POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Systems and Methods for Big and Unstructured Data Project

Author(s): **Alessio Buda (10675063)**

Leonardo Cesani (10674905)

Fausto Lasca (10661818)

Gabriele Munafò (10654531)

Matteo Paraboschi (10681473)

Group Number: **20**

Academic Year: 2022-2023

Contents

1 | Problem specification

1.1. Description

The aim of the project is to implement a database storage solution for a large-scale dataset recording scientific publications. The database contains various types of scientific publications, including journal articles, books, thesis, conference and workshop papers. Each publication is stored together with important information regarding the publication itself (e.g., its author(s), title, year of publication . . .).

The system should allow users to search for specific publications or to find publications based on some of their characteristics.

1.2. Assumptions

The assumptions considered are the following:

- Every conference edition is a separate entity, since a conference can be held after variable time spans. For journals however a journal entity is needed because it is a stable institution, that publishes the journal edition at a fixed pace.
- The author cannot be affiliated to no organization, and he can be affiliated to at most one organization. That means that there is not a record of all organizations where the author worked during his life.
- The names of organizations, publishers and schools are considered to be unique.

2 | ER model

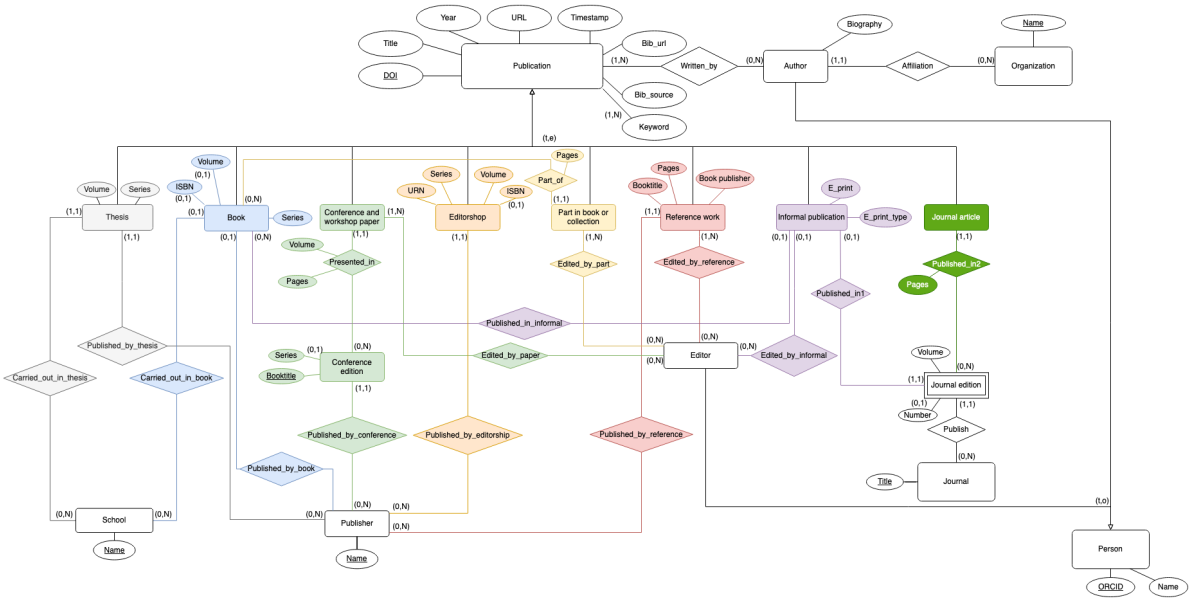


Figure 2.1: ER diagram.

Note: The relationships should be read as follows: a publication is written by at least one author and at most N authors. An author may have written from 0 to N publications.

The ER model presents the following entities:

- **Publication**: represents a generic scientific publication. It is identified by DOI (primary key), the title, the year of publication, the URL of the publication, the timestamp, the Bib URL, the Bib source and potentially multiple keywords associated to the publication. Publications are divided into the subgenres:
  - **Book**: specifically represents a book. Apart from the publication attributes, it is also characterized by an optional ISBN, an optional volume and a series. A book can be carried out in a school (`Carried_out_in_book`) and it is published by a publisher (`Published_by_book`).

  - **Journal article**: specifically represents a journal article. It is published in a journal edition (`Published_in2`), specifically in a certain interval of pages.
  - **Conference and workshop paper**: specifically represents a conference or workshop paper. It is presented in a conference edition (`Presented_in`), specifically in a certain volume and in a certain interval of pages, and it is edited by an editor (`Edited_by_paper`).
  - **Part in book or collection**: specifically represents a part of a book or collection. It is part of a book (`Part _of`), specifically it is a certain interval of pages of that book, and it is edited by an editor (`Edited_by_part`).
  - **Editorship**: specifically represents an editorship. Apart from the publication attributes, it is also characterized by a series, a URN, a volume and an optional ISBN. An editorship is published by a publisher (`Published_by_editorship`).
  - **Reference work**: specifically represents a reference work. Apart from the publication attributes, it is also characterized by a book title which it refers to, a certain interval of pages and the book publisher. A reference work is published by a publisher (`Published_by_reference`) and it is edited by an editor (`Edited_by_reference`).
  - **Informal publication**: specifically represents an informal publication. Apart from the publication attributes, it is also characterized by the E print and the E print type. It can also be published in multiple books (`Published_in_informal`), it can be edited by an editor (`Edited_by_informal`) and it can be published in a journal edition (`Published_in1`).

- **Author**: represents the author who wrote the publication (`Written_by`). It is a type of person. Apart from the person attributes, he is characterized by a biography. He is affiliated to an organization (Affiliation).
- **Organization**: represents the organization to which the author is affiliated (Affiliation). It is identified by a name (primary key), which is considered to be unique.
- **Journal**: represents a journal. It is identified by a title (primary key), which is considered to be unique. Since a journal can publish different editions (`Publish`) of the journal, this element is represented with a weak entity:
  - **Journal edition**: represents a single journal edition, characterized by a volume and an optional number. In a journal edition, a journal article (`Published_in2`) or an informal publication (`Published_in1`) are published.

- **Conference edition**: the conference edition where a conference and workshop paper are presented (`Presented_in`). It is identified by a book title (primary key) and an optional series. A conference edition is published by a publisher (`Published_by_conference`).
- **Editor**: represents the editor of a conference and workshop paper (`Edited_by_paper`), part in book or collection (`Edited_by_part`), reference work (`Edited_by_reference`) or informal publication (`Edited_by_informal`). An editor is a type of person.
- **Publisher**: represents the publisher of a book (`Published_by_book`), thesis (`Published_by_thesis`), conference edition (`Published_by_conference`), editorship (`Published_by_editorship`) or reference work (`Published_by_reference`). It is identified by a name (primary key), which is considered to be unique.
- **School**: represent a school where a thesis (`Carried_out_in_thesis`) or a book (`Carried_out_in_book`) have been carried out in. It is characterized by a name (primary key) which is considered to be unique
- **Person**: represents a generic person, who can be an editor, an author or both of them (overlapping). It is characterized by a ORCID (primary key) and a name.

3 | Neo4j

3.1. Creation of the database

3.1.1. Data set

The data has been taken from the `dblp` database provided by the organization as a `XML` file and a `DLD` file, which formally explains the `XML`'s structure. Since the original `XML` contained a huge number of records, it has been divided into smaller files and reconstructed by taking some pieces in order to have the largest variety of data possible. The resulting subset contains roughly 113000 nodes.
The detailed description of the data set, provided by the `dblp` organization can be found at `https://dblp.org/xml/docu/dblpxml.pdf`.

3.1.2. Data importation in Neo4j

The `XML` file has been translated into a `CSV` file, an useful format, especially for data importation in Neo4j. It has been done by using a dedicated python script built specifically to translate the `dblp XML` database (link: `https://github.com/ThomHurks/dblp-to-csv`). By using the `DLD` file, this script is able to split the data in different `CSV` files, one for each entity. This way, the data gets separated from the headers, making easier to import and create sets of files representing the relationships. The script also assigns a unique ID to each record, in order to recognize the elements in the files.
The usage of the python script in order to translate the files format makes the import of data in Neo4j a lot easier to perform, by only generating the following command:

```
neo4j-admin import --database=dblp --delimiter ";" --array-delimiter "|" --id-type INTEGER
--nodes=Publication:Book="output_book_header.csv,output_book.csv"
--nodes=Publication:Thesis="output_phdthesis_header.csv,output_phdthesis.csv"
--nodes=Publication:Thesis="output_mastersthesis_header.csv,output_mastersthesis.csv"
--nodes=Publication:Editorship="output_proceedings_header.csv,output_proceedings.csv"
--nodes=Publication:Article="output_article_header.csv,output_article.csv"
--nodes=Publication:Book_part="output_incollection_header.csv,output_incollection.csv"
--nodes=Publication:Conference_paper="output_inproceedings_header.csv,output_inproceedings.csv"
--nodes=Publication="output_www_header.csv,output_www.csv"
--nodes=Publisher="output_publisher.csv"
--relationships=published_by="output_publisher_published_by.csv"
--nodes=Person:Editor="output_editor.csv"
--relationships=edited_by="output_editor_edited_by.csv"
--nodes=Journal="output_journal.csv"
--relationships=published_in="output_journal_published_in.csv"
--nodes=Person:Author="output_author.csv"
--relationships=written_by="output_author_written_by.csv"
--nodes=School="output_school.csv"
--relationships=submitted_at="output_school_submitted_at.csv"
```

Figure 3.1: Importation command.

To import the data, it is necessary to put the `CSV` files, generated by the python script, in the `bin` in folder of the DBMS where the database will run. After the command has been run, a database called `dblp` is created. When it is opened, the database is already populated with the specified nodes and relationships.

3.1.3. First schema

Below is the representation of the schema loaded in Neo4j:

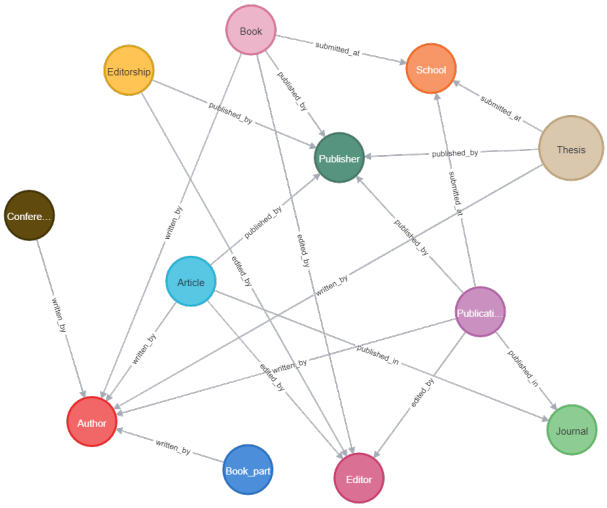


Figure 3.2: First schema.

The data set only contains a list of publications, the nodes that don't represent publication only contain a single attribute, since they had to be extracted from the publications' data. The nodes that represent publications contain all the data present in the data set. It is possible to identify the following nodes:

- **Article**: corresponds to the entity "`journal article`" and "`Informal publication`" in the ER diagram.
- **Author**: corresponds to the entity "`author`" in the ER diagram.
- **Book**: corresponds to the entity "`book`" in the ER diagram.
- **Book part**: corresponds to the entity "`part in a book or collection`" in the ER diagram.
- **Conference paper**: corresponds to the entity "`conference and workshop paper`" in the ER diagram.
- **Editor**: corresponds to the entity "`editor`" in the ER diagram.
- **Editorship**: corresponds to the entity "`editorship`" the ER diagram.
- **Journal**: corresponds to the entity "`journal`" in the ER diagram.
- **Publication**: corresponds to the entity "`publication`" in the ER diagram.
- **Publisher**: corresponds to the entity "`publisher`" in the ER diagram.
- **School**: corresponds to the entity "`school`" in the ER diagram.
- **Thesis**: corresponds to entity "`thesis`" in the ER diagram.

To represent inheritance in the graph, the interested nodes have the label of both the parent and the child. For example, an article is both `Article` and `Publication`.
Regarding relationships between nodes, they are shown in the following table:

Relationship	Start node	End node
written by	Article	Author
	Conference_paper	
	Book	
	Book_part	
	Thesis	
	Publication	
Edited_by	Article	Editor
	Editorship	
	Book	
	Publication	
Published_in	Article	Journal
	Publication	
Published_by	Article	Publisher
	Editorship	
	Book	
	Thesis	
	Publication	
Submitted_at	Thesis	School
	Book	
	Publication	

Table 3.1: Relationships

3.1.4. Differences with the ER diagram

Due to the limitations imposed by the chosen data set, there are some inevitable differences between the ER diagram and the graph implementation. The main difference is the absence of some entities:

1. Organization;
2. Journal edition: to compensate for the absence of this entity, the relationships that were connected to it are now instead routed directly to the Journal entity;
3. Reference work;

4. `Conference edition`: the same approach used to the `Journal edition` also applies here.

5. `Journal article` and `Informal publication`: this two entities have been combined in the more general `Article` label;

3.2. Queries

In this section, we provide a brief description of the queries and their output by showing some screenshots of the result.

3.2.1. Data creation and update

Query 1

The following query merges the duplicates `Authors` and `Editors` (i.e. nodes with the same name).
First, in the `match` clause, we select the pattern to be found in the graph is selected, and in the `where` clause the `Authors` and `Editors` with the same name are merged in the same node using the `set` operator. To conclude the operation, the editors that have been merged and deleted from the graph are matched.

```
1 match(a:Author), (e:Editor), (p:Publication)-[]→(e) where a.author = e.editor set a:Author:Editor
  create (p)-[r:edited_by]→(a)
2 match(a:Author), (e:Editor) WHERE a.author = e.editor detach delete e
```

Figure 3.3: Query 1.

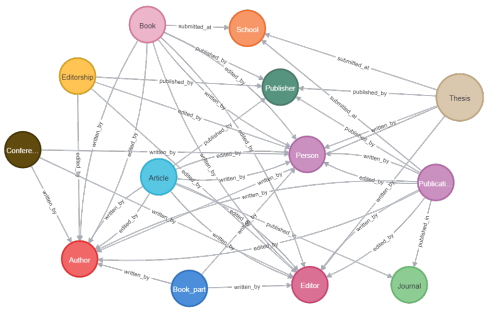
Figure 3.4: Query 1: result.

Query 2

The following query creates the generalization `Person` for `Authors` and `Editors`.
For `Author` and `Editor` labels, the query adds an additional label Person using the `set` operator.

```
1 MATCH(a:Author) SET a:Author:Person Return a
2 MATCH (n:Editor) set n:Editor:Person return n
```

Figure 3.5: Query 2.



3

Figure 3.6: Query 2: result.

Query 3

The following query searches for two authors that have written two or more articles together, and creates a relationship `friend_with` between them.
In the `match` clause the pattern to be searched in the graph is specified: in this case `Authors` that have a `Publication` in common are searched. In the `where` clause are filtered the matches with the same `Article` or the same `Author`. The condition `id(a1)>id(a2)` is added to avoid the creation of two relationships between the same nodes. With the `merge` clause the relationships `friend_with` are created between the selected `Authors`.

```
1 match (a1:Author)<-[r1:written_by]-(p1:Publication)-[r2:written_by]->(a2:Author)<-[r3:written_by]-
  (p2:Publication)-[r4:written_by]->(a1)
2 where p1.title <> p2.title and a1.author <> a2.author and id(a1) > id(a2) merge (a1)-
  [r:friend_with]-(a2) return a1, a2
```

Figure 3.7: Query 3.

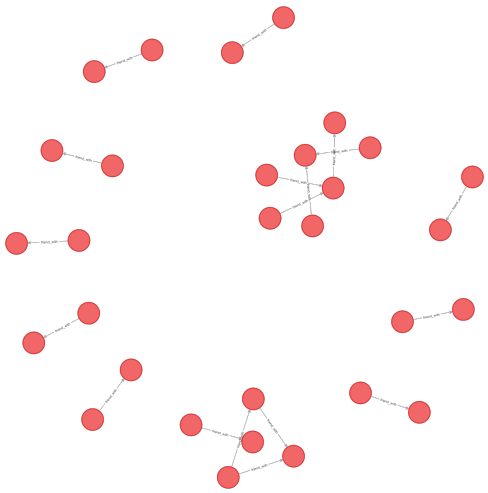


Figure 3.8: Query 3: result.

Query 4

The following query creates the relationship `part_of` between `Book_part` and `Book`, and the relationship `part_of_editorship` between `Conference_paper` and `Editorship`.
In the `match` clauses the desired nodes are selected and in the `where` clauses they are filtered with respect to the key of the `Editorships` and `Books`. In the `merge` clauses the new relationships are created.

```
1 match(c:Conference_paper), (e:Editorship)
2 where e.key = c.crossref
3 merge (c)-[:part_of_editorship]→(e)
4 match(bp:Book_part) match (b:Book)
5 where bp.crossref = b.key
6 merge (bp)-[:part_of]→(b)
7 return bp, b
```

Figure 3.9: Query 4.

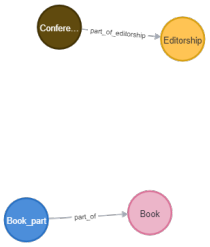


Figure 3.10: Query 4: result.

Query 5

The following query creates and an `Article` written by them.
In the first part of the query some new `Authors` are created, followed by the creation of relationships between them. An `Article` is created and linked with its `Authors` by the `written_by` relationships.

```
create(a1:Author {author: "Leonardo Cesani"})
create(a2:Author {author: "Matteo Paraboschi"})
create(a3:Author {author: "Fausto Lasca"})
create(a4:Author {author: "Alessio Buda"})
create(a5:Author {author: "Gabriele Munafò"})

create(a1)-[:friend_with]→(a2)
create(a1)-[:friend_with]→(a3)
create(a1)-[:friend_with]→(a4)
create(a1)-[:friend_with]→(a5)

create(a2)-[:friend_with]→(a3)
create(a2)-[:friend_with]→(a4)
create(a2)-[:friend_with]→(a5)

create(a3)-[:friend_with]→(a4)
create(a3)-[:friend_with]→(a5)

create(a4)-[:friend_with]→(a5)

create(art:Article {article: 25942,
authors: ["Leonardo Cesani", "Matteo Paraboschi", "Fausto Lasca", "Alessio Buda", "Gabriele Munafò"],
title: "Project Work 2022/2023, System and Methods for Big and Unstructured Data", year: 2022})
create(art)-[:written_by]→(a1)
create(art)-[:written_by]→(a2)
create(art)-[:written_by]→(a3)
create(art)-[:written_by]→(a4)
create(art)-[:written_by]→(a5)
return a1, a2, a3, a4, a5, art
```

Figure 3.11: Query 5.

Figure 3.12: Query 5: result.

3.2.2. Data manipulation

Query 6

The following query finds the shortest path in the graph between two given `Authors`.
First, in the `match` clauses, two `Authors` are selected and, using the function `shortestPath`, the shortest path between the two authors is found.

```
1 match (a:Author {author: "Javier Gozálvez"}),
2 match(b:Author {author : "Harvey Glickenstein"}),
3 p = shortestPath((a)-[*]-(b))
4 return p
```

Figure 3.13: Query 6.

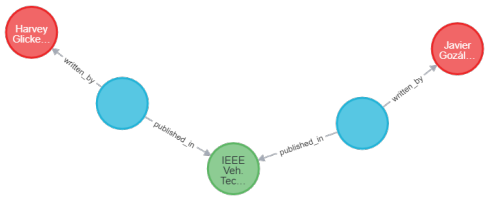


Figure 3.14: Query 6: result.

Query 7

The following query finds the `Author` with the highest number of publications published in a `Journal`.
In the `match` clause, the pattern that links a `Person` to a `Journal` is specified. In the `with` clause, `Publications` linked to both `Author` and `Journal` are collected, and the number of relationships incoming to each `Journal` are counted using the `count` function. In the `return` clause the `Author` with the most number of `Publications` in a `Journal`, his `Publications` and the `Journal` are returned. In the table, the first 3 results of the query are reported; to improve the readability, only the `Author`'s name and `Journal`'s name are reported.

```
1 match (p:Person)←[:written_by]-(c:Publication)-[r1:published_in]→(j:Journal)
2 with p, j, count(r1) as in_journal, collect(c) as publication
3 RETURN p, j, publication, in_journal order by in_journal desc limit 1
```

Figure 3.15: Query 7.

Table | Text | Code

	p.author	j.journal	in_journal
1	"Lajos Hanzo"	"IEEE Trans. Veh. Technol."	99
2	"Christoph Meinel"	"Universität Trier, Mathematik/Informatik, Forschungsbericht"	54
3	"Harvey Glickenstein"	"IEEE Veh. Technol. Mag."	51

Figure 3.16: Query 7: table.

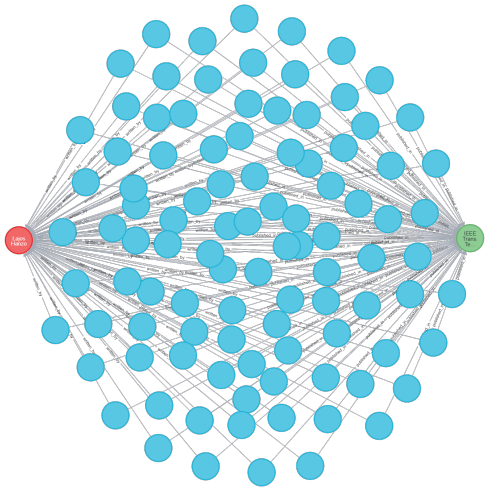


Figure 3.17: Query 7: result.

Query 8

The following query extracts the `Publications` of two `Authors` who are colleagues and have the highest sum of `Publications`.

With the first two `match` clauses, we select two authors with their respective publications. The `with` operator allows us to perform aggregate operations, namely to `count` the number of publications for each of the two authors, to rename fields (`as`), and to form a list composed of all publications (`collect`). Then, for each of the two authors, we check that the number of publications is greater than 30, for performances reason. By doing so, we exclude authors with fewer publications, making the query more efficient. Moreover, the condition on the IDs in the second `where` is added to make the query more efficient by

avoiding considering already visited nodes. To conclude, only authors that are colleagues (`friend_with`) are considered by the third `match` condition. The result is ordered based on the sum of publications of the two authors and only the first record, with the highest number of publications, is returned. In the table the results of the query are reported; for readability reasons, only the name of the `Author`s involved and the joint number of `Publication`s are reported.

```
1 match (p1:Person)<-[w1:written_by]-(pub1:Publication)
2 with p1, count(w1) as n_pub1, collect(pub1) as publication1
3 where n_pub1 > 30
4 match (p2:Person)<-[w2:written_by]-(pub2:Publication)
5 with p1, n_pub1, publication1, p2, count(w2) as n_pub2, collect(pub2) as publication2
6 where id(p1) > id(p2) and n_pub2 > 30
7 match (p1)-[:friend_with]-(p2)
8 return p1, p2, publication1, publication2, n_pub1 + n_pub2 order by n_pub1 + n_pub2 desc limit 1
```

Figure 3.18: Query 8.

Table
Text
Code

	p1.author	p2.author	n_pub1 + n_pub2
1	"Zhu Han 0001"	"Lajos Hanzo"	160
2	"Victor C. M. Leung"	"Lajos Hanzo"	157

Figure 3.19: Query 8: table.

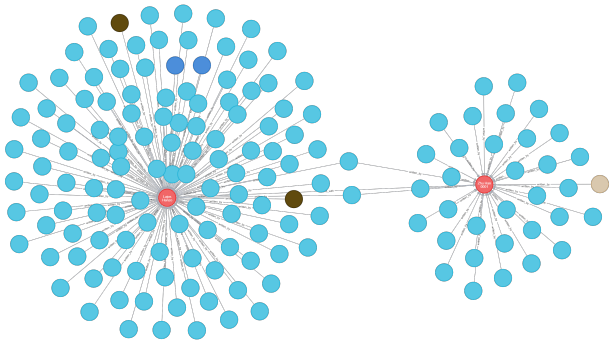


Figure 3.20: Query 8: result.

Query 9

The following query finds the school with the highest number of `Publications` authored by students whose name starts with the letter 'X'.
First, in the `match` clause, the pattern that links a `School` to a `Person` is specified and, in the `where` clause, the `Authors` are filtered with respect to their names. In the `with` clause the number of `Publications` are counted using the `count` function and the `Person` and `Publications` are collected using the `collect` function. In the `return` clause, the output is sorted in descending order with respect to the counted `Publications`, and the output is limited to the first match to get the `School` with the highest number of `Publications`.

```
1 match (per:Person)<-[]-(pub:Publication)-[s:submitted_at]->(school:School)
2 where per.author starts with 'X'
3 with school, count(s) as submissions, collect(pub) as publications, collect(per) as people
4 return school, people, publications, submissions order by submissions desc limit 1
```

Figure 3.21: Query 9.

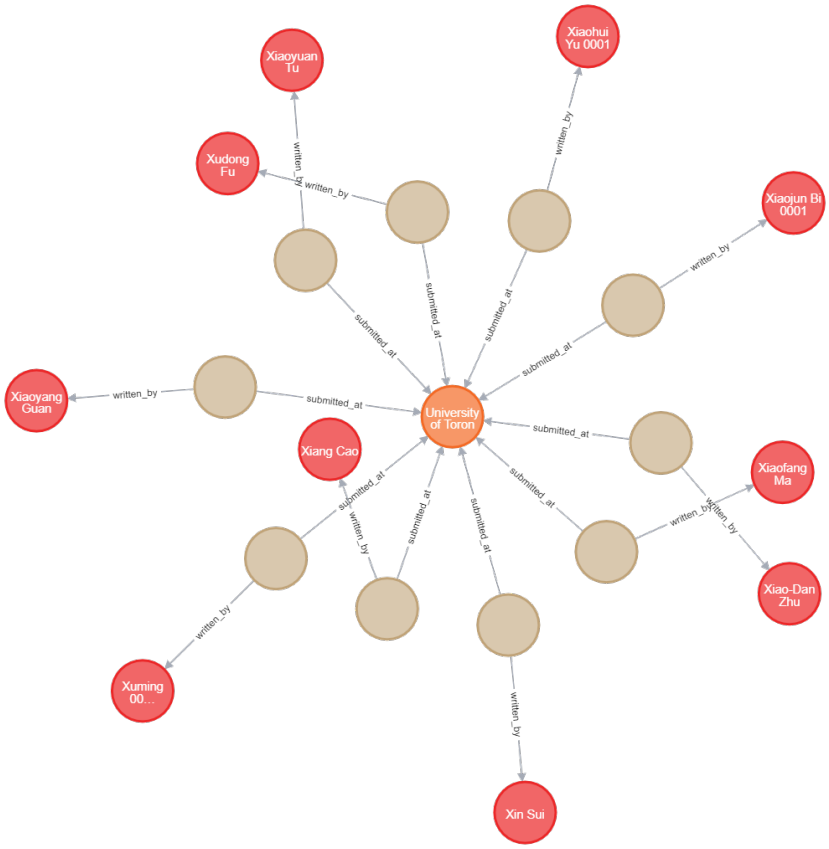


Figure 3.22: Query 9: result.

Query 10

The following query extract all the 2-level colleagueships of the `Author` named "Lajos Hanzo".

In `match` condition, all people distant at most two steps from the author Lajos Hanzo, considering relationships of the type `friend_with`, are selected. Then, the `where` condition checks that the second person (`p2`) is different from the first one: using the `where`, we avoid the possibility that an author could be colleague of another author who is colleague to the first one (`p1`). Finally, the two authors are returned.

```
1 match(p1:Person {author : "Lajos Hanzo"})-[:friend_with*..2]-(p2:Person)
2 where id(p1) <> id(p2)
3 return p1, p2
```

Figure 3.23: Query 10.

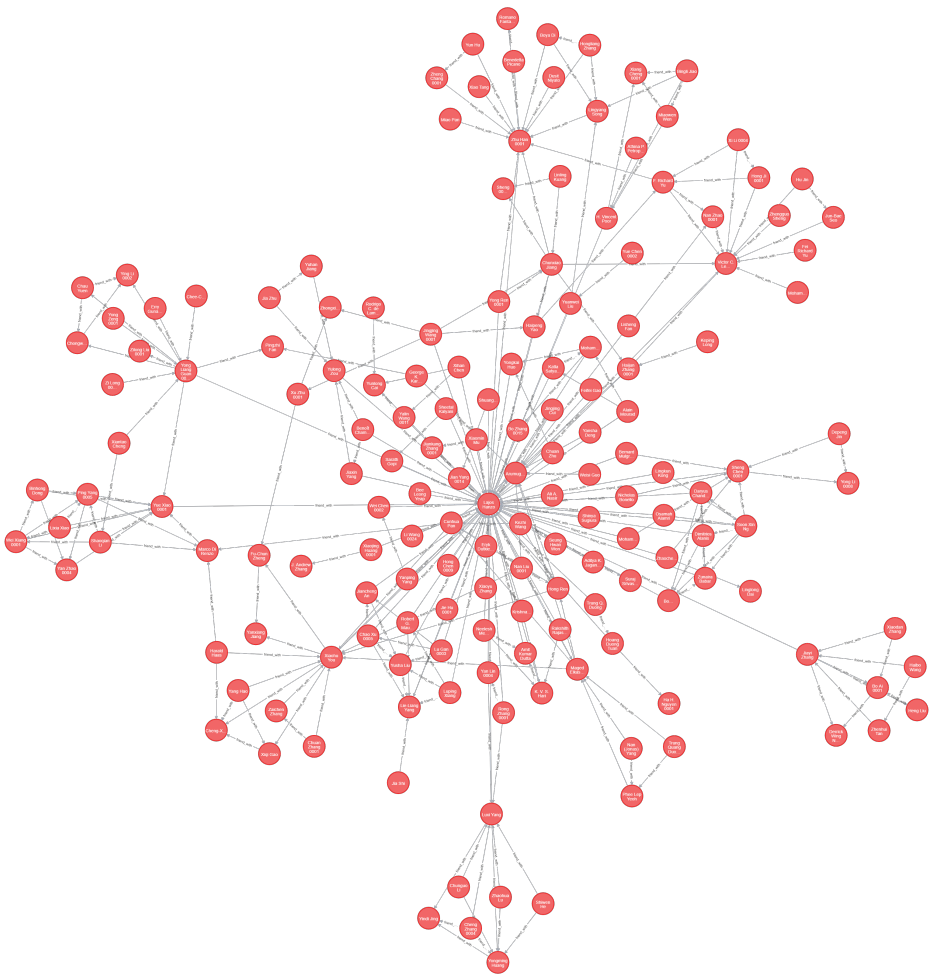


Figure 3.24: Query 10: result.

Query 11

The following query finds the book that has been written by the highest number of people, its parts and `Authors`.

With the `match` clause, all people who have written a book part, the book part and corresponding book are selected. Then, in the `with` clause, for each book, count the number of different people who have written parts of it (`authors`), collect the book parts and authors. We also need to include `b` in the `with` in order to be able to return it. Finally, the result is ordered in decreasing order based on the number of authors, and only the first element, thus the book with the most authors, is returned. In the table the first 3 results of the query are reported; for readability, only the name of the `Book` and the number of `Authors` are reported.

```
1 match (p:Person)←[r:written_by]-(bp:Book_part)-[r1:part_of]→(b:Book)
2 with b, count(distinct(p)) as authors,  collect(bp) as book_part, collect(p) as people
3 return b, book_part, people, authors order by authors desc limit 1
```

Figure 3.25: Query 11.

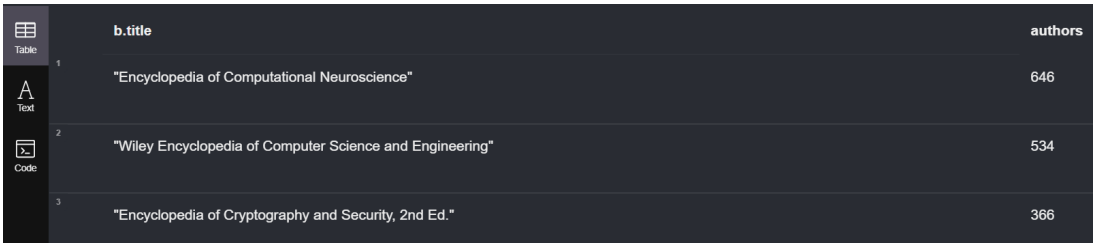


Figure 3.26: Query 11: table.

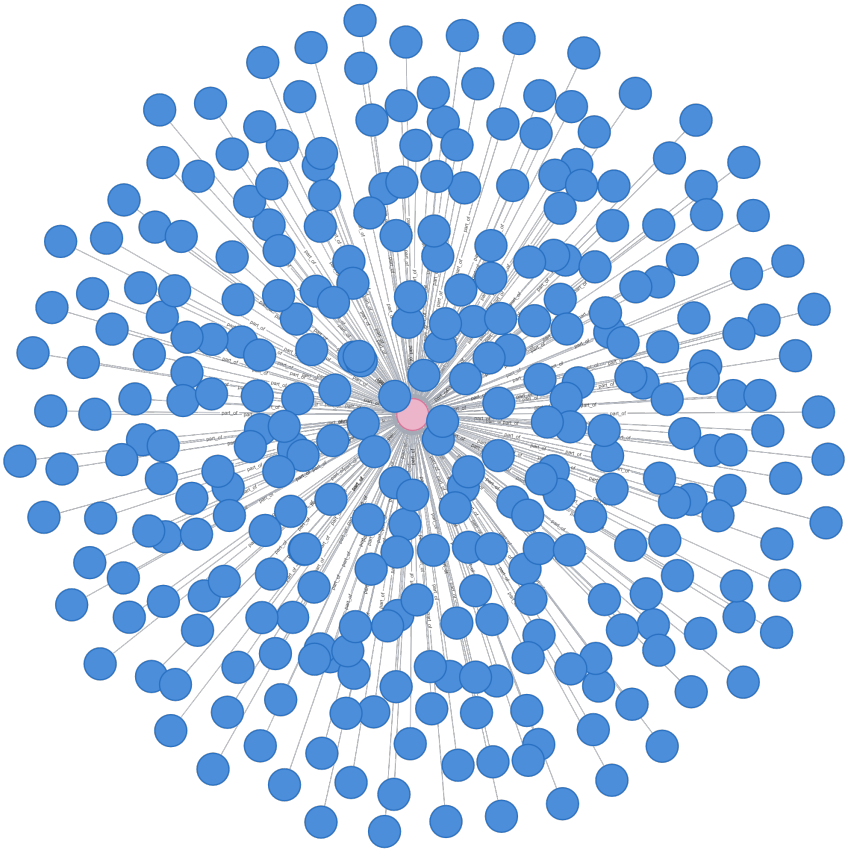


Figure 3.27: Query 11: result.

Query 12

The following query finds the `Authors` that have published a `Conference_paper` in 2006, the `Conference_paper` and the related `Editorship`.

The `match` clause selects all the people that have written a conference paper and the relative editorship. Then, with the `where` clause we select only editorships that have been written in 2006. Finally, the authors, conference papers and editorships are returned.

```
1 match (p:Person)←[:written_by]-(c:Conference_paper)-[:part_of_editorship]→(e:Editorship)
2 where e.year = 2006
3 return p, c, e
```

Figure 3.28: Query 12.

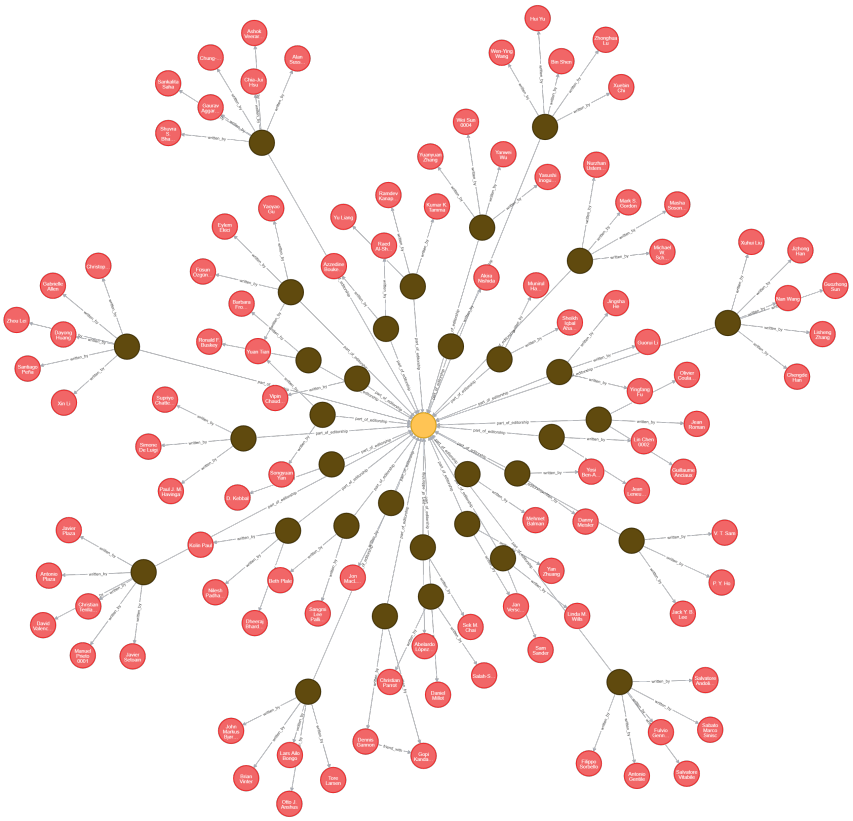


Figure 3.29: Query 12: result.

Query 13

The following query searches for the `Author` with the most `Publications` published in a `Journal` in a single year.

The `match` clause selects all authors that have written a publication, the publication and the journal in which it has been published. Then, we `count` the number of publications by year, we sort the result in decreasing order and return the first record.

```
1 MATCH (a:Author)<-[:written_by]-(p:Publication)-[:published_in]->(Journal)
2 return a.author,  p.year, count(p) as c  order by c desc limit 1
```

	a.author	p.year	c
1	"Lajos Hanzo"	2021	13

Figure 3.30: Query 13.

Query 14

This query returns the journal articles published by `Authors` that submitted a `Thesis` at the Polytechnic University of Milan.

In the `match` clause, we select: the `School` with the name "Polytechnic University of Milan, Italy"; the pattern that links it to the `Journal` that have published some `Articles` written by students of the selected `School`. To improve the readability of the output, only `School`, `Thesis`, `Author` and `Article` are reported.

```
1 match (s:School {school:"Polytechnic University of Milan, Italy"})←[:submitted_at]-(t:Thesis)-
  [:written_by]→(au:Author)←[:written_by]-(ar:Article)-[:published_in]→(j:Journal)
2 return s, t, au, ar
```

Figure 3.31: Query 14.

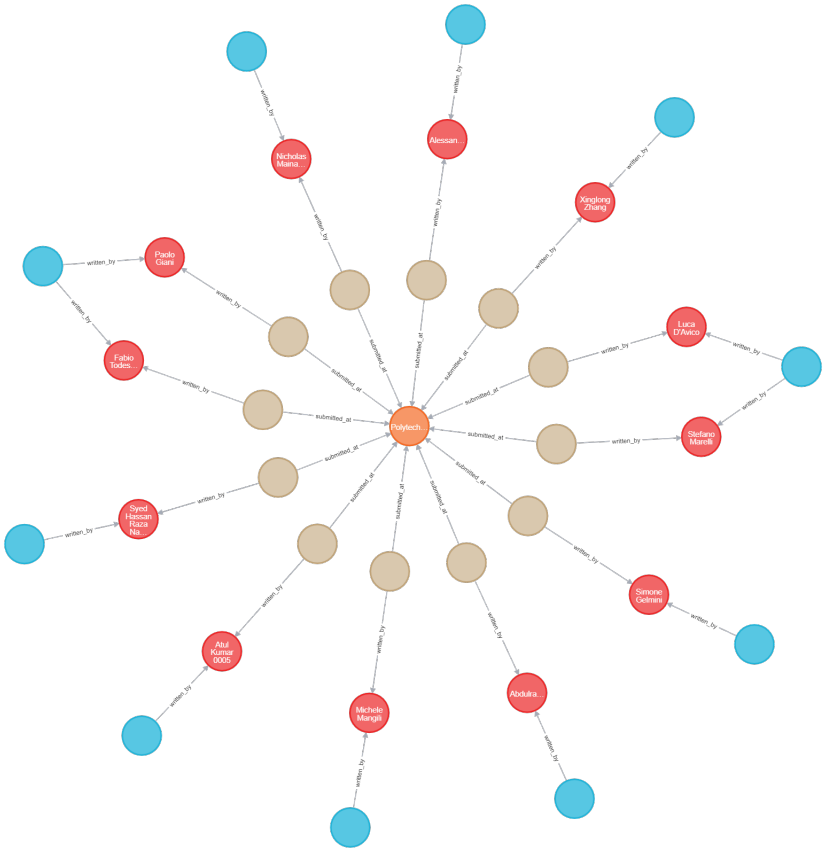


Figure 3.32: Query 14: result.

Query 15

This query returns a list of the Journals where `Articles`, written by `Authors` that submitted their thesis at the Polytechnic University of Milan, have been published. The list is ordered by the number of articles that have been published in each journal.
After specifying the path from a `Publication` to a `Journal`, in the `where` clause, matches are filtered with respect to the following `exists` condition: only the `Publications` written by `Authors` that have submitted a `Thesis` to the Polytechnic University of Milan are

considered. The output is composed of the names of the `Journal` and the number of publications, sorted in descending order.

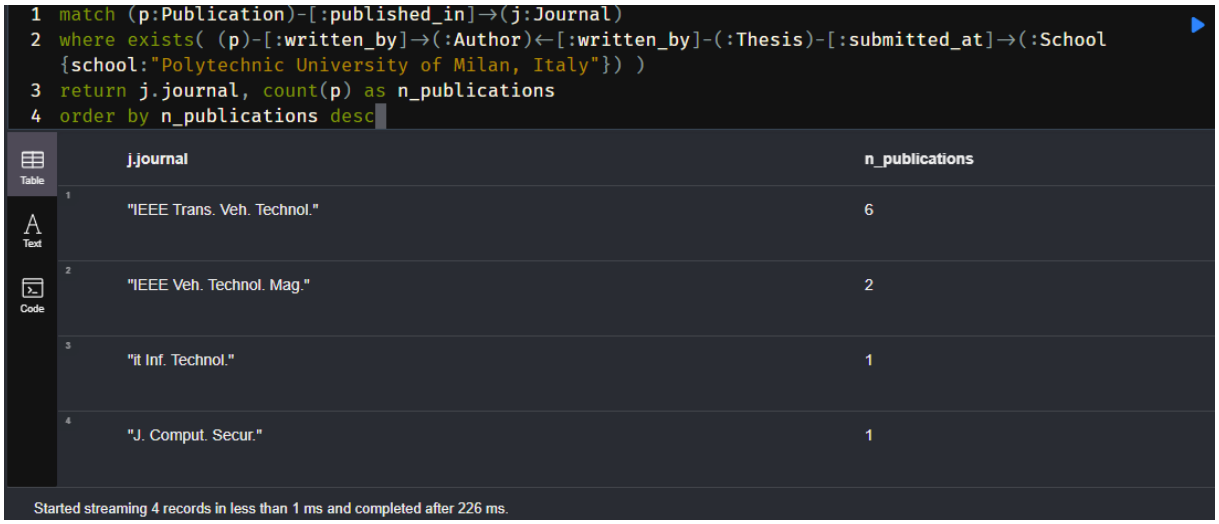


Figure 3.33: Query 15.

4 | MongoDB

4.1. Document structure

Regarding the document structure, we followed the assignment's specification, including:

- `title`: the article's title;
- `authors`: a vector of embedded documents, each representing one of the authors of the article. Each author is described by the following attributes: name, affiliation, email and biography;
- `keywords`: a vector of strings representing the article's keywords;
- `journal`: an embedded document representing the journal edition in which the article has been published. For the journal the following fields are present: name of the journal, volume, number, the date of publication and the pages in which the article may be found;
- `abstract`: representing the article's abstract;
- `sections`: an array of documents representing the sections of the article. Each section has a title and one or more paragraphs, represented by an array of strings. It may also have one or more figures with the relative captions, represented as documents with two fields: url of the image and caption. Moreover, each section may also contain subsections. These are presented as an array of documents. Their structure is similar to that of a section with the only difference that a subsection cannot contain other subsections.
- `bibliography`: references to other documents (articles) cited in the present article.

4.1.1. Example of a document

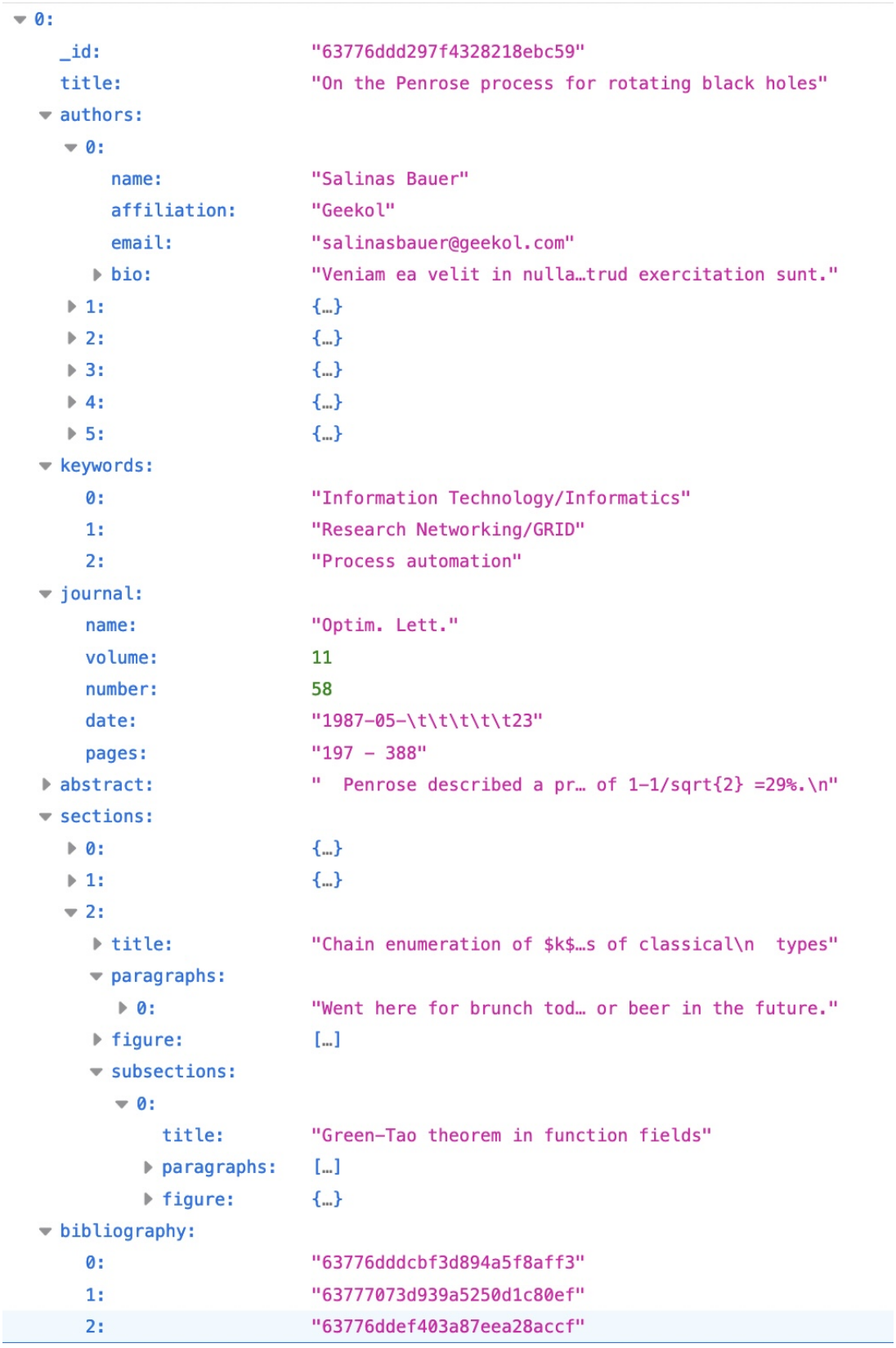


Figure 4.1: Example of a document.

Figure 4.1 shows an example document. It has a unique identifier `_id`, a title, an array of authors, an array of keywords, the details of the journal edition on which it has been published, an abstract, an array of sections and potentially subsections, a bibliography array contaning `_id`s of articles cited in the document. For every author, journal, section and subsection relevent information is provided in the form of embedded documents with all the relevant field described Section 4.1.

4.1.2. Differences with the ER model

There are no relevant differences between the attributes of the entity `article` defined in the ER model and the fields of the article document as described in section 4.1.
Minor differences are:

- the `DOI` attribute of the **Publication** entity is now replaced by a unique identifier `_id`;
- the `title` of the journal in the ER model is now referred to as `name` of the journal;
- the `year` attribute of the **Publication** entity inherited by the **Article** entity becomes the `date` of publication of the journal edition;
- the following non relevant attributes of the **Publication** entity are not present in the document structure: `URL`, `timestamp`, `bib_source` and `bib_url`;
- the **Author** has an `email` associated to him, which was not present in the ER model.

4.2. Data upload and transformation

4.2.1. Creation of the dataset

To create the dataset to import in MongoDB we proceeded as follows.
First, we defined the structure of the document according to what was specified in the delivery of the project, as described in Section 4.1.
Then, we used the online tool `https://json-generator.com/` to create three JSON files of 2000 records, each with a slight variation of the defined structure: the types of documents only differ for the number of sections or the presence (or absence) of subsections and figures (for the structure of the three types of documents see Appendix A).
To assign values to each field, we used functions provided by the tool for the generation of random content. For example, we generated random names, company names and emails

to fill the data of the different authors. Then, we used functions that generated content from the *Lorem ipsum* text to fill more complex fields, where a longer portion of text was required, such as the body of the different paragraphs.
This, however, poses a problem: the content of the articles is not varied enough, with many articles similar to each other.

4.2.2. Data modification

To solve this issue we decided to include text coming from different sources, selected among those suggested in the project delivery.
In particular, to fill in the data relative to keywords we use a list scientific terms taken from the official website of the Chamber of Commerce of Genoa. As journal names, articles and paragraphs titles, we used those present in the `dblp` database, used in Chapter 3. To complete the abstracts we used a dataset from the Kaggle website, containing data from the ArXiv dataset: in this particular case we selected only the relevant data for our application. In the end, paragraphs have been taken from the Yelp review dump. The data have been inserted into separate `CSV` files.

Python script

To substitute the fields mentioned in Section 4.2.2, we wrote a simple Python script which opens the `CSV` files containing data to be substituted and writes them in the original `JSON` file. The used script may be found in Appendix B.

4.2.3. Data upload

The generated dataset has been imported in MongoDB by using the built-in feature provided by MongoDB Compass.
A new database was created and then, inside it, a new collection. The data has been inserted in the created collection by using the import feature of MongoDB Compass (*ADD DATA*), selecting `JSON` as the *input file type* and the three documents obtained using the Python script as *files* to be imported.

4.3. Queries

4.3.1. Data creation and update

Query 1: Insertion of an article

The following query adds an article to the `articles` collection. The inserted article contains all the fields defined in Section 4.1.

```
Atlas atlas-l7mawr-shard-0 [primary] journal_articles> db.articles.insertOne({
  "title": "SMBUD Project",
  "authors": [
    { "name": "Alessio Buda",
      "affiliation": "Politecnico di Milano",
      "email": "alessio.buda@mail.it",
      "bio": "Alessio's biography"
    },
    { "name": "Leonardo Cesani",
      "affiliation": "Politecnico di Milano",
      "email": "leonardo.cesani@mail.it",
      "bio": "Leonardo's biography"
    }
  ],
  "keywords": ["Big data", "University", "Project"],
  "journal": {
    "name": "Frontiers in Data Science",
    "volume": 1,
    "number": 12,
    "date": ISODate('2022-11-19T00:00:00.000Z'),
    "pages": "123-456",
  },
  "abstract": "Article's abstract",
  "sections": [{
    "title": "Section's title",
    "paragraphs": ["Text 1", "Text 2", "Text 3"]
  }],
  "bibliography": [ObjectId('63776dddd15bc88e6627df47')]
})
```

Figure 4.2: Query 1.

```
{ acknowledged: true,
  insertedId: ObjectId("63791cd1d3b5ae78ecabcd4a") }
> db.articles.find({"_id": ObjectId("63791cd1d3b5ae78ecabcd4a")})
{ _id: ObjectId("63791cd1d3b5ae78ecabcd4a"),
  title: 'SMBUD Project',
  authors:
   [ { name: 'Alessio Buda',
       affiliation: 'Politecnico di Milano',
       email: 'alessio.buda@mail.it',
       bio: 'Alessio\'s biography' },
     { name: 'Leonardo Cesani',
       affiliation: 'Politecnico di Milano',
       email: 'leonardo.cesani@mail.it',
       bio: 'Leonardo\'s biography' } ],
  keywords: [ 'Big data', 'University', 'Project' ],
  journal:
   { name: 'Frontiers in Data Science',
     volume: 1,
     number: 12,
     date: 2022-11-19T00:00:00.000Z,
     pages: '123-456' },
  abstract: 'Article\'s abstract',
  sections:
   [ { title: 'Section\'s title',
       paragraphs: [ 'Text 1', 'Text 2', 'Text 3' ] } ],
  bibliography: [ ObjectId("63776dddd15bc88e6627df47") ] }
```

Figure 4.3: Query 1: result.

Query 2: Insertion of an article with missing fields

The following query inserts an article in the `articles` collection. The inserted article does not contain fundamental fields such as `authors` and `journal`. This shows how a document-based database can be flexible and does not require a rigid structure for the articles that are being added.

```
Atlas atlas-1x21pr-shard-0 [primary] SMBUD_Database> db.Articles.insertOne({
                                                      "title":"Test_article",
                                                      "sections":[{
                                                        "title":"Test_section",
                                                        "paragraphs":[
                                                          "Example of section text"
                                                        ],
                                                        "subsections":[{
                                                          "title":"Test_subsection",
                                                          "paragraphs":[
                                                            "Example of subsection text"
                                                          ]
                                                        }]
                                                      }]
                                                    })
```

Figure 4.4: Query 2.

```
{ _id: ObjectId("63794ef4499a81e5ffd91bdc"),
  title: 'Test_article',
  sections:
   [ { title: 'Test_section',
       paragraphs: [ 'Example of section text' ],
       subsections:
        [ { title: 'Test_subsection',
            paragraphs: [ 'Example of subsection text' ] } ] } ] }
```

Figure 4.5: Query 2: result.

Query 3: Update an abstract of an article

The following query modifies the value of the attribute `abstract` in an existing document. After selecting the article using its `_id`, the `$set` operator is used to modify the `abstract` field.

```
Atlas atlas-1x21pr-shard-0 [primary] SMBUD_Database> db.Articles.updateOne(
                                                      {"_id":ObjectId("63794ef4499a81e5ffd91bdc")},
                                                      {$set:{"abstract":"Example of descrition of an article"}}
                                                      )
```

Figure 4.6: Query 3.

```
{ _id: ObjectId("63794ef4499a81e5ffd91bdc"),
  title: 'Test_article',
  sections:
   [ { title: 'Test_section',
       paragraphs: [ 'Example of section text' ],
       subsections:
        [ { title: 'Test_subsection',
            paragraphs: [ 'Example of subsection text' ] } ] } ],
  authors:
   [ { name: 'Marco Brambilla',
       affiliation: 'Politecnico di Milano',
       email: 'marco.brambilla@example.com',
       bio: 'Teaches theory part of SMBUD' },
     { name: 'Andrea Tocchetti',
       affiliation: 'Politecnico di Milano',
       email: 'andrea.tocchetti@example.com',
       bio: 'Teaches excercise part of SMBUD' } ],
  abstract: 'Example of descrition of an article' }
```

Figure 4.7: Query 3: result.

Query 4: Add authors to an article

The following query adds the attribute `authors` to the article previously created (Query 2) by using the `$set` operator as seen in Query 3. This shows how the `update` operator can be used not only to update already existing information in an existing document, but also to add new information to it.

```
Atlas atlas-lx2lpr-shard-0 [primary] SMBUD_Database> db.Articles.updateOne(
                                                     {"_id":ObjectId("63794ef4499a81e5ffd91bdc")},
                                                     {$set:{"authors":[{
                                                       "name":"Marco Brambilla",
                                                       "affiliation":"Politecnico di Milano",
                                                       "email":"marco.brambilla@example.com",
                                                       "bio":"Teaches theory part of SMBUD"
                                                     },
                                                     {
                                                       "name":"Andrea Tocchetti",
                                                       "affiliation":"Politecnico di Milano",
                                                       "email":"andrea.tocchetti@example.com",
                                                       "bio":"Teaches excercise part of SMBUD"
                                                     }]
                                                     }})
```

Figure 4.8: Query 4.

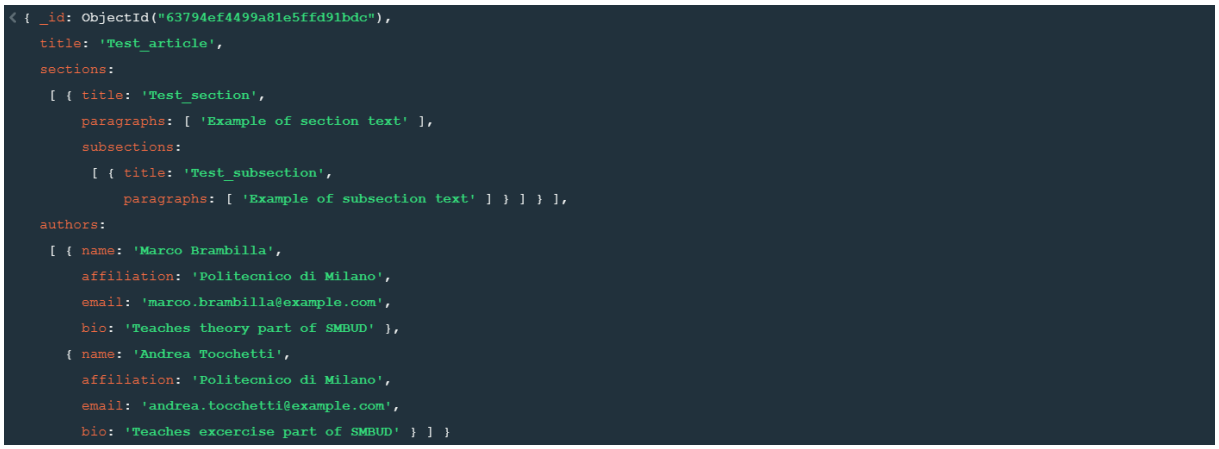


Figure 4.9: Query 4: result.

Query 5: Add a section to an article

The following query adds a section to an article, for instance the one created in Query 1. After matching the article using its `_id`, the `sections` array is accessed using the dot notation (`sections.1`). Since there is only one section in the array (with index equal to 0), a new section is created. For the created sections the `title` and `paragraphs` are specified as fields of an embedded document.
Figure 4.10 shows the document before the update, the query is presented in Figure 4.11. Finally, the result of the query can be seen in Figure 4.12.

Figure 4.10: Query 5: before.

```
Atlas atlas-l7mawr-shard-0 [primary] journal_articles > db.articles.updateOne({
                                                          "_id": ObjectId("637a39089b27565984ad56cf")
                                                        },{
                                                          $set: {"sections.1": {"title": "New section's title", "paragraphs": ["Section text 1"]}}
                                                        })
```

Figure 4.11: Query 5.

```
{ _id: ObjectId("637a39089b27565984ad56cf"),
  title: 'SMBUD Project',
  authors:
   [ { name: 'Alessio Buda',
       affiliation: 'Politecnico di Milano',
       email: 'alessio.buda@mail.it',
       bio: 'Alessio\'s biography' },
     { name: 'Leonardo Cesani',
       affiliation: 'Politecnico di Milano',
       email: 'leonardo.cesani@mail.it',
       bio: 'Leonardo\'s biography' } ],
  keywords: [ 'Big data', 'University', 'Proiect' ],
  journal:
   { name: 'Frontiers in Data Science',
     volume: 1,
     number: 12,
     date: 2022-11-19T00:00:00.000Z,
     pages: '123-456' },
  abstract: 'Article\'s abstract',
  sections:
   [ { title: 'Section\'s title',
       paragraphs: [ 'Text 1', 'Text 2', 'Text 3' ],
       subsections: [ { title: 'Subsection\'s title', paragraphs: [ 'Par text 1' ] } ] },
     { title: 'New section\'s title',
       paragraphs: [ 'Section text 1' ] } ],
  bibliography: [ ObjectId("63776dddd15bc88e6627df47") ] }
```

Figure 4.12: Query 5: result.

Query 6: Add a subsection to an article

The following query adds a subsection to an article, for instance the one one created in Query 1. After matching the article using its `_id`, the first element of the `sections` array is accessed using the dot notation (`sections.0`). In this element, represented by an embedded document, a new `subsections` field is created. The new subsection is added as a document in the array of subsections.
Figure 4.13 shows the document before the update, the query is presented in Figure 4.14. Finally, the result of the query can be seen in Figure 4.15.

Figure 4.13: Query 6: before.

Figure 4.14: Query 6.

Figure 4.15: Query 6: result.

4.3.2. Data manipulation

Query 7: Articles mentioning "ramen"

The following query returns the articles that contain the word "ramen" in their paragraphs, filtering out articles with few authors or sections.

To be precise the query returns only the articles that have: 8 or more authors, 3 or more sections or at least both 6 authors and 2 sections.
The query uses `$exists` to filter the size of the authors and sections arrays, since `$size` only accepts a specific value, not a range of values.
To search for the word "ramen" in the articles, `$text` and `$search` are used. Before executing the query, a textual index on `sections.paragraphs` was created, allowing for the text search.
A projection is also made to make it easier to read the results. Figure 4.17, presenting the results, only shows 2 of the 81 articles returned by the query.

```
Atlas atlas-l7mawr-shard-0 [primary] journal_articles> db.articles.find({
                                                         $or: [
                                                           {"authors.7": {$exists: true}},
                                                           {"sections.2": {$exists: true}},
                                                           {"authors.5": {$exists: true}, "sections.1": {$exists: true}}
                                                         ],
                                                           $text: {$search: "ramen"}
                                                       },{
                                                         "title": 1,
                                                         "authors.name":1,
                                                         "sections.paragraphs": 1
                                                       })
```

Figure 4.16: Query 7.

```
{ _id: '63776dde8b7cb44a3185b6c7',
  title: 'Aspects of Higgs Production at the LHC',
  authors:
   [ { name: 'Briggs Stokes' },
     { name: 'Rosario Barlow' },
     { name: 'Davenport Lamb' },
     { name: 'Palmer Gardner' },
     { name: 'Boyd Hunt' },
     { name: 'Craig Clay' } ],
  sections:
   [ { paragraphs:
       [ 'My wife and I ordered a cheese steak and a thin crust square pizza. The cheese steak was hot, gooey and tasty.  The pizza was crispy and very tasty.  They have off s
         'DELICIOUS!! Had the signature Mac n cheese. IT was good. But I didn\'t care for the truffle oil. I had the salad with the salmon accompanied with the house made dres
         'Nice decors and atmosphere. Cute little details about this place stands out from other ramen places. The ramen was good. I really like the broth that I got and the p
     { paragraphs:
       [ 'Enterprise loves to advertise it\'s $9.99/a day weekend special, but neglects to mention that all locations in or near downtown Philadelphia do NOT participate in th
         'What just happened? Did I just get thrown out?!?! It was 10:20AM and the place was not busy at all! If they don\'t want customers they should put a sign up!!' ] },
     { paragraphs:
       [ 'this place is wonderful. food is delicious and they are very accommodating. Was here for a large party and told them I had food allergies and they made separate plat
         'This place is pretty awesome. Love the ambiance of being in this essentially wooden warehouse type thing. Friendly staff. Nice collection of interesting cocktails. T
{ _id: '63776dde1f4551e2daea9e32',
  title: 'Color Magnetic Flux Tubes in Dense QCD. II: Effective World-Sheet Theory',
  authors:
   [ { name: 'Bridges Goodwin' },
     { name: 'Wells Cleveland' },
     { name: 'Bridgette Figueroa' },
     { name: 'Cooke Becker' },
     { name: 'Hayden Shelton' },
     { name: 'Alfreda Rhodes' },
     { name: 'Jody Klein' } ],
  sections:
   [ { paragraphs: [ 'I absolutely love, love, love this place. The menu changes seasonally and every month there is a new price Fixe menu. The chef is extremely talented. We l
     { paragraphs:
       [ 'Despite being an Asian food devotee, I\'ve never eaten Filipino food before.  I was stoked when we decided to take a day trip out to the Northeast to sample Philly\'
         'Delicious goodness :). Corn fritters hit the spot, and the breading on the chicken is so yummy. Mashed potatoes were average but made for very good dipping for chick
         'This is literally the worst ramen bar I have ever been to. The noodles are not completely cooked and are sticky. The sushi doesn\'t have enough sauce on it. The soy
     { paragraphs:
       [ 'I never write reviews but I had to this time. This is the second time I\'ve tried this place and I will never have it again. The first time, I tried their spaghetti.
         'Good for a quick eat.\n\nIt\'s cheap and the exact same as the uptown version off Claiborn.  I also heard that they are going to start being open until 2am.  Food is
```

Figure 4.17: Query 7: result.

Query 8: Conditions on keywords

The following query finds one document in the dataset that contains the keyword “Radar” and at least one of the keywords “Remote Control” or “Narrow Band Technologies”. The `$or` operator allows to filter based on conditions defined on multiple attributes and to perform a logical OR operation between them. For readability reasons, only the authors, the title and the keywords are displayed.

```
Atlas atlas-l7mawr-shard-0 [primary] journal_articles> db.articles.findOne({
                                                         "$or": [
                                                           {"keywords": "Remote Control"},
                                                           {"keywords": "Narrow Band Technologies"}
                                                           ],
                                                           "keywords": "Radar"
                                                       },{
                                                       "sections":0,
                                                       "journal":0,
                                                       "abstract":0,
                                                       "bibliography":0
                                                       })
```

Figure 4.18: Query 8.

```
{ _id: '63776dddcc75ab6dad8fa5aa',
  title: 'On the evolution of superposition of squeezed displaced number states\n  with the multiphoton Jaynes-Cummings model',
  authors:
   [ { name: 'Randolph Glenn',
       affiliation: 'Exoteric',
       email: 'randolphglenn@exoteric.com',
       bio: 'Tempor dolor aute aliqua esse in ad esse laboris fugiat eiusmod ullamco incididunt enim excepteur.' },
     { name: 'Debbie Glover',
       affiliation: 'Unq',
       email: 'debbieglover@unq.com',
       bio: 'Culpa non nulla consequat sunt aute fugiat.' },
     { name: 'Wilda Benton',
       affiliation: 'Lexicondo',
       email: 'wildabenton@lexicondo.com',
       bio: 'Cillum cillum sunt deserunt pariatur officia eu.' },
     { name: 'Tricia Roberson',
       affiliation: 'Zoid',
       email: 'triciaroberson@zoid.com',
       bio: 'Est nisi consectetur adipisicing adipisicing tempor occaecat nostrud est incididunt aliqua fugiat pariatur reprehenderit.' },
     { name: 'Ann Faulkner',
       affiliation: 'Concility',
       email: 'annfaulkner@concility.com',
       bio: 'Sunt sunt voluptate minim laboris sint qui quis.' },
     { name: 'Leigh Freeman',
       affiliation: 'Bezal',
       email: 'leighfreeman@bezal.com',
       bio: 'Nulla dolor anim veniam aliquip non anim adipisicing.' } ],
  keywords:
   [ 'Speech Processing/Technology',
     'Radar',
     'Narrow Band Technologies' ] }
```

Figure 4.19: Query 8: result.

Query 9: Interesting articles in specific journals

The following query returns the articles published in a IEEE or IBM journal dealing with specific topics. The article topics are filtered using keywords: the only articles shown are the ones that contain both "Information Technology/Informatics" and "Printed circuits

and integrated circuits".
This query uses `$regex` to find the journal titles that contain either "IEEE" or "IBM". A projection is also made to make it easier to read the results.

```
Atlas atlas-l7mawr-shard-0 [primary] journal_articles> db.articles.find({
                                                       keywords: {$all: ["Information Technology/Informatics", "Printed circuits and integrated circuits"] },
                                                       $or: [
                                                         {"journal.name": {$regex: "IEEE"}},
                                                         {"journal.name": {$regex: "IBM"}},
                                                         ]
                                                       },{
                                                       "title": 1,
                                                       "journal.name":1,
                                                       "keywords":1
                                                       })
```

Figure 4.20: Query 9.

```
{ _id: '63776ddd66d412fcbb597fdb',
  title: 'Operation of a 1-Liter-Volume Gaseous Argon Scintillation Counter',
  keywords:
   [ 'Environmental and Biometrics Sensors/Actuators',
     'Information Technology/Informatics',
     'Printed circuits and integrated circuits' ],
  journal: { name: 'IBM J. Res. Dev.' } }
{ _id: '63776ddebd78270b31375a65',
  title: 'Comment on \'Note on the dog-and-rabbit chase problem in introductory\n  kinematics\'',
  keywords:
   [ 'Printed circuits and integrated circuits',
     'Information Technology/Informatics',
     'Computer Technology/Graphics/Meta Computing' ],
  journal: { name: 'IEEE Trans. Veh. Technol.' } }
{ _id: '63776dde85d9a98e5725aecb',
  title: 'Constructing quantum games from a system of Bell\'s inequalities',
  keywords:
   [ 'Research Networking/GRID',
     'Information Technology/Informatics',
     'Printed circuits and integrated circuits' ],
  journal: { name: 'IEEE Veh. Technol. Mag.' } }
{ _id: '637770ab6b6626100195d0ae',
  title: 'Parallel electron-hole bilayer conductivity from electronic interface\n  reconstruction',
  keywords:
   [ 'Human Language Technologies',
     'Information Technology/Informatics',
     'Printed circuits and integrated circuits' ],
  journal: { name: 'IEEE Trans. Veh. Technol.' } }
{ _id: '637770ac0e83266ada6ca9ce',
  title: 'The Scattering Approach to the Casimir Force',
  keywords:
   [ 'Printed circuits and integrated circuits',
     'Information Technology/Informatics',
     'Data Protection/Storage/Cryptography/Security' ],
  journal: { name: 'IEEE Trans. Veh. Technol.' } }
```

Figure 4.21: Query 9: result.

Query 10: Conditions on fields of embedded documents

The following query finds 5 documents in the dataset for which at least one of the following conditions holds:

- are written by an author whose affiliation is “Valreda”;
- are published in “Comput. J.”;
- are written by an author whose affiliation is “Stralum”.

The titles of the documents are displayed.
This query uses the `limit()` method to limit the search only to 5 documents. The author, the name of the journal and the affiliation are in embedded documents: the author's name and its affiliation are in the Author document whereas the journal name is in the Journal document. Thus, to access these fields, we need to use the dot notation.

```
db.Big_Data_V2.find({
  "$or":[{"authors.affiliation":"Valreda"},{"journal.name":"Comput. J."}, {"authors.affiliation":"Stralum"}],
},
{
  "title":1,
}).limit(5)
```

Figure 4.22: Query 10.

```
{ _id: '63776dddc6eded596bb86c5a',
  title: 'Charmonium in the vector channel at finite temperature from QCD sum\n  rules' }
{ _id: '63776ddd2f7fbf02b6ecfdf3',
  title: 'HD 77361: A new case of super Li-rich K giant with anomalous low 12C/13C\n  ratio' }
{ _id: '63776ddd2f51020922409414',
  title: 'A Bayesian Approach Accounting for Stochastic Fluctuations in Stellar\n  Cluster Properties' }
{ _id: '63776dddc850bb526d54eef1',
  title: 'Molecular Line Observations of Infrared Dark Clouds II: Physical\n  Conditions' }
{ _id: '63776ddd5c37f2c5afb5342c',
  title: 'Adding a New Dimension to DNA Melting Curves' }
```

Figure 4.23: Query 10: result.

Query 11: Average number of articles published in a journal

The following query returns the average number of articles published in a journal. The query uses an aggregation pipeline, the main operations are:

- `$group` and `$count` are used to return the number of articles published by each journal;
- Another `$group` is used, together with `$avg`, to compute the average on all the records of the list obtained with the previous step. To collect all the records in one, "null" is used as `_id`.

```
Atlas atlas-52pbmf-shard-0 [primary] Project> db.Big_Data_V2.aggregate([{
                                              $group: {
                                               _id: '$journal.name',
                                               count: {
                                                $count: {}
                                               }
                                              }
                                             }, {
                                              $group: {
                                               _id: null,
                                               average: {
                                                $avg: '$count'
                                               }
                                              }
                                             }, {
                                              $project: {
                                               _id: 0
                                              }
                                             }])
```

Figure 4.24: Query 11.

```
{ average: 214.28571428571428 }
```

Figure 4.25: Query 11: result.

Query 12: Conditions on journals (without using count())

The following query shows the number of documents in the dataset that were published in the journal "Comput. J." and that are the ninth volume of a series. This query uses the aggregation pipeline, and is divided in 2 principal sections:

- The `$match` operator allows to filter the documents, finding only the required ones.
- The `$group` operator allows to aggregate the documents and to sum them using the `$sum` operator which sums every document and returns their total number.

```
db.Big_Data_V2.aggregate([{
    $match: {
      "journal.name":"Comput. J.",
      "journal.volume": {"$gt":9}
    }
  },
  {
    $group: {
      _id: 0,
      count: {
        $sum: 1
      }
    }
  },
  {
    $project: {
      _id: 0,
      count:1,
    }
  }])
```

Figure 4.26: Query 12.

```
{ count: 182 }
```

Figure 4.27: Query 12: result.

Query 13: Most used keywords

This query returns the number of articles that contain each keyword, sorted from the most used to the least used. Since the keywords are stored in an array, the use of `$unwind` is necessary. The query uses an aggregation pipeline, with these operations:

- The previously mentioned `$unwind` is used to separate the elements of the keywords array. Every article is separated into multiple copies, one for each element of the array keywords;
- `$group` is used to group together all the articles that contain the same keyword from the obtained list. `$count` is used to count the number of articles grouped together.
- Finally, `$sort` is used to sort the elements in descending order.

```
Atlas atlas-52pbmf-shard-0 [primary] Project> db.Big_Data_V2.aggregate([{
                                              $unwind: {
                                               path: '$keywords'
                                              }
                                             }, {
                                              $group: {
                                               _id: '$keywords',
                                               count: {
                                                $count: {}
                                               }
                                              }
                                             }, {
                                              $sort: {
                                               count: -1
                                              }
                                             }])
```

Figure 4.28: Query 13.

```
{ _id: 'Quantum Informatics', count: 397 }
{ _id: 'Optical Networks and Systems', count: 387 }
{ _id: 'Magnetic and superconductory materials/devices',
  count: 383 }
{ _id: 'Embedded Systems and Real Time Systems', count: 383 }
{ _id: 'High Frequency Technology/Microwaves', count: 382 }
{ _id: 'Printed circuits and integrated circuits', count: 375 }
{ _id: 'D printing', count: 373 }
{ _id: 'Smart cards and access systems', count: 372 }
{ _id: 'Electronic circuits/components and equipment',
  count: 368 }
{ _id: 'Environmental and Biometrics Sensors/Actuators',
  count: 367 }
{ _id: 'Electronic engineering', count: 363 }
{ _id: 'Peripherals Technologies (Mass Data Storage/Displays)',
  count: 353 }
{ _id: 'Semiconductors', count: 326 }
{ _id: 'Computer Hardware', count: 211 }
{ _id: 'e-Government', count: 209 }
{ _id: 'Speech Processing/Technology', count: 208 }
{ _id: 'Radar', count: 206 }
{ _id: 'Operation Planning and Scheduler System', count: 205 }
{ _id: 'Serious Games', count: 203 }
{ _id: 'Broadband Technologies', count: 203 }
```

Figure 4.29: Query 13: result.

Query 14: Titles of articles cited in the bibliography

The following query retrieves the titles of all the articles cited in the bibliography of a specific article.
The original article, in which the bibliography is present in the form of references to other documents, can be seen in Figure 4.30.
The query uses the aggregation pipeline. The most relevant operations are the following:

- The equivalent of an SQL `JOIN` is performed using the `$lookup` operator, joining articles' references present in the bibliography with the corresponding article based on their `_id`s
- An `$unwind` is performed returning a temporary set of articles in which, for each article present in the `bibliography` array, a new document is created. There are now multiple documents with the same values for each field of the original document,

but each containing only one article of the ones present in the `bibliography` array.

For readability reasons, only the title of the original document and the title of the article cited in the bibliography are shown. The query can be seen in Figure 4.31, while the result is shown in Figure 4.32.

```
{ _id: '63776dddd15bc88e6627df47',
  title: 'Chiral Topological Insulators, Superconductors and other competing\n  orders in three dimensions',
  bibliography:
   [ '63777073b372ec5c13bc54a1',
     '63776ddea074e5f9d38061bd',
     '63777073e31058db9972c204',
     '637770ab916065b1fe8a3cbc' ] }
```

Figure 4.30: Query 14: before.

```
Atlas atlas-l7mawr-shard-0 [primary] journal_articles> db.articles.aggregate([
                                                       {
                                                         $match: {"_id": ObjectId('63776ddde00236845a2fb904')}
                                                       },{
                                                         $lookup: {
                                                         from: "articles",
                                                         localField: "bibliography",
                                                         foreignField: "_id",
                                                         as: "bibliography" }
                                                       },{
                                                         $unwind: "$bibliography"
                                                       },{
                                                         $project: {"title": 1, "bibliography.title":1}
                                                       }])
```

Figure 4.31: Query 14.

```
{ _id: '63776dddd15bc88e6627df47',
  title: 'Chiral Topological Insulators, Superconductors and other competing\n  orders in three dimensions',
  bibliography: { title: 'The stable AR-quiver of a quantum complete intersection' } }
{ _id: '63776dddd15bc88e6627df47',
  title: 'Chiral Topological Insulators, Superconductors and other competing\n  orders in three dimensions',
  bibliography: { title: 'Equations and syzygies of the first secant variety to a smooth curve' } }
{ _id: '63776dddd15bc88e6627df47',
  title: 'Chiral Topological Insulators, Superconductors and other competing\n  orders in three dimensions',
  bibliography: { title: 'PCA consistency in high dimension, low sample size context' } }
{ _id: '63776dddd15bc88e6627df47',
  title: 'Chiral Topological Insulators, Superconductors and other competing\n  orders in three dimensions',
  bibliography: { title: 'On the structure of non-full-rank perfect codes' } }
```

Figure 4.32: Query 14: result.

Query 15: Cited articles in chronological order

The following query returns the titles of the articles, ordered from the most recent to the oldest, cited in the bibliography of two selected articles. The `bibliography` arrays of the two articles before the operation can be seen in Figure 4.33. To obtain the result, the aggregation pipeline offered by the `aggregate()` function is used. The most relevant operations are the following:

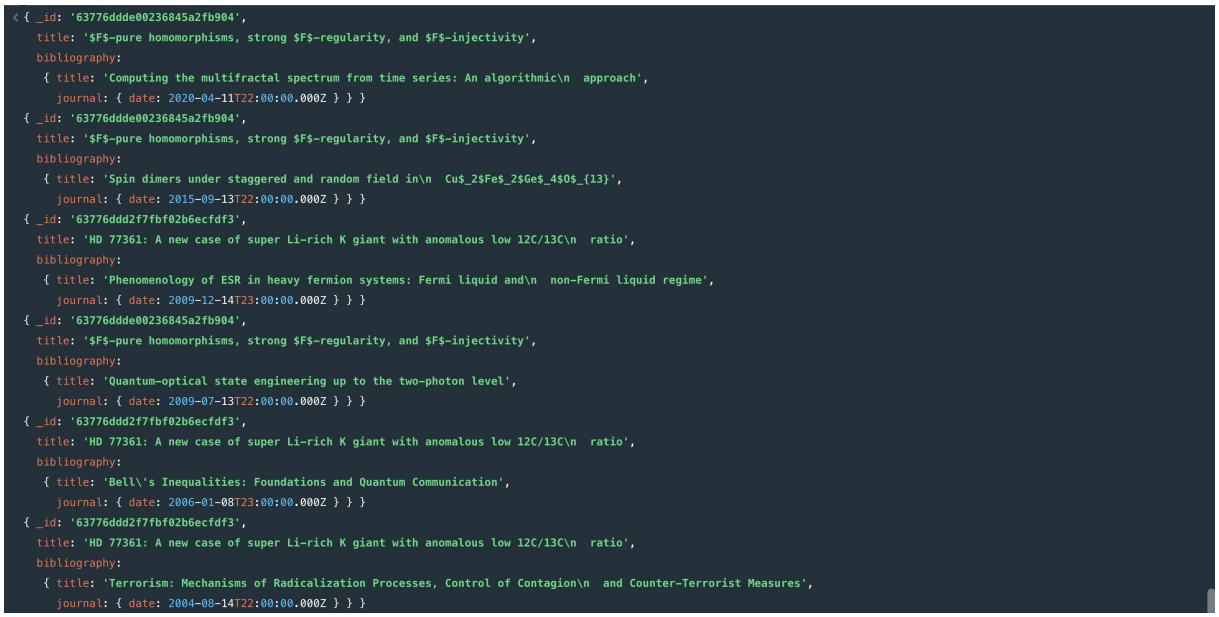
- The equivalent of an SQL `JOIN` is performed using the `$lookup` operator, joining articles' references present in the bibliography with the corresponding article based on their `_id`s;

- an `$unwind` is performed returning a temporary set of articles in which, for each article present in the `bibliography` array, a new document is created. There are now multiple documents with the same values for each field of the original document, but each containing only one article of the ones present in the `bibliography` array;
- the `$sort` operation orders the article in inverse chronological order.

For readability reasons, only the title of the original document, the title of the article cited in the bibliography and its publication date are shown. The query can be seen in Figure 4.34, while the result is shown in Figures 4.35 and 4.36.

Figure 4.33: Query 15: before.

Figure 4.34: Query 15.



```
{ _id: '63776ddde00236845a2fb904',
  title: '$F$-pure homomorphisms, strong $F$-regularity, and $F$-injectivity',
  bibliography:
   { title: 'Computing the multifractal spectrum from time series: An algorithmic\n  approach',
     journal: { date: 2020-04-11T22:00:00.000Z } } }
{ _id: '63776ddde00236845a2fb904',
  title: '$F$-pure homomorphisms, strong $F$-regularity, and $F$-injectivity',
  bibliography:
   { title: 'Spin dimers under staggered and random field in\n  Cu$_2$Fe$_2$Ge$_4$O$_{13}',
     journal: { date: 2015-09-13T22:00:00.000Z } } }
{ _id: '63776ddd2f7fbf02b6ecfdf3',
  title: 'HD 77361: A new case of super Li-rich K giant with anomalous low 12C/13C\n  ratio',
  bibliography:
   { title: 'Phenomenology of ESR in heavy fermion systems: Fermi liquid and\n  non-Fermi liquid regime',
     journal: { date: 2009-12-14T23:00:00.000Z } } }
{ _id: '63776ddde00236845a2fb904',
  title: '$F$-pure homomorphisms, strong $F$-regularity, and $F$-injectivity',
  bibliography:
   { title: 'Quantum-optical state engineering up to the two-photon level',
     journal: { date: 2009-07-13T22:00:00.000Z } } }
{ _id: '63776ddd2f7fbf02b6ecfdf3',
  title: 'HD 77361: A new case of super Li-rich K giant with anomalous low 12C/13C\n  ratio',
  bibliography:
   { title: 'Bell\'s Inequalities: Foundations and Quantum Communication',
     journal: { date: 2006-01-08T23:00:00.000Z } } }
{ _id: '63776ddd2f7fbf02b6ecfdf3',
  title: 'HD 77361: A new case of super Li-rich K giant with anomalous low 12C/13C\n  ratio',
  bibliography:
   { title: 'Terrorism: Mechanisms of Radicalization Processes, Control of Contagion\n  and Counter-Terrorist Measures',
     journal: { date: 2004-08-14T22:00:00.000Z } } }
```


Figure 4.35: Query 15: result (1st part).

```
{ _id: '63776ddde00236845a2fb904',
  title: '$F$-pure homomorphisms, strong $F$-regularity, and $F$-injectivity',
  bibliography:
   { title: 'Incorporating characteristics of human creativity into an evolutionary\n  art algorithm',
     journal: { date: 1990-10-30T23:00:00.000Z } } }
{ _id: '63776ddde00236845a2fb904',
  title: '$F$-pure homomorphisms, strong $F$-regularity, and $F$-injectivity',
  bibliography:
   { title: 'On the magnetic equation of state in (2+1)-flavor QCD',
     journal: { date: 1973-03-24T23:00:00.000Z } } }
```


Figure 4.36: Query 15: result (2nd part).

Query 16: Sections containing figures

The following query returns the title of the sections containing at least an image of a specific article. To obtain this result, we used the `aggregate` operator. To select the article, we use the `$match` operator, then to create a separate document each containing only one section of the original article, we use the `$unwind` operator, specifying the array to perform the unwind on. After that, we select the documents containing a figure in their section by using a `$match` operator combined with a `$exists` operator on the attribute figure of the section, and the return the title of those by using a `$project` operator.

```
Atlas atlas-lx2lpr-shard-0 [primary] SMBUD_Database> db.Articles.aggregate([
                                                     {$match:{_id:ObjectId("63776ddd297f4328218ebc59")}},
                                                     {$unwind:{"path":"$sections"}},
                                                     {$match:{"sections.figure":{$exists:true}}},
                                                     {$project:{"sections.title":1}}
                                                     ])
```

Figure 4.37: Query 16.

```
< { _id: '63776ddd297f4328218ebc59',
    sections: { title: 'Simulation of equatorial von Neumann measurements on GHZ states using\n  nonlocal resources' } }
  { _id: '63776ddd297f4328218ebc59',
    sections: { title: 'Chain enumeration of $k$-divisible noncrossing partitions of classical\n  types' } }
```

Figure 4.38: Query 16: result.

Query 17: Authors affiliated to a company

The following query returns the number of authors affiliated to a specific company called "Letpro". To obtain this result, we used the `aggregate` operator. The query needs to be executed on the whole data set, but there is no need to execute the next operations on articles which do not contain at least an author with the correct `affiliation`, so first of all, we use the `$match` operator. After that, we use the `$unwind` operator to create separate documents each containing only one author among the ones of the original articles. To select only the authors with the correct `affiliation`, we use a `$match` operator on the `affiliation` attribute of the `authors`. Finally, to count the distinct `authors`, we decided to exploit the `$group` operator, grouping by `author`, and then the `$count` operator to return the corresponding integer.

```
Atlas atlas-lx2lpr-shard-0 [primary] SMBUD_Database> db.Articles.aggregate([
                                                    {$match: {"authors.affiliation":"Letpro"}},
                                                    {$unwind:{"path":"$authors"}},
                                                    {$match:{"authors.affiliation":"Letpro"}},
                                                    {$group:{"_id":"$authors"}},
                                                    {$count:"number of authors with affiliation to Letpro"}
                                                    ])
```

Figure 4.39: Query 17.

```
    ])
< { 'number of authors with affiliation to Letpro': 25 }
```

Figure 4.40: Query 17: result.

Query 18: Subsections with figures

The following query returns the titles of the subsections containing at least an image of a specific article. To obtain this result, we use the `aggregate` operator. First of all, we select the article by using a `$match` operator, then, to divide the document into smaller documents each containing only one subsection, we used a combination of two `$unwind` operations, the first one on the `sections` and the second one on the `subsections` of each section. This is possible because the document based databases use a pipeline. As done in the query 16, we select the documents containing a figure in their subsection by using a `$match` operator combined with a `$exists` operator on the attribute figure of the subsection, and the return the title of those by using a `$project` operator.

```
Atlas atlas-1x2lpr-shard-0 [primary] SMBUD_Database> db.Articles.aggregate([
                                                     {$match:{_id:ObjectId("63777073646da6e0070c5f8a")}},
                                                     {$unwind:{"path":"$sections"}},
                                                     {$unwind:{"path":"$sections.subsections"}},
                                                     {$match:{"sections.subsections.figure":{$exists:true}}},
                                                     {$project:{"sections.subsections.title":1}}
                                                     ])
```

Figure 4.41: Query 18.

```
[ { _id: '63777073646da6e0070c5f8a',
    sections: { subsections: { title: 'Higher-level canonical subgroups for p-divisible groups' } } }
  { _id: '63777073646da6e0070c5f8a',
    sections: { subsections: { title: 'A one-dimensional Keller-Segel equation with a drift issued from the\n  boundary' } } }
```

Figure 4.42: Query 18: result.

5 | Apache Spark

5.1. Dataset description

For the Apache Spark database, we decided to use a reduced portion of the dataset used for the Neo4j database (see Chapter 3). The following entities have been kept:

- **Articles**: representing journal's and other form of articles, they are the main focus of this project;
- **Authors**: representing the Authors of the articles. They have a strong link to articles, being one of the most important part of the dataset;
- **Journals**: they represent the journal in which articles are published. They have been selected for query expressiveness reasons.

To the above entities, a bridge entity (`written_by`) has been added to represent the many-to-many relationship between Articles and their Authors: it contains all the pairs (article id, author id) allowing to navigate from the articles to their authors and viceversa. Another possibility to match articles and their authors could have been to add a column to the articles' dataFrame representing an array containing the IDs of the authors of the article. Although a valid option, this would have been more difficult to implement: in the article's `CSV` file, authors are represented as a single string with names divided by a `"|"`. We should have transformed the string in an array of values and found the IDs corresponding to every author. On the other hand, a table matching articles' IDs and authors' IDs had already been created to import data in the Neo4j part of the project. Since the table was already present, we opted for the bridge table solution.

5.2. Data import

5.2.1. Dataset creation

To create of the dataset we used the script mentioned in the Chapter 1 (`https://github.com/ThomHurks/dblp-to-csv`), generating four different CSV files containing the entities

mentioned above. This script also generates the IDs used in the database.

Since the DPLP dump does not contain information (except for the name) of Authors and Journals, we decided to generate some fields using the random generator tool of Excel (`RANDBETWEEN`). To be precise, we decided to add the birthdate and nationality in Authors, the foundation date and the type for Journals.

In the Article dataFrame, we decided to cut some attributes of the original file using Excel to make the output of the queries more readable, since there were some fields with a lot of *null* values. We also replaced the Journal attribute with the ID of the new Journal dataFrame. Forse non l'abbiamo fatto

5.2.2. Data upload

To upload the data in the database we first had to import the basic `pyspark` library and create an entry point to the PySpark Application. The used commands can be see in Figure 5.1.

```python
import pyspark
from pyspark.sql import SparkSession

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

Python

Figure 5.1: PySpark library import and entry point creation.

Then, after importing all the necessary libraries, the schema of the different dataFrames has been defined by using the `StructType` class which allows to define the structure of a row, i.e. a list of fields with their name, type and an option to allow the value of that field to be null.
Lastly, we print out the schema to be sure of the success of the operation and that every field is of the correct type (to make this document more readable this last result is only shown for the `articles` dataFrame, all other dataFrames have been checked and are correct).

`schema_articles` specifies the schema of the `articles`' dataFrame (Figure 5.2).

- **ID**: the key of the articles table;
- **Authors** (Remember to check this part): a list of the name of the authors. Due to conversion reason, we decided to keep this attribute even if information about authors is also present in the `written_by` dataFrame;
- **Journal**: foreign key pointing to a row in the Journals table;

- **Key**: refers to the key used in the DBLP dataset, from which we took the data;
- **Date**: refers to the publication date of the article;
- **Type**: can be either informal or formal;
- **Title**: a string containing the article's title;

We then create the `articles`' dataFrame by reading data from the `CSV` file and applying the defined schema.

```python
from pyspark.sql.types import StructType, StructField, StringType, FloatType, ArrayType, IntegerType, TimestampType, DateType

schema_articles = StructType([ \
    StructField("ID", IntegerType(), True), \
    StructField("Authors", StringType(), True), \
    StructField("Journal", StringType(), True), \
    StructField("Key", StringType(), True), \
    StructField("Date", DateType(), True), \
    StructField("Publisher", StringType(), True), \
    StructField("Type", StringType(), True), \
    StructField("Title", StringType(), True), \
    StructField("Year", IntegerType(), True) \
])
articles = spark.read.csv("output_article_copia.csv",header=True, sep=";", schema=schema_articles)
articles.printSchema()
```

Python

Figure 5.2: Articles' schema definition.

```
root
 |-- ID: integer (nullable = true)
 |-- Authors: string (nullable = true)
 |-- Journal: integer (nullable = true)
 |-- Key: string (nullable = true)
 |-- Date: date (nullable = true)
 |-- Publisher: string (nullable = true)
 |-- Type: string (nullable = true)
 |-- Title: string (nullable = true)
 |-- Year: integer (nullable = true)
```

Figure 5.3: Articles' schema.

`schema_journal` specifies the schema of the `journals` dataFrame (Figure 5.4).

- **ID**: the key of the journals table;
- **Name**: the name of the journal;
- **FoundationDate**: the foundation date of the journal;
- **Type**: this value can be either online or on paper.

```python
schema_journal = StructType([ \
    StructField("ID", IntegerType(), True), \
    StructField("Name", StringType(), True), \
    StructField("FoundationDate", DateType(), True), \
    StructField("Type", StringType(), True)
])
journals = spark.read.csv("output_journal.csv",header=True, sep=";", schema=schema_journal)
journals.printSchema()
```

Python

Figure 5.4: Journals' schema definition.

`schema_authors` specifies the schema of the `journals` dataFrame (Figure 5.5).

- **ID**: the key of the authors table;
- **Name**: the authors' names;
- **BirthDate**: the authors' birth date;
- **Nationality**: the authors' nationality.

```python
schema_authors = StructType([ \
    StructField("ID", IntegerType(), True), \
    StructField("Name", StringType(), True), \
    StructField("BirthDate", DateType(), True), \
    StructField("Nationality", StringType(), True)
])
authors = spark.read.csv("output_author.csv",header=True, sep=";", schema=schema_authors)
authors.printSchema()
```

Figure 5.5: Authors' schema definition.

`schema_written_by` specifies the schema of the `written_by` dataFrame (Figure 5.6). This table represents the `written_by` relationship in the ER model.

- **Article**: the foreign key in the articles table;
- **Author**: the foreign key in the authors table.

```python
schema_written_by = StructType([ \
    StructField("Article", IntegerType(), True), \
    StructField("Author", IntegerType(), True), \
])
written_by = spark.read.csv("output_author_authored_by.csv",header=True, sep=",", schema=schema_written_by)
written_by.printSchema()
```

Figure 5.6: Written_by schema definition.

5.3. Queries

5.3.1. Data creation and update

Query 1: Article entry creation

The following query inserts a new article in the `articles` dataFrame. The article presents all the fields defined in the schema.
To define the new row we first need to define the content of the fields of the article: we do so by listing all the fields' values. We then create a new dataFrame (`createdataFrame()`) that is added to the existing articles' dataFrame using the `union()` function. The inserted row is then retrieved to check the result of the operation. The result of the query is shown (Figure 5.8) to verify the correct outcome of the operation.

```python
from pyspark.sql.functions import col

new_row = [(123456, "Alessio Buda, Leonardo Cesani, Fausto Lasca, Gabriele Munafò, Matteo Paraboschi", "Frontiers in Data Science", "article_key",
            "2022-12-06", "PoliMI", "University Project", "SMBUD Project", "2022")]
new_article = spark.createDataFrame(new_row)

articles = articles.union(new_article)

articles.filter(col("ID") == 123456).show(truncate=True)
```

Python

Figure 5.7: Query 1.

```
+------+--------------------+--------------------+-----------+----------+---------+------------------+-------------+----+
|    ID|             Authors|             Journal|        Key|      Date|Publisher|              Type|        Title|Year|
+------+--------------------+--------------------+-----------+----------+---------+------------------+-------------+----+
|123456|Alessio Buda, Leo...|Frontiers in Data...|article_key|2022-12-06|   PoliMI|University Project|SMBUD Project|2022|
+------+--------------------+--------------------+-----------+----------+---------+------------------+-------------+----+
```

Figure 5.8: Query 1: result.

Query 2: Update the Publisher of an article

The following query updates all the articles for which the field `Publisher` is equal to "PoliMI", namely the article inserted in the previous query, by substituting it with "Politecnico di Milano".
In Figure 5.10 the state of the dataFrame before the operation is shown: there is only one article for which the Publisher field is equal to "PoliMI" and no articles such that the same field is equal to "Politecnico di Milano". Then, the `withColumn()` function is used to substitute the value of the column `Publisher` for all articles for which the condition stated in the `when` operator is satisfied. In the `when` operator the new value for the column is also specified. Lastly, the `otherwise` operator specifies the operation to perform for all articles not satisfying the condition: in this case the value is left unchanged.
Since a dataFrame is immutable, to effectively change the value the resulted dataFrame

should be reassigned to the original one.
To conclude, the result of the query is shown (Figure 5.11) to verify the correct outcome of the operation.

```python
from pyspark.sql.functions import col, when

articles.filter(col("Publisher") == "PoliMI").show()
articles.filter(col("Publisher") == "Politecnico di Milano").show()

articles = articles.withColumn("Publisher", when(articles.Publisher == "PoliMI","Politecnico di Milano") \
      .otherwise(articles.Publisher))

articles.filter(col("Publisher") == "PoliMI").show()
articles.filter(col("Publisher") == "Politecnico di Milano").show()
```

Python

Figure 5.9: Query 2.

```
+------+--------------------+--------------------+----------+----------+---------+-----------------+------------+----+
|    ID|             Authors|             Journal|       Key|      Date|Publisher|             Type|       Title|Year|
+------+--------------------+--------------------+----------+----------+---------+-----------------+------------+----+
|123456|Alessio Buda, Leo...|Frontiers in Data...|article_key|2022-12-06|   PoliMI|University Project|SMBUD Project|2022|
+------+--------------------+--------------------+----------+----------+---------+-----------------+------------+----+

+---+-------+-------+---+----+---------+----+-----+----+
| ID|Authors|Journal|Key|Date|Publisher|Type|Title|Year|
+---+-------+-------+---+----+---------+----+-----+----+
+---+-------+-------+---+----+---------+----+-----+----+
```

Figure 5.10: Query 2: before.

```
+---+-------+-------+---+----+---------+----+-----+----+
| ID|Authors|Journal|Key|Date|Publisher|Type|Title|Year|
+---+-------+-------+---+----+---------+----+-----+----+
+---+-------+-------+---+----+---------+----+-----+----+

+------+--------------------+--------------------+----------+----------+--------------------+-----------------+------------+----+
|    ID|             Authors|             Journal|       Key|      Date|           Publisher|             Type|       Title|Year|
+------+--------------------+--------------------+----------+----------+--------------------+-----------------+------------+----+
|123456|Alessio Buda, Leo...|Frontiers in Data...|article_key|2022-12-06|Politecnico di Mi...|University Project|SMBUD Project|2022|
+------+--------------------+--------------------+----------+----------+--------------------+-----------------+------------+----+
```

Figure 5.11: Query 2: result.

Query 3: Journal entry creation

The following query inserts a new journal in the `journals` dataFrame. The inserted journal contains all the fields defined in the journal's schema.
We proceed by defining the values of each field of the new row. After that we create a new dataFrame by using the `createdataFrame()` function and we add it to the previously existing journals' dataFrame by using the `union()` function. Lastly, we use a `filter()` function to show the result of the operation.
The result of the query is shown (Figure 5.13) to verify the correct outcome of the operation.

```python
from pyspark.sql.functions import *

row = [(54632123, "Computer Science Journal", "2022-12-07", "online")]
journal = spark.createDataFrame(row)
journals = journals.union(journal)

journals.filter(col("ID") == 54632123).show()
```

Python

Figure 5.12: Query 3.

```
+--------+--------------------+--------------+------+
|      ID|                Name|FoundationDate|  Type|
+--------+--------------------+--------------+------+
|54632123|Computer Science ...|    2022-12-07|online|
+--------+--------------------+--------------+------+
```

Figure 5.13: Query 3: result.

Query 4: Update the foundation date of a journal

The following query modifies the `FoundationDate` value of an already existing row of the `journals`' dataFrame. Every journal founded in 2022-12-07 will be converted into a journal founded in 2022-12-08.

We proceed by checking the initial state of the dataFrame, so we show the rows with FoundationDate equal to 2022-12-07 or 2022-12-08. After that, we proceed to modify the `FoundationDate`, when it equals to 2022-12-07, by using the `withColumn()` function. To avoid modifying the wrong rows, we use the `otherwise` operator to indicate what to do when the condition is not satisfied. Finally, we show the result of the query by showing again the rows with `FoundationDate` equal to 2022-12-07 or 2022-12-08, so that we can confront them with the initial ones.

The result of the query is shown (Figure 5.16) to verify the correct outcome of the operation.

```python
from pyspark.sql.functions import *

journals.filter(col("FoundationDate") == "2022-12-07").show()
journals.filter(col("FoundationDate") == "2022-12-08").show()

journals = journals.withColumn("FoundationDate", when(journals.FoundationDate == "2022-12-07", "2022-12-08").otherwise(journals.FoundationDate))

journals.filter(col("FoundationDate") == "2022-12-07").show()
journals.filter(col("FoundationDate") == "2022-12-08").show()
```

Python

Figure 5.14: Query 4.

```
+--------+--------------------+--------------+------+
|      ID|                Name|FoundationDate|  Type|
+--------+--------------------+--------------+------+
|54632123|Computer Science ...|    2022-12-07|online|
+--------+--------------------+--------------+------+


+---+----+--------------+----+
| ID|Name|FoundationDate|Type|
+---+----+--------------+----+
+---+----+--------------+----+
```

Figure 5.15: Query 4: before.

```
+---+----+---------------+----+
| ID|Name|FoundationDate|Type|
+---+----+---------------+----+
+---+----+---------------+----+

+--------+--------------------+---------------+------+
|      ID|                Name|FoundationDate|  Type|
+--------+--------------------+---------------+------+
|54632123|Computer Science ...|    2022-12-08|online|
+--------+--------------------+---------------+------+
```

Figure 5.16: Query 4: result.

Query 5: Author entry creation

The following query adds a new entry in the `authors` dataFrame.
To do so, we firstly create another dataFrame where we insert the data of the new entry. The columns of the new dataFrame must have the same types of the original one. Then we use the `union()` function to do the union operation between the two dataFrames.
The result of the query is shown (Figure 5.18) to verify the correct outcome of the operation.

```python
from pyspark.sql.functions import col

new_row = [(2022, "Marco Brambilla", "2022-1-1", "Italian")]
new_author = spark.createDataFrame(new_row)

authors = authors.union(new_author)

authors.filter(col("Name") == "Marco Brambilla").show(truncate=False)
```

Python

Figure 5.17: Query 5.

```
+----+---------------+---------+-----------+
|ID  |Name           |BirthDate|Nationality|
+----+---------------+---------+-----------+
|2022|Marco Brambilla|2022-1-1 |Italian    |
+----+---------------+---------+-----------+
```

Figure 5.18: Query 5: result.

Query 6: Author entry deletion

The following query removes an entry from the `authors` dataFrame.
To do so, it uses the `filter()` function filtering every entry apart from the one to remove. Since a dataFrame is immutable, to effectively change the value the resulted dataFrame should be reassigned to the original one.
The result of the query is shown (Figure 5.21) to verify the correct outcome of the operation.

```python
authors.filter(col("Name") == "Marco Brambilla").show(truncate=False)
authors = authors.filter(col("Name") != "Marco Brambilla")
authors.filter(col("Name") == "Marco Brambilla").show(truncate=False)
```

Figure 5.19: Query 6.

```
+----+---------------+---------+-----------+
|ID  |Name           |BirthDate|Nationality|
+----+---------------+---------+-----------+
|2022|Marco Brambilla|2022-1-1 |Italian    |
+----+---------------+---------+-----------+
```

Figure 5.20: Query 6: before.

```
+---+----+---------+-----------+
|ID |Name|BirthDate|Nationality|
+---+----+---------+-----------+
+---+----+---------+-----------+
```

Figure 5.21: Query 6: result.

5.3.2. Data manipulation

Query 7: Italian authors on AI

The following query retrieves the Italian authors born later than 1970 that have written something about AI.

The first part of the query (whose result is saved as `italian_authors`) creates a support dataFrame by filtering in the `authors`' dataFrame the Italian authors born later than 1970.

The second part (whose result is saved as `query1`) performs a first `join()` between the `italian_authors` dataFrame and the bridge table (`written_by`), in order to find the `IDs` of articles written by Italian authors. A second `join` is performed between obtained dataFrame, containing information about authors and the articles, and the `articles`' dataFrame, adding the columns of the `articles` dataFrame.

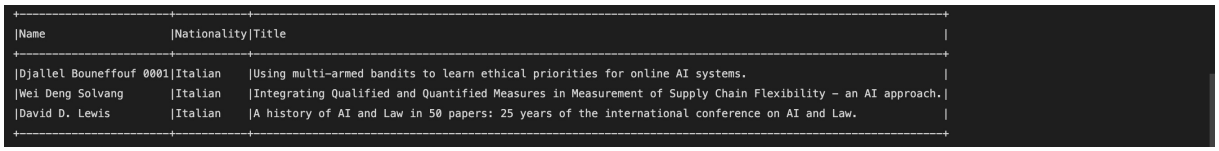
In the end, the table thus obtained, is filtered on the `Title` column, searching for the titles containing the "AI" word.

The result of the query is shown in Figure 5.23.

```python
from pyspark.sql.functions import *
italian_authors = authors\
    .filter(year(col("BirthDate")) >= 1970)\
    .filter(col("Nationality") == "Italian")\

query1 = italian_authors.join(written_by, italian_authors.ID == written_by.Author, "inner") \
    .join(articles, articles.ID == written_by.Article, "inner")\
    .select(italian_authors.Name, italian_authors.Nationality, articles.Title)\
    .filter(col("Title").like("% AI %"))\
    .show(truncate=False)
```

Figure 5.22: Query 7.



Name	Nationality	Title
Djallel Bouneffouf 0001	Italian	Using multi-armed bandits to learn ethical priorities for online AI systems.
Wei Deng Solvang	Italian	Integrating Qualified and Quantified Measures in Measurement of Supply Chain Flexibility – an AI approach.
David D. Lewis	Italian	A history of AI and Law in 50 papers: 25 years of the international conference on AI and Law.

Figure 5.23: Query 7: result.

Query 8: Prolific authors

The following query finds the top 5 authors by number of articles written, whose name begins with the letter "D".
In the first part of the query, a `filter()` is performed to find Authors whose name starts with "D", then the resulting dataFrame is joined with the bridge table (`written_by`), to retrieve the IDs of articles written by each author and a final `join()` allows to retrieve the information about articles.
Then a `groupBy()` operation is performed to group the articles written by each author and, using the aggregation, the number of articles for each authors is calculated.
Finally, the output is sorted in descending order based on the number of written articles and the first five results are taken.
The result of the query is shown in Figure 5.25.

```python
D_authors = authors.filter(col("Name").like("D%"))\
    .join(written_by, authors.ID == written_by.Author, "inner") \
    .join(articles, articles.ID == written_by.Article, "inner")\
    .groupBy(col("Name"))\
    .agg(count("Name").alias("Articles written"))\
    .orderBy(col("Articles written").desc())\
    .limit(5)\
    .show()
```

Figure 5.24: Query 8.

Name	Articles written
Douglas R. Stinson	29
Dieter Jungnickel	28
Darren Dalcher	24
Dennis Bodson	22
Dusit Niyato	18

Figure 5.25: Query 8: result.

Query 9: Articles of the oldest online journal

The following query finds articles published by the oldest online journal.
Firstly, the query extracts the foundation date of the oldest online journal, then it uses this information to extract the name of the oldest journal from the `journals` dataFrame

and use it to extract all articles published by this journal. If more than one journal has been founded on the date retrieve at the previous step, then only the first one returned is considered.
The result of the query is shown in Figure 5.27.

```python
from pyspark.sql.functions import *
oldest_date = journals.filter(col("Type") == "online")\
    .agg(min("FoundationDate")).collect()[0][0]

oldest_journal = journals.filter(col("FoundationDate") == oldest_date).select(journals.Name).collect()[0][0]

print("The oldest journal was " + str(oldest_journal) + ", founded on " + str(oldest_date) + ".")
print("It published the following articles:")
oldest_journal_publications = journals.filter(col("Name") == oldest_journal)\
    .join(articles, articles.Journal == journals.ID)\
    .select(articles.Title)\
    .show()
```

Python

Figure 5.26: Query 9.

```
The oldest journal was meltdownattack.com, founded on 1941-02-27.
It published the following articles:
+--------------------+
|               Title|
+--------------------+
|16-bit 1-MS/s SAR...|
|3D Transformative...|
|A behavior-based ...|
|A big data approa...|
|A block Chebyshev...|
|A Channel Quality...|
|A classification ...|
|A conservative co...|
|A Construction of...|
|A Continuous Time...|
|A Cross-Layer Qua...|
|A Customizable Te...|
|A Data Model for ...|
|A digital phase-b...|
|A dynamic logic f...|
|A fast solver for...|
|A framework for c...|
|A geometric appro...|
|A Hard-Sphere Mod...|
|  A Hybrid City Car.|
+--------------------+
only showing top 20 rows
```

Figure 5.27: Query 9: result.

Query 10: Active authors in 2020

The following query shows the information of the authors who wrote at least one article in 2020.
We proceed by filtering the articles created in 2020 and turning them into a collection. Then we create an array containing the IDs of the articles just selected. We do that in order to use the `isin()` function, which requires a list of values. In fact, we now find the IDs of the authors who wrote at least one of the articles in the array.
Again, we turn the result in a collection and create an array to use the `isin()` function. Finally, we filter the authors whose ID is present in the array.
The result of the query is shown in Figure 5.29.

```python
from pyspark.sql.functions import *

articles_2020 = articles.filter((col("Date") >= "2020-01-01") & (col("Date") <= "2020-12-31")).collect()

articles_IDs = [ai[0] for ai in articles_2020]

authors_ID_filtered = written_by.filter(col("Article").isin(articles_IDs)).collect()

authors_IDs = [ai[1] for ai in authors_ID_filtered]

authors.filter(col("ID").isin((authors_IDs))).show()
```

Figure 5.28: Query 10.

```
+-----+--------------------+----------+-----------+
|   ID|                Name| BirthDate|Nationality|
+-----+--------------------+----------+-----------+
|42024|         Paul Kocher|1980-12-12|    Italian|
|42025|       Daniel Genkin|1960-10-06|    Spanish|
|42026|        Daniel Gruss|1978-10-25|     French|
|42027|    Werner Haas 0004|1966-03-05|  Giapanese|
|42028|        Mike Hamburg|1954-04-21|   American|
|42029|         Moritz Lipp|1971-03-12|   Canadian|
|42030|      Stefan Mangard|1956-11-07|    Turkish|
|42031|Thomas Prescher 0002|1994-06-28|     German|
|42032|Michael Schwarz 0001|1962-03-29|      Swiss|
|42033|         Yuval Yarom|1948-07-04|    Chinese|
|42040|Alejandro P. Buch...|1953-11-26|    Mexican|
|42056|   Christoph Beierle|1958-06-30|    Spanish|
|42057|          Udo Pletat|1950-01-07|    Finnish|
|42071|        Albert Maier|1948-10-16|    Mexican|
|42076|     Stefan B@ttcher|1995-11-21|    Turkish|
|42078|      Toni Bollinger|1975-12-14|    Chinese|
|42082|      Otthein Herzog|1959-04-24|  Norwegian|
|42122|       Gregor Erbach|1960-10-07|    Spanish|
|42130|  Kenneth L. Deckert|1992-08-01|      Swiss|
|42183|    Raymond A. Lorie|1952-08-30|    Swedish|
+-----+--------------------+----------+-----------+
only showing top 20 rows
```

Figure 5.29: Query 10: result.

Query 11: Top ten authors

This query returns a list of the top 10 authors that published more articles.
Firstly, we use the `join()` operator to connect the `authors` dataFrame and the `written_by` dataFrame. After that, we use the `groupBy()` operator to group the records by the authors' names and we use the `count()` operator to obtain the number of articles in each group, storing this data with the default alias "count". Then, we use the `select()` operator to only keep the authors' names and count columns, and we rename the "count" column to "#Articles". Finally, we use the `sort()` and `desc()` operators to sort the result in descending order of the value of the "#Articles" column and we show the first ten results.
The result of the query is shown in Figure 5.31.

```python
authors.join(written_by, written_by.Author == authors.ID, "inner") \
    .groupby(col("Name")).count() \
    .select(col("Name"), col("count").alias("#Articles")) \
    .sort(col("#Articles").desc()) \
    .show(10)
```

Figure 5.30: Query 11.

```
+------------------+---------+
|              Name|#Articles|
+------------------+---------+
|       Lajos Hanzo|      124|
|    Gerrit Bleumer|       69|
|     Alex Biryukov|       64|
|  Christoph Meinel|       57|
|Harvey Glickenstein|       51|
|       Xuemin Shen|       51|
|   Javier Gozálvez|       49|
| Friedrich L. Bauer|       49|
|      Bart Preneel|       47|
|  Panos M. Pardalos|       45|
+------------------+---------+
only showing top 10 rows
```

Figure 5.31: Query 11: result.

Query 12: Authors between 1990 and 1995

The following query returns the number of Authors born between 1990 and 1995 for each nationality.

First, the `filter()` function filters authors born between 1990 and 1995. Then, using the `groupBy()` function, authors with the same nationality are grouped together. The `count()` method counts them. Lastly, the `sort()` function sorts the output in descending order based on the "Count" column from the previous step.

The result of the query is shown in Figure 5.33.

```python
authors.filter((col("Birthdate") >= '1990-01-01') & (col("Birthdate") <= '1995-12-31')) \
    .groupby(col("Nationality")).count() \
    .sort(col("count").desc()) \
    .show()
```

Figure 5.32: Query 12.

```
+-------------+-----+
|  Nationality|count|
+-------------+-----+
|      Mexican|  335|
|      Bosnian|  330|
|      Italian|  329|
|        Swiss|  327|
|      Chinese|  325|
|   Portuguese|  324|
|      Turkish|  323|
|       Danish|  323|
|      Russian|  319|
|    Brazilian|  319|
|     Canadian|  317|
|South African|  315|
|   Australian|  313|
|      Swedish|  311|
|    Norwegian|  309|
|       German|  308|
|     American|  308|
|     Egyptian|  307|
|      Finnish|  301|
|  Argentinian|  301|
+-------------+-----+
only showing top 20 rows
```

Figure 5.33: Query 12: result.

Query 13: Most common birth dates

The following query returns the 10 dates in which the most Authors were born, in descending chronological order.

First, all the authors with the same date of birth are grouped together by apllying the `groupBy()` function on the `authors` dataFrame and for each date the number of authors is counted using the `count()` function. Then, the columns "Birthdate" and "count" are selected and the `alias()` operator is used to rename them to "Date" and "#Authors". Finally, rows are sorted in descending order with the `sort()` function based on the number of Authors. Only shows the first 10 rows of the result are shown.

The result of the query is shown in Figure 5.35.

```python
authors.groupby(col("Birthdate")).count() \
    .select(col("Birthdate").alias("Date"), col("count").alias("#Authors")) \
    .sort(col("#Authors").desc()) \
    .show(10)
```

Python

Figure 5.34: Query 13.

```
+----------+--------+
|      Date|#Authors|
+----------+--------+
|1962-02-05|      14|
|1953-01-06|      13|
|1962-07-10|      13|
|1972-04-25|      12|
|1948-07-19|      12|
|1960-08-24|      12|
|1993-02-27|      11|
|1974-09-20|      11|
|1986-09-29|      11|
|1956-05-29|      11|
+----------+--------+
only showing top 10 rows
```

Figure 5.35: Query 13: result.

Query 14: Journal with most articles

The following query finds the journals that have published more than 200 articles in 2020 and the exact number of published articles.

First, we used the `filter()` function to select only the articles published in 2020. Then, the `groupBy()` method is used to group the articles depending on the journal on which they have been published.

The aggregation (`agg()`) and `count()` functions are used to count the number of articles published by each journal. The `alias()` operator renames the obtained column in "Number of articles".

A second `filter()` is performed to remove all the journal groups with less than 200 articles. Lastly, the `sort()` function orders the journals based on the number of articles they have published, in decreasing order.

The result of the query is shown in Figure 5.37.

```python
from pyspark.sql.functions import count, col

articles.filter((col("Date") > "2020") & ((col("Date") < "2021"))) \
        .groupBy("Journal") \
        .agg(count("Journal") \
        .alias("Number of articles")) \
        .filter(col("Number of articles") > "200") \
        .sort(col("Number of articles").desc()).show()
```

Python

Figure 5.36: Query 14

```
+-------+-------------------+
|Journal|Number of articles|
+-------+-------------------+
| 112304|                271|
| 112331|                259|
| 112324|                255|
| 112311|                255|
| 112309|                248|
| 112337|                244|
| 112314|                241|
| 112339|                241|
| 112301|                240|
| 112319|                239|
| 112341|                238|
| 112306|                238|
| 112303|                237|
| 112342|                237|
| 112318|                235|
| 112329|                234|
| 112338|                233|
| 112322|                233|
| 112313|                233|
| 112307|                232|
+-------+-------------------+
only showing top 20 rows
```

Figure 5.37: Query 14: result.

Query 15: Authors by nationality

The following query finds the number of authors of a certain nationality, sorted in decreasing order.

First, authors are grouped by their nationality using the `groupBy()` function. Then, for each group, the number of authors is counted using the aggregate function `count()` and the `alias()` method is used to rename the obtained column in "Number of Authors". All nationality groups with less than 1000 authors are excluded with the `filter()` operation. Lastly, the `sort()` function orders the nationalities based on their number of authors, in descending order. The result of the query is shown in Figure 5.39.

```python
from pyspark.sql.functions import count, col

authors.groupBy("Nationality") \
       .agg(count("Nationality") \
       .alias("Number of Authors")) \
       .filter(col("Number of Authors") > 1000) \
       .sort(col("Number of Authors").desc()).show()
```

Python

Figure 5.38: Query 15.

```
+------------+-----------------+
| Nationality|Number of Authors|
+------------+-----------------+
|       Swiss|             2810|
|     Italian|             2790|
|     Russian|             2787|
|     Bosnian|             2785|
|South African|             2758|
|    Canadian|             2757|
|   Brazilian|             2755|
|      German|             2749|
|     Chinese|             2721|
|     Turkish|             2714|
|  Australian|             2713|
|   Norwegian|             2711|
|      Danish|             2705|
|     Swedish|             2696|
|     Mexican|             2694|
|    Egyptian|             2690|
|     British|             2688|
|  Portuguese|             2683|
|   Giapanese|             2680|
| Argentinian|             2671|
+------------+-----------------+
only showing top 20 rows
```

Figure 5.39: Query 15: result.

Query 16: Types of journals after 1945

This query finds the number of each type of journal founded after 1945.
Firstly, we use the `filter` function to find the journals whose foundation date is after 1945. After that, we use the `groupBy` operator to group the journals by type. Finally, we use the `agg` operator in pair with the `count` operator to count the elements of each group, storing this piece of data with the "Number of Journals" alias, and we show the result.
The result of the query is shown in Figure 5.41.

```python
journals.filter(col("FoundationDate") > "1945") \
        .groupBy("Type").agg(count("ID").alias("Number of Journals")).show()
```

Figure 5.40: Query 16.

```
+-------+-----------------+
|   Type|Number of Journals|
+-------+-----------------+
| online|               22|
|on paper|               15|
+-------+-----------------+
```

Figure 5.41: Query 16: result.

Query 17: Journals publishing last Italian article

The following query searches firstly searches for the year the last article written by an Italian author was published, then searches for the information of all the journals which have published at least one article during that year.

We proceed by using the `filter` operator on the authors with Italian Nationality and we use the `collect` operator to turn the result into a collection. We then create an array containing the IDs of the authors just found. After that, we find the articles' IDs related to those authors by using the `isin` function. Then, we use a `join` function to find the information of the articles whose IDs where previously selected and we get the maximum value of the column year. To do that, we use the `max` function and the `collect` function, that lets us obtain a single value. Finally, we find all the distinct journals whose articles were published during that year by using `filter`, `select` and `distinct` operators.
The result of the query is shown in Figure 5.43.

```python
from pyspark.sql.functions import *

italian_authors = authors.filter(col("Nationality") == "Italian").select(col("ID")).collect()

italian_IDs = [ia[0] for ia in italian_authors]

italian_articles_IDs = written_by.filter(col("Author").isin(italian_IDs))

italian_articles = italian_articles_IDs.join(articles, italian_articles_IDs.Article == articles.ID, "inner")

last_article_year = italian_articles.select(max("Year")).collect()[0][0]

articles.filter(col("Year") == last_article_year).select(col("Journal")).distinct().show()
```

Python

Figure 5.42: Query 17.

```
+-------+
|Journal|
+-------+
| 112304|
| 112340|
| 112331|
| 112324|
| 112322|
| 112334|
| 112326|
| 112310|
| 112314|
| 112316|
| 112311|
| 112325|
...
| 112332|
+-------+
only showing top 20 rows
```

Figure 5.43: Query 17: result.

Query 18: Most productive nationality

The following query finds all the authors with same nationality of the author who published the highest number of articles.

We proceed by finding the highest number of articles published by the same author. To do that, we use the `groupBy` operator to group written_by by author and we use the `agg` operator in pair with the `count` operator to count the rows of each group and we pick the highest value by using the `select` operator and the `max` operator. We now use

the `collect` operator to turn the previous result into a single number. After that, to find the ID of the author with the most publications, we use the `groupBy` operator to group written_by by author and we use the `agg` operator in pair with the `count` operator to count the rows of each group and now we compare the number with the maximum previously found by using a `filter` operator. Then, we find the nationality of the author by using a `join` function between the author ID and the authors dataFrame and then we use the `select` operator to find the nationality. It is possible that more than one author has written the maximum number of articles, so we decide to take the first one in the output using the `collect` function, obtaining a single value. Finally, we find the authors with the nationality just found by using a `filter` operator.
The result of the query is shown in Figure 5.45.

```python
from pyspark.sql.functions import *

max_number_of_articles = written_by.groupBy("Author").agg(count("Article").alias("Number of articles")).select(max("Number of articles")).collect()[0][0]

author_with_most_publications = written_by.groupBy("Author").agg(count("Article").alias("Number of articles")).filter(col("Number of articles") == max_number_of_articles).\
                                select(col("Author"))

nationality_with_most_publications = authors.join(author_with_most_publications, authors.ID == author_with_most_publications.Author).select(col("Nationality")).collect()[0][0]

authors.filter(col("Nationality") == nationality_with_most_publications).show()
```

Python

Figure 5.44: Query 18.

```
+-----+--------------------+---------+-----------+
|   ID|                Name|BirthDate|Nationality|
+-----+--------------------+---------+-----------+
|42044|    Benjamin Hurwitz|     null|    Russian|
|42105|        Helmar Gust|     null|    Russian|
...
|42602|        Paul Leyland|     null|    Russian|
+-----+--------------------+---------+-----------+
only showing top 20 rows
```

Figure 5.45: Query 18: result.

Query 19: Journals with at least 520 articles

The following query shows the name of the journals with more than 520 articles published on them, the exact number of articles and their foundation date. The results are ordered in chronological order based on the journal's foundation date.
To obtain this result, articles have been first joined with the journals they have been published on based on their journal ID (`articles.journal`). This is necessary to retrieve the journal's foundation date.
The result is then grouped by Journal. Name and foundation date of the journal are also present in the `groupBy()` operation in order to have this fields in the final result. This does not influence the query since `Name` and `FoundationDate` are uniquely identified by the Journal ID.

The `agg()` function allows to perform aggregate operations. In this case it is used to count the number of articles for each journal;
Then, a second `filter` operation is performed after the `groupBy`, evaluating a condition on the obtained groups: in this case that the number of articles is greater than 520. This is equivalent to an SQL `HAVING` operator.
Finally, the journal are ordered from the oldest to the most recent one based on their foundation date and relevant attributes are selected.
The result of the query is shown in Figure 5.47.

```python
from pyspark.sql.functions import count, col

articles.join(journals, articles.Journal == journals.ID)\
    .groupBy("Journal", "Name", "FoundationDate").agg(count("Title").alias("Number of articles"))\
        .filter(col("Number of articles") >= 520).sort("FoundationDate").select(col("Name"), col("Number of articles"), col("FoundationDate")).show()
```

Figure 5.46: Query 19.

```
+--------------------+------------------+--------------+
|                Name|Number of articles|FoundationDate|
+--------------------+------------------+--------------+
|Digital System Re...|               522|    1942-03-03|
|   Universität Trier|               531|    1942-04-27|
|Hydrology and Ear...|               522|    1942-08-30|
|IEEE Trans. Veh. ...|               531|    1946-01-05|
|          ETH Zurich|               521|    1948-09-25|
|          Comput. J.|               525|    1953-03-22|
|Des. Codes Cryptogr.|               528|    1954-10-16|
|Research Report /...|               528|    1959-03-10|
|   J. Comput. Secur.|               542|    1968-01-27|
|           ANSI X3H2|               534|    1968-10-16|
|Electron. J. Inf....|               522|    1969-10-21|
+--------------------+------------------+--------------+
```

Figure 5.47: Query 19: result.

Query 20: Most productive authors

The following query returns the nationality of the authors for which there are more articles in the dataset.
Firstly, we form a dataFrame containing the association between articles and their authors, by sequentially using the `join` operator for two times. After that, we use the `groupBy` operator to group by nationality and we use the `agg` function in pair with the `count` function to count the number of articles in each group and we store this piece of data in the "Number of articles" alias. Then we use the `filter` operator to only consider nationalities with at least 2000 articles published. Finally, we use the `sort` function to sort in descending order of number of articles and we show the top 5 ones by using the `limit` operator.
The result of the query is shown in Figure 5.49.

```python
from pyspark.sql.functions import count, col

articles.join(written_by, articles.ID == written_by.Article).join(authors, col("Author") == authors.ID).groupBy("Nationality").agg(count("Title").alias("Number of articles")).\
    filter(col("Number of articles") >= 2000).sort(col("Number of articles").desc()).limit(5).show()
```

Python

Figure 5.48: Query 20.

```
+-------------+-----------------+
|  Nationality|Number of articles|
+-------------+-----------------+
|      Russian|             2423|
|     Egyptian|             2408|
|South African|             2383|
|      Italian|             2374|
|  Argentinian|             2357|
+-------------+-----------------+
```

Figure 5.49: Query 20: result.

Query 21: Authors born after journal foundation

The following query returns the number of articles present in the journals written by an author born after the foundation of the journal. Since an article may have multiple authors, the result of the query represents the number of times the above stated condition is satisfied. Firstly, we form a table containing the association between articles and their authors, by sequentially using the `join` operator for two times. After that, we use the `filter` operator to obtain the rows where the author's date of birth is after the foundation date of the journal where they published their article. Then, we use the `groupBy` operator to aggregate the rows based on the journal and we use the `agg` operator in pair with the `count` operator to find the number of articles present in each group, storing this piece of data with the "Number of articles" alias. Finally, we use the `sort` function to sort in descending order of number of articles and we show the journal names and relative number of articles.
The result of the query is shown in Figure 5.51.

```python
from pyspark.sql.functions import count, col

journals = journals.withColumnRenamed("Name", "Journal_name")

articles.join(written_by, articles.ID == written_by.Article).\
    join(authors, col("Author") == authors.ID).\
        join(journals, col("Journal") == journals.ID).\
            filter(col("BirthDate") >= col("FoundationDate")).\
                groupBy("Journal", "Journal_name").agg(count("Title").alias("Number of articles")).\
                    sort(col("Number of articles").desc()).\
                        select(col("Journal_name"), col("Number of articles")).show()
```

Python

Figure 5.50: Query 21.

```
+--------------------+-----------------+
|        Journal_name|Number of articles|
+--------------------+-----------------+
|   Universit@t Trier|             1478|
|Hydrology and Ear...|             1453|
|IEEE Trans. Compu...|             1443|
|  meltdownattack.com|             1427|
|IEEE Trans. Veh. ...|             1425|
|Digital System Re...|             1422|
|Technical Rep. UK...|             1387|
|          ETH Zurich|             1353|
|Vietnam. J. Compu...|             1320|
|Research Report /...|             1260|
|          LILOG-Memo|             1217|
|IEEE Veh. Technol...|             1193|
|        LILOG-Report|             1177|
|          Comput. J.|             1167|
|Trans. I R E Prof...|             1165|
|Softw. Process. I...|             1140|
|Informationstechn...|             1124|
...
|Research Report /...|             1094|
+--------------------+-----------------+
only showing top 20 rows
```

Figure 5.51: Query 21: result.

A | Appendix A

A.1. Article type 1

```
[
'{{repeat(2000)}}',
{
  _id:'{{objectId()}}',

  title: '{{lorem(1, "sentence")}}',

  authors:[
     '{{repeat(1, 7)}}',
    {
      name: '{{firstName()}} {{surname()}}',
      affiliation: '{{company()}}',
      email : '{{email()}}',
      bio :  '{{lorem(1, "sentence")}}'
    }
  ],

  keywords: [
    '{{repeat(3)}}',
    '{{lorem(1, "words")}}'
  ],

  journal: {
      name: '{{city()}}',
      volume : '{{integer(1, 12)}}',
      number: '{{integer(1, 100)}}',
      date: '{{date(new Date(1970, 0, 1), new Date(2022, 11, 11),
```

```
    "YYYY-MM-dd")}}',
    pages: '{{integer(100, 200)}} - {{integer(201, 400)}}'
  },

abstract: '{{lorem(1, "sentence")}}',

sections: [
//section 1: paragraphs only (no figures, no subsections)
  {
      title: '{{lorem(1, "sentence")}}',

      paragraphs: [
        '{{repeat(1,3)}}',
        '{{lorem(1, "sentence")}}'
      ]
  },
//section 2: paragraphs, figures (no subsections)
    {
      title: '{{lorem(1, "sentence")}}',

      paragraphs: [
        '{{repeat(1,3)}}',
        '{{lorem(1, "sentence")}}'
      ],

      figure:
        {
          url: 'https://ccs-aerospaziale.polimi.it/wp-content/
          uploads/2020/02/01_Polimi_centrato_COL_positivo.png',
          caption: 'Polimi logo'
        }
    },

          //section 3: paragraphs, figures and subsections
{

      title: '{{lorem(1, "sentence")}}',
```

```
        paragraphs: [
          '{{repeat(1,3)}}',
          '{{lorem(1, "sentence")}}'
        ],

        figure: [
          {
            url: 'https://ccs-aerospaziale.polimi.it/wp-content/
            uploads/2020/02/01_Polimi_centrato_COL_positivo.png',
            caption: 'Polimi logo'
          }
      ],

        subsections: [
          '{{repeat(1, 2)}}',
          {
            title: '{{lorem(1, "sentence")}}',

            paragraphs: [
              '{{repeat(1,3)}}',
              '{{lorem(1, "sentence")}}'
            ],

            figure:
              {
                url: 'https://ccs-aerospaziale.polimi.it/wp-content
                /uploads/2020/02/
                01_Polimi_centrato_COL_positivo.png',
                caption: 'Polimi logo'
              }
          }
        ]
      }
  ],

  bibliography: [
```

```
        '{{repeat(3,5)}}',
        '{{integer(100)}}'
      ]
    }
  ]
```

A.2. Article type 2

```
    [
    '{{repeat(2000)}}',
    {
      _id:'{{objectId()}}',

      title: '{{lorem(1, "sentence")}}',

      authors:[
         '{{repeat(1, 7)}}',
        {
          name: '{{firstName()}} {{surname()}}',
          affiliation: '{{company()}}',
          email : '{{email()}}',
          bio :  '{{lorem(1, "sentence")}}'
        }
      ],

      keywords: [
        '{{repeat(3)}}',
        '{{lorem(1, "words")}}'
      ],

      journal: {
          name: '{{city()}}',
          volume : '{{integer(1, 12)}}',
          number: '{{integer(1, 100)}}',
          date: '{{date(new Date(1970, 0, 1), new Date(2022, 11, 11),
          "YYYY-MM-dd")}}',
          pages: '{{integer(100, 200)}} - {{integer(201, 400)}}'
```

```
    },

  abstract: '{{lorem(1, "sentence")}}',

  sections: [
    //section 1: paragraphs, figures and subsections
     {

       title: '{{lorem(1, "sentence")}}',

       paragraphs: [
         '{{repeat(1,3)}}',
         '{{lorem(1, "sentence")}}'
       ],

       figure: [
         {
           url: 'https://ccs-aerospaziale.polimi.it/wp-content/
           uploads/2020/02/01_Polimi_centrato_COL_positivo.png',
           caption: 'Polimi logo'
         }
      ],

       subsections: [
         '{{repeat(1, 2)}}',
         {
           title: '{{lorem(1, "sentence")}}',

           paragraphs: [
             '{{repeat(1,3)}}',
             '{{lorem(1, "sentence")}}'
           ],

           figure:
             {
               url: 'https://ccs-aerospaziale.polimi.it/
               wp-content/uploads/2020/02
```

```
                /01_Polimi_centrato_COL_positivo.png',
                caption: 'Polimi logo'
              }
          }
        ]
      },

    //section 2: paragraphs, figures and subsections (without figures)
    {

      title: '{{lorem(1, "sentence")}}',

      paragraphs: [
        '{{repeat(1,3)}}',
        '{{lorem(1, "sentence")}}'
      ],

      figure: [
        {
          url: 'https://ccs-aerospaziale.polimi.it/wp-content/
          uploads/2020/02/01_Polimi_centrato_COL_positivo.png',
          caption: 'Polimi logo'
        }
      ],

      subsections: [
        '{{repeat(1, 2)}}',
        {
          title: '{{lorem(1, "sentence")}}',

          paragraphs: [
            '{{repeat(1,3)}}',
            '{{lorem(1, "sentence")}}'
          ]
        }
      ]
    }
```

```
    ],

    bibliography: [
      '{{repeat(3,5)}}',
      '{{integer(100)}}'
    ]
  }
]
```

A.3. Article type 3

```
[
'{{repeat(2000)}}',
{
  _id:'{{objectId()}}',

  title: '{{lorem(1, "sentence")}}',

  authors:[
     '{{repeat(1, 7)}}',
    {
      name: '{{firstName()}} {{surname()}}',
      affiliation: '{{company()}}',
      email : '{{email()}}',
      bio :  '{{lorem(1, "sentence")}}'
    }
  ],

  keywords: [
    '{{repeat(3)}}',
    '{{lorem(1, "words")}}'
  ],

  journal: {
      name: '{{city()}}',
      volume : '{{integer(1, 12)}}',
      number: '{{integer(1, 100)}}',
```

```
      date: '{{date(new Date(1970, 0, 1), new Date(2022, 11, 11),
      "YYYY-MM-dd")}}',
      pages: '{{integer(100, 200)}} - {{integer(201, 400)}}'
    },

  abstract: '{{lorem(1, "sentence")}}',

  sections: [
  //section 1: paragraphs only, no figures, no subsection
    {
        title: '{{lorem(1, "sentence")}}',

        // testo opzionale
        paragraphs: [
          '{{repeat(1,3)}}',
          '{{lorem(1, "sentence")}}'
        ]
    },
    //section 2: paragraphs only, no figures, no subsections
      {
        title: '{{lorem(1, "sentence")}}',

        paragraphs: [
          '{{repeat(1,3)}}',
          '{{lorem(1, "sentence")}}'
        ]
    },

    //section 3: paragraphs, figures, subsections
{
        title: '{{lorem(1, "sentence")}}',

        paragraphs: [
          '{{repeat(1,3)}}',
          '{{lorem(1, "sentence")}}'
        ],
```

```
            figure: [
              {
                url: 'https://ccs-aerospaziale.polimi.it/wp-content/
                uploads/2020/02/01_Polimi_centrato_COL_positivo.png',
                caption: 'Polimi logo'
              }
          ],

            subsections: [
              '{{repeat(1, 2)}}',
              {
                title: '{{lorem(1, "sentence")}}',

                paragraphs: [
                  '{{repeat(1,3)}}',
                  '{{lorem(1, "sentence")}}'
                ],

                figure:
                  {
                    url: 'https://ccs-aerospaziale.polimi.it/
                    wp-content/uploads/2020/02
                    /01_Polimi_centrato_COL_positivo.png',
                    caption: 'Polimi logo'
                  }
              }
            ]
          }
      ],

      bibliography: [
        '{{repeat(3,5)}}',
        '{{integer(100)}}'
      ]
    }
  ]
```

B | Appendix B

B.1. Python code

```
import json
import csv
import random


def read_write_json():
    title = 1
    text = 1
    journals = 1
    abstracts = 1
    with open("article_type1.json", "r", encoding='utf-8') as
    read_articles_json:
        json_data = json.load(read_articles_json)
        for i in range(2000): # cambiare con numero articles
            json_data[i]['keywords'] = get_keywords()
            json_data[i]['abstract'] = get_abstract(abstracts)
            abstracts = (abstracts + 1) % 67186
            json_data[i]['journal']['name'] = get_journal(journals)
            journals = (journals + 1) % 20254
            json_data[i]['title'] = get_title(title)
            title = title + 1

            sections_length = len(json_data[i]['sections'])
            for j in range(sections_length):
                json_data[i]['sections'][j]['title'] = get_title(title)
                title = title + 1
                paragraphs_length = len(json_data[i]['sections']
```

```
            [j]['paragraphs'])
            for k in range(paragraphs_length):
                json_data[i]['sections'][j]['paragraphs'][k] =
                get_text(text)
                text = (text + 1) % 34999
        if j == 2:
            subsections_length = len(json_data[i]['sections']
            [j]['subsections'])
            for a in range(subsections_length):
                json_data[i]['sections'][j]['subsections'][a]['title'] =
                get_title(title)
                title = (title + 1) % 67186
                subsection_paragraphs_len = len(json_data[i]['sections']
                [j]['subsections'][a]['paragraphs'])
                for m in range(subsection_paragraphs_len):
                    json_data[i]['sections'][j]['subsections'][a]
                    ['paragraphs'][m] = get_text(text)
                    text = (text + 1) % 34999

        bib_length = len(json_data[i]['bibliography'])
        for k in range(bib_length):
            json_data[i]['bibliography'][k] = get_id()

        with open('output_articles_type1.json', 'w', encoding='utf-8')
        as write_articles_json:
            json.dump(json_data, write_articles_json)
            write_articles_json.close()
        print(str(i) + "␣done.")
    read_articles_json.close()
    write_articles_json.close()



    with open("article_type2.json", "r", encoding='utf-8') as
    read_articles_json:
        json_data = json.load(read_articles_json)
        for i in range(2000):
```

```
json_data[i]['keywords'] = get_keywords()
json_data[i]['abstract'] = get_abstract(abstracts)
abstracts = (abstracts + 1) % 67186
json_data[i]['journal']['name'] = get_journal(journals)
journals = (journals + 1) % 20254
json_data[i]['title'] = get_title(title)
title = (title + 1) % 67186

sections_length = len(json_data[i]['sections'])
for j in range(sections_length):
    json_data[i]['sections'][j]['title'] =
    get_title(title)
    title = (title + 1) % 67186
    paragraphs_length = len(json_data[i]['sections']
    [j]['paragraphs'])
    for k in range(paragraphs_length):
        json_data[i]['sections'][j]['paragraphs'][k] =
        get_text(text)
        text = (text + 1) % 34999
    subsections_length = len(json_data[i]['sections']
    [j]['subsections'])
    for a in range(subsections_length):
        new_title = get_title(title)
        json_data[i]['sections'][j]['subsections'][a]
        ['title'] = new_title
        title = (title + 1) % 67186
        subsection_paragraphs_len = len(json_data[i]
        ['sections'][j]['subsections'][a]['paragraphs'])
        for m in range(subsection_paragraphs_len):
            json_data[i]['sections'][j]['subsections']
            [a]['paragraphs'][m] = get_text(text)
            text = (text + 1) % 34999

bib_length = len(json_data[i]['bibliography'])
for k in range(bib_length):
    json_data[i]['bibliography'][k] = get_id()
```

```
            with open('output_articles_type2.json', 'w',
            encoding='utf-8') as write_articles_json:
                json.dump(json_data, write_articles_json)
                write_articles_json.close()
            print(str(i) + "␣done.")
        read_articles_json.close()
        write_articles_json.close()

with open("article_type3.json", "r", encoding='utf-8') as
read_articles_json:
    json_data = json.load(read_articles_json)
    for i in range(2000):
        json_data[i]['keywords'] = get_keywords()
        json_data[i]['abstract'] = get_abstract(abstracts)
        abstracts = (abstracts + 1) % 67186
        json_data[i]['journal']['name'] = get_journal(journals)
        journals = (journals + 1) % 20254
        json_data[i]['title'] = get_title(title)
        title = (title + 1) % 67186

        sections_length = len(json_data[i]['sections'])
        for j in range(sections_length):
            json_data[i]['sections'][j]['title'] = get_title(title)
            title = (title + 1) % 67186
            paragraphs_length = len(json_data[i]['sections']
            [j]['paragraphs'])
            for k in range(paragraphs_length):
                json_data[i]['sections'][j]['paragraphs'][k] =
                get_text(text)
                text = (text + 1) % 34999
        if j == 2:
            subsections_length = len(json_data[i]['sections'][j]
            ['subsections'])
            for a in range(subsections_length):
                json_data[i]['sections'][j]['subsections'][a]
                ['title'] = get_title(title)
                title = (title + 1) % 67186
```

```
                        subsection_paragraphs_len = len(json_data[i]
                        ['sections'][j]['subsections'][a]['paragraphs'])
                        for m in range(subsection_paragraphs_len):
                            json_data[i]['sections'][j]['subsections'][a]
                            ['paragraphs'][m] = get_text(text)
                            text = (text + 1) % 34999

                bib_length = len(json_data[i]['bibliography'])
                for k in range(bib_length):
                    json_data[i]['bibliography'][k] = get_id()

                with open('output_articles_type3.json', 'w',
                encoding='utf-8') as write_articles_json:
                    json.dump(json_data, write_articles_json)
                    write_articles_json.close()
                print(str(i) + "␣done.")
            read_articles_json.close()
            write_articles_json.close()


def get_journal(index):
    with open('journals.csv', encoding='Latin-1') as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        rows = list(csv_reader)
        ret = rows[index][0]
        csv_file.close()
    return ret


def get_abstract(index):
    with open('abstracts.csv', encoding='utf-8') as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        rows = list(csv_reader)
        ret = rows[index][0]
        csv_file.close()
    return ret
```

```
def get_text(index):
    with open('reviews.csv', encoding='utf-8') as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        rows = list(csv_reader)
        ret = rows[index][0]
        csv_file.close()
    return ret


def get_keywords():
    text_file = open("keywords.txt", "r")
    lines = text_file.readlines()
    text_file.close()
    rnd_1 = random.randint(1, 98)
    rnd_2 = random.randint(1, 98)
    while rnd_1 == rnd_2:
        rnd_2 = random.randint(1, 98)
    rnd_3 = random.randint(1, 98)
    while rnd_3 == rnd_1 or rnd_3 == rnd_2:
        rnd_3 = random.randint(1, 98)

    out_1 = lines[rnd_1][:(len(lines[rnd_1]) - 1)]
    out_2 = lines[rnd_2][:(len(lines[rnd_2]) - 1)]
    out_3 = lines[rnd_3][:(len(lines[rnd_3]) - 1)]

    text_file.close()

    return [out_1, out_2, out_3]


def get_title(index):
    with open('titles.csv', encoding='utf-8') as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        rows = list(csv_reader)
        ret = rows[index][0]
        csv_file.close()
```

```
    return ret


def get_id():
    article_type = random.randint(1,3);
    with open("article_type" + str(article_type) + ".json", "r",
    encoding='utf-8') as read_articles_json:
        json_data = json.load(read_articles_json)
        idx = random.randint(0, len(json_data)-1)
        ret_idx = json_data[idx]['_id']
        read_articles_json.close()
    return ret_idx


def main():
    read_write_json()


if __name__ == '__main__':
    main()
```

List of Figures

List of Tables