POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Systems and Methods for Big and Unstructured Data Project

Author(s): **BRUNETTI Enrico**
LODI RIZZINI Pietro
PAESANO Emanuele
PIZZUTI Gianluca
SIMONIN Mélissande

Group Number: **27**

Academic Year: 2022-2023

Contents

1 | Introduction

1.1. Description of the problem

The aim of this project is to design a database using different modeling techniques. In this report, we will focus on ER modeling, graph modeling, Documental approach, and spark technology. The data we use to realize the database come from the "DBLP" repository, which contains bibliographic records of publications in the computer science field. Each publication is written by one or more scientific authors, and can belong to a book, a Journal, or a Proceeding. Editors take care of book checking and proceeding organization, and both editors and authors can have a web page to promote their activities.
We imagine that a database containing this kind of information would be mainly used by scholars. For example, the database would allow them to discover publications by authors or journals that are relevant to them, or look for the most active and influential authors in the computer science field in general, including links to their personal web page. More specifically, a graph model would be particularly powerful to explore the network of citations between authors and publications, or the network between coauthors of papers. For this reasons, all of our queries are assumed to happen with this scenario in mind.

1.2. Assumptions

- Each person is identified by an ~~ID~~ orcid and a name. We assume a person to always use one exclusive name.
- Author and editor are the only types of person:
  - An author writes publications. To be in the database, he must have written at least one publication.
  - An editor checks books and organizes proceedings. He can check/organize one or several books/proceedings.
  - An author can be an editor, and an editor can be an author.

  - Author and editor can both have a web page to promote their work.
- A web page is identified by an ID, it has a title, a last modification date and a URL (ee). It can belong to one or more persons.
- A publication is identified by and ID, it has a title, a URL (ee), a last modification date, ~~a number~~, and a year of publication.
- A publication is written by at least one author.
- A publication can cite other publications and can be cited by other publications.
- Thesis, incollection, inproceeding and article are different types of publications:
  - A thesis is written in at least one school.
  - Phdthesis and masterthesis are the only types of thesis.
  - A Phdthesis has an isbn, and a serie.
  - An incollection is contained in one and only one book.
  - An inproceeding is held in one and only one proceeding.
  - An article belongs to one and only one journal.
- A book is identified by an ID, has a title, a serie, an isbn and a year of publication.
- A book contains at least one incollection, in a certain volume, and on a certain pages.
- A book is checked by zero, one, or several editors.
- A proceeding is identified by an ID, has an isbn and a URL (ee).
- A proceeding holds at least one inproceeding, on certain pages.
- A proceeding is published by at least one publisher.
- A proceeding can be organized by editors.
- A journal is identified by an ID and has a name.
- A journal contains at least one article, in a certain volume and on certain pages
- A school is identified by an ID, and has a name.
- In every school was written at least one thesis.

2 | ER Model

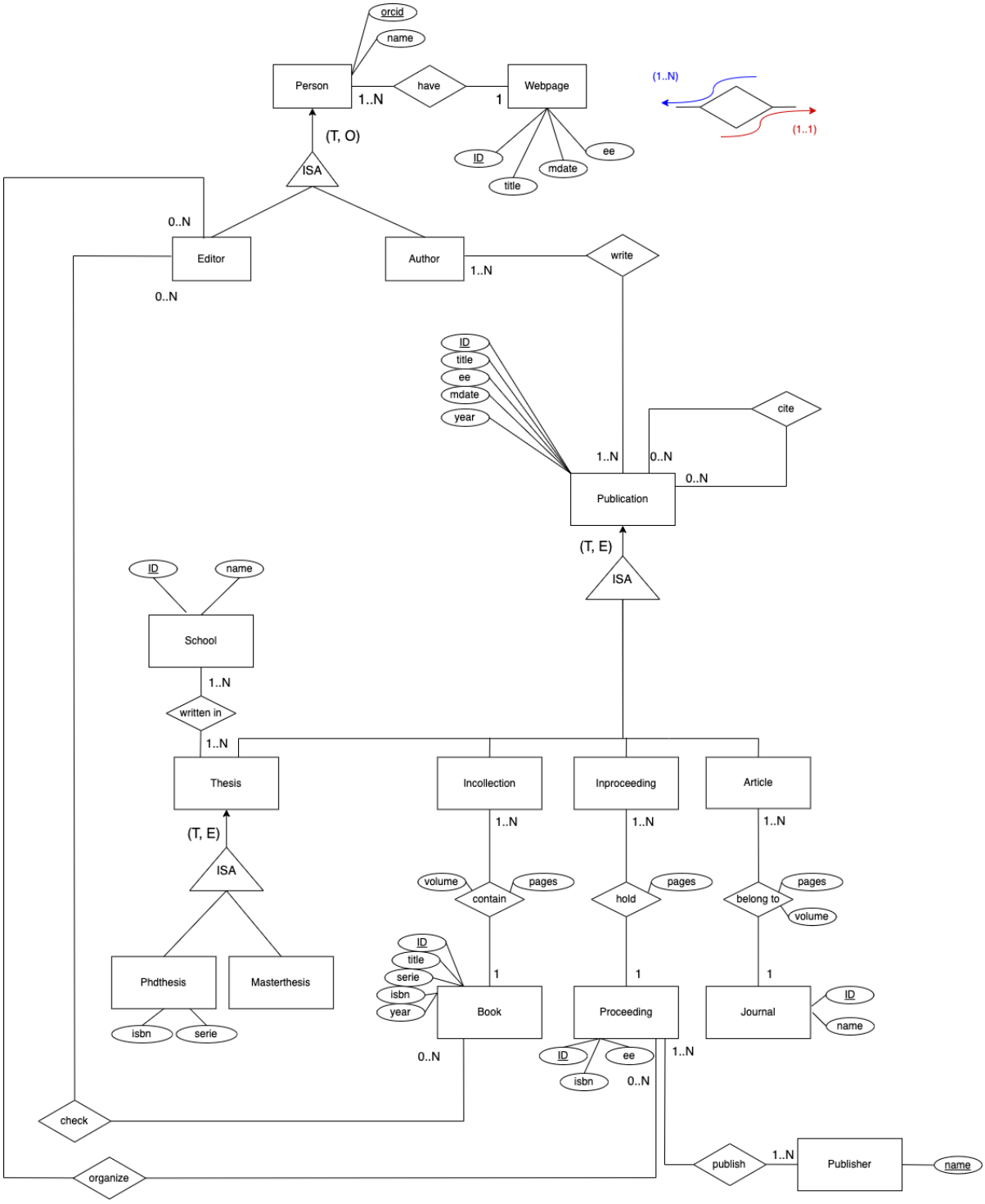
2.1. Identified entities and relationships

Entities:
Person, Author, Editor, Webpage, Publication, Thesis, Masterthesis, Phdthesis, School, Incollection, Inproceeding, Article, Journal, Book, Proceeding and Publisher.

Relationships:

- WRITE : Author - Publication
- HAVE : Person - Webpage
- CITE : Publication - Publication
- CHECK : Editor - Book
- WRITTEN IN : Thesis - School
- CONTAIN : Book - Incollection, with attributes "pages" and "volume"
- HOLD : Proceeding - Inproceeding , with attribute "pages"
- BELONG TO : Article - Journal , with attributes "pages", "volume"
- ORGANIZE: Editor - Proceeding
- PUBLISH: Publisher - Proceeding

2.2. Diagram


orcid
name
Person
have
1..N
1
Webpage
(1..N)
(1..1)
(T, O)
ISA
ID
title
ee
mdate
0..N
Editor
Author
write
1..N
0..N
ID
title
ee
mdate
year
cite
1..N
0..N
0..N
Publication
(T, E)
ISA
ID
name
School
1..N
written in
1..N
Thesis
Incollection
Inproceeding
Article
1..N
1..N
1..N
(T, E)
ISA
volume
pages
contain
hold
pages
belong to
pages
volume
Phdthesis
Masterthesis
ID
title
serie
isbn
year
Book
1
1
1
Proceeding
Journal
ID
name
isbn
serie
0..N
ID
ee
isbn
0..N
1..N
check
organize
publish
1..N
Publisher
name

3 | Dataset

3.1. Data description

The original data consists of over 10 million records of publications in the computer science field, coming from the DBLP repository. For computational reasons, we reduced the dataset, to a size of around 130k records via random sampling. There are 8 different types of records in the final dataset, each with relative fields. We only report the fields that are meaningful for our subsequent queries.

```
article = ID;author;author-bibtex;author-orcid;cite-label;crossref;editor;
          journal;key;mdate;number;ee;pages;title;volume;year

proceeding = ID;author;cite;cite-label;editor;editor-orcid;journal;key;mdate

inproceeding = ID;author;author-bibtex;author-orcid;cite;cite-label;crossref;
               editor;editor-orcid;key;mdate;month;note;note-type;number;ee;
               pages;title;volume;year

book = ID;author;author-bibtex;author-orcid;booktitle;cite;cite-label;crossref;
       editor;editor-orcid;key;mdate;pages;school;title;volume;year

incollection = ID;author;author-orcid;booktitle;cdrom;chapter:int;cite;cite-label;
               crossref;key;mdate;note;number;ee;pages;title;year

www = ID;author;author-bibtex;cite;crossref;editor;ee;key;mdate;title;
      url;url-type;year

phdthesis = ID;author;author-orcid;ee;key;mdate;number;pages;school;volume;year

mastersthesis = ID;author;key;mdate;school;title;year
```

3.2. Pre-processing and upload

In order to make the DBLP data usable to make queries in a graph database structure, we needed a pre-processing part before importing the data into Neo4J to create a new database.

XML to CSV

The original data was in a very large XML file containing all of the different record types together. In order to give it a more readable and flexible structure, the first step of the pre-processing involved converting the data to several csv files, one for each record type. This was achieved through a python script made for the purpose, which is publicly available on GitHub (https://github.com/ThomHurks/dblp-to-csv).

Data sampling

The large size of the DBLP data (>10mln records) made it infeasible to store and query a database on it using a regular PC. For this reason, we shrunk the dataset to a fraction of it, striving to not alter the characteristics of the original dataset.
To this aim, we sampled the data in such a way that every book and proceeding listed in the dataset would also have all of the publications contained in it, and that no publication would be inserted without its relative book or proceeding.
To achieve this, we implemented a python script where books and proceedings would be randomly sampled, and then include all of their respective publications. The sample sizes were chosen such that they would be proportional to the relative occurrence of books and proceedings in the original dataset. In particular, the original dataset contained around 19k books and 52k proceedings, amounting to 71k records, 73% of which were proceedings. We aimed to reduce this number to 1000 records, so we sampled 730 proceedings and 270 books. We then retrieved all the publications relative to them, and proceeded to sample a number of journal articles, Phd thesis and websites in a similar way.
After this process, the data consisted of around 130k records.

Import into Neo4J

We then proceeded to upload the reduced csv files on a new Neo4J database instance. One way to achieve this is to use the "neo4j-admin import" shell command, which allows to import also large amounts of data. We report the command used to generate the database in the following lines:

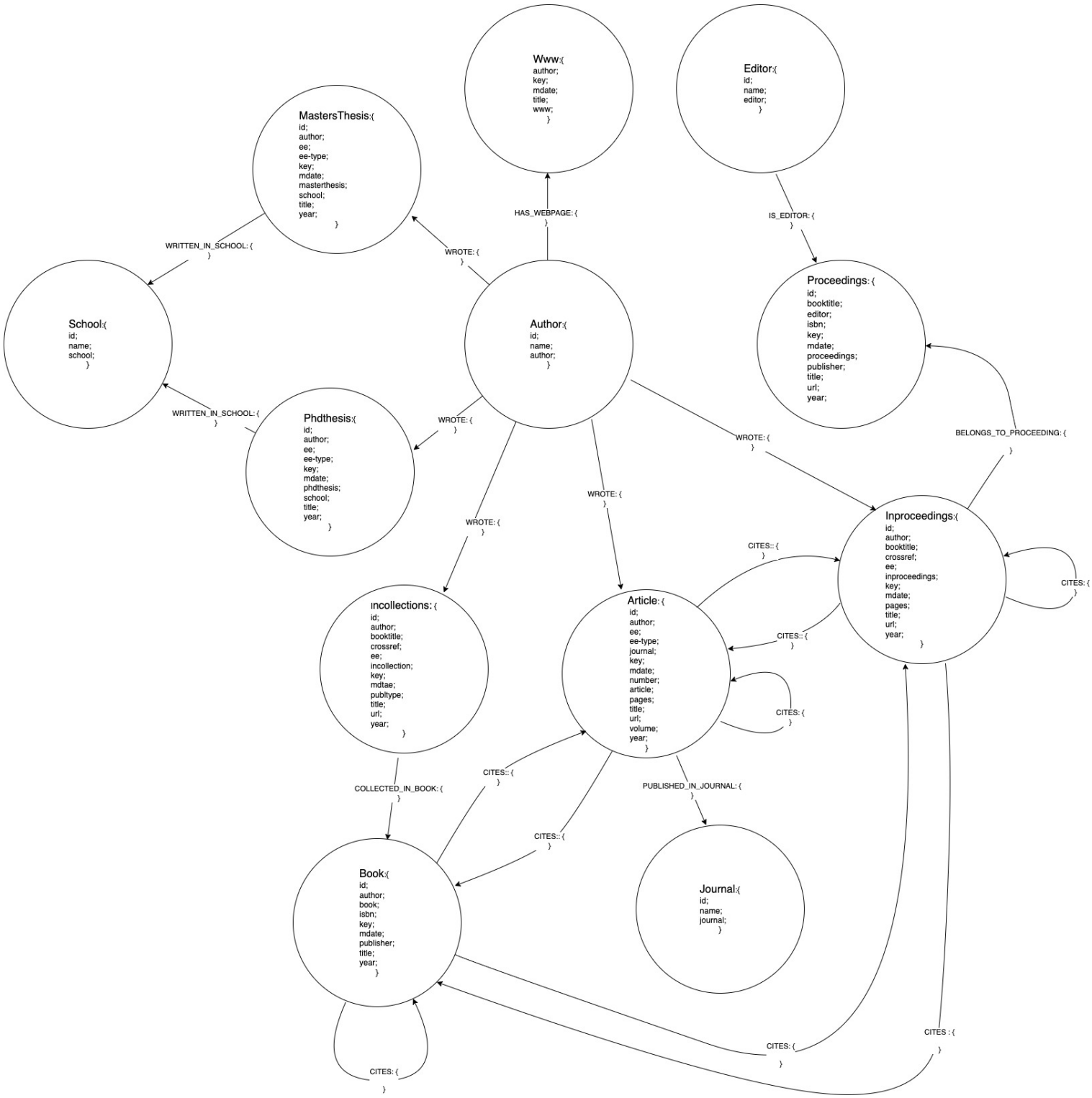
```
bin\neo4j-admin import --database=SMBUD-proj
    --delimiter ";" --array-delimiter "|"
    --id-type INTEGER
    --nodes=inproceedings="output_inproceedings_header.csv,
inproceedings_small.csv"
    --nodes=www="output_www_header.csv,www_small.csv"
    --nodes=article="output_article_header.csv,article_small.csv"
    --nodes=book="output_book_header.csv,book_small.csv"
    --nodes=masterstheis="output_mastersthesis_header.csv,
output_mastersthesis.csv"
    --nodes=phdthesis="output_phdthesis_header.csv,phd_small.csv"
    --nodes=proceedings="output_proceedings_header.csv,
proceedings_small.csv"
    --nodes=incollection="output_incollection_header.csv,
incollection_small.csv"
```

This command creates nodes of one type for every csv file, each with a specified label name and attributes corresponding to the content of the header files. There is still no relationship between nodes. The very first queries to the db are devoted to the creation of additional entities and relationships.

4 | Graph Diagram

In this chapter you can see the Graph diagram structure of the database. Author, Editor, Www, School, MastersThesis, PhdThesis, Proceedings, Inproceedings, Incollections, Article, Book and Journal are represented as nodes and they have the same attributes shown in the ER model. The relationships of the database are WROTE, CITES, HAS_WEBPAGE, IS_EDITOR, WRITTEN_IN_SCHOOL, COLLECTED_IN_BOOK, PUBLISHED_IN_JOURNAL and BELONGS_TO_PROCEEDING where:

- WROTE, represents the relationship between an author and the publications that he made
- CITES, refers to the fact that in some publications can be cited other ones
- HAS_WEBPAGE, relates to the possibility of having authors who own a web page
- IS_EDITOR, links proceedings with their editors
- WRITTEN_IN_SCHOOL, links mastersthesis and phdthesis with the school where they have been made
- COLLECTED_IN_BOOK, refers to incollections contained in a book
- PUBLISHED_IN_JOURNAL, relates to articles contained in a journal
- BELONGS_TO_PROCEEDING, gives us the relationship between an inproceeding and the proceeding to which it refers


Www:{
author;
key;
mdate;
title;
www;
}
Editor:{
id;
name;
editor;
}
MastersThesis:{
id;
author;
ee;
ee-type;
key;
mdate;
masterthesis;
school;
title;
year;
}
HAS_WEBPAGE: {
}
IS_EDITOR: {
}
WRITTEN_IN_SCHOOL: {
}
WROTE: {
}
School:{
id;
name;
school;
}
Author:{
id;
name;
author;
}
Proceedings: {
id;
booktitle;
editor;
isbn;
key;
mdate;
proceedings;
publisher;
title;
url;
year;
WRITTEN_IN_SCHOOL: {
}
Phdthesis:{
id;
author;
ee;
ee-type;
key;
mdate;
phdthesis;
school;
title;
year;
}
WROTE: {
}
WROTE: {
}
BELONGS_TO_PROCEEDING: {
}
WROTE: {
}
WROTE: {
}
Inproceedings:{
id;
author;
booktitle;
crossref;
ee;
inproceedings;
key;
mdate;
pages;
title;
url;
year;
}
CITES:: {
}
CITES: {
}
incollections: {
id;
author;
booktitle;
crossref;
ee;
incollection;
key;
mdtae;
publtype;
title;
url;
year;
}
Article: {
id;
author;
ee;
ee-type;
journal;
key;
mdate;
number;
article;
pages;
title;
url;
volume;
year;
}
CITES:: {
}
CITES: {
}
CITES:: {
}
COLLECTED_IN_BOOK: {
}
PUBLISHED_IN_JOURNAL: {
}
CITES:: {
}
CITES:: {
}
Book:{
id;
author;
book;
isbn;
key;
mdate;
publisher;
title;
year;
}
Journal:{
id;
name;
journal;
}
CITES: {
}
CITES : {
}
CITES: {
}

5 | Neo4j Queries

5.1. Creation & Update commands

Starting from the imported data, the first database queries are oriented to the creation of additional nodes and relationships between nodes, with the aim of making the database more consistent with a graph modelling approach such as the one Neo4J offers. This also eases subsequent queries made through the Cypher language.

1. Author nodes.
As a first step, we create a node for every author that appears in our dataset. A list of author names is contained as a field in most of the publication types, as well as in books and websites. For the latter, we will later rename the relationship to HAS_WEBSITE instead of WROTE.

```
//author creation
MATCH (n) WHERE n.author IS NOT NULL
UNWIND n.author as authorname
MERGE (author:person:author{name:authorname})
CREATE (author)-[r:WROTE]->(n)
SET r.year = n.year
```

This query finds every type of node that has an author field, which is an array of strings containing the author names, "unwinds" the array, considering each name separately, then creates new nodes without duplicates based on the author names. At this point, the query also creates a WROTE relationship, where every author is connected to their work. This relationship is also added the publication year as an attribute.

2. Editor nodes.
We now proceed to do the same with editor nodes. Editors are present as fields inside "proceedings" type nodes. We want to extract them as nodes and create a relationship between the editor and the proceedings they organized instead.

```
//IS_EDITOR creation
MATCH (n:proceedings) WHERE n.editor IS NOT NULL
UNWIND n.editor as editorname
MERGE (editor:person{name:editorname})
SET editor:editor
CREATE (editor)-[:IS_EDITOR]->(n)
```

This query works in a similar way to the author creation one, with the difference that we need to add the additional "editor" label after the MERGE clause (as opposed to inside it), in order to avoid creating an additional node when there was already a person (possibly an author) with the same name.

3. "incollection" and book nodes.
In our data, "incollection" nodes represent papers that are published inside a book. Every "incollection" node is related to its book via a "crossref" attribute that holds the foreign key corresponding to a book record. In order to make this more coherent with a graph structure, we add a BELONGS_TO_BOOK relationship between incollection and book to represent this behaviour.

```
//BELONGS_TO_BOOK creation
MATCH (n:incollection) where n.crossref is not null
WITH n.crossref AS book,n.pages as pages n as n
MATCH (b:book {key:book})
CREATE (n)-[:BELONGS_TO_BOOK{pages:pages}]->(b)
```

The query first retrieves all of the "crossref" fields of "incollection" records from the database, then uses them to find the node(s) with a "book" label that have a matching result inside the "key" attribute. A BELONGS_TO_BOOK relationship is created between any such couple of nodes.

4. Article and journal nodes.
Article records contain a "journal" field which represents the journal they were published in. We want to create journal nodes based on this, and then connect with a relationship any article was published inside a specific journal.

```
//journal nodes and PUBLISHED_IN_JOURNAL creation
MATCH(n:article) WHERE n.journal IS NOT NULL
WITH n.journal as jourName, n.volume AS volume,
n.number AS number, n.pages as pages, n AS n
```

```
MERGE (j:journal{name:jourName})
CREATE (n)-[r:PUBLISHED_IN_JOURNAL{volume:volume,number:number,pages:pages}]->(j)
```

The query retrieves all the journal fields from articles, creates journal nodes with that name as only attribute, without duplicates, and finally creates a PUBLISHED_IN_JOURNAL relationship between an article and its journal.

5. School nodes.
Some nodes of type "phdthesis" or "mastersthesis" have a "school" field which represents the university they were written in. This should be rather represented as a relationship between a school node and the publication that was written in it.

```
// school nodes creation
MATCH (n)
WHERE n.school IS NOT NULL
MERGE (:school{name:n.school})
```

6. CITES relationship.
Finally, some of the publication records have a "cite" field which contains "keys" pointing to other records that are cited in it. This can happen two records of any type, so we want to create a relationship to reflect this which is not limited to specific node labels.

```
// CITES relation
MATCH (citing) WHERE citing.cite IS NOT NULL
UNWIND citing.cite as cited_key
MATCH (cited {key:cited_key})
CREATE (citing)-[:CITES]->(cited)
```

The query once again compares the entries of any non-null "cite" field (which is an array of keys) inside the "citing", with the key field in another record,the "cited", of any type. A successful match of these two results in the creation of a CITES relationship between the "citing" and the "cited" nodes.

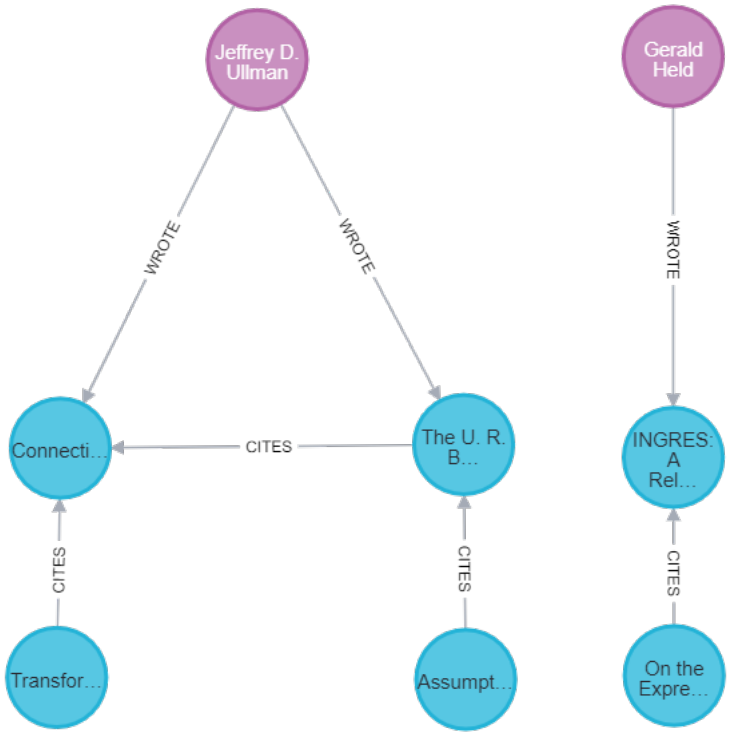
5.2. Queries

1. Publications citing coauthors.
Suppose we are a researcher that wants to use the database to find out in how much their coauthors were influential in other papers besides the one they collaborated on. In the next example, we suppose that the author "David Maier 0001" queries the database to

achieve this.

```
// list of publications that cite one of your coauthors
MATCH (n:author{name:"David Maier 0001"})-[:WROTE]->(p),
      (p)<-[:WROTE]-(coauthor:author)
MATCH (coauthor)-[:WROTE]->(byCo)
WHERE NOT "David Maier 0001" IN byCo.author
MATCH (citesCo)-[:CITES]->(byCo)
RETURN citesCo, byCo, coauthor
```

OUTPUT:



This query retrieves all of the coauthors of "David Maier 0001", then accesses all of their publications through the :WROTE relationship, and finally looks for all of the nodes that have a :CITE relationship with these publications. In the image, nodes in blue are publications and nodes in purple are authors. In this case we see that the 2 coauthors of "David Maier 0001" are cited by 3 external papers.

2. Find persons who wrote together at least one article and were editors together for at least one proceedings.

```
MATCH (p1:person)-[:IS_EDITOR]->(pro:proceedings),
      (p2:person)-[:IS_EDITOR]->(pro),
      (p1)-[:WROTE]->(a:article),
      (p2)-[:WROTE]->(a)
RETURN a AS article, pro AS proceeding, collect(DISTINCT p1)
```

This query returns, for every couple (article, proceeding), the list of persons that were together authors of the article and editors of the proceeding.

"article"	"proceeding"	"collect(distinct p1)"
{"title":"Art_1"}	{"title":"Pro_1"}	[{"name":"P_3"},{"name":"P_2"}, {"name":"P_1"}]
{"title":"Art_2"}	{"title":"Pro_2"}	[{"name":"P_4"},{"name":"P_5"}]
{"title":"Art_5"}	{"title":"Pro_4"}	[{"name":"P_8"},{"name":"P_9"}]
{"title":"Art_4"}	{"title":"Pro_4"}	[{"name":"P_8"},{"name":"P_9"}]

3. Find the author with the greatest number of cocoauthors

```
MATCH (f:person)-[:WROTE]->(a:article)<-[:WROTE]-(coauthor:person),
      (coauthor)-[:WROTE]->(a2:article)<-[:WROTE]-(cocoauthor:person)
WHERE (NOT (f)-[:WROTE]->(:article)<-[:WROTE]-(cocoauthor))
RETURN f AS author, count(*) AS cocoCount
ORDER BY cocoCount DESC
LIMIT 1
```

The query returns the author with the highest number of coco-authors (authors that wrote a publication with his coauthors but not with him), and their actual number cocoCount

"author"	"cocoCount"
{"name":"Zhu Han 0001"}	718

4. Find persons that wrote an article citing one master Thesis and one Phd Thesis written in the same school

```
MATCH (p:person)-[:WROTE]->(a:article),
      (a)-[:CITES]->(mt:mastersthesis),
      (a)-[:CITES]->(phdt:phdthesis),
      (mt)-[:WRITTEN_IN]->(:school)<-[:WRITTEN_IN]-(phdt)
RETURN DISTINCT p AS Author
```

"Author"
{"name":"Author001"}
{"name":"Author002"}

5. A query that returns the name, the webpage and the phd thesis publication year of the most recent author who made both master thesis and phd thesis in the same school

```
MATCH (phd:phdthesis)-[wphd:WRITTEN_IN_SCHOOL]->(s1:school),
      (master:mastersthesis)-[wmas:WRITTEN_IN_SCHOOL]->(s2:school),
      (a: author)-[hw:HAS_WEBPAGE]->(w:www),
WHERE phd.author = master.author
AND   s1.name = s2.name
AND   a.name = phd.author
WITH  phd, s1, master, s2, a, w
ORDER BY phd.year DESC
LIMIT 1
RETURN  phd.author AS authorName,
        w.key AS webpage,
        s1.name AS schoolName,
        master.year AS masterYear,
        phd.year AS phdYear
```

This query retrieves all of the authors that made both master thesis and phd thesis in the same school, then check if these authors have own webpages and using ORDER_BY ... DESC sort the authors in descending order based on the year of phd realization. In

the end, with LIMIT 1 the query returns only one author who is the most recent that satisfies the condition.

"authorName"	"webpage"	"schoolName"	"masterYear"	"phdYear"
"Yan-Jing Wu"	"homepages/57/8473"	"Massachusetts Institute of Technology, Cambridge, USA"	"2016"	"2022"

6. Filtering cited articles on the same inproceeding by last modification date

```
MATCH (citing:inproceedings)-[r:CITES]->(cited: article),
      (inPro:inproceedings)-[:BELONGS_TO_PROCEEDING]->(pro:proceedings)
WHERE cited.mdate < "2022" and cited.mdate > "2017"
AND   citing.title = inPro.title
WITH  citing, cited, inPro, pro
ORDER BY cited.mdate DESC
RETURN  cited.author AS ArticleAuthor,
        citing.title AS InproceedingTitle,
        cited.mdate AS ArticleLastModification,
        pro.title AS Proceeding
```

This query returns articles, their authors, their mdates and the name of the proceeding that contains the inproceeding where these articles are cited, whose date of last modification is between 2018 and 2021(included). The output is given on a descending order based on the last modification date of the cited articles.

"ArticleAuthor"	"InproceedingTitle"	"ArticleLastModification"	"Proceeding"
"Marco A. Casanova"	"A Theory of Data Dependencies over Relational Expressions."	"2021"	"European Data Conference"

"Franco Zambonelli"	"A Theory of Data Dependencies over Relational Expressions."	"2021"	"European Data Conference"
"Anthony C. Klug"	"A Theory of Data Dependencies over Relational Expressions."	"2020"	"European Data Conference"
"Werner Retschitzegger"	"A Theory of Data Dependencies over Relational Expressions."	"2019"	"European Data Conference"
"Mark A. Peri"	"A Theory of Data Dependencies over Relational Expressions."	"2018"	"European Data Conference"

7. Find the number of articles written and proceedings edited by persons that have done both activities in 2022 (=both numbers must be at least 1)

```
MATCH (p:person)-[:WROTE]->(a:article),
      (p)-[:IS_EDITOR]->(pro:proceedings)
WHERE a.year = 2022 and p.year = 2022
RETURN p.name, count(DISTINCT a), count(DISTINCT pro)
```

"p.name"	"count(distinct a)"	"count (DISTINCT pro)"
"Kurt P. Brown"	16	3
"Rita Ley"	7	2

8. Returns all editors that a determined author has worked with and the

percentage of working with each one of them with respect to the number of inproceedings of that author

```
MATCH (a:author)-[r:WROTE]->(i:inproceedings)
      -[b:BELONGS_TO_PROCEEDING]->(p:proceedings)
      <-[o:IS_EDITOR]-(e:editor)
WHERE a.name = "Cristiano Tolomei"
WITH collect(distinct e) AS ed, count(e) AS numEdit
UNWIND ed AS edit
RETURN distinct(edit.name) AS EditorName,
       count(edit.name) AS NumOfInproceeding,
       toString(round((1.0 * (count(edit.name)) / numEdit) * 100, 1))
       + "%" AS Percentage
ORDER BY Percentage DESC
```

This query search all the possible concatenated relations between a specified author (in this case "Cristiano Tolomei") and all editors he has worked with. After that it returns a table with for each row the name of one of the editors, the number of inproceedings that the author have made for proceedings organized by that editor and the percentage of inproceedings made for that specific editor with respect to the total number of inproceeding made by that author. The percentage is computed by dividing the number of inproceedings made for an editor for the total number of inproceedings made by the author and multiplying the result for 100.

"EditorName"	"NumOfInproceeding"	"Percentage"
"Editor3"	3	"50.0%"
"Editor1"	2	"33.3%"
"Editor2"	1	"16.7%"

9. Returns all phdthesis that have been modified in the same school in the same year

```
MATCH (t:phdthesis)-[r:WRITTEN_IN_SCHOOL]->(s)
WITH [item IN split(t.mdate, "/") | toInteger(item)] AS parts,
     t AS thesis,
```

```
     s AS school
WHERE "University of Siegen" IN school.name
AND toString(date({day: parts[0], month: parts[1],year: parts[2]}).year)
    = "2021"
RETURN thesis.title AS Title, thesis.mdate AS ModDate
```

This query returns all the phdthesis that has been modified for the last time in the same school in a determined year. For this example we used phdthesis modified in 2021 in the "University of Siegen". Since the attribute mdate, which represent the date of the last modification is in string format, we decided to use a comprehension to extract from it single elements (day, month and year) in order to create a date in cypher date format and compare the year with the given one. At the end we return the titles of modified phdthesis and the corresponding modification dates.

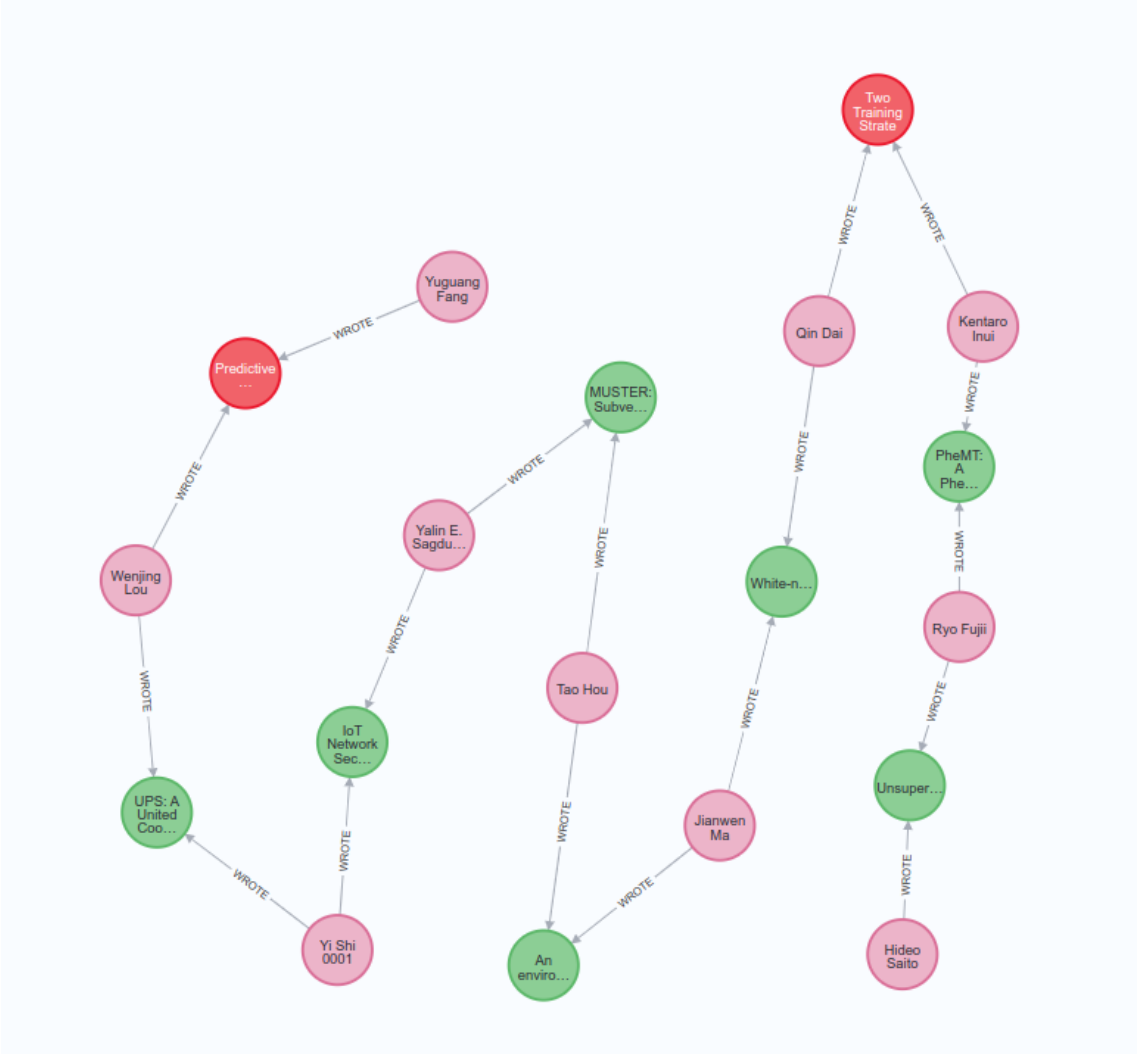
"Title"	"ModDate"
"Web Services basierte Referenzarchitektur für Enterprise Application Integration."	"17/07/2021"
"Technology appropriation in transnational networks of social activists: a study of the European Social Forum."	"17/07/2021"
"Analyzing and influencing search engine results: business and technology impacts on web information retrieval."	"17/07/2021"
"An efficient inflation method for segmentation of medical 3D images."	"17/07/2021"
"Integration serieller Feldbussysteme in hochdynamische Regelkreise."	"17/07/2021"

10. Find shortest path between two persons

```
match (p1:person {name: "Hideo Saito"}),
      (p2:person {name: "Yuguang Fang"}),
      p = shortestPath((p1)-[*]-(p2))
```

```
return p
```

The query returns the shortest path between Hideo Saito and Yuguang Fang, through every relationship.



6 | Document-oriented Database

6.1. Description of the document structure

We decided to implement a documental database with only one type of document, which represents a Publication. Here there are the attributes and sub documents we have decided to add:

- **title** (String) the title of the publication
- **isbn** (String) the ISBN number of the publication
- **abstract** (String) the abstract of the publication
- **authors** array containing author sub-documents, consisting of:
  - **name** (String) full name of the author
  - **title** (String) study title of the author: either "Ing", "Dr" or "Phd"
  - **email** (String)
  - **affiliation** (String) the organization where the author do his/her research activity
  - **bio** (String) a short biography of the author
  - **avatar** (String) a URL to the author's avatar image
  - **streetAddress** (String) the address of the author's office
  - **countryCode** (String) country code of the nationality of the author (eg "IT", "US", "FR"...)
  - **bitcoinAddress** (String) bitcoin address of the author for donations
  - **ethereumAddress** (String) ethereum address of the author for donations
  - **gender** (String) gender of the author, either "M" for male, or "F" for female.

- **keywords** array of Strings that are keywords for the publication
- **journal** (String) name of the journal which the publication belongs to
- **volume** (Int32) volume of the journal the publication belongs to
- **number** (Int32) number of the journal the publication belongs to
- **publicationDate** (Date) the date on which the article was published
- **modificationDate** (Date) the last modification date of the record
- **pages** (Int32) number of pages of the article
- **sections** array containing one sub-document for every section consisting of:
  - **title** (String) title of the section
  - **text** (String) introductory text of the section
  - **subsections** array containing one sub-document for every sub-section of the current section, consisting of:
    - **title** (String) title of the sub-section
    - **text** (String) part of the text belonging to the subsection
  - **images** array containing one image sub-document for every image of the current sectiom, consisting of:
    - **url** (String) URL endpoint to the image
    - **caption** (String) caption of the image
- **bibliography** array of objects ids, which are references to publications in the bibliography of the current publication

6.2. Sample document


```
{
   "_id":{"$oid":"637bb9d894b847a5ac78beb7"},
   "title":"Cras pellentesque volutpat dui.",
   "isbn":"340616975-9",
   "abstract":"In hac habitasse platea dictumst. Morbi vestibulum, velit id pretium iaculis, diam erat
↪  fermentum justo, nec condimentum neque sapien placerat ante. Nulla justo. [...]",
   "authors":[
      {
         "name":"Nani Barwick",
         "title":"Phd",
         "email":"nbarwick0@omniture.com",
```

```
            "countryCode":"ES",
            "affiliation":"La Salle Universities - International Programmes",
            "bio":"vestibulum proin eu mi nulla ac enim in tempor turpis nec euismod scelerisque [...]",
            "avatar":"https://robohash.org/dolorumvoluptateconsequatur.png?size=50x50&set=set1",
            "streetAddress":"9026 Fair Oaks Plaza",
            "bitcoinAddress":null,
            "ethereumAddress":null,
            "gender":"F"
         },
         [...]
      ],
      "keywords":["amet", "felis", "blandit", "proin", "neque", "libero"],
      "journal":"aenean auctor",
      "volume":27,
      "number":10,
      "publicationDate":{"$date":"2008-05-04T14:02:19.000Z"},
      "modificationDate":{"$date":"2019-06-15T09:31:47.000Z"},
      "pages":282,
      "sections":[
         {
            "title":"ridiculus mus vivamus vestibulum sagittis sapien cum sociis",
            "text":"Morbi a ipsum. [...]",
            "subsections":[
               {
                  "title":"morbi vestibulum velit",
                  "text":"ipsum ac tellus semper interdum mauris ullamcorper purus  [.,.]"
               },
               {
                  "title":"vel augue vestibulum rutrum rutrum neque aenean auctor",
                  "text":"ipsum primis in faucibus orci luctus et ultrices posuere [...]"
               },
               [...]
            ],
            "images":[
               {
                  "url":"http://dummyimage.com/104x183.png/dddddd/000000",
                  "caption":"Nulla tellus."
               }
               [...]
            ]
         },
         [...]
      ],
      "bibliography":[
         {"$oid":"637bb9d894b847a5ac78bf38"},
         {"$oid":"637bb9d894b847a5ac78c0c1"},
         [...]
      ]
}
```

6.3. ER Diagram

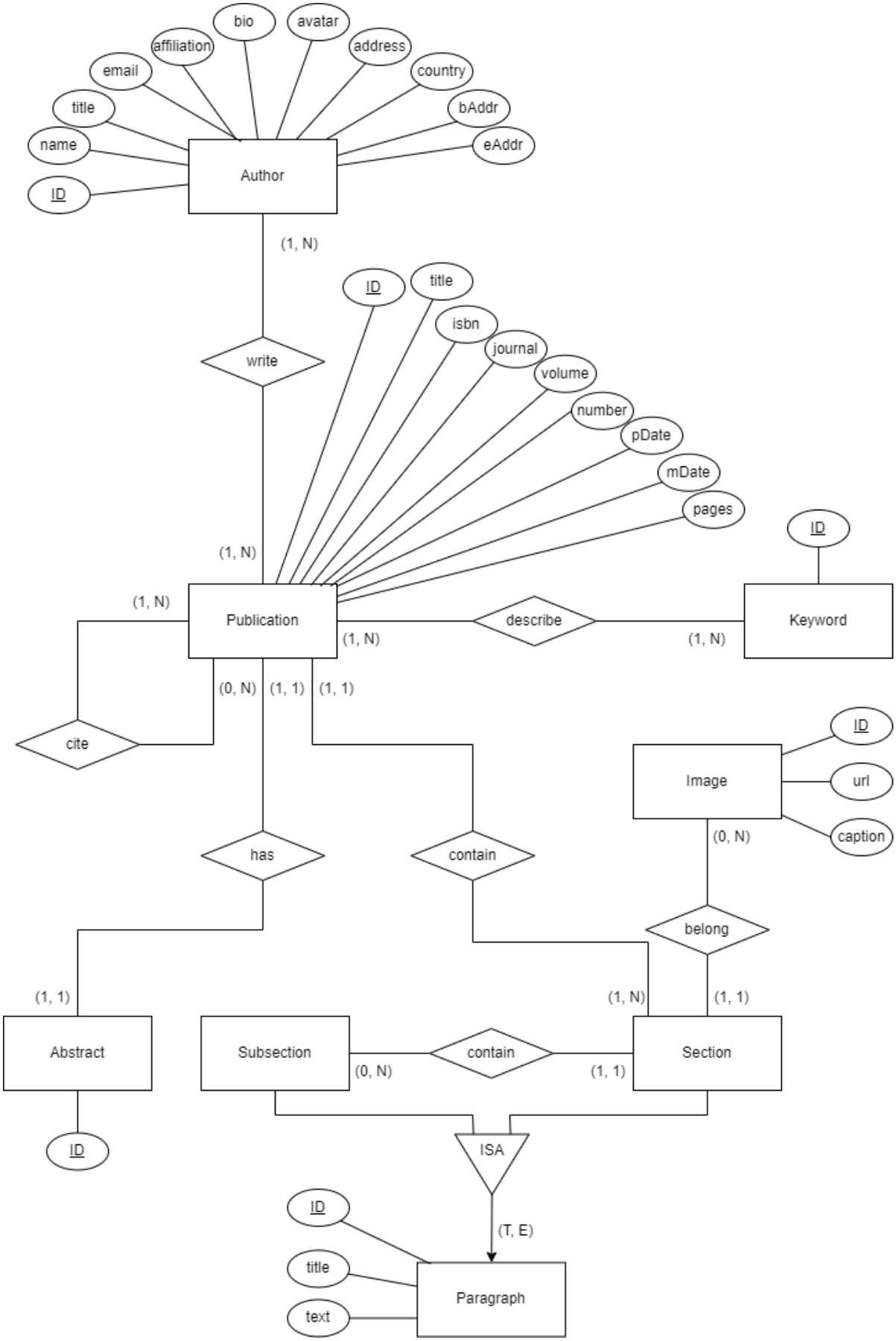
In order to allow a better comprehension of the dataset we have generated, we decided also to provide an ER diagram that can visually explain the way in which different elements of our dataset are related.

Entities:
Author, Publication, Keyword, Image, Abstract, Section, Subsection and Paragraph.

Relationships:

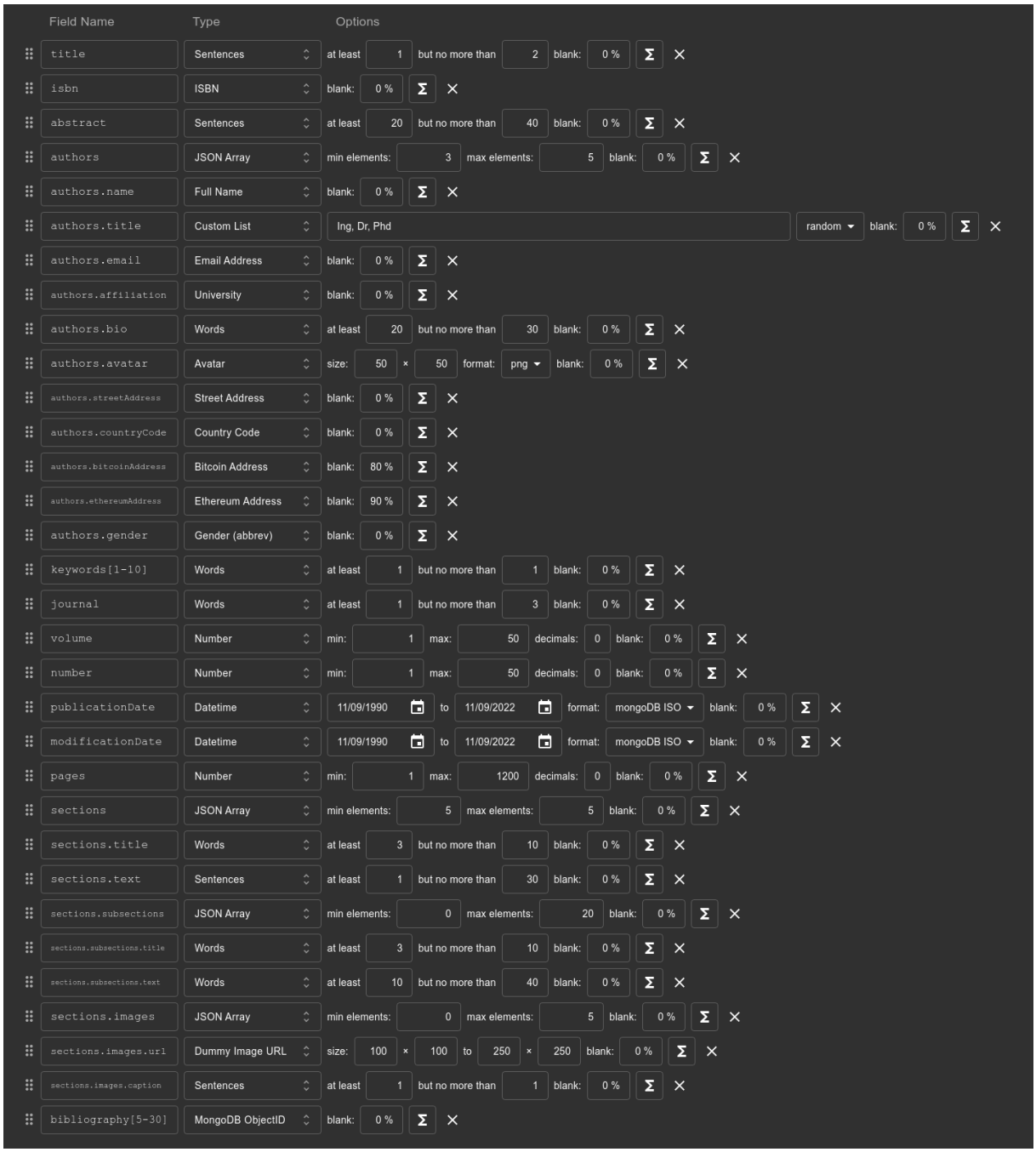
- WRITE : this relation explains the fact that every author has written at least one publication (otherwise they won't be in the dataset) and every publication has been written by at least one author.
- DESCRIBE : all keywords describe at least one publication (otherwise they won't be in the dataset) and all publications are described by at least one keyword.
- HAS : every publication has one and only one abstract.
- BELONG : every image belongs to one specific section and each section can contain from zero to many images.
- CITE : this relation represents the bibliography in our dataset. We assume that every publication cites at least one other publication which is in its bibliography. On the other hand, every publication can be cited by zero or several other publications and be in their bibliography.
- CONTAIN : this relation appears two times since every publication contains one or more sections and each section can contain from zero to many subsections. On the other direction, every subsection is related to only one section, and every section is related only to a specific publication.

In our scheme, both Subsection and Section entities inherit from Paragraph entity. This happens because in the dataset we generated, both subsections and sections have a title and a text, but a section could also contain subsections that are an addition to his text but not a part of it.


bio
avatar
affiliation
address
email
country
title
bAddr
name
eAddr
Author
ID
(1, N)
ID
title
isbn
journal
volume
number
pDate
mDate
pages
write
(1, N)
(1, N)
Publication
describe
Keyword
ID
(1, N)
(1, N)
(0, N)
(1, 1)
(1, 1)
cite
ID
Image
url
caption
has
contain
(0, N)
belong
(1, 1)
(1, N)
(1, 1)
Abstract
Subsection
contain
Section
(0, N)
(1, 1)
ID
ISA
ID
(T, E)
title
Paragraph
text

6.4. Data generation and upload

We generated the data with the tool available at `https://www.mockaroo.com/`, that allows to generate fake data based on a specific schema defined by the user. Multiple different basic types are available (e.g. Full Name, University, Words, Street address, ...). This is how we have set the schema in order to generate documents that fit the structure definde above:



7 | MongoDB Queries

7.1. Creation Update commands

Starting from the imported data, the first database queries are oriented to the creation of additional documents with the aim of making the database more consistent with a document-oriented approach such as the one MongoDB offers. This also eases subsequent queries made through Mongo.

1. New document

The first creation query we do refers to the use of the **"insertOne()"** method: we add a new publication whose authors are the student of this working group. The publication fields concerned in this query are title, authors, keywords, publicationDate, modificationDate and pages. The authors fields considered are name, title, countryCode and gender of the student.

```
db.publications.insertOne({
        title: "Systems and Methods for Big and Unstructured Data project",
        authors: [{
                name: "Gianluca Pizzuti",
                title: "student",
                countryCode: "IT",
                gender: "M"
        }, {
                name: "Melissande Simonin",
                title: "student",
                countryCode: "FR",
                gender: "F"
        }, {
                name: "Enrico Brunetti",
                title: "student",
                countryCode: "IT",
```

```
                gender: "M"
        }, {
                name: "Emanuele Paesano",
                title: "student",
                countryCode: "IT",
                gender: "M"
        }, {
                name: "Pietro Lodi Rizzini",
                title: "student",
                countryCode: "IT",
                gender: "M"
        }],
        keywords: ["Big", "Data"],
        publicationDate: ISODate('2022-11-01T05:17:05.350Z'),
        modificationDate: ISODate('2022-11-13T10:20:00.500Z'),
        pages: 100
})
```

```
{ acknowledged: true,
  insertedId: ObjectId("63762190a67a2ca08dca4394") }
```

```
>_MONGOSH
> db.publications.find({_id: ObjectId("63762190a67a2ca08dca4394")})
< { _id: ObjectId("63762190a67a2ca08dca4394"),
    title: 'Systems and Methods for Big and Unstructured Data project',
    authors:
     [ { name: 'Gianluca Pizzuti',
         title: 'student',
         countryCode: 'IT',
         gender: 'M' },
       { name: 'Melissande Simonin',
         title: 'student',
         countryCode: 'FR',
         gender: 'F' },
       { name: 'Enrico Brunetti',
         title: 'student',
         countryCode: 'IT',
         gender: 'M' },
       { name: 'Emanuele Paesano',
         title: 'student',
         countryCode: 'IT',
         gender: 'M' },
       { name: 'Pietro Lodi Rizzini',
         title: 'student',
         countryCode: 'IT',
         gender: 'M' } ],
    keywords: [ 'Big', 'Data' ],
    publicationDate: 2022-11-01T05:17:05.350Z,
    modificationDate: 2022-11-13T10:20:00.500Z,
    pages: 100 }
```

2. Creation of two documents with same bibliography

We now proceed to use the **"insertMany()"** method: we create two publications that share in their bibliographies the same reference to the document created in the previous query. To do this we pass the **"_id"** of the publication as a reference.

```
db.publications.insertMany([{
        title: "SMBUD part 1",
        abstract: "ER model, Neo4j queries..",
        authors: [{
                name: "Cristal Oxlade",
                title: "Ing",
```

```
                email: "coxlade2@goodreads.com",
                countryCode: "RU",
                affiliation: "Siberian State Medical University",
                avatar: "https://robohash.org/etmollitiaautem.png?size=50x50",
                bitcoinAddress: "1AMDaHr3wGgrjE6HdExF4Bu1No6wkUP6Rt",
                gender: "F"
        }],
        publicationDate: ISODate('2022-11-02T11:10:35.000Z),
        modificationDate: ISODate('2022-11-02T11:15:35.000Z),
        pages: 103,
        bibliography: [ObjectId("63762190a67a2ca08dca4394")]
        }, {
        title: "SMBUD part 2",
        abstract: "MongoDB queries..",
        authors: [{
                name: "Barry Matlock",
                title: "Dr",
                email: "bmatlock1@bing.com",
                countryCode: "CN",
                affiliation: "Inner Mongolia Normal University",
                avatar: "https://robohash.org/nemoautqui.png?size=50x50",
                bitcoinAddress: "1KPgsVqtjywLS3G7uBXnApSbrchscpfzsn",
                gender: "M"
        }],
        publicationDate: ISODate('2022-11-15T16:10:35.000Z'),
        modificationDate: ISODate('2022-11-16T11:10:35.000Z'),
        pages: 88,
        bibliography: [ObjectId("63762190a67a2ca08dca4394")]
}])
```

```
{ acknowledged: true,
  insertedIds:
   { '0': ObjectId("63763e65a67a2ca08dca4396"),
     '1': ObjectId("63763e65a67a2ca08dca4397") } }
```

```
> db.publications.find({bibliography: {$in: [ObjectId('63762190a67a2ca08dca4394')]}})
< { _id: ObjectId("63763e65a67a2ca08dca4396"),
    title: 'SMBUD part 1',
    abstract: 'ER model, Neo4j queries..',
    authors:
     [ { name: 'Cristal Oxlade',
         title: 'Ing',
         email: 'coxlade2@goodreads.com',
         countryCode: 'RU',
         affiliation: 'Siberian State Medical University',
         avatar: 'https://robohash.org/etmollitiaautem.png?size=50x50&set=set1',
         bitcoinAddress: '1AMDaHr3wGgrjE6HdExF4Bu1No6wkUP6Rt',
         gender: 'F' } ],
    publicationDate: 2022-11-02T11:10:35.000Z,
    modificationDate: 2022-11-02T11:15:35.000Z,
    bibliography: [ ObjectId("63762190a67a2ca08dca4394") ] }
  { _id: ObjectId("63763e65a67a2ca08dca4397"),
    title: 'SMBUD part 2',
    abstract: 'MongoDB queries..',
    authors:
     [ { name: 'Barry Matlock',
         title: 'Dr',
         email: 'bmatlock1@bing.com',
         countryCode: 'CN',
         affiliation: 'Inner Mongolia Normal University',
         avatar: 'https://robohash.org/nemoautqui.png?size=50x50&set=set1',
         bitcoinAddress: '1KPgsVqtjywLS3G7uBXnApSbrchscpfzsn',
         gender: 'M' } ],
    publicationDate: 2022-11-15T16:10:35.000Z,
    modificationDate: 2022-11-16T11:10:35.000Z,
    bibliography: [ ObjectId("63762190a67a2ca08dca4394") ] }
```

To verify the success of the query we do a search on the entire dataset using the **"find()"** method: we use the **$in** comparison query operator to match the interested **_id** in the bibliography arrays of the documents.

3. Publications on the same journal

In this query we use the **"insertMany()"** method to add 5 new documents in the dataset. These publications shared the **journal** field and some of them also the **volume** field.

```
db.publications.insertMany( [
        {
            title: 'Ad maiora',
            journal: 'tempus',
            volume: 12,
            number: 5,
            pages:5
        }, {
            title: 'Ora et labora',
            journal: 'tempus',
            volume: 12,
            number: 7,
            pages:35
        }, {
            title: 'Mens sana in corpore sano',
            journal: 'tempus',
            volume: 12,
            number: 17,
            pages: 25
        }, {
            title: 'Verba volant, scripta manent',
            journal: 'tempus',
            volume: 36,
            number: 10,
            pages: 26
        }, {
            title: 'Tempus fugit',
            journal: 'tempus',
            volume: 36,
            number: 21,
            pages: 15
        }, {
        title: 'Ubi maior, minor cessat',
```

```
        journal: 'tempus',
        volume: 46,
        number: 21,
        pages: 17
    }
] )
```

4. Pages number increase

In this query we use the **"updateMany()"** method to increase by 5 the number of the pages of the publications contained in the same journal and which volume must be greater than or equal to 20 and less or equal to 30.

```
db.publications.updateMany(
        {
                "$and": [{
                        journal: "aenean auctor"
                        }, {
                        volume: { $gte: 20,  $lte: 30 }
                }]
        }, {
                $inc: {
                        pages: 5
                }
        }
)
```

5. Last author deletion

In this query we use the **"updateMany()"** method and the **"$pop"** operator to delete the last author of the publications that have 5 authors or more than 500 pages or the number of bibliography references less or equal to 15.

```
db.publications.updateMany(
        {
        "$or": [{
                authors: { $size: 5 }
                }, {
                pages: { "$gt": 500}
```

```
                }, {
                "$expr": { $lte: [{$size: "$bibliography"}, 15] }
         }]
        }, {
        $pop:{ authors: 1 }
        }
)
```

7.2. Queries

1. Volumes ranking of a journal

With this query we want to find out through the **"aggregate()"** method how many volumes there are in the journal "tempus" and how many articles each volume contains, all returned in the form of the final classification. We use the **$match** operator to select only the volumes that refer to the journal in question. Then we use the **$group** operator to group the articles according to the volume they belong to: the **_id** field must be equal to the value of the volume field. With the **$count** operator we count for each volume how many articles it contains and finally we classify the groups by the number of articles with the **$sort** operator.

```
db.publications.aggregate([
    {
    $match: {
        journal: "tempus"}
    },{
    $group: {
        _id: "$volume",
        articles_number: {
            $count: {}
        }
    }},{
    $sort: {
        articles_number: -1
    }}
])
```

```
{ _id: 12, articles_number: 4 }
{ _id: 36, articles_number: 3 }
{ _id: 46, articles_number: 2 }
{ _id: 23, articles_number: 1 }
{ _id: 49, articles_number: 1 }
```

2. Authors and countries

The pipeline that describes the operations done in this query is composed by these phases:

- In the first phase, we do a search with the **$match** operator with the aim to find a specific publication, filtering using the title and the number of the pages.
- In the second stage, we want to add a new field called **"authorsAndCountries"** to the document provided by the previous step, where this field contains an array whose elements are the names of the authors and their country code. To do this we use the **$addFields** operator and the **$map** operator to construct the array. Here we pass the values of the authors array as input, as the value of the 'as' field the name 'currentAuthor': this is the name of the variable that allows us to access each time the single element of the **authors** array. In the end we use the **$concat** operator to concatenate the strings described in the query and define each element of the new array in the form "author name - author country code".

```
db.publications.aggregate([
        {$match: {
                title: "Systems and Methods for Big and
                        Unstructured Data project",
                pages: 100
        }},
        {
        $addFields: {
              authorsAndCountries: {
                  $map: {
                         input: " $authors ",
                         as: " currentAuthor ",
                         in: {
                           $concat: [
                             " $$currentAuthor.name ",
```

```
                                " - ",
                                " $$currentAuthor.countryCode "
                            ]
                        }
                    }
                }
            }}
])
```

```
{ _id: ObjectId("6378ff6261485f115233eb3e"),
  title: 'Systems and Methods for Big and Unstructured Data project',
  authors:
   [ { name: 'Gianluca Pizzuti',
       title: 'student',
       countryCode: 'IT',
       gender: 'M' },
     { name: 'Melissande Simonin',
       title: 'student',
       countryCode: 'FR',
       gender: 'F' },
     { name: 'Enrico Brunetti',
       title: 'student',
       countryCode: 'IT',
       gender: 'M' },
     { name: 'Emanuele Paesano',
       title: 'student',
       countryCode: 'IT',
       gender: 'M' },
     { name: 'Pietro Lodi Rizzini',
       title: 'student',
       countryCode: 'IT',
       gender: 'M' } ],
  keywords: [ 'Big', 'Data' ],
  publicationDate: 2022-11-01T05:17:05.350Z,
  modificationDate: 2022-11-13T10:20:00.500Z,
  pages: 100,
  authorsAndCountries:
```

```
[ 'Gianluca Pizzuti - IT',
  'Melissande Simonin - FR',
  'Enrico Brunetti - IT',
  'Emanuele Paesano - IT',
  'Pietro Lodi Rizzini - IT' ] }
```

3. Books' publication

This query aims to find 5 books published between January 1, 2000 and January 1, 2001, and which have less than 100 pages or more than 400 pages.

We use the **find()** method, with the **logical operator "$and"** combine to the **logical operator "$or"**. In this way, All the documents whose publication date is greater than "2000-01-01" and less than "2001-01-01", and which have less than 100 pages or more than 400 pages will be returned.

In order to correctly perform operations on dates, we use the **ISODate** function which allows us to standardize the format of these. We then do a **projection** in order to display only the title, the publication date, and the number of pages of these books.

Finally, we use the **limit()** function to return only 5 results.

```
db.publications.find({
  "$and": [{
      $or: [
        {"pages": {"$lt": 100}},
        {"pages": {"$gt": 400}}
      ]
      },
      {"publicationDate": {"$gt": ISODate('2000-01-01')}},
      {"publicationDate": {"$lt": ISODate('2001-01-01')}},
    ]},
    {"title": 1, "publicationDate": 1, "pages": 1}
).limit(5)
```

4. Book and Portugal

We want, with this request, to know the total number of books written by at least one portuguese writer.

To do it, we will do two operations in a row, so we have to use the **"$aggregate"** operator. We use as first operation a **"$match"** on the subfield "countryCode" of authors, and we want it to be "PT" as "Portugal". Finally, we gather the results thanks to the **"$group"** operator, and we count it with the **"$count"** operator.

```
db.publications.aggregate([
    {"$match": {
        "authors.countryCode":  "PT"
    }},
    {"$group": {
        "_id": "authors.countryCode", "count": {"$sum": 1}
    }}
])
```

```
{ _id: 'authors.countryCode', count: 122 }
```

5. Women authors

We are looking through this query, the number of women authors in the collection.
It could be similar to the previous one, but here we want to count every single female writer, not just documents written by at least one female author. In this way, we use the **aggregate()** method, in order to do a pipeline of different functions. In the first operation, we use the **"$unwind"** operator on the field **"authors"** in order to transform each document into multiple documents containing only one author. By doing this, we can count the occurrence of documents written by a woman writer, without skipping anyone. In a second time, we use the **"$match" operator** with the nested attribute "authors.gender" to get documents written by a woman.
Then, as a third operation in the pipeline, we group the results by genre using the **"$group"** operator, and we count the number of results using the **"$sum"** operator.

```
db.publications.aggregate([
    {
        "$unwind": "$authors"
    },
    {
        "$match": {
            "authors.gender": "F"
        }
    },
    {
      "$group": {"_id": "authors.gender", "count": {"$sum": 1}}
    }
])
```

```
{ _id: 'authors.gender', count: 2118 }
```

6. Find authors that have written the greater number of publications

This query exploits the **aggregate** function to build a pipeline. With **$unwind** the con-

tent of *authors* array is expanded for each document and then the different authors are grouped. In this way, we are able to count occurrences of different authors in the whole dataset and we return for each of them name, country and the number of publications written. At the end we sort authors for the number of written publications in decreasing order and limit the result to the 5 most active authors. For this query we also assume that there aren't homonymous authors.

```
db.publications.aggregate([
    {
        "$unwind": {
            "path": "$authors"
        }
    }, {
        "$group": {
            "_id": "$authors.name",
            "country": {"$first": "$authors.countryCode"},
            "publicationsWritten": {"$sum": 1}
        }
    }, {
        "$sort": {
            "publicationsWritten": -1
        }
    }, {
        "$limit": 5
    }
])
```

7. Return the number of male and female authors for publications that has at least 2 female authors.

The main purpose of this query is to find out the number of male and female authors of each publication. To restrict the field a little bit, we start our pipeline with a **$match**, which allow us to consider only publications with more than 2 authors and more than 200 pages. After that we exploit **$addFields** for 2 consecutive times, in order to create the new fields containing the number of male and female authors. To obtain that value we compute the **$size** of a **$filter** which returns only male or female authors of a publication.

In the next stage, we **$match** only publications with more than 2 female authors and in the last stage we exploit **$projects** to produce our output.

```
db.publications.aggregate([
    {
        $match: {
            $and:[{
                "$expr": {
                    $gt: [{$size: "$authors"}, 2]
                    }
                }, {
                    pages: {$gt: 200}
                }
            ]
        }
    }, {
        $addFields: {
            femaleAuthors: {
                "$size": {
                    "$filter": {
                        input: "$authors",
                        as: "author",
                        cond: {
                            "$eq": ["$$author.gender", "F"]
                        }
                    }
                }
            },
            maleAuthors: {
                "$size": {
                    "$filter": {
                        input: "$authors",
                        as: "author",
                        cond: {
                            "$eq": ["$$author.gender", "M"]
```

```
                        }
                    }
                }
            }
        }
    }, {
        $match: {
            femaleAuthors: {$gt: 2}
        }
    }, {
        $project: {
            _id: 0,
            title: 1,
            femaleAuthors: 1,
            maleAuthors: 1
        }
    }
])
```

```
{ title: 'Cras pellentesque volutpat dui.',
  femaleAuthors: 3,
  maleAuthors: 1 }
{ title: 'Integer tincidunt ante vel ipsum.',
  femaleAuthors: 4,
  maleAuthors: 1 }
{ title: 'Vestibulum sed magna at nunc commodo placerat. Praesent blandit.',
  femaleAuthors: 3,
  maleAuthors: 2 }
{ title: 'Suspendisse potenti. Nullam porttitor lacus at turpis.',
  femaleAuthors: 3,
  maleAuthors: 2 }
{ title: 'Morbi non quam nec dui luctus rutrum. Nulla tellus.',
  femaleAuthors: 3,
  maleAuthors: 1 }
{ title: 'In congue.', femaleAuthors: 3, maleAuthors: 2 }
{ title: 'Sed ante. Vivamus tortor.',
  femaleAuthors: 3,
  maleAuthors: 0 }
```

8. Find the publication with the highest number of sub-sections.

To achieve the purpose, we first make a projection on

- **_id**
- **title**
- **numberOfSubSections**. To obtain the number of sub-sections, we use **$map** applied on the **$sections** array in order to map every document of this array into the size of its **subsections** field. All the sizes are then summed up with **$sum**

After that, we apply a descendent sorting and limit the results to the first.

```
db.publications.aggregate([{
    $project: {
        _id: 1,
        title: 1,
        numberOfSubSections: {
            $sum: {
```

```
                $map: {
                    input: '$sections',
                    'in': {
                        $size: '$$this.subsections'
                    }
                }
            }
        }
    }
    },
    {$sort: {numberOfSubSections: -1}},
    {$limit: 1}
])
```

```
{ _id: ObjectId("637bb9d894b847a5ac78c041"),
  title: 'Integer ac leo.',
  numberOfSubSections: 91 }
```

9. Find largest articles written by authors whose mail has a ".gov" or ".edu" domain.

In this query we are looking for articles that come from a governmental source, or from an educational institution. While there are several ways to achieve this, we will take advantage of the **$regex** operator offered by MongoDB.

This query works through the **.aggregate()** function. Inside it, the **$unwind** operator is used to convert arrays of authors into single author objects, and then the **$match** operator is applied to return the relevant articles.
MongoDB allows the usage of Regular Expressions inside a **$match** clause through the **$regex** operator. Our regex was designed to check that the string is formatted as an email, and that it contains exactly ".edu" or ".gov" at its end. The results are then sorted according to the number of pages, in descending order, projected on the most relevant fields to improve readability, and finally limited to 3, so that only the 3 largest documents are displayed.

```
db.publications.aggregate([
  {$unwind: '$authors'},
  {$match:
          {
            'authors.email':
            {
            $regex:^[a-z0-9._\%+-]+@[a-z0-9.-]+\.(?:|edu|gov)$/
            }
          }
  },
  {$sort:{pages:-1}},
  {$project:
      {
      title:1,
      authors:{name:1, email:1,countryCode:1, gender:1},
      pages: 1
      }
  },
  {$limit: 3}
  ])
```

```
{ _id: ObjectId("637bb9d894b847a5ac78c131"),
  title: 'In hac habitasse platea dictumst. Maecenas ut massa quis augue luctus tincidunt.',
  authors:
   { name: 'Alexei Luckett',
     email: 'aluckett1@ed.gov',
     countryCode: 'CO',
     gender: 'M' },
  pages: 1199 }
{ _id: ObjectId("637bb9d894b847a5ac78bf0d"),
  title: 'Morbi non quam nec dui luctus rutrum. Nulla tellus.',
  authors:
   { name: 'Dolph Maybery',
     email: 'dmaybery3@berkeley.edu',
     countryCode: 'CN',
     gender: 'F' },
  pages: 1194 }
{ _id: ObjectId("637bb9d894b847a5ac78c18c"),
  title: 'Vestibulum ac est lacinia nisi venenatis tristique.',
  authors:
   { name: 'Eveline Jellman',
     email: 'ejellman3@nih.gov',
     countryCode: 'RU',
     gender: 'M' },
  pages: 1191 }
```

10. New Array exploring bibliography references

The goal of this query is to access the bibliography field of the first two publications present in the dataset and add to these two a field that contains an array with the documents nested in the bibliography field through their **_id**. Moreover, the document fields representing the elements of this new array will be exclusively the **title** and the number of **pages** of the relative publication. Note that the new array will be empty if the referenced publication has less than 500 pages. To do this operation we use the **aggregate** method: in particular, in the first step we use the **$lookup** stage who performs a left outer join to our collection:

- in the **from** field we specify our collection to perform the join operation;
- in the **let** field we declare the variable that we will use in the pipeline, that in our case is **pages**;
- in the **pipeline** field we make known the pipeline (operations to do) to run on the joined collection;
- in the **as** field we specify the name of the new array field to add the joined documents as "citedPublications".

With the last two phases of the aggregate method we make sure that we execute the process only for the first two elements of the dataset with the **$limit** operator and to return only the fields that interest us and that we specify in the **$project** operator.

```
db.publications.aggregate([{
    $lookup: {
        from: "publications",
        let: {
            "pages": "$pages"
        },
        pipeline: [{
            $match: {
                $expr: {
                    $gte: ["$$pages", 500]
                }
            }
        }, {
            $limit: 1
        }, {
            $project: {
                title: 1,
                pages: 1
            }
        }],
        as: "citedPublications"
    }}, {
    $limit: 2
    }, {
```

```
    $project: {
        title: 1,
        pages: 1,
        citedPublications: 1
    }
}])
```

```
< { _id: ObjectId("637bb9d894b847a5ac78beb7"),
    title: 'Cras pellentesque volutpat dui.',
    pages: 282,
    citedPublications: [] }
  { _id: ObjectId("637bb9d894b847a5ac78beb8"),
    title: 'Nulla ut erat id mauris vulputate elementum.',
    pages: 1058,
    citedPublications:
     [ { _id: ObjectId("637bb9d894b847a5ac78beb7"),
         title: 'Cras pellentesque volutpat dui.',
         pages: 282 } ] }
```

8 | Spark

This section will be dedicated to the implementation of a Spark database for the reduced "dblp" repository dataset, which was already mentioned in chapter 3.
Spark is a comprehensive computation platform meant to be used with big data. It features a distributed computation engine, where data can be shared and processed remotely by several worker nodes. However, in this report we are going to use a local session of Spark, through the "pyspark" python library. Two of Spark's native data structures are the RDD (Resilient Distributed Dataset) and the DataFrame. The RDD was created with the purpose of enabling parallel computation on a dataset, while granting it fault-tolerance and distributing the data across nodes; the DataFrame was built on top of the RDD concepts and features improved syntax and performance. In this report we are going to create DataFrames starting from the "dblp" data, initially stored in csv files, and then perform some queries using the Python API for DataFrames.

8.1. SparkSession start and Data Upload

We will access Spark's functionalities through the "pyspark.sql" library. To achieve this, as a first step we import all the necessary dependencies in a Python notebook.

```
from pyspark.sql import SparkSession
from functools import reduce
from pyspark.sql.types import *
from pyspark.sql.functions import col, split, expr, array_contains
```

Once we imported the required packages, we can now create a local instance of SparkSession, which constitutes the entry point of our PySpark application. We then assign it to a variable named "spark".

```
spark = SparkSession.builder \
      .master("local") \
      .appName("MyFirstSparkApplication") \
      .getOrCreate()
```

Now we can import our data and create DataFrames out of it.
Every DataFrame is structured as a collection of "row" objects, that have a predefined common schema. This means that we have to feed the schema of our data to Spark when importing, or the data type of each column will default to "StringType". This is not ideal in our case, since we want to treat certain columns as numbers or arrays.

In the original csv files the name and type of each column was specified in the header, so we wrote a script to extract a "StructType" object, which represents a schema for a DataFrame, starting from the format of our header. We report this script in the next figure.

```
def HeaderToSchema(header):
    structTypeArg = [] # argument to be passed to StructType function
    arrayCols = [] # holds the name of the columns that have an array type

    for col in header.columns:
        typeStr = col.split(":")[1]
        colStr = col.split(":")[0]
        if typeStr == "string[]":
            newCol = StructField(colStr,StringType(),True)
            arrayCols.append(colStr)
        elif typeStr == "string":
            newCol = StructField(colStr,StringType(),True)
        elif typeStr == "int":
            newCol = StructField(colStr,IntegerType(),True)
        elif typeStr == "date":
            newCol = StructField(colStr,TimestampType(),True)
        elif typeStr == "ID":
            newCol = StructField(colStr,IntegerType(),True)
        else :
            newCol = StructField(colStr,StringType(),True)

        structTypeArg.append(newCol);

    schema = StructType(structTypeArg)

    return schema, arrayCols

# returns a new df with the columns in cols converted from String to Array,
# by splitting the string wrt '|'
def convertStrToArray(df, cols):
    return reduce(
        lambda df, colname: df.withColumn(colname, split(col(colname), "\\|").alias(colname)),
        cols,
        df
    )
```

These two functions are used, respectively, to generate the schema starting from the header, and to alter the schema by converting normal strings to arrays of strings where is needed, for example in the case of a list of authors. In the following code we exploit the functions to import the csv file containing articles.

```
#1. Articles
header = spark.read.option("header", True).option("delimiter", ";").csv("script_e_dati/output_article_header.csv")
schema, cols = HeaderToSchema(header)
articles = spark.read.option("header", False).option("delimiter", ";").csv("script_e_dati/output_article.csv",schema = schema)

# convert to array
articles = convertStrToArray(articles, cols)
```

We can display the schema of the DF with articles.printSchema(). This yields the following output:

```
root
 |-- article: integer (nullable = true)
 |-- author: array (nullable = true)
 |    |-- element: string (containsNull = false)
 |-- author-aux: string (nullable = true)
 |-- author-orcid: array (nullable = true)
 |    |-- element: string (containsNull = false)
 |-- booktitle: string (nullable = true)
 |-- cdate: timestamp (nullable = true)
 |-- cdrom: string (nullable = true)
 |-- cite: array (nullable = true)
 |    |-- element: string (containsNull = false)
 |-- cite-label: array (nullable = true)
 |    |-- element: string (containsNull = false)
 |-- crossref: string (nullable = true)
 |-- editor: array (nullable = true)
 |    |-- element: string (containsNull = false)
 |-- editor-orcid: array (nullable = true)
 |    |-- element: string (containsNull = false)
 |-- ee: array (nullable = true)
 |    |-- element: string (containsNull = false)
 |-- ee-type: array (nullable = true)
 |    |-- element: string (containsNull = false)
 |-- i: array (nullable = true)
 |    |-- element: string (containsNull = false)
 |-- journal: string (nullable = true)
 |-- key: string (nullable = true)
 |-- mdate: timestamp (nullable = true)
 |-- month: string (nullable = true)
 |-- note: array (nullable = true)
```

```
|    |-- element: string (containsNull = false)
|-- note-label: string (nullable = true)
|-- note-type: array (nullable = true)
|    |-- element: string (containsNull = false)
|-- number: string (nullable = true)
|-- pages: string (nullable = true)
|-- publisher: string (nullable = true)
|-- publnr: string (nullable = true)
|-- publtype: string (nullable = true)
|-- sub: array (nullable = true)
|    |-- element: string (containsNull = false)
|-- sup: array (nullable = true)
|    |-- element: string (containsNull = false)
|-- title: string (nullable = true)
|-- title-bibtex: string (nullable = true)
|-- tt: array (nullable = true)
|    |-- element: string (containsNull = false)
|-- url: array (nullable = true)
|    |-- element: string (containsNull = false)
|-- volume: string (nullable = true)
|-- year: integer (nullable = true)
```

Every row of the articles DF is structured this way, and it is coherent with the specifications of the csv header.
The full data upload process consists in the application of this procedure to all of the csv files of the "dblp" data:

- articles
- inproceedings
- proceedings
- books
- incollection
- www
- phdthesis
- mastersthesis

In the end, we have a DataFrame for each of the files, and can go on to perform queries on them.

9 | Spark queries

9.1. Creation Update operations

1. Deletion of rows containing null value

This query aims to **remove rows** from the **"mastersthesis" dataframe** that contain a **null value** in the **"ee" column**, which corresponds to the URL of the websites where the documents are available. Since this data is important for our future queries, we want to sort the data and delete those that do not contain this information.

For display reasons, we will only select a limited number of columns.

Before :

```
# Projection of the masterthesis Dataframe
mastersthesis.select("mastersthesis", "author", "ee", "school", "title")\
    .show()
```

```
+-------------+--------------------+--------------------+--------------------+--------------------+
|mastersthesis|              author|                  ee|              school|               title|
+-------------+--------------------+--------------------+--------------------+--------------------+
|          586|     Stephan Vollmer|[http://dbis.uni-...|Diplomarbeit, Uni...|Portierung des DB...|
|          587|    Vanessa C. Klaas|[http://www.pms.i...|Diplomarbeit, LMU...|Who's Who in the ...|
|          588|       Kurt P. Brown|                null|University of Wis...|PRPL: A Database ...|
|          589|         Tolga Yurek|                null|University of Cal...|Efficient View Ma...|
|          590|            Rita Ley|[http://dblp.uni-...|Diplomarbeit, Uni...|Der Einfluss klei...|
|          591|Oliver Hoffmann 0002|[http://dblp.uni-...| University of Trier|Regelbasierte Ext...|
|      9365350|           Liang Dai|[https://www.esch...|University of Cal...|Online Controlled...|
|      9370075|Rodrigo Pastl Pontes|[http://search.nd...|Instituto Tecnoló...|Contribuições do ...|
|      9374896|     Satyan R. Coorg|[https://hdl.hand...|Massachusetts Ins...|Partitioning non-...|
|      9419456|       Peter Van Roy|                null|University of Cal...|A Prolog Compiler...|
|      9419466|Christian Schulte...|                null|Universität Karls...|Entwurf und Imple...|
|      9436290|        Salim Perchy|[https://tel.arch...|Pontifical Xavier...|Multimedia Intera...|
|      9448531|        Wangda Zhang|[https://doi.org/...|University of Hon...|Evaluating multi-...|
|      9452075|Salma Hosni Emam ...|[http://n2t.net/a...|University of Cal...|Context-Aware Run...|
|      9455480|         Tatu Ylönen|                null|Helsinki Universi...|Shadow Paging Is ...|
+-------------+--------------------+--------------------+--------------------+--------------------+
```

After :

```
# Projection of the masterthesis Dataframe, which doesn't contain null values anymo
```

```
masterthesis.na.drop(subset=["ee"]) \
    .select("masterthesis", "author", "ee", "school", "title")\
    .show()
```

```
+------------+--------------------+--------------------+--------------------+--------------------+
|masterthesis|              author|                  ee|              school|               title|
+------------+--------------------+--------------------+--------------------+--------------------+
|         586|     Stephan Vollmer|[http://dbis.uni-...|Diplomarbeit, Uni...|Portierung des DB...|
|         587|    Vanessa C. Klaas|[http://www.pms.i...|Diplomarbeit, LMU...|Who's Who in the ...|
|         590|           Rita Ley|[http://dblp.uni-...|Diplomarbeit, Uni...|Der Einfluss klei...|
|         591|Oliver Hoffmann 0002|[http://dblp.uni-...| University of Trier|Regelbasierte Ext...|
|     9365350|           Liang Dai|[https://www.esch...|University of Cal...|Online Controlled...|
|     9370075|Rodrigo Pastl Pontes|[http://search.nd...|Instituto Tecnoló...|Contribuições do ...|
|     9374896|     Satyan R. Coorg|[https://hdl.hand...|Massachusetts Ins...|Partitioning non-...|
|     9436290|        Salim Perchy|[https://tel.arch...|Pontifical Xavier...|Multimedia Intera...|
|     9448531|        Wangda Zhang|[https://doi.org/...|University of Hon...|Evaluating multi-...|
|     9452075|Salma Hosni Emam ...|[http://n2t.net/a...|University of Cal...|Context-Aware Run...|
+------------+--------------------+--------------------+--------------------+--------------------+
```

2. Deleting proceedings without editors

The aim of this query is to add a new column regarding the number of the editors for each proceeding and to remove the proceeding that don't have at least one editor.

Before :

```
proceedings.withColumn("number_of_editors",size(col("editor")))\
.select(col("editor"), col("number_of_editors"))\
.show()
```

```
+--------------------+-----------------+
|              editor|number_of_editors|
+--------------------+-----------------+
|[Eric Madelaine, ...|                2|
|[Galia Angelova, ...|                3|
|[Hideo Saito, Rai...|                3|
|                null|               -1|
|[Gilles Adda, Kar...|                3|
| [Manfred Paul 0001]|                1|
|                null|               -1|
|[David Daly, Jos...|                3|
|[Hugo Velthuijsen...|                2|
|[Andr Thomas 000...|                4|
|[Craig Anslow, Fe...|                4|
+--------------------+-----------------+
```

After :

```
proceedings.withColumn("number_of_editors",size(col("editor")))\
.filter(col("number_of_editors") > 0) \
.select(col("editor"), col("number_of_editors"))\
.show()
```

```
+--------------------+-----------------+
|              editor|number_of_editors|
+--------------------+-----------------+
|[Eric Madelaine, ...|                2|
|[Galia Angelova, ...|                3|
|[Hideo Saito, Rai...|                3|
|[Gilles Adda, Kar...|                3|
| [Manfred Paul 0001]|                1|
|[David Daly, Jos...|                3|
|[Hugo Velthuijsen...|                2|
|[Andr Thomas 000...|                4|
|[Craig Anslow, Fe...|                4|
|[Holger Pirk, Tho...|                2|
+--------------------+-----------------+
```

3. Remove a specific author from the author array of the articles dataframe

The purpose of this operation is to update the articles dataframe by removing a specific author from the author array.

```
articles = articles.withColumn("author", when(array_contains(articles.author,
        "Marina L. Gavrilova"),
        array_remove(articles.author, "Marina L. Gavrilova"))
    .otherwise(articles.author))
```

4. Remove an article

The purpose of this operation is to update the articles dataframe by removing a tuple based in its id.

```
articles = articles.filter(articles.article != 10)
```

5. Create a dataframe containing the authors of articles

The aim of this command is to create a new dataframe containing the authors. This is done by applying the explode function on the author column of the articles dataframe.

```
articles.select(explode(articles.author).alias("singleAuthor"), articles.key)
```

9.2. Queries

1. Books and articles written by the same author

This query is of complexity **'Where" and "join"**. The purpose of this query is to make a join between the **"books" and "articles" dataframes**, in order to display the **copies written by the same author**. We decided to do a join on the **"author-orcid" attribute** because it makes it possible to reference the authors in a unique way, despite the homonyms. We have also given an **alias** to the tables, in order to be able to **access the attribute** via a string (Indeed, the title of the column "author-orcid" contains a "-", which makes it impossible to select it directly via the dataframe).

For display reasons, we decided to print the results vertically.

```
# Selection of authors which have written both books and articles
books.alias("books").join(articles.alias("articles"),
        col("books.author-orcid") == col("articles.author-orcid"), "inner") \
    .select("books.author", "books.title", "articles.title") \
    .show(n=5, truncate=False, vertical=True)
```

```
-RECORD 0-----------------------------------------------------------------------------------------------------
 author | [David Zhang 0001, Kebin Wu]
 title  | Pathological Voice Analysis
 title  | BDPCA plus LDA: a novel fast feature extraction technique for face recognition.
-RECORD 1-----------------------------------------------------------------------------------------------------
 author | [David Zhang 0001, Fangmei Chen, Yong Xu 0001]
 title  | Computer Models for Facial Beauty Analysis
 title  | BDPCA plus LDA: a novel fast feature extraction technique for face recognition.
-RECORD 2-----------------------------------------------------------------------------------------------------
 author | [Heiko Angermann, Naeem Ramzan]
 title  | Taxonomy Matching Using Background Knowledge, Linked Data, Semantic Web and Heterogeneous Repositories.
 title  | An efficient optimisation scheme for scalable surveillance centric video communications.
-RECORD 3-----------------------------------------------------------------------------------------------------
 author | [David Zhang 0001, Kebin Wu]
 title  | Pathological Voice Analysis
 title  | Selecting a Reference High Resolution for Fingerprint Recognition Using Minutiae and Pores.
-RECORD 4-----------------------------------------------------------------------------------------------------
 author | [David Zhang 0001, Fangmei Chen, Yong Xu 0001]
 title  | Computer Models for Facial Beauty Analysis
 title  | Selecting a Reference High Resolution for Fingerprint Recognition Using Minutiae and Pores.
only showing top 5 rows
```

2. Most recent articles containing "Analysis"

This query is of complexity **'Where", "limit" and "like"**. This query aim to find the articles containing the word "Analysis" in its "title" attribute. We also want to filter the results on the date, and only display only the most up-to-date articles, i.e. modified after 2020. This query could be perform by someone who want to find updated articles talking

about this subject.

For display reasons, we decided to print the results vertically.

```
# Selection of articles which contained the word "Analysis", modified after 2020
articles.filter((col("title").like("%Analysis%")) &
        (to_date(col("mdate"), "dd/MM/yyyy") >= lit("2020-01-01"))) \
    .select("article", "title", "mdate") \
    .limit(5) \
    .show(truncate = False, vertical = True)
```

```
-RECORD 0-----------------------------------------------------------------------------------------------------------
 article | 1045702
 title   | A Mean Field Game Analysis of Distributed MAC in Ultra-Dense Multichannel Wireless Networks.
 mdate   | 14/11/2020
-RECORD 1-----------------------------------------------------------------------------------------------------------
 article | 2800677
 title   | A Cost-Benefit Analysis of Capacitor Allocation Problem in Radial Distribution Networks Using an Improv
ed Stochastic Fractal Search Algorithm.
 mdate   | 09/03/2021
-RECORD 2-----------------------------------------------------------------------------------------------------------
 article | 2592559
 title   | BER Analysis for Spatial Modulation in Multicast MIMO Systems.
 mdate   | 01/09/2020
-RECORD 3-----------------------------------------------------------------------------------------------------------
 article | 578901
 title   | Time Series-Analysis Based Engineering of High-Dimensional Wide-Area Stability Indices for Machine Lear
ning.
 mdate   | 12/08/2021
-RECORD 4-----------------------------------------------------------------------------------------------------------
 article | 1774974
 title   | Contingency Analysis Based on Partitioned and Parallel Holomorphic Embedding.
 mdate   | 13/04/2021
```

3. Authors of incollection and articles

This query is of complexity **'Where", "in" and "Nested query"**. This query allows you to find **the authors** who have **written both an article and an incollection in 2020**. We first get the **orcid of the authors who have written an incollection in 2020** into a GroupFrame, we then **transform this GroupFrame into a list**, and finally we **search for articles** who have an **author-orcid identical to at least one** of those present **in the list**. We also remove columns from the "incollection" dataframe which have a null value in the author-orcid column, in order to perform a more efficient search and remove unnecessary data. This query could also be done by a "join" between the dataframe "Incollection" and "Articles".

For display reasons, we will only select a limited number of columns : Author, Author-orcid and title

```
#  Selection of authors which have written incollection in 2020
authors = incollection.filter((to_date(col("mdate"), "dd/MM/yyyy") >=
        lit("2020-01-01")) & (to_date(col("mdate"), "dd/MM/yyyy") <=
        lit("2021-01-01"))) \
    .groupBy("author-orcid") \
    .count() \
    .select(explode(col('author-orcid'))) \
    .collect()

# Transform GroupData into a List
authors_orcid_list = []
for row in authors :
    if row['col'] != None :
        authors_orcid_list.append(row['col'])

#Deletion of null value in the incollection DataFrame
incollection.na.drop(subset=["author-orcid"]) \
    .select("author-orcid", "author")\
    .show()

# WHERE, IN, NESTED QUERY
articles.filter((to_date(col("mdate"), "dd/MM/yyyy") >= lit("2020-01-01"))
        & (to_date(col("mdate"), "dd/MM/yyyy") <= lit("2021-01-01"))) \
    .select(col('author'), explode(col('author-orcid')), col('title')) \
    .withColumnRenamed('col', 'orcid') \
    .filter(col('orcid').isin(authors_orcid_list)) \
    .show()
```

```
+--------------------+------------------+--------------------+
|              author|             orcid|               title|
+--------------------+------------------+--------------------+
|[Davide Scaramuzz...|0000-0002-5805-8892|Visual Odometry [...|
|[Agostino Dovier,...|0000-0003-2052-8593|Extending Logic P...|
|[Bin Ning, Guanro...|0000-0003-1381-7418|Approximation-Bas...|
|[John C. Grundy, ...|0000-0003-4928-7076|Experiences in De...|
|[Oscar Castillo 0...|0000-0002-7385-5689|Interval-Related ...|
|[Adam C. Faust, A...|0000-0001-9926-7036|A Study of BER-Op...|
|[Alfredo García H...|0000-0002-8435-680X|Improving the eff...|
|[Guanrong Chen, J...|0000-0003-1381-7418|Bifurcations of T...|
|[Christian Blum 0...|0000-0002-1736-3559|An Ant Colony Opt...|
|[Jesús González-B...|0000-0002-9281-4209|Toward interactiv...|
|[Jesús González-B...|0000-0003-4949-9220|Toward interactiv...|
|[A. K. Ray, Raghu...|0000-0003-3035-761X|Signal phase reco...|
|[Emanuel Ontivero...|0000-0002-6784-6305|New Methodology t...|
|[Emanuel Ontivero...|0000-0002-7385-5689|New Methodology t...|
|[Guanrong Chen, S...|0000-0003-1381-7418|Generation of n ×...|
+--------------------+------------------+--------------------+
```

4. Number of inproceedings related to each publisher.

This query is of a complexity of at least **"Group By", "1 join", "as"**. Since each inproceeding belongs to a proceeding and each proceeding is organized by a publisher first of all we need an inner join on the attribute *booktitle* of both dataframes to make every proceeding match with all his related inproceedings. After that we select *title* of inproceedings and explode *publisher* of proceeding, since they can be more than one and are stored in an array of strings, and group them by *singlePublisher*. At this point, we count the number of inproceedings for every single publisher and return publisher name and number of inproceedings sorted in decreasing order.

```
inproceedings.join(proceedings,
    inproceedings.booktitle == proceedings.booktitle, "inner") \
    .select(inproceedings.title,
        explode(proceedings.publisher).alias("singlePublisher")) \
    .groupby("singlePublisher") \
    .count() \
    .withColumnRenamed("count","numberOfInproceedings") \
    .sort(col("numberOfInproceedings").desc()) \
    .show(truncate=False)
```

```
+-----------------------------------------------+---------------------+
|singlePublisher                                |numberOfInproceedings|
+-----------------------------------------------+---------------------+
|IEEE                                           |5018                 |
|IEEE Computer Society                          |2208                 |
|ACM                                            |1171                 |
|Springer                                       |1087                 |
|Schloss Dagstuhl - Leibniz-Zentrum fr Informatik|316                |
+-----------------------------------------------+---------------------+
only showing top 5 rows
```

5. Number of articles written by a specific author over the years

This query is of complexity **'Where", "Group By"**. The aim of this query is to find the number of articles written by a specific author in every year. This is achieved by first filtering over the articles written by the author, then grouping by year. Finally, the count() function is applied and the results are sorted by year.

```
articles.filter(array_contains(articles.author, "Joaquim Filipe")) \
.groupby(articles.year) \
.count() \
.sort("year") \
.show()
```

```
+----+-----+
|year|count|
+----+-----+
|2005|    1|
|2006|    2|
|2007|    2|
|2008|    2|
|2009|    1|
|2013|    2|
|2014|    1|
|2015|    1|
+----+-----+
```

6. Return number of inproceedings written by each author that has written more than one of them.

This query is of a complexity of at least **'Group By", "Having", "As"**. In every inproceeding the authors are stored in an array of strings. So, to perform this query, first of all we have to apply an explode to the *author* attribute of *inproceedings* df in order to have a row for each author of each inproceeding. Then, we group by the author's name and count occurrences. At the end a filter is applied in order to consider only authors that have written more than an inproceeding and at the end they are sorted by decreasing order of *writtenInproceedings*.

```
inproceedings.select(explode(inproceedings.author).alias("singleAuthor")) \
     .groupby("singleAuthor") \
     .count() \
     .withColumnRenamed("count","writtenInproceedings") \
     .filter("count > 1") \
     .sort(col("writtenInproceedings").desc()) \
     .show()
```

```
+--------------------+--------------------+
|        singleAuthor|writtenInproceedings|
+--------------------+--------------------+
|    Edwin R. Hancock|                  27|
|         Jianyu Yang|                  24|
|       Marco Liserre|                  22|
|       Adriano Camps|                  22|
|         Bingfang Wu|                  21|
|        Qian Du 0001|                  21|
|         Jie Wu 0001|                  20|
|          Qinhuo Liu|                  20|
|    Masanobu Shimada|                  19|
|     Kamal Al-Haddad|                  19|
|        Philip S. Yu|                  19|
|        Liangyun Liu|                  18|
|Haruo Kobayashi 0001|                  17|
|         Yulin Huang|                  17|
|Mohamed-Slim Alouini|                  17|
|     Bhim Singh 0001|                  17|
|          Jihua Wang|                  17|
```

```
|          Anna Kuwana|                  17|
|        David Lo 0001|                  17|
|       Liangpei Zhang|                  17|
+--------------------+--------------------+
only showing top 20 rows
```

7. Returns the number of articles whose title starts with a specific letter, written by certain authors

This query is of complexity **'Where', 'Group By', 'Having' and 'As'**. The goal of this query is to output the number of articles whose title starts with the letter **A**, written by two authors specified in the **filter()** parameters of the second line. Everything is output in descending order with the help of the **sort()** function.

```
articles.filter(col('title').startswith('A')) \
.filter(array_contains(articles.author, 'Ivan Yotov') |
                    array_contains(articles.author, 'Paola Bonizzoni'))\
.groupby(col('author')) \
.count() \
.withColumnRenamed('count', 'number_of_articles') \
.sort(col('number_of_articles').desc()) \
.show(truncate= False)
```

```
+---------------------------------------------+------------------+
|author                                       |number_of_articles|
+---------------------------------------------+------------------+
|[Paola Bonizzoni, Tao Jiang 0001, Yuri Pirola]|2                 |
|[Ivan Yotov, Mary F. Wheeler]                |1                 |
+---------------------------------------------+------------------+
```

The correct functioning of this operation is easily verifiable with the help of the consequent query:

```
articles.filter(array_contains(articles.author, 'Ivan Yotov') |
                    array_contains(articles.author, 'Paola Bonizzoni'))\
.select(col('title'), col('author')) \
.show()
```

```
+--------------------+--------------------+
|               title|              author|
+--------------------+--------------------+
|A Posteriori Erro...|[Ivan Yotov, Mary...|
|An Efficient Algo...|[Paola Bonizzoni,...|
|Mixed Finite Elem...|[Ivan Yotov, Lawr...|
|A new theory: Nes...|[Paola Bonizzoni,...|
|Complexity insigh...|[Florian Sikora, ...|
+--------------------+--------------------+
```

8. Article of a specific journal having the maximum number of authors
This query is of complexity **'Where", "Nested Query", "Group By"**.

```
articles_exploded = articles.filter(col('journal') == "Sci. Eng. Ethics") \
.select(col('article'), col('title'), explode(col('author')))

articles_exploded.groupby(col('article'), col('title')) \
.count() \
.sort(col('count').desc()) \
.limit(1).show()
```

```
+-------+--------------------+-----+
|article|               title|count|
+-------+--------------------+-----+
|  29658|Working with Rese...|   22|
+-------+--------------------+-----+
```

9. Proceedings edited by a specific person containing more than 50 inproceedings
This query is of complexity **'Where", "Group By", "Having", "Join"**. This query aims to extract the keys of the proceedings that were edited by "Joaquim Filipe" (among other editors) and contains more than 50 inproceedings.

```
proceedings.join(inproceedings, array_contains(inproceedings.crossref,
    proceedings.key)) \
```

```
.filter(array_contains(proceedings.editor, "Joaquim Filipe")) \
.select(proceedings.key, proceedings.proceedings) \
.groupby(proceedings.key).count() \
.where("count > 50") \
.show()
```

```
+------------------+-----+
|               key|count|
+------------------+-----+
| conf/ijcci/2011-1|   75|
|conf/icaart/2011-2|   69|
|conf/icinco/2006ra|   89|
+------------------+-----+
```

10. Return the list of inproceedings written by authors that have a website and have written more than one inproceedings but no articles.
This query is of a complexity of at least **'Where", "Group By", "Having", " 2 Joins"**. The title of this query may sound a little weird but this is meant in order to be able to perform 2 join operations with different purposes in the same query and to concatenate them with other complexity requirements. Since we will work on *articles*, *inproceedings* and *www* and in each of them the *author* field is an array of strings (because there could be more than one author), we decided to create three other versions of these dataframes where an **explode** function is used in order to have a row for every single author.

```
art = articles.select(explode(articles.author)
                    .alias("singleAuthor"), articles.key)

inp = inproceedings.select(explode(inproceedings.author)
                        .alias("singleAuthor"), inproceedings.key)

wwwSingleAuthors = www.select(explode(www.author)
                        .alias("singleAuthor"), www.key, www.mdate)
```

At this point the main query can be performed. First of all we decide, with a filter, to take in consideration only websites that have been modified at least once since 2010. As second operation we execute an inner join on the author between websites and inproceedings. In this way we obtain a table containing all matches, in other words all combinations between inproceedings and authors that have a website. Subsequently we group by *singleAuthor*

but apply an aggregate operation with an array_join in order to have, for each author, a string with the keys of all inproceedings written by him. Now we perform the second join (leftanti join on *singleAuthor* with *articles* df) in order to eliminate authors that have written at least one article. At the end we use a split to transform the string with all keys in an array of strings in order to check its size and filter only authors that have written more than one inproceeding.

```
wwwSingleAuthors.filter(year(to_date(col("mdate"), "dd/MM/yyyy")) >= "2010") \
                .join(inp, wwwSingleAuthors.singleAuthor == inp.singleAuthor,
                    "inner") \
                .drop(inp.singleAuthor) \
                .groupby(wwwSingleAuthors.singleAuthor) \
                .agg(
                    array_join(
                        collect_list(inp.key),
                        delimiter=',',
                    ).alias("key")
                ) \
                .join(art, wwwSingleAuthors.singleAuthor == art.singleAuthor,
                    "leftanti") \
                .select(wwwSingleAuthors.singleAuthor, split("key", ",")
                    .alias("key")) \
                .filter(size("key") > 1) \
                .show(n=20)
```

```
+--------------------+--------------------+
|        singleAuthor|                 key|
+--------------------+--------------------+
|       Abdellah Adib|[conf/sita/Sekkat...|
|     Abhishek Sehgal|[conf/isie/Sehgal...|
|         Adam Widera|[conf/iscram/Wide...|
|  Adrian David Cheok|[conf/ismar/ChoiC...|
|       Akira Shibata|[conf/igarss/Shib...|
|Alexander Pretschner|[conf/IEEEares/Mo...|
|   Alexander Tekleab|[conf/icde/StorlT...|
|Alexandre Bouchar...|[conf/aistats/Jun...|
```

```
|      Alexey Kurakin|[conf/iclr/Kuraki...|
|     Alice Dalphinet|[conf/igarss/Suqu...|
|         Amel Yessad|[conf/icwl/Yessad...|
|       Andreas Baude|[conf/robocup/Blu...|
|        Andrei Lobov|[conf/iecon/Lobov...|
|       Antoine Isaac|[conf/esws/Hollin...|
|     Antonella Guzzo|[conf/ideal/Rullo...|
|      Antske Fokkens|[conf/eacl/Reuver...|
|       Austin Melton|[conf/icitst/Alba...|
|     Ayako Abe-Ouchi|[conf/igarss/Rodr...|
|     Bernd Michaelis|[conf/smc/Panning...|
|        Bilal Yousuf|[conf/ectel/Yousu...|
+--------------------+--------------------+
only showing top 20 rows
```