POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Systems and Methods for Big and Unstructured Data Project

Author(s): **Nicolò Caleffi**
Davide Esposito
Simone Garbazio
Francesco Pastori
Gianluca Vigo

Group Number: **18**

Academic Year: 2022-2023

Contents

1 | Introduction

In this chapter there is a presentation of the general problem. In particular, the scope of the project is to build a bibliographic database.

1.1. The problem

The bibliography is an element of great importance in a scientific work: it is the organized list of sources and resources (books, articles, essays, web pages, conferences, etc.). They are actually consulted and used by the researcher during his work because they are considered of some relevance for his own investigation.

Drafting a good bibliography gives value and scientific value to the work. In fact, the list of sources consulted provides clues for a first assessment of the quality of the survey conducted and the bibliography is often the part that is immediately consulted by the competent reader, together with title, summary and citation apparatus. The bibliography in our times must confront with the use of digital tools and the offer of increasingly innovative services, while keeping the user at the center of the organized world of knowledge.

We have been talking about the digital library for decades, because it is clearly a projection that is perfectly in line with the concepts of speed and reachability that permeate the historical moment in which we live and our society.

A bibliographical database is an electronic archive of bibliographic references that can be dynamically interrogated. Typically, it contains only citations of articles, often with abstracts, with a link to the full text version of the articles or their localization. Compared to a search conducted on a generic engine, a database offers numerous advantages in the quality and quantity of the results obtainable. In fact, the scientific value of the information is guaranteed by the human selection of the contents carried out by qualified scientific committees.

1.2. Specification and hypotheses

1.2.1. Specification

The analysis of the specifications consists in the clarification and organization of the information collected.

Author-related phrases

Author of a written work, which may or may not be published. Every author has personal data such as name, surname, and the list of different affiliations where he has carried out research. In addition, a researcher can have an ORCID that uniquely identifies him.

Affiliation-related phrases

In academic publishing, the affiliation of an author is the place (institution) at which the author conducted the research that they have reported/written about.

Editor-related phrases

In a publication, the editor is basically who edits the manuscripts or the articles before submitting it for approval to the publisher. An editor's job is to really make a literary work that meets up to the standard of the content.

Publisher-related phrases

As the Editor, the publisher also play vital roles in turning a manuscript into a book and an article to appear in a magazine. A publisher basically decides whether or not a literary material is good enough to be profited from.

Article (research paper)-related phrases

A Research paper is a work by one or more authors that can be published. If an article has been published, it can have its own DOI, which uniquely identifies it.

Publishing-related phrases

Publishing is the entrepreneurial activity of production, management of contents repro-

ducible in series, and of their diffusion and commercialization in forms communicable through the mass media. There are different methods of publication and those considered within the present reality of interest will be mentioned below.

Book-related phrases

A Book (as the monograph) is a work by one or more authors that can be published. If a book (monograph) has been published, it can have its own DOI, which uniquely identifies it. Moreover, if it is a printed publication it also has the ISBN code.

Conference-related phrases

Conference (Scientific congress) of researchers (not necessarily academics) to present and discuss the results of their work. With universities and scientific communication, congresses are an important channel for the exchange of information between scholars. Usually, a printed publication is issued, called conference or congress proceedings, as a report of the event itself.

Journal (Scientific Journal)-related phrases

Collection of academic research papers written by several authors based on a theme and published from time to time. A journal has its own ISSN and is collected into volumes, each of which is composed of journal numbers. The latter contain several publications by one or more authors.

Series-related phrases

Collection of publications of one or more different authors, also of different types, which also cover different topics of the same field.

Identification codes of the reality of interest

DOI-related phrases

The DOI (Digital Object Identifier) is a standard that allows to identify persistently, within a digital network, any object of intellectual property and to associate its reference data, metadata, according to a structured and extensible scheme. An object can be arbitrarily identified at any level of granularity: this means that, for example, you can register a DOI on the head of a magazine, on its single number, on the single article of a given number, on the single table of a given article. Hence, each DOI uniquely identifies the object it is associated with in time.

ISBN-related phrases

The International Standard Book Number (ISBN) is a numerical identifier of commercial books that must be unique. Publishers purchase ISBN from an affiliate of the international ISBN agency. An ISBN is assigned to each edition and separate variation (except reprints) of a publication. The ISBN consists of ten digits if allocated before 2007 and thirteen digits if allocated from 1 January 2007

ORCID-related phrases

ORCID disambiguates researchers, and connects people with their research activities. By using ORCID in a system, you can uniquely identify the researchers and discover information about them. This includes employment affiliations, research outputs, funding, peer review activity, research resources, society membership, distinctions and other scholarly infrastructure.

ISSN-related phrases

An 8-digit code used to identify newspapers, journals, magazines and periodicals of all kinds and on all media–print and electronic.

1.2.2. Hypotheses

Starting from the general specification of the problem, we have made the following hypotheses:

- Not all publications have a DOI code. If they have one then it uniquely identifies that publication.
- Not all researchers have an ORCID code. If they have one then it uniquely identifies that researcher.
- Conferences can have different editions and for each one of them more than one Conference Proceeding can be released.
- Given a published work, the system stores the main information, the abstract (if available) and its localization (with DOI code and URL).
- Given an unpublished work, the system stores only the main information and the abstract (if available).
- An author can have different affiliations.
- Any type of work can cover several topics, that can be described by a set of keywords.
- The publisher of a journal number is the journal itself.
- Every volume of a journal corresponds to the year of publication.
- Every book is published.

2 | Implementation

2.1. The ER diagram

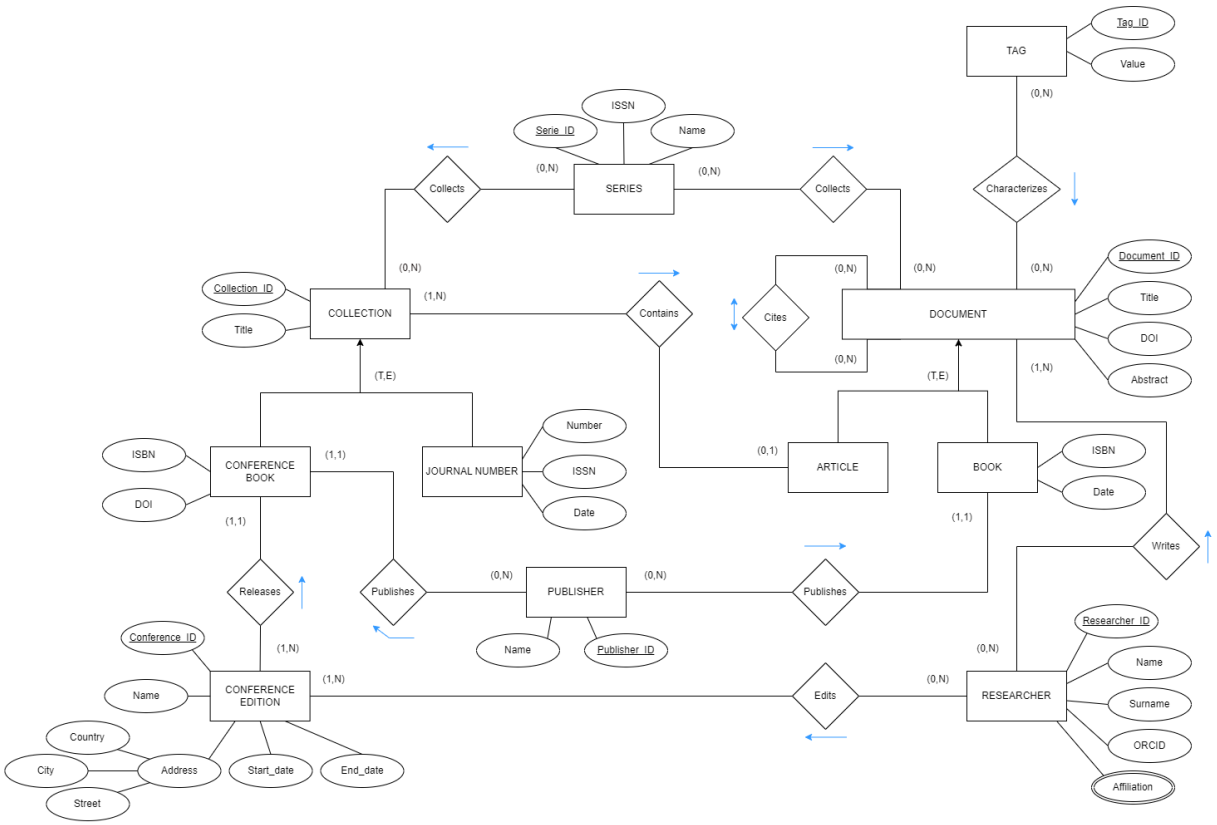


Figure 2.1: The ER model of our Bibliographic Database

The structure of our ER diagram is mainly derived from the previous general study and the consequent assumptions that we have made. Furthermore, to detect the importance of the various entities (and their attributes) we observed various bibliographic databases that already exist, such as DPLB, a bibliographic database completely dedicated to computer science publications.

We started the analysis from what we perceived as the most important concepts of the

world that we are trying to schematize: the **document** and the **collection**.

By document we mean a written work that contains the work of a researcher on a given topic, that is usually suggested by its *Title*. A document has to be related to something else in order to be considered as *published*. It can be characterized by a *DOI*, that stands for *Digital Object Identifier*, and an *Abstract*, a brief description of the topics that are covered by the document. The *DOI* is a standard that allows to identify any object of intellectual property, but not every document has a DOI, in particular the older ones. In our case, the DOI field is a URL that contains the DOI code. This URL points to the web page in which the article is located.

We have detected two different specializations for the document: **Article** and **Book**. The former needs to be contained in a collection in order to be considered as "published", while the latter exists and makes sense as an entity in its own right, but it still needs a **Publisher**, that is responsible for its release. Furthermore, the book is always identified by a *ISBN* code and a publication *Date*.

In our model, the Publisher is characterized only by a *Name*.

An article is linked with a collection by the *Contains* relationship, that states that an article is contained (at most) into one collection. The articles that are not connected to any collection are considered as **unpublished**.

The ***Cites*** relationship that links the Document entity with itself is used to represent that some documents may contain citations or refer to other documents.

We now come to the **Collection** entity. This entity is considered as a "container" for Articles. Like Documents, Collections are always characterized by a *Title*.

In this case, the two different specializations that we have detected are **Conference Book** and **Journal Number**. The Conference Book (i.e., Conference Proceeding) is a collection that is produced after a conference is held and its Articles are about the different topics that were presented or discussed during the Conference. Since it is an actual book, it has an *ISBN* code, a *DOI* code and it is related to a Publisher. As stated before, Articles can also be published on Journals, so the Journal Number entity represents a single issue of a Journal and it is characterized by a *Number* and a publication *Date*. The *ISSN* code is assigned to a Journal and it is not different between the various Numbers published by the same Journal The *Contains* relationship links the Journal Number with the Articles that it contains.

A special note for the **Journal**: in general, a Journal is characterized by an *ISSN* code and a *Title*. Thus, the Journal Number could be represented as a weak entity of the Journal

itself. But, since we have decided to represent the Journal Number as a specialization of Collection, we store the *ISSN* code and the *Title* of the Journal in the Journal Number entity. Obviously, Numbers that belong to the same Journal will have the *Title* and *ISSN* fields identical to each other, but they will be different in *Number* and *Date*.

Since the Conference Books are the "result" of a Conference, we have a link between them and a **Conference Edition** represented by the *Releases* relationship. A Conference Edition is characterized by a *Name*, the *Address* where the Conference Edition is held (a composite attribute in our model), a *Start Date* and an *End Date*. It is possible that there is more than one Conference Book related to a single Conference Edition.

Every Document has at least one Author. The Author is included in our ER with the more generic **Researcher** entity. This choice was made to include into one entity the concepts of "author of documents" and "editor of conferences". In fact, the Researcher is linked with Documents by the *Writes* relationship and with the Conference Editions with the *Edits* relationship. A Conference Edition has at least one Researcher who plays the role of Editor, while a Document has at least one **Researcher** who plays the role of **Author**.

The Researcher is characterized by a *Name*, a *Surname*, a list of *Affiliations* (multiple attribute) and an *ORCID* code. We decided not to use the *ORCID* code as a Primary Key for Researcher because not all Researchers have one.

In conclusion, a brief description of the remaining two entities: **Tag** and **Series**. Tags are keywords (stored in the *Value* attribute) that indicate the content of a Document (e.g. "Big Data", "Data Mining", etc.). The Collections are supposed to inherit the Tags that are linked to the contained Articles.

Series are a collection of Documents or Collections. They are characterized by an *ISSN* code and a *Title*.

As a design choice, we decided to assign each entity an integer primary key, a solution often used in other DBs for performance reasons.

2.2. Neo4J implementation

2.2.1. Amendments to the ER

In this section are reported some minor adjustments that we have made to our ER schema before implementing it into Neo4j.

- Hierarchies: all the hierarchies have been resolved downwards. The attributes of Document and Collection were included into the specialized entities. Anyway, since Neo4j allows it, we have attached:
  - the Document label to Articles and Books nodes.
  - the Collection label to Conference Books and Journal Numbers.

  Note that the ID field used as primary key is intended as unique within the "sons" entities.
- Document: the *Abstract* attribute was not considered because we do not have it into our dataset.
- Date attribute of Journal Number: due to our dataset, the *Date* attribute of Journal Number has been divided into two attributes: *Month* and *Year*.
- Date attribute of Book: due to our dataset, the *Date* attribute of Book contains basically a year, so we renamed it *Year*.
- Name and Surname attributes of Researcher were merged into a single attribute.
- Affiliation attribute on Researcher: since it was intended as a multiple attribute, it became a standalone table: `AFFILIATION(Affiliation_ID, Affiliation_Name)`

2.2.2. Dataset

Creation of the entities

We used different tools to build our dataset: **entities** were populated thanks to different Python programs, Mockaroo and the web. The relationships were defined using Excel and another Python program.

We started from the **DBLP dump**. The first Python program (`https://github.com/ThomHurks/dblp-to-CSV`) was able to convert this XML dump into different CSV files but, due to the high volume of data and to its low level of completeness, we took only samples from them. A long operation of control was necessary in order to highlight the

missing values. Then, we inserted realistic and feasible data to complete our dataset. In this way, we created smaller but more complete CSV files and so different entities were ready to be used such as: Researcher, Article, Book, Conference Book, Series, Affiliation and Publisher.

Note that some of the constraints that we have made on the uniqueness of some attributes were discarded because we have replaced some empty rows with default values (as in the case of the DOI and ORCID).

Researcher_ID	Researcher_Name	ORCID
1	Russell Turpin	0103-1034-5433-6784
2	Frank Olken	0104-1035-5434-6785
3	Guido Frisch	0105-1036-5435-6786
4	Piero Borga	0103-1034-5433-6785

Figure 2.2: The Researcher Table

Article_ID	Article_Title	DOI
1	Towards Deeper Understanding of the Search Interfaces of the Deep Web.	https://doi.org/10.1007/s11280-006-0010-9
2	Information Extraction from Web Pages Using Presentation Regularities and Domain Knowledge.	https://doi.org/10.1007/s11280-007-0021-1
3	Improving the Performance of Online Auctions Through Server-side Activity-based Caching.	https://doi.org/10.1007/s11280-006-0011-8
4	Patterns Based Classifiers.	https://doi.org/10.1007/s11280-006-0012-7

Figure 2.3: The Article Table

Book_ID	Book_Title	DOI	Year	ISBN
1	The Semantic Web: Real-World Applications from Industry	https://doi.org/10.1007/978-1-4939-0790-8	2016	978-3-89012-177-2
2	Semantic Management of Middleware	https://doi.org/10.1007/978-1-4614-8806-4	2007	978-3-8244-0003-4
3	Social Networks and the Semantic Web	https://doi.org/10.1007/978-3-662-43505-2	2022	978-3-446-15577-0
4	Semantic Web Services, Processes and Applications	https://doi.org/10.1007/978-3-540-92913-0	2021	978-3-88585-788-4

Figure 2.4: The Book Table

Conference_Book_ID	Conference_Book_Title	ISBN	DOI
1	Handbook of Genetic Programming Applications	978-3-319-20882-4	https://doi.org/10.1007/978-3-319-20883-1
2	Handbook of Medical and Healthcare Technologies	978-1-4614-8494-3	https://doi.org/10.1007/978-1-4614-8495-0
3	On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated Int	3-540-20498-9	https://doi.org/10.1007/b94348
4	On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE, OTM Confederated Inte	3-540-23662-7	https://doi.org/10.1007/b102176

Figure 2.5: The Conference Book Table

Serie_ID	Serie_Name	ISSN
1	Technical Report / University of Wisconsin, Madison / Computer Sciences Department	5852-7358
2	DISDBIS	2931-9917
3	Schriften zur Informationswissenschaft	9574-3032
4	Mathematical Centre Tracts	2112-7208

Figure 2.6: The Series Table

Publisher_ID	Publisher_Name
1	LBL Technical Report
2	Excelsior
3	Aka Akademische Verlagsgesellschaft Aka GmbH, Berlin
4	Verlag Dr. Hut

Figure 2.7: The Publisher Table

The second Python program was developed by us and it was necessary to define the Journal Number entity. Given a list of Journals and their ISSNs, the script creates a CSV file containing different Journal Numbers characterized by a *Number*, a *Month* and a *Year*.The number of the single journal instance is incremental (starting from 1) within the same year.

Journal_ID	Journal_Title	ISSN	Number	Month	Year
1	World Wide Web	8660-6338	1	7	1990
2	World Wide Web	8660-6338	1	7	2005
3	SIGMOD Rec.	8492-2950	1	2	2014
4	SIGMOD Rec.	8492-2950	1	2	2019

Figure 2.8: The Journal Number Table

The Mockaroo website (https://mockaroo.com/) is a random data generator and outputs CSV files and this was mainly used to create the Conference Edition entity. It was mainly used to generate places and dates for different editions of the same Conference.

Conference_ID	Country	City	Street	Start_Date	End_Date	Name
1	China	Tanshan	Oriole	02/03/2018	05/03/2018	Surgery Simulation and Soft Tissue Modeling
2	Honduras	Ocote Paulino	Evergreen	12/09/2002	15/09/2002	Designing Interactive Systems
3	Belarus	Kalodzishchy	Donald	11/09/2014	14/09/2014	Smart Objects and Technologies for Social Good GOODTECHS
4	China	Nanjing	Merchant	15/06/2019	18/06/2019	IFCIS International Conference on Cooperative Information Systems

Figure 2.9: The Conference Edition Table

The Tag entity was filled up with computer science related keywords that we found on the web.

Tag_ID	Tag_Name
1	Big Data Analytics
2	Artificial Intelligence (AI)
3	Machine Learning (ML)
4	Natural Language Processing (NLP)

Figure 2.10: The Tag Table

Finally, we have collected more than 3000 entries in our entities.

Creation of the relationships

The relationships were mainly created by using Excel functionalities: in fact, most of them were implemented in such a way that one half of the relation is manually checked and the other end could be random. All the relationships were created as a table of two columns where each row is a tuple composed of the keys of the related elements (e.g. the table of the *Characterizes* relationship between Tags and Articles is composed by a *Tag_ID* column and a *Article_ID* one).

Another Python program was used to define the relationships involving the Tags: in order to avoid meaningless links (for instance, a book regarding Web development associated to the "Data Mining" tag), the program verifies if the title of a Document contains a given Tag. If so, the association between them is created.

Since the DBLP dataset was only our starting point, our dataset cannot be considered as a representation of reality. Nevertheless, we tried our best to present at least plausible data.

Once all the CSV files were created, each corresponding to an entity or a relationship of the ER schema, they were allocated in the "import" folder inside the Neo4j folder, placed on the personal computer.

Import in Neo4j

In order to populate the database, we took advantage of the importing capabilities of Neo4j, which provides commands for loading data directly from CSV files. Extra care was punt into writing load commands that avoided the presence both of duplicate entities and of identical relations between the same nodes.

In addition to the imported data, in some cases, extra nodes and relations were added to better display the functioning of the queries.

In the following chapter are presented the LOAD commands first for the entities and then for the relationships.

1. **Article**

```
LOAD CSV WITH HEADERS FROM "file:///Entities/Article.CSV" AS row
FIELDTERMINATOR ';'
MERGE (a:Document:Article{id:row.Article_ID})
ON CREATE SET a.title=row.Article_Title, a.DOI=row.DOI
```

2. **Affiliation**

```
LOAD CSV WITH HEADERS FROM "file:///Entities/Affiliation.CSV" AS row
FIELDTERMINATOR ';'
MERGE (a:Affiliation{id:row.Affiliation_ID})
ON CREATE SET a.name=row.Affiliation_Name
```

3. **Book**

```
LOAD CSV WITH HEADERS FROM "file:///Entities/Book.CSV" AS row
FIELDTERMINATOR ';'
MERGE (b:Document:Book{id:row.Book_ID})
ON CREATE SET b.title=row.Book_Title, b.DOI=row.DOI, b.year=row.Year,
    b.ISBN=row.ISBN
```

4. **ConferenceBook**

```
LOAD CSV WITH HEADERS FROM "file:///Entities/ConferenceBook.CSV" AS
    row
FIELDTERMINATOR ';'
MERGE (cb:ConferenceBook{id:row.Conference_Book_ID})
ON CREATE SET cb.title=row.Conference_Book_Title, cb.ISBN=row.ISBN, cb
    .DOI=row.DOI
```

5. **JournalNumber**

```
LOAD CSV WITH HEADERS FROM "file:///Entities/JournalNumber.CSV" AS row
FIELDTERMINATOR ';'
MERGE (jn:JournalNumber{id:row.Journal_ID})
ON CREATE SET jn.title=row.Journal_Title, jn.ISSN=row.ISSN, jn.number=
    row.Number, jn.publication_date=date({year:toInteger(row.Year),
    month:toInteger(row.Month)})
```

6. **Publisher**

```
LOAD CSV WITH HEADERS FROM "file:///Entities/Publisher.CSV" AS row
FIELDTERMINATOR ';'
MERGE (p:Publisher{id:row.Publisher_ID})
ON CREATE SET p.name=row.Publisher_Name
```

7. **Researcher**

```
LOAD CSV WITH HEADERS FROM "file:///Entities/Researcher.CSV" AS row
FIELDTERMINATOR ';'
MERGE (r:Researcher{id:row.Researcher_ID})
ON CREATE SET r.name=row.Researcher_Name, r.ORCID=row.ORCID
```

8. **Serie**

```
LOAD CSV WITH HEADERS FROM "file:///Entities/Serie.CSV" AS row
FIELDTERMINATOR ';'
MERGE (s:Serie{id:row.Serie_ID})
ON CREATE SET s.name=row.Serie_Name, s.ISSN=row.ISSN
```

9. **Tag**

```
LOAD CSV WITH HEADERS FROM "file:///Entities/Tag.CSV" AS row
FIELDTERMINATOR ';'
MERGE (t:Tag{id:row.Tag_ID})
ON CREATE SET t.name=row.Tag_Name
```

10. **ConferenceEdition**

```
LOAD CSV WITH HEADERS FROM "file:///Entities/ConferenceEdition.CSV" AS
    row
FIELDTERMINATOR ';'
MERGE (ce:ConferenceEdition{id:row.Conference_ID})
ON CREATE SET ce.country=row.Country, ce.city=row.City, ce.street=row.
    Street, ce.start_date=date(row.Start_Date), ce.end_date=date(row.
    End_Date), ce.name=row.Name
```

11. **Researcher-AffiliatesTo-Affiliation**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/Researcher-
    affiliatesTo-Affiliation.CSV" AS row
FIELDTERMINATOR ';'
MATCH (r:Researcher{id:row.Researcher_ID}), (a:Affiliation{id:row.
    Affiliation_ID})
MERGE (r)-[:AffiliatesTo]->(a)
```

12. **Researcher-Writes-Article**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/Researcher-writes-
    Article.CSV" AS row
FIELDTERMINATOR ';'
MATCH (r:Researcher{id:row.Researcher_ID}), (a:Article{id:row.
    Article_ID})
MERGE (r)-[:Writes]->(a)
```

13. **Article-Cites-Article**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/Article-cites-
    Article.CSV" AS row
FIELDTERMINATOR ';'
```

```
MATCH (a:Article{id:row.Article1_ID}), (ab:Article{id:row.Article2_ID
    })
MERGE (a)-[:Cites]->(ab)
```

14. **Article-Cites-Book**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/Article-cites-Book.
    CSV" AS row
FIELDTERMINATOR ';'
MATCH (a:Article{id:row.Article_ID}), (b:Book{id:row.Book_ID})
MERGE (a)-[:Cites]->(b)
```

15. **Book-Cites-Article**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/Book-cites-Article.
    CSV" AS row
FIELDTERMINATOR ';'
MATCH (a:Article{id:row.Article_ID}), (b:Book{id:row.Book_ID})
MERGE (b)-[:Cites]->(a)
```

16. **Book-Cites-Book**

```
PR6 Book cites Book
LOAD CSV WITH HEADERS FROM "file:///Relationships/Book-cites-Book.CSV"
     AS row
FIELDTERMINATOR ';'
MATCH (b1:Book{id:row.Book1_ID}), (b2:Book{id:row.Book2_ID})
create (b1)-[:Cites]->(b2)
```

17. **ConferenceBook-Contains-Article**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/ConferenceBook-
    contains-Article.CSV" AS row
FIELDTERMINATOR ';'
MATCH (cb:ConferenceBook{id:row.Conference_Book_ID}), (a:Article{id:
    row.Article_ID})
MERGE (cb)-[:Containss]->(a)
```

18. **JournalNumber-Contains-Article**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/JournalNumber-
    contains-Article.CSV" AS row
FIELDTERMINATOR ';'
MATCH (j:JournalNumber{id:row.Journal_ID}), (a:Article{id:row.
    Article_ID})
MERGE (j)-[:Containss]->(a)
```

19. **Publisher-Publishes-Book**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/Publisher-publishes-
    Book.CSV" AS row
FIELDTERMINATOR ';'
MATCH (p:Publisher{id:row.Publisher_ID}), (b:Book{id:row.Book_ID})
MERGE (p)-[:Publishes]->(b)
```

20. **Publisher-Publishes-ConferenceBook**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/Publisher-publishes-
    ConferenceBook.CSV" AS row
FIELDTERMINATOR ';'
MATCH (p:Publisher{id:row.Publisher_ID}), (cb:ConferenceBook{id:row.
    Conference_Book_ID})
merge (p)-[:Publishes]->(cb)
```

21. **Researcher-Writes-Book**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/Researcher-writes-
    Book.CSV" AS row
FIELDTERMINATOR ';'
MATCH (r:Researcher{id:row.Researcher_ID}), (b:Book{id:row.Book_ID})
MERGE (r)-[:Writes]->(b)
```

22. **Serie-Collects-Article**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/Serie-collects-
    Article.CSV" AS row
FIELDTERMINATOR ';'
MATCH (s:Serie{id:row.Serie_ID}), (a:Article{id:row.Article_ID})
MERGE (s)-[:Collects]->(a)
```

23. **Serie-Collects-Book**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/Serie-collects-Book.
    CSV" AS row
FIELDTERMINATOR ';'
MATCH (s:Serie{id:row.Serie_ID}), (b:Book{id:row.Book_ID})
MERGE (s)-[:Collects]->(b)
```

24. **Serie-Collects-ConferenceBook**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/Serie-collects-
    ConferenceBook.CSV" AS row
FIELDTERMINATOR ';'
MATCH (s:Serie{id:row.Serie_ID}), (cb:ConferenceBook{id:row.
    Conference_Book_ID})
MERGE (s)-[:Collects]->(cb)
```

25. **Serie-Collects-JournalNumber**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/Serie-collects-
    JournalNumber.CSV" AS row
FIELDTERMINATOR ';'
MATCH (s:Serie{id:row.Serie_ID}), (jn:JournalNumber{id:row.Journal_ID
    })
MERGE (s)-[:Collects]->(jn)
```

26. **Tag-Characterizes-Book**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/Tag-
    characterizes_Book.CSV" AS row
FIELDTERMINATOR ';'
MATCH (t:Tag{id:row.Tag_ID}), (b:Book{id:row.Book_ID})
MERGE (t)-[:Characterizes]->(b)
```

27. **Tag-Characterizes-Article**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/Tag-characterizes-
    Article.CSV" AS row
FIELDTERMINATOR ';'
MATCH (t:Tag{id:row.Tag_ID}), (a:Article{id:row.Article_ID})
MERGE (t)-[:Characterizes]->(a)
```

28. **ConferenceEdition-Releases-ConferenceBook**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/ConferenceEdition-
    releases-ConferenceBook.CSV" AS row
FIELDTERMINATOR ';'
MATCH (ce:ConferenceEdition{id:row.Conference_ID}), (cb:ConferenceBook
    {id:row.Conference_Book_ID})
MERGE (ce)-[:Releases]->(cb)
```

29. **Researcher-Edits-ConferenceEdition**

```
LOAD CSV WITH HEADERS FROM "file:///Relationships/Researcher-Edits-
    ConferenceEdition.CSV" AS row
FIELDTERMINATOR ';'
MATCH (ce:ConferenceEdition{id:row.Conference_ID}), (r:Researcher{id:
    row.Researcher_ID})
MERGE (r)-[:Edits]->(ce)
```

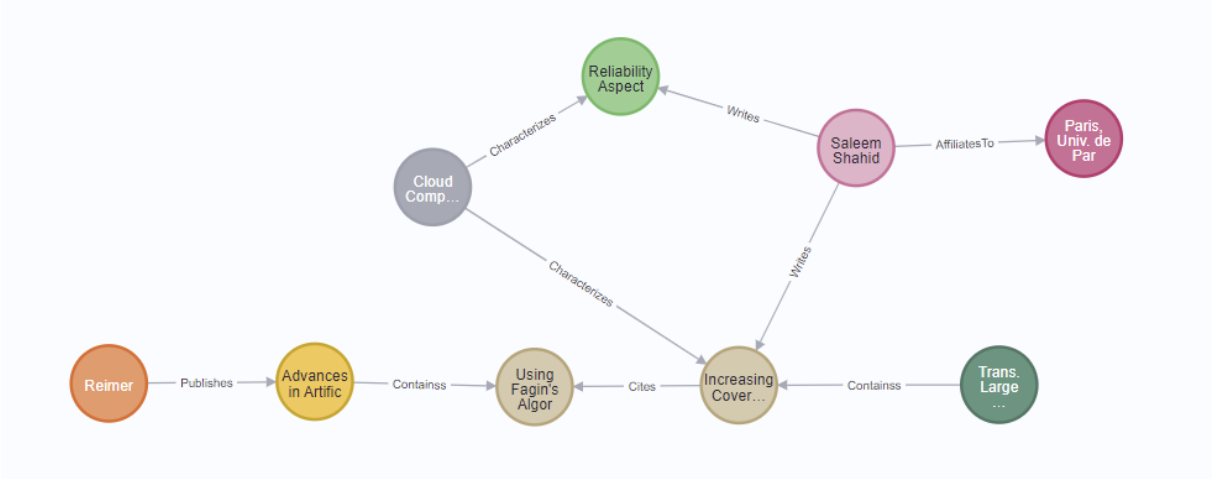


Figure 2.11: DB portion

2.2.3. Queries

In this chapter are reported and commented some of the possible queries that can be executed on the database. For each statement is provided a short description, the code and an example of the result produced upon its execution.

Q1. Published books per publisher

This query returns the number of published books by year by a given publisher. It returns a list with the entries containing the year and the number of publications in that year. Besides, the books are filtered according to a set of defined tags and the returned list is sorted in descending order with respect to the year.

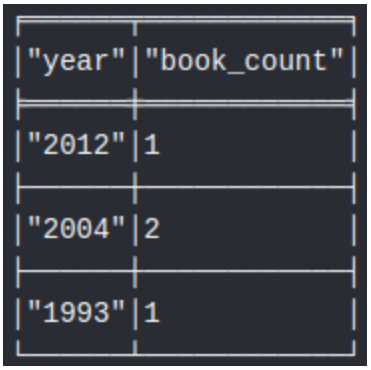
Parameters:

1. $publisher_id: the id of the interested publisher
2. $tag_values: the list of tags for the filtering of the articles

```
MATCH (p:Publisher)-[:Publishes]->(b:Book)<-[:Characterizes]-(t:Tag)
WHERE p.id=$publisher_id
WITH b, collect(DISTINCT(t.name)) AS t_values
WHERE all(x IN $tag_values WHERE x in t_values)
WITH b.year AS year, COUNT(DISTINCT(b)) AS book_count
ORDER BY year DESC
RETURN year, book_count
```

Result by setting:

1. $publisher_id: "90"
2. $tag_values: ["Artificial Intelligence (AI)", "Data Mining"]



"year"	"book_count"
"2012"	1
"2004"	2
"1993"	1

Figure 2.12: Results of the query Q1

Performance

Performance resulting from an execution of the query in our dataset (around 3000 nodes):

- Started streaming 3 records after 40 ms and completed after 43 ms - 896 total db hits in 43 ms

Q2.Top researcher of an affiliation for number of citations to its works

This query returns the best researcher in a given affiliation. The best researcher is picked depending on the number of citations to its works whether it be an article or a book, either published or not. On the other hand, the citatons are considered only if coming from a published work. In other words, the reference is counted only if it was made in a book, an article on a journal or in an article of a conference book.

In case multiple researchers have the same number of publications then only one is picked.

Parameters:

1. $affiliation_id: the id of the affiliation of which we are interested in finding the best researcher

```
MATCH (documentCite)-[:Cites]->(documentCited)<-[:Writes]-(r:Researcher)-[:
    AffiliatesTo]->(a:Affiliation)
WHERE
        a.id = $affiliation_id AND
        (
        "Book" IN labels(documentCite) OR
        exists ((documentCite)<-[:Containss]-(:JournalNumber)) OR
        exists ((documentCite)<-[:Containss]-(:ConferenceBook))
        )
WITH r AS researcher, COUNT(DISTINCT(documentCite)) AS number_of_citations
ORDER BY number_of_citations DESC
RETURN researcher.name, researcher.ORCID , number_of_citations
LIMIT 1
```

Result by setting:

1. $affiliation_id: "3"

"researcher.name"	"researcher.ORCID"	"number_of_citations"
"Marta Gandolla"	"0104-1035-5434-6849"	15

Figure 2.13: Results of the query Q2

Performance

Performance resulting from an execution of the query in our dataset (around 3000 nodes):

- Started streaming 1 records after 123 ms and completed after 132 ms - 1150 total db hits in 132 ms

Q3.Top author for publisher and journal

This query returns the best author per number of publications for each publisher or journal along with the number of publications. More specifically, it returns a list with the following fields:

- the name of the publisher
- the name of the researcher (author)
- the number of publications

The query is the union of two different sub-queries. The first one is responsible for finding the best author for each publisher. While the second part, after the union, computes the best author per each journal.

In case multiple researchers published the same number of works then only is chosen.

In the query the relation [*1..2] (between publisher and the node (b)) is used to allow the count of both the books that are directly connected to the publisher and the articles contained in the conference books.

```
MATCH (p:Publisher)
WITH collect(p) AS publishers
UNWIND publishers AS publisher
CALL{
    WITH publisher
    MATCH (publisher)-[*1..2]->(b)<-[:Writes]-(r:Researcher)
    WITH publisher, r, count(DISTINCT(b)) as number_of_books
    ORDER BY publisher.id, number_of_books DESC
    RETURN publisher AS p, r, number_of_books AS number_of_pubblications
    LIMIT 1
}
RETURN p.name AS publisher_or_journal, r.name AS researcher,
   number_of_pubblications
UNION
MATCH (j:JournalNumber)
WITH collect(DISTINCT(j.title)) AS journals
UNWIND journals AS journal
CALL{
    WITH journal
    WITH journal AS jt
    MATCH (j:JournalNumber{title:jt})-[:Containss]->(a:Article)<-[:Writes
        ]-(r:Researcher)
    WITH jt, r, count(DISTINCT(a)) as number_of_articles
    ORDER BY jt, number_of_articles DESC
    RETURN jt, r, number_of_articles AS number_of_pubblications
    LIMIT 1
}
RETURN jt AS publisher_or_journal, r.name AS researcher,
```

number_of_pubblications

"publisher_or_journal"	"researcher"	"number_of_pubblications"
"LBL Technical Report"	"Alberto Quattrini Li"	1
"Excelsior"	"Russell Turpin"	2
"Aka Akademische Verlagsgesellschaft Aka GmbH, Berlin"	"Serena Monti"	1
"Verlag Dr. Hut"	"Stefano Brenna"	1
"Deutsch"	"Emanuele Panigati"	1
"Der Andere Verlag"	"Bruno Scaglioni"	1
"Niemeyer"	"Chiara Paganelli"	1
"VDI-Verlag"	"Damiano Belloli"	1
"Verlag der Augustinus-Buchhandlung"	"Nima Enayati"	1
"Utz"	"Davide Meroni"	2

Figure 2.14: Results of the query Q3

Performance

Performance resulting from an execution of the query in our dataset (around 3000 nodes):

- Started streaming 150 records after 1 ms and completed after 231 ms. - 44810 total db hits in 231 ms.

Q4.Topics addressed in a conference edition

The query returns the list of the topics touched during a conference. We consider as topics discussed in a conference those for which there exist a tag, and such tag is associated with at least one of the article presented in the very same conference. The query consists of a simple MATCH clause for finding the tags related with a conference, then the condition in the WHERE clause filters the valid matches so that we only consider the desired conference. Finally in the RETURN is produced a list containing all the topics that have been discussed without duplicates thanks to the presence of DISTINCT.

Parameters:

1. $conference_edition_id: the id of the conference edition of which we want to know the touched topics

```
MATCH (ce:ConferenceEdition)-[:Releases]->(cb:ConferenceBook)-[:Containss
    ]->(a:Article)<-[:Characterizes]-(t:Tag)
WHERE ce.id=$conference_edition_id
RETURN collect(DISTINCT(t.name)) AS addressed_topics
```

Result by setting:

1. $conference_edition_id: "20"

```
╒═══════════════════════════════════════════════════════════════════════════╕
|"addressed_topics"                                                         |
╞═══════════════════════════════════════════════════════════════════════════╡
|["Cloud Computing","Embedded and Real time systems","Cryptography","In|
|telligent System","Block chain Technology"]                                |
╘═══════════════════════════════════════════════════════════════════════════╛
```

Figure 2.15: Results of the query Q4

Performance

Performance resulting from an execution of the query in our dataset (around 3000 nodes):

- Started streaming 1 records after 57 ms and completed after 62 ms. - 337 total db hits in 62 ms

Q5. Insights on conferences

This query provides some insight about conferences. In particular, for each conference, it shows the number of editions, the total number of published books and the total number of articles related to that conference. The result is sorted by total number of articles and number of conference books. The query matches all the conference editions and the associated conference books and articles then for each conference (different editions of the same conference have the same name) computes the informations aforementioned. The limit is set to 10 but it can be changed to obtain more or less resulting tuples.

```
MATCH (ce:ConferenceEdition)-[:Releases]->(cb:ConferenceBook)-[:Containss
    ]->(a:Article)
WITH ce.name AS conference_name, count(DISTINCT(ce)) AS number_of_editions,
     count(DISTINCT(cb)) AS total_number_of_books, count(a) AS
    total_number_of_articles
ORDER BY total_number_of_articles DESC
RETURN conference_name, number_of_editions, total_number_of_books,
    total_number_of_articles
LIMIT 10
```

"conference_name"	"number_of_editions"	"total_number_of_books"	"total_number_of_articles"
"IEEE International Workshop on Visualizing Software for Understanding and Analysis VISSOFT"	12	17	50
"Surgery Simulation and Soft Tissue Modeling"	9	14	41
"Theoretical Computer Science"	10	14	40
"Cooperative Information Systems International Conference"	7	12	34
"International Conference on Wearable Micro and Nano Technologies for Personalized Health"	9	12	34

Figure 2.16: Results of the query Q5

Performance

Performance resulting from an execution of the query in our dataset (around 3000 nodes):

- Started streaming 10 records after 84 ms and completed after 116 ms - 5284 total db hits in 116 ms.

Q6. Find the most indirectly cited author for every researcher

For every researcher 'r' we print the name of the most indirected cited author by 'r' and the times he is cited. A directed cite is a cite between the author of a document 'd' and the author of the document cited by 'd'. An indirected cite is a between the author of a document 'd' and the author of the document cited by a document cited by 'd', in the graph it means a chain of relations "Cites". This chain must be composed by at least two and at most three relations "Cites". In the example (Figure 2.17) Ettore Speziale is the author directed cited by Marco Bessi, while Daniele Macera is the author indirected cited by Marco Bessi.

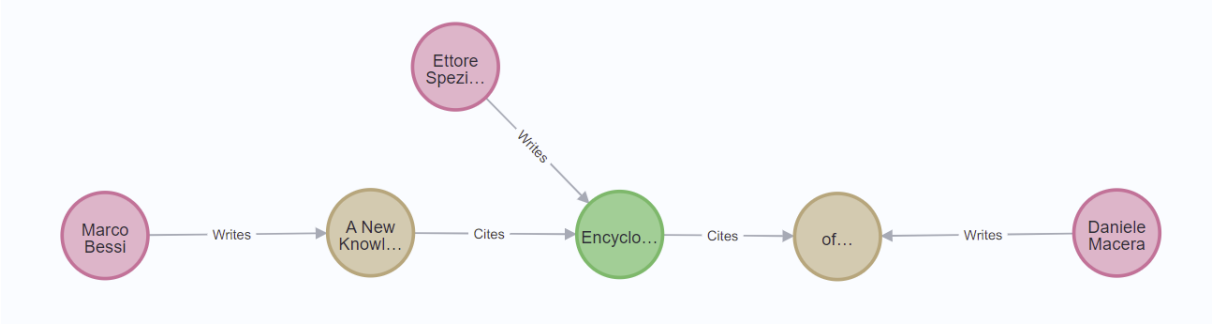


Figure 2.17: Example of indirected citation

```
MATCH (r:Researcher)
WITH collect(DISTINCT r) AS allResearcher
UNWIND allResearcher AS singleResearcher
CALL {
    WITH singleResearcher
    MATCH (singleResearcher)-[:Writes]->(:Document)-[:Cites*2..3]->(:
        Document)<-[:Writes]-(r1:Researcher)
    WITH singleResearcher.name AS researcherName, r1.name AS
        researcherInderectedCited, count(*) AS numberOfIndirectedCited
    WHERE numberOfIndirectedCited>1
    RETURN researcherName, researcherInderectedCited,
        numberOfIndirectedCited
    ORDER BY numberOfIndirectedCited DESC
    LIMIT 1
}
RETURN researcherName, researcherInderectedCited, numberOfIndirectedCited
ORDER BY numberOfIndirectedCited DESC
```

"researcherName"	"researcherInderectedCited"	"numberOfIndirectedCited"
"Giovanni Quattrocchi"	"Giorgia Ramponi"	27
"Marco Bergamasco"	"Simone Formentin"	19
"Basak Sakcak"	"Francesca Lunardini"	19
"Davide Meroni"	"Giorgia Ramponi"	18
"Alice Geminiani"	"Giorgia Ramponi"	18
"Sebastian Troia"	"Giorgia Ramponi"	18

Figure 2.18: Results of the query Q6

Performance

Performance resulting from an execution of the query in our dataset (around 3000 nodes):

- Started streaming 337 records after 61 ms and completed after 390 ms - 368252 total db hits in 390 ms This query is known to be heavy because it returns something for every researcher.

Q7. Relative distance between two researchers

A scientific bibliography can be seen as a network of researchers. In this network, the connections between researchers are given by the web of citations among their scientific papers and their shared works. Therefore it is possible to compute the minimum distance between two researchers and estimate whether they work in the same field/context or not. This can be used to identify "communities" within the researchers, just like in a social network.

In the following query the shortest path function is used to determine the minimum relative distance between two specified researchers. Moreover, the path between them must include only relationships of type "Writes" or "Cites". This condition was specified so as to avoid meaningless connections for our objective. For instance, it would be wrong to consider two researchers strongly related, only because they published a book through the same publisher.

We consider two researchers:

- Strongly related when the minimum number of relations connecting them is less or equal 5 and greater than 2.
- Weakly related when the minimum number of relations connecting them is greater than 5
- Worked together when the length of the minimum path is equal to 2

Parameters:

1. $researcher1_id: the id of the first researcher
2. $researcher2_id: the id of the second researcher

```
MATCH (r1:Researcher{id:$researcher1_id}), (r2:Researcher{id:
    $researcher2_id}),
        p = shortestPath((r1)-[*]-(r2)) WHERE ALL(r IN relationships(p)
WHERE type(r) IN ["Writes","Cites"])
WITH r1.name AS researcher_1, r2.name AS researcher_2, length(p) AS
    connection_length
RETURN researcher_1, researcher_2,
CASE WHEN connection_length=2 THEN "Worked together"
    WHEN connection_length>2 AND connection_length<=5 THEN "Strongly
        related"
    WHEN connection_length>5 THEN "Weakly related"
END AS connection_character
```

Results by setting:

1. $researcher1_id: "375"
2. $researcher2_id: "98"

"researcher_1"	"researcher_2"	"connection_character"
"Alberto Quattrini Li"	"Alessio Mauro Franchi"	"Strongly related"

Figure 2.19: Results of the query Q7

Performance

Performance resulting from an execution of the query in our dataset (around 3000 nodes):

- Started streaming 1 records after 73 ms and completed after 171 ms. - 3007 total db hits in 171 ms

Q8. Topic-trend in journals in a specific period

The query returns the topic mainly treated in the newspapers (in a specified period), the number of articles involved and the list of journals that have published the articles.

Parameters:

1. $lowerBoundYearInput: Starting year for the research.
2. $upperBoundYearInput: Ending year for the research.

```
MATCH (j:JournalNumber)-[:Containss]-(a:Article)-[:Characterizes]-(t:Tag)
WHERE j.publication_date>=date({year:$lowerBoundYearInput}) AND j.
    publication_date<=date({year:$upperBoundYearInput})
RETURN t.name as topic, count(a) as amount_of_article,collect(DISTINCT j.
    title) as list_of_journals
ORDER BY  count(a) DESC
LIMIT 1
```

Result by setting:

1. $lowerBoundYearInput: 1998
2. $upperBoundYearInput: 1999

topic	amount_of_article	list_of_journals
"Network"	5	["IEEE Embed. Syst. Lett.", "Syst. Control. Lett.", "Sci. Comput. Program.", "EAI Endorsed Trans. Ubiquitous Environ."]

Figure 2.20: Results of the query Q8

Performance

Performance resulting from an execution of the query in our dataset (around 3000 nodes):

- Started streaming 1 records after 32 ms and completed after 123 ms. - 3007 total db hits in 171 ms

Q9. Get the series with the most conference book from a certain country

Given a group of country, return the series that contains the most number of conference book from one of the country of the group. A conference book is from country 'c' when it has been released by a conference edition that takes place in 'c'.

```
MATCH(s:Serie)-[:Collects]->(confB:ConferenceBook)<-[r:Releases]-(confE:
    ConferenceEdition)
WHERE confE.country IN $countryGroup //match all the series that have a
    conference book of a conference taken in one of the country of
    $countryGroup
WITH s.id AS serieId,s.name AS serieName ,count(DISTINCT confB) AS
    numberOfConferenceBookOfCountry
ORDER BY numberOfConferenceBookOfCountry DESC
LIMIT 1
RETURN serieId,serieName,numberOfConferenceBookOfCountry
```

Result by setting:

1. $countryGroup: ["China","Indonesia","Philippines","Malaysia"]

"serieId"	"serieName"	"numberOfConferenceBookOfCountry"
"10"	"HNI-Verlagsschriftenreihe"	4

Figure 2.21: Results of the query Q9

Performance

Performance resulting from an execution of the query in our dataset (around 3000 nodes):

- Started streaming 1 records after 2 ms and completed after 37 ms.

- 492 total db hits in 127 ms

Q10. Researchers that only edits

This query returns all the researchers that have edited the conference books relative to at least one conference edition, but didn't write any book of their own.

```
MATCH (r:Researcher)-[:Edits]->(ce:ConferenceEdition)
WHERE NOT exists((r)-[:Writes]->(:Document))
RETURN DISTINCT r.name
```

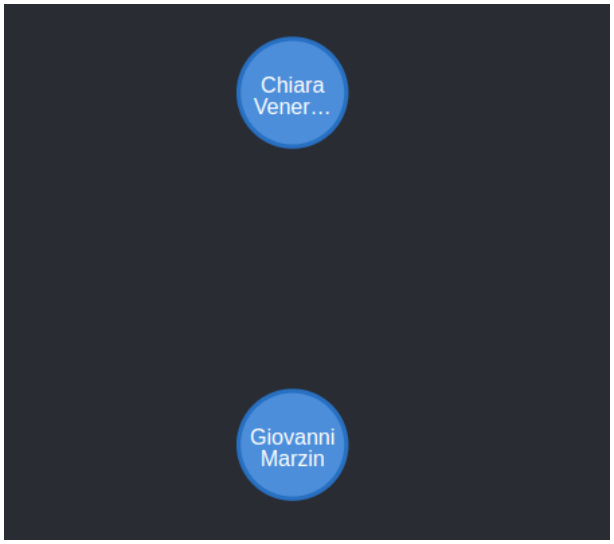


Figure 2.22: Results of the query Q10

Performance

Performance resulting from an execution of the query in our dataset (around 3000 nodes):

- Started streaming 2 records after 49 ms and completed after 64 ms.

- 2519 total db hits in 64 ms.

New "Journal" entity added to improve query Q3 and Q8

So far, every Journal Number published by a given Journal share the same ISSN and title. In this way, every query filtering for a specific journal would have to go through all the journal numbers titles or ISSN codes. In order to avoid it, we can add **Journal Nodes** in our database. Such nodes contain the title and ISSN that are common to a group of Journal Numbers to which the Journal node is connected with a *Publish* relationship.

This first command creates a node with label :Journal for each distinct ISSN found in all :JournalNumber nodes.

```
match (jn:JournalNumber) merge (j:Journal{ISSN:jn.ISSN}) on create set j.
    title = jn.title
```

Then with the following statement we maintain the relation between a given journal and the journal numbers it publishes

```
match (j:Journal), (jn:JournalNumber) where j.ISSN=jn.ISSN  merge (j)-[:
    Publishes]->(jn)
```

Finally the redundant data from the journalNumber nodes are deleted

```
match (jn:JournalNumber) remove jn.title, jn.ISSN
```

With the new entity, the query Q3 would be:

```
MATCH (j:Journal)-[:Publishes]->(jn:JournalNumber)-[:Containss]->(a:Article
    )<-[:Characterizes]-(t:Tag)
WHERE j.publication_date>=date({year:$lowerBoundYearInput}) AND j.
    publication_date<=date({year:$upperBoundYearInput})
RETURN t.name as topic, count(a) as amount_of_article,collect(DISTINCT j.
    title) as list_of_journals
ORDER BY  count(a) DESC
LIMIT 1
```

The new query Q8 would be:

```
MATCH (p:Publisher)
WITH collect(p) AS publishers
UNWIND publishers AS publisher
CALL{
    WITH publisher
    MATCH (publisher)-[*1..2]->(b)<-[:Writes]-(r:Researcher)
    WITH publisher, r, count(DISTINCT(b)) as number_of_books
    ORDER BY publisher.id, number_of_books DESC
    RETURN publisher AS p, r, number_of_books AS number_of_pubblications
    LIMIT 1
```

```
}
RETURN p.name AS publisher_or_journal, r.name AS researcher,
    number_of_pubblications
UNION
MATCH (journal:Journal)
WITH collect (journal) AS journals
UNWIND journals AS journal
CALL{
    WITH journal
    MATCH (journal)-[:Publishes]->(jn:JournalNumber)-[:Containss]->(a:
        Article)<-[:Writes]-(r:Researcher)
    WITH journal as j, r, count(a) as number_of_articles
    ORDER BY j, number_of_articles DESC
    RETURN j, r, number_of_articles AS number_of_pubblications
    LIMIT 1
}
RETURN j.title AS publisher_or_journal, r.name AS researcher,
    number_of_pubblications
```

Performance

Performance resulting from an execution of the query in our dataset (around 3000 nodes):

- Started streaming 150 records after 4 ms and completed after 223 ms.

- 40078 total db hits in 223 ms.

By executing the previous queries, we haven't noticed any significant performance improvement. By the way, this could be due to the "small" dimension of our dataset. Instead, these changes could be useful if it were necessary to change attributes of the Journals or if the number of Journal Numbers contained in the dataset would increase.

2.2.4. Commands

In the following you are presented with some of the possible commands that can be executed on the database. For each statement is provided a short description, the code and an example of the result produced upon its execution.

Commands can be either creation, update, delete statements or a combination of those three.

C1: Publish a unpublished article

This update command changes the label of the interested article from :Article to :Book(published article) and binds it to a publisher, moreover it adds a specified ISBN, publication year and DOI. The publication takes place only if the article is not contained in any other published work such as a journal number or a conference book. Besides, both specified article and publisher must already exist in the database.

Parameters:

1. $article_id: the id of the article to be published
2. $publisher_id: the id of the publisher which publishes the book
3. $new_book_ISBN: the ISBN of the published book
4. $publication_year: the year in which the book is published
5. $book_DOI: the DOI associated with the created book

```
MATCH (a:Article), (p:Publisher)
WHERE   a.id = $article_id AND
        p.id = $publisher_id AND
        NOT exists((:JournalNumber)-[:Containss]->(a)) AND
        NOT exists((:ConferenceBook)-[:Containss]->(a))
CALL{
        MATCH (b:Book)
        WITH max(toInteger(b.id)) AS max_id
        RETURN toString(max_id+1) AS new_book_id
}
REMOVE a:Article
SET a:Book, a.isbn=$new_book_ISBN, a.year=$publication_year, a.id=
    new_book_id, a.DOI=$book_DOI
```

```
MERGE (p)-[:Publishes]->(a)
```

Results by setting:

1. $article_id: "420"
2. $publisher_id: "2"
3. $new_book_ISBN: "7777-33242-2323-222"
4. $publication_year: "2022"
5. $book_DOI: "https://doi.org/0"

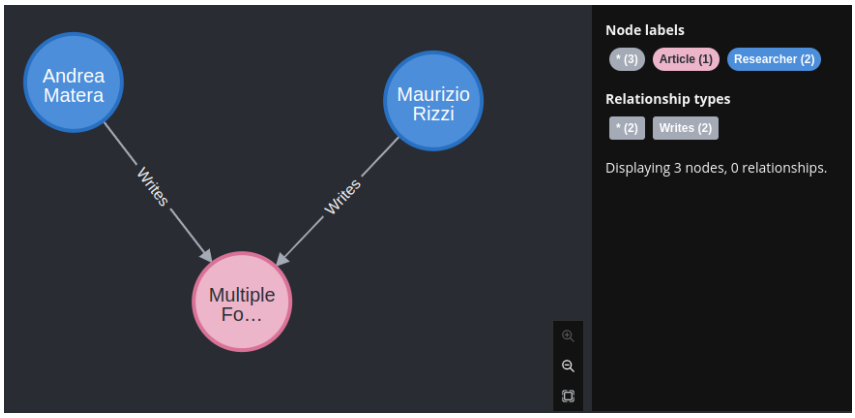


Figure 2.23: Before executing C1

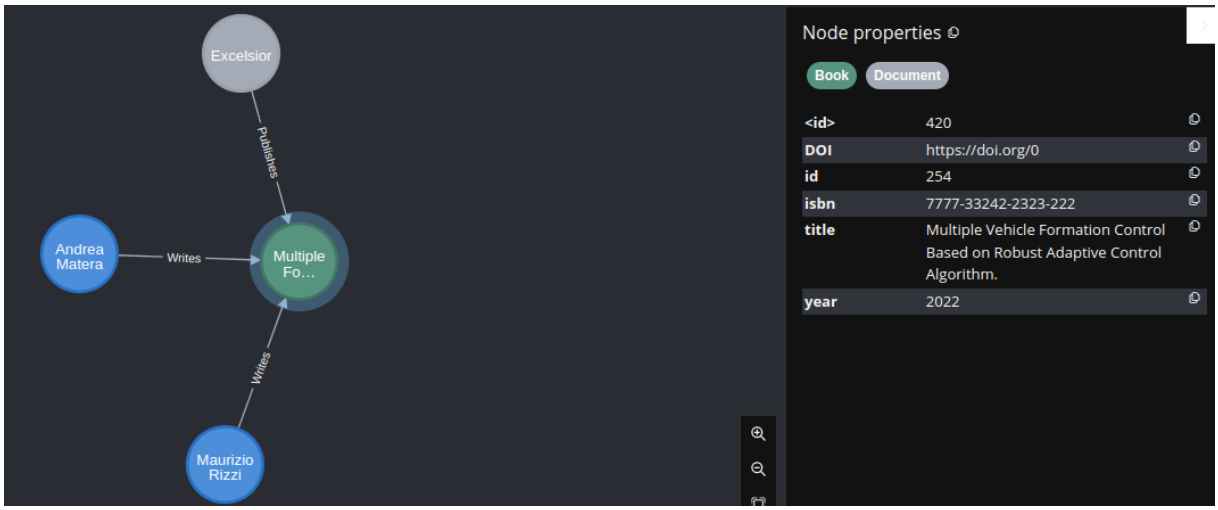


Figure 2.24: After executing C1

C2: Creation of a new Tag

This command is used to create a new Tag entity with *Name* set to "Network". Using the same dataset generation criterion, the "Network" tag is linked to Documents (i.e. Articles and Books) that contain "Network" in their title but not "Neural Network".

```
CREATE (t:Tag{name:"Network"})
WITH t AS new_tag
MATCH (d:Document)
WHERE d.title CONTAINS 'Network' AND NOT (d.title CONTAINS 'Neural Network
    ')
CREATE (new_tag)-[c:Characterizes]->(d)
RETURN new_tag,c,d
```

The query returns the new Tag node, the linked documents and the relation between them. For example, the Document entitled "Query Evaluation in Peer-to-Peer Networks of Taxonomy-Based Sources" is now linked with the "Network" tag.

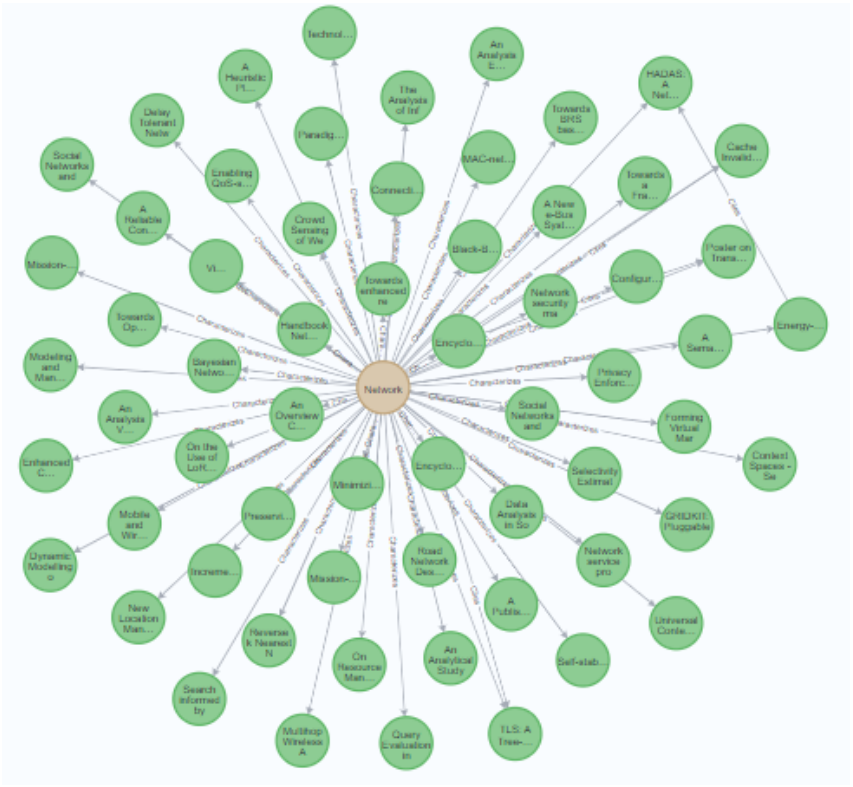


Figure 2.25: Results of command C2

C3: Creation of a new Series

This command is used to create a new Series. As stated before, a Series can be a collection of different publications on any topic. In this case, a new Series called "Network related works" is created (with a generated ISSN) and linked to the documents that were previously linked with the "Network" tag.

```
CREATE (s:Serie{name:"Network Related Works", ISSN:"1234-5678"})
WITH s AS new_series
MATCH (d:Document)<-[:Characterizes]-(t:Tag{name:"Network"})
CREATE (new_series)-[c:Collects]->(d)
RETURN new_series, c, d
```

The query returns the new Series node, the linked documents and the relation between them.

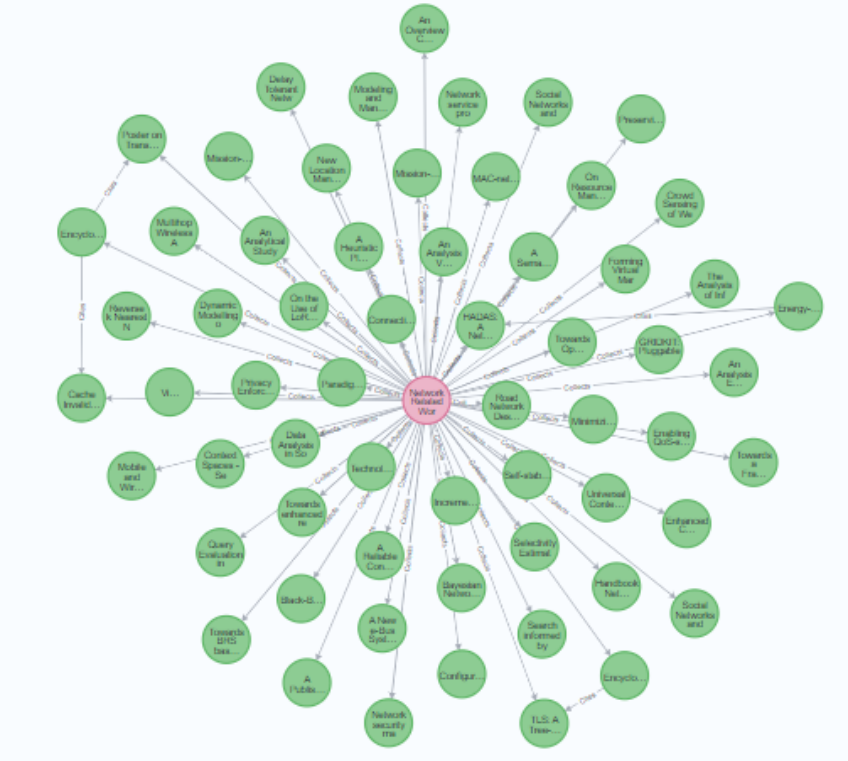


Figure 2.26: Results of command C3

C4: Updating a Conference Edition location

This command is used to update a conference edition location. Having in input the conference name, starting date and ending date a specific conference edition can be accessed and so the address can be modified.

```
MATCH (ce:ConferenceEdition)
WHERE ce.name = $ce_name AND ce.start_date = date($ce_startdate) AND ce.
    end_date = date($ce_enddate)
SET ce.country = $ce_new_country, ce.city = $ce_new_city, ce.street =
    $ce_new_street
RETURN ce
```

Parameters:

1. $ce_name: conference edition name.
2. $ce_startdate: starting date of the conference.
3. $ce_enddate: ending date of the conference.
4. $ce_new_country: new country.
5. $ce_new_city: new city.
6. $ce_new_street: new street.

The query returns the modified conference edition.

Results by setting:

1. $ce_name: "Theoretical Computer Science"
2. $ce_startdate: "2018-10-23"
3. $ce_enddate: "2018-10-26"
4. $ce_new_country: "Italy"
5. $ce_new_city: "Bologna"
6. $ce_new_street: "Monte Rosa"

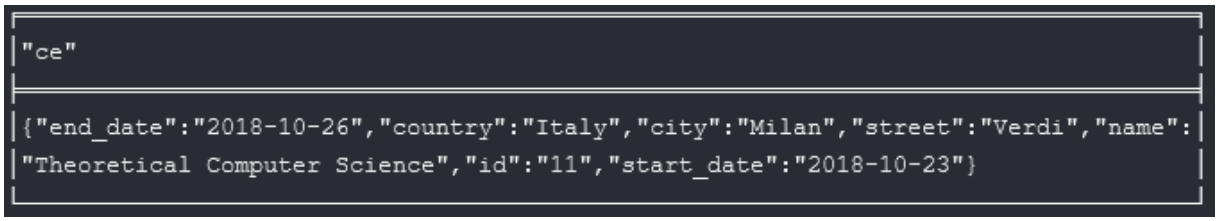


Figure 2.27: Before executing C4

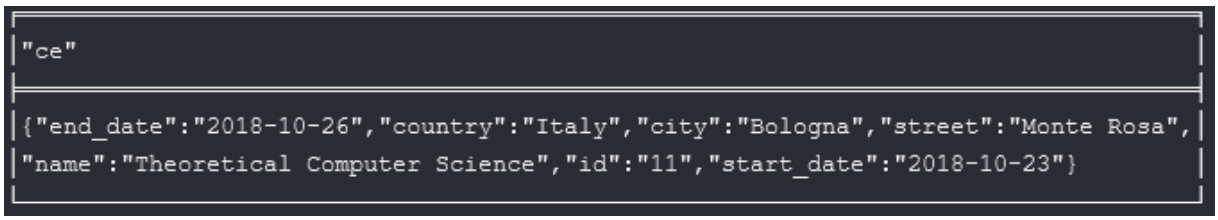


Figure 2.28: After executing C4

C5: Adding a new Book

The following commands are used to create a new book and to link it with a given author, a given publisher and the appropriate tags. Let's start from the book creation.

```
CREATE (b:Book{Title:"$b_title", DOI:"$b_doi", ISBN:"b_isbn", year:"b_year
    "})
```

Then, we create the links.

```
MATCH (b:Book{ISBN:"$b_isbn})
CREATE (p:Publisher{name:"$p_name"})-[:Publishes]->(b), (r:Researcher{name
    :"$r_name"})-[:Writes]->(b), (t:Tag{name:"$t_name"})-[:Characterize]->(b
    )
```

Results by setting:

1. $b_title: "New Network Book"
2. $b_doi: "https://doi.org/0"
3. $b_isbn: "978-0-1734-4056-8"
4. $b_year: 2022
5. $p_name: "IEEE Computer Society"
6. $r_name: "Davide Todeschini"
7. $t_name: "Network"

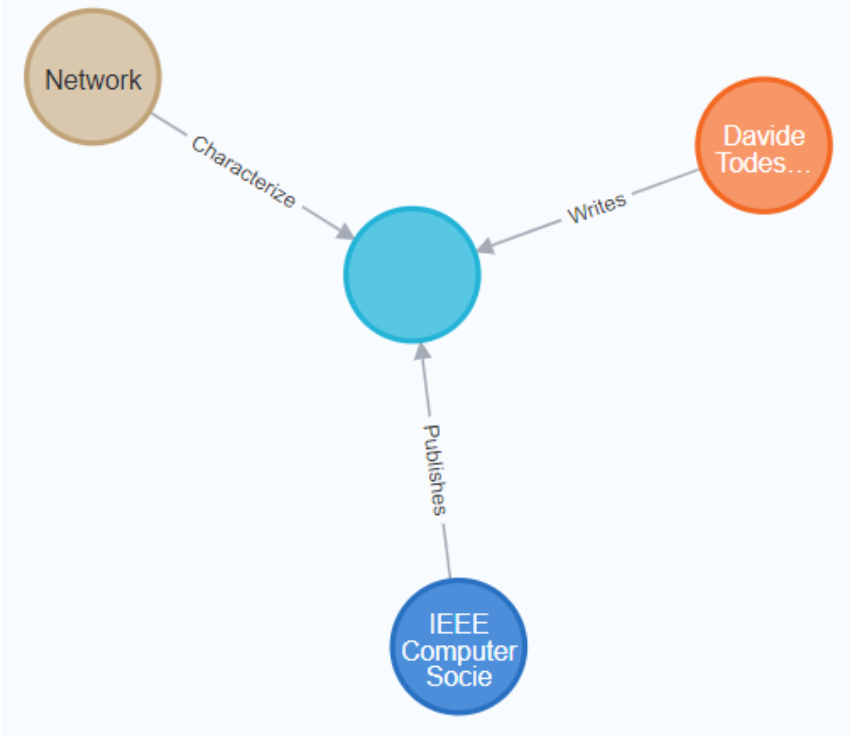


Figure 2.29: The new node with its relationships.

2.3. MongoDB implementation

This section will cover the implementation of a collection in the MongoDB document database related to the reality presented in chapter 1 "*Introduction*".

2.3.1. Amendments to the ER

The scope of the MongoDB implementation is a bit different than the one of Neo4J. In fact, the main objective is to organize the structure of a collection (within a document database) intended as the **set of articles published within journals only**, maintaining, for each individual article, the information about the journal, its bibliography, the authors and their affiliations, and other information that will be explained below.

Since we are only considering scientific papers published in journals, the ER diagram (Figure 2.1) presented in the ER diagram section can be greatly reduced.

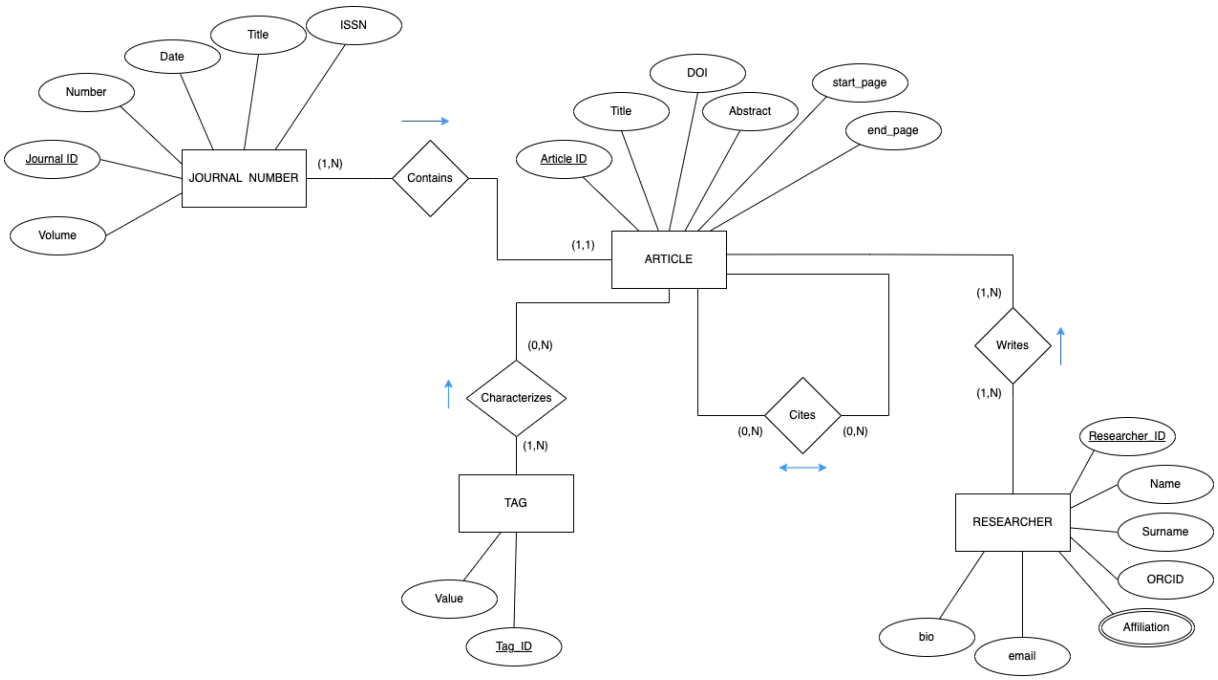


Figure 2.30: The "reduced" ER model

As shown in the Figure 2.70 above, the context is restricted to:

- Considering only articles as typology of written work, excluding books and unpublished articles.
- Considering journals as the only form of publication, excluding conferences. This means that all articles in the collection are published in journals and the publisher of the article is the journal itself (the "Publisher" entity corresponds to the journal one). We still consider the single instance of a journal, that is what we have called so far a "Journal Number".
- Considering researchers as only authors (the figure of the "editor" is not considered).
- Not considering any kind of collection except journals.

For the entities that are in the reduced model, the same attributes of the original ER diagram are kept, with the additions of others:

- *Volume* on **Journal Number**: in the previous formalization of the ER, the attribute was not considered because we had seen that a volume of journals basically corresponds to the journal numbers (of the same journal) published in the same year. Thus, it could be derived with a query. Anyway, in this case we decided to make it explicit.
- *Start Page* and *End Page* on **Article**: this attributes were added to highlight the position of an article in the Journal Number.
- *Email* on **Researcher**: that contains the email of the author.
- *Bio* on **Researcher**: that contains a biography of the author.

It is important to note that:

- The participation of the **Article** entity to the *Contains* relationship is mandatory, since we are considering only published articles.
- The participation of the **Tag** entity to the *Characterizes* relationship is mandatory, since only tags related to an article are considered.
- The participation of the **Researcher** entity to the *Writes* relationship is mandatory, since we are considering only researchers that are actually "authors".
- The reduced ER does not represent the structure of a document, that is presented in the following section.

2.3.2. Document structure

The structure of the document of collection in the document database and the related documentation are shown below.

```
{
  "_id": <ObjectID>
  "title": <str>,
  "abstract": <str>,
  "researchers":[
    {
        "name": <str>,
        "surname": <str>,
        "ORCID": <str>,
        "email": <str>,
        "bio": <str>,
        "affiliations":[<str>]
    }
  ],

  "metadata":{
    "doi": <str>,
    "tags":[<str>],
  },

  "journal":{
    "title": <str>,
    "volume": <int>,
    "number": <int>,
    "ISSN": <str>,
    "date": <ISODate>,
    "start_page": <int>
    "end_page": <int>
  },

  "sections":[
    {
      "title": <str>,
      "paragraphs": [<str>],
      "subsections":[
        {
          "title": <str>,
          "paragraphs": [<str>],
        }
      ],
      "images":[
        {
          "url": <str>
          "caption": <str>
        }
      ]
    }
  ],

  "bibliography":[
    {
      "title": <str>,
      "researchers":[
        {
          "name": <str>,
          "surname": <str>,
          "ORCID": <str>,
          "affiliations":[<str>]
        }
      ]
      "journal_title": <str>,
      "year": <int>,
      "doi": <str>
    }
  ]
}
```

Figure 2.31: The structure of our document

- **_id**: unique identifier for each document, assigned by default by MongoDB.
- **title**: string that represents the title of the article.
- **researchers**: vector of sub-documents, each of which is a researcher. The vector represents the set of authors of the article. Each researcher is characterized by a name, a surname, an ORCID code (unique identifier for each researcher), an email, a biography, and a vector of strings containing all the researcher's affiliations.
- **metadata** is a sub-document composed by:
  - a DOI, that uniquely identifies a published article.
  - a vector of tags associated to the article.
- **journal** is a sub-document in which are reported all the main information related to the publication of the article. It is composed by the following fields:
  - **title**, that represents the name of the journal.
  - **volume**, that is the number of the volume that contains the journal number in which the article is contained.
  - **number**, that is the journal number in which the article is published.
  - **ISSN**, a string that uniquely identifies a journal.
  - **date**, that represents the date of publication of the article in the journal.
  - **start_page**, that represents at which page of the journal number the article starts.
  - **end_page**, that represents at which page of the journal number the article ends.
- **sections** is a vector of sub-document where each of these represents one section of the article. It is composed by:
  - **title**, that represents the title of the section.
  - **paragraphs**, a vector of strings containing all the paragraphs of the section.
  - **subsections**, a vector of sub-documents where each of these is a subsection of the section. It is composed by:
    - **title**, that represents the title of the subsection .

        * **paragraphs**, a vector of string where each string is a paragraph of the subsection.

    - **images**, that is a vector of sub-documents where each of these represents an image. It is composed by:

        * **url**, that identifies the location of the image on the web.

        * **caption**, a string containing a brief description of the image.

- **bibliography** is a vector of sub-documents where each of these represents the preview of a cited article. Every preview contains the main information related to the cited article, its authors and its publication. In particular, all the fields within the preview are the same mentioned above, except for the *year* field which intends to inform only about the year of publication (unlike 'date' which also takes into account the day and month of publication).

2.3.3. Dataset

Creation

The dataset was built following the generic structure that we have provided in the previous section. The starting point was an already existing dataset of articles released by *Elsevier*, a Polish academic publishing company specialized in scientific, technical, and medical content. The dataset can be found at the following link.

Since the original dataset was a collection of articles published by the same publisher and that the document structure did not match the one that we created, the documents were reshaped using a Python program. In particular, some fields were modified or created from the ground up with the following policies:

- If the abstract of some article was missing, it was replaced with a mock text (*"Lorem ipsum dolor sit amet..."*).
- Since the document structure of the original dataset did not consider an *ORCID* field for the authors, it was not possible to guarantee a unique code for each author. Thus, we used a fixed code for the *ORCID* field.
- The *bio* of each author is a mock text (*"Lorem ipsum dolor sit amet..."*).
- The list of possible *affiliations* of the authors was taken from the dataset used for the Neo4j delivery. Each author was assigned to a random number (between 1 and 3) of affiliations.

- The Journals were taken from the dataset used for the Neo4J delivery. Each article was assigned randomly to a Journal Number. The day of the publication of each Journal Number was fixed to the first of the month.
- The *start_page* and the *end_page* fields were randomly generated.
- Each paragraph has a random number of images (between 0 and 3). The *url* field is fixed, while the *caption* field is composed of "Image of" plus a random keyword of the article.
- The *bibliographies* were randomly generated. None of the articles can cite an article that is not available into our dataset. Since the generation of this field follows a first phase of dataset generation, all the information about the cited articles was taken from already converted documents.
- In general, if some field was missing, it was not inserted in the JSON file.

The dataset that we obtained consists of 1000 documents.

Import

The output of the Python program was a set of JSON files, one for each article. The next step is to import all documents into our collection (called articles) and export a single JSON file, in order to simplify future population processes for other users.

Below it is reported the script that was used in a MAC terminal to load all the documents within the collection at the same time. This script must be launched from the folder containing all the JSON files.

```
ls -1 *.json | while read jsonfile; do mongoimport --db SecondDelivery --
    collection articles --file $jsonfile --type json; done
```

There are two available options in order to import the dataset from the file obtained in the previous phase (using **mongoimport**):

- connecting to a local server (localhost).
- connecting to an online database deployment.

Localhost

In the command prompt write the following order:

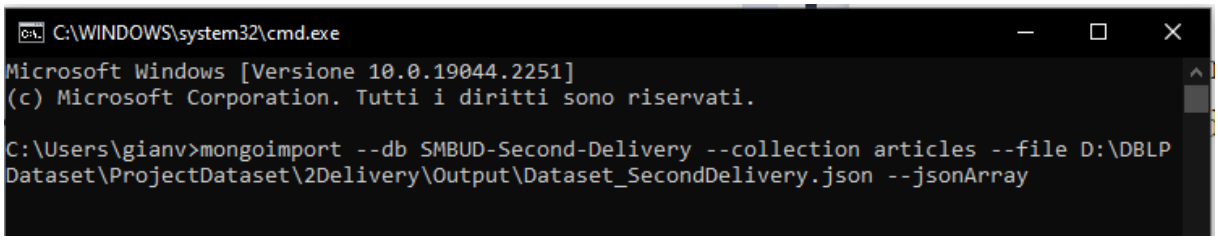


Figure 2.32: Localhost importing string

Mongoimport requires different fields to be specified:

- `db`: name of the database already created on MongoDB.
- `collection`: name of collection.
- `file`: JSON file dump path.
- `jsonArray`: to specify the type of dump file.

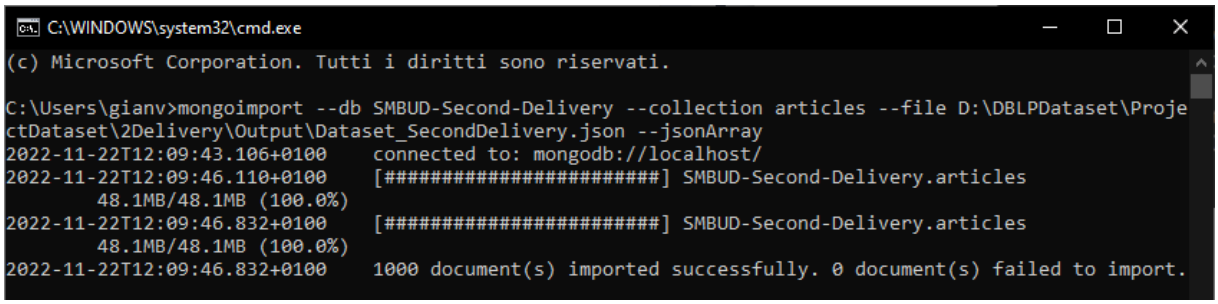


Figure 2.33: Localhost importing result

Connect to an online database deployment

In order to import the dataset in an online database deployment, we first have to access to MongoDB and click "Command Line Tools"

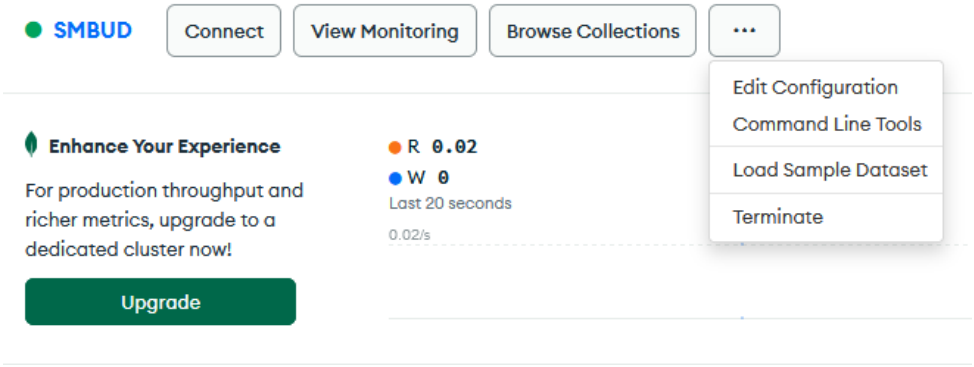


Figure 2.34: Command Line Tools

At this point, we reach the "Data Import and Export Tools" section: we can copy the connection string.

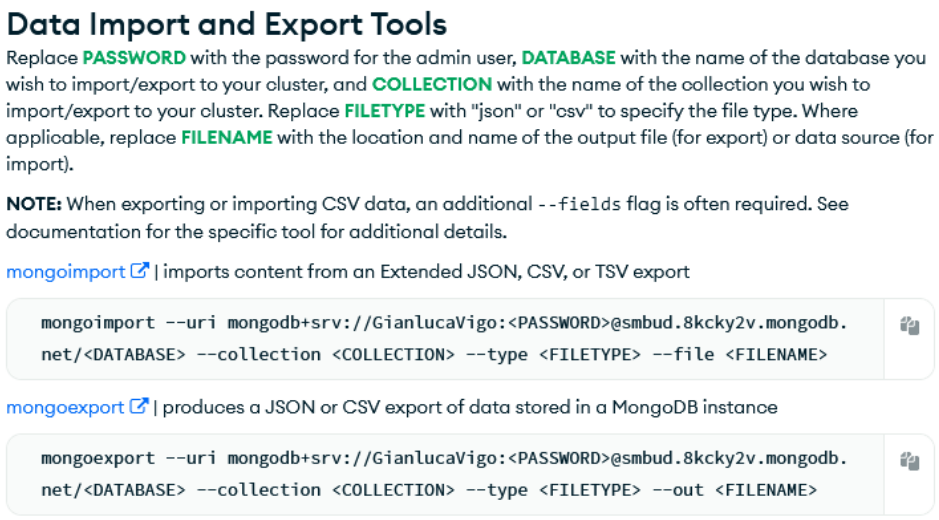


Figure 2.35: Data Import and Export Tools

Now we have to substitute all the words within < > and insert the right information.

```
C:\Users\gianv>mongoimport --uri mongodb+srv://GianlucaVigo:<PASSWORD>@smbud.8kcky2v.mongodb.net/SMBUD_
Second_Delivery --collection articles --type JSON --file D:\DBLPDataset\ProjectDataset\2Delivery\Output
\Dataset_SecondDelivery.json --jsonArray
```

Figure 2.36: Online importing string

If the process is successful, we obtain the following answer:

```
SMBUD_Second_Delivery --collection articles --type JSON --file D:\DBLPDataset\ProjectDataset\2Delivery\
Output\Dataset_SecondDelivery.json --jsonArray
2022-11-22T12:22:05.694+0100    connected to: mongodb+srv://[**REDACTED**]@smbud.8kcky2v.mongodb.net/SM
BUD_Second_Delivery
2022-11-22T12:22:08.707+0100    [########################] SMBUD_Second_Delivery.articles       48.1MB/
48.1MB (100.0%)
2022-11-22T12:22:11.263+0100    [########################] SMBUD_Second_Delivery.articles       48.1MB/
48.1MB (100.0%)
2022-11-22T12:22:11.264+0100    1000 document(s) imported successfully. 0 document(s) failed to import.
```

Figure 2.37: Online importing result

2.3.4. Queries

In this chapter are reported and commented some of the possible queries that can be executed on the database. For each statement is provided a short description, the code and an example of the result produced upon its execution.

A particular note on the queries that perform a **textual search**: since a textual index is needed for the fields on which to search and MongoDB supports only one index at a time, indexes are created and then deleted after the execution of the query.

Q1. Find the most eminent journals over a specific topic

We define that a journal is eminent in a certain field if it is cited a lot in other articles that are about that field. The query goal is to find the most relevant journals over a specific topic. The topics of an article are given by the associated keywords.

```
db.articles.aggregate([
    {"$project": {"bibliography.journal_title": 1, "metadata.keywords":
       1}},
    {"$unwind": "$bibliography"},
    {"$unwind": "$metadata.keywords"},
    {"$group": {
      "_id": {"JournalName": "$bibliography.journal_title", "Tag": "
         $metadata.keywords"},
      "tag-journal_relevance": {"$sum": 1}
    }},
    {"$sort": {"_id.Tag": 1, "tag-journal_relevance": -1}},
     {"$group": {
      "_id": {"Tag": "$_id.Tag"},
      "journal": {"$first": "$_id.JournalName"},
      "relevanceLevel": {"$first": "$tag-journal_relevance"}
    }},
    {"$sort":{"relevanceLevel":-1}},
    {"$limit": 10}
  ])
```

With the project clause we filter out the useless information: only the keywords and the cited articles of an article are kept.

Then, the first two unwind operations let us have different documents for each bibliography and keyword linked to the article (the documents obtained after the first unwind are copied and enriched with one keyword of the list).

At this point, it is possible to group these documents considering the tag and the journal name values. In this way we can compute the relevance of a journal with respect of a tag (by summing 1 for each document). In order to highlight our result, we sort these documents first by the tag value and then by the relevance between a tag and a journal in a decreasing order.

The second group operation is the one that let us keep only the documents that include the result data. In fact, we will **hold the journal with higher relevance in a specific topic**. Then, the results are ordered by the relevance level (in decreasing order) and limited to 10 rows.

```
{ _id: { Tag: 'Iceland' },
  journal: 'SIGMOD Record',
  relevanceLevel: 14 }
{ _id: { Tag: 'B. SEM' },
  journal: 'IEEE Embed. Syst. Lett.',
  relevanceLevel: 12 }
{ _id: { Tag: 'Microplastics' },
  journal: 'SIGMOD Record',
  relevanceLevel: 11 }
{ _id: { Tag: 'B. TEM' },
  journal: 'IEEE Embed. Syst. Lett.',
  relevanceLevel: 8 }
{ _id: { Tag: 'Corticosterone' },
  journal: 'SIGMOD Record',
  relevanceLevel: 7 }
{ _id: { Tag: 'oxygen fugacity' },
  journal: 'J. Assoc. Inf. Syst.',
  relevanceLevel: 7 }
{ _id: { Tag: 'Moon' },
  journal: 'SIGMOD Record',
  relevanceLevel: 7 }
{ _id: { Tag: 'PICS bags' },
  journal: 'SIGMOD Record',
  relevanceLevel: 6 }
{ _id: { Tag: 'C. Pitting corrosion' },
  journal: 'Electron. Notes Theor. Comput. Sci.',
  relevanceLevel: 6 }
{ _id: { Tag: 'ALSPAC' },
  journal: 'J. Assoc. Inf. Syst.',
  relevanceLevel: 6 }
```

Figure 2.38: Result of the query Q1

PERFORMANCE: using *.explain("executionStats")* we obtain that the performance of the query is executionTimeMillis: 170

Q2. Most researched field in a given university in a specified period

This query allows to find the most researched topic in a university during a specified period of time. This allows to obtain insight on the evolution and on the background of an affiliation.

```
db.articles.aggregate([
  {"$match":{"$and":[{"researchers.affiliations":{"$regex":/Hong Kong
      Polytechnic/i}},
                     {"journal.date" : { "$gte" : ISODate('2000-01-01')}},
                     {"journal.date" : { "$lte" : ISODate('2020-01-01')}}]}
  },
  { "$unwind" : "$metadata.keywords" },
  {"$group" : {
    "_id" : "$metadata.keywords",
    "number_of_articles" : { "$sum" : 1 }
  }},
  { "$sort" : { "number_of_articles" : -1 }},
  { "$limit" : 1 }
]);
```

First of all the documents are filtered so as to keep only the ones containing at least one researcher that was working for the interested affiliation when the article was written. Additional conditions to the match clause enables the filtering of the documents to discard those outside the time interval in which we are interested. After the filtering, the remaining documents are unwound to allow the counting of the occurrence of each keywords across all the articles. Finally such numbers are sorted and only the most used keyword is kept.

```
{ _id: 'Inflammation', number_of_articles: 2 }
```

Figure 2.39: Results of the query Q2

PERFORMANCE: using *.explain("executionStats")* we obtain that the performance of the query is executionTimeMillis: 6

Q3. Find the most used tags by the most active journals

The query goal is to find the most used tags across all the most productive and active journals.

```
db.articles.aggregate([
  //FIRST PART: most active journals
    {"$unwind": "$metadata.keywords"},
    {"$group": {
      "_id": {"JournalName":"$journal.title"},
      "articlesWritten": {"$addToSet": "$title"},
      "tagsUsed": {"$addToSet": "$metadata.keywords"}
    }},
    {"$addFields": {
      "numOfArtWritten": {"$size": "$articlesWritten"}
    }},
    {"$sort": {numOfArtWritten: -1}},
    {"$limit": 20},

    //SECOND PART: most used tags across the selected journals
    {"$unwind": "$tagsUsed"},
    {"$group":{
      "_id": {"Tag": "$tagsUsed"},
      "tagUsage": {"$sum": 1}
    }},
    {"$sort": {tagUsage: -1}},
    {"$limit": 10}
  ]);
```

The query is divided in two main parts.

The first one is responsible for the identification of the most active journals. We first unwind the articles considering the related keywords such that one article is published by one journal and it has one tag. Then we group the documents considering the journal name, we add the article title in the articlesWritten set, in order to prevent duplication, and also the same for the tags. To highlight the most productive journals and so the journals that have published more articles, at the group result we add a new field called numOfArtWritten that will be equal to the size of the articlesWritten set and so the total number of article written by a specific journal. We sort the results in a descending order considering the number of articles and we hold only the first 20 major journals.

The second part defines which are the most used tags across the previous selected journals. We first unwind all the documents considering the keywords contained in the tagsUsed

array. So we can combine together all the papers that share common tags and we can keep track of these thanks to the tagUsage variable. For our result, we are going to hold the tag name and the number of times it was used. tagUsage is used also for sorting in a descending order the output. We decided to keep only the first ten tags.

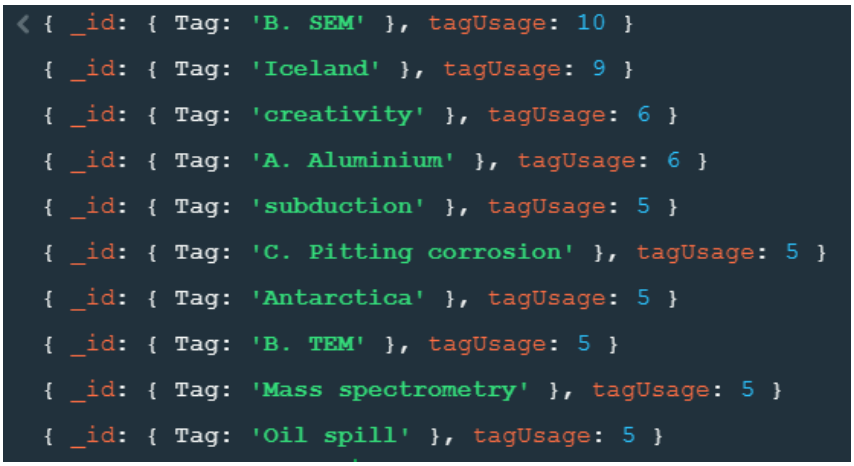


Figure 2.40: Results of the query Q3

PERFORMANCE: using *.explain("executionStats")* we obtain that the performance of the query is executionTimeMillis: 32

Q4. Articles that contain images of a given topic in a certain period

In order to find images about a certain topic, the query needs to focus on the *caption* field of the images. A research of this type will consider only images that have a "descriptive" caption, in the sense that it contains a sentence related to the subject of the image (e.g., images that nave a caption like "Figure 1" will not be considered if a word like "database" is searched). Since we are performing a text search, a textual index on the *caption* field has to be created.

Furthermore, the query extracts articles that belong to articles that are published only in a given period (indicated by two dates).

```
db.articles.aggregate([
  {"$match":{"$text":{"$search":"DNA"}}},
  {"$match":
    {"$expr":
      {"$and":[
        {"$gte":["$journal.date",ISODate("2010-01-01")]},
        {"$lte":["$journal.date",ISODate("2020-01-01")]}
      ]}
    }
```

```
},
{"$project":{"title":1,"metadata.doi":1,"journal.date":1}}
])
```

The query above looks for captions that contain the word "DNA" (or "dna" since the text operator is not case sensitive) and then filters the obtained articles with the second *$match* by the two given dates (that in this case are 01/01/2010 and 01/01/2020 included). Then, the last projection shows only the title, the doi and the publication date of the articles that have respected the previous conditions.

In the following picture is reported the result of the query with the given parameters.

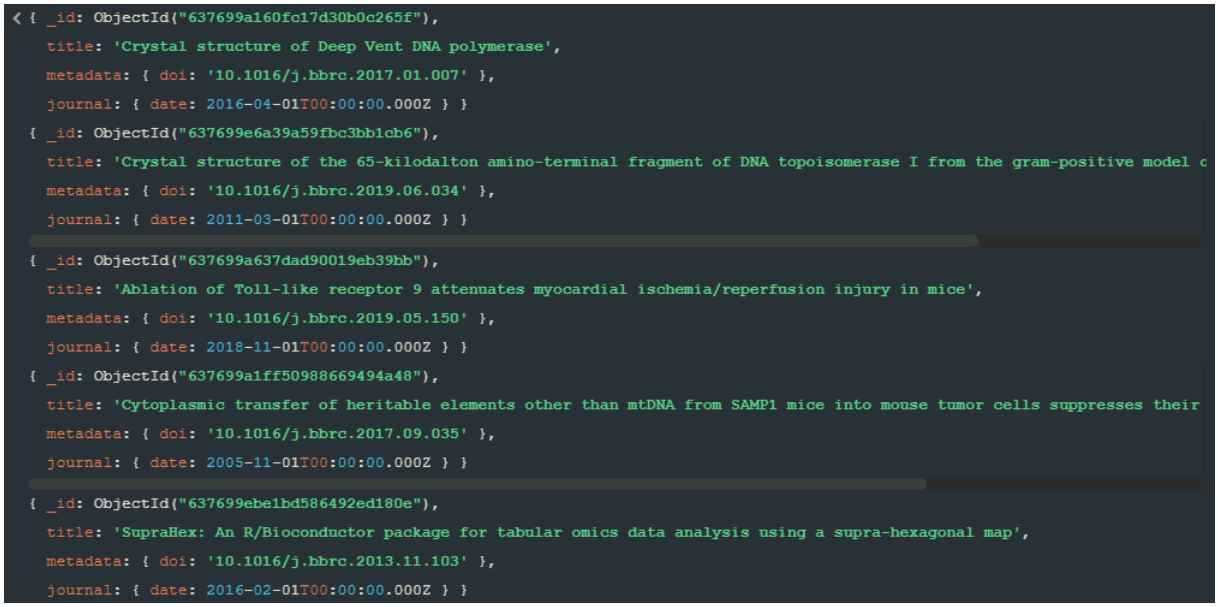


Figure 2.41: Results of the query Q4

PERFORMANCE: using *.explain("executionStats")* we obtain that the performance of the query is executionTimeMillis: 0

Q5.Find articles that have too many images with respect to the number of pages

The query goal is to find articles with too many images. In this use case example the query finds the articles with less than 3 pages and more then 10 images.

```
db.articles.aggregate([
    { $match :{
        $expr:{$lt:[ {$subtract:["$journal.last_page" , "$journal.
            start_page"]},3]}
        }
    },
    { $unwind: {path:"$sections"} },
    { $unwind: {path: "$sections.images"} },
    { $group:
        {
        _id: { id:"$_id" ,title:"$title"},
        numberOfImagesPerArticle:{ $sum:1}
        }
    },
    { $match : {numberOfImagesPerArticle:{$gt:10} }},
    { $sort : {numberOfImagesPerArticle :-1 } },
    { $project:{ numberOfImagesPerArticle:1, "_id.title":1}
    }
]);
```

In this query we use the *aggregate* method. In the first stage of aggregation we match the articles with 3 pages long. To retrieve the number of pages of an article it computes: number of last page – number of first page. In the second stage we unwind the sections and then we unwind the images. In the following we group per article and so we count how many images are contained in each article. In the fifth stage we match only the articles with at least 10 images. Finally, in order to provide a more readable output, the resulting documents are sorted for number of images per article and then only the important fields are projected.

```
{ _id: { title: 'Are white matter abnormalities associated with "unexplained dizziness"?' },
  numberOfImagesPerArticle: 17 }
{ _id: { title: 'The efficient computation of the nonlinear dynamic response of a foil-air bearing rotor system' },
  numberOfImagesPerArticle: 15 }
{ _id: { title: 'Higher-order WKB analysis of reflection from tapered elastic wedges' },
  numberOfImagesPerArticle: 15 }
{ _id: { title: 'Earth\'s deepest earthquake swarms track fluid ascent beneath nascent arc volcanoes' },
  numberOfImagesPerArticle: 14 }
```

Figure 2.42: Results of the query Q5

Example on how the unwind work in stage 2 and 3

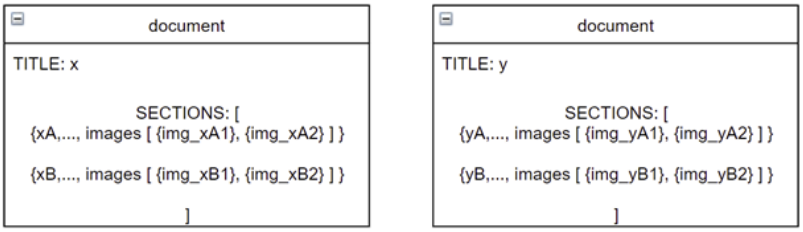


Figure 2.43: before the double unwind

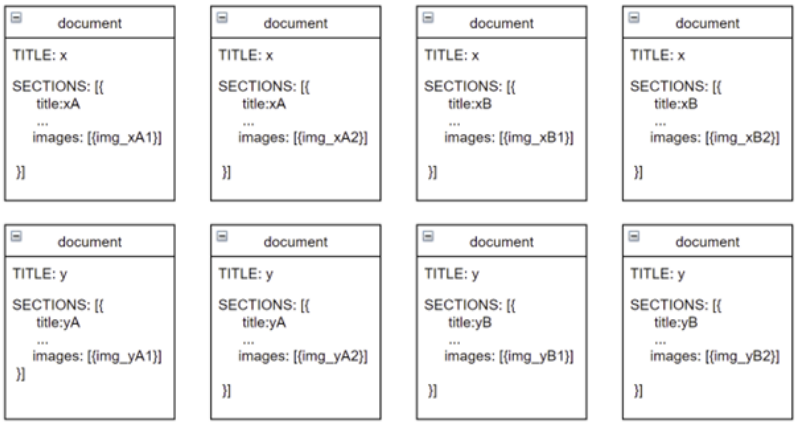


Figure 2.44: after the double unwind

PERFORMANCE: using *.explain("executionStats")* we obtain that the performance of the query is executionTimeMillis: 6

Q6. Find all recent articles, wrote by an important author, that are about a given topic

The query goal is to find articles that regards a given topic, furthermore those articles must be written at least by one important author. An author to be considered important must have at least two affiliations and must have the ORCID (since we have a dataset with an ORCID fixed for all authors, the importance of an author depends only on the number of his affiliations). In this example the topic is: "DNA" and with "recent articles" we mean articles written in the last 10 years.

```
db.articles.find(
  {
  $text : {$search : "DNA"},
  "journal.date": {$gt : ISODate('2012-01-01')},
   researchers:{
        "$elemMatch": {
                ORCID:{$exists: true},
                "affiliations.1" : {$exists: true}
        }
    }
  },{
    "title":1,"_id":0
    });
```

This query uses *find* method in which we apply three filters. The first filter consists in a text research for the given topic, to perform a research requires a textual index thus we must create a new one if it doesn't already exist. In this case we create a textual multi-key index that references "title", "abstract" and "sections.title".

Figure 2.45: The textual index

The second filter is a condition on the date of the article. The third filter checks if at least one element of the researchers array satisfies the two conditions, to check this we must use an *$elemMatch*. The query ends with a project of the titles.

The result of the query can be seen below.

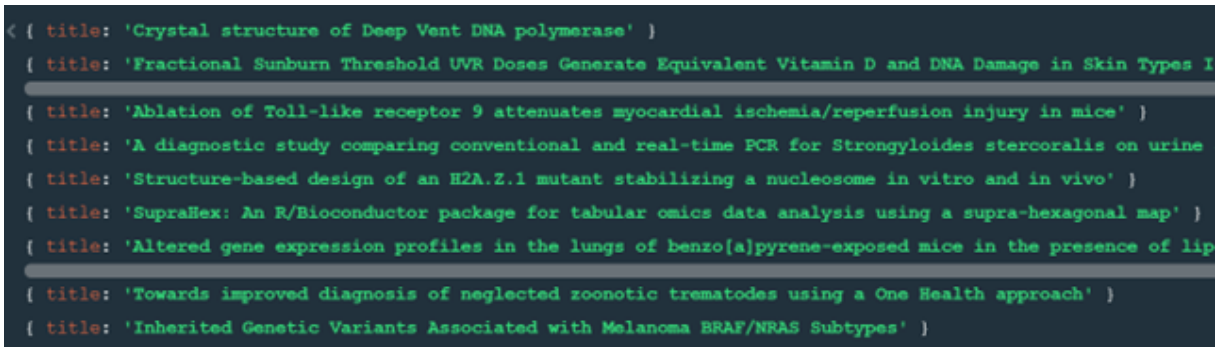


Figure 2.46: Results of the query Q6

PERFORMANCE: using *.explain("executionStats")* we obtain that the performance of the query is executionTimeMillis: 2

Q7. Find out all researchers who have had/have Italian affiliations and relative number of published articles.

The following query provides the main information on the authors who conducted research in Italy. In particular, it returns the name of the researcher, its surname, the total number of written articles and the name of the affiliations associated with the author. The results are sorted in descending order on the number of articles written by the researcher.

```
db.articles.aggregate([
    {
    "$unwind" : "$researchers"
    },{
        "$group":{
            "_id": {"name" : "$researchers.name", "surname" : "$researchers
                .surname"},
            "affiliations": {"$first": "$researchers.affiliations"},
            "written_amounts": {"$sum": 1}
        }
    },{
        "$match": {"affiliations": { "$regex" : /italy/i }}
    },{
        "$sort": {"written_amounts": -1}
    },{
        "$limit": 5
}]);
```

The query firstly executes the 'unwind' operation on the array of researchers so that the documents can be grouped by the author's first name and surname. For each author

added to the group we add 1 to the variable counting the number of articles written by the author. Obviously, for each author we keep the array collecting its affiliations via the *$first* command.
Once we have grouped the individual authors, we just have to exclude those who do not have affiliations in Italy. To do this, through the command *$regex*, we filter between those who have the word *italy* in the carrier affiliations. (*/italy/i* lets you filter the word *italy* regardless of whether it is upper or lower case).

```
{ _id: { name: 'Thor', surname: 'Thordarson' },
  affiliations:
   [ 'Ludwig Maximilians University Munich',
     'University of Hamburg',
     'University of Bologna, Italy' ],
  written_amounts: 4 }
{ _id: { name: 'Christopher E.M.', surname: 'Griffiths' },
  affiliations:
   [ 'University of Calabria, Italy',
     'Graduate University for Advanced Studies, Japan' ],
  written_amounts: 4 }
{ _id: { name: 'Gregory A.', surname: 'Brennecka' },
  affiliations:
   [ 'University of Koblenz Landau',
     'University of Trier, Germany',
     'University of Calabria, Italy' ],
  written_amounts: 2 }
{ _id: { name: 'Jennifer', surname: 'Chesters' },
  affiliations:
   [ 'University of Potsdam, Germany',
     'Technical University of Hamburg, Germany',
     'University of Calabria, Italy' ],
  written_amounts: 2 }
{ _id: { name: 'Alice D.C.', surname: 'Du Vivier' },
  affiliations:
   [ 'Polytechnic University of Milan, Italy',
     'Karlsruhe Institute of Technology, Germany' ],
  written_amounts: 2 }
```

Figure 2.47: Results of the query Q7

PERFORMANCE: using *.explain("executionStats")* we obtain that the performance of the query is executionTimeMillis: 56

By moving the *$match* command before the group statement, we reduce the execution-TimeMillis from 56 to 28

Q8. Find articles cited by a specific article dealing with a certain subject

The following query aims to verify if the articles cited by a given article deal with a set of specific topics. If this is the case, the query returns the title of the articles cited. Information about topics covered by an article is present within the metadata field of the document (external to the bibliography array field). It is necessary to join between the collection of all the articles mentioned by a given article and the entire collection in order to trace the information on the topics covered.

```
db.articles.aggregate([{
    "$match": {"metadata.doi": "10.1016/j.actatropica.2013.07.006"}
},{
   "$unwind": "$bibliography"
  },{
    "$lookup": {
      from: "articles",
      localField: "bibliography.doi",
      foreignField: "metadata.doi",
      as: "join_result"
    }
  },{
    "$match": {
      "join_result.metadata.keywords": {"$regex": /.*Sound radiation/i}
    }
  },{
    "$project": {"join_result.title": 1,"_id": 0}
  }]);
```

Before executing the join operation, we need to get the document of a specific article, by filtering on its DOI through the *$match* command. Thanks to the *$unwind* command we get each article cited and its DOI.
The join operation (*$lookup*) is executed on the DOI of each article cited from the local collection previously built and the DOI of each article in the primitive collection. As result, we get all the information of the articles mentioned (including keywords).
The last step before returning the result is to filter the vector of tags associated with each article according to the affected topic, through the *$regex* command.

```
{ join_result: [ { title: 'Numerical analysis of sound radiation from rotating discs' } ] }
```

Figure 2.48: Result of the query Q8

To verify if the query is working properly, we can execute another one that returns all

the articles mentioned by the given article (which has the doi equal to ...) and related keywords associated with each of them.

```
db.articles.aggregate([{
    "$match": {"metadata.doi": "10.1016/j.actatropica.2013.07.006"}
},{
   "$unwind": "$bibliography"
  },{
    "$lookup": {
      from: "articles",
      localField: "bibliography.doi",
      foreignField: "metadata.doi",
      as: "join_result"
    }
  },{
  "$project": {"join_result.metadata.keywords": 1,"_id": 0,"join_result.
     title": 1}

  }]);
```

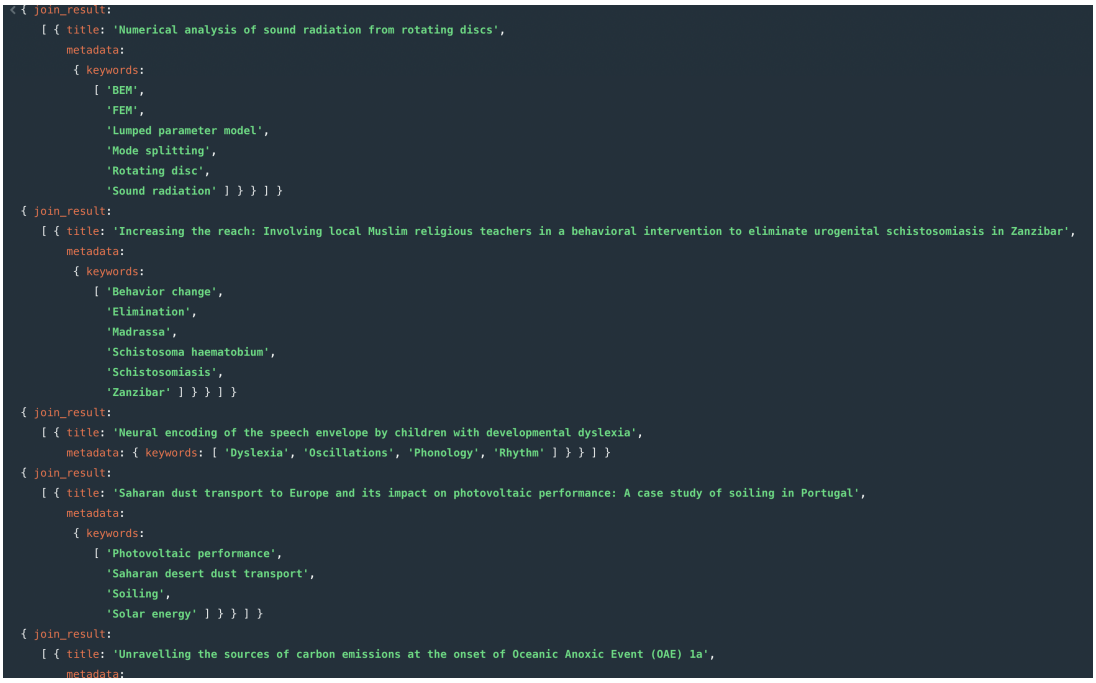


Figure 2.49: Portion of the results of the query Q8

PERFORMANCE: using *.explain("executionStats")* we obtain that the performance of the query is executionTimeMillis: 17

Q9. Top 10 longest articles by number of paragraphs

The aim of this query is to extract the 10 longest articles by number of paragraphs contained in the document. In order to compute a total number of paragraphs, the query considers both those linked to sections and those linked to subsections.

```
db.articles.aggregate([{
  "$unwind": "$sections"
},
{
  "$unwind": {path: "$sections.subsections",preserveNullAndEmptyArrays:
     true}
},
{
  "$group": {
    "_id": ["$title","$sections.title"],
    "amount_of_paragraphs_of_section": {"$avg": {"$sum": {"$size": "
       $sections.paragraphs"}}},
    "amount_of_paragraphs_of_subsections": {"$sum":
      {
        "$cond": {
          if: {"$isArray": "$sections.subsections.paragraphs"},
          then: {"$size": "$sections.subsections.paragraphs"},
          else: 0
        }
      }
    }
  }
},{
    "$project": {"_id": 1, "total_par":{"$add":["
       $amount_of_paragraphs_of_subsection","
       $amount_of_paragraphs_of_sections"]}}
},{
  "$group": {
    "_id": {"$first": "$_id"},
    "total_length": {"$sum": "$total_par"}
  }
},{
  "$sort": {"total_length": -1}
},{
  "$limit": 10
}]);
```

The first part of the query is composed of two `unwind` commands: the first one on the sections (that creates a document for each section of an article) and the second one on the **eventual** subsections (that creates a document for each subsection of a section). In fact, not every section has a set of subsections and, for this reason, the second unwind has the parameter *preserveNullAndEmptyArrays* set to **true**. In this way, documents that contain a section that does not have a subsection are kept.

The first *$group* operator is used to group back documents characterized by the same title and section title. The aim of this group is to count at the same time the number of paragraphs contained in a section and the sum of all the paragraphs contained in the subsections linked to the section itself. In particular, the two fields of the group are:

- `amount_of_paragraphs_of_section`: the number of paragraphs of a section is calculated by computing the size of the array that contains the paragraphs. Since the `group` operator needs an accumulator, the size of the vector is added and then averaged, in order to keep the right value.
- `amount_of_paragraphs_of_subsections`: the number of paragraphs of a subsection is given by the sum of all the sizes of the paragraphs arrays (related to the subsections). As already stated before, it is possible that some documents do not contain subsections. For this reason, the *$cond* operator is needed: if *sections.subsections.paragraphs* is an array, its size is added. Otherwise, the added amount is 0.

Then, we use a *$project* operator to sum the two previous "partial" results (with the *$add* operator) and the total number of paragraphs **per section** is obtained. Only one last step is needed: the sections that belong to the same article have to be added together.

The second *$group* operator uses as the *id* the title of the article (retrieved with the *$first* operator). Then, *total_length* will contain the sum of the paragraphs of the whole document.

Finally, the obtained results are sorted by the total length and the number of results is limited to 10.

An example of the result of the query can be seen below.

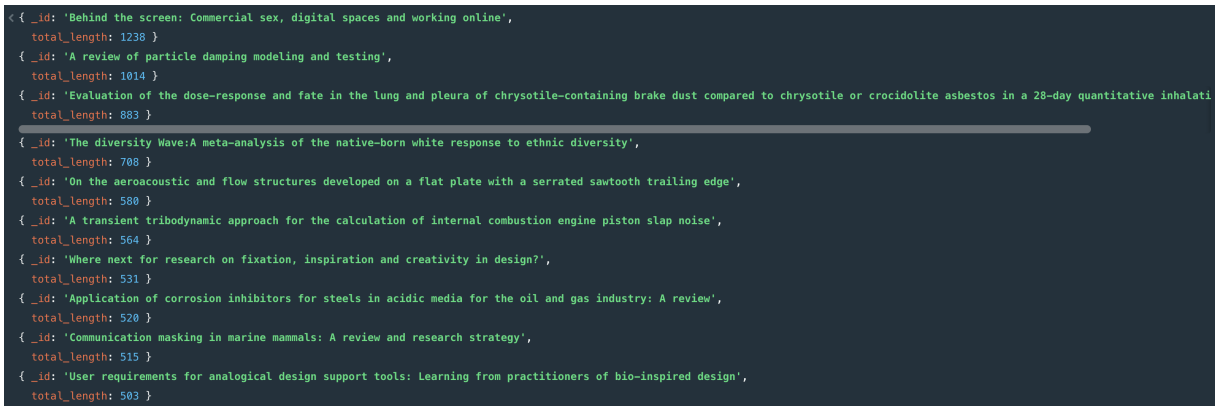


Figure 2.50: Result of the query Q9

PERFORMANCE: using *.explain("executionStats")* we obtain that the performance of the query is executionTimeMillis: 49

Q10. Articles never cited by other articles

The following query finds the title and doi for all the published articles that are never mentioned by other articles and were published on a given journal.

```
db.articles.aggregate([
  { "$match" : { "journal.ISSN" : "4277-3779" }},
  { "$lookup" : {
      from : "articles",
      localField : "metadata.doi",
      foreignField : "bibliography.doi",
      as : "citing_papers"
    }
  },
  { "$match" : { "citing_papers.0" : { "$exists" : false }}},
  { "$project" : { "title" : 1, "metadata.doi" : 1 }}
]);
```

In the query the documents are filtered so that only the ones containing articles published on the specified journal are kept (the ISSN of the journal is given in input). For each article, the join produces an array collecting all the documents containing the articles citing it. By verifying the presence of at least one element in such array it is possible to determine which are the articles that are never cited.

Similarly we can find the articles only cited once, twice and so on and so forth by simply

increasing the number 0 in the match clause.

```
{ _id: ObjectId("637699ba9949b8f73bbb5218"),
  title: 'Development of a regional glycerol dialkyl glycerol tetraether (GDGT)-temperature calibration for Antarctic and sub-Antarctic lakes',
  metadata: { doi: '10.1016/j.epsl.2015.11.018' } }
{ _id: ObjectId("63769a098544c7e05cd14e91"),
  title: 'Fracture strength of electroslag welding joint with high-performance steel',
  metadata: { doi: '10.1016/j.jcsr.2018.11.010' } }
```

Figure 2.51: before the update

PERFORMANCE: using *.explain("executionStats")* we obtain that the performance of the query is executionTimeMillis: 395

2.3.5. Commands

In the following you are presented with some of the possible commands that can be executed on the database. For each statement is provided a short description, the code and an example of the result produced upon its execution.

Commands can be either creation, update, delete statements or a combination of those three.

C1: Insertion of a new article

This command allows us to insert in the dataset a new article. The insertion operation can be performed in two ways:

- import a JSON file thanks to the mongoimport utility
- using the insertOne command

We will show the second option because the first one was already presented in the 2.3.3 import section.

Due to the large dimension of the document we want to import, we provide only its first part.

Figure 2.52: Portion of the new article

In order to add this paper to the collection, we execute the following instruction:

```
db.articles.insertOne({
    << article1001 >>
    })
```

This is the result of the operation:

Figure 2.53: Result of the command C1

C2: Updating a researcher's bio

This command allows to modify the biography of a researcher provided its name and surname. Besides, given that the same researcher object is duplicated in multiple document, it is necessary to update its value for every single occurrence in the various documents.

The update command first filters the documents containing the researcher in which we are interested, then the property is updated. The researchers authoring an article are organized in an array of authors, we then want to modify only the biography of the designed researcher. In order to do so the positional operator $ is used; it allows to identify an element in an array without specifying its position.

The same command can be used both for modifying information and adding further details about any researcher. To do so it's enough to change "bio" in the set clause with any field of interest.

```
db.articles.updateMany(
  {"$and":[{"researchers.name":"Laura J.A."},{"researchers.surname":"
     Hardwick"}]},
  {"$set":{"researchers.$.bio":"Full time researcher specialized in ..."}}
  );
```

The result of the command can be seen at the following page.

```
{ name: 'Laura J.A.',
  surname: 'Hardwick',
  ORCID: 'https://orcid.org/0000-0000-0000-0000',
  bio: 'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore mag
liqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis au
rure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupida
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.',
```


Figure 2.54: Laura J.A. before the update


```
[ { name: 'Laura J.A.',
    surname: 'Hardwick',
    ORCID: 'https://orcid.org/0000-0000-0000-0000',
    bio: 'Full time researcher specialized in ....',
```


Figure 2.55: Laura J.A. after the update

C3: Updating all occurrences of a string field to date type

This command is used to convert a field type from string type to date type in all the document in which that field occurs.

This query is particularly useful because dataset has often date in gg-mm-yyyy format, that is not directly convertible to ISODate format because ISODate requires yyyy-mm-gg. To solve this problem the query use "$convert" that works on a wider range of date format.

```
db.articles.updateMany(
  {"journal.date" : {$exists:true} },
  [{ "$set": { "journal.date": { "$convert": {input:"$journal.date", to:"
      date", onError:"err" } } }}]
);
```


```
{ acknowledged: true,
  insertedId: null,
  matchedCount: 1000,
  modifiedCount: 1000,
  upsertedCount: 0 }
```


Figure 2.56: Result of the updates

C4. Pull out researchers with no name and surname

We are aware of the fact that in the dataset there are some researchers with *null* name and surname: the objective of the actual command is to remove only those authors.

First let's find the number of documents with at least one author with null name and surname.

```
db.articles.aggregate([
  {"$match": {"$and": [{"researchers.name": {"$eq": null}}, {"researchers.
     surname": {"$eq": null}}] }},
  {"$group": {
    _id: true,
    articlesWithNullResearchers: {"$sum": 1},
  }}
  ])
```

We keep only those documents in which at least one author has null name and surname. Then we group all of them considering a fake id and we update the articles with null researchers counter.

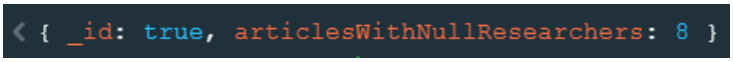


Figure 2.57: Results of the command C4.a

We can see that there are only eight articles that have at least one author with no name and surname.

In order to be able to check if the query works, let's consider for example two articles in which at least one author has name and surname equal to null.

```
db.articles.find({
  "$and": [{"researchers.name": null},{"researchers.surname": null}]
},
{
  "_id": 1, "title": 1, "researchers.name": 1, "researchers.surname": 1
}).limit(2)
```

We want to find two documents in which at least one author has null name and surname. We print out the document id, the article title and all the article researchers.

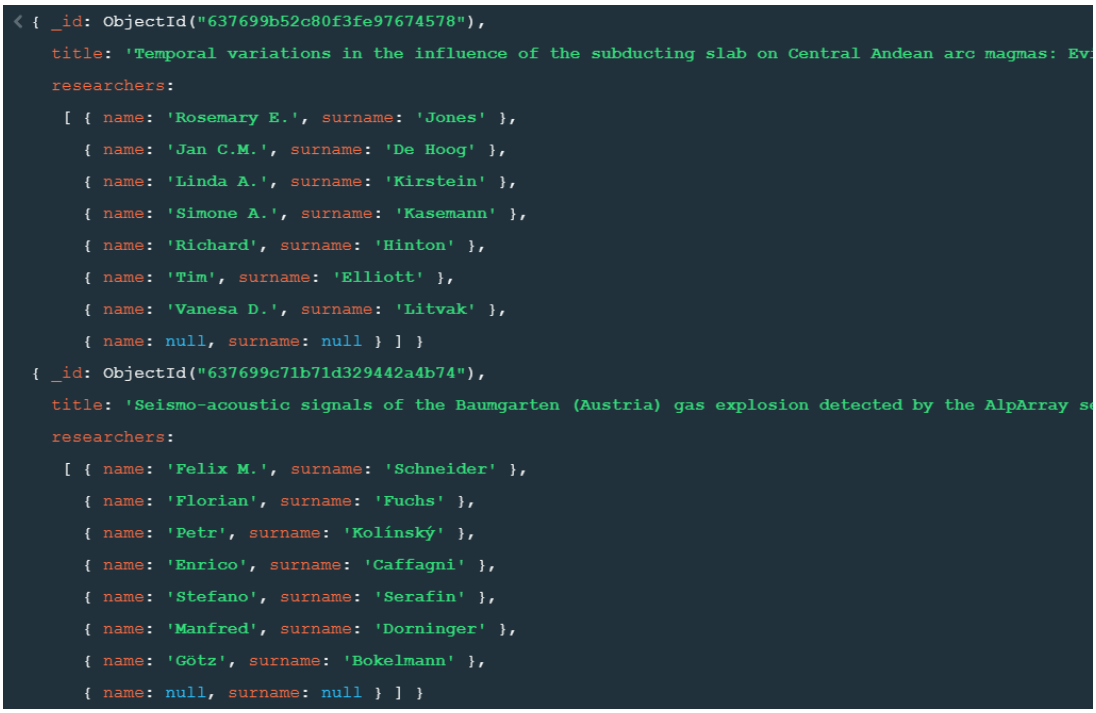


Figure 2.58: Results of the command C4.b

Now it is possible to run the main query that will be able to pull out from the authors array only those researchers with no name and surname.

```
db.articles.updateMany(
  {},
  {$pull: {researchers: {name: null, surname: null}}}
  )
```

Since we want to consider all the documents within the dataset, the filter section is empty. From the researchers array we pull out only the researchers having null name AND surname.

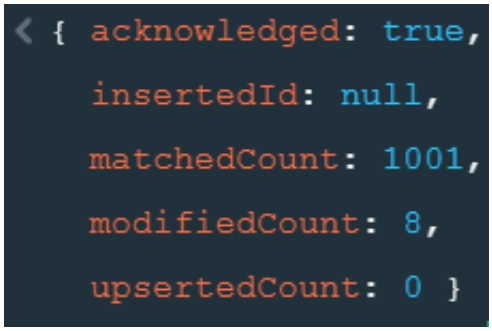


Figure 2.59: Results of the command C4.c

So the query checked all the documents as the "matchedCount" field suggests and the update regarded only eight documents as we have highlighted previously.

Finally we can check the initial two selected articles. They can be found through their id.

```
db.articles.find(
  {_id: ObjectId("637699b52c80f3fe97674578")},
  {"_id": 1, "title": 1, "researchers.name": 1, "researchers.surname": 1}
  )
```

```
{ _id: ObjectId("637699b52c80f3fe97674578"),
  title: 'Temporal variations in the influence of the subducting slab on Central An
  researchers:
   [ { name: 'Rosemary E.', surname: 'Jones' },
     { name: 'Jan C.M.', surname: 'De Hoog' },
     { name: 'Linda A.', surname: 'Kirstein' },
     { name: 'Simone A.', surname: 'Kasemann' },
     { name: 'Richard', surname: 'Hinton' },
     { name: 'Tim', surname: 'Elliott' },
     { name: 'Vanesa D.', surname: 'Litvak' } ] }
SMBUD-Second-Delivery>
```

Figure 2.60: Results of the command C4.d1

```
db.articles.find(
  {_id: ObjectId("637699c71b71d329442a4b74")},
  {"_id": 1, "title": 1, "researchers.name": 1, "researchers.surname": 1}
  )
```

```
{ _id: ObjectId("637699c71b71d329442a4b74"),
  title: 'Seismo-acoustic signals of the Baumgarten (Austria) gas explosion detected
  researchers:
   [ { name: 'Felix M.', surname: 'Schneider' },
     { name: 'Florian', surname: 'Fuchs' },
     { name: 'Petr', surname: 'Kolínský' },
     { name: 'Enrico', surname: 'Caffagni' },
     { name: 'Stefano', surname: 'Serafin' },
     { name: 'Manfred', surname: 'Dorninger' },
     { name: 'Götz', surname: 'Bokelmann' } ] }
SMBUD-Second-Delivery>
```

Figure 2.61: Results of the command C4.d2

The empty authors have been deleted.

If we check the number of documents with authors that have no null name and surname it should be equal to the entire dataset.

```
db.articles.aggregate([
  {"$match": {"$and": [{"researchers.name": {"$ne": null}}, {"researchers.
     surname": {"$ne": null}}] }},
  {"$group": {
    _id: true,
    articlesWithNotNullResearchers: {"$sum": 1}
  }},
  ])
```

```
{ _id: true, articlesWithNotNullResearchers: 998 }
```

Figure 2.62: Results of the command C4.e

Since we have obtained a number that does not represent the whole dataset dimension, it is evident that there is an inconsistency. This means that there are also article in which the name **or** the surname are not `null`. We can check by executing the following query:

```
db.articles.find({
  "$or": [{"researchers.name": null},{"researchers.surname": null}]
},
{
  "_id": 1, "title": 1, "researchers.name": 1, "researchers.surname": 1
})
```

These are the researchers that we are looking for:

```
{ name: null, surname: 'Karakaş' } ] }
```

Figure 2.63: Results of the command C4.f1

```
{ name: null, surname: 'EIMF' } ] }
```

Figure 2.64: Results of the command C4.f2

```
{ name: null, surname: 'Keskin' },
```

Figure 2.65: Results of the command C4.f3

By executing again the command C4.a paying attention at changing the operator "$and" to "$or" we confirm our hypothesis.

```
< { _id: true, articlesWithNullResearchers: 3 }
```

Figure 2.66: Results of the command C4.g

We apply the same policy also to these researchers so they will be pulled out from their arrays. The command C4.c can be reused with some little modifications: we inserted the OR operator.

```
db.articles.updateMany(
  {},
  {
    "$pull": {researchers: {"$or":[{name: null}, {surname: null}] }}
  }
  )
```

```
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1001,
    modifiedCount: 3,
    upsertedCount: 0 }
```

Figure 2.67: Results of the command C4.h

Three documents have been modified and now we can use again the C4-e command in order to check if our dataset contains incomplete data regarding the researchers' name and surname.

```
< { _id: true, articlesWithNotNullResearchers: 1001 }
```

Figure 2.68: Results of the command C4: Final Check

C5. Delete a document given the title

This command allows us to delete an article once specified its title.

We are going to delete the article inserted previously thanks to the command C1.

In order to delete this paper, we execute the following instruction:

```
db.articles.deleteOne({
  "title": "A new article: #1001"
})
```

This is the result of the operation:

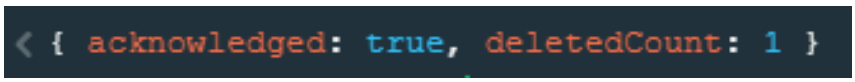


Figure 2.69: Result of the command C5

2.4. Spark implementation

This section will cover the implementation of the reality presented in chapter 1 "*Introduction*" with the Spark Technology.

2.4.1. Amendments to the ER

Since our goal is to focus only on relevant entities and to have a more readable model with a well sized data-set, in this delivery we have decided to remove the Series entity and its relationships. This deletion is done because the Series entity and its relationships do not add any kind of complexity to the project.
Thanks to the Spark technology (that allows *lists* as entity attributes), we are able to represent the Affiliations as we intended to (so, like a multiple attribute). For the same reason, the Tag entity was reduced to a multiple attribute of the Document entity.
Finally, as we did in Neo4j, all hierarchies have been resolved downwards.

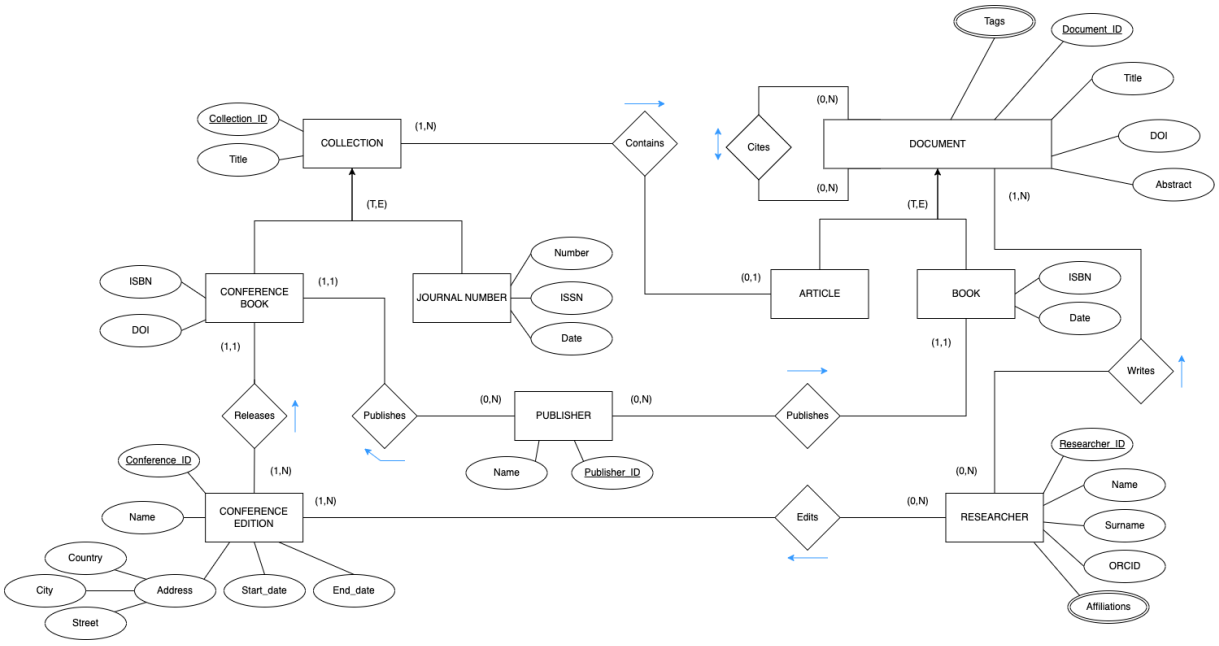


Figure 2.70: The ER model considered for Spark

2.4.2. Dataset

In order to translate the ER into Spark Dataframes, we have followed the following policies:

- Every entity has its own dataframe.
- (N:N) relationships have their own dataframe, made of two columns (each one is a key of the linked entities).
- For (1:N) relationships we used the concept of "foreign keys". In the dataframes of the entities on the "1" side, a column that stores the IDs of the linked entity was added.

For each table (CSV file) of our dataset, we have built its own dataframe. For each of them, we import data from a CSV and then we define the schema.

Creation

The dataset used for this delivery is mostly based on the one used for the Neo4j implementation (its description can be found at section 2.2.2).
As stated before, the first difference consists in the introduction of the "foreign keys", which substitute the tables used for 1 to N relationships. These foreign keys are stored as fields in the entity that belongs to the side "1" of the relationship.
In order to create the lists of Affiliations and Tags, a Python program have been used. For the Affiliations, each Researcher have been linked with its Affiliations by looking for his ID in the old CSV table that represented the relationship. On the other hand, Tags for each Document have been randomized (since in Neo4J we had few).
So, the resulting schema for each entity of the dataset is:

```
Article (Article_ID;Title;DOI;JournalNumber_ID;ConferenceBook_ID;Tags)
Book (Book_ID;Title;DOI;Year;ISBN;Tags;Publisher_ID)
ConferenceEdition (Conference_ID;Name;Country;City;Street;Start_date;End_date)
JournalNumber (JournalNumber_ID;Title;ISSN;Number;Month;Year)
Researcher (Researcher_ID;Name;ORCID;Affiliations)
Publisher (Publisher_ID;Name)
ConferenceBook (ConferenceBook_ID;Title;ISBN;DOI;ConferenceEdition_ID;Publisher_ID)
```

Import

Once the dataset is defined, the next step is to create an entry point for the application through the pyspark interface and define the individual dataframes within it.

```
spark = SparkSession.builder\
    .master("local") \
    .appName("SparkApplicationThirdDelivery") \
    .config("spark.ui.port","4050")\
    .getOrCreate()

#All the DataFrames definitions...

text = input("type enter to close the session")
if text == "":
    spark.stop()
    print("session closed")
```

Through the SparkSession class constructor method we can initialize and configure a session. In particular through *.master()* we set the URL of the Spark master to connect to (in this case locally) and via *.config()* the port number of the localhost to which the Apache Spark dashboard user interface is connected (accessible via the *localhost:4050* URL after the spark program starts). Because the local server starts when the program starts, but closes at the end, the code at the end of the program keeps the session open until the user closes it manually. In this way, the user is allowed to interact with the Web UI.

For each table (CSV file) of our dataset, we build its dataframe, where for each of these we import data from CSV and define the schema. The creation of the dataframe for Articles is presented below.

```
#Schema definition
ArticleSchema = StructType([ \
    StructField("Article_ID", LongType(), True), \
    StructField("Title", StringType(), True), \
    StructField("DOI", StringType(), True),\
    StructField("JournalNumber_ID", LongType(), True),\
    StructField("ConferenceBook_ID", LongType(), True),\
    StructField("Tags", StringType(), True)
])
```

```
#CSV File import to a dataframe
Article_DF = spark.read.format("CSV")\
     .option("header","true")\
     .option("delimiter",";")\
     .schema(ArticleSchema)\
     .load("Article.CSV")

#Conversion of the "Tag" field into an array of strings
Article_DF = Article_DF.withColumn("Tags", split(regexp_replace(col("Tags")
    , '[\[\]"]', ""), ","))

Article_DF.printSchema()
Article_DF.show()
```

For each schema the type of structure is defined, specifying the name of the columns, the type and whether the field can be *null* or not. Particular attention should be given to multi-value attributes, which have a *ArrayType()* structure. Since a CSV data source does not support *ArrayType()* data type, it was necessary to read the columns of the multi-value attributes as a single string *StringType()* and, after the creation of the DataFrame, the selected column is divided by specifying the type of delimiter between the individual elements. This automatically updates the column structure type from *StringType()* to *ArrayType()*. The result is shown in Figure 2.71, by executing the command *printSchema()*.

```
|-- Article_ID: long (nullable = true)
|-- Title: string (nullable = true)
|-- DOI: string (nullable = true)
|-- JournalNumber_ID: long (nullable = true)
|-- ConferenceBook_ID: long (nullable = true)
|-- Tags: array (nullable = true)
|    |-- element: string (containsNull = false)
```

Figure 2.71: ArrayType structure

Since the dataset is very similar to the one used for the Neo4j delivery, the total number of tuples is about 3000.

2.4.3. Queries

In this section are reported and commented some of the possible queries that can be executed on the database. For each statement is provided a short description, the code and an example of the result produced upon its execution.

Q1: Books published by a given publisher touching given topics

The following query returns the title and the ISBN of the books published by a specific publisher and that are characterized by at least one of the two specified tags.

```
# Params
publisher = "Logos-Verlag"
tag_1 = "Data Mining"
tag_2 = "Machine Learning (ML)"

#1. Filtering the publishers
filtered_publishers = Publisher_DF.filter((Publisher_DF.Name == publisher))

#2. Filtering the books
filtered_books = Book_DF.filter((array_contains(Book_DF.Tags, tag_1))|(
    array_contains(Book_DF.Tags, tag_2)))

#3. Joining the dfs so as to retrieve only the title of the books published
     by the specified publisher
filtered_books.join(filtered_publishers, filtered_books.Publisher_ID ==
    filtered_publishers.Publisher_ID, "inner")\
        .select("Title", "ISBN")\
        .show(truncate=False)
```

The first *filter* filters the publishers so that only the one in which we are interested is kept. While the second *filter* excludes the books that are not characterized by any of the two given tags. Finally with the *join* only the tuples of books published by the publisher of interest are kept.

```
+-----------------------------------------------------------------------------------------------------+----------------+
|Title                                                                                                |ISBN            |
+-----------------------------------------------------------------------------------------------------+----------------+
|Ada 2012 Reference Manual. Language and Standard Libraries - International Standard ISO/IEC 8652/2012 (E)|978-3-658-14814-0|
|Mission-Oriented Sensor Networks and Systems: Art and Science - Volume 1: Foundations                |978-3-658-12615-5|
+-----------------------------------------------------------------------------------------------------+----------------+
```

Figure 2.72: Result of the query Q1

Q2: Researchers from Italian affiliations

The following question shows the researchers with the list of Italian affiliations and the number of entries on the list. The entire list is sorted in descending order of the number of affiliations.

```
#1. Explode on affiliations column for each researcher.
explode_on_affiliations = Researcher_DF.select(col("Researcher_ID"),col("
    Name"), explode(col=("Affiliations")).alias("Affiliation"))

#2. Filter only Italian affiliations.
let_italian_affiliations = explode_on_affiliations.where(
    explode_on_affiliations.Affiliation.ilike("%italy%"))

#3. Grouping for each researcher, collecting all the affiliations into a
    list and counting the amount. the list is sorted in descending order by
    number of affiliations.
group_for_researcher = let_italian_affiliations.groupBy("Researcher_ID").
    agg(collect_list("Affiliation").alias("Affiliations_list"),count("Name")
    .alias("Affiliations_amount"))\
        .orderBy(col("Affiliations_amount").desc())\
            .limit(10).show(truncate=False)
```

The first operation performed is the *explode* function on the affiliations column for each researcher within the *Reasearcher* dataframe. Next, filtering the different affiliations that contain the word Italy through the function *like* (similar to *like*, but case-insensitive), we store information only on Italian affiliations. The last operation of the query consists in sorting the results, grouping the Italian affiliations for each researcher within a single list, through the *collect_list* function. Using the *count* function, we calculate the number of items in each list. The result of the query is displayed in the image below.

```
+--------------------+------------------------------------------------------------------+----------------+
|Name                |Affiliations_list                                                 |Affiliations_num|
+--------------------+------------------------------------------------------------------+----------------+
|Guido Frisch        |[Polytechnic University of Milan - Italy, University of Calabria - Italy]|2               |
|Fernando Palluzzi   |[Polytechnic University of Milan - Italy]                         |1               |
|Giulio Ermanno Pibiri|[University of Bologna - Italy]                                   |1               |
|Hamid R. Barzegar   |[University of Bologna - Italy]                                   |1               |
|Nicola Peserico     |[Polytechnic University of Milan - Italy]                         |1               |
|Pietro Ciccarella   |[Polytechnic University of Milan - Italy]                         |1               |
|Paula Trujillo      |[University of Calabria - Italy]                                  |1               |
|Paolo Trigilio      |[Polytechnic University of Milan - Italy]                         |1               |
|Piero Borga         |[University of Bologna - Italy]                                   |1               |
|Chiara Veneroni     |[Polytechnic University of Milan - Italy]                         |1               |
+--------------------+------------------------------------------------------------------+----------------+
```

Figure 2.73: Result of the query Q2

Q3: Researchers that have written unpublished articles

The goal of this query is to extract Researchers that have written articles that are still not published. An article is considered "unpublished" if it is not contained in a Journal Number or in a Conference Book. In our case, we can see that an Article is unpublished if it has both the *Journal* and the *ConferenceBook* fields equal to *null*.

```
#1. Finding "unpublished" articles
unpub_articles = Article_DF.filter(col("JournalNumber_ID").isNull() & col("
    ConferenceBook_ID").isNull())

#2. Extraction of the IDs of the articles
articles_ids = unpub_articles.select("Article_ID").collect()
articles_ids = [ai[0] for ai in articles_ids]

#3. Filtering the "writes" relationship and extracting IDs
researchers_of_articles = Writes_Article_DF.filter(col("Article_ID")\
                                                   .isin(articles_ids))

#4. Extraction of the IDs of the Researchers
authors_ids = researchers_of_articles.select("Researcher_ID").collect()
authors_ids = [ai[0] for ai in authors_ids]

#5. Final extraction of the Researchers
Researcher_DF.filter(col("Researcher_ID").isin(authors_ids)).show(truncate=
    False)
```

The first *filter* of the query is the one responsible for checking that the two fields mentioned before are *null*. Then, the query selects only the field *Article_ID* from the RDD and turns it into a list. Then, with the third step, from the RDD that represents the relationship between Researchers and Articles, the query takes only the rows that have their *Article_ID* contained in the list extracted from the previous step. Then, a list of the *Researcher_IDs* is made. These IDs belong to the authors that have written the previously extracted unpublished articles. Finally, the query shows all the desired researchers (with their data) by verifying that their ID is contained in the *authors_ids* list. The result of this query is displayed in the image below.

Researcher_ID	Name	ORCID	Affiliations
106	Alessandro Caspani	0103-1034-5433-6819	[University of Ha...
107	Davide Nicolis	0104-1035-5434-6820	[Johannes Gutenbe...
108	Soufiane Meddouri	0105-1036-5435-6821	[Martin Luther Un...
109	Costanza Messeri	0103-1034-5433-6820	[Technical Univer...
110	Nicola Lusardi	0104-1035-5434-6821	[Technical Univer...
111	Shima Zahmatkesh	0105-1036-5435-6822	[Gesamthochschule...
112	Matteo Sangiorgio	0103-1034-5433-6821	[Dortmund Univers...
113	Yu Zou	0104-1035-5434-6822	[Bundeswehr Unive...
114	Marco Ceriani	0105-1036-5435-6823	[Ludwig Maximilia...
115	Eirini Stamoulakatou	0103-1034-5433-6822	[Helmut Schmidt U...
116	Alessio La Bella	0104-1035-5434-6823	[Akademie der Lan...
117	Riccardo Cattaneo	0105-1036-5435-6824	[University of Lu...
118	Mario Laudato	0103-1034-5433-6823	[University of Re...
119	Dmytro Cherniak	0104-1035-5434-6824	[University of Bern]
120	Miguel Sotaquira	0105-1036-5435-6825	[Goethe Universit...
121	Cesare Buffa	0103-1034-5433-6824	[University of Er...
122	Giancarlo Mantovani	0104-1035-5434-6825	[University of Wu...
123	Ramona Cabiddu	0105-1036-5435-6826	[University of Fr...
124	Marco Ciccone	0103-1034-5433-6825	[University of Ma...
125	Stefano Dellea	0104-1035-5434-6826	[University of Be...

Figure 2.74: Result of the query Q3

Q4: Top 10 biggest journal numbers

This query returns the top 10 journal numbers that contain more articles.

```
#group by JournalNumber_ID and count, join to know journal title
Article_DF.groupBy( col("JournalNumber_ID") ).count() \
    .join(
        JN_DF.select(col("JournalNumber_ID"), col("Title")),
        Article_DF.JournalNumber_ID == JN_DF.select(col("JournalNumber_ID")
            ,col("Title") ).JournalNumber_ID
    ) \
    .withColumnRenamed("count", "number of articles contained")\
    .drop(JN_DF.JournalNumber_ID)\
    .sort(col("number of articles contained").asc())\ .limit(10).show(
       truncate=False)
    #renaming sort and limit the output
```

First of all, the query groups the articles by "Journal_Number_ID", thus it counts how many articles there are in every journal number. Then, it joins the result with some relevant columns of journal number data-frame, the join allows to know the journal title of the journal number. The last part of the query regards only some adjustments of the output, including renaming, sort and limit.

```
+---------------+--------------------------+-----------------------------------------+
|JournalNumber_ID|number of articles contained|Title                                   |
+---------------+--------------------------+-----------------------------------------+
|26             |5                         |Bull. dInformatique Approfondie et Appl. |
|29             |5                         |Trans. Large Scale Data Knowl. Centered Syst.|
|65             |5                         |Inf. Technol. Dev.                       |
|19             |5                         |EAI Endorsed Trans. Ubiquitous Environ.  |
|54             |5                         |Found. Comput. Math.                     |
|22             |5                         |Int. J. Trust. Manag. Comput. Commun.    |
|7              |5                         |SIGMOD Record                            |
|77             |5                         |J. Assoc. Inf. Syst.                     |
|34             |5                         |Trans. Large Scale Data Knowl. Centered Syst.|
|50             |5                         |Sci. Comput. Program.                    |
+---------------+--------------------------+-----------------------------------------+
```

Figure 2.75: Result of the query Q4

Q5: List of Conference Book Articles regarding a specific topic

The query goal is to retrieve for each topic a list of articles that have been published through a conference book.

```
#1. Collecting only the articles belonging to conference book
ConferenceBookArticles = Article_DF.filter(col("ConferenceBook_ID").
    isNotNull())

#2. Keep the title column and unwind the Tags array
ArticleAndTags = ConferenceBookArticles.select(col("Title"), explode("Tags
    "))

#3. Rename the column col
ArticleAndTags = ArticleAndTags.withColumnRenamed("col", "Tags")

#4. Group by tag values: count the articles and collect in a list those
    articles characterized by a specific tag
listOfArticlesPerTag = ArticleAndTags.groupBy("Tags")\
    .agg(count("Title").alias("Number_of_Articles"), collect_set("Title").
        alias("List_of_Articles"))\
    .sort(col("Number_of_Articles").asc())

listOfArticlesPerTag.show(truncate = False)
```

At the beginning, we keep only those articles that belong to a conference book and so the ones that have a not null foreign key.

Then we unwind the articles considering their tags and we also hold the article title.

Now we are able to design the result table: we group the previous tuples considering the values in the 'Tags' column, we count the articles with same tag and we insert them in the 'List of articles' and finally we sort them considering the 'Number of Articles' value in an ascending order to show in an easier way more results.

Tags	Number_of_Articles	List_of_Articles
Artificial Intel...	1	[DIIM: A Foundati...
Machine Learning...	3	[Interworkflow Sy...
Neural Networks	6	[Smart Mobility a...
Cloud Computing	8	[Towards Tweet Co...
Block chain Techn...	8	[A Model for Proc...
Knowledge Processing	8	[Generic Workflow...
Digital Image Pr...	9	[D2D group commun...
Internet of Thin...	9	[Application Mode...
Large Scale Networks	9	[IoT Healthcare: ...
Robotics	9	[Integrating gami...
5G Wireless System	10	[Managing Data Qu...
Intelligent System	10	[Distributed Sema...
Natural Language...	12	[Efficient Matchi...
Virtual Reality	12	[Fine-Grained Par...
Virtual Reality	13	[Collective intel...
High performance ...	13	[Transparent Prot...
Semantic Web	13	[A Conceptual Mar...
Data Mining	14	[Knowledge Extrac...
Computer Vision	14	[CoviHealth: Nove...
Web and Informat...	15	[The Role of Data...

only showing top 20 rows

Figure 2.76: Compact result of Q5

Artificial Intelligence (AI)	1	[DIIM: A Foundation for Translating Loosely-Specified Queries into Executable Plans in Large-Scale Information Systems.]
Machine Learning (ML)	3	[Interworkflow System: Coordination of Each Workflow amoong Multiple Organizations., IoT-Based Health Monitoring System for Active and Assisted Living., Panel on Cooperative MultiAgent Systems for the Web.]
Neural Networks	6	[Smart Mobility and Sensing: Case Studies Based on a Bike Information Gathering Architecture., GRIDKIT: Pluggable Overlay Networks for Grid Computing., A Multiformalism Approach to Formalize Intelligent Cooperative Information Systems., Action Port Model: A Mixed Paradigm Conceptual Workflow Modeling Language., A Meta-service for Event Notification., Enabling Smart Objects in Cities Towards Urban Sustainable Mobility-as-a-Service: A Capability - Driven Modeling Approach.]
Cloud Computing	8	[Towards Tweet Content Suggestions for Museum Media Managers., Agreement without knowing everybody: a first step to dynamicity., Usage-Oriented Evolution of Ontology-Based Knowledge Management Systems., Active Human Agency in Artificial Intelligence Mediation., A Web-service architectural perspective on Risk Management in work environments., A component model for homogeneous implementation of reconfigurable service-based distributed real-time applications., Model Driven Architecture: Three Years On., Cooperative Information Systems in Integrated Manufacturing Environmen

Figure 2.77: More verbose result of Q5

Q6: Countries that hosted more conference in a given period of time

The query goal is to highlight the countries that have hosted more conferences in a specific time interval.

```
#1. Conference editions held in a specified time interval
CE_Q6_DF = CE_DF.filter((CE_DF.Start_date >= "2010-01-01") & (CE_DF.
    End_date <= "2020-12-31"))

#2. Count the conferences editions that took place in each country
CE_Q6_DF = CE_Q6_DF.groupBy("Country").count()

#3. Renaming the column count
CE_Q6_DF = CE_Q6_DF.withColumnRenamed("count", "Number_of_conferences")

#4. Apply a condition to the aggregated metric
CE_Q6_DF = CE_Q6_DF.filter(col("Number_of_conferences") >= 4).sort(col("
    Number_of_conferences").desc())

CE_Q6_DF.show()
```

The first part of the query is able to keep only those conferences that have been held in a specified time interval: in this case between '2010' and '2020'. The remaining tuples have been grouped together considering the country property value and, for each one of them, we can count the number of conferences. Then we have renamed the column 'count' with 'Number of conferences': a more semantic name. Finally we have filtered out all the countries with a number of conferences less than four and we have sorted this result always considering the number of conferences.

Country	Number_of_conferences
China	21
Czech Republic	9
Russia	6
Indonesia	5
Philippines	5
United States	4
Poland	4

Figure 2.78: Result of the query Q6

Q7: Find publishers which have published more than 1 book in the last 5 years

This query returns the publisher ID and their number of book published in the last 5 years only if they have published more than 1 book in the last 5 years.

```
yearsAgo = datetime.now().year - 5 #computes last five years
#filter on the year of pubblication, group by Publisher_ID and count the
    number of book published
Book_DF.filter(col("Year") >= yearsAgo )\
    .groupBy(col("Publisher_ID")).count()\
    .filter(col("count") > 1 )\
    .withColumnRenamed("count","number of book published").show(truncate=
        False)
```

At the beginning the query computes the current year and subtract five to get five years ago. The first operation of the query is a filter on the year of publication of the books. In the second operation the query groups the remaining books by "Publisher_ID", thus it counts the number of books published by each publisher. In the third operation the query applies a filter on the counter of the books. Finally the query renames the column to clarify the output.

Publisher_ID	number of book published
22	2
77	2
126	2
31	2
87	2
51	2
28	2
120	2
85	2
100	2
83	2
102	2

Figure 2.79: Result of the query Q7

Q8: Books about the most popular topics

Like Articles, Books are characterized by a set of Tags, and each one of them represents a topic of the book. The goal of this query is to find the books that deal with at least one of the top 5 most used tags (among books only).
The query is divided into two parts: the first one is responsible of finding the most popular tags; the second one is responsible of finding the books that contain at least one of them.

```
#1. We explode the book table (for the tags)
exploded_books_for_tags = Book_DF\
                          .select(col("Book_ID"),explode(col("Tags")))\
                          .withColumnRenamed("col", "Tag")
#2. Then, we group by tags
aggregated_books = exploded_books_for_tags.groupBy("Tag")\
    .agg(count("Book_ID").alias("Number_Of_Books"))\
    .sort(col("Number_Of_Books").desc(),col("Tag").asc())\
    .limit(5)
aggregated_books.show(truncate=False)
```

The query starts by exploding the *Tags* field (that is an array). Then, the column that contains the exploded tags is renamed *"Tag"*. Now, *exploded_ books_for_ tags* is a dataframe with every row that corresponds to a single tag.
Then, the following part groups the rows by the values contained in the *Tag* column and the different IDs are counted. In this way, we know how many books contain a given tag. Then, the result of the aggregation is renamed *"Number_ Of_ Books"*, the rows are sorted (in descending order by *Number_ Of_ Books* and in lexicographical order by *Tag*) and then limited to 5 rows. Now we have the 5 most used tags. For completeness, a result of this sub-query is reported below.

```
+------------------------+---------------+
|Tag                     |Number_Of_Books|
+------------------------+---------------+
|Big Data Analytics      |14             |
|Semantic Web            |12             |
| Algorithm and complexity|11            |
|Cyber Security          |11             |
|Internet of Things (IoT) |11            |
+------------------------+---------------+
```

Figure 2.80: Top 5 most used tags among books

```
#3. Then, we make a list of tags
list_of_tags = aggregated_books.select("Tag").collect()
list_of_tags = [t[0] for t in list_of_tags]
#4. We extract a list of book IDs that have a tag in the list
book_ids = exploded_books_for_tags.filter(col("Tag").isin(list_of_tags))\
                                  .select("Book_ID").collect()
book_ids = [b[0] for b in book_ids]
#5. We display the books that match the previous conditions
Book_DF.filter(col("Book_ID").isin(book_ids))\
    .select(col("Book_Title"),col("ISBN"),col("Tags")).show(truncate=False)
```

The second part of the query looks for the books that are actually about the topics found at the previous step. In order to do this, we take advantage of the previously obtained dataframes:

- *list_of_tags* is a list that contains the most popular tags.
- *book_ids* is a list that contains the IDs of the books that are about the most popular tags. It is obtained by verifying that the value of the *Tag* column is in the *list_of_tags*.

At this point, we can use the original book dataframe to filter the rows: only the ones that have an ID contained in *book_ids* list are kept. Finally, only the *Book_Title*, *ISBN* and *Tags* column are shown.

```
+--------------------+----------------+--------------------+
|               Title|            ISBN|                Tags|
+--------------------+----------------+--------------------+
|Semantic Manageme...|978-3-8244-0003-4|[Internet of Thin...|
|Guide to Software...|978-3-519-02606-8|[Big Data Analyti...|
|Utilizing Problem...|978-3-929443-24-0|[Big Data Analyti...|
|Fundamentals of J...|    3-540-61426-5|      [Semantic Web]|
|Virtual Sociocult...|978-3-8440-2399-2|[Internet of Thin...|
|Software Reliabil...|978-3-519-00310-6|[Big Data Analyti...|
|Hybrid Soft Compu...|978-3-86073-159-8|[Semantic Web,  E...|
|Design Concepts f...|978-3-642-35358-1|[Internet of Thin...|
|From Snapshots to...|978-3-8244-2042-1|[Semantic Web,  A...|
|Error Detection a...|978-3-528-06494-5|[Cyber Security, ...|
|Internet of Thing...|978-3-486-22186-2|[Semantic Web,  L...|
|Modeling Companio...|978-3-7502-6538-7|[Web and Informat...|
|Shape Analysis an...|978-3-658-13262-0|[Big Data Analyti...|
|Handbook of Discr...|978-3-89936-625-9|[Internet of Thin...|
|Handbook of Finit...|978-3-8364-7734-5|[Semantic Web,  5...|
|Handbook of Datab...|978-3-8364-6360-7|[Internet of Thin...|
```

Figure 2.81: Result of the query Q8

Q9: Conference editions that have published two proceedings

The following question returns the list of conference editions held in a certain period and that have released exactly two proceedings.

```
#1. Filtering the edition in a certain period
filter_by_date = CE_DF.filter((CE_DF.Start_date >= "1999-01-01") & (CE_DF.
    Start_date <= "2010-12-31"))

#2. Joining the resulting dataframe with the Conference_Book DataFrame.
join_with_books = filter_by_date.join(CB_DF, filter_by_date.Conference_ID
    == CB_DF.ConferenceEdition_ID,"inner")

#3. Grouping for each edition, considering those that have released exactly
     two proceedings (having on aggregate operation).
group_for_edition = join_with_books.groupBy(join_with_books.Conference_ID,
    join_with_books.Name,join_with_books.Start_date,join_with_books.End_date
    )\
.agg(count(col("Conference_ID")).alias("ConferenceBook_Released")).filter(
    col("ConferenceBook_Released") == 2)

#4. Sorting in descending order by number of proceedings released.
group_for_edition.sort(col("ConferenceBook_Released").desc())\
            .select(col("Name"),col("Start_date"),col("End_date"),col("
                ConferenceBook_Released")).show(truncate=False)
```

The main operation consists in joining the dataframe of the editions held in the specified period and the dataframe of the individual books released by the editions, thus considering only the books released in that period.

Next, grouping for each conference edition, through the *count* function we calculate the number of proceedings released by that edition. To get the only editions that have released exactly two proceedings it is necessary to use a filter on the result of the *count* function. For completeness, a result of this sub-query is reported below.

Name	Start_date	End_date	ConferenceBook_Released
Workshop on Deep Learning Approaches for Low-Resource NLP	2000-07-31	2000-08-03	2
Network and Distributed System Security Symposium	2010-09-15	2010-09-18	2
Workshop on Computational Linguistics and Clinical Psychology	2002-07-13	2002-07-16	2
Simulation in Research and Development	2003-05-18	2003-05-21	2
Annual International Conference on New Technologies of Distributed Systems	2000-12-24	2000-12-27	2
Conference on Analysis of Neural Network Applications	2001-11-08	2001-11-11	2
Cooperative Information Systems International Conference	2003-07-06	2003-07-09	2
Parallel and Large-Scale Computers: Performance _ Architecture _ Applications IMACS World Congress	2008-02-11	2008-02-14	2
International Tyrrhenian Workshop TIWDC	2010-08-13	2010-08-16	2
Designing Interactive Systems	2009-02-16	2009-02-19	2
On The Move to Meaningful Internet Systems: CoopIS DOA and ODBASE	2003-01-02	2003-01-05	2
International Conference on Wearable Micro and Nano Technologies for Personalized Health	2004-05-25	2004-05-28	2
International Conference on Grid Computing & Applications GCA	2004-05-17	2004-05-20	2
International Conference on Protocol Engineering ICPE	2002-10-09	2002-10-12	2
International Conference on Wearable Micro and Nano Technologies for Personalized Health	2007-05-30	2007-06-02	2
Signal Processing and Communications Applications Conference SIU	2002-10-01	2002-10-04	2
Cooperative Information Systems International Conference	2000-09-18	2000-09-21	2
International Conference on Grid Computing & Applications GCA	2006-08-25	2006-08-28	2
International Workshop on Cognitive Wireless Networks CWNETS	2005-01-26	2005-01-29	2
Simpasio Brasileiro de Banco de Dados	2006-07-04	2006-07-07	2

Figure 2.82: Result of the query Q9

Q10: Best authors of articles about a given topic

The presented query returns the list of researchers that wrote a minimum number of articles about a given topic.

```
# Params
topic = "Machine Learning (ML)"
minimum_number_articles = 2

#1. joining researchers and writtend articles
double_join = Researcher_DF.join(Writes_Article_DF, Writes_Article_DF.
    Researcher_ID == Researcher_DF.Researcher_ID, "inner")\
                .join(Article_DF, Article_DF.Article_ID ==
                    Writes_Article_DF.Article_ID , "inner")
#2. Filter on the topic of the articles
filtered_by_subject = double_join.filter(array_contains(double_join.Tags,
    topic))

#3. Count the number of written articles per Researcher and keep only the
    tuples having count >= 2
filtered_by_subject.groupBy("Name", "ORCID").agg(
     count("*").alias("Number_of_Articles"))\
     .filter(col("Number_of_Articles") >= minimum_number_articles)\
     .sort(col("Number_of_Articles").desc())\
     .show(truncate=False)
```

Fist of all, two *joins* are performed in order to produce a single table containing a tuple for each researcher and written article. These tuples are filtered to keep only the ones associated to articles regarding the topic in which we are interested. The remaining tuples are then grouped on the researcher name field and the ORCID field and it is computed the number of articles written by each researcher. Finally only the researchers that wrote more than a specified number of articles are returned along with the actual number of publications. The tuples are sorted by number of publications in a descending order.

```
+---------------+------------------+------------------+
|Name           |ORCID             |Number_of_Articles|
+---------------+------------------+------------------+
|Veronica Penza |0104-1035-5434-6787|3                 |
|Marco Ciccone  |0103-1034-5433-6825|2                 |
|Matteo Giuliani|0105-1036-5435-6929|2                 |
|Luca Giulioni  |0104-1035-5434-6807|2                 |
|Bruno Di Giorgi|0104-1035-5434-6928|2                 |
|Ramona Cabiddu |0105-1036-5435-6826|2                 |
|Federico Nardi |0103-1034-5433-6786|2                 |
|Davide Tamborini|0105-1036-5435-6788|2                 |
|Amine Barkat   |0103-1034-5433-6806|2                 |
+---------------+------------------+------------------+
```

Figure 2.83: Q10 result

2.4.4. Commands

C1: Add new publishers

The command goal is to insert in the publisher dataframe new tuples.

```
#1. Write the new publishers to be added
publishersData = [(131,"A new publisher to add"),
                  (132,"An other publisher to add")]

#2. Convert the previous data in a dataframe using the publisher schema
publishersToAdd = spark.sparkContext.parallelize(publishersData).toDF(
    PublisherSchema)

#3. Performing the union between the two dataframes
updatedPublisherDB = Publisher_DF.union(publishersToAdd)
updatedPublisherDB.sort(col("Publisher_ID").desc()).show(truncate = False)
```

First we define the new tuples in the publishersData list: a publisher in simply characterized by an ID and a name. In order to add these two new tuples to the main dataframe we have to convert the tuples in a dataframe called publishersToAdd and then thanks to the union operation we can merge the two dataframes. To show the effect of the query we sort the publisher dataset in a descending order to check wheter the new tuples have been inserted or not.

```
+------------+--------------------------------+
|Publisher_ID|Name                            |
+------------+--------------------------------+
|132         |An other publisher to add       |
|131         |A new publisher to add          |
|130         |IEEE Computer Society           |
|129         |Springer International Publishing|
|128         |Univ.-Verlag der TU             |
|127         |Nomos-Verlag-Ges.               |
|126         |Wissenschaftsverlag Mainz       |
|125         |Georg Olms Verlag               |
|124         |Wiss. Berichtswesen der DLR     |
|123         |Addison-Wesley                  |
|122         |Schweikert                      |
|121         |Westdt. Verlag                  |
|120         |Utz, Wiss.                      |
|119         |Intemann                        |
|118         |Verkehrsverlag Fischer          |
|117         |Knirsch                         |
|116         |J.B. Metzler                    |
|115         |Waxmann                         |
|114         |TEWISS Verlag                   |
|113         |wbg Academic                    |
+------------+--------------------------------+
only showing top 20 rows
```

Figure 2.84: Result of the command C1

C2: Fixing Books without a DOI

As mentioned previously, the dataset that we used for Spark is mainly derived from the one used for the Neo4j delivery. Thus, as stated in the section 2.2.2, a fixed DOI code was given to Documents (Books and Articles) that in the DBLP dump were without a DOI. In Spark, we can directly assign a *null* value to the field. Thus, the following query is used to replace those fixed values with a *null* one. Since in our dataset there are no Articles with a fixed DOI, the operation is executed on Books only.

```
Book_DF.filter(col("DOI") == "https://doi.org/0")\
        .select(col("Book_ID"),col("Title"),col("DOI")).show()
#Modifying the DOI column
Book_DF = Book_DF.\
            withColumn("DOI",when(col("DOI")=="https://doi.org/0" ,None).
                otherwise(col("DOI")))
Book_DF.select(col("Book_ID"),col("Title"),col("DOI")).show()
```

The first line is used to display the books with the fixed DOI. The core of the command is the second line. We start from the *Book_DF* (the dataframe that contains books) and we apply the *withColumn* operator. This takes two parameters: the name of the column to replace ("DOI") and the new column. The new column is generated with *when*: for each row, if the *DOI* field is equal to "https://doi.org/0", the value is replaced with *None*, otherwise the value is not modified. The last line is used to show the new *Book_DF* dataframe. We can see that the book with *Book_ID=15* is one of those that had the fixed DOI. Now the value is replaced by *null*.

Book_ID	Title	DOI
15	Martin Davis on C...	https://doi.org/0
18	Self Aware Securi...	https://doi.org/0
20	Input/Output in P...	https://doi.org/0
21	Handbook on Ontol...	https://doi.org/0
22	Semantic Acquisit...	https://doi.org/0
23	Utilizing Problem...	https://doi.org/0
25	Security, Privacy...	https://doi.org/0
35	Software Reliabil...	https://doi.org/0
42	The Logic of Info...	https://doi.org/0
47	Hybrid Soft Compu...	https://doi.org/0
65	Measuring Systemi...	https://doi.org/0
68	Carl Adam Petri -...	https://doi.org/0
75	Fourier Analysis ...	https://doi.org/0
82	Handbook of Energ...	https://doi.org/0
97	Handbook of Natur...	https://doi.org/0
105	Encyclopedia of B...	https://doi.org/0
106	Encyclopedia of B...	https://doi.org/0
107	Encyclopedia of B...	https://doi.org/0
108	Handbook of Spati...	https://doi.org/0
110	Handbook of Ambie...	https://doi.org/0

Book_ID	Title	DOI
1	The Semantic Web:...	https://doi.org/1...
2	Semantic Manageme...	https://doi.org/1...
3	Social Networks a...	https://doi.org/1...
4	Semantic Web Serv...	https://doi.org/1...
5	Test and Analysis...	https://doi.org/1...
6	Nonsequential and...	https://doi.org/1...
7	Situation Awarene...	https://doi.org/1...
8	Dynamische nicht-...	https://doi.org/1...
9	Guide to Software...	https://doi.org/1...
10	SQL Server Databa...	https://doi.org/1...
11	Reconfigurable Ac...	https://doi.org/1...
12	Security in E-Lea...	https://doi.org/1...
13	Modelling and Con...	http://www.crcnet...
14	Digital Humanism ...	https://doi.org/1...
15	Martin Davis on C...	null
16	Handbook of Big D...	https://doi.org/1...
17	Ada 2012 Referenc...	https://doi.org/1...
18	Self Aware Securi...	null
19	Simplicity is Com...	https://doi.org/1...
20	Input/Output in P...	null

Figure 2.85: Books with a fixed DOI before and after the command C2.

C3: Delete a given publisher

This command goal is to delete a certain publisher, given his name, and thus to delete all books published by this publisher. In this example the given name is "IEEE Computer Society"

```
#retrieve Publisher_ID
publisherID_toDelete=Publisher_DF.filter(col("Name") == "IEEE Computer
    Society").select(col("Publisher_ID")).collect()[0][0]

#shows the books published by this publisher
Book_DF.filter(col("Publisher_ID") == publisherID_toDelete).show(truncate=
    True)
#filter out these books
Book_DF = Book_DF.filter(col("Publisher_ID") != publisherID_toDelete)
#show dataframe after the delete
Book_DF.filter(col("Publisher_ID") == publisherID_toDelete).show(truncate=
    True)

#show publisher dataframe before the delete
Publisher_DF.filter(col("Name") == "IEEE Computer Society").show(truncate=
    False)
#filter out publisher
Publisher_DF = Publisher_DF.filter(col("Name") != "IEEE Computer Society")
#show publisher dataframe after the delete
Publisher_DF.filter(col("Name") == "IEEE Computer Society").show(truncate=
    False)
```

First the command retrieves the ID of the publisher. Then it proceeds to filter out the book published by this publisher showing the books data-frame before and after the delete. Last the command filters out the publisher from publisher data-frame, showing the publisher data-frame before and after the delete.

```
+-------+--------------------+--------------------+----+----------------+--------------------+------------+
|Book_ID|               Title|                 DOI|Year|            ISBN|                Tags|Publisher_ID|
+-------+--------------------+--------------------+----+----------------+--------------------+------------+
|    130|Baltic Computer S...|https://doi.org/1...|2015|978-3-8364-4694-5|[Parallel Computi...|         130|
|    132|Ausgezeichnete In...|                null|2007|978-3-940317-55-1|                null|         130|
+-------+--------------------+--------------------+----+----------------+--------------------+------------+

+-------+-----+---+----+----+----+------------+
|Book_ID|Title|DOI|Year|ISBN|Tags|Publisher_ID|
+-------+-----+---+----+----+----+------------+
+-------+-----+---+----+----+----+------------+

+------------+--------------------+
|Publisher_ID|Name                |
+------------+--------------------+
|130         |IEEE Computer Society|
+------------+--------------------+

+------------+----+
|Publisher_ID|Name|
+------------+----+
+------------+----+
```

Figure 2.86: Before and after the delete

C4: Append a new tag to an existing document

The following command allows to add a new tag to an article given its ID.

The parameters that we need to give in input to the command are shown at the beginning of this code snippet. The command itself is preceded and followed by two commands for showing the tags associated to the article before ad after the insertion of the new tag.

```
#Params
article_ID = "2"
new_tag = "SMBUD"
# article_ID = "3" for the article having no tags

#Show tags before update
article_tags = Article_DF.filter(Article_DF.Article_ID == article_ID).
    select("Article_ID", "Tags")
article_tags.show(truncate=False)


#C4: Add new tag to article
Article_DF = Article_DF.withColumn(
    "Tags",
    F.when((col("Article_ID") == article_ID) & (col("Tags").isNotNull()),
              F.array_union(col("Tags"), F.array(F.lit(new_tag))))
     .when((col("Article_ID") == article_ID) & (col("Tags").isNull()),
              F.array(F.lit(new_tag)))
     .otherwise(col("Tags"))
)

#Show tags after update
article_tags = Article_DF.filter(Article_DF.Article_ID == article_ID).
    select("Article_ID", "Tags")
article_tags.show(truncate=False)
```

The command needs to distinguish between two cases. In the first case the column *Tags* is empty (null), while in the second instance the column contains an array with at least one element. When the *Tags* column is null we first need to create an array, using the method *array()* and then assign it to the column. Instead, when the column already contains an array we use the function *array_union()* to append the new tag to the existing vector.

The *lit()* function produces a literal that is required as input for the *array()* and the *array_union()* functions.

The selection of the desired article is done by means of the *when()* function which allows

to specify multiple conditions to discriminate the tuples on which to execute a given operation.

The command overwrites the Tags column hence it is necessary to specify, in the otherwise clause, to report the tags as they are for the documents that do not satisfy the condition specified in the *when()* method.

The following pictures show the tags of two articles before and after the insertion of he new one. In the first image the article already had one tag assigned while in the second picture the article had no tags.

```
+----------+-----------------+
|Article_ID|Tags             |
+----------+-----------------+
|2         |[Virtual Reality]|
+----------+-----------------+

+----------+------------------------+
|Article_ID|Tags                    |
+----------+------------------------+
|2         |[Virtual Reality, SMBUD]|
+----------+------------------------+
```

Figure 2.87: Adding a tag to an article with other tags

```
+----------+----+
|Article_ID|Tags|
+----------+----+
|3         |null|
+----------+----+

+----------+-------+
|Article_ID|Tags   |
+----------+-------+
|3         |[SMBUD]|
+----------+-------+
```

Figure 2.88: Adding a tag to an article without tags

C5:Reschedule a conference edition

This command allows to update *country*, *city*, *street*, *start_ date* and *end_ date* of a conference edition given by id. The parameters give in input are: the *Conference_ ID* of the conference edition to update, the new *country*, the new *city*, the new *street*, the new *start_ date* and the new *end_ date*.

```
from datetime import datetime
#reschedule of an edition of a conference
CE_id=2
#show CE before update
CE_DF.filter( col("Conference_ID") == CE_id).show()
country_new='Italy'
city_new='Milan'
street_new='Zara'
start_date_new = datetime.strptime('2002-09-13', '%Y-%m-%d').date()
end_date_new =datetime.strptime('2002-09-16', '%Y-%m-%d').date()

#retrieve name of the CE we want to reschedule
name=CE_DF.filter( col("Conference_ID") == CE_id).select( col("Name") ).
    collect()[0][0]

#remove old CE
CE_DF=CE_DF.filter( col("Conference_ID") != CE_id)
CE_DF.filter( col("Conference_ID") == CE_id).show()

#create a new CE
newCE_data=[(CE_id,name,country_new,city_new,street_new,start_date_new,
    end_date_new)]
newCE=spark.sparkContext.parallelize(newCE_data).toDF(CESchema)

#union
CE_DF=CE_DF.union(newCE)

#show updated CE
CE_DF.filter( col("Conference_ID") == CE_id).show()
```

In the beginning the command shows the given conference edition before the update. Then it retrieves the name of the conference edition that we want to update, thus it removes this conference edition from the data-frame and shows that there is no longer in the data-frame. It proceeds building a new data-frame composed only by the updated version of the conference edition. Finally it adds the updated conference edition to the original data-frame using the command *union()* and shows the new version of the conference edition.

```
+-------------+--------------------+--------+-------------+---------+----------+----------+
|Conference_ID|                Name| Country|         City|   Street|Start_date|  End_date|
+-------------+--------------------+--------+-------------+---------+----------+----------+
|            2|Designing Interac...|Honduras|Ocote Paulino|Evergreen|2002-09-12|2002-09-15|
+-------------+--------------------+--------+-------------+---------+----------+----------+

+-------------+----+-------+----+------+----------+--------+
|Conference_ID|Name|Country|City|Street|Start_date|End_date|
+-------------+----+-------+----+------+----------+--------+
+-------------+----+-------+----+------+----------+--------+

+-------------+--------------------+-------+-----+------+----------+----------+
|Conference_ID|                Name|Country| City|Street|Start_date|  End_date|
+-------------+--------------------+-------+-----+------+----------+----------+
|            2|Designing Interac...|  Italy|Milan|  Zara|2002-09-13|2002-09-16|
+-------------+--------------------+-------+-----+------+----------+----------+
```

Figure 2.89: Result of C5

List of Figures