POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Systems and Methods for Big and Unstructured Data Project

Author(s): **Riccardo Inghilleri (10713236)**

Manuela Merlo (10670533)

Matteo Negro (10642961)

Paolo Pertino (10729600)

Leonardo Pesce (10659489)

Group Number: **2**

Academic Year: 2022-2023

Contents

1 | Introduction

f

1.1. Introduction

Anyone who is writing a thesis, dissertation, or research paper should review the literature and previous findings. He or she is most likely looking for reliable resources, probably peer-reviewed research articles.
The project aims to provide an efficient database storage solution for a large scale dataset recording scientific publications, such as articles, books, journals, conferences, together with their authors, editors, publishers and other metadata.

1.2. Assumptions

The project relies on the hypotheses described below:

- Since from the analysis of the data provided not all authors have a unique ORCID, we decided to provide an additional automatically generated unique Id.
- We decided to use a relationship of type (0,n) between publication and author to allow the inclusion of anonymous publications.
- Since not all publications have a DOI we decided to identify them via an automatically generated Id.
- Author and Editor are both people. Hence it is possible to create an ISA relationship between those entities and a new entity named "Person".
- Some attributes (e.g. "n_citations" in the er-diagram, and "papercount", "citationcount" and "referencecount" in mongoDB documents) could be removed since it is possible to get them by exploring the other relationships. However they can be convenient in queries and since the initial dataset already contained them we decided to keep them.

- Since other types of publication (e.g. Incollections, Web Pages) are possible in addition to the ones we have chosen, we decided to opt for a partial/exclusive type hierarchy for the publications. Furthermore, not everyone has an ISBN so we decided to use an auto-generated Id here as well.
- Publications can cite themselves.
- Information of authors identified by different names was merged in case it could be traced back to the same author. Otherwise, the entities were treated as distinct.
- During data analysis, we checked for the presence of other types of publications (Incollections, Web Pages), but since these were not relevant for our project, we decided not to use them, and therefore not to import them, in our database.
- We decided to model the hierarchy of people as total and overlapping since there might be someone who is both an author and an editor. However, in our implementation of the database, we decided to treat an editor and an author with the same name as different entities.

1.3. DataBase ER Model

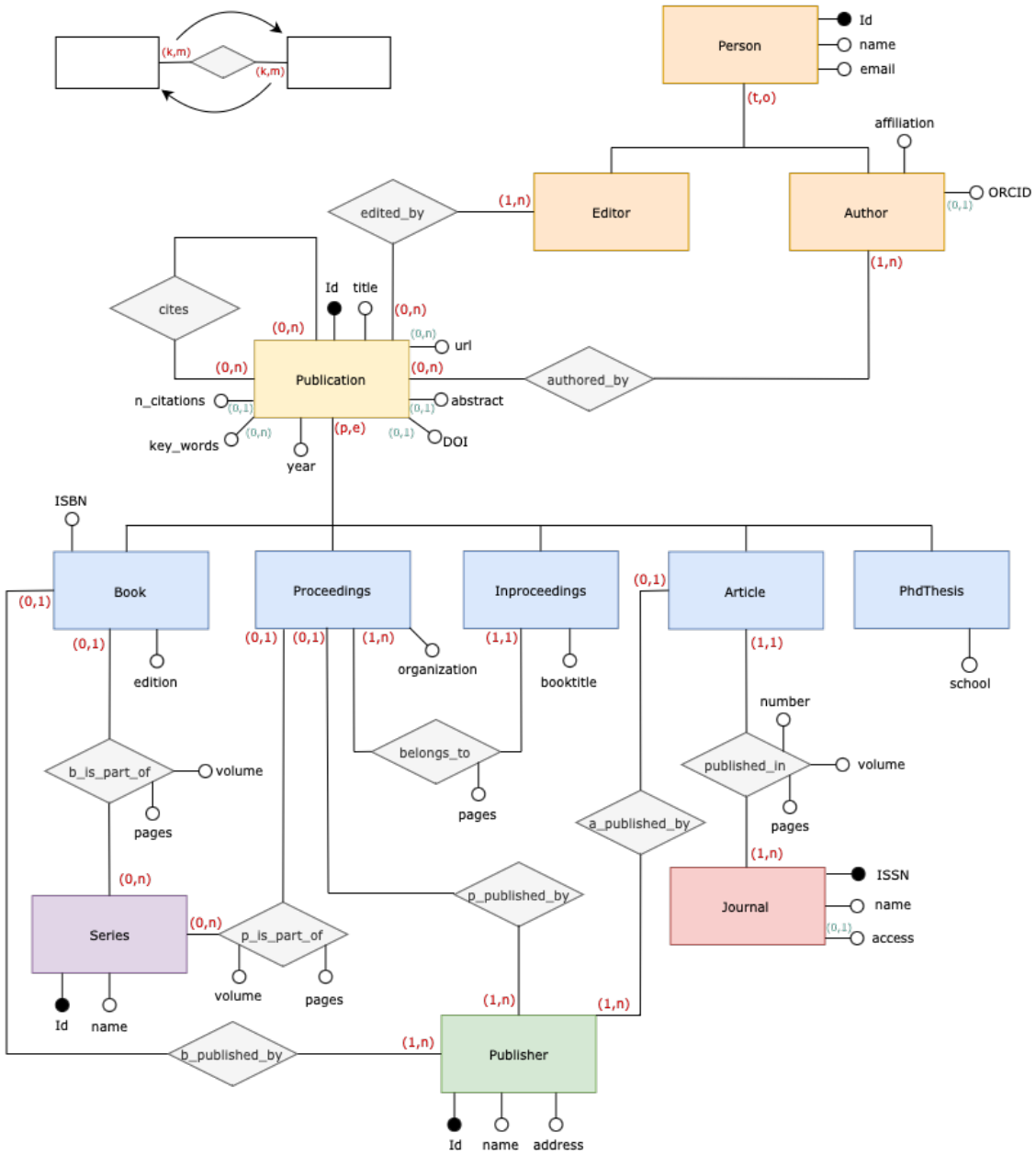


Figure 1.1: *ER schema*

1.3.1. Entities

Publication: a paper identified by a unique Id, written by authors who may be scientists or technicians, on a scientific topic and then published, once validated, through the communication channels of the scientific community.

Attribute	Multiplicity	Description
Id	(1,1)	Self-generated unique identifier
Title	(1,1)	Title of the publication
Year	(1,1)	The year of publication (or, if unpublished, the year of creation)
Url	(0,n)	Internet address
DOI	(0,1)	Unique and never-changing string assigned to online (journal) articles, books, and other works
Abstract	(0,1)	Summary of the publication
Keywords	(0,n)	Words that capture the essence of the paper
n_citation	(0,1)	Number of citations of the publication instance in other papers

We decided to include the most important types of publications:

Book: a book where the publisher is clearly identifiable.

Proceeding: a conference proceeding.

Inproceeding: a paper that has been published in conference proceedings.

PhdThesis: a document submitted in support of candidature for an academic degree or professional qualification presenting the author's research and findings.

Article: an article from a journal, magazine, newspaper, or periodical.

Some entities have specific attributes:

Book

ISBN	(1,1)	Numeric commercial book identifier that is intended to be unique
Edition	(1,1)	Edition number of a book

Proceeding

Organization	(1,1)	Name of the institution that organized or sponsored the conference

Inproceeding

Booktitle	(1,1)	Title of the proceeding it is part of

PhdThesis

School	(1,1)	Name of the university or degree awarding institution where the thesis was written

Person: identifies a person in the database, may the latter be an editor or an author.

Attribute	Multiplicity	Description
Id	(1,1)	Self-generated unique identifier
Name	(1,1)	Name of the Author
email	(0,1)	Electronic mail of the author

Author: represents the individual researcher who has worked on a publication alone or in a group to make public the methods and results of his scientific work. Each author is identified by a unique Id since not all are provided ORCID.

Editor: a person having managerial and sometimes policy-making responsibility related to the writing, compilation, and revision of content for a publishing firm or for a newspaper, magazine, or other publication.

Author

Affiliation	(1,1)	The place (institution) at which the author conducted the research that he has reported / written about
ORCID	(0,1)	Persistent digital identifier that distinguishes a researcher from any other
external_id	(0,n)	Identifier in external organizations

Journal: collection of articles written by researchers, professors and other experts that focus on a specific discipline or field of study.

Attribute	Multiplicity	Description
ISSN	(1,1)	8-digit code used to identify newspapers, journals, magazines and periodicals of all kinds and on all media–print and electronic
Name	(1,1)	Name of the Journal
Access	(0,1)	Open-close access to documents

Publisher: can refer to a publishing company or organization, or to an individual who leads a publishing company, imprint, periodical or newspaper.

Attribute	Multiplicity	Description
Id	(1,1)	Self-generated unique identifier
Name	(1,1)	Name of the publisher
Address	(0,1)	The location of the publisher

Series: a sequence of books, proceedings or PhdThesis having certain characteristics in common that are formally identified together as a group.

1.3.2. Relationship

Authored_by: describes the relationship between a publication and its authors/co-authors.

Edited_by: relationship between a publication and its possible editors.

Published_by: relationship between a publication and its possible publisher.

Cites: recursive relationship between a publication and one or more other publications. Describes the bibliographical references made in the publications.

Published_in: relationship between an article or a proceeding and the journal in which it was published.

Belongs_to: relationship between the inproceeding and the proceeding it belongs to.

Part_of: relationship between book/proceeding and the series they are part of.

Attribute	Multiplicity	Description
Volume	(0,1)	The volume number of the journal or multi-volume book
Number	(0,1)	The issue number for a journal article
Pages	(0,1)	The page range where the article can be found in the specific issue of that journal, or for a section in a book it could be a series of page numbers

2 | Neo4j

2.1. Dataset

The datasets suggested to us (that for the sake of simplicity we will call from now on D1 the one coming from AMiner.org and D2 the DBLP.org xml dump) where beyond messy.

2.1.1. Description and our choice

In D1 the data were given in json format, but trying to import the file lead to an error, stating that it was not complying with the json format. By opening it sequentially in the command line (since its dimension did not allow us to open it with standard editors), we discovered that numbers were encapsulated in the function "Number()" (probably due to some external libraries' way of exporting), thus breaking the json semantics.
On the other hand D2 was in xml format. All the standard parser we tried did not work, but luckily we found a github repository made on purpose for DBLP's dataset. The script give as output some csv files representing the entities with their properties and their links. These files contain information about articles, journals in which they are published in, books, inproceedings and their corresponding proceedings, series, phd and master thesis, as well as their authors, editors and publishers as described in the sections above.

2.1.2. Cleaning

Although we were able to extract data from the xml file, it has still been necessary to create a python script (cucciolodifoca.py) to better re-arrange those information (for examples many attributes were useless, some others had not meaningful names and were not well described by the source or they were not in the place we wanted them to be).
On top of that with the same script we took other data about the authors from a third source [1–3] (Semantic Scholar) and merged them with the one we had extracted from dblp so far.
In the end we came up with a dataset concerning approximately 10 Million

entities and 25 Million relationships.

2.2. Data Upload

As previously discussed, the dataset need some preprocessing operation before being able to be imported in Neo4J. The following sections will go through the whole process.

2.2.1. Pre-processing operations

Requirements:

1. Python 3.*
   You can check if you have already installed python by opening the terminal ›_ and using the following command:

   ```
   python --version
   ```

   If you don't already have it, you can install it using the following linked guide .
2. Go to our github repository and copy into your workspace the **requirements.txt** file.
   The following command will allow you to install all the needed libraries:

   ```
   pip3 install .r requirements.txt
   ```

Steps:

Before importing the database, you need to perform the following instructions:

1. Go to *dbpl site* and download the **dblp.dtd** and the **dblp.xml.gz** files.
2. Go to *Semantic Scholar Dataset mirror* [1–3] and download the **author.cvs.gz** file.
3. Unzip the **dblp.xml.gz** and the **author.cvs.gz** files (You can use WINRAR).
4. Go to *this page* and copy into your folder the file **XMLToCSV.py**
5. Enter the folder where you have downloaded the files, open the terminal ›_ and run this command:

```
python XMLToCSV.py --annotate --neo4j dblp.xml dblp.dtd output.csv
--relations author:authored_by
journal:published_in publisher:published_by
school:submitted_at editor:edited_by
cite:has_citation series:is_part_of
```

Once this script has finished working (estimated time ≈ 20 min) you will have in the directory a bunch of .csv files containing the nodes and the relationships. Since these data are not complete, they need some other pre-processing operations before being able to be imported into Neo4J.

6. Go to our github repository and copy into the folder the **cucciolodifoca.py** file.
   Run it with the command:

   ```
   python cucciolodifoca.py
   ```

7. You will be asked to choose what operation you want to perform:

   - Choose **1** for the Neo4j Setup;

   This script will provide the final restructuring and cleaning operations.

8. We provide, together with the pre-processing script, the **neo4j_import.sh** file. It provides the *neo4j-admin import* command to run in the next steps for your operating system. Open it with a text editor to copy it.

Now you are finally ready to import your dataset into Neo4J.

2.2.2. Upload process

Now that the .csv files have been cleaned up, we can import them into Neo4J.

1. First of all make sure you have installed neo4j desktop:
   In case you don't already have it, download it from *this page*
   If during the installation of the latest version you encounter this problem :

   ```
   Initialization error:Error:Shashum of the downloaded file did not
   match the expected value.  Please try again.
   ```

   Try installing the previous version.

2. Open Neo4J Desktop.

3. Open a new project:

Figure 2.1: *New Project*

4. Create the instance by clicking on ✚ **Add**, selecting **Local DBMS** and choosing a name and a password.
   NB: the default version is the latest, but if it gives you some problems try with the 4.4.5.

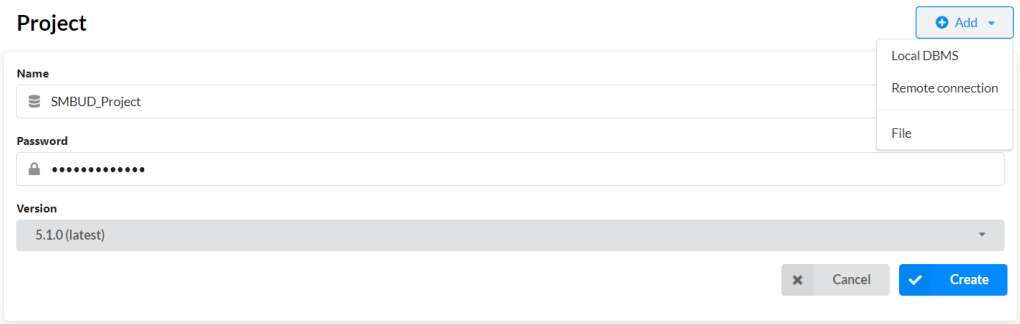


Figure 2.2: *Settings*

5. Once you have created the DBMS, click on **...**, go to **Open Folder** and then click on **Import**. Finally copy all the .csv files you got in the pre-processing phase in the *import* directory which opens.
6. Click again on **...** and open the **terminal**. ›_
7. **Only for Windows users**
   Move to the bin folder by typing:

   ```
   cd bin
   ```

8. Copy in the terminal the content of the file **neo4j_import.sh** we provided, based on your OS (or).
9. Once the process in finished, click on **START**.
10. Click on **Open** and switch from **neo4j** to **system**.

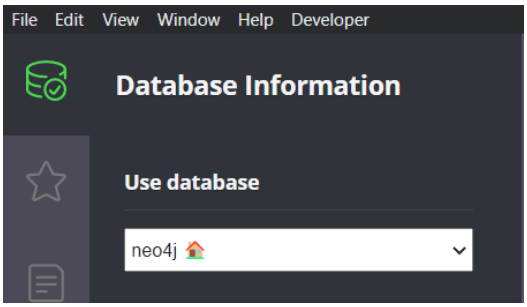


Figure 2.3: *Use database: system*

11. Run the following command:

```
CREATE database dblp.db
```

12. In the same section of (10) change from system to dblp.db

You have successfully imported your data into Neo4J.

2.3. Graph Diagram

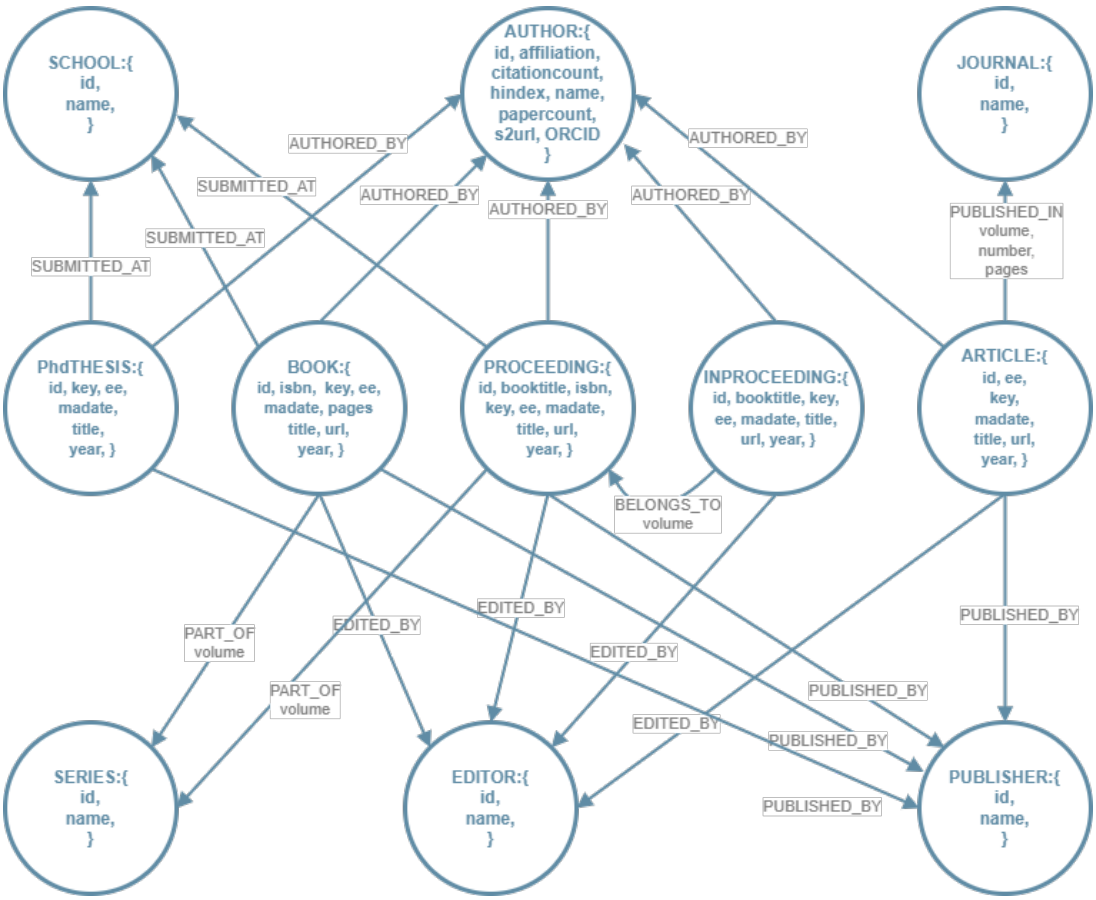


Figure 2.4: *Graph-Diagram*

All publications can cite each other. These were not represented in the diagram for clarity.

2.3.1. Nodes

Most of the attributes have been described in detail in the 1.3.1 Introduction section. Follow examples of nodes with an explanation of the attributes we decided to keep for the graph database in order to lose as less as possible information.
Attributes present in all nodes:

Attribute	Description
key	Unique dblp key of the record
label	Free text descriptive information used as guidelines for some situation

Author


```
{
"author": {
"identity": 9719264,
"labels": [
        "Author" ],
"properties": {
        "citationcount": "2116.0",
        "name": "Marco Brambilla",
        "affiliations": "['Politecnico di Milano']",
        "hindex": "12.0",
        "papercount": "81.0",
        "s2url":"https://www.semanticscholar.org/author/40350773",
        "externalids.ORCID": "['0000-0002-8753-2434']",
      }
    }
}
```


Attribute	Description
citationcount	Number of citations
hindex	An author-level metric that measures both the productivity and citation impact of the publications
papercount	Number of publications
s2url	Url of the page dedicated to the author on Semantic Schoolar
homepage	Url of the homepage of the author

Editor

```
{
"editor": {
"identity": 12286352,
"labels": [
        "Editor"],
"properties": {
        "editor": "Otmar Scherzer"
      }
    }
}
```

Follow a table with the attributes present in most of the node of type Publication:

Attribute	Description
ee	Position of the electronic edition
mdate	The date of the last modification of the record
publtype	Optional attribute that further specifies the type of record (encyclopedia,informal, edited, survey, data, software, withdrawn)

Book

```
{
"book": {
"identity": 3033437,
"labels": [
        "Book",
        "Publication"
      ],
"properties": {
        "ee": ["https://d-nb.info/891654135"],
        "pages": ["1-370" ],
        "mdate": "2021-07-17",
        "year": 1989,
        "book": 1306,
        "isbn": ["978-3-89012-177-2"],
        "title": "Planspieltechnik und Computer-based-Training zur
        Schulung von Einkäufern im Handel.",
        "key": "phd/dnb/Curth89"
```

```
      }
    }
}
```

Proceeding

```
{
"proceeding": {
"identity": 6260243,
"labels": [
        "Proceeding",
        "Publication"
      ],
"properties": {
        "ee": ["https://doi.org/10.1007/978-3-319-20883-1",
        "mdate": "2020-03-27",
        "year": 2015,
        "isbn": ["978-3-319-20882-4"],
        "proceedings": 104049,
        "title": "Handbook of Genetic Programming Applications",
        "key": "reference/genetic/2015",
        "url": ["db/reference/genetic/genetic2015.html"]
      }
    }
}
```

Inproceeding

```
{
"inproceeding": {
"identity": 6283597,
"labels": [
        "Inproceeding",
        "Publication"
      ],
"properties": {
        "ee": ["http://www.mitre.org/support/swee/rosentha.html"],
        "mdate": "2019-07-30",
        "year": 1998,
```

```
        "booktitle": "SWEE",
        "title": "The Future of Classic Data Administration: Objects + Databases + CASE",
        "inproceedings": 3229742,
        "key": "www/org/mitre/future",
        "url": "db/conf/swee/swee1998.html"
      }
    }
}
```


Article


```
{
"article": {
"identity": 4243565,
"labels": [
        "Article",
        "Publication"],
"properties": {
        "ee": ["https://arxiv.org/abs/2101.03757"],
        "volume": "abs/2101.03757",
        "mdate": "2021-01-21",
        "year": 2021,
        "ee-type": ["oa"],
        "publtype": "informal",
        "title": "VaccinItaly: monitoring Italian conversations around vaccines on Twitter.",
        "article": 7485435,
        "key": "journals/corr/abs-2101-03757",
        "url": ["db/journals/corr/corr2101.html#abs-2101-03757"]
      }
    }
  }
```


Phd Thesis


```
{
"phdThesis": {
"identity": 6147540,
```

```
"labels": [
        "PhdThesis"],
"properties": {
        "mdate": "2002-01-03",
        "school": ["University of Texas, Austin" ],
        "year": [1992],
        "title": "Programming Data Structures in Logic.",
        "phdthesis": 3,
        "key": "phd/Turpin92"
      }
    }
  }
```

Journal

```
{
"journal": {
"identity": 5460,
"labels": [ "Journal"],
"properties": {
        "journal": "World Wide Web"
      }
    }
}
```

Series

```
{
"series": {
"identity": 3021851,
"labels": ["Series"],
"properties": {
        "series": "Technical Report / University of Wisconsin, Madison /
   Computer Sciences Department"
      }
    }
}
```

Publisher

```
{
"publisher": {
"identity": 7645,
"labels": ["Publisher"],
"properties": {
            "publisher": "Excelsior"
      }
    }
  }
```

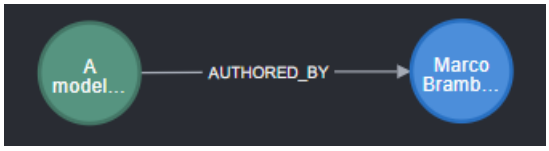

School


```
{
"school": {
"identity": 3079631,
"labels": ["School"],
"properties": {
        "school": "Polytechnic University of Milan, Italy"
      }
    }
  }
```


2.3.2. Links

Most relationships have only one attribute, a unique id.

AUTHORED_BY: link between a Publication and an Author.



EDITED_BY: link between a Publication (except PhdThesis) and an Editor.

CITES: link between two Publications.

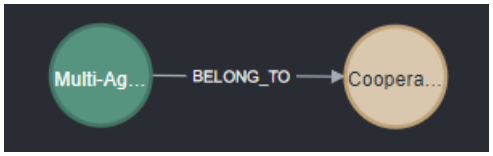
PUBLISHED_BY: link between a Publication (except Inproceeding) and its Publisher.

PUBLISHED_IN: link between an Article and a Journal.

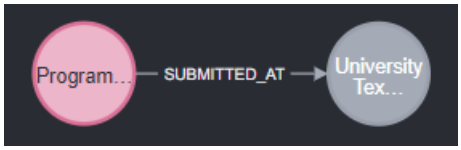
Attributes: volume – number – pages

PART_OF: link between a Publication (Book/Proceedings) and a Series.
Attributes: volume

BELONG_TO: link between a Inproceeding and a Proceeding.
Attributes: volume



SUBMITTED_AT: link between a Book, Proceeding or Phd_Thesis and a School.



2.4. Commands

From specification, we have to deliver **minimum 5 different data creation/update commands**. We decided to include 5 commands that could be exploited in a likely scenario in a web-app for managing a scientific bibliography.

1. Loading authors.

```
LOAD CSV WITH HEADERS FROM "file:///Dummy_authors.csv" AS row
FIELDTERMINATOR '\u003B'
MERGE (author:Author {id:row.ID})
ON CREATE SET author.name = row.name, author.affiliation =
row.affiliation, author.homepage = row.homepage,
author.s2url = row.s2url, author.ORCID = row.ORCID
```

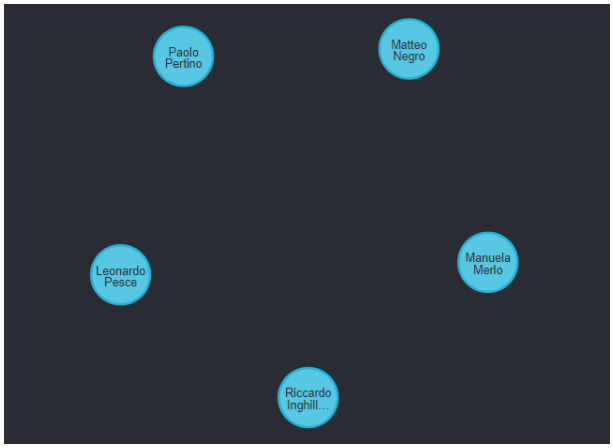


Figure 2.5: Result of Authors loading process

2. Loading papers.

```
LOAD CSV WITH HEADERS FROM "file:///Dummy_papers.csv" AS row
FIELDTERMINATOR '\u003B'
MERGE (paper:Publication {id:row.ID})
ON CREATE SET paper.cdate = row.cdate, paper.ee = row.ee,
paper.key = row.key, paper.mdate = row.mdate,
paper.publtype = row.publtype, paper.title = row.title,
paper.url = row.url, paper.year = row.year
```

3. Creation of the relationship between authors and papers.

```
LOAD CSV WITH HEADERS FROM "file:///Dummy_AUTHORED_BY.csv" AS row
FIELDTERMINATOR '\u003B'
MATCH (author:Author {id: row.AUTHOR_ID})
MATCH (paper:Publication {id: row.PAPER_ID})
MERGE (paper)-[:AUTHORED_BY]->(author)
```

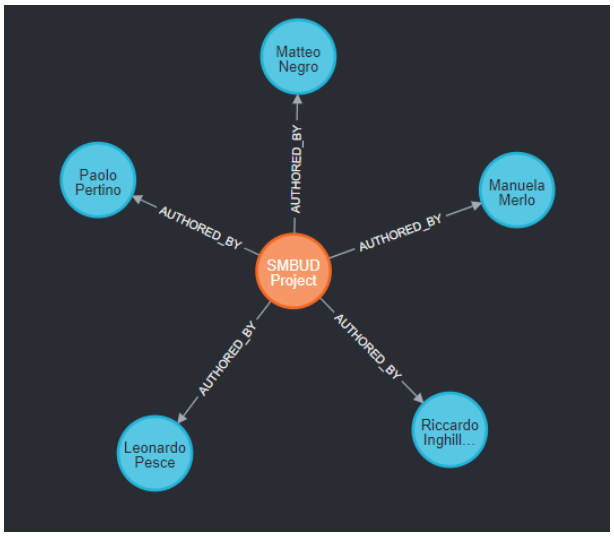


Figure 2.6: Creation of the relationship AUTHORED_BY

4. Creation query to set the citationcount value of each author.

```
MATCH (aut:Author)<-[:AUTHORED_BY]-(paper)<-[:CITES]-(c)
with COUNT(c) AS num, aut
SET aut.citationcount = num
```

5. Creation query to set the paper count value of each author.

```
MATCH (aut:Author)-[:AUTHORED_BY]-(paper)
with aut, COUNT(paper) AS num
SET aut.papercount = num
```

6. During the implementation of the database, we decided to delete nodes without relationships or whose relationships were lost. Creation query that deletes all orphan nodes.

```
MATCH (n)
WHERE NOT (n)--()
DETACH DELETE n
```

2.5. Queries

1. Find the 5 authors with whom Marco Brambilla collaborated the most, returning their number of publications and total citations. **(TYPE 3)**

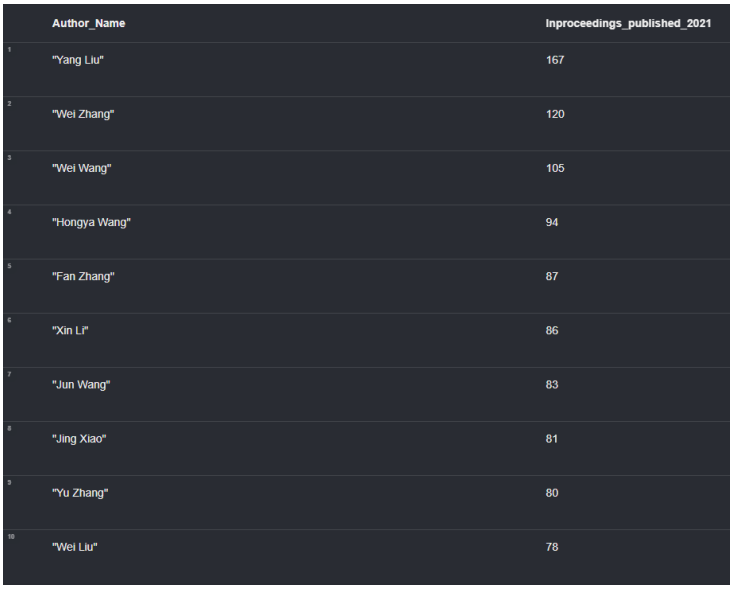
```
MATCH (author:Author)<-[r:AUTHORED_BY]-(pub:Publication)-
[:AUTHORED_BY]->(author2:Author)<-[:AUTHORED_BY]-(pub2:Publication)<-
[:CITES]-(citation:Publication)
WHERE author.name = "Marco Brambilla" AND author.name <> author2.name
WITH count(DISTINCT pub) AS p, author2, COUNT(DISTINCT citation)
AS cit, author
MATCH (pub_second_auth:Publication)-[:AUTHORED_BY]->(author2)
WITH *
ORDER BY p DESC
RETURN author.name AS First_Author_Name , author2.name
AS Second_Author_Name, p AS Joint_Publications,
COUNT(DISTINCT pub_second_auth) AS Publications_Second_Author, cit
AS Number_Of_Citations_Second_Author LIMIT 5
```

	First_Author_Name	Second_Author_Name	Joint_Publications	Publications_Second_Author	Number_Of_Citations_Second_Author
1	"Marco Brambilla"	"S. Ceri"	72	363	567
2	"Marco Brambilla"	"P. Fraternali"	49	206	57
3	"Marco Brambilla"	"Sara Comai"	18	117	8
4	"Marco Brambilla"	"Ioana Manolescu"	8	178	6
5	"Marco Brambilla"	"Fabio Casati"	2	284	13

Figure 2.7: Result Query 1

2. Find the 10 authors who wrote the most inproceedings published in proceedings of 2021. **(TYPE 2)**

```
MATCH (a:Author)-[:AUTHORED_BY]-(ip:Publication:Inproceeding)-
[:BELONG_TO]-(p:Publication:Proceeding)
WHERE p.year = 2021
WITH count(ip) AS publicationCount, a
RETURN a.name AS Author_Name, publicationCount
AS Inproceedings_published_2021
ORDER BY publicationCount DESC
LIMIT 10
```



	Author_Name	Inproceedings_published_2021
1	"Yang Liu"	167
2	"Wei Zhang"	120
3	"Wei Wang"	105
4	"Hongya Wang"	94
5	"Fan Zhang"	87
6	"Xin Li"	86
7	"Jun Wang"	83
8	"Jing Xiao"	81
9	"Yu Zhang"	80
10	"Wei Liu"	78

Figure 2.8: Result Query 2

3. g-index (the maximum reachable value of the h-index if a fixed number of citations can be distributed freely over a fixed number of papers). **(TYPE 2)**

```
MATCH (:Author {name: "F. Olken"})-[:AUTHORED_BY]-(p)
CALL{
    WITH p
    MATCH ()-[:CITES]->(p)
    RETURN COUNT(*) AS citation
}
WITH citation ORDER BY citation DESC
WITH COUNT(*) as c, collect(citation) AS citations
UNWIND RANGE(c, 0, -1) AS i
WITH i, REDUCE(tot = 0, number in citations[0..i] | tot + number)
AS sumFirstN
WHERE sumFirstN >= i*i
```

```
RETURN max(sumFirstN) AS g_index
```

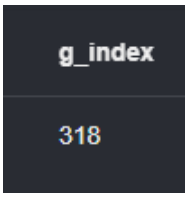


Figure 2.9: Result Query 3

4. i-10-index (the number of publications with at least 10 citations). **(TYPE 2)**

```
MATCH (:Author {name: "F. Olken"})-[:AUTHORED_BY]-(p)
CALL{
    WITH p
    MATCH ()-[:CITES]->(p)
    RETURN COUNT(*) AS citation
}
WITH collect(citation) AS citations
RETURN size([x in citations WHERE x >= 10]) AS i_10_index
```

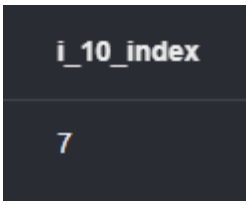


Figure 2.10: Result Query 4

5. h-index (author-level metric that measures both the productivity and citation impact of the publications). **(TYPE 2)**

```
MATCH (:Author {name: "F. Olken"})-[:AUTHORED_BY]-(p)
CALL{
    WITH p
    MATCH ()-[:CITES]->(p)
    RETURN COUNT(*) AS citation
}
WITH count(*) as c, collect(citation) AS citations
UNWIND RANGE(c, 0, -1) AS i
WITH i. citations
WHERE size([x in citations WHERE x >= i]) >= i
```

```
RETURN max(i) AS h_index
```

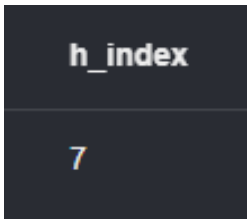


Figure 2.11: Result Query 5

6. Find the publisher who has published most of the works authored by Stefano Ceri. **(TYPE 2)**

```
MATCH (a:Author)<-[r:AUTHORED_BY]-(p:Publication)-[p1:PUBLISHED_BY]->
(pub:Publisher)
WHERE a.name = "S. Ceri"
WITH a, pub, COUNT(*) AS publications
RETURN a.name AS Author, pub.publisher AS Publisher, publications
AS Number_Of_Published_Documents ORDER BY publications DESC
LIMIT 1
```

Author	Publisher	Number_Of_Published_Documents
"S. Ceri"	"Springer"	2

Figure 2.12: Result Query 6

7. Find polytechnic of milan Phd students who published their thesis during the Covid period. **(TYPE 1)**

```
MATCH (a:Author)<-[:AUTHORED_BY]-(phd:PhdThesis)-[:SUBMITTED_AT]->
(school:School {school: "Polytechnic University of Milan, Italy"})
UNWIND phd.year AS year
WITH *
WHERE year >= 2019 AND year < 2022
RETURN phd.title AS Thesis_Title, a.name AS Author_Name, year AS Year
```

	Thesis_Title	Author_Name	Year
1	"Advances in wave digital modeling of linear and nonlinear systems."	"A. Bernardini"	2019
2	"Perception as behaviour inducing mechanism: a reinforcement learning perspective."	"Mirza Ramicic"	2019
3	"Visible light communications for next generation in-flight systems."	"Dario Tagliaferri"	2019
4	"Respiratory motion modeling in particle therapy."	*null*	2019
5	"A randomized model structure selector for complex dynamical systems."	"Federico T Bianchi"	2020
6	"Beyond the traditional analyses and resource management in real-time systems."	"F. Reghenzani"	2021
7	"Machine learning-driven integration, knowledge extraction and uncertainty management for scientific data."	"Gabriele Scalia"	2021
8	"Network sensing and context awareness."	"Marouan Mizmizi"	2019
9	"Exploration and mapping of deep neural networks to low-power hardware accelerators and FPGAs."	"A. Erdem"	2020
10	"Machine learning algorithms for the optimization of internet advertising campaigns."	"Alessandro Nuara"	2021
11	"Patient-cooperative mechatronic solutions for rehabilitative and assistive upper-limb exoskeletons."	"S. D. Gasperina"	2021
12	"Time-resolved multichannel optoelectronic instrumentation based on pulsed lasers and single-photon detectors."	"M. Renna"	2020

Figure 2.13: Fragment of the result of Query 7

8. Find authors who have submitted a Phd dissertation and a book to the same school, sorted by h-index. **(TYPE 3)**

```
MATCH (school:School)<-[:SUBMITTED_AT]-(phdThesis:PhdThesis)-[:
AUTHORED_BY]->(aut:Author)<-[:AUTHORED_BY]-(book:Book)-[:SUBMITTED_AT
]->(school2:School)
WHERE school = school2
RETURN aut, school, phdThesis, book
ORDER BY aut.hindex DESC
LIMIT 10
```

Figure 2.14: Result Query 8

9. Find authors which never worked with S. Ceri but that worked with his colleagues and returns the one which could present to him the best possible unknown coauthor. **(TYPE 3)**

```
MATCH (Ceri:Author {name: "S. Ceri"})
CALL {
    WITH Ceri
    MATCH (Ceri)<-[:AUTHORED_BY]-(n:Publication)-[:AUTHORED_BY]->
    (coAut:Author),(coAut)<-[:AUTHORED_BY]-(n2:Publication)-
    [:AUTHORED_BY]->(cocoAut:Author)
    WHERE NOT (Ceri)<-[:AUTHORED_BY]-(:Publication)-
    [:AUTHORED_BY]->(cocoAut) AND Ceri <> cocoAut
    RETURN cocoAut AS Possible_CoAuthor, COUNT(*) AS Strength
    ORDER BY Strength DESC LIMIT 1
}
WITH *
MATCH (Ceri)<-[:AUTHORED_BY]-(:Publication)-[AUTHORED_BY]->
(coAut1:Author)<-[:AUTHORED_BY]-(:Publication)-[:AUTHORED_BY]->
(Possible_CoAuthor)
RETURN DISTINCT Ceri.name AS Author, coAut1.name AS Middleman,
Possible_CoAuthor.name AS Possible_CoAuthor
```

	Author	Middleman	Possible_CoAuthor
1	"S. Ceri"	"S. Paraboschi"	"S. Vimercati"
2	"S. Ceri"	"Giuseppe Psaila"	"S. Vimercati"
3	"S. Ceri"	"E. Damiani"	"S. Vimercati"
4	"S. Ceri"	"P. Samarati"	"S. Vimercati"

Figure 2.15: Result Query 9

10. Shortest path between two authors. **(TYPE 4)**

```
MATCH (MarcoBrambilla:Author {name:"Marco Brambilla"}),
(OtherAuthor:Author {name:"V. Liberali"}), p = shortestPath(
(MarcoBrambilla)-[*]-(OtherAuthor))
RETURN p
```

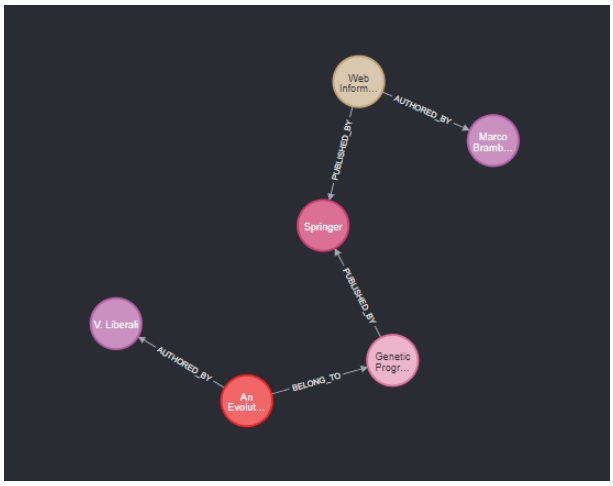


Figure 2.16: Result Query 10

2.5.1. An overview on the performances

By performing the following command

```
PROFILE [query for which we want to know the performance]
```

we can understand how the queries work behind the scene, and we can retrieve some information about the execution, such as the execution plan, the number of rows each operator must deal with, the number of dbhits and the execution time in milliseconds. DBHits are an indicative loose measurement of how much effort the database has to do to get the results of a query, so we used that parameter to have an idea on how much a query is complex. By looking at the queries we have done, we can see we often refer to the name of an Author. Given that, we tried to create an index predicating over the names of the Authors:

```
CREATE INDEX authors_name_index IF NOT EXISTS FOR (a:Author) ON (a.name)
```

This resulted in an improvement of the metric specified before for the queries concerning the authors' names. The results we obtained are reported in the following table:

Query	DBHits Without Index	DBHits With Index	Avg execution Time Without Index (ms)	Avg execution Time With Index (ms)
1	43775342	830089	7000	250
3	9135904	694	1250	10
4	9135904	694	1270	8
5	9135904	694	1235	10
6	1245198	3994	170	9
9	142182864	133019225	67600	64000
10	18270503	86	2790	320

N.B.: in order to calculate the average, we skipped the very first execution of the query, since it may be slower than the subsequent (for this experiment, 5) runs, as the query will have to be planned, and the part of the graph touched may not be in the pagecache.

3 | MongoDB

3.1. Introduction

For the second delivery, the application domain has changed: now the goal is not to focus on managing a library by highlighting the relationships between publications, authors etc as much as designing a documental database that manages as efficiently as possible the storage of papers in all their parts.

In order to give a more in-depth view over the application domain, we decided to present an updated version of our ER schema which better focuses on the new aspects.

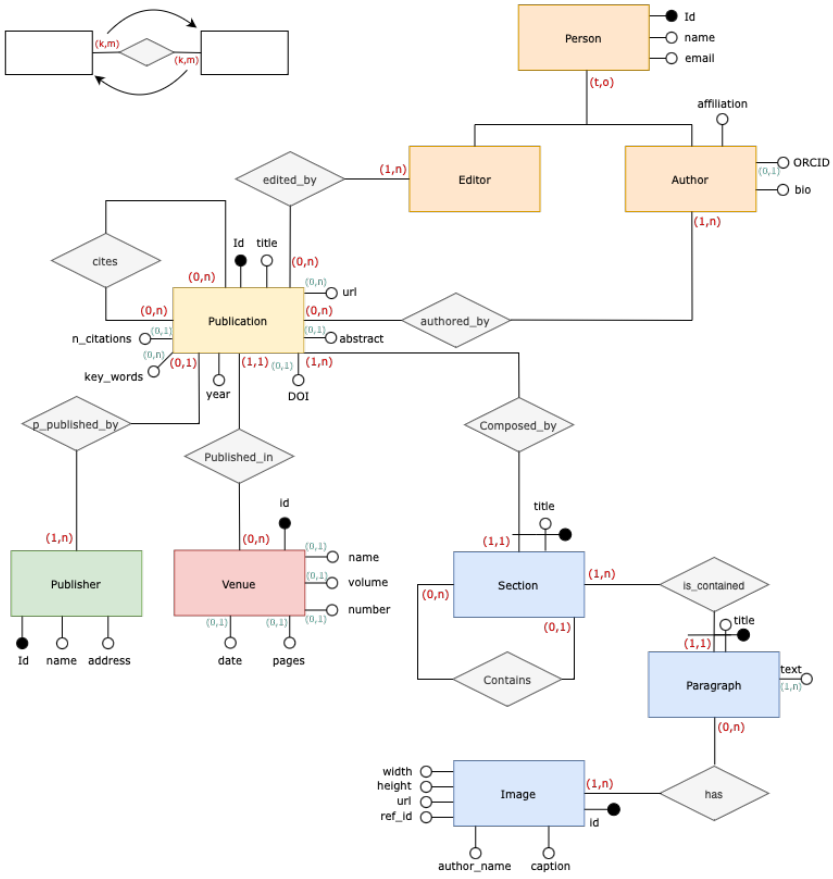


Figure 3.1: *MongoDB ER schema*

3.2. Dataset

Since the application domain has been limited to articles and their intrinsic structure, we had to perform some adjustments to the dataset we used for the previous delivery.

We tried our best to keep the documents and their information set as coherent as possible during the pre-processing phase, by randomly generating as few properties as we could. This has been done according to the available data we found online, the possibility to integrate them in a proper way and the minimum requests of the assignment.

3.2.1. Description and our choice

Specifically, this time, we have dealt with a small subsection of all the possible articles stored in DBLP (5000 in total).

We picked them from the *LREC'22 paper "D3: A Massive Dataset of Scholarly Metadata for Analyzing the State of Computer Science Research"* [1][2][3] (more info about the metadata retrieved can be found here). Articles and their metadata are stored in *JSON-Line* format. Since we had to provide a more structured document, we had to integrate some other information both about the authors and the content of the article, i.e. its sections and paragraphs with their respective text fields and images.

All of these operations are handled by our python script, which sequentially performs the following actions:

1. **JSONL file reduction**
   Reduces the articles dataset by keeping only the first 5000 entries which effectively own a DBLP identifier.

2. **Authors information integration**
   For each article, additional information about all its authors, such as the hindex, affiliation, homepage and others, are integrated into the same document in addition to the already present author_id and name. These ones are filtered out of another dataset coming from the same source we cited before.

3. **Text integration**
   Since we couldn't find in time the texts of our articles (we requested access to this source but we received a response too late to perform all the filtering procedures for the deadline) we decided to use medical articles texts of the *COVID-19 Open Research Dataset (CORD-19)* [4] to fill out the content of our documents.

Given a set of 5000 JSON documents containing the texts of 5000 different publications, the script merges them together and one by one are added to our set of DBLP's papers. In this way even if the publication content is not fully coherent, we still have a proper document structure.

4. **Sections refactoring**

Initially the text is stored as follow:

```
        "body_text": [
        {
            "text": "Some text...",
            "section": "Background"
        },
        {
            "text": "Some other text...",
            "section": "Background"
        },
        {
            "text": "Way too much text...",
            "section": "MongoDB"
        },
        ]

```

The script refactor this structure in the following way:

```
        "sections": [
        {
            "section_title": "Background",
            "paragraphs": [
                {
                    "text" : "Some text...",
                },
                {
                    "text" : "Some other text...",
                },
            ]
        },
        {
```

```
            "section_title": "MongoDB",
            "paragraphs": [
                {
                    "text" : "Way too much text...",
                },
            ]
        },
        ]

```

5. **Integrate images in paragraphs**
   Since the figures in the initial dataset we fetched online had only few properties describing them, then we decided to generate these objects randomly by exploiting API calls to the PicSum service. A set of 100 images is retrieved and then assigned randomly to already present images' instances of the articles. We then also redistributed them inside the paragraphs' objects to align our structure to the one requested in the delivery assignment.

6. **Subsections generation**
   Given the section structure we exposed before, we were unable to find an element to further aggregate our text into subsections. It is for this reason that the script generates them recursively. Each section starts with the 50% of probability of having subsections. If it is the case, then an array of section-like objects is created. Each subsection, as we said before is recursively created, and so it has a probability to have some subsections itself. At each recursive call the probability to generate a deeper level of nesting is halved. In this way we are guaranteed to make the worst-case generation process end quite quickly.

7. **Citations generation**
   The initial data we had about an article did not contain any reference to the cited ones. Therefore, we generated links between the papers contained in our subset by applying these rules:

   - an article cannot cite itself;
   - an article A can cite an article B only once

   Furthermore, we kept the properties *referencecount* (number of incoming citations) and *citationcount* (number of outgoing citations) consistent with respect to the generated items.

8. **Date format fix**
   The dates were initially stored as string. We wanted to import them as ISODate Objects into MongoDB. So, we passed from this structure:

```
        ...
        "publicationdate" : "2011-10-11"
        ...

```

   to the following one:

```
        ...
        "publicationdate" : {
            "$date" : "2011-10-11"
        }
        ...

```

   This last structure is internally interpreted by MongoDB and stored as an ISODate formatted string.

The result of this process is a JSON file called *papers_full_info.json*, which is ready to be directly imported in MongoDB.

3.3. Data Upload

3.3.1. Pre-processing operations

Requirements:

1. Python 3.*
   You can check if you have already installed python by opening the terminal ›_ and using the following command:

   ```
   python --version
   ```

   If you don't already have it, you can install it using the following linked guide .

2. Go to our github repository and copy into your workspace the **requirements.txt** file.
   The following command will allow you to install all the needed libraries:

   ```
   pip3 install .r requirements.txt
   ```

Steps:

Before importing the database, you need to perform the following instructions:

1. Follow *this link* and download the **Semantic Scholar Authors dataset** csv file.
2. Follow *this link* and download the **Semantic Scholar Papers dataset** jsonl file.
3. Follow *this link* and download the **S2ORC Papers with text dataset**.
4. After downloading the package, unzip all the subcompressed archives until you have a folder containing 5000 json files.
5. Place those files in a new directory called 'json_docs' in your workspace folder (i.e. where you will placed the script and other files);
6. Go to our github repository and copy into the folder the **cucciolodifoca.py** file.
   Run it with the command:

   ```
   python cucciolodifoca.py
   ```

7. You will be asked to choose what operation do you want to perform:
   - Choose **2** for the default MongoDB Setup;
   - Choose **3** for the Random MongoDB Setup;
   - Choose **10** to Exit;

   This script will provide the final restructuring and cleaning operations.

Alternatively you can download a pre-compiled dataset from *this link* .
Now you are finally ready to import your dataset into MongoDB.

3.3.2. Upload process

Now that the files have been cleaned up, we can import them into MongoDB.

1. First of all make sure you have installed MongoDB Compass.
   In case you don't already have it, download it from *this page*
2. Open MongoDB Compass.
3. Connect to the MongoDB server.

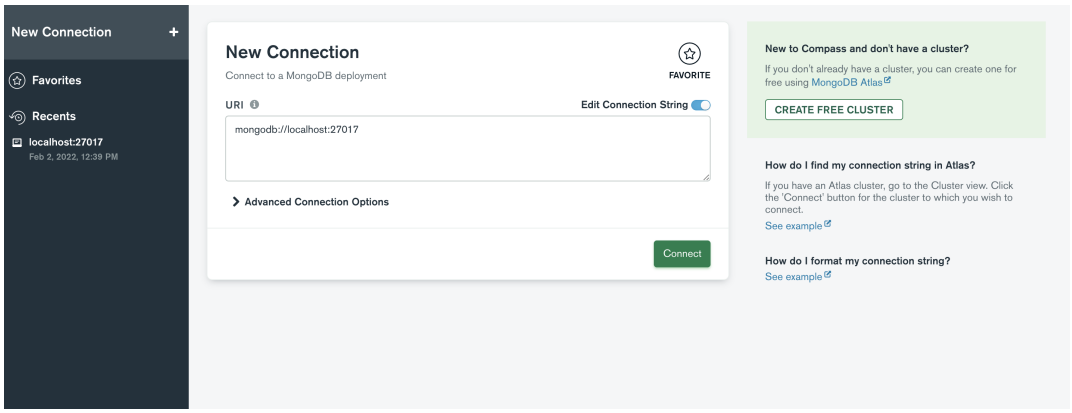


Figure 3.2: *MongoDB Connection*

4. Create a new database by using the bottom-left **✚ CREATE DATABASE**, by specifying as collection name **'publications'**.

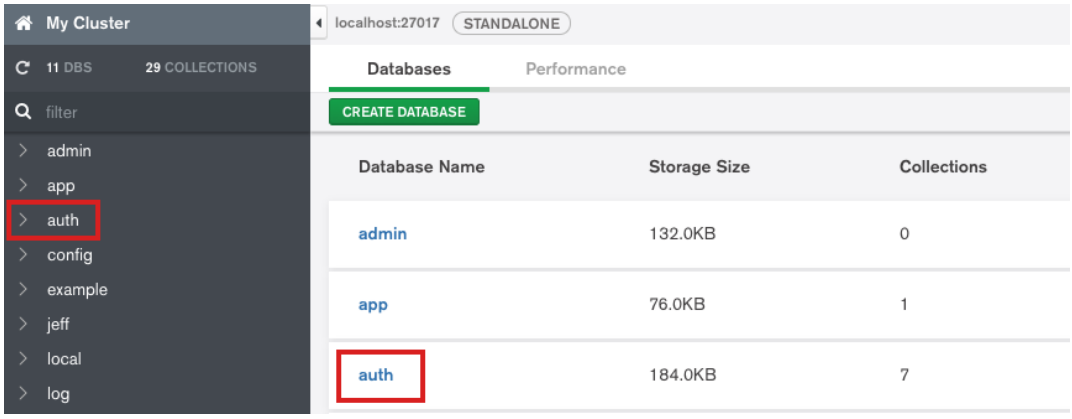


Figure 3.3: *MongoDB Create Database*

5. Open the database and the 'publications' collection.
6. Finally click on **'Import data'** and pick from your PC the **papers_full_info.json** file generated by our script.

You have successfully imported your data into MongoDB.

3.4. Data Model

By using a document-oriented approach it is possible to take advantage of the nesting to describe the relationships. There could be three approaches:

- Create a single Authors collection. Each author would have a property called Publications, which contains an array of documents. Each sub-document is, therefore,

a publication. In this way, once we pick the author, all his publications can also be found.

Pros	Cons
The access is simple if we have to locate an author	To locate a publication we have to select the author and all his publications will necessarily return
Updating the author or a publication can be executed by a single request to the database	

- Create a Publications collection whereby each publication has an authors field that contains an array of documents, each one describing an author. In this way, after choosing a publication, all its authors are also found.

Pros	Cons
The access is simple if we have to locate a single paper	In order to locate all the publications of an author, we have to look for the author in the Publications collection as a grafted property
Updating a publication can be performed with a single query	We would have a repetition of the author information for each publication
Updating an author can be performed with a single query, but this must consider different documents	

- Create a collection that contains the publications and a collection that contains the authors. In this way, both types of reading data can be satisfied without finding unnecessary data.

Pros	Cons
By choosing the starting collection, we can easily obtain the necessary data without duplication of information.	Updating the publication or author, since it involves both collections, must be performed with two different queries (one per collection) to ensure data consistency.

Therefore, the document model might not respect the atomicity and consistency requirements of the transactions: the writing must be performed in two sequential actions, thus violating the atomicity principle. There is also a time, albeit short, in which a collection contains some documents that have not yet been updated.

It is possible to make some data consistency checks, but not others. For example: by using the third approach, it is possible to survey the publications of a non-existent author. Indeed, in the Publications collection, you could insert a document whose author is not registered in the Authors collection.

Since the main goal is to manage a paper in all its parts, we decided to use the second approach.

Structure of a Json-document

```
{
    "corpusid": 0,
    "abstract": "",
    "updated": "",
    "externalids": {
        "DBLP": "",
        "MAG": "",
        "CorpusId": "",
        "DOI": ""
    },
    "url": "",
    "title": "",
    "authors": [
        {
            "authorid": 0,
            "name": "",
            "papercount": 0,
            "citationcount": 0,
            "hindex": 0,
            "s2url": "",
            "externalids.DBLP": ""
        }
    ],
    "venue": "",
    "year": 0,
    "referencecount": 0,
    "citationcount": 0,
    "isopenaccess": true,
    "s2fieldsofstudy": [
```

```
        "",
        ""
    ],
    "publicationtypes": [
        ""
    ],
    "publicationdate": "",
    "journal": {
        "name": "",
        "volume": "",
        "pages": ""
    },
    "content": {
        "sections": [
            {
                "section_title": "",
                "paragraphs": [
                    {
                        "text": "",
                        "figures_in_paragraph": [
                            {
                                "start": 0,
                                "end": 0,
                                "ref_id": "",
                                "caption": "",
                                "text": "",
                                "type": "",
                                "author": "",
                                "width": 0,
                                "height": 0,
                                "url": "",
                                "download_url": ""
                            }
                        ]
                    }
                ],
                "subsections": [
```

```
                {
                    "section_title": "",
                    "paragraphs": [
                        {
                            "text": "",
                            "figures_in_paragraph": [
                                {
                                    "start": 0,
                                    "end": 0,
                                    "ref_id": "",
                                    "caption": "",
                                    "text": "",
                                    "type": "",
                                    "author": "",
                                    "width": 0,
                                    "height": 0,
                                    "url": "",
                                    "download_url": ""
                                }
                            ]
                        }
                    ]
                }
            ]
        }
    ],
    "list_of_figures": [
        {
            "type": "",
            "author": "",
            "url": "",
            "fig_id": ""
        },
        {
            "type": "",
            "author": "",
            "url": "",
```

```
                "fig_id": ""
            }
        ]
        "keywords": []
    },
    "citations": [
        {
            "reference_id": 0,
            "title": "",
            "authors": [
                "",
                ""
            ]
        }
    ]
}
```

Follows a description of the attributes that have not been explained previously or whose name/meaning have changed:

Attribute	Type	Description
corpusid	Int	The unique identifier of the paper on Semantic Scholar
updated	ISODate	Date of the last change made to the paper
externalids	Array[String]	Array of unique identifiers of the publication in different locations
authors	Array[Document]	Array of authors who participated in writing the publication
authors.externalids .DBLP	String	Unique author identifier on DBLP
venue	String	Name of the venue where the paper was published
year	Int	Year of publication of the paper
referencecount	Int	Number of publications citing this publication
citationcount	Int	Number of publications cited by this publication
isopenaccess	Bool	Specifies whether the publication mode allows free and unrestricted access
s2fieldsofstudy	Array[String]	The fields of study of the paper according to SemanticScholar
publicationtypes	Array[String]	Array of categories to which the publication belongs (i.e. Journal Article, Conference, Review, Book ..)
publicationdate	ISODate	Date of publication

journal	Document	Document whose fields describe the name, volume, and page range of the journal in which the publication was placed
content	Document	Document containing the sections, list of images and keywords of the publication
content.sections	Array[Document]	Array containing documents describing the section title, paragraphs within it with related text and images, and its subsections. Each subsection is a document of the same type as the Section
list_of_figures.	Array[Document]	Array of documents describing the images
list_of_figures.type	String	Image type (Image,Table)
keywords	Array[String]	list of keywords of the article
citations	Array[Document]	Array of documents containing the id, title and authors of papers cited by the publication

Below is a table describing the attributes of the documents contained in the list_of_figures array:

Attribute	**Type**	**Description**
start	Int	The index in the text where the references starts
end	Int	The index in the text where the references ends
ref_id	String	The unique identifier of the image
caption	String	Brief explanation of the image
text	String	Description of the image
type	String	Type of the image (Image-Table)
author	String	Author of the image
width	Int	Width of the image in pixels
height	Int	Height of the image in pixels
url	String	Address of the image on the Web
download_url	String	Address where it is possible to download the image from the Web

Below is a list of references to sections where the remaining attributes were described:

- abstract, url, title: 1.3.1
- authors. citationcount, papercount, hindex, s2url : 2.3.1
- journal. volume, pages : 1.3.2

3.5. Commands

From specification, we have to deliver minimum 5 different data creation/update commands. We decided to include 5 commands which maps with the article data model explained before and which can be used to create/update/delete items from the database. **NB**: Before running the commands, switch to the correct database:

1. Open the MongoDB Compass shell.
2. Run the command :

```
use name_of_the_database
```

(we used publications as name_of_the_database)

Commands

1. Insert a document:

```
db.publications.insertOne({
    "corpusid": 5001,
    "abstract": "SMBUD proj",
    "updated": ISODate(),
    "externalids": {"CorpusId": "5001"},
    "url": "https://github.com/leonardopesce/Big-Data-Project-2022-
    Inghilleri-Merlo-Negro-Pertino-Pesce",
    "title": "Systems and Methods for Big and Unstructured Data",
    "authors": [{
        "authorid": 999999999999,
        "name": "SMBUD Group 2",
        "papercount": 1,
        "citationcount": 0,
        "email": "smbud@gmail.com"
        }],
    "venue": "SMBUD Exam",
    "year": 2022,
    "referencecount": 0,
    "citationcount": 1,
    "isopenaccess": false,
    "s2fieldsofstudy": ["Computer Science", "Big Data"],
    "publicationtypes": ["JournalArticle"],
    "publicationdate": ISODate("2022-11-24T10:16:42.597Z"),
```

```
"content": {
    "sections": [{
        "section_title": "MongoDB",
        "paragraphs": [
        {"text": "For the second delivery, the application domain has
changed: now the goal is not to focus on managing a library by
highlighting the relationships between publications, authors etc as
much as designing a document database that manages as efficiently as
possible the storage of papers in all their parts."}
        ],
        "subsections": [{
            "section_title": "Introduction",
            "paragraphs": [
                {"text": "In order to give a more in-depth view over
the application domain, we decided to present an updated version of
our ER schema which better focuses on the new aspects.",
                 "figures_in_paragraph": [{
                    "ref_id": "FIGREF0",
                    "caption": "MongoDB ER schema.",
                    "type": "figure",
                    "author": "SMBUD Group 2",
                    "width": 2022,
                    "height": 2022,
                    "url": "https://github.com/leonardopesce/Big-Data-
Project-2022-Inghilleri-Merlo-Negro-Pertino-Pesce",
                    "download_url": "https://github.com/leonardopesce/
Big-Data-Project-2022-Inghilleri-Merlo-Negro-Pertino-Pesce"
                }]}]}]
    }],
    "list_of_figures": [{
        "type": "figure",
        "author": "SMBUD Group 2",
        "url": "https://github.com/leonardopesce/Big-Data-Project
-2022-Inghilleri-Merlo-Negro-Pertino-Pesce",
        "fig_id": "FIGREF0"
    }],
    "keywords": ["Big", "Data", "DataSet"]
},
"citations": [{
```

```
        "reference_id": 21667,
        "title": "Performance analysis and optimization of switched
    transmit diversity in MIMO systems",
        "authors": [
            "Seyeong Choi",
            "Young-Chai Ko",
            "E. Powers"
        ]
    },
    {
        "reference_id": 16,
        "title": "The U-City Paradigm: Opportunities and Risks for E-
    Democracy in Collaborative Planning",
        "authors": [
            "F. Rotondo"
        ]
    }]
    }
)
```

```
{ acknowledged: true,
  insertedId: ObjectId("637bde96dda22dfce1c811e0") }
```

Figure 3.4: Result Command 1

2. Update *referencecount* for all the publications cited in the new inserted document:

```
db.publications.updateMany(
  { "corpusid": {$in: [21667, 16]}},
  { $inc: {"referencecount" : 1}}
)
```

```
{ acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0 }
```

Figure 3.5: Result Command 2

3. Text index creation:

```
db.publications.createIndex(
   {
     "title": "text",
     "abstract": "text",
     "content.sections.paragraphs.text": "text",
     "content.sections.paragraphs.figures_in_paragraph.text": "text",
     "content.sections.subsections.paragraphs.figures_in_paragraph.text":
    "text",
     "content.sections.subsections.paragraphs.text": "text",
     "content.sections.paragraphs.figures_in_paragraph.caption": "text",
     "content.sections.subsections.paragraphs.figures_in_paragraph.caption
    ": "text",
     "s2fieldsofstudy": "text",
     "content.keywords": "text"
   }
 )
```

N.B. : we could also use a *Wildcard Text Index*, which indexes every field that contains string data.

```
db.publications.createIndex( { "$**": "text" } )
```

Since we wanted to make things clear and our data are quite uniformly structured, we decided instead to explicitly describe those fields we wanted to be indexed.

```
> db.publications.createIndex(
    {
      "title": "text",
      "abstract": "text",
      "content.sections.paragraphs.text": "text",
      "content.sections.paragraphs.figures_in_paragraph.text": "text",
      "content.sections.subsections.paragraphs.figures_in_paragraph.text": "text",
      "content.sections.subsections.paragraphs.text": "text",
      "content.sections.paragraphs.figures_in_paragraph.caption": "text",
      "content.sections.subsections.paragraphs.figures_in_paragraph.caption": "text",
      "s2fieldsofstudy": "text",
      "content.keywords": "text"
    }
  )
< 'title_text_abstract_text_content.sections.paragraphs.text_text_content.sections.paragraphs.figur
```

Figure 3.6: Result Command 3

4. Delete old files:

```
db.publications.deleteMany(
    {$expr: {
        $lt: [
        "$publicationdate",
            { $dateSubtract: { startDate: "$$NOW", unit: "year", amount:
    50 }}
        ]
    }}
)
```

```
< { acknowledged: true, deletedCount: 475 }
```

Figure 3.7: Result Command 4

5. Update *updated* field:

```
db.publications.updateOne(
    {"corpusid": 5001},
    {"$set":
        {"updated": ISODate(), "abstract": "test"}}
)
```

```
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
```

Figure 3.8: Result Command 5

3.6. Queries

1. Find, for each Field of Study, the most important article which has at least 3 authors and have been published in the first decade of the 21^{st} century (calculated with respect to its ingoing citations) : **(TYPE 4)**

```
db.publications.aggregate([
    { $project: {"s2fieldsofstudy":1, "title":1, "referencecount":1, "
    authors":1, "corpusid":1, "year":1}},
    { $match: {"year": {$gte: 2000, $lte:2010}, "$expr": {$gte:[{$size:"
    $authors"}, 3]}}},
    { $unwind: "$s2fieldsofstudy" },
    { $group: {
        _id: "$s2fieldsofstudy",
        top_article_metadata: {
            $top: {
                output : {
                    "article_id" : "$corpusid",
                    "article_title" : "$title",
                    "article_authors" : "$authors.name"
                },
                sortBy: {"referencecount" : -1}
            }
        },
        max_ingoing_citations : { "$max" : "$referencecount" }
    },
}])
```

```
{ _id: 'Business',
  top_article_metadata:
   { article_id: 13824,
     article_title: 'mSCTP for soft handover in transport layer',
     article_authors: [ 'S. Koh', 'Moonjeong Chang', 'Meejeong Lee' ] },
  max_ingoing_citations: 69 }
{ _id: 'Geography',
  top_article_metadata:
   { article_id: 16074,
     article_title: 'Active-GNG: model acquisition and tracking in cluttered backgrounds',
     article_authors:
      [ 'Anastassia Angelopoulou',
        'A. Psarrou',
        'J. G. Rodríguez',
        'Gaurav Gupta' ] },
  max_ingoing_citations: 63 }
{ _id: 'Education',
  top_article_metadata:
   { article_id: 751,
     article_title: 'The role of time in considering collections',
     article_authors: [ 'Françoise Gayral', 'D. Kayser', 'F. Lévy' ] },
  max_ingoing_citations: 87 }
{ _id: 'Law',
  top_article_metadata:
   { article_id: 8565,
     article_title: 'A lawyer directory service using legal documents and profile information as support',
     article_authors: [ 'Gloria T. Lau', 'T. Pawłowski', 'R. Pasadas', 'W. Kubalski' ] },
  max_ingoing_citations: 47 }
```


Figure 3.9: Result Query 1

2. Get the publication in which a certain word appears the most in the abstract and has at least 50 incoming citations: **(TYPE 1/4)**

```
db.publications.aggregate([
  { $match: {"referencecount": {$gte:50}}},
  { $project : { abstract: { $split: ["$abstract", " "] }, qty : 1 } },
  { $unwind : "$abstract" },
  { $match: {"abstract": /statistical/}},
  { $group: {_id: "$_id", count:{$sum: 1}}},
  { $sort: {count:-1, _id:1}},
  { $limit: 1},
  { $lookup:{
    from: "publications",
    localField: "_id",
    foreignField: "_id",
    as: "pub"
  }}
])
```

```
{ _id: ObjectId("637bd6758f6ee021c7bfe0cf"),
  count: 4,
  pub:
   [ { _id: ObjectId("637bd6758f6ee021c7bfe0cf"),
       corpusid: 13009,
       abstract: 'In this thesis we explore robust statistical syntax analysis for French. Our main concern
       updated: 2022-09-03T05:25:22.848Z,
       externalids: { DBLP: 'phd/hal/Urieli13', MAG: '647189206', CorpusId: '13009' },
       url: 'https://www.semanticscholar.org/paper/e383d8f7b4c0a815b395fa3b50d537b569300b07',
       title: 'Robust French syntax analysis: reconciling statistical methods and linguistic knowledge in t
       authors:
        [ { authorid: 1905124,
            name: 'A. Urieli',
            papercount: 9,
            citationcount: 137,
            hindex: 4,
            s2url: 'https://www.semanticscholar.org/author/1905124',
            'externalids.DBLP': '[\'Assaf Urieli\']',
            email: 'A.Urieli@gmail.com' } ],
       venue: '',
       year: 2013,
       referencecount: 64,
       citationcount: 81,
       isopenaccess: false,
       s2fieldsofstudy: [ 'Computer Science' ],
       publicationdate: 2013-12-17T00:00:00.000Z,
       journal: { name: '', volume: '' },
       content:
```


Figure 3.10: Result Query 2

3. Get the first 5 publications (ordered by incoming citations count) in the field of Computer Science written by an author with an affiliation and an h-index greater than 50: **(TYPE 3)**

```
db.publications.find({
  "s2fieldsofstudy" : "Computer Science",
  "authors": {"$elemMatch": { "hindex": { "$gt": 50 }, "affiliations" : {
    "$exists" : true } }}
},{
  "authors.name":1, "title":1, "referencecount":1
}).sort({
  "referencecount":-1
}).limit(5)
```

```
{ _id: ObjectId("637bd6648f6ee021c7bfd7ed"),
  title: 'MBT: A Memory-Based Part of Speech Tagger-Generator',
  authors:
   [ { name: 'Walter Daelemans' },
     { name: 'Jakub Zavrel' },
     { name: 'P. Berck' },
     { name: 'S. Gillis' } ],
  referencecount: 77 }
{ _id: ObjectId("637bd6878f6ee021c7bfe7b4"),
  title: 'Adaptive ground plane estimation for moving camera-based 3D object tracking',
  authors:
   [ { name: 'Tao Liu' },
     { name: 'Yong Liu' },
     { name: 'Zheng Tang' },
     { name: 'Jenq-Neng Hwang' } ],
  referencecount: 74 }
{ _id: ObjectId("637bd6758f6ee021c7bfe281"),
  title: 'Active image: A shape and topology preserving segmentation method using B-spline free form deformations',
  authors: [ { name: 'C. Li' }, { name: 'Ying Sun' } ],
  referencecount: 74 }
{ _id: ObjectId("637bd6648f6ee021c7bfd75b"),
  title: 'A dynamic approach to characterizing termination of general logic programs',
  authors:
   [ { name: 'Yi-Dong Shen' },
     { name: 'Jia-Huai You' },
     { name: 'Li-Yan Yuan' },
     { name: 'S. S. Shen' },
     { name: 'Qiang Yang' } ],
  referencecount: 70 }
{ _id: ObjectId("637bd6758f6ee021c7bfe019"),
  title: 'Batched multi triangulation',
  authors:
   [ { name: 'Paolo Cignoni' },
     { name: 'F. Ganovelli' },
     { name: 'E. Gobbetti' },
     { name: 'F. Marton' },
     { name: 'F. Ponchio' },
     { name: 'Roberto Scopigno' } ],
  referencecount: 68 }
```


Figure 3.11: Result Query 3

4. Get the publications' titles of all the articles published between 2004 and 2008 of type JournalArticle grouped by venue and year: **(TYPE 1/2)**

```
db.publications.aggregate([
  { $match: {"year": {$gte:2004, $lte:2008}, "publicationtypes":"
    JournalArticle"}},
  { $group: {
      '_id': { 'venue': '$venue', 'year': '$year'},
      'titles': {'$push':'$title'}
}}])
```

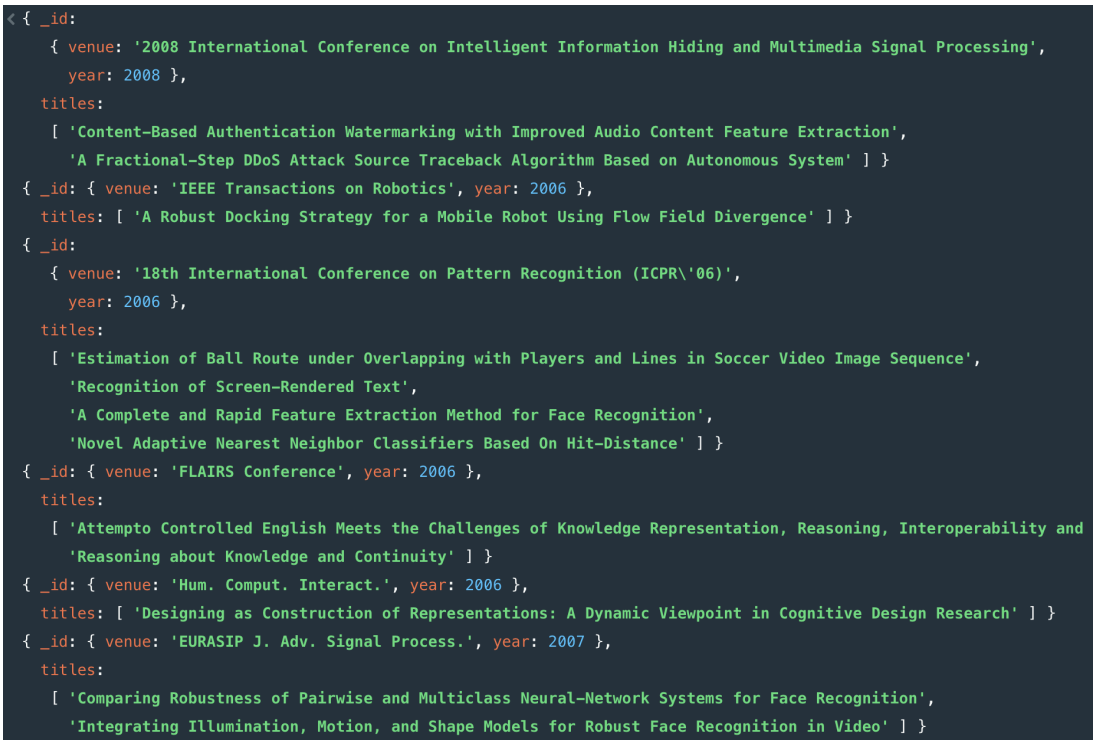


Figure 3.12: Result Query 4

5. Get from an *author's* publication of *type: "Journal Article"* the cited publication which match the most with the first publication tags (how much a publication and its cited publications are 'relevant' on the same topic): **(TYPE 5)**

```
db.publications.aggregate([
        { $match: {'authors.authorid': 2194162, "publicationtypes": "
    JournalArticle"}},
        { $project: {'s2fieldsofstudy': 1, 'citations.reference_id': 1}},
        { $lookup:{
            from: "publications",
            localField: "citations.reference_id",
            foreignField: "corpusid",
            as: "citations"
        }},
        { $project: {'s2fieldsofstudy': 1, 'citations.corpusid':1, '
    citations.s2fieldsofstudy': 1, "citations.title": 1}},
        { $unwind: "$citations"},
        { $project: {"citationsid":"$citations.corpusid", 's2fieldsofstudy
    ':1, 'compare': '$citations.s2fieldsofstudy', "title":"$citations.
    title"}},
        { $unwind: "$s2fieldsofstudy"},
```

```
        { $unwind: "$compare" },
        { $group: {_id: {"_id":"$_id", "citationsid":"$citationsid", "
    title":"$title"},
          count: {
            $sum: {
              $cond: [{$eq:["$s2fieldsofstudy","$compare"]}, 1, 0]
            }
          }
        }},
        { $group: {_id: "$_id._id",
          top_article_data: {
            $top: {
              output: {"title":"$_id.title", "citationsid":"$_id.
    citationsid"},
              sortBy:{"count":-1}
            }
          }
        }}
      ])
```

```
{ _id: ObjectId("637bd6648f6ee021c7bfd72d"),
  top_article_data:
   { title: 'Feature extraction and classification of neuromuscular diseases using scanning EMG',
     citationsid: 29118 } }
```

Figure 3.13: Result Query 5

6. Get publications written by the same author and count the tag to understand which is the main topic of that author in the last 20 years: **(TYPE 1/4)**

```
db.publications.aggregate([
    {'$match': {'authors.authorid': 145967056,
        $expr: {
            $gt: [
            "$publicationdate",
                { $dateSubtract: { startDate: "$$NOW", unit: "year",
    amount: 20 } }
            ]
        }
    }},
    {'$project': {'s2fieldsofstudy':1}},
    {'$unwind': '$s2fieldsofstudy'},
```

```
    {'$group': {'_id': '$s2fieldsofstudy', occurrences:{'$sum':1}}},
    {'$sort': {occurrences: -1}},
    {'$limit': 1}
])
```

```
{ _id: 'Computer Science', occurrences: 21 }
```

Figure 3.14: Result Query 6

7. Get all the titles of publications which have been discussed in a conference and have been published by the same venue, during the same year (starting from 2015). **(TYPE 1/2)**

```
db.publications.aggregate([
  { "$match" : { "publicationtypes" : "Conference" }},
    {"$group": {
        "_id": { "venue": "$venue", "year": "$year"},
        "titles": {"$push":"$title"}
    }
    },
  { "$match" : {"_id.year" : { "$gte" : 2015 }} }
])
```

```
{ _id:
   { venue: '2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing an
     year: 2015 },
  titles:
   [ 'Walk and Jog Characterization Using a Triaxial Accelerometer',
     'A Novel QoS Routing Method for LEO Rosette Constellation Network' ] }
{ _id: { venue: 'NIPS', year: 2016 },
  titles:
   [ 'Threshold Bandits, With and Without Censored Feedback',
     'Interaction Screening: Efficient and Sample-Optimal Learning of Ising Models',
     'Coresets for Scalable Bayesian Logistic Regression' ] }
{ _id: { venue: 'SIGIR', year: 2016 },
  titles: [ 'Linking Organizational Social Network Profiles' ] }
{ _id:
   { venue: '2017 IEEE International Conference on Computer Vision (ICCV)',
     year: 2017 },
  titles: [ 'Learning to Reason: End-to-End Module Networks for Visual Question Answering' ] }
{ _id: { venue: 'ICML', year: 2016 },
  titles: [ 'Learning privately from multiparty data' ] }
{ _id: { venue: 'WWW', year: 2017 },
  titles: [ 'Identifying Value in Crowdsourced Wireless Signal Measurements' ] }
{ _id:
   { venue: '2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)',
     year: 2017 },
  titles:
   [ 'Natural calamity assessment by innovative methods',
     'Coarse to fine difference edge detection with binary neural firing model' ] }
```

Figure 3.15: Result Query 7

8. Get all different images that have been posted on usplash.com: **(TYPE 2/3/4)**

```
db.publications.aggregate([
  {"$match" : {"$and" : [{"content.list_of_figures" : { "$exists" : true
   }}, {"content.list_of_figures" : {"$elemMatch": {"type" : "figure", "
   url" : {"$exists" : true}}}}]}},
  {"$project" : {"content.list_of_figures" : 1}},
  {"$unwind" : "$content.list_of_figures"},
  {"$match" : {"content.list_of_figures.url" : {"$regex" : "unsplash.com"
   }}},
  {"$group" : {"_id" : {
                  "url" : "$content.list_of_figures.url",
                  "author" : "$content.list_of_figures.author"}}}
])
```

```
{ _id:
   { url: 'https://unsplash.com/photos/I_9ILwtsl_k',
     author: 'Paul Jarvis' } }
{ _id:
   { url: 'https://unsplash.com/photos/ZJsseAxEcqM',
     author: 'Justin Leibow' } }
{ _id:
   { url: 'https://unsplash.com/photos/uDUiRS8YroY',
     author: 'Christopher Sardegna' } }
{ _id:
   { url: 'https://unsplash.com/photos/aeVA-j1y2BY',
     author: 'Shyamanta Baruah' } }
{ _id:
   { url: 'https://unsplash.com/photos/j9nfqTi5T5o',
     author: 'May Pamintuan' } }
{ _id:
   { url: 'https://unsplash.com/photos/knYQ6arClBE',
     author: 'Ireneuilia' } }
{ _id:
   { url: 'https://unsplash.com/photos/N7XodRrbzS0',
     author: 'Alejandro Escamilla' } }
{ _id:
   { url: 'https://unsplash.com/photos/57vHdjeZ0yg',
     author: 'Cierra' } }
{ _id:
   { url: 'https://unsplash.com/photos/N-1XGL54pQg',
     author: 'Ryan Mcguire' } }
```

Figure 3.16: Result Query 8

9. Check, between those authors which have a quite high citationcount (≥ 1000) or hindex (≥ 50), how much the self citing phenomena is spread. **(TYPE 4)**

```
db.publications.aggregate([
    {"$unwind": "$authors"},
    {"$unwind": "$citations"},
    {"$unwind": "$citations.authors"},
    {"$match" : { "$or" : [ { "authors.hindex" : { "$gte" : 50 } }, { "
    citationcount" : { "$gte" : 1000 } } ] } },
    {"$group": {_id: {"author": "$authors.authorid", "name": "$authors.
    name", "citationcount" : "$authors.citationcount", "hindex" : "
    $authors.hindex"}, count:
    {"$sum": {"$cond": [{"$eq":["$authors.name", "$citations.authors"
    ]},1,0]}}}},
    {"$sort": {"count":-1}}
])
```


```
{ _id:
   { author: 1742047,
     name: 'N. Jindal',
     citationcount: 14811,
     hindex: 50 },
  count: 3 }
{ _id:
   { author: 1744800,
     name: 'V. Paxson',
     citationcount: 45838,
     hindex: 94 },
  count: 2 }
{ _id:
   { author: 1807433,
     name: 'S. Stolfo',
     citationcount: 24020,
     hindex: 68 },
  count: 2 }
{ _id:
   { author: 145197953,
     name: 'D. Fox',
     citationcount: 57315,
     hindex: 111 },
  count: 2 }
{ _id:
   { author: 145967056,
     name: 'H. Poor',
     citationcount: 98071,
     hindex: 139 },
  count: 2 }
```


Figure 3.17: Result Query 9

10. Get the publications' title of the same year (1998) from the same venue (ACL) ordered by *referencecount*: **(TYPE 1)**

```
db.publications.find(
    { $and: [{"venue" : "ACL"}, {"year" : 1998}] },
    { "title":1 , "referencecount" : 1, "venue" : 1, "year" : 1}
).sort({ "referencecount" : -1})
```


```
{ _id: ObjectId("637bd6648f6ee021c7bfd927"),
  title: 'Automatically Creating Bilingual Lexicons for Machine Translation from Bilingua
  venue: 'ACL',
  year: 1998,
  referencecount: 66 }
{ _id: ObjectId("637bd6648f6ee021c7bfdac8"),
  title: 'Learning Correlations between Linguistic Indicators and Semantic Constraints: R
  venue: 'ACL',
  year: 1998,
  referencecount: 54 }
{ _id: ObjectId("637bd6648f6ee021c7bfd9b1"),
  title: 'Integrating Text Plans for Conciseness and Coherence',
  venue: 'ACL',
  year: 1998,
  referencecount: 53 }
{ _id: ObjectId("637bd66e8f6ee021c7bfde1b"),
  title: 'Efficient Linear Logic Meaning Assembly',
  venue: 'ACL',
  year: 1998,
  referencecount: 51 }
{ _id: ObjectId("637bd6648f6ee021c7bfda9f"),
  title: 'An Empirical Evaluation of Probabilistic Lexicalized Tree Insertion Grammars',
  venue: 'ACL',
  year: 1998,
  referencecount: 50 }
{ _id: ObjectId("637bd6648f6ee021c7bfd79f"),
  title: 'Optimal Multi-Paragraph Text Segmentation by Dynamic Programming',
  venue: 'ACL',
  year: 1998,
  referencecount: 48 }
{ _id: ObjectId("637bd6648f6ee021c7bfda45"),
  title: 'Error-Driven Pruning of Treebank Grammars for Base Noun Phrase Identification',
  venue: 'ACL',
  year: 1998,
  referencecount: 48 }
```


Figure 3.18: Result Query 10

11. Find all those articles which contains the sentence 'U-City Paradigm' and have been published in the last 15 years (**N.B.** before running this query the index must be created, look in the command section): **(TYPE 1/3)**

```
db.publications.find({
    $text: {$search: "\"U-City Paradigm\"", $language: "en"},
    $expr: {
        $gt: [
        "$publicationdate",
            { $dateSubtract: { startDate: "$$NOW", unit: "year", amount:
    15 } }
        ]
    },
}).sort({ publicationdate: 1 })
```


```
{ _id: ObjectId("637bd6648f6ee021c7bfd72d"),
  corpusid: 16,
  abstract: 'Volunteered Geographic Information (VGI) tools appear to enhance the possibilities of
  updated: 2022-08-28T10:16:42.597Z,
  externalids:
   { DBLP: 'journals/fi/Rotondo12',
     MAG: '2142154367',
     CorpusId: '16',
     DOI: '10.3390/fi4020563' },
  url: 'https://www.semanticscholar.org/paper/f484c290244e8fa9b9e64a19a6385ad05543886d',
  title: 'The U-City Paradigm: Opportunities and Risks for E-Democracy in Collaborative Planning',
  authors:
   [ { authorid: 2194162,
       name: 'F. Rotondo',
       papercount: 44,
       citationcount: 231,
       hindex: 9,
       s2url: 'https://www.semanticscholar.org/author/2194162',
       'externalids.DBLP': '[\'Francesco Rotondo\']',
       email: 'F.Rotondo@gmail.com' } ],
  venue: 'Future Internet',
  year: 2012,
  referencecount: 60,
  citationcount: 20,
  isopenaccess: true,
  s2fieldsofstudy: [ 'Economics', 'Computer Science' ],
  publicationtypes: [ 'JournalArticle' ],
  publicationdate: 2012-06-05T00:00:00.000Z,
  journal: { name: 'Future Internet', volume: '4', pages: '563-574' },
  content:
   { sections:
```


Figure 3.19: Result Query 11

12. Given the title of an article, pick from its cited publications of type *Book* published between 2010 and 2015 which have been cited more than 50 times and return their title and the venue in which they have been published in. **(TYPE 5)**

```
db.publications.aggregate([
    { $match: { "title" : "The load, capacity and availability of quorum
    systems"} },
    { $project: {'citations.reference_id': 1} },
    { $lookup:{
          from: "publications",
          localField: "citations.reference_id",
          foreignField: "corpusid",
          as: "citations"
    }},
    { $unwind: "$citations"},
    { $project: {"publicationtypes": "$citations.publicationtypes", "year"
    : "$citations.year", "referencecount": "$citations.referencecount", "
    venue": "$citations.venue", "journal": "$citations.journal", "title":
    "$citations.title"} },
    { $match: { $and: [{"publicationtypes" : "Book"}, { "year" : { $gte :
    2010 } }, { "year" : { $lte : 2015 } }, { "referencecount" : { $gte :
    50 } }] } },
])
```

```
{ _id: ObjectId("637bd67d8f6ee021c7bfe626"),
  publicationtypes: [ 'Book', 'JournalArticle' ],
  year: 2015,
  referencecount: 53,
  venue: 'SIGUCCS',
  journal: { name: 'Proceedings of the 2015 ACM SIGUCCS Annual Conference' },
  title: 'Re-Inventing the Helpdesk. Again. In Five Weeks or Less' }
{ _id: ObjectId("637bd67d8f6ee021c7bfe626"),
  publicationtypes: [ 'Book', 'JournalArticle' ],
  year: 2011,
  referencecount: 56,
  venue: 'ACM Trans. Graph.',
  journal: { name: 'ACM SIGGRAPH 2011 papers' },
  title: 'High-quality passive facial performance capture using anchor frames' }
{ _id: ObjectId("637bd67d8f6ee021c7bfe626"),
  publicationtypes: [ 'Book', 'JournalArticle' ],
  year: 2010,
  referencecount: 63,
  venue: 'CHI Extended Abstracts',
  journal: { name: 'CHI \'10 Extended Abstracts on Human Factors in Computing Systems' },
  title: 'Eye tracking analysis of preferred reading regions on the screen' }
```

Figure 3.20: Result Query 12

3.6.1. Queries performances

By performing the following command:

```
db.publications.find(<QUERY>).explain("executionStats")
```

or (for aggregations)

```
db.publications.aggregate(<QUERY>).explain("executionStats")
```

a useful document containing information about the query execution plan and performances is returned as output. It is structured as follow:

```
   { explainVersion: '1',
 stages:
  [ { '$cursor':
       { queryPlanner:
          { namespace: 'DBLPDataset.publications',
            indexFilterSet: false,
            parsedQuery: ···,
            ···
            winningPlan:
             { stage: 'PROJECTION_SIMPLE',
               inputStage:
                { stage: 'COLLSCAN',
                  filter: ··· },
            rejectedPlans: [] },
         executionStats:
          { executionSuccess: true,
            nReturned: 1370,
            executionTimeMillis: 111,
            totalKeysExamined: 0,
            totalDocsExamined: 5000,
            executionStages:
             { stage: 'PROJECTION_SIMPLE',
               ···
               <STATS_ON_THE_STAGE>
             },
      ··· ],
 serverInfo: ···,
 serverParameters: ···,
 command: ···,
 ok: 1 }
```

Given this, for a first look analysis, we are mostly interested in some properties of the *executionStats* object. More precisely we'll focus on:

- ***executionTimeMillis***: time, expressed in milliseconds, required by the query to perform its task;
- ***nReturned***: number of documents that the query matches and returns;
- ***totalKeysExamined***: number of index entries scanned by MongoDB (0 if no index is used to perform the query);
- ***totalDocsExamined***: number of documents scanned by MongoDB to find the *nReturned* documents which have been returned.

As the difference between the number of matching documents, identified by *nReturned* and the number of examined documents, identified with *totalDocsExamined*, grows it may be useful to create an index to improve efficiency, since the query might benefit from that. By giving a quick look to the queries we presented, we mostly predicate over the following fields:

- *authors*;
- *year*;
- *s2fieldofstudy*;
- *referencecount*;
- *publicationtypes*;
- *publicationdate*;

Since *authors*, *s2fieldsofstudy* and *publicationtypes* are arrays, then ***Multikey Indexes*** will be created, whereas for the other ones MongoDB will generate ***Single Field Indexes*** (the type of index will not be specified since it will be handled automatically by MongoDB depending on the type of the field assessed by the index, whether it is an array value or a single field).
Since the fields we mentioned could be reasonably used many times also in other queries in a more general scenario, we created indexes covering those properties:

```
db.publications.createIndex( { year: -1 } )
db.publications.createIndex( { referencecount: 1 } )
db.publications.createIndex( { publicationdate: -1 } )
db.publications.createIndex( { authors: 1 } )
db.publications.createIndex( { s2fieldsofstudy: 1 } )
```

```
db.publications.createIndex( { publicationtypes: 1 } )
```

Finally, let's analyze the performances by comparing the queries performed in a Database without indexes and another one in which they have been created:

Without Index

Query	execTime (ms)	totalDocs Examined (tDE)	nReturned (nR)	tDE-nR	totalKey Examined
1	573	5000	1370	3630	0
2	3900	5000	3737	1263	0
3	15	5000	5	4995	0
4	30	5000	864	4136	0
5	21	5000	1	4999	0
6	12	5000	21	4979	0
7	12	5000	228	4772	0
8	300	5000	4076	924	0
9	2700	5000	5000	0	0
10	7	5000	7	4993	0
11	1500	2002	1	2001	2362
12	15	5000	4999	1	0

With Index

Query	execTime (ms)	totalDocs Examined (tDE)	nReturned (nR)	tDE - nR	totalKey Examined
1	40	2545	1370	1175	2545
2	600	3737	3737	0	3737
3	5	199	5	194	199
4	15	1435	864	571	1435
5	22	4959	1	4958	4959
6	20	3610	21	3589	3610
7	12	766	228	538	766
8	290	5000	4076	924	0
9	2700	5000	5000	0	0
10	1	101	7	94	101
11	1500	2091	1	2090	2489
12	15	5000	4999	1	0

Without the indexes, the queries always scan the whole collection containing 5000 documents to return, in more than an half of the cases, only few of them. In order to make the queries scan only the interested fields during the filtering process, we tried to project

every time only the properties needed in later stages of the pipeline. This has been done also to potentially avoid to scan the entirety of each document, potentially pulling them into memory. This would results in expensive and potentially slow queries operations, especially in case of much larger datasets.

[ADDENDUM]: by further analysing our results we can notice that few queries, like the sixth one, have higher execution time while invoked in an indexed environment with respect to an unindexed one. This is due to the fact that we built indexes over single properties of the documents, while performing multiple filtering conditions within the same statement. In fact in query 6 we are filtering both on authors.authorid and publicationdate. By creating a *Compound Index* over both properties by using:

```
db.publications.createIndex( { "authors.authorid": 1, "publicationdate": -1
} )
```

we get the following result:

With compound index - Query 6

Query	execTime (ms)	totalDocs Examined (tDE)	nReturned (nR)	tDE - nR	totalKey Examined
6	2	21	21	0	21

Now the query is as efficient as possible.
This approach leads to very low memory consumption and fast execution time, but we should also take into account, for larger datasets/database instances a tradeoff with disk space consumption due to the creation of multiple indices.

4 | Spark

4.1. Introduction

From the beginning, we were required to analyze and process a dataset consisting of millions of data points. Apache Spark, with its in-memory caching and optimized query execution, allows us to perform fast analytical queries on data of any size. In addition, Apache Spark natively supports Java, Scala, R, and Python, offering a variety of languages including the one we are currently using for data pre-processing. These APIs make our work easier because they hide the complexity of distributed processing behind simple high-level operators that dramatically reduce the amount of code needed.

For simplicity, we reduce the number of entities to only 3, which represent the Authors(a Person) the Articles(a Publication), and the Journals. We represented each entity with a table. Below we illustrate an updated version of the ER diagram (a reduction from the general one).

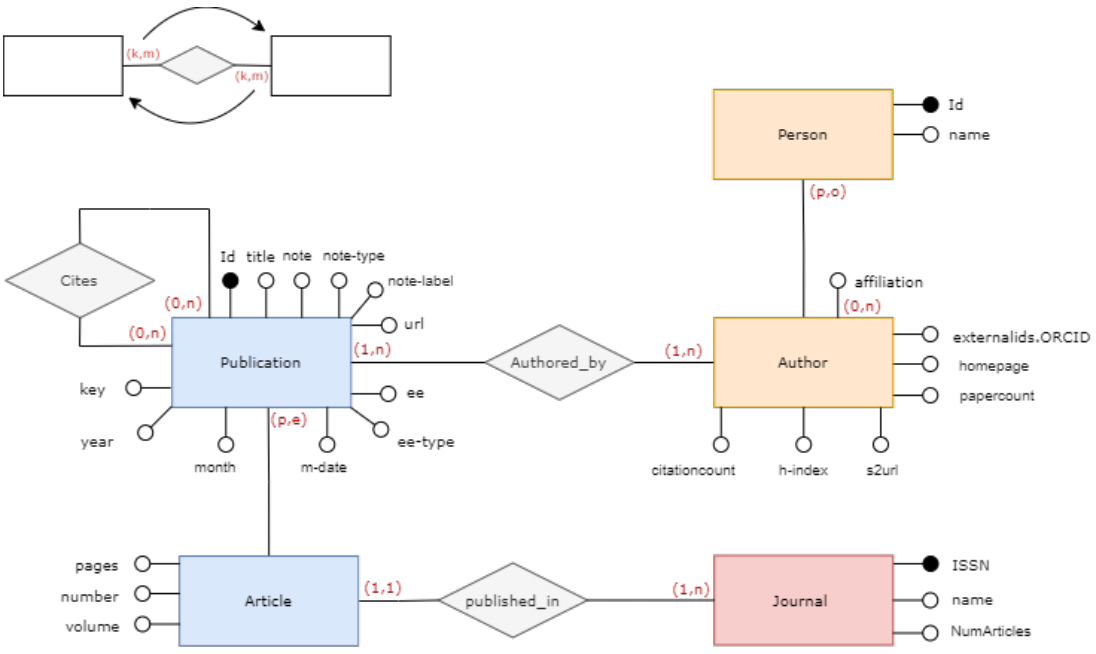


Figure 4.1: *ER schema*

4.2. Dataset

The dataset used in this part is the same one used in the first delivery. We, therefore, began the data processing from the same csv files. According to the updated model presented in the previous section, we picked *Authors* ($\approx 3M$ elements), *Articles* ($\approx 3.1M$ elements) and Journals ($\approx 2K$ elements) datasets.

4.2.1. Pre-processing operations

In order to use the csv files and data from the first delivery, we had to perform the following pre-processing operations:

- **Type checking**
  Since we were reading data from a csv file and putting them, again, in Pandas dataframes, we had to perform some type conversions (e.g.: from string to integer, or from string to Array<String>).
- **References between tables**
  In order to deal with the relationships between the 3 entities we have in our dataset, we performed the following restructuring operations:
  - **Authors dataframe**: for each author entry, we added an array of integers containing all the IDs of the articles he has written;
  - **Articles dataframe**: for each article entry, we added three arrays of integers: one containing all the IDs of its authors, one the IDs of the cited articles and the last containing all the IDs of the incoming citations; furthermore we added an integer with the journal ID in which the article was published;
  - **Journal dataframe**: for each journal entry, we added an array of integers containing all the IDs of the articles published in it.

  We decided to make the *AUTHORED_BY* and *CITES* relationships bidirectional. We made this modelling and implementation choice with the aim of making the way queries can be executed as flexible as possible, by avoiding the use of bridging dataframes (e.g. : select all the articles of an author VS find all the authors of an article).
- **Parquet file creation**
  In order to load the files faster and in a more efficient way we decided to save the elaborated pandas dataframe into parquet files (instead of creating pyspark

dataframes from pandas ones).

4.2.2. Table attribute

Below are represented the attribute and a brief description of what they represent:

Author

Attribute	Description
Id	Unique identifier
name	Name of the Person
affiliation	The place (institution) at which the author conducted the research that they have reported/written about.
externalids.ORCID	Persistent digital identifier that distinguishes a researcher from any other
homepage	the webpage of the author
papercount	number of papers written by the author
s2url	semantic scholar webpage of the author
h-index	h-index of the author (according to semantic scholar)
citationcount	number of citations received by the publications of the author

Article

Attribute	Description
Id	unique identifier of the article
title	title of the article
url	internet address in which the article is published
ee	url of the pdf source of the article
m-date	last date in which the article has been modified
month	month in which the article has been published
year	year in which the article has been published
key	unique identifier of the article in DBLP.org website
pages	the page range where the article can be found in the specific issue of that journal
number	the issue number for a journal article
volume	the volume number of the journal

Journal

Attribute	Description
ISSN	unique identifier of the journal
name	name of the journal
NumArticles	the number of articles present in this Journal

4.3. Data Upload

4.3.1. Pre-processing operations

Requirements:

1. Python 3.*
   You can check if you have already installed python by opening the terminal ›_ and using the following command:

   ```
   python --version
   ```

   If you don't already have it, you can install it using the following linked guide .

2. Go to our github repository and copy into your workspace the **requirements.txt** file.
   The following command will allow you to install all the needed libraries:

   ```
   pip3 install .r requirements.txt
   ```

Steps:

1. Since for this delivery we are going to reuse the dataset we have processed for neo4j, if you have already created it you can skip this step, otherwise go back and perform the first 5 instructions of this section: 2.2.1

2. Go to our github repository and copy into the folder the **cucciolodifoca.py** file.
   Run it with the command:

   ```
   python cucciolodifoca.py
   ```

   You will be asked to choose what operation you want to perform:
   Choose **4** and then **1** for the Spark Setup;

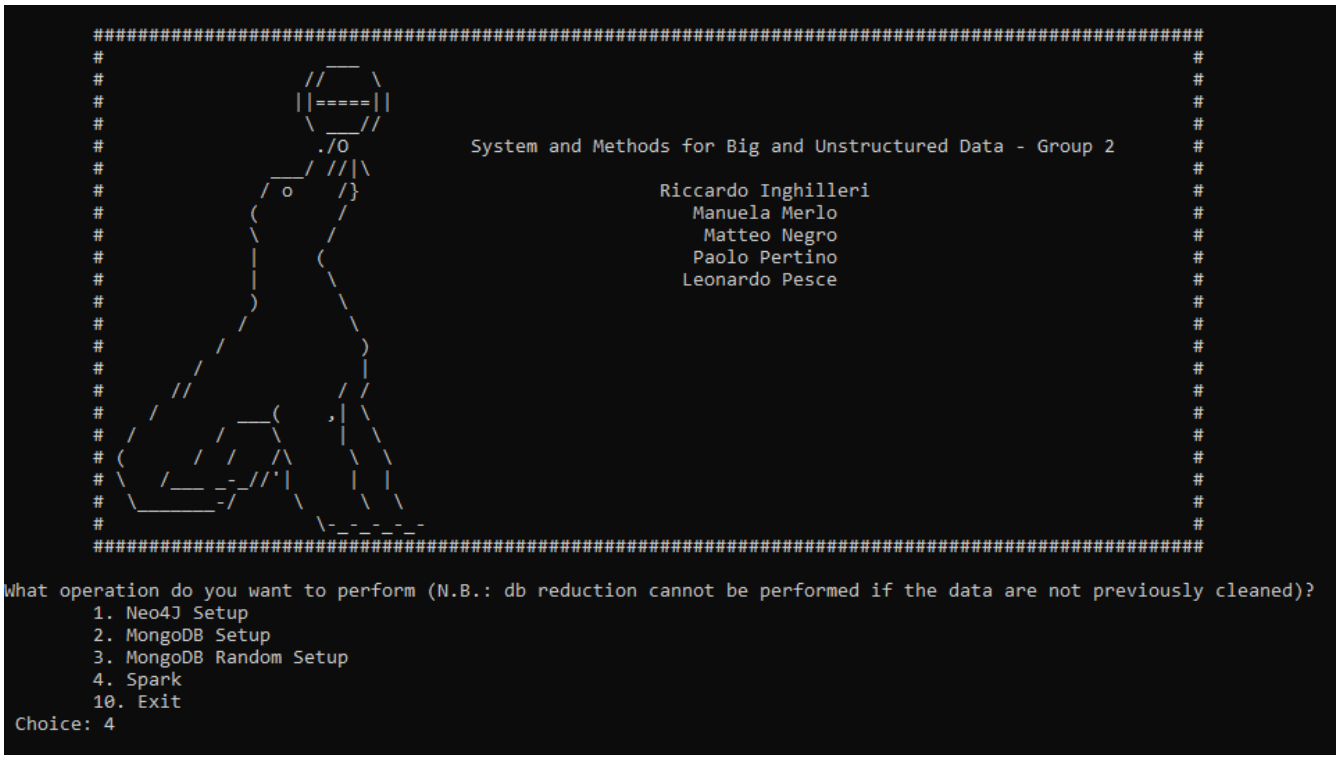


Figure 4.2: cucciolodifoca.py - General menu

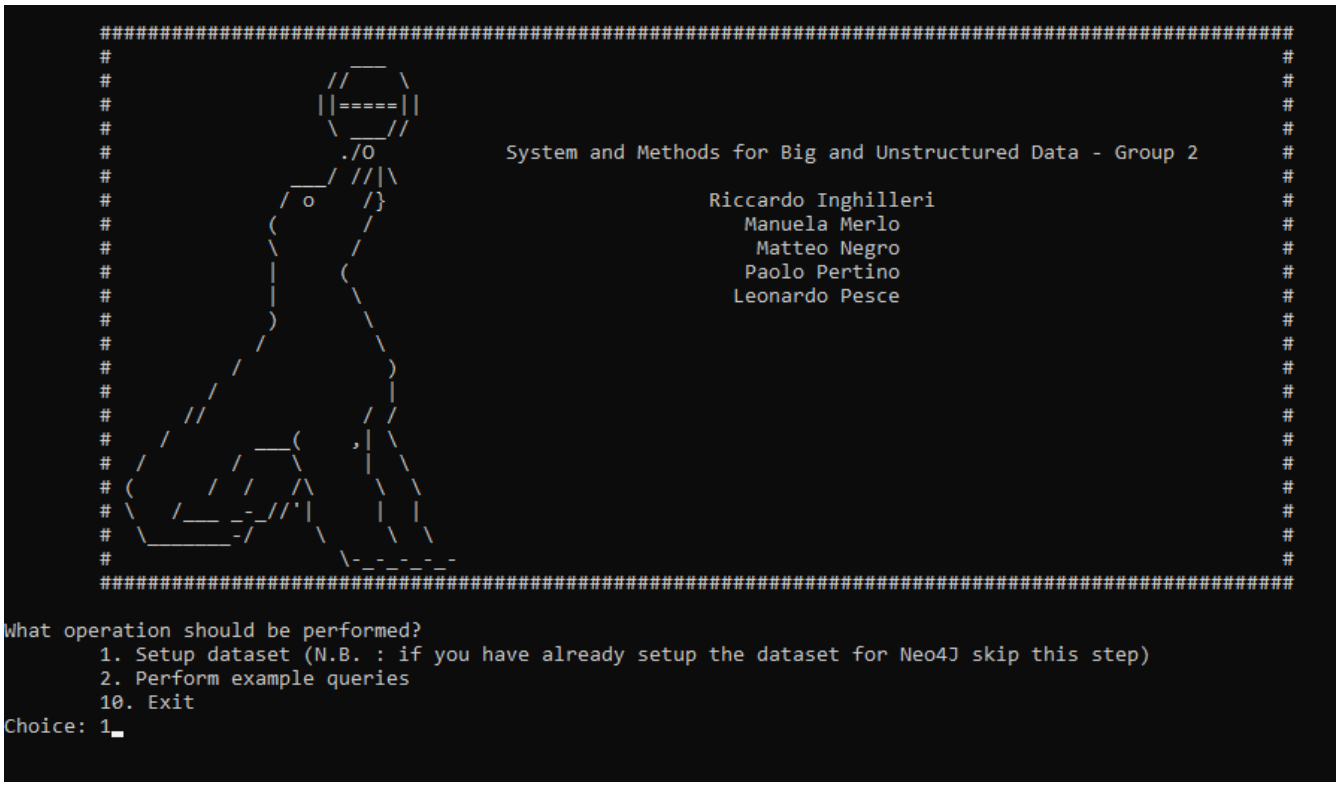


Figure 4.3: cucciolodifoca.py - Spark Menu

4.4. Data Model

Since we are using Spark there's no need to represent the relation (0:n)-(0:n) with a separate table (as in SQL). This is possible because Spark provides the array datatype and the ids of the relationship could be represented in an array of Integer. We thought of several possible ways to implement the relationships, each one with its advantages and disadvantages (for simplicity we report the two interpretations at the extremes of the trade-off):

- A first implementation is achieved by adding to the articles an array of authors' ids and the id of the Journal in which it is published.

Pros	Cons
There's no duplication of data and all the tables are reachable.	To check which articles have been written by an author we need to check all the authors of all the articles, by going through the array of authors of all the article entries in the articles dataframe). This is not efficient.

- A second implementation (starting from what was done in the first implementation) is achieved by adding, to the author an array representing what articles he/she wrote, and to the journals which articles contain. This could be seen as a double-arrow relationship.

Pros	Cons
The queries are faster (ex. it is possible from the author table to get the article he/she wrote simply by getting the array of article ids and matching them in the article table.	Some data are duplicated (the id of the entries of the tables).

Below we represent the Structure of the different tables and after each one, we describe the attributes that represent the relationships between the entities (that weren't described in the previous sections).

Author entity

```
schema = StructType([ \
    StructField(":ID", IntegerType(), False), \
    StructField("name", StringType(), True), \
    StructField("affiliations", ArrayType(StringType()), True) \
    StructField("homepage", StringType(), True) \
    StructField("papercount", IntegerType(), True), \
    StructField("citationcount", IntegerType(), True), \
```

```
    StructField("hindex", IntegerType(),True)\
    StructField("s2url", StringType(), True), \
    StructField("externalids.ORCID", StringType(), True), \
    StructField("written_articles_ids", ArrayType(IntegerType()), True) \
])
```

Attribute	Description
written_articles_ids	an array containing the ID of all the articles written by the author

Article entity:

```
schema = StructType([ \
    StructField("ID", IntegerType(), False), \
    StructField("ee", ArrayType(StringType()), True) \
    StructField("ee-type", ArrayType(StringType()), True)\
    StructField("key",StringType(),True)\
    StructField("mdate", DateType(), True), \
    StructField("month", StringType(), True), \
    StructField("note", ArrayType(StringType()), True), \
    StructField("note-label", StringType(), True), \
    StructField("publtype", StringType(), True), \
    StructField("title", StringType(), True), \
    StructField("url", ArrayType(StringType()), True), \
    StructField("year", IntegerType(), True),
    StructField("authors_ids", ArrayType(IntegerType()), True), \
    StructField("journal_id", IntegerType(), True), \
    StructField("number", StringType(), True), \
    StructField("pages", StringType(), True), \
    StructField("volume", StringType(),True), \
    StructField("citations", ArrayType(IntegerType()), True), \
    StructField("incoming_citations", ArrayType(IntegerType()), True), \
])
```

Attribute	Description
authors_ids	an array containing the ID of all the authors that wrote this article
journal_id	ID of the Journal in which the article is published
citations	an array containing the ID of all the cited article
incoming_citations	an array containing the ID of all articles that cite this one

Journal entity:

```
schema = StructType([ \
    StructField("ISSN", IntegerType(), False), \
    StructField("name", StringType(), True), \
    StructField("NumArticles", IntegerType(), True), \
    StructField("Articles", ArrayType(IntegerType()), True), \
])
```

Attribute	Description
articles	an array containing the ID of all the articles published in this Journal

4.5. Commands

Perform at least 5 data creation/update operations (add a new row, remove a row, etc.)

1. From the Author dataframe "delete" the row(s) whit the author named "S. Ceri"

```
df_author = spark.read.parquet(PARQUET_AUTHOR)
# ... df_author has to be considered as the complete dataset of the author
    imported from the parquet file.
df_author = df_author.filter(df_author.name != "S. Ceri")
```

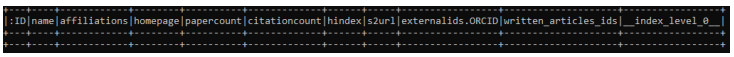
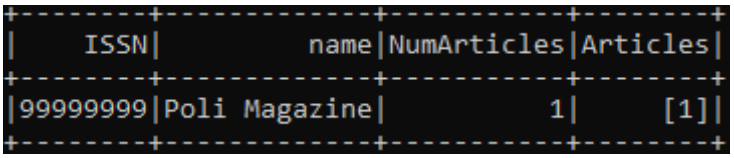


Figure 4.4: Filter on S. Ceri author after its deletion

2. Inserting a new row in the Journal Dataframe (with ISSN:99999999, name:Poli Magazine, NumArticles:1, Articles:1 (article with ID:1))

```
df_journal = spark.read.parquet(PARQUET_JOURNAL)
df_journal = df_journal.withColumn("NumArticles",col("NumArticles").
cast(IntegerType()))
# ... df_journal has to be considered as the complete dataset of the
journal imported from the parquet file.
journal_schema = StructType([StructField("ISSN", IntegerType(), False)
, StructField("name", StringType(), True), StructField("NumArticles",
IntegerType(), True), StructField("Articles", ArrayType(IntegerType())
, True)])
new_Journal = spark.createDataFrame(data = [(99999999, "Poli Magazine"
, 1, [1])], schema = journal_schema)
df_journal = df_journal.union(new_Journal)
```



ISSN	name	NumArticles	Articles
99999999	Poli Magazine	1	[1]

Figure 4.5: New journal added to journal dataframe

3. Setup the authors collection and saves it into a parquet file in the script directory [cucciolodifoca.py - lines 1231 - 1259]:

```
def setup_authors_collection():
    """Setup the authors collection and saves it into a parquet file in
    the script directory."""
```

```
# Reading data from csv files exploiting pandas library.
authors = pd.read_csv(AUTHORS_NODE_EXTENDED_PATH, sep=";")
articles = pd.read_csv(ARTICLE_FINAL_PATH, sep=";", low_memory=False,
header=None)
articles = pd.DataFrame({':START_ID' : articles.iloc[:, 0]})
authored_by = pd.read_csv(AUTHORED_BY_FINAL_PATH, sep=";")

# modifying types for each attributes to the correct ones while
dealing with NaN vals.
authors['affiliations'] = authors['affiliations'].apply(lambda x:
literal_eval(x) if pd.notna(x) else None)
authors['externalids.ORCID'] = authors['externalids.ORCID'].apply(
lambda x: literal_eval(x) if pd.notna(x) else None)
authors['externalids.ORCID'] = authors['externalids.ORCID'].str[0]
authors['hindex'] = authors['hindex'].fillna(-1)
authors['hindex'] = authors['hindex'].astype('int32')
authors['papercount'] = authors['papercount'].fillna(-1)
authors['papercount'] = authors['papercount'].astype('int32')
authors['citationcount'] = authors['citationcount'].fillna(-1)
authors['citationcount'] = authors['citationcount'].astype('int32')

# Merging articles with authored_by relationship to get for each
author his written articles.
joined_df = pd.merge(articles, authored_by, on = ":START_ID")
joined_df = joined_df.groupby(':END_ID')[":START_ID"].apply(list).
reset_index(name='written_articles_ids')
joined_df.rename(columns={':END_ID': ':ID'}, inplace=True)

authors = pd.merge(authors, joined_df, on=":ID", how="left")
authors["written_articles_ids"] = authors['written_articles_ids'].
apply(lambda x: x if isinstance(x, list) else [])

# Finally saving to parquet files out dataframe.
authors.to_parquet(PARQUET_AUTHOR, compression=None)
```

4. Setup the articles collection and saves it into a parquet file in the script directory [cucciolodifoca.py - lines 1261 - 1323]:

```
def setup_articles_collection():
    """Setup the articles collection and saves it into a parquet file in
```

```
the script directory."""
 # insert the correct headers in the article.csv
 if not isfile(ARTICLE_FINAL_PATH_EXTENDED):
     with open(ARTICLE_FINAL_HEADER_PATH) as header_file:
         header_list = list(csv.reader(header_file, delimiter=";"))
         header_list = header_list[0]
     for i in range(len(header_list)):
         header_list[i] = header_list[i].split(":")
         header_list[i] = header_list[i][0]
     header_list.insert(0,":ID")
     del header_list[1]
     articles = pd.read_csv(ARTICLE_FINAL_PATH, sep=";", low_memory=
False)
     articles.to_csv(ARTICLE_FINAL_PATH_EXTENDED, header=header_list,
sep=";", index=False)

 # modifying types for each attributes to the correct ones while
dealing with NaN vals.
 articles = pd.read_csv(ARTICLE_FINAL_PATH_EXTENDED, sep=";",
low_memory=False)
 articles.drop(['cdate', 'note-type'], inplace=True, axis=1)
 articles[":ID"] = articles[":ID"].astype("int32")
 articles['ee'] = articles['ee'].apply(lambda x: x.split("|") if pd.
notna(x) else None)
 articles['ee-type'] = articles['ee-type'].apply(lambda x: x.split("|")
  if pd.notna(x) else None)
 articles['mdate'] = articles['mdate'].apply(lambda x: pd.to_datetime(x
).date() if pd.notna(x) else None)
 articles['note'] = articles['note'].apply(lambda x: x.split("|") if pd
.notna(x) else None)
 articles["url"] = articles["url"].apply(lambda x: x if pd.notna(x)
else "")
 articles["url"] = articles["url"].apply(lambda x: x.split("|"))
 articles['year'] = articles['year'].fillna(-1)
 articles['year'] = articles['year'].astype('int32')

 # For each article we want to compile an array containing the ids of
its authors.
 authored_by = pd.read_csv(AUTHORED_BY_FINAL_PATH, sep=";")
```

```
authored_by = authored_by.groupby(':START_ID')[":END_ID"].apply(list).
reset_index(name='authors_ids')
authored_by.rename(columns={':START_ID': ':ID'}, inplace=True)

articles = pd.merge(articles, authored_by, on=":ID", how="left")
articles["authors_ids"] = articles['authors_ids'].apply(lambda x: x if
 isinstance(x, list) else None)

# Fetching the journal in which the paper is published in and saving
into the article entity its identifier.
published_in = pd.read_csv(PUBLISHED_IN_PATH, sep=';', low_memory=
False)
published_in[':END_ID'] = published_in[':END_ID'].fillna(-1)
published_in[':END_ID'] = published_in[':END_ID'].astype('int32')
published_in.rename(columns={':START_ID': ':ID', ':END_ID': '
journal_id'}, inplace=True)
articles = pd.merge(articles, published_in, on=":ID", how="left")
articles['journal_id'] = articles['journal_id'].apply(lambda x: int(x)
 if pd.notna(x) else None)

# for each article we want to compile 2 arrays:
#   - citations contains all the ids of the cited papers by the
current one.
#   - incoming_citations contains all the ids of the papers which cite
 the current one.
cite = pd.read_csv(CITE_RELATIONSHIP, sep=';', low_memory=False)
cite['START_ID'] = cite[':START_ID'].fillna(-1)
cite[':START_ID'] = cite[':START_ID'].astype('int32')
cite[':END_ID'] = cite[':END_ID'].fillna(-1)
cite[':END_ID'] = cite[':END_ID'].astype('int32')
cite[':START_ID'] = cite[':START_ID'].apply(lambda x: int(x))
cite[':END_ID'] = cite[':END_ID'].apply(lambda x: int(x))
citations = cite.groupby(':START_ID')[":END_ID"].apply(list).
reset_index(name='citations')
citations.rename(columns={':START_ID': ':ID'}, inplace=True)
articles = pd.merge(articles, citations, on=":ID", how="left")
incoming_citations = cite.groupby(':END_ID')[":START_ID"].apply(list).
reset_index(name='incoming_citations')
incoming_citations.rename(columns={':END_ID': ':ID'}, inplace=True)
```

```
articles = pd.merge(articles, incoming_citations, on=":ID", how="left"
)

# Saving the modified dataframe to parquet file.
articles.to_parquet(PARQUET_ARTICLE, compression=None)
```

5. Setup the journals collection and saves it into a parquet file in the script directory [cucciolodifoca.py - lines 1326 - 1367]:

```
def setup_journals_collection(spark: SparkSession):
"""Setup the journals collection and saves it into a parquet file in
the script directory.

Args:
    spark (SparkSession): spark session currently active.
"""
# Reading data from csv files exploiting pandas library.
df_journal = spark.read.csv(JOURNAL_PATH, sep=";", header=True,
inferSchema=True)
df_journal = df_journal.withColumnRenamed(":ID","ISSN").
withColumnRenamed("journal:string","name")
df_journal_published_in = spark.read.csv(PUBLISHED_IN_PATH, sep=";",
header=True, inferSchema=True)
df_journal_published_in = df_journal_published_in.withColumnRenamed(":
START_ID","START_ID").withColumnRenamed(":END_ID","END_ID")

# Extracting the Journals' ids.
set_journal = df_journal_published_in.select(collect_set("END_ID")).
collect()[0][0]

# Building a dataframe that contains an array of Articles' ids per
each Journal.
dataTmp = {}
csv_file = pd.read_csv(PUBLISHED_IN_PATH, sep=";", low_memory=False)
for i in range(1, len(csv_file.index)):
    if csv_file.loc[i, ':END_ID'] in dataTmp.keys():
        dataTmp[csv_file.loc[i, ':END_ID']].append(int(csv_file.loc[i,
 ':START_ID']))
    else:
        dataTmp[csv_file.loc[i, ':END_ID']] = [(int(csv_file.loc[i, ':
```

```
START_ID'']))]
        data = []
        columns = StructType([
                    StructField("END_ID", IntegerType(), False),
                    StructField("NumArticles", IntegerType(), True),
                    StructField("Articles", ArrayType(IntegerType()), True)
                ])
        for el in set_journal:
            data.append([el, len(dataTmp[el]), dataTmp[el]])
        df_tmp = spark.createDataFrame(data = data, schema = columns)

        # Merging the Journals with their article ids.
        df_journal_final = df_journal.join(df_tmp, df_journal.ISSN == df_tmp.
        END_ID, "left").drop(df_tmp.END_ID)

        # spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")
        # spark.conf.set("spark.sql.execution.arrow.enabled", "true")

        # Saving the modified dataframe to parquet file.
        result_pdf = df_journal_final.select("*").toPandas()
        result_pdf.to_parquet(PARQUET_JOURNAL, compression=None)
```

In order to effectively import the data stored in the parquet files we use the following piece of code [cucciolodifoca.py - lines 1143 - 1159]:

```
# We need these casts to Integer or Array<Integer> because Pandas saves parquet
    files with Int64 format.
# Since our ids are Int32 numbers, but are read by pyspark as Int64 (long), we
   cast them in the right form.
print("Importing articles")
df_article = spark.read.parquet(PARQUET_ARTICLE)
df_article = df_article.withColumn("citations",col("citations").cast(ArrayType(
   IntegerType())))
df_article = df_article.withColumn("incoming_citations",col("incoming_citations
   ").cast(ArrayType(IntegerType())))
df_article = df_article.withColumn("authors_ids",col("authors_ids").cast(
   ArrayType(IntegerType())))
df_article = df_article.withColumn("journal_id",col("journal_id").cast(
   IntegerType()))
df_article = df_article.withColumnRenamed(":ID", "ID")
```

```
df_article = set_df_columns_nullable(spark, df_article, ["ID"], False)
print("Importing journals")
df_journal = spark.read.parquet(PARQUET_JOURNAL)
df_journal = df_journal.withColumn("NumArticles",col("NumArticles").cast(
    IntegerType()))
df_journal = set_df_columns_nullable(spark, df_journal, ["ISSN"], False)
print("Importing authors")
df_author = spark.read.parquet(PARQUET_AUTHOR)
df_author = df_author.withColumn("written_articles_ids",col("
    written_articles_ids").cast(ArrayType(IntegerType())))
df_author = df_author.withColumn(":ID",col(":ID").cast(IntegerType()))
df_author = set_df_columns_nullable(spark, df_author, [":ID"], False)
```

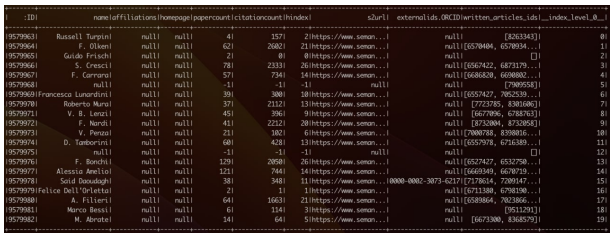


Figure 4.6: Imported authors dataframe

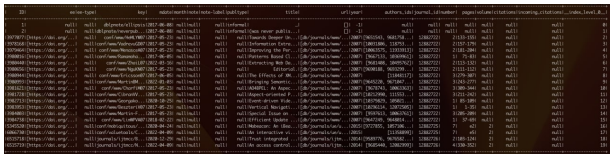


Figure 4.7: Imported articles dataframe

ISSN	name	NumArticles	Articles
12882722	World Wide Web	1170	[7163905, 7163906...]
12882723	SIGMOD Rec.	1826	[8347648, 8347650...]
12882724	SIGMOD Record	null	null
12882725	EAI Endorsed Tran...	58	[8938880, 8938881...]
12882726	Int. J. Trust. Ma...	60	[6515718, 6515719...]
12882727	Expert Syst. Appl. X	17	[6515778, 6515779...]
12882728	Bull. dInformatiq...	422	[6515795, 6515796...]
12882729	Trans. Large Scal...	322	[6516217, 6516218...]
12882730	Stud. Inform. Univ.	175	[6516736, 6516737...]
12882731	IEEE Intell. Tran...	781	[6516765, 6516766...]
12882732	Control. Cybern.	14	[6517546, 6517547...]
12882733	Syst. Control. Lett.	2869	[6517560, 6517561...]
12882734	Sci. Comput. Prog...	2350	[9295119, 9295120...]
12882735	Found. Comput. Math.	646	[6520832, 6520833...]
12882736	Theor. Comput. Sci.	14170	[6521078, 6521079...]
12882737	Inf. Technol. Dev.	437	[6535390, 6535391...]
12882738	EURASIP J. Adv. S...	3887	[6537216, 6537217...]
12882739	IEEE Embed. Syst....	420	[6539714, 6539715...]
12882740	J. Assoc. Inf. Syst.	763	[6540134, 6540135...]
12882741	EAI Endorsed Tran...	151	[6540897, 6540898...]

Figure 4.8: Imported journals dataframe

4.6. Queries

In the following section we report the python code relative to the 10 queries we wrote. You can perform them also by running our script and selecting the following options:

```
python cucciolodifoca.py
```

You will be asked to choose what operation you want to perform:
Choose **4** and then **2** to perform the queries; finally select, by using a number from 1 to 10, the query you would like to perform.

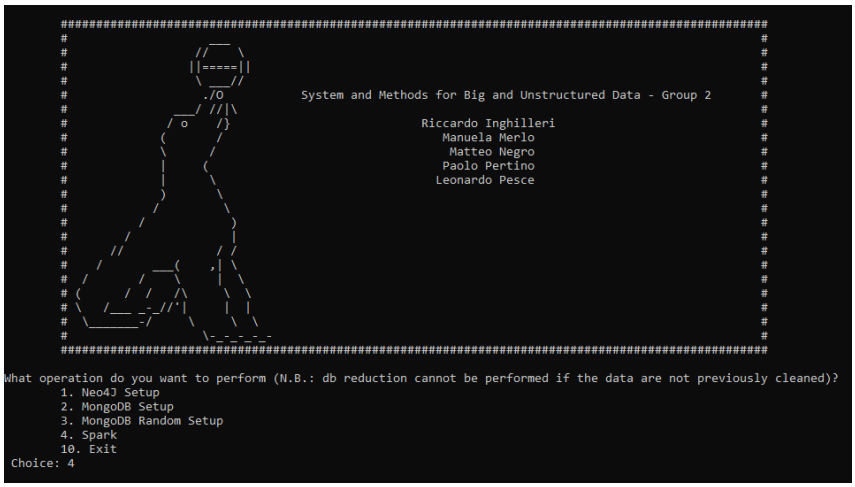


Figure 4.9: cucciolodifoca.py - General menu

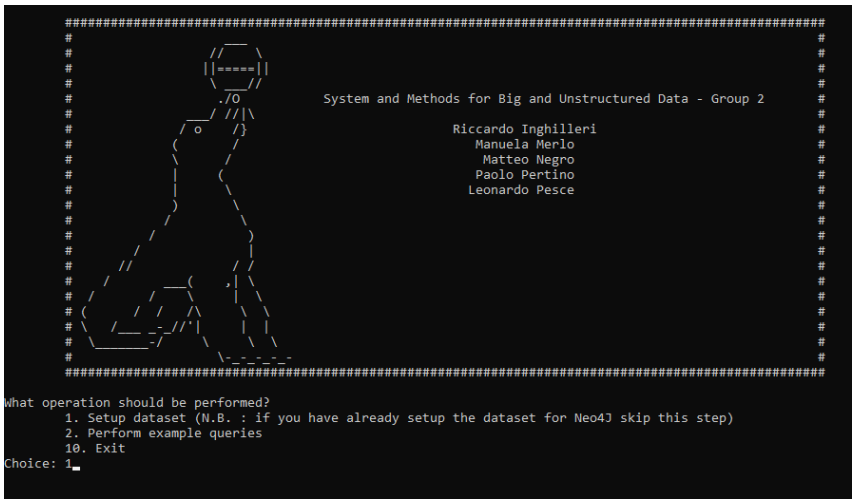


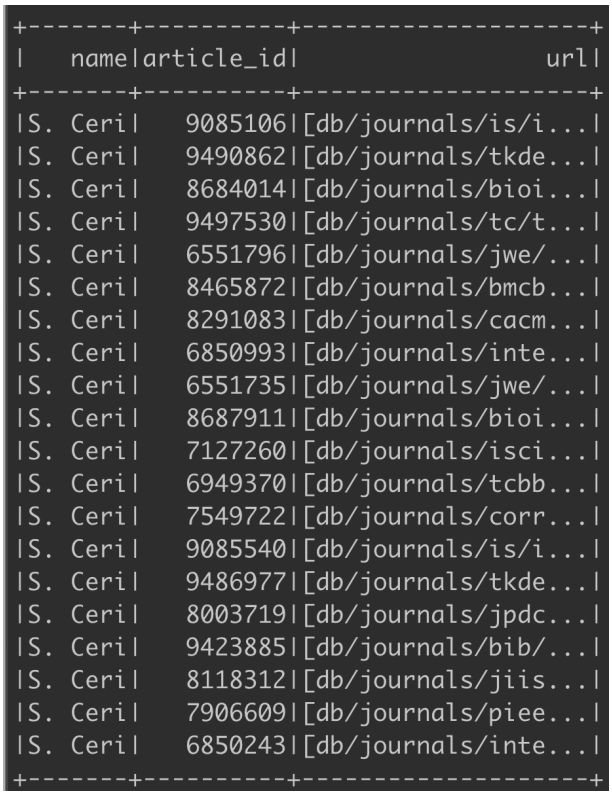
Figure 4.10: cucciolodifoca.py - Spark Menu

Figure 4.11: cucciolodifoca.py - Spark Queries Menu

1. Take the articles of a given author. (WHERE, JOIN) [cucciolodifoca.py - lines 1372 - 1389]

```
# Select the author's dataframe entry whose attribute "name" is equal to "
    S. Ceri", explode the field 'written_articles_ids' and rename the
    resulting column as 'article_id'.
df_selected_authors = df_authors.filter(df_authors.name.rlike("S.\sCeri$")
    )
exploded_df_authors = df_selected_authors.select(df_authors.name, explode(
    df_authors.written_articles_ids)).withColumnRenamed("col", "article_id
    ")

# Perform the join between the exploded_df_authors and the df_articles
    dataframes and select the fields "name", "article_id" and "url"
exploded_df_authors = exploded_df_authors.join(df_articles,
    exploded_df_authors.article_id == df_articles.ID)
exploded_df_authors.select(exploded_df_authors.name, exploded_df_authors.
    article_id, exploded_df_authors.url).show()
```



name	article_id	url
S. Ceri	9085106	[db/journals/is/i...
S. Ceri	9490862	[db/journals/tkde...
S. Ceri	8684014	[db/journals/bioi...
S. Ceri	9497530	[db/journals/tc/t...
S. Ceri	6551796	[db/journals/jwe/...
S. Ceri	8465872	[db/journals/bmcb...
S. Ceri	8291083	[db/journals/cacm...
S. Ceri	6850993	[db/journals/inte...
S. Ceri	6551735	[db/journals/jwe/...
S. Ceri	8687911	[db/journals/bioi...
S. Ceri	7127260	[db/journals/isci...
S. Ceri	6949370	[db/journals/tcbb...
S. Ceri	7549722	[db/journals/corr...
S. Ceri	9085540	[db/journals/is/i...
S. Ceri	9486977	[db/journals/tkde...
S. Ceri	8003719	[db/journals/jpdc...
S. Ceri	9423885	[db/journals/bib/...
S. Ceri	8118312	[db/journals/jiis...
S. Ceri	7906609	[db/journals/piee...
S. Ceri	6850243	[db/journals/inte...

Figure 4.12: Result Query 1

2. Fetch the articles which have Machine Learning in their title. (WHERE, LIMIT, LIKE) [cucciolodifoca.py - lines 1391 - 1399]

```
df_articles.filter(df_articles.title.like("%Machine Learning%")).limit(5).
    show()
```

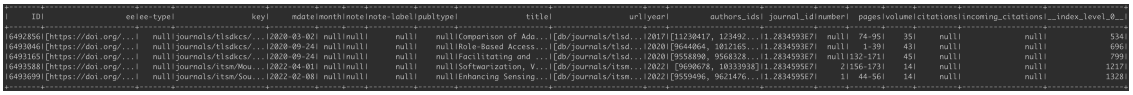


Figure 4.13: Result Query 2

3. Fetch all the articles written by the 5 authors with highest citationcount. (WHERE, IN, Nested Query) [cucciolodifoca.py - lines 1401 - 1421]

```
# Sort the dadatframe in descending order with respect to the '
    citationcount' field and return the ids of the 5 authors with the
    highest 'citationcount' in a list
top_authors = df_authors.orderBy(col("citationcount").desc()).select(col("
    :ID"), df_authors.citationcount, df_authors.name).withColumnRenamed(":
    ID", "auth_id").limit(5)
top_authors_ids = top_authors.select(collect_set("auth_id")).collect()
    [0][0]
```

```
# Explode the authors_ids field and rename the resulting column as '
    author_id'.
explode_df_articles = df_articles.select(df_articles.ID, df_articles.title
    , explode(df_articles.authors_ids)).withColumnRenamed("col", "
    author_id")
# Filter the articles whose author_id field value is contained in the list
     of the 5 top authors
articles_of_top_authors = explode_df_articles.filter(col("author_id").isin
    (top_authors_ids))
# Perform the join between the articles_of_top_authors and the top_authors
     dataframes and drop the columns corresponding to the "ID", "author_id
    ", "auth_id", "citation" fields.
articles_of_top_authors = articles_of_top_authors.join(top_authors,
    articles_of_top_authors.author_id == top_authors.auth_id).drop("ID", "
    author_id", "auth_id", "citation")
articles_of_top_authors.show()
```

```
+--------------------+-------------+-------------+
|               title|citationcount|         name|
+--------------------+-------------+-------------+
|Diffusion of Cont...|       372099|Yoshua Bengio|
|The Bottleneck Si...|       372099|Yoshua Bengio|
|A structural vari...|       486145|    E. Lander|
|      Deep learning.|       372099|Yoshua Bengio|
|A spatial model p...|       345821|B. Vogelstein|
|Neural net langua...|       372099|Yoshua Bengio|
|A Connectionist A...|       372099|Yoshua Bengio|
|Scaling Large Lea...|       372099|Yoshua Bengio|
|Context-dependent...|       372099|Yoshua Bengio|
|On integrating a ...|       372099|Yoshua Bengio|
|Multi-way, multil...|       372099|Yoshua Bengio|
|Learning the dyna...|       372099|Yoshua Bengio|
|Detonation Classi...|       372099|Yoshua Bengio|
|Decision trees do...|       372099|Yoshua Bengio|
|Learning normaliz...|       372099|Yoshua Bengio|
|Brain tumor segme...|       372099|Yoshua Bengio|
|Machine learning ...|       372099|Yoshua Bengio|
|Taking on the cur...|       372099|Yoshua Bengio|
|A Hybrid Pareto M...|       372099|Yoshua Bengio|
|Learning long-ter...|       372099|Yoshua Bengio|
+--------------------+-------------+-------------+
```

Figure 4.14: Result Query 3

4. Fetch the journal containing the highest number of articles written in 2010. (GROUP BY, 1 JOIN, AS) [cucciolodifoca.py - lines 1423 - 1447]

```
# Filter the articles whose field year is equal to 2010.
df_articles = df_articles.filter(df_articles.year == 2010)
#  Perform the join between df_articles and df_journal dataframes.
articles_in_journal = df_articles.join(df_journal, df_articles.journal_id
    == df_journal.ISSN)
# Group with respect to the fields "year", "journal_id", "name" , count
    how many articles have been puslished in a journal and rename the
    resulting column as 'Number of Articles in Journal in 2010'.
num_articles_per_year_per_journal = articles_in_journal \
    .groupby("year", "journal_id", "name") \
    .agg(count("*").alias("Number of Articles in Journal in 2010"))
# Sort the dadaframe in descending order with respect to the 'Number of
    Articles in Journal in 2010' field and return the first of the
    remaining entries.
best_journal_in_2010 = num_articles_per_year_per_journal.orderBy(col("
    Number of Articles in Journal in 2010").desc()).limit(1)
# Drop the 'journal_id' column.
best_journal_in_2010.drop("journal_id").show()
```

```
+----+----+-----------------------------------+
|year|name|Number of Articles in Journal in 2010|
+----+----+-----------------------------------+
|2010|CoRR|                               7179|
+----+----+-----------------------------------+
```

Figure 4.15: Result Query 4

5. Calculate the h-index of S. Ceri. (WHERE, GROUP BY) [cucciolodifoca.py - lines 1449 - 1480]

```
# Select the author entry whose attribute "name" is equal to "S. Ceri",
    explode the field 'written_articles_ids' and rename the resulting
    column as 'publication_ID'.
author_name = "S. Ceri"
df_selected_authors = df_authors.filter(df_authors.name == author_name)
articles_written_by_selected_authors = df_selected_authors.select(
    df_authors.name, explode(df_authors.written_articles_ids)).
    withColumnRenamed("col", "publication_ID")
# Count the number of articles of the selected author
total_articles = articles_written_by_selected_authors.count()
# Perform the join between the articles_written_by_selected_authors and
    the df-articles dataframes.
articles_written_by_selected_authors_data =
    articles_written_by_selected_authors.join(df_articles,
    articles_written_by_selected_authors.publication_ID == df_articles.ID)
# Select the fields "name" and "publication_ID", explode the the field '
    incoming_citations' and rename the resulting column as 'inc_cit_id'.
articles_written_by_selected_authors_data_expanded_ingoing_cit =
    articles_written_by_selected_authors_data.select(col("name"), col("
    publication_ID"), explode("incoming_citations")).withColumnRenamed("
    col", "inc_cit_id")
# Count the number of incoming citations for each article and remane the
    resulting column as "num_ingoing_cit".
articles_with_num_ing_cit =
    articles_written_by_selected_authors_data_expanded_ingoing_cit.groupBy
    ("name", "publication_ID").agg(count("*").alias("num_ingoing_cit"))
# Return a list containing the number of incoming citations for each
    article of the author.
incoming_cit_num_for_articles = articles_with_num_ing_cit.select(
    collect_list("num_ingoing_cit")).collect()[0][0]

indices = []
for i in range(total_articles, -1, -1):
    if incoming_cit_num_for_articles.count(i) >= i:
        indices.append(i)

print(f"{author_name}'s h-index: {np.max(indices)}")
```

```
S. Ceri's h-index: 4
```

Figure 4.16: Result Query 5

6. Return all the years in which there are at least 20 articles. (GROUP BY, HAVING, AS) [cucciolodifoca.py - lines 1482 - 1493]

```
# Group with respect to the fields "year", count how many articles have
    been puslished per year and return the years in which were published
    at least 20 articles.
df_articles.groupBy("year").agg(count("*").alias("article_per_year")).
    filter(col("article_per_year") > 20).show(truncate = False)
```

```
+----+----------------+
|year|article_per_year|
+----+----------------+
|1959|446             |
|1990|12306           |
|1975|3334            |
|1977|3754            |
|2003|44941           |
|2007|71568           |
|2018|184779          |
|1974|3130            |
|2015|129096          |
|2023|2842            |
|1955|156             |
|2006|64648           |
|2022|248763          |
|1978|3943            |
|1961|683             |
|2013|115489          |
|1952|34              |
|1956|220             |
|1988|10138           |
|1997|24645           |
+----+----------------+
```

Figure 4.17: Result Query 6

7. Return the years in which at least 20 articles with at least 20 incoming citations have been published. (WHERE, GROUP BY, HAVING, AS) [cucciolodifoca.py - lines 1496 - 1518]

```
# Explode the field 'incoming_citations' and rename the resulting column
    as 'inc_cit'.
exploded_df_articles = df_articles.select(df_articles.ID, df_articles.year
    , explode(df_articles.incoming_citations))
exploded_df_articles = exploded_df_articles.withColumnRenamed("col", "
    inc_cit")
```

```
# Group with respect to the field 'ID', count the number of incoming
    citations for each article and return a list of articles' IDs cited at
     least 20 times.
good_articles_ids = exploded_df_articles.groupBy(exploded_df_articles.ID).
    agg(count("ID").alias('num_cit')).where(col("num_cit") > 20).select(
    collect_set("ID")).collect()[0][0]

# Filter the articles whose filed 'ID' is contained in the list "
    good_articles_ids"
df_articles = df_articles.filter(df_articles.ID.isin(good_articles_ids))

# Group articles according to the year of publication
# Count the number of articles published for each year and rename the
    resulting columns as 'Articles number per year'
# Filter the years with at least 20 publications
# Sort in descending order the years according to the number of articles
    published
df_articles.groupBy(df_articles.year).agg(count("*").alias('Articles
    number per year')).where(col("Articles number per year") > 20).orderBy
    (col("year").desc()).show()
```

```
+----+------------------------+
|year|Articles number per year|
+----+------------------------+
|1990|                      22|
|1986|                      28|
|1984|                      31|
|1983|                      26|
|1982|                      22|
|1979|                      27|
+----+------------------------+
```

Figure 4.18: Result Query 7

8. Calculate the I-10 index of S. Ceri. (WHERE, Nested Query (i.e., 2-step Queries), GROUP BY) [cucciolodifoca.py - lines 1521 - 1545]

```
# Select the author's dataframe entry whose attribute "name" is equal to "
    S. Ceri"
author_name = "S. Ceri"
df_authors = df_authors.filter(df_authors.name == author_name)
# Explode the field 'written_articles_ids' and rename the resulting column
     as 'publication_ID'.
```

```
exploded_df_authors = df_authors.select(df_authors.name, explode(
    df_authors.written_articles_ids))
exploded_df_authors = exploded_df_authors.withColumnRenamed("col", "
    publication_ID")
# Explode the field 'incoming_citations'
exploded_df_articles = df_articles.select(col("ID"), explode(df_articles.
    incoming_citations))
# Group with respect to the fields "ID" and count the number of incoming
    citations for each article.
exploded_df_articles = exploded_df_articles.groupby(col("ID")).count().
    withColumnRenamed("count", "num_of_incoming_citations")
# Filter articles with at least 10 incoming citations
exploded_df_articles = exploded_df_articles.filter("
    num_of_incoming_citations >= 10")
# Perform the join between the  exploded_df_authors and the
    exploded_df_articles dataframes.
exploded_df_authors = exploded_df_authors.join(exploded_df_articles,
    exploded_df_authors.publication_ID == exploded_df_articles.ID, "inner"
    )
# Group with respect to the author's name and return the I-10 index.
exploded_df_authors.groupBy(df_authors.name).count().withColumnRenamed("
    count", "I-10 index").show()
```

```
+-------+----------+
|   name|I-10 index|
+-------+----------+
|S. Ceri|         6|
+-------+----------+
```

Figure 4.19: Result Query 8

9. Find those journals which refer to the medicine topic in their name and contains at least 10 articles with the medicine keyword in their title. (WHERE, GROUP BY, HAVING, 1 JOIN) [cucciolodifoca.py - lines 1548 - 1572]

```
# Select the Journals whose filed "name" contains the word 'Medicine'.
medical_journals = df_journals.filter(df_journals.name.like("%Medicine%"))
# Explode the field 'Articles' and rename the resulting column as '
    article_id'.
medical_journals = medical_journals.select("ISSN", "name", explode("
    Articles")).withColumnRenamed("col", "article_id")
# Perform the join between the medical_journals and the df_articles
```

```
    dataframes.
article_of_journals_data = medical_journals.join(df_articles,
    medical_journals.article_id == df_articles.ID)
# Filter the article's dataframe entries whose field 'title' (of the
    article) contains the word 'medicine' or 'Medicine'
#Group with respect to the fields "ISSN" and "name" (of the journal).
#Count the Number of pertinent articles for each journal.
#Filter the journals with at least 10 pertinent articles.
article_of_journals_data = article_of_journals_data \
    .filter(article_of_journals_data.title.rlike("Medicine|medicine")) \
    .groupBy("ISSN", "name") \
    .agg(count("*").alias("Num Articles with Medicine in title in journal"
    )) \
    .where(col("Num Articles with Medicine in title in journal") >= 10)
article_of_journals_data.show()
```

```
+--------+--------------------+-------------------------------------------+
|    ISSN|                name|Num Articles with Medicine in title in journal|
+--------+--------------------+-------------------------------------------+
|12836420|Comput. Biol. Med...|                                         44|
|12835142|Comput. Math. Met...|                                         25|
|12836185|Int. J. Funct. In...|                                         17|
|12835388|Artif. Intell. Me...|                                         76|
+--------+--------------------+-------------------------------------------+
```

Figure 4.20: Result Query 9

10. Find those journals on which authors coming from Politecnico di Milano have published more than 5 articles. (WHERE, GROUP BY, HAVING, 2 JOINs) [cucciolodifoca.py - lines 1574 - 1608]

```
# Explode the field 'written_articles_ids' and rename the resulting column
    as 'publication_ID'.
exploded_df_authors = df_authors.select(df_authors.name, df_authors.
    affiliations, explode(df_authors.written_articles_ids))
exploded_df_authors = exploded_df_authors.withColumnRenamed("col", "
    publication_ID")

# Explode the field 'affiliations' and rename the resulting column as '
    affiliation'.
exploded_df_authors = exploded_df_authors.select(exploded_df_authors.
    publication_ID, explode(exploded_df_authors.affiliations))
exploded_df_authors = exploded_df_authors.withColumnRenamed("col", "
    affiliation")
```

```
# Rename the column "name" as 'journal_name'.
df_journals = df_journals.withColumnRenamed("name", "journal_name")

# Perform the join between the df_articles and the df_journals dataframes.
df_articles = df_articles.join(df_journals, df_articles.journal_id ==
    df_journals.ISSN, "inner")
df_articles = df_articles.select(df_articles.journal_name, df_articles.ID,
     df_articles.journal_id)

# Perform the join between the exploded_df_authors and the df_articles
    dataframes.
exploded_df_authors = exploded_df_authors.join(df_articles,
    exploded_df_authors.publication_ID == df_articles.ID, "inner")

# Group with respect to the fields "affiliation", "journal_id" and"
    journal_name".
#Count the number of publications for each affiliation in each journal
# Filter the journals which contain at least 5 articles written by author
    from 'Politecnico di milano".
# Delete the column "journal_id"
exploded_df_authors \
     .groupBy("affiliation", "journal_id", "journal_name") \
     .count() \
     .where(("count >= 5")) \
     .withColumnRenamed("count", "Number of publication of affiliation in
    journal") \
     .filter(exploded_df_authors.affiliation == "Politecnico di Milano") \
     .drop("journal_id") \
     .show(truncate = False)
```

```
+--------------------+-------------------------+-----------------------------------------------+
|affiliation         |journal_name             |Number of publication of affiliation in journal|
+--------------------+-------------------------+-----------------------------------------------+
|Politecnico di Milano|CoRR                     |75                                             |
|Politecnico di Milano|J. Web Eng.              |5                                              |
|Politecnico di Milano|Mach. Learn.             |5                                              |
|Politecnico di Milano|J. Comput. Phys.         |9                                              |
|Politecnico di Milano|IACR Cryptol. ePrint Arch.|6                                             |
+--------------------+-------------------------+-----------------------------------------------+
```

Figure 4.21: Result Query 10

4.6.1. Performance

The setup in which we were able to perform the command and the query is a single node (PC). This did not allow us to take advantage of one of the main features of Spark: parallelization on multiple nodes. Also, in our specific case, the usage of "only" 6 million entries (approximately and across all the tables) is not really what Spark was built for. In fact, Spark has been developed as a tool for fast analytics on huge amounts of data. To optimize for this dimensions parallelization is necessary.

5 | Changelog

5.1. Version 1.1 - Updated on 13-12-2022

- **Assumptions**
  We changed some assumptions which better fit with later stages of the project.
- **General ER diagram**
  Part_of relationship between *PhDThesis* and *Series* have been deleted.
- **Neo4J Section**
  Graph Data Model modified.
  In the previous version the relationship between *Inproceedings* and *Proceedings* was directed in the wrong way.
- **MongoDB queries performances**
  Explicited an addendum with further reasoning, since some queries after the creation of indices were actually losing in terms of performances.

List of Figures

Bibliography

[1] W. Ammar, D. Groeneveld, C. Bhagavatula, and I. Beltagy. Construction of the literature graph in semantic scholar. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, pages 84–91, New Orleans - Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-3011. URL https://aclanthology.org/N18-3011.

[2] K. Lo, L. L. Wang, M. Neumann, R. Kinney, and D. Weld. S2ORC: The semantic scholar open research corpus. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4969–4983, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.447. URL https://www.aclweb.org/anthology/2020.acl-main.447.

[3] J. P. Wahle, T. Ruas, S. M. Mohammad, and B. Gipp. D3: A massive dataset of scholarly metadata for analyzing the state of computer science research. In *Proceedings of The 13th Language Resources and Evaluation Conference*, Marseille, France, July 2022. European Language Resources Association.

[4] L. L. Wang, K. Lo, Y. Chandrasekhar, R. Reas, J. Yang, D. Burdick, D. Eide, K. Funk, Y. Katsis, R. M. Kinney, Y. Li, Z. Liu, W. Merrill, P. Mooney, D. A. Murdick, D. Rishi, J. Sheehan, Z. Shen, B. Stilson, A. D. Wade, K. Wang, N. X. R. Wang, C. Wilhelm, B. Xie, D. M. Raymond, D. S. Weld, O. Etzioni, and S. Kohlmeier. CORD-19: The COVID-19 open research dataset. In *Proceedings of the 1st Workshop on NLP for COVID-19 at ACL 2020*, Online, July 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.nlpcovid19-acl.1.