that journal

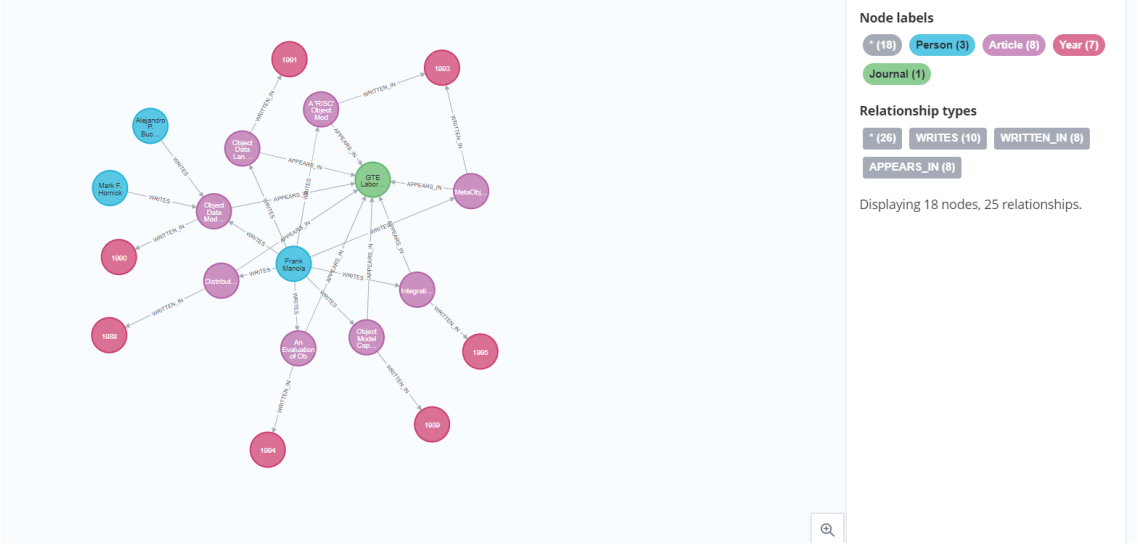


Figure 7.2: Frank Manola before the command

Note: the articles present in a journal aren't all written in the same year, so supposing that an article can be added to a journal also in a later moment, the attribute in the relationship is supposed to keep updated the last time a writer wrote something for that journal.

Complexity: A new link for each person who wrote an article is created every time, in fact, before any creation all the previous relations of this type are deleted in order to keep updated the attribute if a new article is added to a journal, without creating multiple relationships.

```
1 call{
2 match (p:Person)-[:WRITES]→(a:Article)-[:APPEARS_IN]→(j), (a)--(y:Year)
3 return p,j, max(y.year) as mostRecent
4 }
5 match (p)-[r:JOURNALIST_IN]-(j)
6 delete r
7 merge (p)-[:JOURNALIST_IN{last_year: mostRecent}]-(j)
```

Table

Code

Set 17420 properties, deleted 17420 relationships, created 17420 relationships, completed after 579 ms.

Figure 7.3: Command

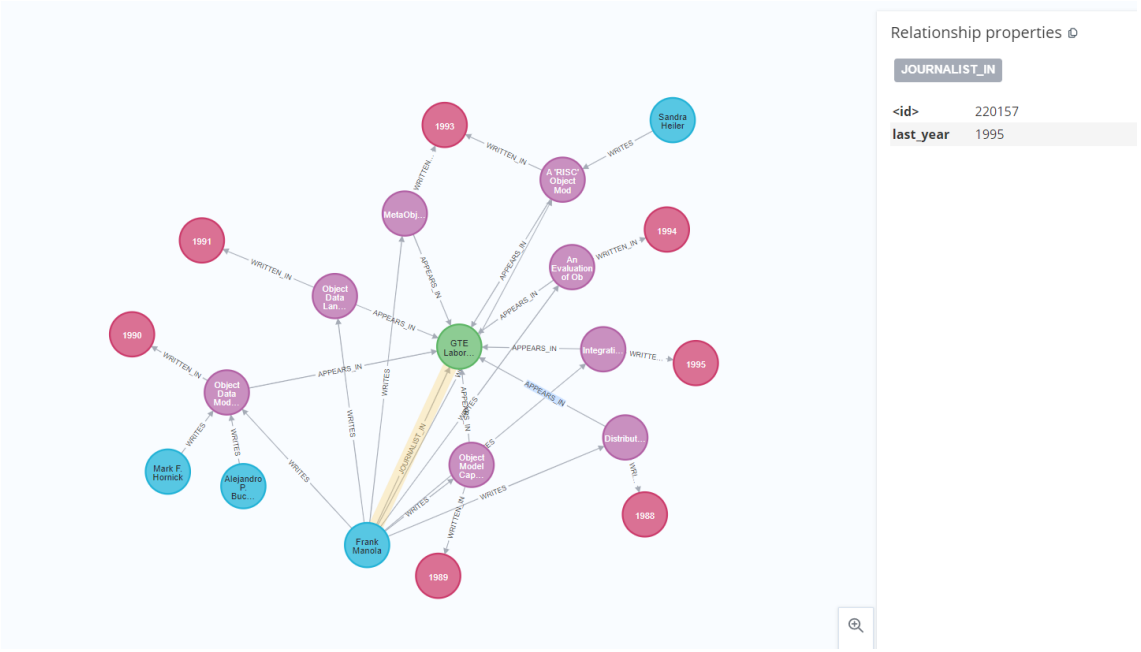


Figure 7.4: Frank Manola and new relationship after the command

- **QUERY #14**

The query asks for the graph representation of the all the co-publications of the writers who worker together for the highest number of times in the whole database: from the first call two pairs of writers are retrieved (a writer has worked very much with the other writer in the pair), then using the second part of the query all their co-publications are retrieved.

```
call{
match path = (p:Person)-[:WRITES]→(b)←[:WRITES]-(p1:Person)
return p.name as writer , p1.name as colleague, count(path) as sum
order by count( path) desc limit 4
}
match path=  (:Person{name:writer})-[:WRITES]→(b)←[:WRITES]-(:Person{name: colleague})
return path
```

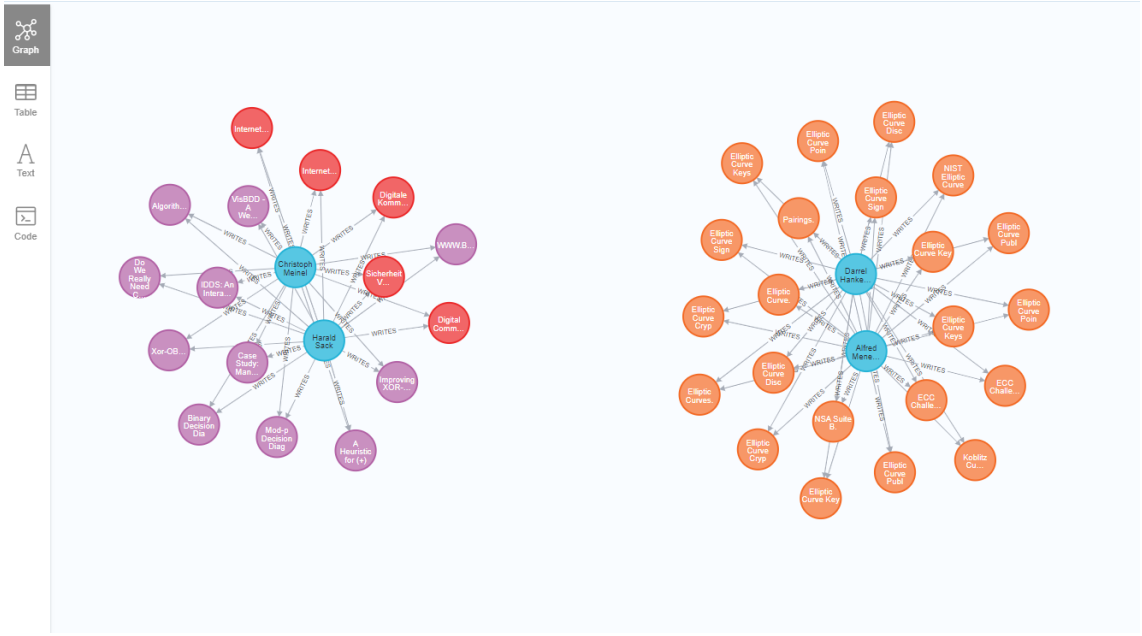


Figure 7.5: Result of first call

Complexity: the most expensive part of the query is the first call since all pairs of writers are compared according to the number of publications written together. The second part is pretty simple because publications are retrieved using a simple pattern, given the 4 authors as starting point for the search.

- **QUERY #15**

Top 3 publishers: with this query we wanted to select the top 3 publishers of proceedings, without showing the first one.

```
1
2 match path = (p:Person)-[:WRITES]→(b)←[:WRITES]-(p1:Person)
3 return p.name as writer , p1.name as colleague, count(path) as sum
4 order by count( path) desc limit 4
5
```

Table Text Code

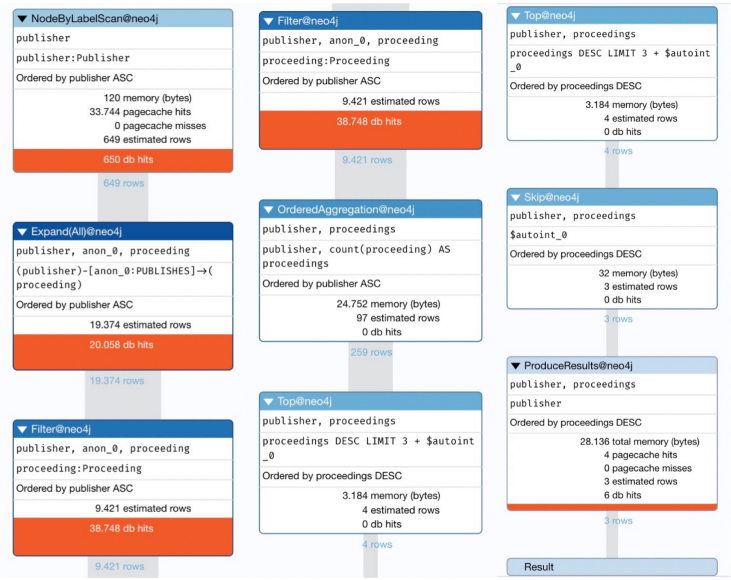
"writer"	"colleague"	"sum"
"Darrel Hankerson"	"Alfred Menezes"	22
"Alfred Menezes"	"Darrel Hankerson"	22
"Christoph Meinel"	"Harald Sack"	16
"Harald Sack"	"Christoph Meinel"	16

Figure 7.6: Final result

```
1 PROFILE
2 MATCH (publisher:Publisher)-[:PUBLISHES]→(proceeding:Proceeding)
3 WITH publisher, count(proceeding) AS proceedings
4 ORDER BY proceedings DESC
5 SKIP 1 LIMIT 3
6 RETURN publisher
```

Graph Table Text Plan Code

"publisher"
{"name":"ACM"}
{"name":"IEEE"}
{"name":"IEEE Computer Society"}



Results: we can see that if we omit "SKIP 1", the result wouldn't be the same.

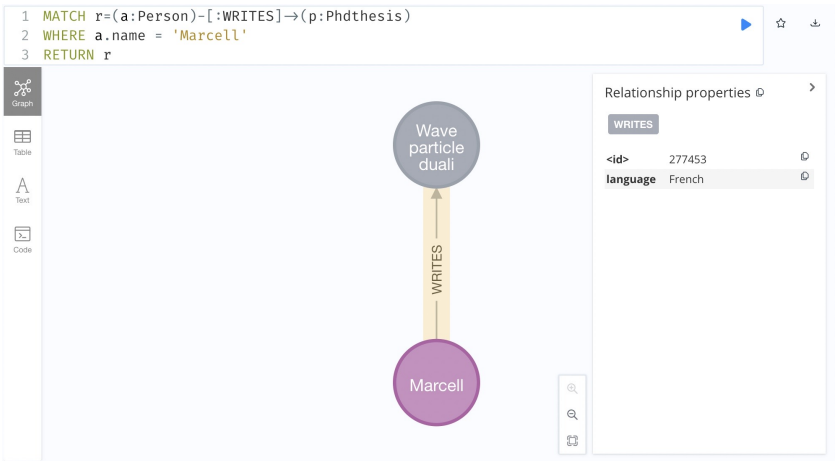
- **QUERY #16**

  Create two new nodes: a Person and a Phdthesis, with their properties. Successfully added.

- **QUERY #17**

  Create the relationship WRITES: we want to connect the two newly created nodes with the WRITES relationship, specifying in which language it was written.

Results: if we search the relationship, we can see that appears also the language used to write the PhdThesis.



Part II

Second delivery

8 Data shaping and document structure

8.1. Document structure

The general structure of a document, which designs an article with multiple attributes, is the following:

```
{
    _id: <...>,
    url: "...",
    title: "...",
    subtitle: "...",
    abstract: "..."
    readingTime: ...,
    metadata: ["...", ...],
    journal: "...",
    volume: ...,
    number: ...,
    date: "...",
    pages: ...,
    authors: [
        {
        name:"...",
        phone: "...",
        userID: <...>,
        email: "...",
        biography: "...",
        affiliations: [
            {
            school: "...",
            city: "...",
            address: "...",
            },
            ...
        ]
        },
        ...
    ],
    readability: {
        level: "...",
        language: "...",
    },
    sections: {
        text: [
            {
            subsection: ...,
            title: "...",
            body: "..."
            },
            ...
        ]
        figures: [
            {
            imageURL: "...",
            caption: "..."
            },
            ...
        ]
    },
    bibliography: [<...>, ...]
}
```

Figure 8.1: Document structure

8.2. Data description

We modeled the fields as follows:

- <**-id**>: integer which represents the document key.
- **url**: article link string
- **title**: string, unique title for each article
- **subtitle**: string with a random subtitle
- **abstract**: string, short random abstract
- **readingTime**: int, number in minutes of reading time
- **metadata**: from 1 to 3 keywords of the article
- **journal**: string, name of the journal to which it belongs
- **volume**: random int from 1 to 10
- **number**: random int from 1 to 20
- **date**: string, date extracted from dataset 'medium-data.csv'
- **pages**: random int from 1 to 10
- **authors**: document which has information about the person, it can be a single author or up to 4 people can collaborate on the same article, the number of authors is randomly selected. Each author has:
  - *name*: string, extracted randomly from a dataset
  - *phone*: string, extracted randomly from an array generated by us
  - *userID*: random int from 1000 to 10000
  - *email*: string composed of 5 random letters + "@gmail.com"
  - *biography*: string, extracted randomly from a dataset
  - *affiliation*: document representing the schools with which the person is affiliated, they can be either one or two and they have a name (chosen from the dataset of schools), the city and the address of where they are allocated.
- **readability**: contains the level and the language of the article, both strings taken from arrays generated by us
- **sections**: composed by text and figures. Text is an object containing from 1 to 5 subsection, title, body representing number, title and body of the paragraphs. Figures is the container of the images, ranging from 1 to 5 per article all with url and caption.
- **bibliography**: from 1 to 5 random numbers, representing the set of references to other articles.

8.3. Example of documents

The result is a file .json of 40 MB, ready for the upload on MongoDB. The following is a screenshot of a document contained within the file:

```
{
  "_id": 2,
  "url": "https://medium.com/swlh/stuck-in-your-past-and-blocked-from-your-future-173e54bdeb04",
  "title": "How to Use ggplot2 in Python",
  "subtitle": ""Sherman, set the WABAC machine to go way, way back…"",
  "abstract": "agistment bankruptcy",
  "readingTime": "5",
  "metadata": [
    "science",
    "politics"
  ],
  "journal": "Better Marketing",
  "volume": 9,
  "number": 5,
  "date": "2019-10-23",
  "pages": 13,
  "authors": [
    {
      "name": "Alfred J. Lewy",
      "phone": "577-496-0339",
      "userID": 6384,
      "email": "auglelg@gmail.com",
      "biography": "alfred j lewy aka sandy lewy graduated from university of chicago in 1973 after studying psychiatry pharmacology and ophthalmology he is a full
      "affiliations": [
        {
          "school": "Park East High School",
          "city": "Brooklyn",
          "address": "223 Graham Avenue"
        }
      ]
    },
    {
      "name": "Hiromu Nonaka",
      "phone": "893-504-0279",
      "userID": 5752,
      "email": "vomyprh@gmail.com",
      "biography": "hiromu nonaka nonaka hiromu born october 20 1925 is a japanese ldp politician and former member of the house of representatives he is a lecturer
      "affiliations": [
        {
          "school": "Global Learning Collaborative",
          "city": "Bronx",
          "address": "250 Hooper Street"
        },
        {
          "school": "Bronx School of Law and Finance",
          "city": "Brooklyn",
          "address": "165-65 84th Avenue"
        }
      ]
    }
  ],
  "readability": {
    "level": "difficult",
    "language": "English"
  },
  "sections": {
    "text": [
      {
        "subsection": 1,
        "title": "Get ready to referee some bot fights.",
        "body": "In a recent appraisal of deep learning (Marcus, 2018) I outlined ten challenges for deep learning, and suggested that deep learning by itself, alth
      },
      {
        "subsection": 2,
        "title": "A Convolutional neural network (CNN) is a neural…",
        "body": "We are in the midst of a gold rush in AI. But who will reap the economic benefits? The mass of startups who are all gold panning? The corporates wh
      },
      {
        "subsection": 3,
        "title": "A breakthrough in Artificial Intelligence or just…",
        "body": "Chatbots are ready to succeed. If you think you have to hack days or even weeks to create a chatbot, you might be wrong. You don't have to be aware
      },
      {
        "subsection": 4,
        "title": "Learn how to break a bad habit by following 4 simple steps. Its as easy as…",
        "body": "A code tutorial in Tensorflow that uses Reinforcement Learning to take free kicks.\nIn my previous article, I presented an AI bot trained to play t
      },
      {
        "subsection": 5,
        "title": "It's time to take charge of your online presence.",
        "body": "This article is a comprehensive overview including a step-by-step guide to implement a deep learning image segmentation model.\nNowadays, semantic
      }
    ],
    "figures": [
      {
        "imageURL": "https://cdn.sofifa.org/players/4/20/188876.png",
        "caption": "Eike Bansen"
      }
    ]
  },
  "bibliography": [
    241,
    652,
    345,
    521
  ]
}
```


Figure 8.2: Example document

9 Data extraction and preprocessing

To make the collection of articles we decided to write a script in python, which takes some information from various dataset found on the site "Kaggle". We decided to search data as much as possible from datasets and then the remaining fields were randomly filled with informations generated by us from scratch. In the end, to build an article in the Json format, we filled every component of documents with data, which we previously stored in arrays. We have tried to use correct information where possible, keeping the structure sketched above unchanged. For example, the fields "name" and "biography" are indeed coherent, thanks to the choice to select data incrementally.


```
1  import csv
2  import json
3  import random
4  import json
5
6  url= []
7  title = []
8  subtitle = []
9  readingTime = []
10 journal = []
11 date = []
12 biography = []
13 abstract = []
14 school = []
15 city = []
16 address = []
17 names = []
18 text = []
19 imageURLs = []
20 captions = []
21
22 with open('/Users/aleksandroaliaj/Documents/aleks/ToJson/medium_data.csv') as csvDataFile:
23     csvReader = csv.reader(csvDataFile)
24     for row in csvReader:
25         url.append(row[1])
26         subtitle.append(row[3])
27         readingTime.append(row[7])
28         journal.append(row[8])
29         date.append(row[9])
30
31 # remove blank space in subtitle array
32 subtitle = [i for i in subtitle if i != '']
```

Figure 9.1: Python script part 1

At the line 32 we simply removed the empty spaces because in the dataset “medium-data.csv” not all articles have the subtitle.


```
35  with open('/Users/aleksandroaliaj/Documents/aleks/ToJson/people_wiki.csv') as csvDataFile1:
36      csvReader = csv.reader(csvDataFile1)
37      for row in csvReader:
38          names.append(row[1])
39          biography.append(row[2])
40
41  with open('/Users/aleksandroaliaj/Documents/aleks/ToJson/abstract.csv') as csvDataFile2:
42      csvReader = csv.reader(csvDataFile2)
43      for row in csvReader:
44          abstract.append(row[1])
45
46  with open('/Users/aleksandroaliaj/Documents/aleks/ToJson/school.csv') as csvDataFile3:
47      csvReader = csv.reader(csvDataFile3)
48      for row in csvReader:
49          school.append(row[1])
50          address.append(row[4])
51          city.append(row[5])
52
53  with open('/Users/aleksandroaliaj/Documents/aleks/ToJson/articles.csv') as csvDataFile4:
54      csvReader = csv.reader(csvDataFile4)
55      for row in csvReader:
56          text.append(row[5])
57
58  with open('/Users/aleksandroaliaj/Documents/aleks/ToJson/imageURLs.csv') as csvDataFile5:
59      csvReader = csv.reader(csvDataFile5)
60      for row in csvReader:
61          imageURLs.append(row[1])
62          captions.append(row[0])
63
64  letters = ["a", "b","c", "d","e", "f","g", "h","i", "j","k", "l","m", "n","o", "p","q", "r","s", "t","u", "v","w", "x","y"
65  topics = ["politics", "news", "history", "science", "literature", "art", "biology", "Personal and Lifestyle","Top", "Local
66  phones = ["984-912-2672","920-525-7265","729-655-1462","516-686-8812","765-366-2090","577-496-0339","931-980-1839","483-81
67  level = ["easy", "medium", "difficult"]
68  language = ["English", "Italian", "Spanish", "French"]
69
```

Figure 9.2: Python script part 2

To create random data, we used a plugin tool from VSCode called “Faker”:

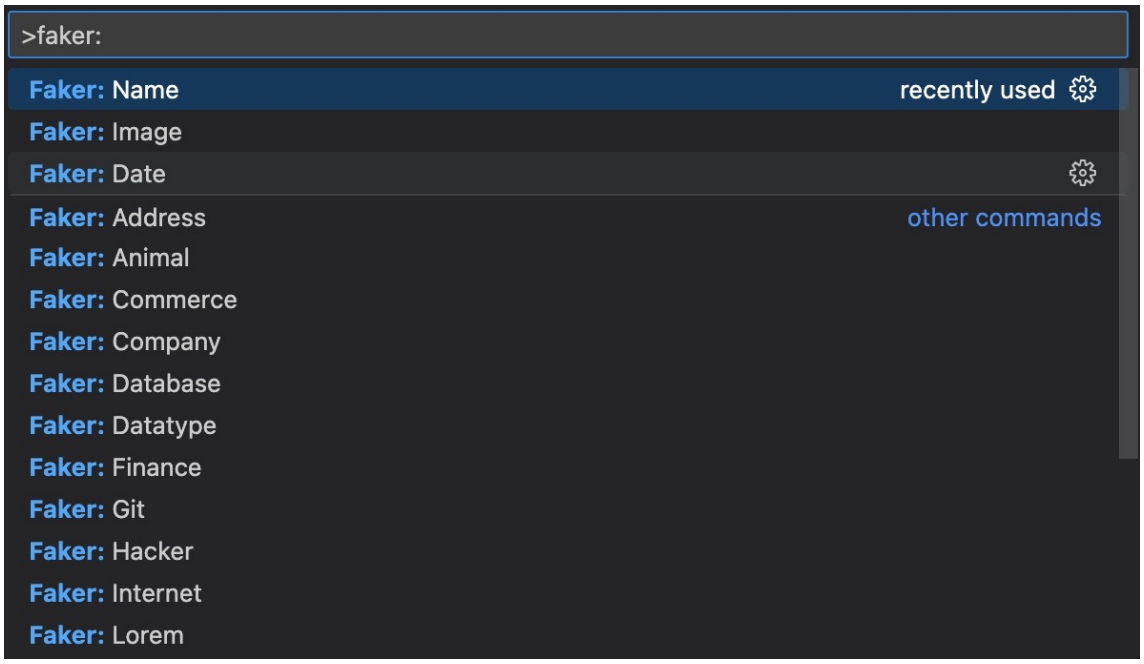
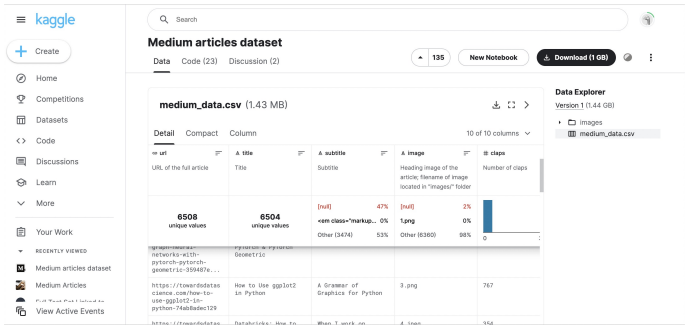
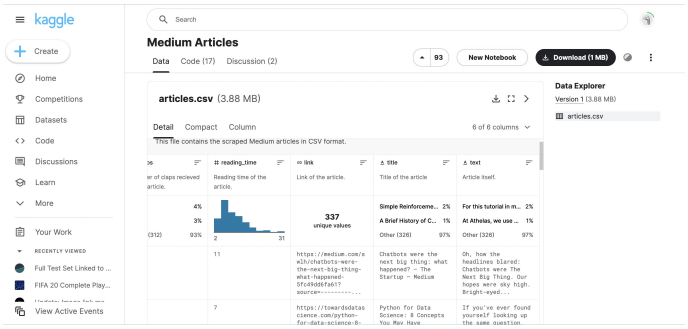
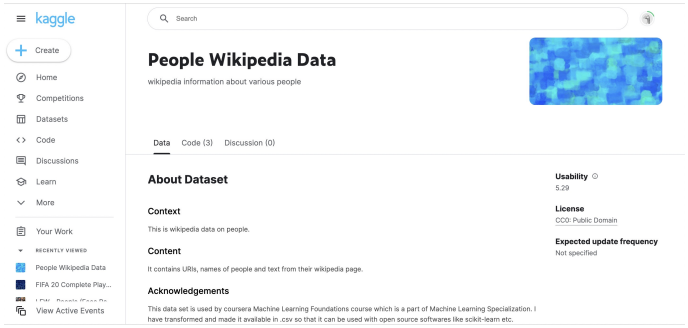


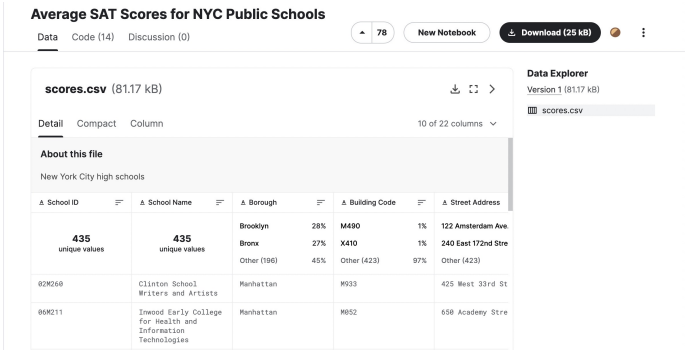
Figure 9.3: Faker plugin

Conversely, we took the real data from the following Kaggle datasets:









Lastly we created 1000 documents written in the "collection.json" file:

```
78  x = [{
79  "_id":i,
80  "url": random.choice(url),
81  "title": title[i],
82  "subtitle": random.choice(subtitle),
83  "abstract": random.choice(abstract),
84  "readingTime": random.choice(readingTime),
85  "metadata": [random.choice(topics) for i in range(random.randint(1,3))],
86  "journal": random.choice(journal),
87  "volume": random.randint(1,10),
88  "number": random.randint(1,20),
89  "date": random.choice(date),
90  "pages": random.randint(1,15),
91  "authors": [
92      {"name": names[i], "phone": random.choice(phones), "userID": random.randint(1000,10000),
93      "email": random.choice(letters)+random.choice(letters)+random.choice(letters)+random.choice(letters)
94              +random.choice(letters)+random.choice(letters)+random.choice(letters)+"@gmail.com",
95      "biography": biography[i],
96      "affiliations": [{"school":random.choice(school),"city": random.choice(city), "address": random.choice(address)}
97                       for m in range(random.randint(1,2))]
98      } for n in range(random.randint(1,4))
99  ],
100 "readability": {"level": random.choice(level), "language": random.choice(language)},
101 "sections": {
102     "text":[
103     {"subsection":k+1,
104     "title": random.choice(subtitle),
105     "body": random.choice(text)} for k in range(random.randint(1,5))
106     ],
107     "figures": [
108         {"imageURL": random.choice(imageURLs),
109         "caption": random.choice(captions)} for i in range(random.randint(1,5))
110     ]
111 },
112 "bibliography": [random.randint(0,999) for i in range(random.randint(1,5))]
113 } for i in range(1000)]
114
115 with open("/Users/aleksandroaliaj/Documents/aleks/ToJson/collection.json", "a") as jsonFile:
116     json.dump(x, jsonFile)
```

Figure 9.4: Python script part 3

At line 115-116 the script writes the json file, adding 1000 articles with the same structure but different data. We have chosen to insert <-id> to avoid the creation of an ObjectId during the “upload phase”, in this way MongoDB Compass will automatically recognize the id as a key, thanks to the fact that it is different for each article.

10 | Data upload

The data upload required us to create a MongoDB Cluster, which we connected to MongoDB Compass, a useful tool for accessing, analyzing, and querying the data.

After the creation of a Database and a Collection, we could finally proceed to upload the data. We decided to import the JSON file obtained during the previous phase because this file format is both a hierarchical data format, like MongoDB documents, and is also explicit about the types of data it encodes. On the contrary CSV data is tabular and each row will be imported into MongoDB as a separate document so it would have been more complicated to obtain the document's structure that we needed.

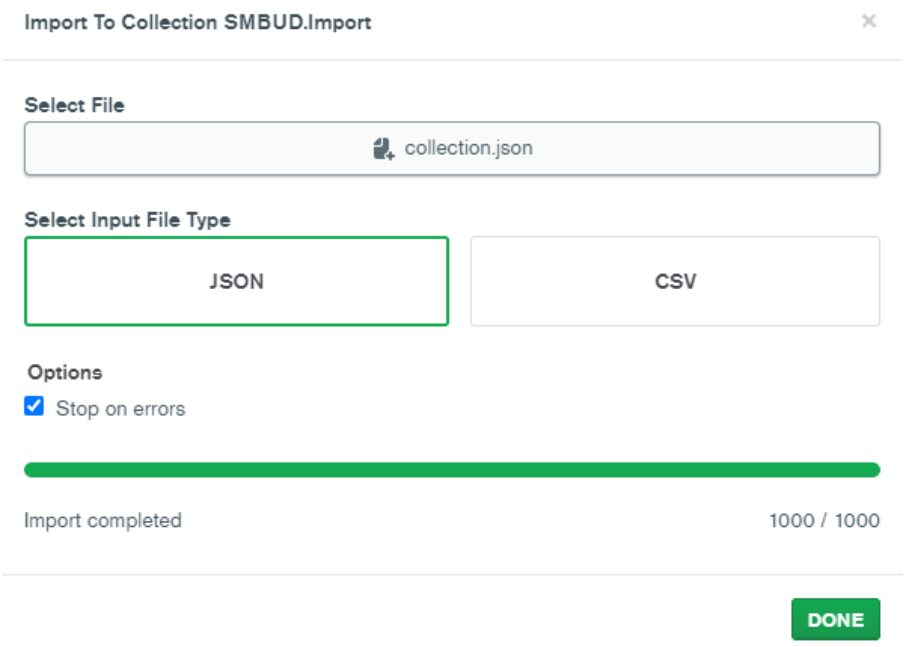


Figure 10.1: Import interface in Compass

The software has automatically recognized the "-id" field as the document's key but, as soon as we imported the data, we realised that the documents were not sorted. So, the first operation we did after the import was the ascending sorting of the documents according to their "-id" value.

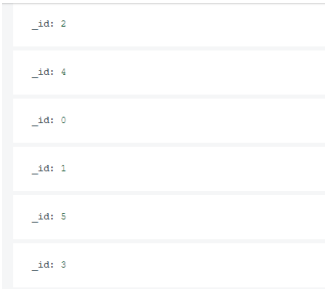


Figure 10.2: Before sorting

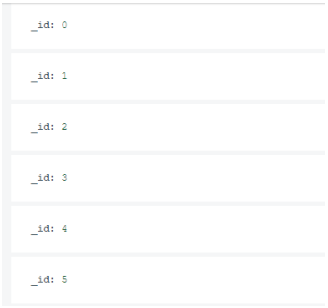


Figure 10.3: After sorting

The algorithm that extracted the data from the csv databases and outputted the JSON file considered all the data either as Int32 values or as String values. Even if these types are valid for all the document's fields, the "date" one is an exception, because a Date type would be more appropriate for it. So, we decided to convert the type of the "date" field of each document from String to Date.

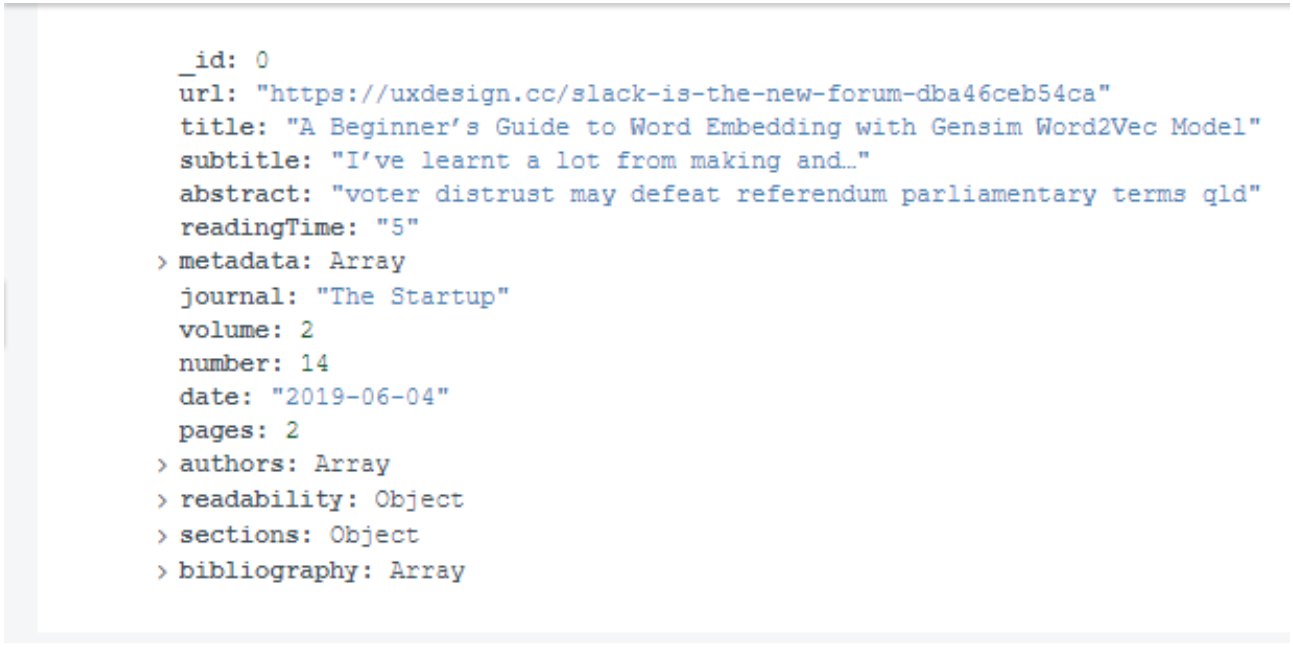


Figure 10.4: Date field with string type

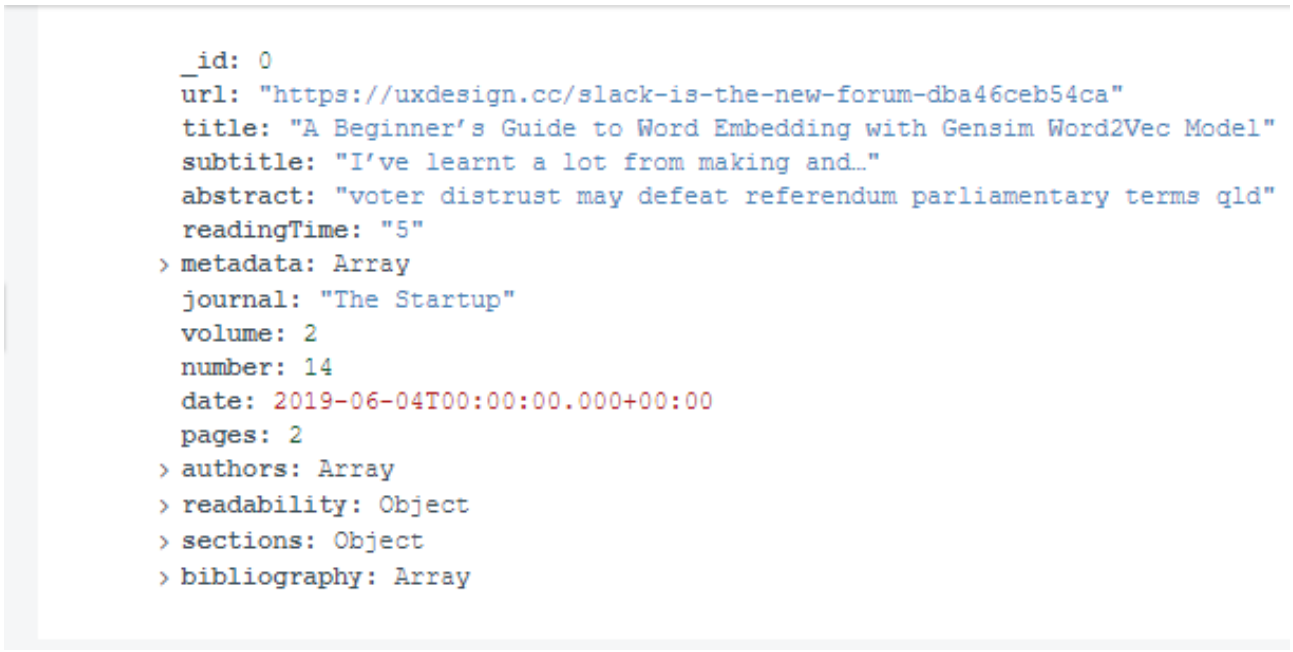


Figure 10.5: Date field with date type

11 | Alternative database generation

The “bibliography” field of each document in an Array containing a set of numbers which are the IDs of the documents it is linked to. To explicit and make more visible this relationship we decided to generate a new collection containg in the basic document form an additional field (“linked_bibliography”) by joining the documents linked to each other.

To do so, we used the $lookup" operator considering "bibliography" as localField and "_id" as foreignField.

The join performed by this operator links a document to all the others whose ID corresponds to the one stored in the “bibliography” field of the local document.

Figure 11.1: Lookup operation

```
_id: 0
url: "https://uxdesign.cc/slack-is-the-new-forum-dba46ceb54ca"
title: "A Beginner's Guide to Word Embedding with Gensim Word2Vec Model"
subtitle: "I've learnt a lot from making and…"
abstract: "voter distrust may defeat referendum parliamentary terms qld"
readingTime: "5"
› metadata: Array
journal: "The Startup"
volume: 2
number: 14
date: 2019-06-04T00:00:00.000+00:00
pages: 2
› authors: Array
› readability: Object
› sections: Object
› bibliography: Array
    0: 331
› linked_bibliography: Array
  › 0: Object
      _id: 331
      url: "https://medium.com/datadriveninvestor/how-to-deploy-tensorflow-to-andr…"
      title: "3 Powerful Trust Signals to Unlock Your Fintech Startups Growth"
      subtitle: "A step by step guide to creating better designs…"
      abstract: "girl ostracised for reporting sex abuse"
      readingTime: "2"
    › metadata: Array
      journal: "Data Driven Investor"
      volume: 10
      number: 17
      date: 2019-09-18T00:00:00.000+00:00
      pages: 9
    › authors: Array
    › readability: Object
    › sections: Object
    › bibliography: Array
```


Figure 11.2: Lookup operation result

This though is not very smart.In fact the resulting database is full of replicas of articles contained in other articles: in practice this solution should never be used for big and complex data.

Here (the bibliography field) the use of reference over containment is fully justified.

For the successive work we will be using the first version of the database without the pre-made join: we will eventually manually join the DB in the aggregation pipeline to obtain interesting results.

12 | Queries

1. **Query #1**

   This query filters all documents after the 03/06/2019 and that has a volume number in [2,3,4,6] or was published before page 10.

   Finally it projects only the title of the article,the volume, the date and the page at which it was published.

```
> db.articles.find({
    "date" : {"$gt" : ISODate("2019-03-06")},
    "$or" : [{"volume" : {"$in" : [2,3,4,6]}},{"pages" : {"$lt" : 5}}]
  } ,
  {"title" : 1,
  "volume" : 1,
  "pages" : 1,
  "date" : 1
  })
< { _id: 0,
    title: 'A Beginner's Guide to Word Embedding with Gensim Word2Vec Model',
    volume: 2,
    date: 2019-06-04T00:00:00.000Z,
    pages: 2 }
  { _id: 3,
    title: 'Databricks: How to Save Files in CSV on Your Local Computer',
    volume: 2,
    date: 2019-12-05T00:00:00.000Z,
    pages: 8 }
  { _id: 4,
```

Figure 12.1: Query and result

It's a rather simple query to perform in fact it's very fast.

```
executionStats:
 { executionSuccess: true,
   nReturned: 543,
   executionTimeMillis: 18,
   totalKeysExamined: 0,
   totalDocsExamined: 1000,
```

Figure 12.2: Statistics

2. **Query #2**

   For this query we first look for all the documents of medium difficulty in Spanish and then we filter for the ones that have at least one of the authors that has both an ID greater or equal to 5000 and at least one school affiliation.

   We project in the result only the title of the article,all the authors names and id's and all their affiliated schools.

```
> db.articles.find(
  {
    "$and" : [{"readability.language" : "Spanish"},{"readability.level" : "medium"}] ,
    "authors" : {
      "$elemMatch" : {
        "userID" : {"$gte" : 5000},
        "affiliations.school" : {"$exists" : true}
      }
    }
  },
  {
    "title" : 1,
    "authors.name" : 1,
    "authors.userID" : 1,
    "authors.affiliations.school":1
  })
```

Figure 12.3: Query

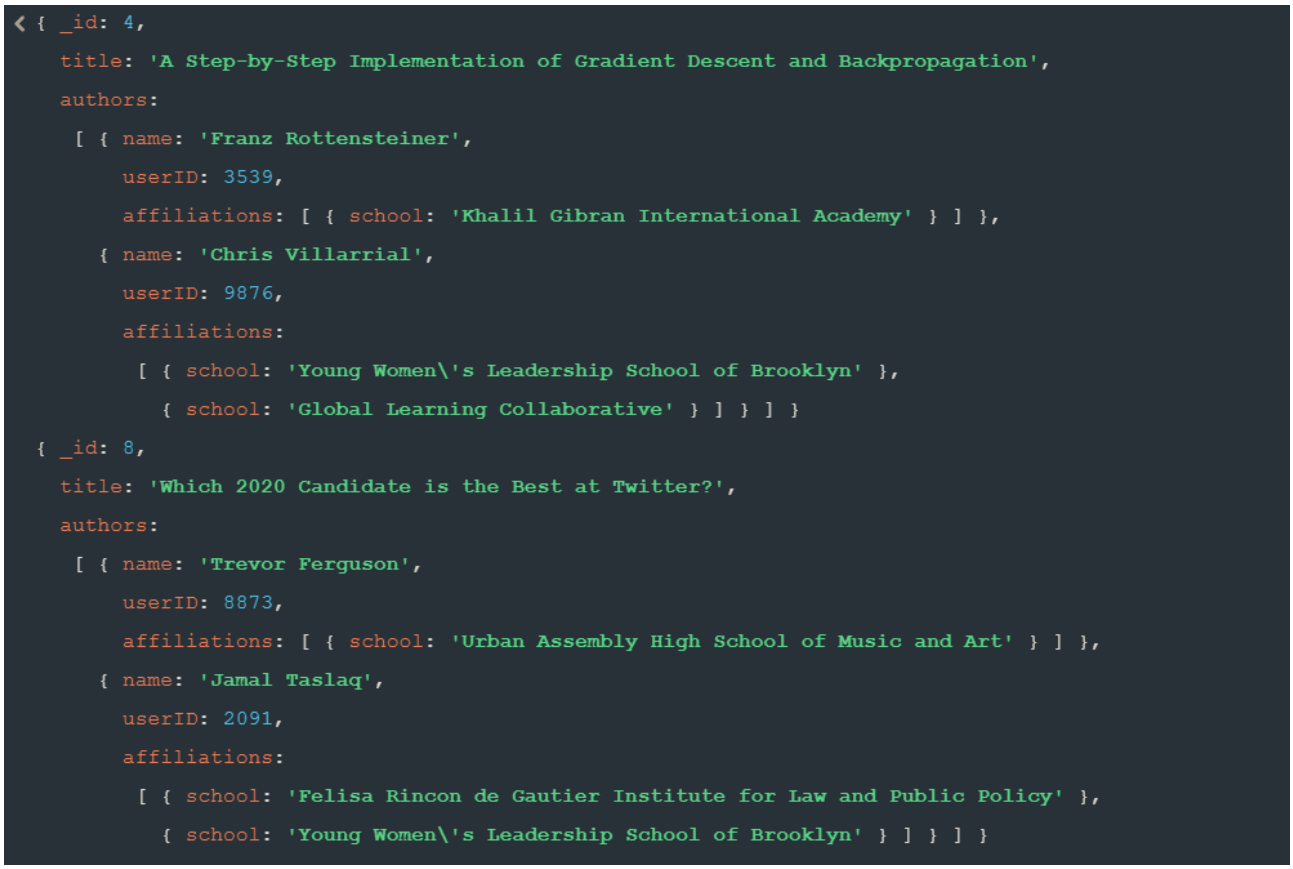


Figure 12.4: Result

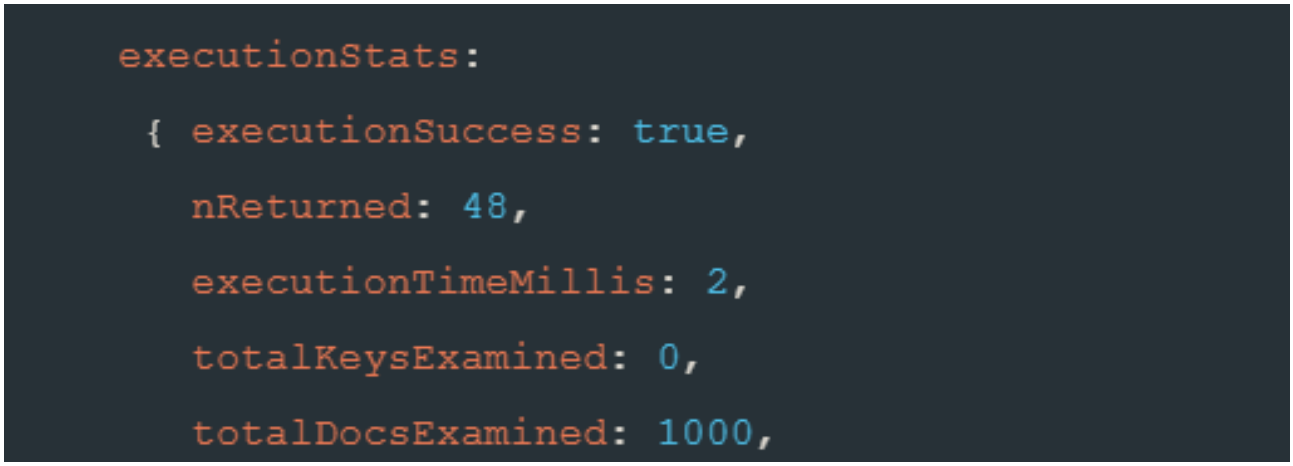


Figure 12.5: Statistics

3. **Query #3**

   Here we used the aggregation pipeline to :

   (a) Filter all the articles that had Beginner in the title thanks to the regex command;

   (b) Unwind each article with respect to the metadata array;

   (c) Group all of the articles based on the metadata tag they had and calculating the average position with the pages field;

   (d) Filtering out all the groups with an average position over eight.

```
> db.articles.aggregate([
    {
      "$match" : {"title" : {"$regex" : /Beginner/}}
    },
    {
      "$unwind" : {"path" : "$metadata"}
    },
    {
      "$group" : {
          "_id" : "$metadata" ,
          "average position of the article" : {"$avg" : "$pages"}
                }
    },
    {
      "$match" : {
          "average position of the article" : {"$lt" : 8}
                }
    }
  ])
< { _id: 'Educational Experiences',
    'average position of the article': 6 }
  { _id: 'politics', 'average position of the article': 6 }
  { _id: 'Current Events', 'average position of the article': 6 }
```

Figure 12.6: Query and result

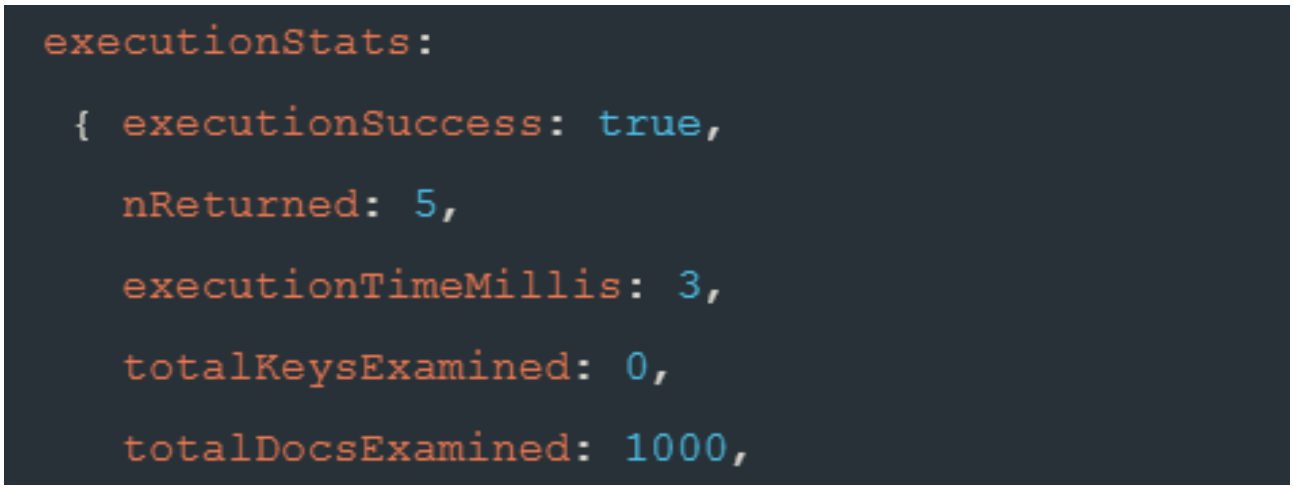


Figure 12.7: Statistics

4. **Query #4**

   Here we used the aggregation pipeline to :

   (a) Filter all the articles that had at least one author named Trevor (or some name containing Trevor);

   (b) Use the command lookup to join those articles with the ones provided by "_id" in the bibliography array: we create a document that contains a field "quoted" that is an array of complete article documents that had the same id that was in the bibliography array;

   (c) Match all the articles that have at least one article in the quoted array that has both "Data" in the title and a readability level of medium or hard;

   (d) Return only all the authors names and date but not the the title of the original article,all the journal names and readability level in the quoted array .

```
> db.articles.aggregate([
    {
      "$match" : {
        "authors.name" : {"$regex" : /Trevor/}
      }
    },
    {
      "$lookup" : {
        from : "articles",
        localField : "bibliography",
        foreignField : "_id",
        as : "quoted"
      }
    },
    {
      "$match" : {
        "quoted" : {
          "$elemMatch" : {
            "journal" : {"$regex" : /Data/},
            "readability.level" : {"$in" : ["medium","hard"]}
          }
        }
      }
    },
    {
      "$project" : {
        "authors.name" : 1,
        "date" : 1,
        "quoted.journal" : 1,
        "quoted.readability.level" : 1
      }
    },
    {
      "$sort" : { "date" : -1}
    }
    ])
```

Figure 12.8: Query

```
{ _id: 617,
  date: 2019-09-18T00:00:00.000Z,
  authors:
   [ { name: 'Trevor Grove' },
     { name: 'Donald Maxwell (baritone)' } ],
  quoted:
   [ { journal: 'The Startup', readability: { level: 'difficult' } },
     { journal: 'Towards Data Science',
       readability: { level: 'difficult' } },
     { journal: 'Towards Data Science',
       readability: { level: 'medium' } },
     { journal: 'UX Collective', readability: { level: 'easy' } } ] }
{ _id: 8,
  date: 2019-05-20T00:00:00.000Z,
  authors: [ { name: 'Trevor Ferguson' }, { name: 'Jamal Taslaq' } ],
  quoted:
   [ { journal: 'The Startup', readability: { level: 'easy' } },
     { journal: 'Towards Data Science',
       readability: { level: 'easy' } },
     { journal: 'Data Driven Investor',
       readability: { level: 'medium' } } ] }
```

Figure 12.9: Result

```
executionStats:
 { executionSuccess: true,
   nReturned: 4,
   executionTimeMillis: 16,
   totalKeysExamined: 13,
   totalDocsExamined: 1013,
```

Figure 12.10: Statistics

5. **Query #5**

   In this query we filter all the articles published in a ".com" site that have "Guide" in the title and are easy to read.

   We are interested in grouping them by journal in which they are published and based on that grouping we retrieve the minimum position where they appear.

```
> db.articles.aggregate([
    {
      "$match" : {
        "url" : {"$regex" : /.com/},
        "title" : {"$regex" : /Guide/},
        "readability.level" : "easy"
      }
    },
    {
      "$group" : {
        "_id" : "$journal",
        "earliest article position" : {"$min" : "$pages"}
      }
    }
    ])
< { _id: 'Data Driven Investor', 'earliest article position': 9 }
  { _id: 'Towards Data Science', 'earliest article position': 13 }
  { _id: 'The Writing Cooperative',
    'earliest article position': 6 }
  { _id: 'The Startup', 'earliest article position': 8 }
```

Figure 12.11: Query and result

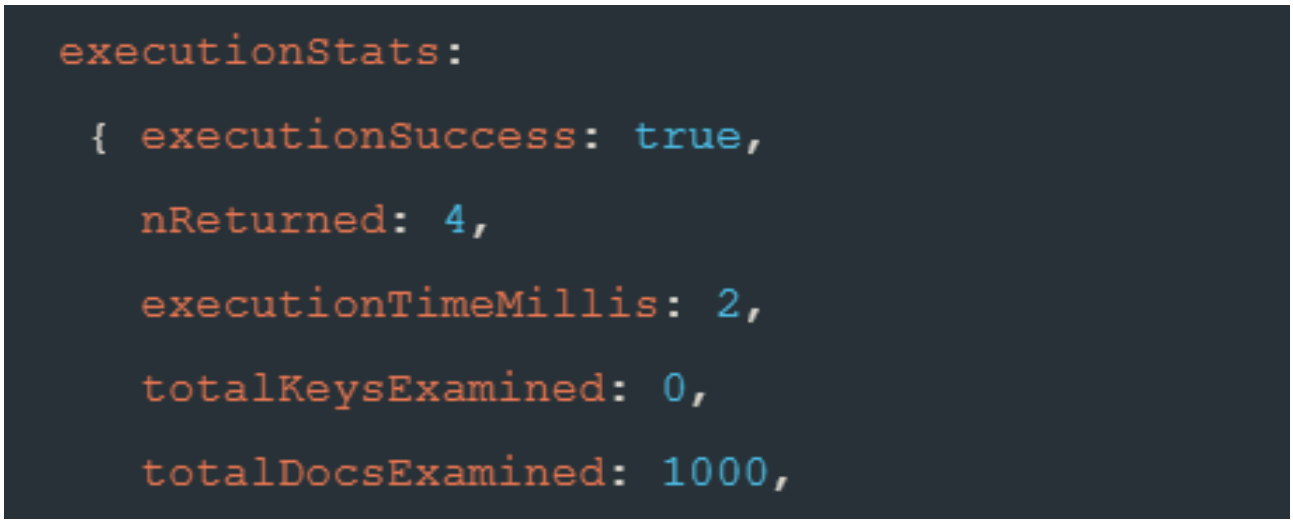


Figure 12.12: Statistics

6. **Query #6**

   The query asks for all the articles whose topics are both literature and politics, sort them by the number of linked publications in ascending order and limit the result to 5: the number is obtained by the bibliography array's size. In the final result only the title and a new filed called "num_of_bibl" are showed, beyond the _id.

```
Atlas atlas-5ns0yb-shard-0 [primary] articles> db.articles.aggregate([
...   {"$match" : {"metadata":{ "$all":[ "literature","politics",]}}}
...   ,{"$project":{"title":1, "num_of_bibl":{ "$size": "$bibliography"}}}
...   , {"$sort" : {"num_of_bibl":-1}},
...   {"$limit":5}
...   ])
[
  {
    _id: 90,
    title: 'Turn Your Productivity on Autopilot',
    num_of_bibl: 5
  },
  {
    _id: 293,
    title: 'Running a Remote Work Stand-up – Rule #2: Upgrade YTB to DCI',
    num_of_bibl: 4
  },
  { _id: 558, title: 'Superhuman AI Is Not a Myth', num_of_bibl: 4 },
  {
    _id: 451,
    title: 'Selenium and SQL Combined- Top Premier League Players',
    num_of_bibl: 4
  },
  { _id: 63, title: 'A Framework for Creativity', num_of_bibl: 4 }
]
```

Figure 12.13: Query and result

Complexity: the first match limits the result in the pipeline then the project and sort operations reorder the result using the new field: "num_of_bibl", finally the limit stop the process when 5 results are obtained.

```
executionStats: {
  executionSuccess: true,
  nReturned: 13,
  executionTimeMillis: 1,
  totalKeysExamined: 0,
  totalDocsExamined: 1000,
```

Figure 12.14: Statistics

7. **Query #7**

The query is used to show the number of articles for each journal. The aggregate operator along with the group clause is used to group all the documents by journal's name, a new filed called "num_of_articles" is used to show the count (using the count operator).

```
Atlas atlas-5ns0yb-shard-0 [primary] articles> db.articles.aggregate([
... {"$group": {
...     "_id": "$journal",
...     "num_of_articles": {"$count":{}}}}
... ])
[
  { _id: 'Better Marketing', num_of_articles: 43 },
  { _id: 'UX Collective', num_of_articles: 86 },
  { _id: 'Better Humans', num_of_articles: 3 },
  { _id: 'Towards Data Science', num_of_articles: 215 },
  { _id: 'Data Driven Investor', num_of_articles: 137 },
  { _id: 'The Startup', num_of_articles: 458 },
  { _id: 'The Writing Cooperative', num_of_articles: 58 }
]
```

Figure 12.15: Query and result

Complexity: All the documents in the collection are retrieved since the "journal" field is used to make the classification

```
executionStats: {
  executionSuccess: true,
  nReturned: 1000,
  executionTimeMillis: 2,
  totalKeysExamined: 0,
  totalDocsExamined: 1000,
```

Figure 12.16: Statistics

8. **Query #8**

The result of the query shows only the title of the articles along with cities where schools, to which authors are affiliated, are placed. In particular, is needed that at least one author is affiliated to necessarily 2 school: one in Manhattan and the other in Brooklyn. If the condition is verified, then also second author's city is shown.

```
Atlas atlas-5ns0yb-shard-0 [primary] articles> db.articles.find(
...     {"authors":
...             {"$elemMatch":
...                     { $and: [{ "affiliations.city" : "Manhattan"},{ "affiliations.city" : "Brooklyn"}]}}},
...     {"title" : 1, "authors.affiliations.city" : 1}
... )
[
  {
    _id: 1,
    title: 'Hands-on Graph Neural Networks with PyTorch & PyTorch Geometric',
    authors: [
      { affiliations: [ { city: 'Flushing' } ] },
      { affiliations: [ { city: 'Brooklyn' }, { city: 'Manhattan' } ] }
    ]
  },
  {
    _id: 8,
    title: 'Which 2020 Candidate is the Best at Twitter?',
    authors: [
      { affiliations: [ { city: 'Long Island City' } ] },
      { affiliations: [ { city: 'Brooklyn' }, { city: 'Manhattan' } ] }
    ]
  },
  {
    _id: 20,
    title: 'Buyers beware, fake product reviews are plaguing the internet. How Machine Learning can help to spot them.',
    authors: [
      { affiliations: [ { city: 'Brooklyn' } ] },
      { affiliations: [ { city: 'Manhattan' }, { city: 'Brooklyn' } ] }
    ]
  },
  {
    _id: 23,
    title: '<strong class="markup--strong markup--h3-strong">White on black or black on white? The pros and cons of Dark Mode</strong>',
    authors: [
      { affiliations: [ { city: 'Brooklyn' } ] },
      { affiliations: [ { city: 'Manhattan' }, { city: 'Brooklyn' } ] }
    ]
  },
  {
    _id: 27,
    title: 'Some proposed color heuristics',
    authors: [
      { affiliations: [ { city: 'Flushing' } ] },
      { affiliations: [ { city: 'Manhattan' }, { city: 'Brooklyn' } ] }
    ]
  },
  {
```

Figure 12.17: Query and result

Complexity: the “elemMatch” is used to verify that there is at least one author that satisfies all the conditions, therefore “authors” array is retrieved for each document in the collection. The query accesses an array and his sub objects, then it uses a field of the sub objects.

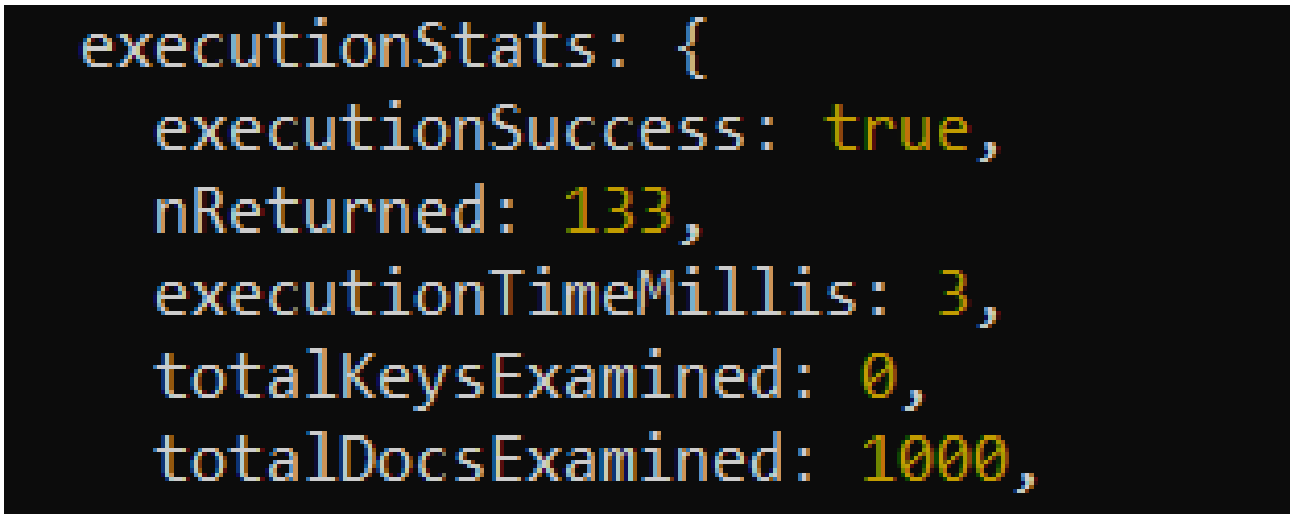


Figure 12.18: Statistics

9. **Query #9**

   The query is used to get the average number of pages for each topic, then the result if filtered according to some conditions later explained. At first the "unwind" operator creates a new object for each topic present in the "metadata" array, then the result is grouped by topic's name while the average number of pages is calculated, a filter condition is used to select only the results with an average greater or equal to 7.7 or of history (its average is 7.6): the or operator is used, finally the result is sorted by the average in decrescent order.

```
Atlas atlas-5ns0yb-shard-0 [primary] articles> db.articles.aggregate([
...     { $unwind: "$metadata" },
...     { $group: {
...               "_id": "$metadata",
...               "avg_pages": { $avg: "$pages" }
...     }},
...     {$match : {
...             $or: [
...                     {"avg_pages": {$gte: 7.7}},
...                     {"_id": "history"}
...             ]
...     }},
...     { $sort: { "avg_pages": -1 } }
... ])
[
  { _id: 'biology', avg_pages: 8.459016393442623 },
  { _id: 'science', avg_pages: 8.285714285714286 },
  { _id: 'literature', avg_pages: 8.185185185185185 },
  { _id: 'Content Marketing', avg_pages: 8.172932330827068 },
  { _id: 'art', avg_pages: 8.08 },
  { _id: 'news', avg_pages: 8.065693430656934 },
  { _id: 'politics', avg_pages: 8.036764705882353 },
  { _id: 'Mental Health and Well-Being', avg_pages: 8 },
  { _id: 'Have You Ever … ?', avg_pages: 7.905511811023622 },
  { _id: 'Top', avg_pages: 7.873949579831932 },
  { _id: 'Personal and Lifestyle', avg_pages: 7.82089552238806 },
  { _id: 'Educational Experiences', avg_pages: 7.706896551724138 },
  { _id: 'history', avg_pages: 7.6231884057971016 }
]
```

Figure 12.19: Query and result

```
executionStats: {
  executionSuccess: true,
  nReturned: 1000,
  executionTimeMillis: 3,
  totalKeysExamined: 0,
  totalDocsExamined: 1000,
```

Figure 12.20: Statistics

Complexity: for each document present in the collection at least one new document is created since the "metadata" array is unrolled and its size is always greater or equal to 1: the number of objects become greater then 1000, the "group" operator shrinks the result to the number of topics present in the db.

10. **Query #10**

    The query asks for the articles classified as "easy" but linked only to articles "not easy" i.e. "medium" or "difficult", at least one for each category must be present. The "lookup" operator is used to join each document to its linked bibliographies, using a "project" operator only the "readability" filed is maintained, along with the readability of its linked bibliographies (the "readability" field is an object with a level of difficulty field). Then the result is filtered using 2 matches: one for the "easy" condition, and the other for the more complex condition on the subfields (here an "and" operator is used).

```
Atlas atlas-5ns0yb-shard-0 [primary] articles> db.articles.aggregate([
...     {$lookup : {
...             from :"articles",
...             localField:"bibliography",
...             foreignField: "_id",
...             as: "sources"}},
...     {$project: {
...             "readability":1,
...             "sources.readability":1
...     }},
...     {$match : {
...             "readability.level": "easy"
...     }},
...     {$match: {
...             $and: [
...                     {"sources.readability.level" : "difficult"},
...                     { "sources.readability.level" : "medium"},
...                     {"sources.readability.level": {$ne: "easy"}}
...             ]
...     }}
... ])
[
  {
    _id: 10,
    readability: { level: 'easy', language: 'English' },
    sources: [
      { readability: { level: 'medium', language: 'English' } },
      { readability: { level: 'difficult', language: 'Spanish' } }
    ]
  },
  {
    _id: 26,
    readability: { level: 'easy', language: 'Spanish' },
    sources: [
      { readability: { level: 'medium', language: 'English' } },
      { readability: { level: 'difficult', language: 'French' } },
      { readability: { level: 'medium', language: 'French' } },
      { readability: { level: 'medium', language: 'English' } }
    ]
  },
  {
    _id: 32,
    readability: { level: 'easy', language: 'English' },
    sources: [
      { readability: { level: 'medium', language: 'Italian' } },
      { readability: { level: 'difficult', language: 'French' } },
      { readability: { level: 'medium', language: 'French' } },
      { readability: { level: 'medium', language: 'English' } },
      { readability: { level: 'medium', language: 'Italian' } }
    ]
  },
  {
    _id: 57,
    readability: { level: 'easy', language: 'Italian' },
    sources: [
      { readability: { level: 'medium', language: 'English' } },
      { readability: { level: 'difficult', language: 'English' } },
      { readability: { level: 'difficult', language: 'French' } }
    ]
  },
  {
```

Complexity: For each element the lookup is executed, then the project and matches operators filter the result reducing the complexity.

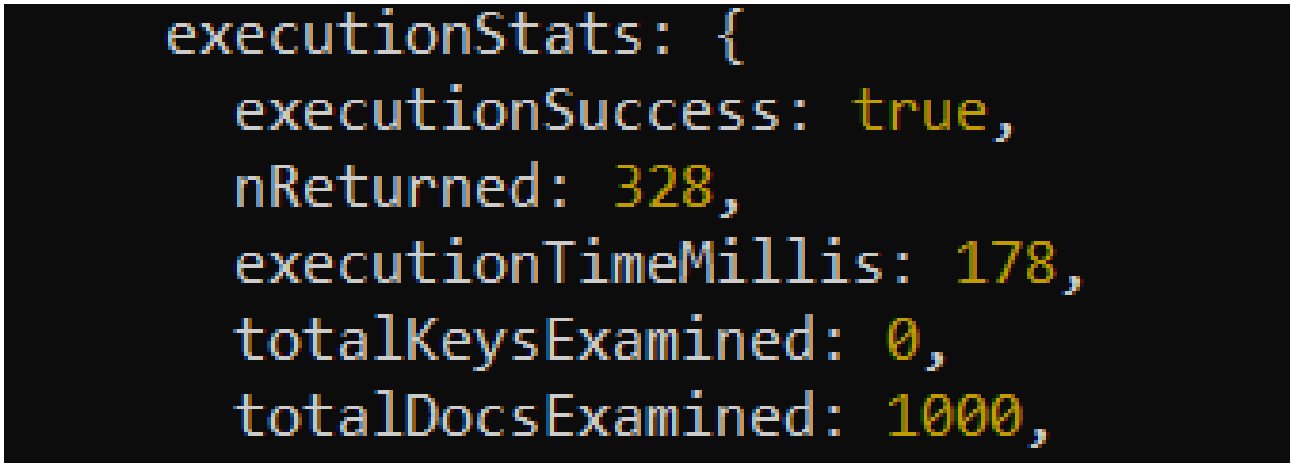


Figure 12.22: Statistics

11. **Query #11**

    This query retrieves all the articles that satisfies all the following conditions:

    (a) Include the word "AI" in the title;

    (b) Were published by the journal "The Startup – New and updated edition";

    (c) Are longer than 6 pages.

    It then shows only the "title", "journal" and "pages" fields.

    Given that the command $regex is used to find the word " AI " in the title (with spaces before and after), it won't select the articles with titles which either have AI as the first word, last word or with some punctuation character.

```
SMBUD> db.Articles.find({$and: [{title: {$regex: / AI /}}, {journal: "The Startup - New and updated edition"}, {pages: {$gt: 6}}]}, {title: 1, journal: 1, pages: 1})
[
  {
    _id: 9,
    title: 'What if AI model understanding were easy?',
    journal: 'The Startup - New and updated edition',
    pages: 11
  },
  {
    _id: 373,
    title: 'An Executive's Guide to Implementing AI and Machine Learning',
    journal: 'The Startup - New and updated edition',
    pages: 14
  },
  {
    _id: 614,
    title: 'How AI will drive profitability in Micro-mobility',
    journal: 'The Startup - New and updated edition',
    pages: 11
  },
  {
    _id: 640,
    title: '<strong class="markup--strong markup--h3-strong">The Impact of AI in Voice Recognition in Africa</strong>',
    journal: 'The Startup - New and updated edition',
    pages: 14
  },
  {
    _id: 696,
    title: 'Data Science & AI Journey: Part 1',
    journal: 'The Startup - New and updated edition',
    pages: 10
  }
]
```

Figure 12.23: Query and result

```
executionStats: {
    executionSuccess: true,
    nReturned: 5,
    executionTimeMillis: 1,
    totalKeysExamined: 0,
    totalDocsExamined: 1003,
```

Figure 12.24: Statistics

12. **Query #12**

    This query shows how many articles are present in the dataset for every category in the "metadata" field.

    To do so, it uses the $unwind command to make one copy for every element of the array in "metadata", then uses the $group command to select (without repetition) the categories present and to count the occurrences for each of them.

```
SMBUD> db.Articles.aggregate({$unwind: {path: "$metadata"}}, {$group: {_id: "$metadata", num_of_categories: {$count: {}}}})
[
  { _id: 'literature', num_of_categories: 135 },
  { _id: 'news', num_of_categories: 137 },
  { _id: 'Content Marketing', num_of_categories: 133 },
  { _id: 'Mental Health and Well-Being', num_of_categories: 154 },
  { _id: 'Personal and Lifestyle', num_of_categories: 134 },
  { _id: 'Top', num_of_categories: 119 },
  { _id: 'Current Events', num_of_categories: 139 },
  { _id: 'biology', num_of_categories: 122 },
  { _id: 'history', num_of_categories: 138 },
  { _id: 'Educational Experiences', num_of_categories: 116 },
  { _id: 'Have You Ever … ?', num_of_categories: 127 },
  { _id: 'science', num_of_categories: 133 },
  { _id: 'politics', num_of_categories: 136 },
  { _id: 'Local News and Events', num_of_categories: 121 },
  { _id: 'art', num_of_categories: 150 }
]
```

Figure 12.25: Query and result

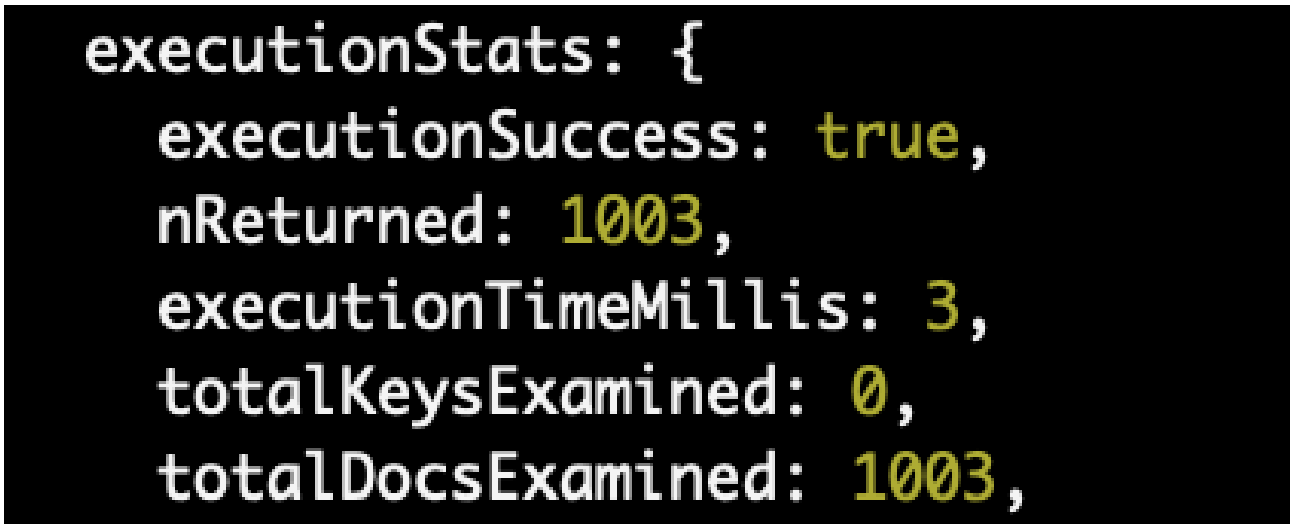


Figure 12.26: Statistics

13. **Query #13**

    This query shows how many articles in the Italian language were written by each journal.

    First it finds all the articles which are written in Italian using the $match command and selecting the "readability.language" field, which is contained in the "readability" subdocument.

    Then it uses $group to show without repetition the journals and their corresponding number of articles written in Italian(using $count).

```
SMBUD> db.Articles.aggregate({$match: {"readability.language": "Italian"}}, {$group: {_id:"$journal", "number_of_articles_in_italian": {$count: {} } } } )
[
  { _id: 'The Writing Cooperative', number_of_articles_in_italian: 15 },
  { _id: 'UX Collective', number_of_articles_in_italian: 19 },
  { _id: 'Data Driven Investor', number_of_articles_in_italian: 36 },
  {
    _id: 'The Startup - New and updated edition',
    number_of_articles_in_italian: 99
  },
  { _id: 'Better Marketing', number_of_articles_in_italian: 18 },
  { _id: 'Better Humans', number_of_articles_in_italian: 2 },
  { _id: 'Towards Data Science', number_of_articles_in_italian: 45 }
]
```

Figure 12.27: Query and result

```
executionStats: {
  executionSuccess: true,
  nReturned: 7,
  executionTimeMillis: 6,
  totalKeysExamined: 0,
  totalDocsExamined: 1003,
```

Figure 12.28: Statistics

14. **Query #14**

    This query finds all the articles which both have more than 4 sections of text AND 4 figures in it. To do so, it utilizes an aggregate() pipeline.

    First, it extract three information from each article: its title, the number of sections and the number of figures. The latter two are retrieved by looking respectively in the "sections.text" and "sections.figures" arrays, which are both contained in the "sections" subdocument, and then by performing the $size command on each of them, that returns the number of elements of an array.

    In the second part, the query utilizes $match to select only the articles with the characteristics described above. Only the title, number of sections and number of figures are returned.

```
SMBUD> db.Articles.aggregate({$project: {_id: "$title", numberOfSections: {$size: "$sections.text" }, numberOfFigures: {$size: "$sections.figures"}}},
       {$match: {$and: [{numberOfSections: {$gt: 4} }, {numberOfFigures: {$gt: 4}}]}})
  {
    _id: 'Data Science Interview Questions',
    numberOfSections: 5,
    numberOfFigures: 5
  },
  {
    _id: 'Your first 500 fans: Lessons learned from one year of consistent writing',
    numberOfSections: 5,
    numberOfFigures: 5
  },
  {
    _id: 'You Might Want to Put That Biodegradable Fork Down',
    numberOfSections: 5,
    numberOfFigures: 5
  },
  {
    _id: 'What I Want To Tell My Younger Self',
    numberOfSections: 5,
    numberOfFigures: 5
  },
  {
    _id: 'The Cellular Space Race is Leaving Rural America Behind',
    numberOfSections: 5,
    numberOfFigures: 5
  },
  {
    _id: 'Predicting vs. Explaining',
    numberOfSections: 5,
    numberOfFigures: 5
  },
  {
    _id: 'Sentiment Analysis: a benchmark',
    numberOfSections: 5,
    numberOfFigures: 5
  },
  {
    _id: 'How does Facebook define Terrorism in Relation to Artificial Intelligence?',
    numberOfSections: 5,
    numberOfFigures: 5
  },
  {
```

Figure 12.29: Query and result

```
executionStats: {
  executionSuccess: true,
  nReturned: 1000,
  executionTimeMillis: 13,
  totalKeysExamined: 0,
  totalDocsExamined: 1000,
```

Figure 12.30: Statistics

1. **Query (update) #1**

   The statement updates a filed of a document, the id is used to filter the result and the

"set" operator sets the "number" field to 13

```
Atlas atlas-5ns0yb-shard-0 [primary] articles> db.articles.updateOne({"_id":0}, {"$set": {"number": 13}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Figure 12.31: Update

2. **Query (update) #2**

   The statement updates a document array: using the id the element to update is retrieved, then a new bibliography is pushed in the array containing all the linked bibliographies.

```
Atlas atlas-5ns0yb-shard-0 [primary] articles> db.articles.updateOne({"_id":0}, { $push: {"bibliography": 40}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Atlas atlas-5ns0yb-shard-0 [primary] articles>
```

Figure 12.32: Update

3. **Query (update) #3**

   This query updates all existing articles that were published by the journal "The Startup", by changing the "journal" attribute to "The Startup – New and updated edition".

```
SMBUD> db.Articles.updateMany({journal: "The Startup"}, {$set: {journal: "The Startup - New and updated edition"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 458,
  modifiedCount: 458,
  upsertedCount: 0
}
```

Figure 12.33: Query

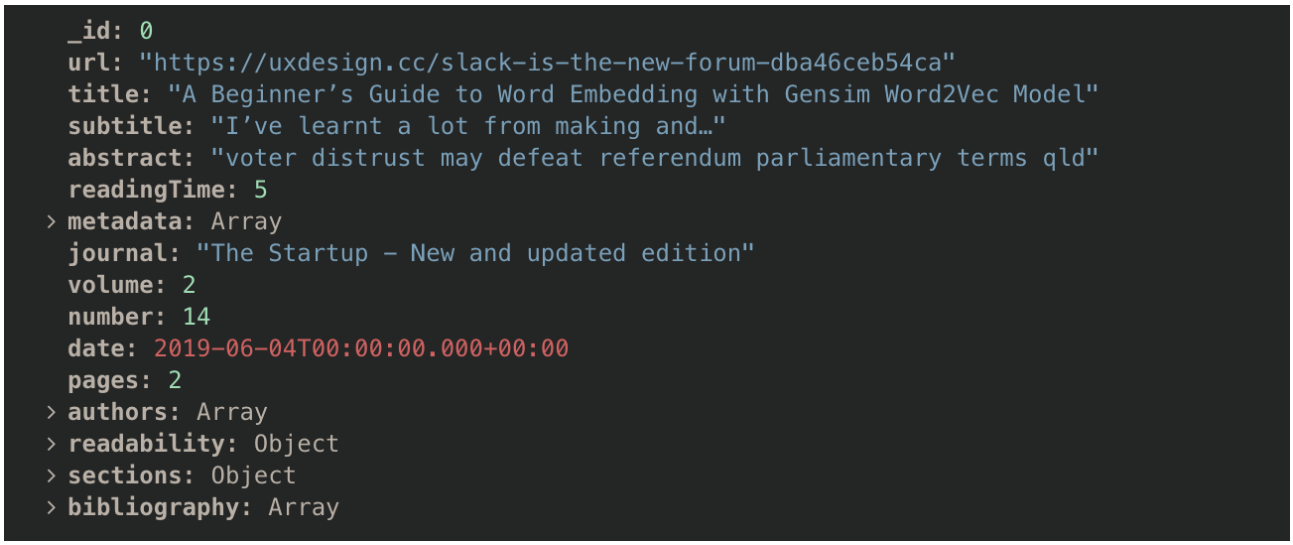


Figure 12.34: Result: notice that the 'journal' field is now: 'The Startup – New and updated edition'

4. **Query (creation) #4**

   This query inserts a new article in the dataset. The new article is missing almost every attribute that the other documents in the collection have (eg. the article has no text or author).

   But, being MongoDB a schema-less database, there are no problems with this operation.

```
SMBUD> db.Articles.insertOne({
...   "url": "https://medium.com/datadriveninvestor/deploy-your-machine-learning-model-using-flask-made-easy-now-635d2f12c50c",
...   "title": "2024 is Mr. Orwell's 1984",
...   "subtitle": "Any company that claims leadership in its industry…",
...   "abstract": "elderly couple killed during street drag race",
...   "journal": "The Writing Cooperative"
... })
{
  acknowledged: true,
  insertedId: ObjectId("637d07102b30ca3bd443e360")
}
```

Figure 12.35: Query for creation

5. **Query (creation) #5**

   With this query we created a second collection that contained information extracted from the articles collection.

   Thanks to the aggregation pipeline we :

   (a) Filter for all the documents with at least one author that has Tom in the name (we did this to improve performance reducing the number of documents for the next stages of the pipeline);

   (b) Unwind with respect to the authors array;

(c) Group by author's name and calculate the number of articles per author and the date of the latest publication;

(d) Filter again by name because before we could not impose that all the authors had Tom in the name but only that an article had an author with such name;

(e) Finally we merge the result in another output collection called tom_articles.

```
> db.articles.aggregate([
    {
      "$match" : {"authors.name" : {"$regex" : /Tom/}}
    },
    {
      "$unwind" : {"path" : "$authors"}
    },
    {
      "$group" : {
        "_id" : "$authors.name",
        "article number" : {"$count" : {}},
        "latest publication" : {"$max" : "$date"}
      }
    },
    {
      "$match" : {"_id" : {"$regex" : /Tom/}}
    },
    {
      "$merge" : {into :"tom_articles"}
    }
    ])
```

Figure 12.36: Query

Figure 12.37: Result

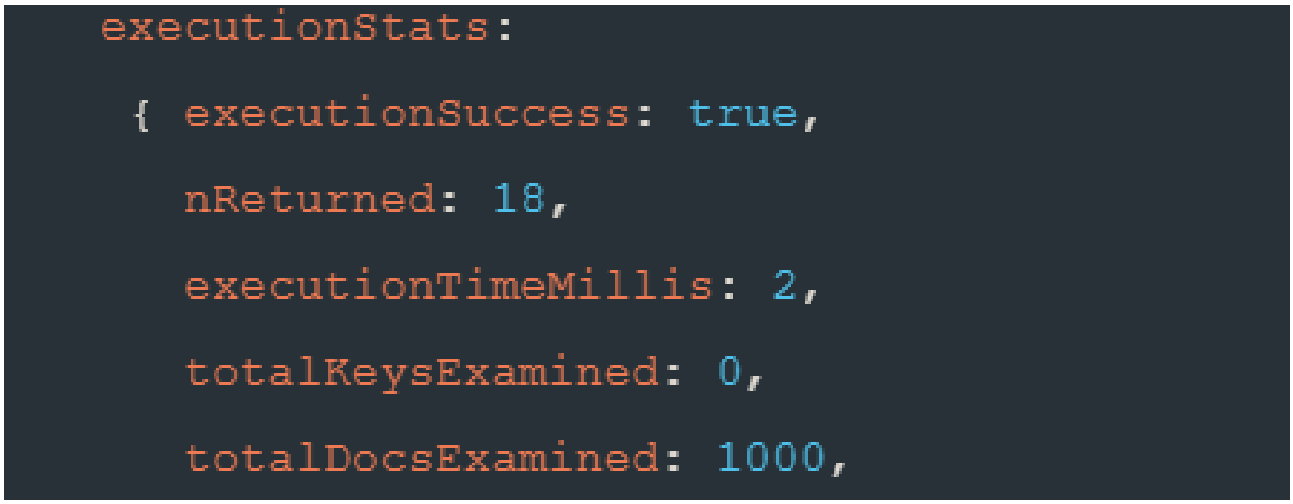


Figure 12.38: Statistics

Part III

Third delivery

13 | Description of the structure of the dataset in pyspark

13.1. Dataset structure

The general structure of the dataset is designed starting from two csv files that we already used for the Neo4j part of the project.

Our data comes from "book" and "incollection" datasets, the commented python code will illustrate all the steps done.

The following images show an example of how collections are made.

author	editor	key	mdate	title	pages	year	month	url
Bir Bhanu		reference/crypt/Bhanu11	2017-07-12	Ear Shape for Biometric Identification.	372-378	2011		db/reference/crypt/c
		reference/crypt/X11w	2017-07-12	ARP Poisoning.	47	2011		db/reference/crypt/c
Marijke De Soete		reference/crypt/Soete11o	2017-07-12	Token.	1305-1306	2011		db/reference/crypt/c
Chen-Mou Cheng		reference/crypt/Cheng11	2017-07-12	Multi-Threaded Implementation for Cryptography and Cryptanalysis.	823-824	2011		db/reference/crypt/c
Shucheng Yu		reference/crypt/YuLR11	2017-09-06	Privacy-Preserving Authentication in Wireless Access Networks.	972-974	2011		db/reference/crypt/c
Wenjing Lou		reference/crypt/YuLR11	2017-09-06	Privacy-Preserving Authentication in Wireless Access Networks.	972-974	2011		db/reference/crypt/c
Kui Ren 0001		reference/crypt/YuLR11	2017-09-06	Privacy-Preserving Authentication in Wireless Access Networks.	972-974	2011		db/reference/crypt/c
Marijke De Soete		reference/crypt/Soete11e	2017-07-12	Issuer.	650	2011		db/reference/crypt/c
		reference/crypt/X11hh	2017-07-12	Malware.	750	2011		db/reference/crypt/c
		reference/crypt/X11jr	2017-07-12	Private Key Cryptosystem.	976	2011		db/reference/crypt/c
		reference/crypt/X11id	2017-07-12	Multicast Stream Authentication.	808	2011		db/reference/crypt/c
Carlisle Adams		reference/crypt/Adams05o	2017-07-12	Personal Identification Number (PIN).		2005		db/reference/crypt/c
Alex Biryukov		reference/crypt/Biryukov05k	2017-07-12	Differential-Linear Attack.		2005		db/reference/crypt/c
Gerrit Bleumer		reference/crypt/Bleumer11n	2017-07-12	Existential Forgery.	433-434	2011		db/reference/crypt/c
Carl M. Ellison		reference/crypt/Ellison11	2017-07-12	Entropy Sources.	421-423	2011		db/reference/crypt/c
		reference/crypt/X11gh	2017-07-12	Inversion in Galois Fields.	632	2011		db/reference/crypt/c

Figure 13.1: Example file csv

13.2. Data description

We modeled the fields as follows:

- **authors**: array of authors, an author is a string type.
- **editors**: array of editors, an editor is a string type.
- **key**: string, it is the unique key of a record.
- **mdate**: dateType, date of last modification.
- **title**: string, unique title for each publication.
- **pages**: string, it tells the starting and ending page of a record.
- **year**: integer, it contains a four-digit number representing the publication year.
- **month**: integer, it contains the publication month.
- **url**: string, publication link.
- **ee**: string, It represents the position of the electronic edition, usually it is a global address.
- **publisher**: string, name of a publishing company.
- **isbn**: string, international standard book number (unique identifier).
- **series**: string.
- **school**: string.
- **booktitle**: string, shortened version of a book title or a convention name.
- **crossref**: string, contains the key of the book.

14 | Python script

14.1. Script used to transform the data in the pyspark format

The following script has been commented out for readability.

```
2   # Import all the modules and functions needed
3   from pyspark.sql import SparkSession
4   from pyspark.sql.types import StructType, StructField, StringType, IntegerType, ArrayType, DateType
5   from pyspark.sql.functions import split, col
6
7   # Create a new Spark connection (local) with name "SMBUD Project"
8   spark = SparkSession.builder.master("local") \
9         .appName("SMBUD Project") \
10        .getOrCreate()
11
```

Figure 14.1: Import pyspark module

```
# Import of Books dataset

# Create schema (with names and types). As CSV files do not support arrays,
# the fields "Authors" and "Editors", whose type should be array, are first
# imported as strings and then converted later.
schema_books = StructType([ \
    StructField("Authors_str", StringType(), True), \
    StructField("Editors_str", StringType(), True), \
    StructField("Key", StringType(), False), \
    StructField("Mdate", DateType(), False), \
    StructField("Title", StringType(), True), \
    StructField("Pages", StringType(), True), \
    StructField("Year", IntegerType(), True), \
    StructField("Month", IntegerType(), True), \
    StructField("Url", StringType(), True), \
    StructField("Ee", StringType(), True), \
    StructField("Cdrom", StringType(), True), \
    StructField("Note", StringType(), True), \
    StructField("Publisher", StringType(), True), \
    StructField("Isbn", StringType(), True), \
    StructField("Series", StringType(), True), \
    StructField("School", StringType(), True) \
])

# Create a temporary dataframe based on the schema above (in which "Authors" and
# "Editors" have a string data type).
df_books_tmp = spark.read.option("delimiter", ";").option("header","true").schema(schema_books).csv("books.csv")

# Create a new dataframe based on the previous one, in which "Authors" and "Editors"
# are split from a string and converted in one array. The columns are then reordered
# using the select() method.
df_books = df_books_tmp.withColumn("Authors", split(col("Authors_str"), ",").cast("array<string>")) \
      .withColumn("Editors", split(col("Editors_str"), ",").cast("array<string>")) \
      .select("Authors", "Editors", "Key", "Mdate", "Title", "Pages", "Year", "Month", \
            "Url", "Ee", "Cdrom", "Note", "Publisher", "Isbn", "Series", "School")

# Print the schema of the final dataframe.
df_books.printSchema()
```

Figure 14.2: Import book dataset

```
##########################
# Import of InCollection dataset

# Same comments as above, the method used is the same.
schema_incol = StructType([ \
    StructField("Authors_str", StringType(), True), \
    StructField("Editors_str", StringType(), True), \
    StructField("Key", StringType(), False), \
    StructField("Mdate", DateType(), False), \
    StructField("Title", StringType(), True), \
    StructField("Pages", StringType(), True), \
    StructField("Year", IntegerType(), True), \
    StructField("Month", IntegerType(), True), \
    StructField("Url", StringType(), True), \
    StructField("Ee", StringType(), True), \
    StructField("Cdrom", StringType(), True), \
    StructField("Note", StringType(), True), \
    StructField("Booktitle", StringType(), True), \
    StructField("Crossref", StringType(), True), \
])

df_incol_tmp = spark.read.option("delimiter", ";").option("header","true").schema(schema_incol).csv("incollections.csv")

df_incol = df_incol_tmp.withColumn("Authors", split(col("Authors_str"), ",").cast("array<string>")) \
      .withColumn("Editors", split(col("Editors_str"), ",").cast("array<string>")) \
      .select("Authors", "Editors", "Key", "Mdate", "Title", "Pages", "Year", "Month", \
            "Url", "Ee", "Cdrom", "Note", "Booktitle", "Crossref")

df_incol.printSchema()
```

Figure 14.3: Import incollection dataset

The output of the script is the printSchema of book and inCollection datasets.

```
root
 |-- Authors: array (nullable = true)
 |     |-- element: string (containsNull = true)
 |-- Editors: array (nullable = true)
 |     |-- element: string (containsNull = true)
 |-- Key: string (nullable = true)
 |-- Mdate: date (nullable = true)
 |-- Title: string (nullable = true)
 |-- Pages: string (nullable = true)
 |-- Year: integer (nullable = true)
 |-- Month: integer (nullable = true)
 |-- Url: string (nullable = true)
 |-- Ee: string (nullable = true)
 |-- Cdrom: string (nullable = true)
 |-- Note: string (nullable = true)
 |-- Publisher: string (nullable = true)
 |-- Isbn: string (nullable = true)
 |-- Series: string (nullable = true)
 |-- School: string (nullable = true)

root
 |-- Authors: array (nullable = true)
 |     |-- element: string (containsNull = true)
 |-- Editors: array (nullable = true)
 |     |-- element: string (containsNull = true)
 |-- Key: string (nullable = true)
 |-- Mdate: date (nullable = true)
 |-- Title: string (nullable = true)
 |-- Pages: string (nullable = true)
 |-- Year: integer (nullable = true)
 |-- Month: integer (nullable = true)
 |-- Url: string (nullable = true)
 |-- Ee: string (nullable = true)
 |-- Cdrom: string (nullable = true)
 |-- Note: string (nullable = true)
 |-- Booktitle: string (nullable = true)
 |-- Crossref: string (nullable = true)
```

15 | Queries

1. **Query #1**

   Using constructs similar to WHERE,LIMIT,LIKE.

   We want to find the first five books sorted from newest to oldest where the title has the word "ANNA" or "Logic" or "Automated Theorem Proving" and the year of publishment was between 1990 and 2000. We return only title,year and authors list.

```
df_books.filter((col("Year") <= 2000) & (col("Year") >= 1990) & ((col("Title").contains("ANNA")) | \
                                                                 (col ("Title").contains("Logic")) |\
                                                                 (col ("Title").contains("Automated Theorem Proving"))))
        .select(df_books.Title,df_books.Year,df_books.Authors)\
        .sort(col("Year") , ascending = False)\
        .limit(5)\
        .show(truncate = False)
```

Figure 15.1: Query 1

```
+-----------------------------------------------+----+-----------------------------------------------------+
|Title                                          |Year|Authors                                              |
+-----------------------------------------------+----+-----------------------------------------------------+
|In Defense of Informal Logic.                  |2000|[Don S. Levi]                                        |
|Mathematical Logic for Computer Science - 2    |1998|[Zhongwan Lu]                                        |
|Fuzzy Logic for Business, Finance, and Management|1997|[George Bojadziev,  Maria Bojadziev,  Lotfi A. Zadeh]|
|Fuzzy-Logic-Based Programming                  |1997|[Chin-Liang Chang]                                   |
|Fuzzy Sets, Fuzzy Logic, Applications          |1996|[George Bojadziev,  Maria Bojadziev]                 |
+-----------------------------------------------+----+-----------------------------------------------------+
```

Figure 15.2: Result

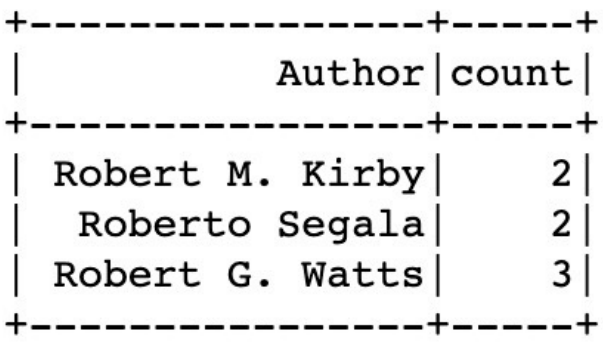
2. **Query #2**

   Using constructs similar to WHERE,GROUPBY.

   We want to find all the authors that have more than one publication that have a name containing "Robert". We filtered all the books that are not affiliated to any school and written after 1987. We grouped then by author name and counted in how many books they appeared.

```
from pyspark.sql.functions import explode, col
#explode along the authors array
exploded_df = df_books.filter((col("School").isNull()) & (col("Year") >= 1987))\
                      .select(col("Title"),explode(df_books.Authors))
#renaming the new col into Author
exploded_df = exploded_df.withColumnRenamed("col", "Author")
#filter rows with author name that contins Robert
exploded_df = exploded_df.filter(col("Author").contains("Robert"))
#group by author name and count
authors_numbers = exploded_df.groupBy("Author").count().filter(col("count") > 1).limit(3).show()
```

Figure 15.3: Query 2



```
+----------------+-----+
|          Author|count|
+----------------+-----+
| Robert M. Kirby|    2|
|  Roberto Segala|    2|
| Robert G. Watts|    3|
+----------------+-----+
```

Figure 15.4: Result

3. **Query #3**

   Using a subquery,WHERE,GROUPBY.

   We wanto to find all the books written by authors that worked with Roberto Segala and how many authors worked at their publication. We first filter through all the books that contain Roberto Segala as an author and collect in a list called collab the result. Then we explode the books written after 1972 and before 2000 along the author array so that we can match all the raws that contain a person in the collab list. Finally we group by title of thenbook snd count ho many rows (aka different authors) there are (Roberto Segala included)

```
#import of the necessary functions
from pyspark.sql.functions import explode, col
#creation of the list collab that has an author that worked with Roberto Segala and himself as an element
collab = df_books.filter(array_contains(df_books.Authors," Roberto Segala"))\
                .select(explode(df_books.Authors))\
                .collect()
#extraction of a normal array from the collab dat structure
collab = [col[0] for col in collab]
#explosion over all authors
exploded_df = df_books.filter(col("Year").between(1972,2019))\
                      .select(col("Title"),explode(df_books.Authors))
exploded_df = exploded_df.withColumnRenamed("col", "Author")
#filter of the books that have an author in the collab list and countign how many authors it has
books = exploded_df.filter(col("Author").isin(collab))\
                   .groupBy("Title")\
                   .count()\
                   .show(truncate = False)
```

Figure 15.5: Query 3

```
+-----------------------------------------------+-----+
|Title                                          |count|
+-----------------------------------------------+-----+
|The Theory of Timed I/O Automata               |4    |
|The Theory of Timed I/O Automata, Second Edition|4    |
+-----------------------------------------------+-----+
```

Figure 15.6: Result

4. **Query #4**

   Using WHERE, IN, Nested Query.

   Find all the books (and their authors) published from 2020 onward, whose title starts with 'A', and written by one or two authors.

```
# Find all the books (and their authors) published from 2020 onward, whose title starts with 'A', and written
# by one or two authors

# List of Rows containing the books published from 2020 onward, whose title starts with 'A'
recent_a_books = df_books.filter((col("Year") >= 2020) & (col("Title").startswith("A"))).select("Title").collect()
# Extract the value of the "Title" field
recent_a_books = [rab[0] for rab in recent_a_books]
# List of Rows containing the books written by three or more authors
three_authors_books = df_books.filter(size(col("Authors"))>=3).select("Title").collect()
# Extract the value of the "Title" field
three_authors_books = [tab[0] for tab in three_authors_books]
# Filtering the books whose title is in the "recent_a_books" but not in "three_authors_books". Books with no
# authors are discarded, too.
# Then only "Title" and "Authors" are selected
result = df_books.filter(col("Title").isin(recent_a_books) & (col("Title").isin(three_authors_books) == False)
                 & (size(col("Authors"))>0)).select(col("Title"),col("Authors"))
# Sorting the result
sorted_result = result.sort(col("Title"))
# Showing the result
sorted_result.show(sorted_result.count(),vertical=True,truncate=False)
```

Figure 15.7: Query 4

```
-RECORD 0-----------------------------------------------------------------------------------------------------------------
 Title   | A Survey of Blur Detection and Sharpness Assessment Methods
 Authors | [Juan Andrade]
-RECORD 1-----------------------------------------------------------------------------------------------------------------
 Title   | Abstraction in Ontology-based Data Management
 Authors | [Gianluca Cima]
-RECORD 2-----------------------------------------------------------------------------------------------------------------
 Title   | Adaptive and Personalized Visualization
 Authors | [Alvitta Ottley]
-RECORD 3-----------------------------------------------------------------------------------------------------------------
 Title   | Advanced Metaheuristic Algorithms and Their Applications in Structural Optimization
 Authors | [Ali Kaveh,  Kiarash Biabani Hamedani]
-RECORD 4-----------------------------------------------------------------------------------------------------------------
 Title   | Affine Arithmetic Based Solution of Uncertain Static and Dynamic Problems
 Authors | [Snehashish Chakraverty,  Saudamini Rout]
-RECORD 5-----------------------------------------------------------------------------------------------------------------
 Title   | Affordance Theory in Game Design: A Guide Toward Understanding Players
 Authors | [Hamna Aslam,  Joseph Alexander Brown]
-RECORD 6-----------------------------------------------------------------------------------------------------------------
 Title   | Aggregation Operators for Various Extensions of Fuzzy Set and Its Applications in Transportation Problems
 Authors | [Akansha Mishra,  Amit Kumar 0003]
-RECORD 7-----------------------------------------------------------------------------------------------------------------
 Title   | Algorithms on Trees and Graphs - With Python Code, Second Edition
 Authors | [Gabriel Valiente]
-RECORD 8-----------------------------------------------------------------------------------------------------------------
 Title   | An Introduction to Numerical Methods for the Physical Sciences
 Authors | [Colm T. Whelan]
-RECORD 9-----------------------------------------------------------------------------------------------------------------
 Title   | An Introduction to Proofs with Set Theory
 Authors | [Daniel A. Ashlock,  Colin Lee]
-RECORD 10----------------------------------------------------------------------------------------------------------------
```

Figure 15.8: Result

5. **Query #5**

   Using GROUP BY, HAVING, AS.

   Find the books with the lowest number of editors published by Springer during the year with the highest number of total publications.

   First all the books are grouped by year of publication, then they are sorted according to the number of publication, and finally only the year which appears at first in the list is extracted. Then the explosion of the dataframe's editor column and the grouping that follows allow to count the number of editors per book. The result is obtained by checking the publication year (that needs to be equal to the year extracted in the first step), by filtering the Publisher and finally by sorting the books according to the number of editors.

```
# Find the books with the lowest number of editors published by Springer during the year with the highest
# number of total publications

# First all the books are grouped by year of publication, then they are sorted according to the number
# of publication, and finally only the year which appears at first in the list is extracted.
# Then the explosion of the dataframe's editor column and the grouping that follows allow to count the
# number of editors per book.
# The result is obtained by checking the publication year (that needs to be equal to the year extracted
# in the first step), by filtering the Publisher and finally by sorting the books according to the number of editors

# List of Rows containing the Year with the highest number of books
best_year = df_books.groupby("Year").count().sort(col("count"), ascending = False).limit(1)
    .select(col("Year")).collect()

# Extract the value of the "Year" field
best_year = [by[0] for by in best_year]

# Filtering the books published in the chosen year
# The column "Editors" is exploded then renamed
exploded_editors = df_books.filter(col("Year").isin(best_year))
    .select(col("Title"),col("Publisher"),explode(col("Editors")),col("Authors"))
exploded_editors = exploded_editors.withColumnRenamed("col", "Editor")

# Books published by Springer grouped by title with the additional "Number of Editors" column containing
# the number of editor of a book
grouped_Springer_books = exploded_editors.filter(col("Publisher")=="Springer")
    .groupBy(col("Title")).agg(count("Editor").alias("Number of Editors"))

# Sorting the books by number of editor
# Only the first 10 publications are shown
grouped_Springer_books = grouped_Springer_books.sort(col("Number of Editors"), ascending = True).limit(10)
    .show(truncate=False)
```

Figure 15.9: Query 5

```
+-----------------------------------------------------------------------------------------------------------------+-
----------------+
|Title                                                                                                            |N
umber of Editors|
+-----------------------------------------------------------------------------------------------------------------+-
----------------+
|Modeling and Control of Dialysis Systems - Volume 2: Biofeedback Systems and Soft Computing Techniques of Dialysis|1
|
|Handbook of Human Computation                                                                                    |1
|
|Similarity-Based Pattern Analysis and Recognition                                                                |1
|
|SemProM - Foundations of Semantic Product Memories for the Internet of Things                                    |1
|
|Artificial Intelligence, Evolutionary Computing and Metaheuristics - In the Footsteps of Alan Turing             |1
|
|Your Virtual Butler - The Making-of                                                                              |1
|
|Recent Advances in Computational Optimization                                                                    |1
|
|Hybrid Metaheuristics                                                                                            |1
|
|Modelling and Control of Dialysis Systems - Volume 1: Modeling Techniques of Hemodialysis Systems                |1
|
|Advances in Probabilistic Databases for Uncertain Information Management                                         |2
|
+-----------------------------------------------------------------------------------------------------------------+-
----------------+
```

Figure 15.10: Result

6. **Query #6**

   Using WHERE, GROUP BY, HAVING, 1 JOIN.

Find all the books titled "Encyclopedia" which are published before 2015, whose editor is more than one person and that contain more than 500 incollections.

First all the books are filtered by the number of editors. Then all the incollections are filtered by booktitle and by publication year. The two filtered dataframes are joined through the key/crossref fields. The number of parts of every single book is obtained by grouping them according to their booktitle, then it is possible to filter the books in order to discard the ones with less than 500 parts.

```
# Find all the books titled "Encyclopedia" which are published before 2015, whose editor is more than
# one person and that contain more than 500 incollections

# First all the books are filtered by the number of editors. Then all the incollections are filtered
# by booktitle and by publication year.
# The two filtered dataframes are joined through the key/crossref fields.
# The number of parts of every single book is obtained by grouping them according to their booktitle,
# then it is possible to filter the books in order to discard the ones with less than 500 parts

# Filtering the books with more than one editor
# Only the "Key" field is selected because it is necessary for the join
filtered_books = df_books.filter(size(col("Editors")) > 1).select("Key")

# Filtering the incollections whose name contains the word "Encyclopedia" and published before 2015
old_encyclopedias = df_incol.filter(col("Booktitle").contains("Encyclopedia") & (col("Year") < 2015))
    .select("Title","Booktitle","Crossref")

# Performing the join
joined_pub = filtered_books.join(old_encyclopedias, filtered_books.Key == old_encyclopedias.Crossref,"inner")

# Grouping the joined dataframe and counting the parts of every encyclopedia
# Encyclopedias filtered by number of parts
result = joined_pub.groupby("Booktitle").agg(count("Title").alias("Number of parts"))
    .filter(col("Number of parts") > 500)

# Showing the sorted result
result.sort(col("Number of parts"),ascending=True).show(truncate=False)
```

Figure 15.11: Query 6

```
+-------------------------------------------------+---------------+
|Booktitle                                        |Number of parts|
+-------------------------------------------------+---------------+
|Encyclopedia of Cryptography and Security (2nd Ed.)|1167           |
+-------------------------------------------------+---------------+
```

Figure 15.12: Result

7. **Query #7**

   Using GROUP BY, 1 JOIN, AS.

   Count how many incollections there are for each publisher.

```
#GROUP BY, 1 JOIN, AS

#count how many incollections there are for each publisher

# inner join: this means that if a book has no incollection it will not appear in the final result and its
# publisher will not be
# considered
df_join = df_books.join(df_incol, df_books.Key == df_incol.Crossref, "inner")

# group by publisher and count the number of incollections
df_publishers = df_join.groupby(col("Publisher")).agg(count(col("Publisher")).alias("num of inc"))

df_publishers.show(truncate=False)

# note: the number of publishers is greater but if there is no incollection under that publisher the number
# is 0 and will not
# appear in the result
```

Figure 15.13: Query 7

```
+-------------------------+----------+
|Publisher                |num of inc|
+-------------------------+----------+
|IGI Global               |98        |
|John Wiley & Sons, Inc.  |328       |
|CRC Press                |81        |
|IOS Press                |49        |
|Springer                 |2422      |
+-------------------------+----------+
```

Figure 15.14: Result

8. **Query #8**

   Using WHERE, GROUP BY, HAVING, AS.

   The query select the authors that have written exactly 3 books under the same publisher, the books must have a date between 1990 and 2020.

```
#  WHERE, GROUP BY, HAVING, AS
# the query select the authors that have written exactly 3 books under the same publisher,
# the books must have a date between
# 1990 and 2020

from pyspark.sql.functions import explode, count

df_expanded = df_books.select("*",  explode(df_books.Authors)) \
    .filter(col("Year").between(1990,2020)) \
    .withColumnRenamed("col", "author") \
    .groupby(col("author"), col("Publisher")) \
    .agg(count(col("*")).alias("num_book_by_author"))

result = df_expanded.filter(df_expanded.num_book_by_author == 3)

result.show(truncate=False)
```

Figure 15.15: Query 8

```
+--------------------------+----------------------------+------------------+
|author                    |Publisher                   |num_book_by_author|
+--------------------------+----------------------------+------------------+
| Abhishek Sehgal          |Morgan & Claypool Publishers|3                 |
| Ahmad-Reza Sadeghi       |Morgan & Claypool Publishers|3                 |
| Ahmed A. Kishk           |Morgan & Claypool Publishers|3                 |
| Amit Kumar 0003          |Springer                    |3                 |
| Arthur C. Sanderson      |WorldScientific             |3                 |
| Atef Elsherbeni          |Morgan & Claypool Publishers|3                 |
| Atef Z. Elsherbeni       |Morgan & Claypool Publishers|3                 |
| Bojan Jovanovic          |Springer                    |3                 |
| Caroline A. Baillie      |Morgan & Claypool Publishers|3                 |
| Christian Borgelt        |Springer                    |3                 |
| Constantinos S. Pattichis|Morgan & Claypool Publishers|3                 |
| Daniel J. Krause         |Morgan & Claypool Publishers|3                 |
| David C. Farden          |Morgan & Claypool Publishers|3                 |
| Diana Inkpen             |Morgan & Claypool Publishers|3                 |
| Eduardo García-Río       |Morgan & Claypool Publishers|3                 |
| Elizabeth J. Chang       |Springer                    |3                 |
| Enrico Santi             |Morgan & Claypool Publishers|3                 |
| Eric Kao                 |Morgan & Claypool Publishers|3                 |
| Ernst-Rüdiger Olderog    |Springer                    |3                 |
| Fan Yang                 |Morgan & Claypool Publishers|3                 |
+--------------------------+----------------------------+------------------+
only showing top 20 rows
```

Figure 15.16: Result

9. **Query #9**

   Using WHERE, GROUP BY, HAVING, 2 JOINs.

The query first selects the authors that have written multiple books and collect them in a list, then uses the list to choose the books that they have written and using a join retrives their linked incollection. Finally group the incollections by year and keeps only the years with 5 incollections.

The correct output is: 0 rows.

```
# WHERE, GROUP BY, HAVING, 2 JOINs
# the query first selects the authors that have written multiple books and collect them in a list,
# then uses
# the list to choose
# the books that they have written and using a join retrives their linked incollection. Finally
# group the incollections by
# year and keeps only the years with 5 incollections

#create new df with title and authors that match the condition
df_exploded1 = df_books.select(col("*"), explode(df_books.Authors)) \
    .withColumnRenamed("col", "author")

df_exploded2 = df_books.select(col("*"), explode(df_books.Authors)) \
    .withColumnRenamed("col", "author2").withColumnRenamed("Title", "title2")

#self join renaming the columns
df_join = df_exploded1.join(df_exploded2, df_exploded1.author == df_exploded2.author2 , "inner")

#select the authors that have multiple books' titles on the same row
df_authors_multiple_books = df_join.filter(df_join.Title != df_join.title2).select("author").distinct().collect()

#create the list with wanted authors (selects first element for each element in the list generated by the collect)
list_authors_multiple_books = [x[0] for x in df_authors_multiple_books]

#from df_exploded only books with that authors are chosen
wanted_books = df_exploded.filter(col("author").isin(list_authors_multiple_books)).distinct()

#selects the incollections extracted from the chosen books
wanted_incollection = df_incol.join(wanted_books, df_incol.Crossref == wanted_books.Key , "leftsemi")

grouped_year = wanted_incollection.groupby(col("Year")) \
    .agg(count(col("*")).alias("num_inc"))

result = grouped_year.filter(grouped_year.num_inc == 5)

result.show(truncate=False, vertical=True)
```

```
(0 rows)
```

Figure 15.17: Query 9

10. **Query #10**

    Using WHERE, JOIN.

    The query remove from the list off all the authors those that have written multiple books and whose name starts with a vocal. .

In [2]:
```
#query1: WHERE, JOIN
#the query remove from the list off all the authors those that have written multiple books and whose name starts with a vocal

from pyspark.sql.functions import *

#create array with vocals and reg_expression to match names that start with a vocal
vocals = ["A", "E", "I", "O", "U"]
reg_str = r"^("+ "|".join(vocals) + ")"

#create new df with title and authors that match the condition
df_exploded = df_books.select(col("Title"), explode(df_books.Authors)) \
    .withColumnRenamed("col", "author") \
    .filter(col("author").rlike(reg_str))


#self join renaming the columns
df_join = df_exploded.join(df_exploded.withColumnRenamed("Title", "Title2").withColumnRenamed("author", "author2"), df_exploded.a

#select the authors that have multiple books' titles on the same row
df_authors_multiple_books = df_join.filter(df_join.Title != df_join.Title2).select("author").distinct()


#remove from the list of all authors the authors that have written multiple books
result = df_books.select(explode(df_books.Authors)) \
    .withColumnRenamed("col", "author") \
    .distinct()\
    .subtract(df_authors_multiple_books)

result.show(truncate=False, vertical=True)
```

```
-RECORD 0--------------------------
 author | Dennis Anderson
```

Figure 15.18: Query 10

```
-RECORD 0---------------------------
 author | Dennis Anderson
-RECORD 1---------------------------
 author | Lukasz Golab
-RECORD 2---------------------------
 author | Krzysztof Cpalka
-RECORD 3---------------------------
 author | Georgios Chalkiadakis
-RECORD 4---------------------------
 author |  Mohammed Ismail 0001
-RECORD 5---------------------------
 author |  Ulrike Stege
-RECORD 6---------------------------
 author |  Alex Post
-RECORD 7---------------------------
 author |  Zhu Han 0001
-RECORD 8---------------------------
 author |  Claudio Gutierrez 0001
-RECORD 9---------------------------
 author |  John Newsome Crossley
-RECORD 10--------------------------
 author | Sen Lin
-RECORD 11--------------------------
 author | Suresh Sundaram 0002
-RECORD 12--------------------------
 author | Grace Hui Yang
-RECORD 13--------------------------
 author |  Machavaram V. Kartikeyan
-RECORD 14--------------------------
 author |  Stefan Brunthaler
-RECORD 15--------------------------
 author |  Daniel Zaldivar 0001
-RECORD 16--------------------------
 author |  Michael M. Bronstein
-RECORD 17--------------------------
```

Figure 15.19: Result

Creation/Update

1. **Query #1**

   Data creation using two subqueries, WHERE, LIKE, SELECT, DISTINCT.

   Creation of a new df_only_incollection_writers from data contained in df_books and df_incol. We simply select all the authors from the explosion of the books and incollections dataframes distinctly and then we remove from the elements in the df_authors_2 the ones from the other dataframe.

```
df_authors_1 = df_books.filter(col("Title").contains("Data")).select(explode(df_books.Authors)).distinct()
df_authors_2 = df_incol.filter(col("Title").contains("Data")).select(explode(df_incol.Authors)).distinct()

df_author = df_authors_2.subtract(df_authors_1).withColumnRenamed("col", "Author").limit(10).show()
```

Figure 15.20: Query 1

```
+--------------------+
|              Author|
+--------------------+
|  Maryann E. Martone|
|          Janet Mann|
|    Robin A. A. Ince|
|Minos N. Garofalakis|
|  Abdullah Uz Tansel|
|    Simon R. Schultz|
|    David N. Kennedy|
|        Ulrike Cress|
|      Emery N. Brown|
|     Kristen LeFevre|
+--------------------+
```

Figure 15.21: Result

2. **Query #2**

   Data creation using two subqueries, WHERE,SELECT,INNER JOIN.

   Creation of a new dataframe joining the articles and books published after year 2000 thanks to the crossref foreign key.

```
new_articles = df_incol.filter(col("Year") >= 2000).select(col("Crossref") , col("Title"))
new_books = df_books.filter(col("Year") >= 2000).select(col("Title") , col("Key"))
new_books = new_books.withColumnRenamed("Title", "Book_title")

new_df = new_books.join(new_articles , new_books.Key == new_articles.Crossref , "inner")\
                    .select(col("Title") , col("Book_title")).show(truncate = False)
```

Figure 15.22: Query 2

```
+------------------------------------------------------------------+----------------------------------------------
+
|Title                                                             |Book_title
|
+------------------------------------------------------------------+----------------------------------------------
+
|Ear Shape for Biometric Identification.                           |Encyclopedia of Cryptography and Security, 2nd Ed.
|
|ARP Poisoning.                                                    |Encyclopedia of Cryptography and Security, 2nd Ed.
|
|Token.                                                            |Encyclopedia of Cryptography and Security, 2nd Ed.
|
|Multi-Threaded Implementation for Cryptography and Cryptanalysis.|Encyclopedia of Cryptography and Security, 2nd Ed.
|
|Privacy-Preserving Authentication in Wireless Access Networks.    |Encyclopedia of Cryptography and Security, 2nd Ed.
|
|Issuer.                                                           |Encyclopedia of Cryptography and Security, 2nd Ed.
|
|Malware.                                                          |Encyclopedia of Cryptography and Security, 2nd Ed.
|
|Private Key Cryptosystem.                                         |Encyclopedia of Cryptography and Security, 2nd Ed.
|
|Multicast Stream Authentication.                                  |Encyclopedia of Cryptography and Security, 2nd Ed.
|
|Personal Identification Number (PIN).                             |Encyclopedia of Cryptography and Security
|
|Differential-Linear Attack.                                       |Encyclopedia of Cryptography and Security
|
|Existential Forgery.                                              |Encyclopedia of Cryptography and Security, 2nd Ed.
|
|Entropy Sources.                                                  |Encyclopedia of Cryptography and Security, 2nd Ed.
|
|Inversion in Galois Fields.                                       |Encyclopedia of Cryptography and Security, 2nd Ed.
|
|Naccache-Stern Higher Residues Cryptosystem.                      |Encyclopedia of Cryptography and Security, 2nd Ed.
|
|Blum-Goldwasser Public Key Encryption System.                     |Encyclopedia of Cryptography and Security
|
|Field.                                                            |Encyclopedia of Cryptography and Security, 2nd Ed.
|
|Relatively Prime.                                                 |Encyclopedia of Cryptography and Security
|
|Alphabet.                                                         |Encyclopedia of Cryptography and Security
|
|Electronic Cheque.                                                |Encyclopedia of Cryptography and Security
|
+------------------------------------------------------------------+----------------------------------------------
+
only showing top 20 rows
```

Figure 15.23: Result

3. **Query #3**

   Create a new dataframe that contains all the series with more than 100 books, sorted by their most recent update.

```
# Create a new dataframe that contains all the series with more than 100 books, sorted by their most recent update

# Grouping the books by their belonging series
# "Number of books" column is created for the filtering
# "Most Recent Publication" column is created for the  sorting
temp_series = df_books.groupby(col("Series")).agg(count(col("Key")).alias("Number of books"),max(col("Year"))
                                                  .alias("Most Recent Publication"))
# Filtering the series to check if they fulfill the requirements
filtered_series = temp_series.filter((col("Number of Books") > 100) & (col("Series")!="null"))
# Creation of the requested dataframe
series_df = filtered_series.sort(col("Most Recent Publication"), ascending=False)
# Showing the resulting dataframe
series_df.show(truncate=False)
```

Figure 15.24: Query 3

```
+-----------------------------------------------------+---------------+-----------------------+
|Series                                               |Number of books|Most Recent Publication|
+-----------------------------------------------------+---------------+-----------------------+
|Studies in Computational Intelligence                |553            |2023                   |
|Studies in Fuzziness and Soft Computing              |134            |2022                   |
|Lecture Notes in Computer Science                    |109            |2022                   |
|Springer Briefs in Electrical and Computer Engineering|122            |2019                   |
+-----------------------------------------------------+---------------+-----------------------+
```

Figure 15.25: Result

4. **Query #4**

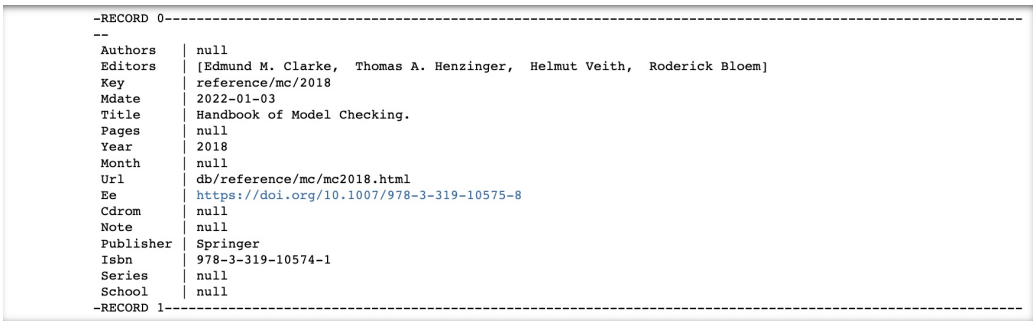
   Create a new dataframe which contains only the books which have an author that is also an editor, or vice versa.

```
# Create a new dataframe which contains only the books which have an author that is also an editor, or vice versa

# First it is necessary to find all the editors and all the authors
# Then, for every book, if one of its authors appears also in the editors set, then the book's title is stored in a
# list. Same procedure for the editors
# Lastly, if a book's title appears in one of the newly created lists, then the corresponding book record is added
# to the new dataframe

# Exploding the editors' column, then obtaining a set containing all the editors with no duplicates
exploded_all_editors = df_books.select(explode("Editors")).withColumnRenamed("col","Editor")
editors_set = exploded_all_editors.select(collect_set("Editor")).collect()[0][0]
# Exploding the authors' column, then obtaining a set containing all the authors with no duplicates
exploded_all_authors = df_books.select(explode("Authors")).withColumnRenamed("col","Author")
authors_set = exploded_all_authors.select(collect_set("Author")).collect()[0][0]
# Exploding the authors' column, then checking if the book's author is also an editor and in case store that
# Title in a list
authors_temp = df_books.select(explode(col("Authors")),col("Title")).withColumnRenamed("col","Author")
authors_titles = authors_temp.filter(col("Author").isin(editors_set)).select("Title").collect()
authors_titles = [at[0] for at in authors_titles]
# Exploding the editors' column, then checking if the book's editor is also an author and in case store that
# Title in a list
editors_temp = df_books.select(explode(col("Editors")),col("Title")).withColumnRenamed("col","Editor")
editors_titles = editors_temp.filter(col("Editor").isin(authors_set)).select("Title").collect()
editors_titles = [et[0] for et in editors_titles]
# Creating the requested dataframe by filtering the titles that appear in the previously created lists
both_df = df_books.filter(col("Title").isin(authors_titles) | col("Title").isin(editors_titles))
# Showing the resulting dataframe
both_df.show(vertical=True,truncate=False)
```

Figure 15.26: Query 4

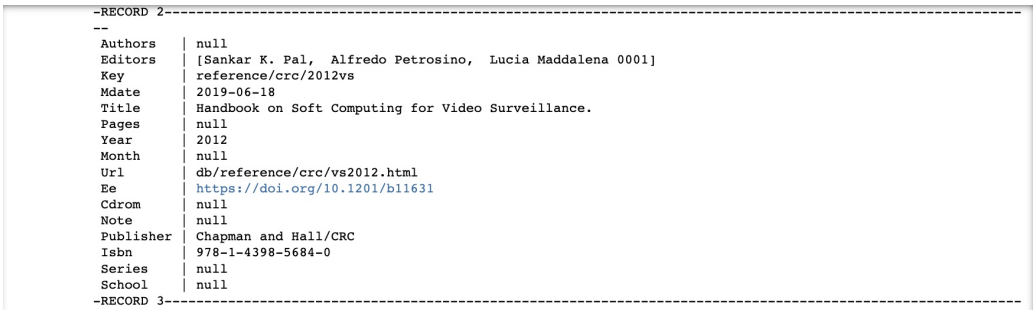


```
-RECORD 0-----------------------------------------------------------------------------------------------------------
--
 Authors   | null
 Editors   | [Edmund M. Clarke,  Thomas A. Henzinger,  Helmut Veith,  Roderick Bloem]
 Key       | reference/mc/2018
 Mdate     | 2022-01-03
 Title     | Handbook of Model Checking.
 Pages     | null
 Year      | 2018
 Month     | null
 Url       | db/reference/mc/mc2018.html
 Ee        | https://doi.org/10.1007/978-3-319-10575-8
 Cdrom     | null
 Note      | null
 Publisher | Springer
 Isbn      | 978-3-319-10574-1
 Series    | null
 School    | null
-RECORD 1-----------------------------------------------------------------------------------------------------------
```

Figure 15.27: Result

```
-RECORD 1-----------------------------------------------------------------------------------------------------------
--
 Authors   | null
 Editors   | [Oded Maimon,  Lior Rokach]
 Key       | reference/dmkdh/2010
 Mdate     | 2020-02-25
 Title     | Data Mining and Knowledge Discovery Handbook, 2nd ed.
 Pages     | null
 Year      | 2010
 Month     | null
 Url       | db/reference/dmkdh/dmkdh2010.html
 Ee        | http://www.springerlink.com/content/978-0-387-09822-7
 Cdrom     | null
 Note      | null
 Publisher | Springer
 Isbn      | 978-0-387-09822-7
 Series    | null
 School    | null
-RECORD 2-----------------------------------------------------------------------------------------------------------
```

Figure 15.28: Result



```
-RECORD 2-----------------------------------------------------------------------------------------------------------
--
 Authors   | null
 Editors   | [Sankar K. Pal,  Alfredo Petrosino,  Lucia Maddalena 0001]
 Key       | reference/crc/2012vs
 Mdate     | 2019-06-18
 Title     | Handbook on Soft Computing for Video Surveillance.
 Pages     | null
 Year      | 2012
 Month     | null
 Url       | db/reference/crc/vs2012.html
 Ee        | https://doi.org/10.1201/b11631
 Cdrom     | null
 Note      | null
 Publisher | Chapman and Hall/CRC
 Isbn      | 978-1-4398-5684-0
 Series    | null
 School    | null
-RECORD 3-----------------------------------------------------------------------------------------------------------
```

Figure 15.29: Result

5. **Query #5**

   Create new dataset: dataset with the avarege year of release for each publisher and choose the 20 with the greater number.

```
# create new dataset: dataset with the avarege year of release for each publisher and choose the 20 with the
# greater number

from pyspark.sql.types import IntegerType

df_books = df_books.withColumn("Year", df_books["Year"].cast(IntegerType()))
year_by_pub = df_books.groupby(col("Publisher")).agg(avg(col("Year")).alias("avg_year"))

year_by_pub = year_by_pub.sort(col("avg_year").desc()).limit(20)
year_by_pub.show(truncate=False)
```

Figure 15.30: Query 5

```
+----------------------------------------+------------------+
|Publisher                               |avg_year          |
+----------------------------------------+------------------+
|Springer International Publishing       |2020.0            |
|GI                                      |2020.0            |
|Cambridge University Press              |2016.0            |
|Éditions RNTI                           |2016.0            |
|Morgan & Claypool Publishers            |2013.680790960452 |
|Springer                                |2013.1483715319662|
|ACM Press and IEEE Computer Society Press|2013.0           |
|Atlantis Press                          |2012.8            |
|IOS Press                               |2012.258064516129 |
|Hermann-Éditions                        |2011.75           |
|null                                    |2011.5172413793102|
|Springer US                             |2011.0            |
|Taylor & Francis                        |2009.6666666666667|
|IGI Global                              |2008.6666666666667|
|Cépaduès-Éditions                       |2008.6666666666667|
|Auerbach Publications                   |2008.0            |
|John Wiley & Sons, Inc.                 |2008.0            |
|Chapman and Hall/CRC                    |2007.888888888889 |
|Elsevier                                |2007.3846153846155|
|Auerbach Publ./CRC Press                |2007.0            |
+----------------------------------------+------------------+
```

Figure 15.31: Result