POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Systems and Methods for Big and Unstructured Data Project

Author(s):**Fabio Lusha - 10882532**

Enrico Simionato - 10698193

Irfan Cela - 10694934

Bianca C. Savoiu Marinas - 10684465

Alberto Sandri - 10698469

Group Number:**31**

Academic Year: 2022-2023

Contents

Part I

Introduction

1 | Introduction to the problem

1.1. Problem description

Scientific research is the engine which leads us through the future providing the technology to improve every single detail of our life. Nowadays, the speed at which humanity and technology advance is outstanding thanks to scientists progress in their fields of study. The reaching of a new result in the researches materializes through scientific articles. Since the advent of the World Wide Web, publishers started to post their papers on the net reaching an audience far larger than before. This has allowed a more convenient and faster way to share knowledge, therefore boosting academic research and consequently the publishing of new papers. Researchers need to navigate and access this massive quantity of knowledge, which is growing every day, thus, if we want to bolster human progress, new methods in organizing and managing these data are mandatory.

1.2. Purpose

The purpose of the project is to create a bibliographic database, namely a system able to store and manage data regarding scientific articles and all their meaningful characteristics and relations with authors and the place where they are published.

1.3. Assumptions and important concepts

The problem we are addressing is very wide, and given the heterogeneity of the data different assumptions can be made. We shaped our database taking in consideration the following aspects:

- A scientific paper can be written by one or more authors;
- All the authors of a paper are considered to be at the same level, so the position of coauthor is not considered;
- The papers the system considers were published on journals, books or were presented

at conferences;

- A paper can appear in just one publication among journals, books and conferences;
- A single affiliation of an author can be registered when a paper is written;
- Papers can have multiple fields of study (fos) and keywords;
- Not every paper has to be related to a field of study;
- Not every paper has to have related keywords;
- The name of the author may be complete or abbreviated;
- A paper may or may not reference other papers or be referenced by others.

2 | ER model

As a bibliographic database the main focus is on scientific articles, also called scientific papers, that can be considered the central entity. Every article is written by one or more authors and each writer is associated to an organization. Articles have several features like title, year of publication, language, abstract, DOI, URL and can have many keywords and can be related to many fields of study. Furthermore, a paper can reference other papers and each article is published on a book, on a journal or is presented during a conference.

The place where an article is published is called in a general way *publication* and each one of them has different properties: conferences take place in a specific location, books have an ISBN and a publisher and journals have an ISSN, volume, issue and a publisher. So the `Publication` entity is extended with a hierarchy that is total and exclusive: a node has to be instantiated as one of the specific types of publication and cannot be more than one type at the time.

The group opted for a straightforward model with only the core concepts and relationships in order to make it easily understandable and represent it in a concise way. Of course the problem domain is very wide and complex but in this modeling phase it was taken into account also the available data to build the model, trying to leave it quite general so that it would be easier to modify it and make it more complex in further versions.

In the following ER model we chose to represent the cardinalities of the relationships always starting from one, so we don't consider entities that are not related to others. When adding data in the database we can have in an initial state some isolated nodes, but in the ER we show a stable and correct state of the system.

As we can see from the diagram, we use in general the attribute `id` as identifier of the entities, but in some cases in the real world, there are better options, for instance the `Book` might be identified by the attribute `ISBN`. We made this assumption because our dataset was not complete, and a lot of `Book` entities didn't have this property. The same reasoning can be applied for `Paper` with `DOI`, `Journal` with `ISSN`, and `Author` with `ORCiD` that we totally disregarded because not present in the used data file.

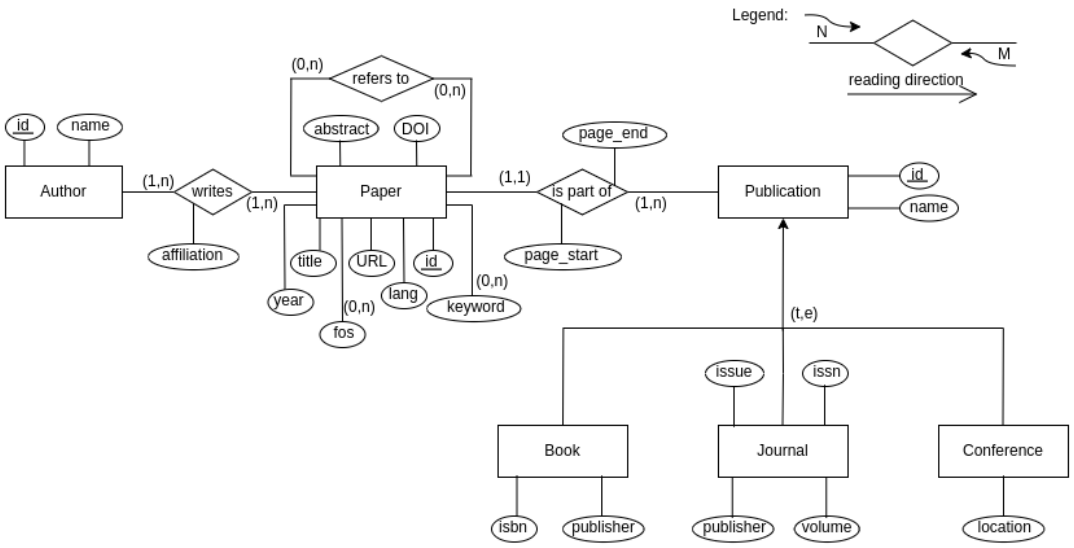


Figure 2.1: ER model of the bibliographic database.

More detailed description of entities

- **Paper**, as said before, is the most important entity indeed it is in the middle of the model. Each paper is characterized by an ID, a title, a publication year, an abstract, a list of keywords, a list of fields of study, a DOI which is a unique global identifier and a URL to reach the site where it is available for consultation.
- **Author** of a paper is identified by an ID and has a name, that contains both the name and the surname of the author.
- **Publication** is a total and exclusive hierarchy that represents the physical place where a paper is published, it has an ID and a name called venue, and it is specialized in:
  - **Conference** that has a location where it took place;
  - **Book** that has a global identifier called ISBN and a publisher;
  - **Journal** that has a global identifier called ISSN, a publisher, a volume and issue.

More detailed description of relationships

- **WRITES** links Author and Paper, each author can write one or more papers and each paper is written by one or more authors. This relationship has an attribute

affiliation that is the organization the author was part of when the article was written.

- **REFERS TO** links two Paper entities, each paper can reference many papers and can be referenced by many papers.
- **IS PART OF** links Paper and Publication, each paper is published on a single publication and each publication can contain several papers. This relationship has as attributes the page_start and page_end of the article within the publication.

More detailed observations

- DOI could have been a good identifier for papers since it is globally unique, but in the used dataset not all articles had it.
- ORCiD is a unique identifier for authors that would have been a valid key for the entity Author but this data was not present in the used dataset.
- Since the focus is on the entity Paper and a simple model for the domain was built, it seemed not meaningful to have a separate entity to model the organization whose author is part of. In addition, that would have had just one attribute. For this reason the concept of affiliation was modeled as an attribute of the relationship `WRITES`, allowing to store multiple affiliations for the single author.

Part II

Neo4j

3 | Dataset description

The main dataset used can be downloaded from the site www.aminer.org, in particular the latest version of data was employed, namely DBLP-Citation network V13. The dataset was not entirely imported in Neo4j because it is really huge and our machines with limited computing power where not able to handle it, so just a part of it was taken, containing information about 2315 papers.

The following table shows the dataset's fields that were used, associated to their respective meaning, name and role in the created database. Actually, the fields publisher and location were not present in the original dataset, that's why we created these information from scratch, but this aspect is further discussed in the following chapter.

Field name	Type	Description	Name in Neo4j	Usage in Neo4j
_id	String	paper ID	id	node Paper
title	String	paper title	title	node Paper
year	Integer	paper year	year	node Paper
lang	String	paper language	lang	node Paper
doi	String	paper DOI	doi	node Paper
url	String	paper URL	url	node Paper
abstract	String	paper abstract	abstract	node Paper
keywords[i]	String	paper keyword	keyword	node Keyword
fos[i]	String	paper field of study	fos	node Fos
references[i]	String	paper reference	REFERENCES	relation REFERENCES
page_start	Integer	start page	page_start	attribute IS_PART_OF
page_end	Integer	end page	page_end	attribute IS_PART_OF
authors._id	String	author ID	id	node Author
authors.name	String	author name	name	node Author
authors.org	String	author affiliation	affiliation	attribute WRITES
venue.raw	String	publication name	name	node Publication
isbn	String	book ISBN	isbn	node Book
publisher	String	publisher name	publisher	node Book/Journal
issn	String	journal ISSN	issn	node Journal
volume	String	journal volume	volume	node Journal
issue	String	journal issue	issue	node Journal
location	String	conference location	location	node Conference

Table 3.1: Description of the data used in the project and relationship between original dataset and imported data.

4 | Data upload

4.1. Pre-processing

Before uploading the data in Neo4j, the dataset was downloaded from the site mentioned above, and just a part of it has been extracted. Then some ad-hoc Python scripts have been used to do a first-level cleaning of the data since we found some inconsistency in it. Specifically the cleaning involved the following aspects:

- Some numerical attributes where wrapped in NumberInt(...) string. This is not a standard way to store numeric values in JSON so it couldn't be parsed by the Neo4j JSON parser. We unwrapped all numeric values stored this way, leaving only the actual value;
- Some of the ISSN attributes did not conform to the standard and correct pattern, so the wrong ones have been eliminated;

After that, to distinguish the type of publication, a Python script was made to infer it based on the attributes of the paper: if it has an ISBN is a Book, if it has an ISSN, volume or issue is a Journal otherwise it is a Conference.

In order to have a more complete domain it has been decided to add the fields publisher to Book and Journal, and location to Conference. These values were taken from the web and added randomly to the papers using a Python script.

Lastly, since just a subset of the dataset has been used, there were very few connections among the nodes, so, thanks to another Python scripts, some values were added randomly to the fields keyword, fos and references of each paper such that the graph is more connected and more meaningful queries can be performed.

4.2. Upload process

The JSON file called `dataset.json` was used in order to upload the data, it was put in the `import` folder of Neo4j and then with the following commands the data were imported

using cypher-shell.

To be sure that at the beginning the database is empty, all possible parameters, nodes and relationships are deleted with this command.

```
:params {};
MATCH(n) DETACH DELETE n;
```

In order to use the importing commands it is first necessary to install the `apoc` library and emulate the following steps in order to be able to use the `apoc.load.json` with which we read the dataset file. For security reasons, procedures that use internal APIs are disabled by default. They can be enabled by specifying config in `NEO4J_HOME/conf/neo4j.conf` e.g. `dbms.security.procedures.unrestricted=apoc.*`.

You can also whitelist procedures and functions in general to be loaded using

`dbms.security.procedures.whitelist=apoc.coll.*,apoc.load.*`

We are particularly interested in the last one.

1. **Create Paper nodes with their attributes, create Fos and Keyword nodes** The following command creates all nodes with label `Paper`, setting the attributes `id`, `title`, `year`, `lang`, `doi`, `url`, `abstract`; then it creates the nodes with label `Keyword` requiring that the value is not null, moreover `Paper` and `Keyword` are linked with the relationship `HIGHLIGHTS`. The same thing is done for `Fos` nodes, that are linked with `Paper` nodes through `BELONGS_TO` relationship.

```
CAll apoc.load.json('dataset.json') YIELD value
CREATE (p:Paper {id:value._id, title:value.title, year:value.year,
    lang:value.lang, doi:value.doi, url:value.url, abstract:value.
    abstract})
WITH p, value
UNWIND value.keywords AS kw
WITH p, value, kw WHERE kw IS NOT NULL
MERGE (k:Keyword {keyword:kw})
MERGE (p)-[h:HIGHLIGHTS]->(k)
WITH p, value
UNWIND value.fos AS fos
WITH p, value, fos WHERE fos IS NOT NULL
MATCH (p:Paper {id:value._id})
MERGE (f:Fos {fos:fos})
MERGE (p)-[b:BELONGS_TO]->(f)

```

2. **Create Author nodes and the relationship WRITES**

The following command creates all nodes with label `Author` requiring that `id` and `name` attributes are not null. Next all `Author` nodes are linked with their `Paper` nodes using the `WRITES` relationship, assigning the property `affiliation` when this information is present.

```
CALL apoc.load.json('dataset.json') YIELD value
UNWIND value.authors AS aut
WITH aut, value
WHERE aut._id IS NOT NULL AND aut.name IS NOT NULL
MERGE (a:Author {id:aut._id, name:aut.name})
WITH a, aut, value
MATCH (p:Paper {id:value._id})
CREATE (a)-[w:WRITES {affiliation:aut.org}]->(p)

```

3. **Create Book nodes**

   The nodes with label `Book` are created, setting also the label `Publication`, and this entity has the attributes `name` and `publisher`. The command also links `Paper` nodes to `Book` nodes using the `IS_PART_OF` relationship, adding to it the properties `page_start` and `page_end`. We also allow these attributes to be null, so we don't check their presence when adding them, because in graph databases we can have heterogeneity without any problem and this permits us to explore the peculiarity and strength of this technology.

```
CALL apoc.load.json('dataset.json') YIELD value
UNWIND value.venue AS ven
WITH value, ven
WHERE value.publication_type = "Book"
CREATE (b:Publication:Book {isbn:value.isbn, publisher:value.
    publisher, name:ven.raw})
WITH b, value
MATCH (p:Paper {id:value._id})
CREATE (p)-[i:IS_PART_OF {page_start:toInteger(value.page_start),
    page_end:toInteger(value.page_end)}]->(b)

```

4. **Create Conference nodes**

   The nodes with label `Conference` are created, setting also the label `Publication`, and `name` and `location` attributes of the conference. Then `Paper` nodes are linked with `Conference` nodes using the `IS_PART_OF` relationship setting the properties `page_start` and `page_end`.

```
CALL apoc.load.json('dataset.json') YIELD value
UNWIND value.venue AS ven
```

```
WITH value, ven
WHERE value.publication_type = "Conference"
CREATE (c:Publication:Conference {name:ven.raw, location:value.
    location})
WITH c, value
MATCH (p:Paper {id:value._id})
CREATE (p)-[i:IS_PART_OF {page_start:toInteger(value.page_start),
    page_end:toInteger(value.page_end)}]->(c)

```

5. **Create Journal nodes**

   The nodes with label `Journal` are created, setting also the label `Publication` and its properties `name`, `publisher`, `issn`, `volume` and `issue`. Then `Paper` nodes are linked with `Journal` nodes using the `IS_PART_OF` relationship setting the properties `page_start` and `page_end`.

```
CALL apoc.load.json('dataset.json') YIELD value
UNWIND value.venue AS ven
WITH value, ven
WHERE value.publication_type = "Journal"
CREATE (j:Publication:Journal {issn:value.issn, publisher:value.
    publisher, name:ven.raw, volume:value.volume, issue:value.issue
    })
WITH j, value
MATCH (p:Paper {id:value._id})
CREATE (p)-[i:IS_PART_OF {page_start:toInteger(value.page_start),
    page_end:toInteger(value.page_end)}]->(j)

```

6. **Create REFERENCES relationship**

   With the following command the `REFERENCES` relationship between `Paper` nodes is created.

```
CALL apoc.load.json('dataset.json') YIELD value
UNWIND value.references AS ref
WITH value,ref
WHERE ref IS NOT NULL
MATCH (a:Paper {id:value._id})
MATCH (b:Paper {id:ref})
MERGE (a)-[r:REFERENCES]->(b)

```

Label	Quantity
Paper	2315
Author	3546
Fos	155
Keyword	4343
Book	372
Conference	612
Journal	1315
Total	12658

Table 4.1: Summary with node quantity for each label.

5 | Graph diagram

In the realization process from the ER model to the graph diagram some changes have been made to better exploit the features of Neo4j, prioritizing connections between data. For this reason nodes with the following labels are present in the graph:

- **Paper** as it is the center entity, has a lot of connections to the other nodes, and its possible attributes are `id`, `title`, `year`, `lang`, `doi`, `url`, and `abstract`;
- **Author** has `id` and `name` as attributes;
- **Keyword** contains a single attribute `keyword`;
- **Fos** contains a single attribute `fos` that represent the field of study;
- **Publication** has the attribute `name` and it can also be associated to one of the following labels:
  - **Book** contains `isbn` and `publisher`;
  - **Conference** contains a `location`;
  - **Journal** contains `publisher`, `issn`, `volume`, `issue`.

Concerning the relationships:

- **REFERENCES** links a `Paper` that references another `Paper`, the connection is made by using the papers' `id`;
- **WRITES** links an `Author` to a `Paper`, it could also contains the `affiliation` attribute;
- **HIGHLIGHTS** links a `Paper` to a `Keyword`;
- **BELONGS_TO** links a `Paper` to a `Fos`;
- **IS_PART_OF** links a `Paper` to a `Publication`, it could also contain the attributes `page_start` and `page_end`.

5.1. Differences with respect to the ER model

`Keyword` and `Fos` were multi-valued attributes in the ER diagram and have been mapped as nodes in the graph. This choice was done since these values are shared among multiple `Paper` nodes creating many connections. In addition, the efficiency increases because when a query is performed just the record files about nodes and relationships need to be uploaded in main memory instead of the file with the properties, that would have been required if these fields were treated as attributes of the `Paper` node; so the file with the properties is not needed when querying this elements and this allows to use the main feature of Neo4j, which is the faster retrieval of data using relationships between nodes, that are a natural embedded structure of the graph databases.

In order to also take advantage of the fact that multiple labels can be assigned to a single node, the nodes with labels `Book`, `Journal` or `Conference` all have the label `Publication` too. This allows to easily perform queries involving a generic publication and also to further extend the database with new types of publications, such as thesis.

When importing the data in the graph database we encounter some differences from the ER diagram, because we don't always have the attributes that were assumed in the modelling part of the database. While importing the data, for some elements we checked if their properties were null before adding them, thus, in Neo4j we don't have those attributes at all. Otherwise, even if an element can be considered null, because for instance it has an empty string, it will be added anyway to the respective node. We decided to avoid further checks on these attributes in order to have a faster importing process.

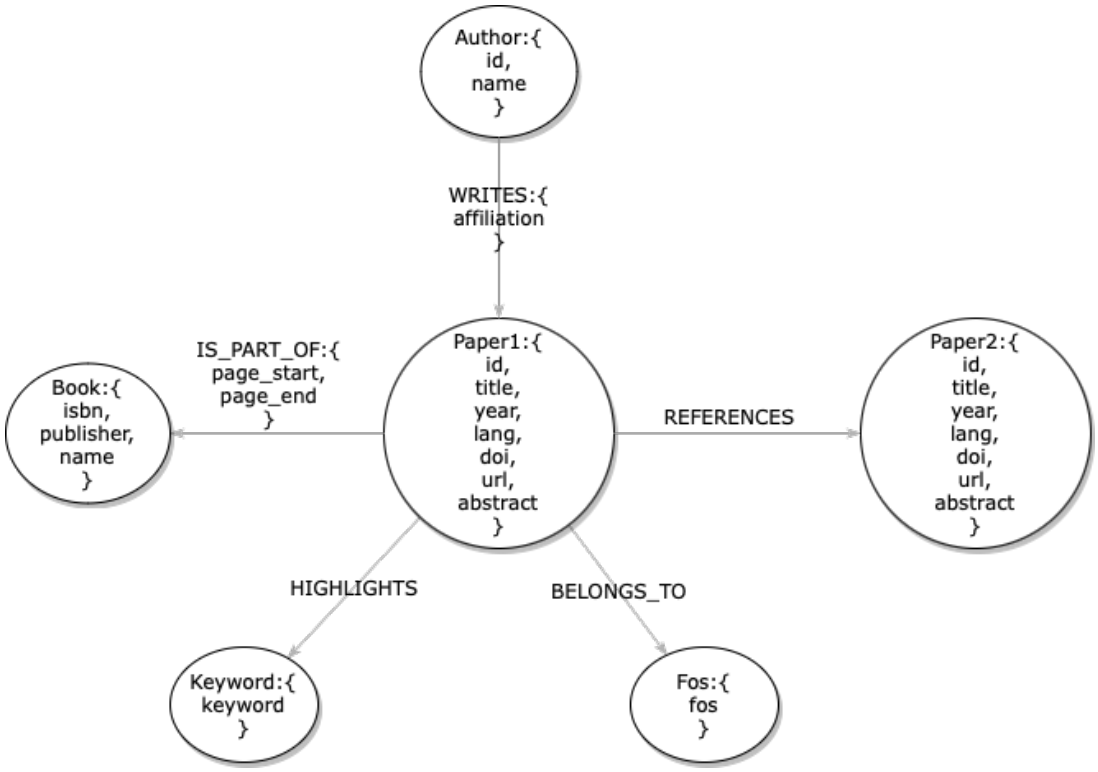


Figure 5.1: Graph diagram of a Paper that references another one, it has a single Author, a Keyword, a Fos and it is part of a Book.

6 | Commands and queries

The following commands and queries are written thinking principally on the way the database will be used. We think an user with the necessity to upload or search for something in the system, will be able to do that with what we provide. Longer and more difficult commands and queries can be written but we want to present only the ones that we find more meaningful.

We choose to use parameters instead of writing fixed values inside the queries in order to make code as flexible as possible. For this reason the commands and queries can be also used changing the value of the parameters and obtaining the results of interest. To execute the following code is required to run first the parameters regarding the command or query we want to perform and then the command or query itself.

6.1. Commands

The following commands are provided to make possible the update of the data. Each command can be easily modified to update parts of the database with specific features, for example it is possible to use the queries of the next section to update the obtained nodes and their attributes.

1. **Create a node, its attributes and some relationships**
   The command creates a set of new nodes. In particular the aim of the query is to add a new paper to the database, whose author is already present within the system, and also set its properties. Here we add a new paper, setting also its attributes, and we create the nodes of the keywords and the fields of study of the paper, if needed. The `MERGE` command is what it is used to create nodes and relationships checking also whether the new instances are already present within the database. This command allow us to avoid the creation of many copies of the same entity. The following command after adding the specified nodes creates also the relationships between them: the ones that relate the paper to its author, to its keywords and to the specific fields of study.

Parameters

```
:param auth_id => "53f463b3dabfaee4dc8430d5";
:param auth_name => "Annabel Sebag";
:param affiliation => "53f463b3dabfaee4dc8430d5";

:param doi => "10.1145/1596685.1596829";
:param paper_id => "53e997cbb7602d9701fbcee3";
:param paper_title => "Yankee gal";
:param year => 2009;
:param lang => "en";
:param page_start => 155;
:param page_end => 155;
:param url => ["https://dx.doi.org/10.1145/1596685.1596829","https
   ://doi.acm.org/10.1145/1596685.1596829","db/conf/siggraph/
   siggraph2009festival.html#Sebag09c","https://doi.org
   /10.1145/1596685.1596829"];
:param abstract => "This is a graduate film from the students of
   Supinfocom Valenciennes.";

:param k1 => "yankee gal";
:param k2 => "supinfocom valenciennes";
:param k3 => "graduate film";
:param fos1 => "animation";
:param fos2 => "siggraph";

```

Command

```
MATCH (a:Author {id:$auth_id, name: $auth_name})
MERGE (new_p:Paper
{doi:$doi,
id:$paper_id,
title:$paper_title,
year:$year,
lang:$lang,
page_start:$page_start,
page_end:$page_end,
url:$url,
abstract:$abstract})
MERGE aff = (a)-[:WRITES {affiliation:$affiliation}]->(new_p)
MERGE f1 = (new_p)-[:BELONGS_TO]->(:Fos {fos: $fos1})
MERGE f2 = (new_p)-[:BELONGS_TO]->(:Fos {fos: $fos2})
MERGE k1 = (new_p)-[:HIGHLIGHTS]->(kw1:Keyword {keyword:$k1})
MERGE k2 = (new_p)-[:HIGHLIGHTS]->(kw2:Keyword {keyword:$k2})
MERGE k3 = (new_p)-[:HIGHLIGHTS]->(kw3:Keyword {keyword:$k3})
```

```
RETURN aff, f1, f2, k1, k2, k3

```

2. **Create the relationship between existing nodes and set its properties**
   The command creates a new relationship between already existing nodes. In particular here is created the relationship which states that the author `"James Ostell"` has written the paper titled `"Grow"` when he was affiliated to organization `"Federal Institute of Technology, Switzerland"`. For a better identification of the book and for avoiding linking all the books having the same title we could have used the identifier of the book instead of the title. We used `MERGE` so if the relationship already exists with the specified affiliation it won't be added.

   Parameters

```
:param name => "James Ostell";
:param title => "Grow";
:param affiliation => "Federal Institute of Technology, Switzerland
    ";

```

   Command

```
MATCH (a:Author {name:$name}), (np:Paper {title:$title})
MERGE res = (a)-[:WRITES {affiliation:$affiliation}]->(np)
RETURN res;

```

3. **Set a new label for a node**
   We want to modify the properties of an already existing node. In particular we want to add one label to a specific node. At start we have new entries that are sparse data and we don't know the exact type and source so we want to infer it, in particular, we check if they have the `isbn` and in that case we update the data, setting the nodes as books. We do this by controlling the papers and using the `SET` keyword to update the properties, and if there are some misaligned data already in the graph this corrects that too. In the command, we can return the nodes in order to be sure that the information was updated correctly. We could do the same for the relationships, and we could also add more than one label.

```
MATCH (p:Publication)
WHERE p.isbn IS NOT NULL
SET p:Book
RETURN p;

```

4. **Update attributes of a relationship (analogue for nodes)**
   The next command updates the values of the attributes of a specific relationship. In particular, the number of the starting and ending page of the paper with id `53e99785b7602d9701f40556` are set through the keyword `SET`. If the user makes a mistake while inserting the properties of the relationship `IS_PART_OF`, let's say the pages, the user can always modify afterwards the entries with the `SET` operation. With the same syntax, if new properties come up or something was missed when adding nodes or relationships it is possible to create new attributes and set them to a specific value.

   Parameters

```
:param id => "53e99785b7602d9701f40556";

```

   Command

```
MATCH (paper:Paper)-[i:IS_PART_OF]->(:Book)
WHERE paper.id = $id
SET i.page_start = 15
SET i.page_end = 33

```

5. **Remove attributes of a relationship (analogue for nodes)**
   The following commands remove some attributes from a relationship. In particular, we remove the attributes concerning the starting and ending numbers of pages within a book when this numbers are negative, and moreover, we remove these numbers when the index of the starting page is greater than the index of the ending page because it doesn't make sense to have such data. This can be seen as some sort of data cleaning, useful to remove incorrect data.

```
MATCH (:Paper)-[i:IS_PART_OF]->(:Book)
WHERE i.page_start < 0
REMOVE i.page_start

```

```
MATCH (:Paper)-[i:IS_PART_OF]->(:Book)
WHERE i.page_end < 0
REMOVE i.page_end

```

```
MATCH (:Paper)-[i:IS_PART_OF]->(:Book)
WHERE NOT i.page_start IS NULL AND NOT i.page_end IS NULL AND i.
   page_end - i.page_start < 0
```

```
REMOVE i.page_start, i.page_end

```

6. **Delete a node**
   With the following command we delete the node `Book` titled `"Mathematics and Computers in Simulation"`, using the `DELETE` keyword. Before performing the delete action, we use the `DETACH` keyword that assures that all the relationships in which the node was involved in are detached from the node itself in order to be able to remove it.

   Parameters

```
:param name => "Mathematics and Computers in Simulation";

```

   Command

```
MATCH (b:Book {name:$name})
DETACH DELETE b;

```

6.2. Queries

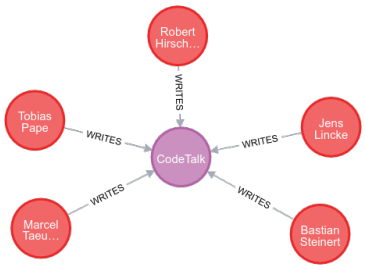
1. **Authors who wrote a certain paper**
   The query returns a graph with all the authors who wrote the papers titled `"CodeTalk"`. This is a very simple query which can be very useful in everyday searches.

   Parameters

```
:param title => "CodeTalk";

```

   Query

```
MATCH g = (:Author)-[:WRITES]->(:Paper {title:$title})
RETURN g

```



2. **Previous affiliations of an author**

   The query returns all the organizations to whom the author `"Annabel Sebag"` has been affiliated sorted by year in which she wrote the paper with that affiliation.

   Parameters

```
:param name => "Annabel Sebag";

```

   Query

```
MATCH (:Author {name:$name})-[r:WRITES]-(p:Paper)
WITH r.affiliation AS affiliation, p.year AS year
ORDER BY year DESC
RETURN DISTINCT affiliation, year

```

"affiliation"	"year"
"Autour De Minuit"	2013
"Premium Films, France"	2009
"Paris, France"	2007
"Supinfocom Arles, Paris, France"	2007
"Supinfocom Valenciennes, Paris, France"	2007
"Supinfocom Valenciennes, Paris, France"	1950

3. **Authors who wrote for a journal in a specific year**

   The query returns all the authors who wrote at least a paper for the journal named

"Briefings in Bioinformatics" in the year 2005. In the return statement we use the keyword DISTINCT in order to count only once each author, because we only want to know if the author wrote on that paper in the specified year, and not how many times.

Parameters

```
:param name => "Briefings in Bioinformatics";
:param year => 2005;

```

Query

```
MATCH (a:Author)-[:WRITES]->(p:Paper)-[:IS_PART_OF]->(j:Journal)
WHERE p.year = $year AND j.name = $name
RETURN DISTINCT a.name AS authorName

```

"authorName"
"L. V. Sudoplatov"
"V. E. Davidson"
"Yu. S. Postol'nik"
"I. S. Molchadskii"
"N. Yu. Taits"
"L. A. Gavur"
"N. P. Fedorin"
"A. P. Tolstopyat"
"N. I. Yalovoi"
"Yu. L. Rastorguev"

4. **Papers that belong to a Journal, written by authors while they were affiliated to a certain organization**
   The query returns the names of all the journals written by authors while they were affiliated to the organization named "Wien".

   In some of the queries we have seen before, we retrieved from the database the affiliations related to the authors. It is also interesting to see how easily we can retrieve the authors that are related to a certain affiliation, and then explore the

relationships of the author in order to obtain meaningful data, useful for everyday search in the bibliography database. The next query is very fast because centered mostly on relationships, a natural characteristic of graphs.

Parameters

```
:param affiliation => "Wien";

```

Query

```
MATCH (:Author)-[:WRITES {affiliation:$affiliation}]->(:Paper)-[:
    IS_PART_OF]->(b:Journal)
RETURN DISTINCT b.name as journalName;

```

```
╒═══════════════════════════════════════╕
|"journalName"                          |
╞═══════════════════════════════════════╡
|"Datenschutz und Datensicherheit"      |
├───────────────────────────────────────┤
|"Math. Meth. of OR"                    |
├───────────────────────────────────────┤
|"Computing"                            |
├───────────────────────────────────────┤
|"Wirtschaftsinformatik"                |
├───────────────────────────────────────┤
|"Biological Cybernetics"               |
├───────────────────────────────────────┤
|"HMD - Praxis Wirtschaftsinform."      |
├───────────────────────────────────────┤
|"Unternehmensforschung"                |
├───────────────────────────────────────┤
|"Zeitschr. für OR"                     |
└───────────────────────────────────────┘
```

5. **Authors which published more papers on journals in a specific year**
   The query returns the five authors that published more papers in the year 2000 on journals sorted by the number of papers they published. We use the function `count()` to obtain for each author the relative number of papers and limit the output at the first five authors discovered during the search.

Parameters

```
:param year => 2000;

```

Query

```
MATCH (a:Author)-[:WRITES]->(p:Paper)-[:IS_PART_OF]->(:Journal)
```

```
WHERE p.year = $year
WITH a, count(*) AS paperNum
RETURN a.name AS authorName, paperNum
ORDER BY paperNum DESC LIMIT 5;

```

"authorName"	"paperNum"
"David Redmiles"	4
"Douglas Blank"	2
"Steve Vestal"	2
"François Hennecart"	1
"Dietmar Dietrich"	1

6. **Number of papers written in conferences by authors whose name starts with specific letters per venue**
   The query returns for each venue the number of papers that have been presented in a conference considering only papers that are written by authors whose name starts for "A" or "S".

   Parameters

```
:param letter1 => "A";
:param letter2 => "S";

```

   Query

```
MATCH (a:Author)-[:WRITES]->(p:Paper)-[:IS_PART_OF]->(c:Conference)
WHERE a.name STARTS WITH $letter1 OR a.name STARTS WITH $letter2
WITH c.name AS conference, c.location AS location,
count(DISTINCT p) AS number_of_papers
RETURN DISTINCT conference, location, number_of_papers
ORDER BY number_of_papers DESC LIMIT 5;

```

"Conference"	"Location"	"number_of_papers"
"SIGGRAPH Computer Animation Festival"	"Bangkok, Thailand"	2
"SIGGRAPH Computer Animation Festival"	"Barcelona, Spain"	2
"NEMS"	"Vancouver, Canada"	1
"AINA Workshops"	"Moscow, Russia"	1
"SIGGRAPH Computer Animation Festival"	"Copenhagen, Denmark"	1

7. **Authors who wrote the most referenced papers which belong to publications of type book, while they were affiliated to a specific organization**
   The query returns the authors whose papers have been referenced the most in papers that are part of books while they were affiliated to `"University of Mannheim"`. We also want to retrieve only authors that published at least 2 papers, so we filter furthermore the result.

   Parameters

```
:param affiliation => "University of Mannheim";

```

   Query

```
MATCH (a:Author)-[w:WRITES]->(:Paper)<-[r:REFERENCES]-(:Paper)-[:
    IS_PART_OF]->(:Book)
WHERE w.affiliation = $affiliation
WITH a, count(DISTINCT r) AS totalReferences
WHERE totalReferences > 2
RETURN a.name AS authorName, totalReferences;

```

"author"	"totalReferences"
"Heiner Stuckenschmidt"	3
"Anne Schlicht"	3

8. **Papers indirectly referenced by other papers at a certain distance of steps**
   The query returns different papers linked to each other by the relation `REFERENCES`

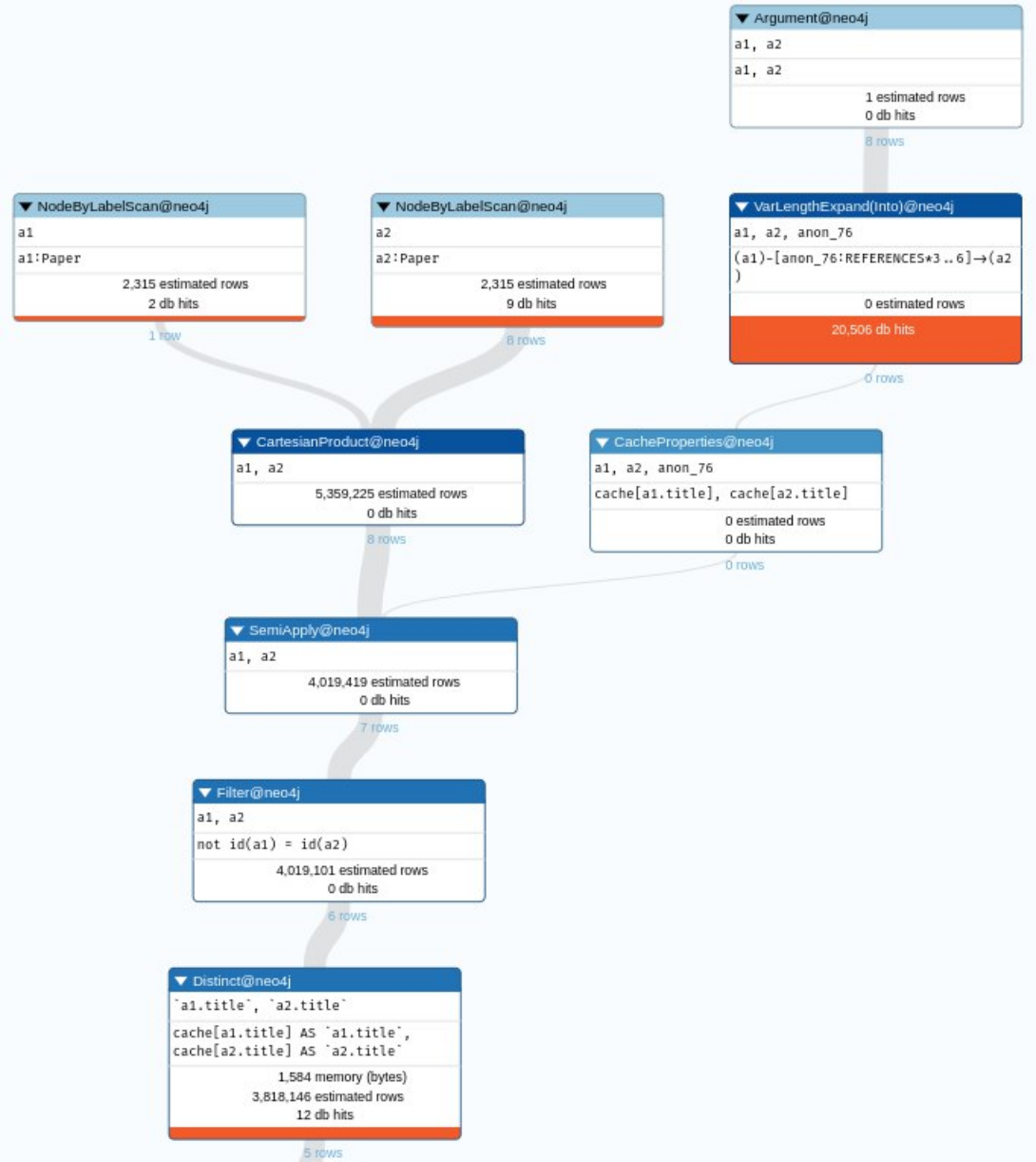
at a distance of steps between 3 and 6. We limit the results in order to get the output as soon as the process finds 5 rows that respect the filter of the search.

```
MATCH (a1:Paper), (a2:Paper)
WHERE id(a1) <> id(a2) AND (EXISTS {MATCH (a1)-[:REFERENCES
    *3..6]->(a2)})
RETURN DISTINCT a1.title AS firstPaper, a2.title AS secondPaper
LIMIT 5;

```

"firstPaper"	"secondPaper"
"3GIO."	"The relationship between canopy parameters and spectrum of winter whe at under different irrigations in Hebei Province."
"3GIO."	"A solution to the problem of touching and broken characters."
"3GIO."	"Timing yield estimation using statistical static timing analysis"
"3GIO."	"360°"
"3GIO."	"300"

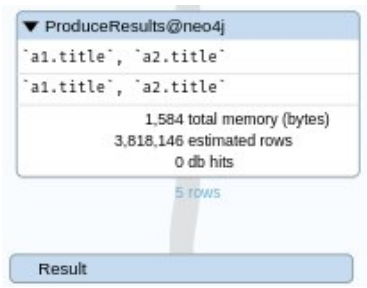
The performance is not optimal and using the `PROFILE` command in the same query we can see how the search is actually made in the graph using this specific interrogation on the database.

```
PROFILE
MATCH (a1:Paper), (a2:Paper)
WHERE id(a1) <> id(a2) AND (EXISTS {MATCH (a1)-[:REFERENCES
    *3..6]->(a2)})
RETURN DISTINCT a1.title AS firstPaper, a2.title AS secondPaper
    LIMIT 5;

```



We can notice that the execution of the query is done in parallel trying on one side to perform the `MATCH` of the two papers separately and on the other side to find the relationships between the two papers given by `REFERENCES` from 3 to 6 steps away. The two results of the search are then merged together.

In this process we estimated a lot of rows and wasted time, by doing the controls individually.



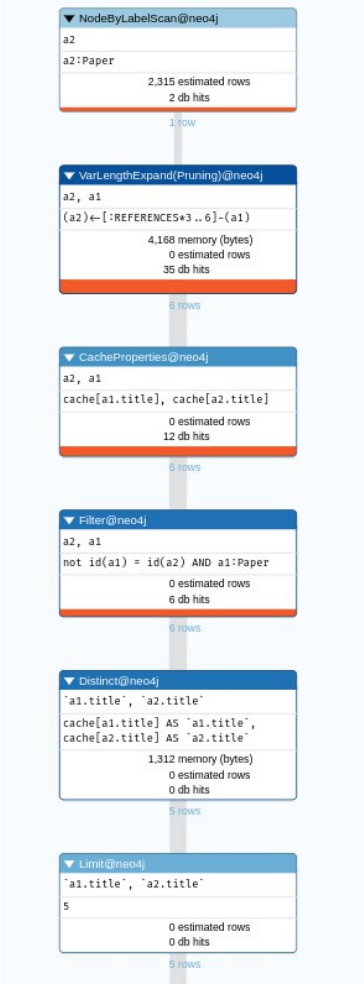
To make the query more efficient we can bring the filtering of the data a step ahead, so we can check directly in the `MATCH` command for the relationships between the papers.

```
MATCH (a1:Paper)-[:REFERENCES*3..6]->(a2:Paper)
WHERE id(a1) <> id(a2)
RETURN DISTINCT a1.title AS firstPaper, a2.title AS secondPaper
LIMIT 5;

```

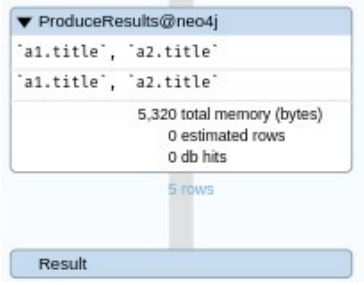
We notice that the results can be different from the precedent approach, due to the use of `LIMIT` that stops the research as soon as it gets the 5 results of interest and in this case the search is done differently so the outputs are not the same.

"firstPaper"	"secondPaper"
"Editorial."	"3GIO."
"Discussion"	"3GIO."
"Acknowledgements"	"3GIO."
"Colophon"	"3GIO."
"Capstone"	"3GIO."

Using this version of the query we have that the search is more linear, because in the `MATCH` we don't look for the papers separately, but once matched a `Paper` we search for its recursive `REFERENCES` relationships and match at once the two papers and their connections. This approach is really fast and improves efficiency.



We can see that reaching the result the command required more memory, but we didn't estimated useless rows of data, extracting only what we are interested in without an extended search on the graph.



Alternatively we can tackle this search from another point of view. We can use a query based precisely on the steps we are interested in, making the query more explicit and getting other results, because in the recursive form the process could stop before reaching this exact amount of steps, especially using the `LIMIT` keyword.

```
MATCH (a1:Paper)-[:REFERENCES]->(a2:Paper)-[:REFERENCES]->(a3:Paper
    )-[:REFERENCES]->(a4:Paper)-[:REFERENCES]->(a5:Paper)-[:
    REFERENCES]->(a6:Paper)
WHERE id(a1) <> id(a2) AND id(a2) <> id(a3) AND id(a3) <> id(a4)
    AND id(a4) <> id(a5) AND id(a5) <> id(a6)
RETURN DISTINCT a1.title AS firstPaper, a6.title AS secondPaper
LIMIT 5;

```

"firstPaper"	"secondPaper"
"Funding"	"Foreword"
"Funding"	"FMOL"
"Funding"	"GCM."
"Funding"	"Digitrama"
"Funding"	"Editorial"

9. **Best collaborator of all times for each author in terms of papers written in journals**

   The following query allows the researcher to understand for each author who is the best collaborator of all times considering their collaborations on journals. In the result we can sometimes see a symmetry in the data, if an author has another as best collaborator and vice versa.

   Firstly we do a `MATCH` imposing our constraints. Secondly we group by authors and co-authors counting for each couple how many papers published on journals they wrote together, and we order the table by this number in descending order. On the next `WITH` clause for each author we extract only the best collaborator, namely the one with whom he wrote more papers on journals.

   Parameters Query

```
MATCH (a:Author)-[:WRITES]->(p:Paper)<-[:WRITES]-(ca:Author), (p)
    -[:IS_PART_OF]->(:Journal)
WITH DISTINCT a, ca, count(DISTINCT p) AS cnt ORDER BY cnt DESC
WITH DISTINCT a, collect(ca.name) AS colabs, collect(cnt) AS cnts
RETURN a.name AS author, colabs[0] AS bestCollaborator, cnts[0] AS
    numberOfPapers LIMIT 5

```

"author"	"bestCollaborator"	"numberOfPapers"
"David J. Lipman"	"Dennis A. Benson"	10
"Dennis A. Benson"	"David J. Lipman"	10
"James Ostell"	"Dennis A. Benson"	9
"Ilene Karsch-Mizrachi"	"David J. Lipman"	6
"K. Prachar"	"W. Nöbauer"	6

10. **Papers which are at the base of a given field of study**

    The query returns the papers which are at the base of a given field of study. To be the base for a field of study means to not reference directly or indirectly other papers of that field of study. With this query we retrieve the papers that are probably at the base of some field of study because an upper bound of 7 is chosen to check the indirect references.

    A variant of this query can be to retrieve the authors who wrote the base papers of a field of study, so the authors who are the "founders" of that field. Obviously the query is more meaningful if run on a very large set of data.

    Parameters

```
:param fos => "Statistics";

```

    Query

```
MATCH (founder:Paper)-[:BELONGS_TO]->(fos:Fos{fos:$fos})<-[:
    BELONGS_TO]-(p:Paper)
WHERE id(founder) <> id(p)
WITH collect(p) AS test, founder, fos
WHERE all(p IN test WHERE NOT EXISTS((founder)-[:REFERENCES*7]->(p)
    ))
RETURN fos AS field, collect(DISTINCT founder.title) AS
    founderPapers

```

"field"	"founderPapers"
{"fos":"Statistics"}	["Timing yield estimation using statistical static timing analysis","Experiments","BUCHBESPRECHUNGEN","Anecdotes","Education.","BPEL4WS.","Alphabet.","Forward","OBITUARY.","Deadline","Dice","Buchbesprechungen","AirEOD"]

11. **Total pages written by the authors of an organization within a journal**
The query returns the total number of pages written by authors affiliated to the organization `"Federal Institute of Technology, Switzerland"` within journals. The function `avg()` can be used to compute the average of the pages written by an author in the papers.

Parameters

```
:param affiliation=>"Federal Institute of Technology, Switzerland";

```

Query

```
MATCH (a:Author)-[w:WRITES {affiliation:$affiliation}]->(p:Paper)-[
    i:IS_PART_OF]->(j:Journal)
WHERE i.page_start IS NOT NULL AND i.page_end IS NOT NULL
WITH a, sum(i.page_end - i.page_start) AS num_pag
ORDER BY num_pag DESC
RETURN a AS Author, num_pag;

```

"Author"	"num_pag"
{"name":"C. Genillard","id":"53f458e5dabfaee4dc81bc6e"}	19
{"name":"A. Strohmeier","id":"53f45b8bdabfaee1c0b4094c"}	19

12. **Organizations which have published by their own more papers within a specific journal with associated fields of study**
The query returns the organizations which have written more papers in an exclusive way, which means that the authors of the papers have all the same affiliation on `"Commun . ACM"`. The query provides the fields of study on which the considered papers were focused on.

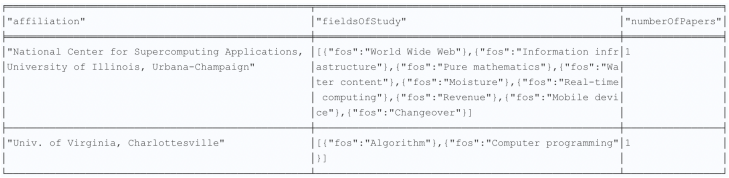
Parameters

```
:param journalTitle => "Commun. ACM";

```

Query

```
MATCH (a1:Author)-[w1:WRITES]->(p1:Paper)-[:IS_PART_OF]->(j:Journal
    {name:$journalTitle}), (p1:Paper)-[:BELONGS_TO]->(fos:Fos), (a2
    :Author)-[w2:WRITES]-(p1:Paper)
WHERE a1 <> a2 AND w1.affiliation IS NOT NULL
```

```
WITH w1.affiliation AS affiliation, collect(DISTINCT fos) AS
   fieldsOfStudy, collect(DISTINCT w2.affiliation) AS others, count
   (DISTINCT p1) AS numberOfPapers
WHERE ALL(organization IN others WHERE affiliation=organization)
WITH affiliation, fieldsOfStudy, numberOfPapers
ORDER BY numberOfPapers DESC
RETURN affiliation, fieldsOfStudy, numberOfPapers
LIMIT 2

```



"affiliation"	"fieldsOfStudy"	"numberOfPapers"
"National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign"	[{"fos":"World Wide Web"},{"fos":"Information infrastructure"},{"fos":"Pure mathematics"},{"fos":"Water content"},{"fos":"Moisture"},{"fos":"Real-time computing"},{"fos":"Revenue"},{"fos":"Mobile device"},{"fos":"Changeover"}]	1
"Univ. of Virginia, Charlottesville"	[{"fos":"Algorithm"},{"fos":"Computer programming"}]	1

13. **Recommended books given one book**

    The query returns the book considered the nearest to the one given as parameter. The similarity between books is evaluated in terms of number of keywords associated to the papers linked to the books. In addition, fields of study which are common to the two books are shown.

```
:param title => "RoboCup 2009";

```

Query

```
MATCH
    (p1:Paper)-[:IS_PART_OF]->(b1:Book),
    (p1:Paper)-[:BELONGS_TO]->(fos1:Fos)<-[:BELONGS_TO]-(p2:Paper),
    (p1)-[:HIGHLIGHTS]->(k1:Keyword)<-[:HIGHLIGHTS]-(p2),
    (p2:Paper)-[:IS_PART_OF]->(b2:Book)
WHERE b1 <> b2 AND p1 <> p2 AND b1.name = $title
WITH b1, b2, collect(DISTINCT fos1) AS fos, count(DISTINCT k1) AS
   score
ORDER BY score DESC
RETURN b2.name AS relatedBook, fos, score
LIMIT 3;

```

"relatedBook"	"fos"	"score"
"SIGGRAPH Sketches"	[{"fos":"Mobile device"},{"fos":"Mobile computing"}]	3
"SIGGRAPH Electronic Art and Animation Catalog"	[{"fos":"Artificial intelligence"}]	3
"SIGGRAPH Abstracts and Applications"	[{"fos":"Suurballe's algorithm"}]	1

14. **Possible new interesting collaborations between authors in the same fields of study**

    The query returns the new possible interesting collaborations, by which we mean the couple of authors who haven't written a paper together but have written many times with common authors and wrote about similar topics, so they are involved in the same fields of study.

    Parameters

```
:param minimum_collabs =>1;

```

    Query

```
MATCH
    (a1:Author)-[w1:WRITES]->(p1:Paper)<-[w2:WRITES]-(a2:Author),
    (a2:Author)-[w3:WRITES]->(p2:Paper)<-[w4:WRITES]-(a3:Author),
    (a1)-[:WRITES]->(p3:Paper)-[:BELONGS_TO]->(f1:Fos)<-[:
   BELONGS_TO]-(p4:Paper)<-[:WRITES]-(a3)
WHERE NOT EXISTS ((a1)-[:WRITES]->(:Paper)<-[:WRITES]-(a3)) AND a1
   <> a3 AND a1 <> a2 AND a2 <> a3
WITH a1, a3, collect(DISTINCT f1.fos) AS fields, count(DISTINCT f1)
    AS common_fields, count(DISTINCT p1) AS common_collabs1, count(
   DISTINCT p2) AS common_collabs2
WHERE common_collabs1 > $minimum_collabs AND common_collabs2 >
   $minimum_collabs
WITH a1, a3, fields, common_fields, common_collabs1 +
   common_collabs2 AS common_collabs
ORDER BY common_fields DESC, common_collabs DESC
RETURN DISTINCT a1.name as Author1, a3.name as Author2, fields,
   common_collabs, common_fields
LIMIT 5

```

"Author1"	"Author2"	"fields"	"common_collabs"	"common_fields"
"Ilene Karsch-Mizrachi"	"Mark S. Boguski"	["Router","Engineering"]	8	2
"Mark S. Boguski"	"Ilene Karsch-Mizrachi"	["Router","Engineering"]	8	2
"K. Mayrhofer"	"H. Muthsam"	["Statistical static timing analysis","Statistics"]	7	2
"K. Mayrhofer"	"J. Tichý"	["Statistical static timing analysis","Statistics"]	7	2
"K. Mayrhofer"	"H. Rindler"	["Statistical static timing analysis","Statistics"]	7	2

15. **Shortest connection between two papers with no common field of study**
With this query we want to find what is the shortest path, by navigating the references, between two papers which do not have any field of study in common. Firstly we bound the papers node and the associated field of studies to variables `p1/p2` and `fos1/fos2` respectively and we impose that `p1` and `p2` must be different. Secondly, we collect the fields of study associated to each paper and put a condition stating that the intersection of this two collection must be zero. Finally we call the `shortestPath` function and impose all the condition on the path to be found; we put an upper bound of 7 to the length of the path to avoid searching to deep in graph and limited the search to the first valid path found using the clause `WITH sp LIMIT 1`, where `sp` refers to the result of the call to the `shortestPath` function.

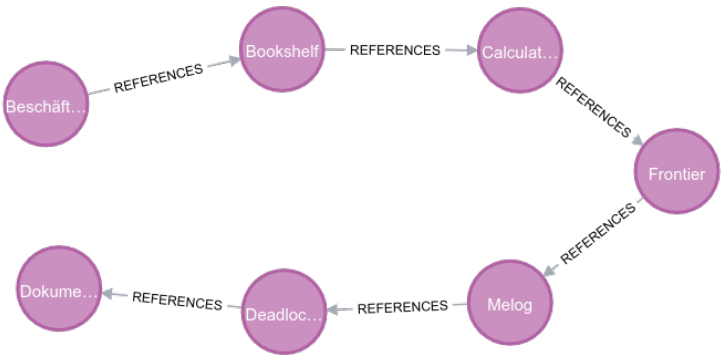
Parameters

```
:param fos1 => "Irrigation";
:param fos2 => "Artificial intelligence";

```

Query

```
MATCH
    (fos1:Fos) <-[:BELONGS_TO]- (p1:Paper) -[:BELONGS_TO]-> (:Fos {fos:$fos1}),
    (fos2:Fos) <-[:BELONGS_TO]- (p2:Paper) -[:BELONGS_TO]-> (:Fos {fos:$fos2})
WHERE p1 <> p2
WITH p1, collect(DISTINCT fos1) AS foss1, p2, collect(DISTINCT fos2) AS foss2
WHERE all(f IN foss1 WHERE NOT f IN foss2)
MATCH sp = shortestPath((p1)-[:REFERENCES*..7]->(p2))
WHERE length(sp) > 1
RETURN sp LIMIT 1;

```

Below we can see the result of the query ran with the aforementioned parameters.


Bookshelf
Calculat…
Frontier
Melog
Deadloc…
Dokume…
Beschäft…
REFERENCES
REFERENCES
REFERENCES
REFERENCES
REFERENCES
REFERENCES

Part III

MongoDB

7 | Introduction to the problem

In this part we implement a bibliographic database through MongoDB, which is a document-oriented database. In the previous part, using Neo4j, the main focus was on the relationships between nodes, instead MongoDB's paradigm is different. It shifts the attention to the document itself, which is the atomic and fundamental element of the database. In this case each document represents a single paper and contains all the related information. All the articles are stored in the same collection called `bibliography`. A bibliographic database is a use case where the documental approach may be very successful. The main focus of this kind of database is to store the information regarding each single paper without the need to relate many articles together and without doing very complex queries.

8 | Document structure

Since we are using a documental database, papers are the main entity and, because of this, it is reasonable to store the text and the structure of the articles. This fields are added to the attributes listed in the previous part regarding graph databases. In particular, each paper is composed by many sections and each section has a title, several paragraphs and a certain number of image URLs with associated captions. Moreover, each section can contain subsections which have the same structure as a section.

In addition, also the email and bio of authors has been added as well as the publication date of the article.

The following table shows the document's fields associated to their respective MongoDB type and meaning.

Field name	Type	Description
_id	String	paper ID
title	String	paper title
authors	Array of Object	list of authors
authors._id	String	author ID
authors.name	String	author name
authors.org	String	author affiliation
authors.email	String	author email
authors.bio	String	author short bio
keywords	Array of String	paper keywords
fos	Array of String	paper fields of study
publication_type	String	paper type
venue	String	publication name
volume	Int32	journal volume
issue	Int32	journal issue
page_start	Int32	start page
page_end	Int32	end page
date	Date	publication date
lang	String	paper language
isbn	String	book ISBN
issn	String	journal ISSN
doi	String	paper DOI
publisher	String	publisher name
location	String	conference location
abstract	String	paper abstract
url	Array of String	paper URLs
references	Array of String	paper references
sections	Array of Object	paper sections
sections.title	String	section title
sections.paragraphs	Array of String	section paragraphs
sections.subsections	Array of Object	section subsections
sections.figures	Array of Object	section figures
sections.figures.URL	String	figure URL
sections.figures.caption	String	figure caption

Table 8.1: Description of the fields with related type and meaning.

Notes:

- `publication_type` can only assume "Journal", "Book" or "Conference" as value;
- a paper of type "Journal" can have the fields `issn`, `volume`, `issue` and `publisher`;
- a paper of type "Book" can have the fields `isbn` and `publisher`;
- a paper of type "Conference" can have the field `location`;
- `subsections` have the same structure as `sections`, so they could contain multiple `subsections` and so on in a recursively way. Actually, we decided to consider only one level of subsections, so `subsections` don't have the `subsections` field.

The following code snippet shows our document structure represented in the JSON format.

```
{
    "_id": <String>,
    "title": <String>,
    "authors": [ {
        "_id": <String>,
        "name": <String>,
        "email": <String>,
        "bio": <String>,
        "org": <String>
    } ],
    "abstract": <String>,
    "keywords": [<String>],
    "fos": [<String>],
    "publication_type": <String>,
    "venue": <String>,
    "volume": <Int32>,
    "issue": <Int32>,
    "issn": <String>,
    "publisher": <String>,
    "isbn": <String>,
    "location": <String>,
    "date": <ISODate>,
    "page_start": <Int32>,
    "page_end": <Int32>,
    "lang": <String>,
    "doi": <String>,
    "url": [<String>],
    "references": [<String>],
    "sections": [ {
        "title": <String>,
        "paragraphs": [<String>],
```

```
        "figures": [ {
            "URL": <String>,
            "caption": <String>
        } ],
        "subsections": [ {
            "title": <String>,
            "paragraphs": [<String>],
            "figures": [ {
                "URL": <String>,
                "caption": <String>
            } ]
        } ]
    } ]
}
```

Below we provide some examples of the three main papers types that appear in our document with their associated attributes. The difference in the attributes is minimal and, as mentioned beforehand, it concerns the fields related to the publishing type of the paper.

Model of the journal: This is the document structure for the papers published in journals.

```
_id: "53e99905b7602d9702145cf7"
title: "An O(n) bin-packing algorithm for uniformly distributed data."
authors: Array
  0: Object
     _id: "53f45291dabfaeecd69dcbf8"
     name: "J Csirik"
     email: "j.csirikf8@gmail.com"
     bio: "My name is J Csirik"
  1: Object
keywords: Array
   0: "bin packing"
   1: "bin packing problem"
   2: "heuristic algorithm"
   3: "probabilistic analysis"
fos: Array
   0: "computer communication networks"
   1: "algorithm"
   2: "probabilistic analysis of algorithms"
   3: "probabilistic logic"
   4: "bin packing problem"
   5: "mathematics"
page_start: 313
page_end: 319
lang: "en"
volume: 36
issue: 4
doi: "10.1007/BF02240206"
url: Array
   0: "http://dx.doi.org/10.1007/BF02240206"
   1: "http://link.springer.com/article/10.1007/BF02240206"
abstract: "We give a first-fit type algorithm, with running timeO(n), for the cla…"
references: Array
   0: "53e9be4ab7602d9704b09bda"
   1: "53e9a81fb7602d970316858e"
   2: "53e9ac28b7602d97035eb4f5"
   3: "53e9ac82b7602d9703658b34"
   4: "53e9a162b7602d9702a5483c"
publication_type: "Journal"
publisher: "IEEE Systems, Man, and Cybernetics Society"
venue: "Computing"
sections: Array
  0: Object
     title: "Excellent"
     paragraphs: Array
        0: "These gummy strawberries are YUMMY and very flavorful. We are big fans…"
        1: "Funny thing about this order: I thought these were a different kind of…"
        2: "These are really great tasting and oh so cute.  They were perfect for …"
     figures: Array
        0: Object
           URL: "https://java.com/nulla/sed.html"
           caption: "Great, healthy dog food for dogs with food allergies"
     subsections: Array
        0: Object
           title: "Huge pack of chewy gummies"
           paragraphs: Array
           figures: Array
        1: Object
  1: Object
  2: Object
  3: Object
date: 1983-10-02T23:49:56.000+00:00
```


Model of the book: This is the document structure for the papers published in books.

```
_id: "53e997ecb7602d9701fe6bb9"
title: "Applicative Information Systems"
authors: Array
  0: Object
    _id: "53f462afdabfaee2a1d9fe5d"
    name: "Mario Coppo"
    email: "mario.coppo5d@gmail.com"
    bio: "My name is Mario Coppo"
  1: Object
  2: Object
keywords: Array
  0: "applicative information systems"
  1: "information system"
fos: Array
  0: "information system"
  1: "computer science"
  2: "theoretical computer science"
  3: "recursive functions"
page_start: 35
page_end: 64
lang: "en"
isbn: "3-540-12727-5"
doi: "10.1007/3-540-12727-5_2"
url: Array
  0: "http://dx.doi.org/10.1007/3-540-12727-5_2"
abstract: "Without Abstract"
references: Array
  0: "53e9b90bb7602d97044f09eb"
  1: "53e99a2bb7602d97022845a9"
  2: "53e99c1ab7602d97024cae1e"
  3: "53e9a500b7602d9702e1ecf6"
publication_type: "Book"
publisher: "Association for Computing Machinery (ACM)"
venue: "CAAP"
sections: Array
  0: Object
    title: "These are delicious!"
    paragraphs: Array
      0: "we love Pop Chips, but they ceased being available in our area.  So I …"
      1: "My parents were visiting us in Las Vegas from the east coast. I was at…"
      2: "Lower in calories and great taste.  I like them just as much as the re…"
      3: "Best chips I've ever eaten but you should sell them in 11 to 15 oz bag…"
    figures: Array
      0: Object
        URL: "http://usatoday.com/eget/eros/elementum.jsp"
        caption: "For the price, these are great for all. Even a crowd. Much better than…"
      1: Object
    subsections: Array
      0: Object
        title: "Unbelievable - healthy and incredibly tasty"
        paragraphs: Array
          0: "These chips are really good. Crisp and delicious. My favorite flavor i…"
          1: "After years of being addicted to Smart Puffs, as soon as I tried these…"
        figures: Array
  1: Object
    title: "Not Your Grandma's Cookies, But They Get The Job Done"
    paragraphs: Array
    figures: Array
    subsections: Array
date: 1954-07-31T08:06:59.000+00:00
```


Model of the conference: This is the document structure for the papers published in conferences.

```
_id: "53e997e9b7602d9701fe36df"
title: "Coalitional affinity games"
authors: Array
  0: Object
      _id: "53f43d50dabfaee1c0ad4b85"
      name: "Simina Brânzei"
      email: "simina.brânzei85@gmail.com"
      bio: "My name is Simina Brânzei"
  1: Object
      _id: "5440ede1dabfae805a709079"
      name: "Kate Larson"
      email: "kate.larson79@gmail.com"
      bio: "My name is Kate Larson"
keywords: Array
    0: "hedonic game"
    1: "coalitional affinity game"
    2: "stable coalition structure"
    3: "interesting class"
    4: "stability property"
    5: "coalition structure"
    6: "stability gap"
    7: "core solution concept"
    8: "social welfare"
    9: "affinity game"
fos: Array
    0: "mathematical economics"
    1: "computer science"
    2: "solution concept"
    3: "social welfare"
    4: "bounded function"
  page_start: 1319
  page_end: 1320
  lang: "en"
  doi: "10.1145/1558109.1558272"
url: Array
    0: "http://dx.doi.org/10.1145/1558109.1558272"
    1: "http://doi.acm.org/10.1145/1558109.1558272"
  abstract: "We present and analyze coalitional affinity games, a family of hedonic…"
references: Array
    0: "53e9a938b7602d970328d1f2"
  publication_type: "Conference"
  location: "Los Angeles, USA"
  venue: "Autonomous Agents & Multiagent Systems/Agent Theories, Architectures, …"
sections: Array
  0: Object
      title: "Disappointed"
    paragraphs: Array
        0: "This toy did not last. I have a boxer and he chewed the rope off the j…"
        1: "My dog destroyed the rubber "cord" in 5 minutes (worse yet, swallowed …"
        2: "I BOUGHT THIS FOR MY DOG AND HE TOOK ONE LOOK AT IT PLAYED WITH IT FOR…"
        3: "When i took this out my puppy couldn't wait to get into it.  I put the…"
    figures: Array
      0: Object
          URL: "https://ox.ac.uk/pede/lobortis.jpg"
          caption: "Of al the Stash iced teas this is the best, and it is still not all th…"
      1: Object
      2: Object
      3: Object
    subsections: Array
      0: Object
          title: "Hours of endless entertainment"
        paragraphs: Array
        figures: Array
      1: Object
  1: Object
  2: Object
  3: Object
date: 2007-10-22T21:29:26.000+00:00
```


8.1. Differences with respect to the previous models

The main advantage of this approach is the fact that it is very easy to get all the relevant data of an article because all the information are stored together and usually allow to avoid performing expensive joins that would be necessary using a relational database.

Furthermore, MongoDB is flexible permitting to have heterogeneous documents, we need to maintain the same skeleton for all documents in the same collection in order to perform meaningful queries and have a clean database, but also differences between documents are admitted. As highlighted in the notes above, different types of articles can have different fields and if an article doesn't have a certain data is not necessary to store an empty field, simply the attribute is not created. In addition, this flexibility makes easy the storage of

the text structure that could, in principle, have many levels of subsections.

One drawback is the duplication of data, for example it is possible that an author has written more than one paper that is stored in the database, so the author's information are saved multiple times. This situation could lead to some inconsistency and requires attention when data updates are performed. But, in the field of bibliography, it is seldom necessary to do updates since all data refer to a specific article and if a paper is modified it is more probable to create a new document with the new version than updating the older one.

9 | Data upload

9.1. Pre-processing

The assignment for this part of the project defined a basic structure of the document with the fields it must have. Some of the fields defined by this structure were not present in our previous dataset so we had to fix this aspect by doing some pre-processing of the data before working with it. The starting point has been the dblp dataset, cleaned with python scripts just like in the Neo4j part, but also enriched with new information and new filters on the data. In particular, the main work revolved around the following fields.

Author email and bio Using a Python script, we added the emails of the authors adopting the following format:

```
author.name + last-two-digits-of-the-author._id + "@gmail.com"
```

By mean of the same script, we added the bio using the format:

```
"My name is " + author.name + "." + "I'm currently working for " +
    author.org
```

The second sentence, `'"I'm currently working for " + author.org'` is present only if the author has an affiliation.

Then we generated random dates adopting the format `ISODate` of `MongoDB` from www.mockaroo.com and added them to the original dataset with a Python script.

Sections and subsections The sections and subsections fields, as mentioned in the assignment instructions were mandatory for the document structure. However, the documents in our dataset didn't have those fields so we had to manually generate them. We did so by using a Python script. More specifically the titles of the sections and subsections and the text for the paragraphs were extracted from the fields `Summary` and `Text` respectively, present in an external dataset named `Reviews.csv` [1], which consists of food

[1] Amazon Fine Food Reviews, accessed on the 14/11/2022 https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews

reviews from Amazon. The script firstly generated a set of unique sections with a random number of paragraphs and a random number of subsections, which are themselves sections but without a subsections field (the boundaries of this random number were tuned in the script). We decided to maintain only a level of subsections in order to have a more concise document that satisfies the requirements. Secondly a (bounded) random number of sections were added to each paper document of the original dataset.

URLs and captions Images URLs have been generated from www.mockaroo.com, while captions are from the `Summary` field of the same dataset `Reviews.csv` used to generate the sections. We wrote a Python script to add both values to the original dataset.

Venue The `venue` field was generated from the `venue.raw` given by the initial dataset. In the version of the `dblp` that we downloaded the venue was a composed field with a lot of information, such as the `raw`, the `name`, the `_id` and the `type`. We chose to maintain only the `raw`, that encoded the more interesting data and we assigned it to the field named `venue`.

9.2. Upload process

The upload of the data was quite straightforward since our dataset was already in JSON format. Since the importing feature of MongoDB was much more efficient than Neo4j we were able to load a much bigger dataset containing 56K documents.

After uploading the data we performed the following filters:

- We deleted some fields from all the documents, because we were not interested in maintaining all the data, but just the ones that matched more or less the chosen structure for the collection. We used the `$unset` command to remove specific fields, such as the `pdf` and the `year`, that was superfluous having the field `date`. An example of this procedure is reported in the third command of the following section.
- We checked all the fields in order to remove empty fields, so we cleaned the dataset from uninteresting values. We executed this cleaning part firstly by checking if the field equals to the empty value `""` and then using the `$unset` command on the respective element. As before we can see and example of this procedure also in the third command of the following section.
- We executed some additional commands to filter the data, from the `FILTERING` option of `MongoDB Compass`, such as the following one:

```
{ $and: [ {
        title: { $not: { $regex: '^[1-9]' } } }, {
        page_start: { $regex: '^[0-9]+$' } }, {
        page_end: { $regex: '^[0-9]+$' } }, {
        volume : { $not : { $eq : "null"} } }, {
        abstract: { $not : { $eq : "null"} } } ] }

```

Controls similar to the one just reported can be executed at any time, for instance after acquiring some new data from an external source, in order to have only documents with interesting values.

10 | Commands and queries

10.1. Commands

1. **Inserting a document**
   To insert in our database only one document at the time, we use the following function `insertOne()`, which takes in input one document and inserts it in the DB. If the document does not specify an `_id` field, as is our case, then mongod [1] will add the `_id` field and assign a unique `ObjectId()` for the document before inserting it. Since when importing this dataset the `_id` were defined as String we decided to explicit the `_id` creation so we can use the `toString()` method to cast it to a String type and be consistent with the data we loaded from the dataset. As shown below, for this example we have retrieved a real-world paper and shaped its information to our document structure. Firstly, we searched for the author by name and since we didn't find him, we created a new `_id` for him. For text formatting reasons in the preview below, we have omitted the values of the `abstract` and `sections` fields. The `references` field is an empty array because all the referenced papers by this one are not present in our DB.

```
db.bibliography.insertOne( {
    _id: ObjectId().toString(),
    title: "Reinforcement Learning for Improving Agent Design",
    authors: [ {
        _id: ObjectId().toString(),
        name: "David Ha",
        email: "hadavid@google.com",
        bio: "AI resercher at Google Brain, based in Tokyo, Japan",
        org: "Google Brain" } ],
    abstract: "...",
    keywords: [
        "computer science",
        "ai",
        "reinforcement learning",
```

[1] mongod is the primary daemon process for the MongoDB system

```
        "cumulative reward",
        "joint learning" ],
    fos: [
        "computer science",
        "ai",
        "reinforcement learning" ],
    publication_type: "Journal",
    publisher: "MIT Press Direct",
    venue: "Artificial Life",
    volume: 25,
    issue: 4,
    issn: "352-365",
    date: ISODate('2019-12-02T10:49:36.000'),
    page_start: 23,
    page_end: 46,
    lang: "en",
    doi: "doi.org/10.1162/artl_a_00301",
    url: [
        "https://arxiv.org/abs/1810.03779",
        "https://arxiv.org/abs/1810.03779v3",
        "https://doi.org/10.48550/arXiv.1810.03779" ],
    references: [ ],
    sections: [ ]
} )

```

An easy way to know if the papers referenced by the one we are about to insert are present in the database, is to query it using the DOIs of the referenced paper (like is shown in the code snippet below) and then get the `_id` values to be added to the references.

```
db.bibliography.aggregate( [ {
    $match: {
        doi: { $in: [
            "10.1109/CMPSAC.2002.1044548",
            "10.1007/s11704-011-0127-6",
            "10.1016/j.datak.2008.09.003" ] } } }, {
    $project: { "_id": 1 } } ] )

```

Using the database structure presented earlier, we realized that managing the data could not be so straightforward, especially when a new document insertion is required. These drawbacks are due to the data sources we had because only a small percentage of the papers had the values for DOI and ORCID but we would have

shrunk too much our dataset keeping only those articles. In a further improved version, some precautions could be taken to have a more clean and easy-to-use database if the DOI field is known for all documents and the ORCID for all authors.

With the above assumptions, we could use the DOI for the references because it is a globally unique and independent identifier usable in any database implementation. In this way, it would be easier to insert a new document because the new paper could be added to the collection even if the references are not present in the dataset, and the related papers could be inserted afterward. Likewise, using the ORCID as the author identifier would also make it easier to add new documents because the ORCID has the same global uniqueness property. It would not be necessary to check if the author is already present in the database to set the same id, risking creating inconsistency, but we could insert it just as it is.

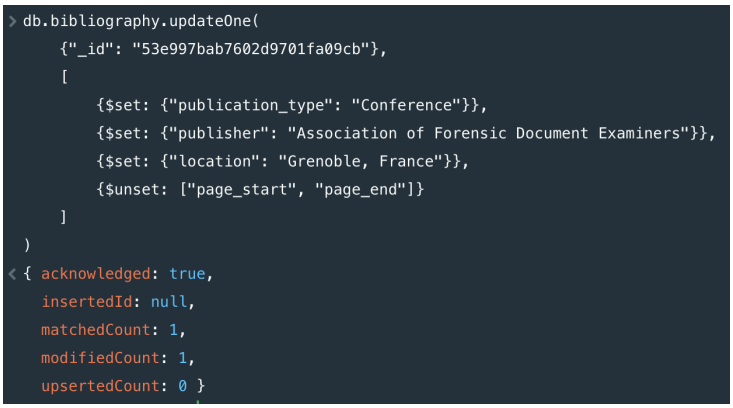
Despite the further checks required in our implementation, when adding the document, we decided to avoid imposing too many limitations on the structure of the data. If the papers have the DOI and the authors have the ORCID is easier to search the collection, but we cannot guarantee the presence of these fields, so in order to have a more flexible database, we accept papers without DOI and ORCID. A less bounded dataset leads to the necessity of adding some checks to maintain data consistency and requires being more careful when inserting data to avoid duplication.

2. **Update a single document**
   The command `updateOne` allows to search for a document that matches a specific filter and modify it accordingly to the update rules expressed inside the query. In general, the command modifies only the first found document, but if no document matches the search, then no changes are applied to the database.

   The `$set` operator sets a new value for the specified attribute or creates it if it does not exist. The `$unset` operator deletes a specified document attribute if the document has the field otherwise, nothing changes. The following command might be useful when new data regarding papers become available or when some information needs to be updated due to mistakes. The command allows to find a document with the field `_id` equal to `"53e997bab7602d9701fa09cb"` and changes its field `publication_type` to `"Conference"` because this is its actual type and also changes its field `publisher` to `"Association of Forensic Document Examiners"` with the command `$set`. This operator is also used to create the field `location` to which is assigned the value `"Grenoble, France"`. The update part of the query also

allows the removal of the fields `page_start` and `page_end` by using the command `$unset`.

```
db.bibliography.updateOne( {
    _id: "53e997bab7602d9701fa09cb" },
    [ {
        $set: { publication_type: "Conference" } }, {
        $set: { publisher: "Association of Forensic Document
   Examiners" } }, {
        $set: { location: "Grenoble, France" } }, {
        $unset: [
            "page_start",
            "page_end" ] } ] )

```



In this case only one document could have been found because the field `_id` is the unique identifier. It is necessary to be careful in updating the database in order to avoid updating the wrong documents, deleting fields and changing important values.

Another important use case of `updateOne` is associated to the need to replace a document with a different one. This exchange can be done using the operator `$replaceWith` and specifying the document to remove and the one that has to be put inside the database instead of it.

3. **Remove fields**
   Uploading the dataset, as we previously explained in the chapter about the upload process, we ended up with a lot of data, some of which were not of our interest. Using the following command we consider all the data entries of the collection `bibliography` and without any filtering we just remove the `year` field from all the documents. We decided to remove this field because we already have the `date` so this is unnecessary.

```
db.bibliography.updateMany( {
    }, {
    $unset: { year: "" } } )

```

In order to maintain only interesting values about the papers, we also remove the empty fields. It doesn't make sense to keep fields that are equal to an empty value, so, for instance, if `doi` is just an empty string, we delete it from the respective document. We keep only useful data that respect the structure of the dataset we want to work with to extract knowledge. We executed the following command for all the fields for the sake of having a clear collection.

```
db.bibliography.updateMany( {
    doi: { $eq: "" } }, {
    $unset: { doi: "" } } )

```

4. **Delete a group of documents**

   The command `deleteMany()` deletes a group of documents that match the condition given inside the specified filter. In this case, the documents that have as date one that occurs before the `1950` are considered obsolete, so we want to delete them. For this reason, it's enough to execute a check on the field `date`, that is of date type, so we have to specify the `ISODate` format and delete all the documents previous to `'1950-01-01T00:00:00.000Z'`.

```
db.bibliography.deleteMany( { date: { $lt: ISODate("1950-01-01T00
    :00:00.000Z") } } )

```

After executing the command successfully, `MongoDB` acknowledges the write operation and returns the number of documents that were eliminated from the collection.

```
> db.bibliography.deleteMany(
        {date: {$lt: ISODate('1950-01-01T00:00:00.000Z')}}
  )
< { acknowledged: true, deletedCount: 3151 }
```

5. **Update a group of documents**

   The command `updateMany` allows the modification of multiple documents at the same time. Since in the database some articles have `"Elsevier"` as `publisher`, and some others have `"Elsevier Ltd."`, and since both values represent the same publisher, we want to merge them. With the following command, all documents

that have value "Elsevier Ltd." in the field publisher are retrieved, then the value is changed in "Elsevier".

```
db.bibliography.updateMany( {
    publisher: "Elsevier Ltd." }, {
    $set: { publisher: "Elsevier" } } )

```

```
> db.bibliography.updateMany(
    {publisher: "Elsevier Ltd."},
    {$set: {publisher: "Elsevier"}}
  )
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1774,
    modifiedCount: 1774,
    upsertedCount: 0 }
```

10.2. Queries

1. **Papers published in a certain period on a specific type of publication**
   The query's goal is to find all the papers published in a particular time frame delimited by two dates (included) of a type of publication, such as a journal. More specifically, the search is conducted on papers published between January 2005 and December 2010 in journals. In order to be compliant with the data type of the date field, the date must be specified in a ISODate format. In the projection, we only show title and venue of a paper, that is the name of the Journal, and we suppress the field _id because not informative.

```
db.bibliography.find( {
    $and: [ {
        date: { $gte: ISODate('2005-01-01T00:00:00.000Z') } }, {
        date: { $lte: ISODate('2010-12-31T00:00:00.000Z') } }, {
        publication_type: "Journal" } ] }, {
    _id: 0,
    title: 1,
    venue: 1 } ).limit(5)

```

```
{ title: 'MESHMdl', venue: 'Pervasive and Mobile Computing' }
{ title: 'Detection and Recovery Techniques for Database Corruption',
  venue: 'IEEE Trans. Knowl. Data Eng.' }
{ title: 'General and Interval Type-2 Fuzzy Face-Space Approach to Emotion Recognition.',
  venue: 'IEEE T. Systems, Man, and Cybernetics: Systems' }
{ title: 'Existence of resolvable H-designs with group sizes 2, 3, 4 and 6',
  venue: 'Des. Codes Cryptography' }
{ title: 'Particle shape analysis of volcanic clast samples with the Matlab tool MORPHEO',
  venue: 'Computers & Geosciences' }
```

Performance: It's obvious to see that this query iterates through the values of a field to filter some documents. If the time range inserted to do the query is small, we can say that the query is a high selective query, that is, a query that returns only few documents from the whole set of scanned document. Selective queries are the ones that make the best usage of indexes, so we thought to do a comparison test of the performance of the query with and without an index on the date.

To have a more meaningful result of the test we removed the `db.collection.limit()` function, otherwise the automatic rearrangement of the queries would limit the query to stop as soon as it matches 5 correct instances. With this limit on the query, the load on the system would be so light that we would see a negligible difference between the query run with and without the indexes.

To get information about the query performance we use `db.collection.explain()`, which returns information about the query plan. Passing the `''executionStats''` argument to this methods makes it return an additional section where a set of useful performance marker are shown.

Results We first try running the query without an index on the field `date`. Using the method `explain` with the argument `''executionStats''` we get the following result (we show only a selection of the markers present in the `''executionStats''` section, the ones meaningful for this discussion):

```
executionStats:
 { executionSuccess: true,
   nReturned: 2476,
   executionTimeMillis: 360,
   totalKeysExamined: 0,
   totalDocsExamined: 56453,
```

MongoDB runs the query optimizer to choose the winning plan, executes the winning plan to completion, and returns statistics describing the execution of the winning

plan. In the reported results we can see different parameters that explain the query, and among these we focus our comparison especially on the following:

- `nReturned`: number of documents that match the query condition
- `executionTimeMillis`: total time in milliseconds required for query plan selection and query execution
- `totalKeysExamined`: number of index entries scanned
- `totalDocsExamined`: number of documents examined during query execution

We can clearly see that the the documents passing the filter are about 4% of the total, so we can say that this query is selective enough and can proceed to test it with the addition of an index. We'd like also to point out that the `totalKeysExamined` parameter has a value of 0, meaning that this query has not performed a search on indexes. Meanwhile, the `totalDocsExamined` shows that all the documents in the DB were examined. We proceed creating an index on the field `date` and we run the query again. This time the execution plan and stats return different values:

```
executionStats:
 { executionSuccess: true,
   nReturned: 2476,
   executionTimeMillis: 61,
   totalKeysExamined: 4521,
   totalDocsExamined: 4521,
   executionStages:
```

We can immediately notice that the execution time has gone down to 61ms from the 360ms of before, a performance gain of more than 5x. Since the indexes are stored in an ordered way the query executioner can iterate through the dates with a much faster binary search algorithm. This means we have to do a much smaller number of accesses to the keys of the database, as shown by the value of `totalKeysExamined` parameter in the figure. In this particular case we also have to access the DB (as shown by the `totalDcosExamined` parameter) because the `publication_type` field is not stored in a index, so we have to retrieve the value of this field from the document itself to complete the filtering condition [2]. Nonetheless, the much lower number of accesses to DB provided by the index on `date` is sufficient to experience a significant gain in performance.

[2]This type of query, where not all the field needed to compute the query are stored as indexes are called "not covered queries"

2. **Search a given word inside the title of papers written on books and published by a specific publisher**
   MongoDB supports query operations that perform a text search, but to enable this functionality is necessary to define a textual index that makes the search possible.

   To achieve more efficient execution of queries, we can define indexes, which are data structures that store useful information to speed up the search. When MongoDB executes a query without indexes, it has to evaluate the conditions on all the documents of the considered collection. If an index is defined and used, MongoDB can limit the number of analyzed documents and requires less time to compute the results. The following command is necessary for creating indexes, and it is essential before executing the following query.

```
db.bibliography.createIndex( {
    title: "text" } )

```

   Once the index is created, the query can be run. It returns the titles of the documents whose title presents the word `''software''` and that were published in a book by the publisher `"Elsevier"`.

```
db.bibliography.find( {
    $text: { $search: "software" },
    publication_type: "Book",
    publisher: "Elsevier" }, {
    title: 1 } ).limit(5)

```

3. **Visualize the common values between field of study and keywords in each document**

The following query extracts all those documents with at least one keyword as their field of study.

In particular, we project `title`, `fos`, `keywords`, and their sizes, and then we find all those documents that contain more than thirty keywords, and more than fifteen fields of study. We select such a huge number of fields of study and keywords because we are interested in papers that are involved in researches that can be important in many areas.

Finally, we project again `title` and the intersection set between keywords and fields of study to extract and show the ones that match.

```
db.bibliography.aggregate( [ {
    $project: {
        title: "$title",
        fos: "$fos",
        keywords:"$keywords",
        fosSize: { $size: "$fos" },
        keywordsSize: { $size: "$keywords" } } }, {
    $match: { $and: [ {
        keywordsSize: { $gt: 30 },
        fosSize: { $gt: 15 } } ] } }, {
    $project: {
        title: "$title",
        intersection: { $setIntersection: [
            "$fos",
            "$keywords" ] } } } ] )

```

Performance: Let's first consider the query just explained with the relative outcome of the execution, using `explain("executionStats")` in order to analyze its performance:

```
db.bibliography.aggregate( [ {
    $project: {
        title: "$title",
        fos: "$fos",
        keywords:"$keywords",
        fosSize: { $size: "$fos" },
        keywordsSize: { $size: "$keywords" } } }, {
    $match: { $and: [ {
        keywordsSize: { $gt: 30 },
        fosSize: { $gt: 15 } } ] } }, {
    $project: {
        title: "$title",
        intersection: { $setIntersection: [
            "$fos",
            "$keywords" ] } } } ] ).explain("executionStats")

```

The query firstly scrolls through all the documents, just selecting some fields that we want to project and calculating the size of the `fos` and of the `keywords`. In the execution this is shown by the fact that all the documents that are examined at this first step are also returned. This procedure has as `explain.executionStats executionTimeMillis` a value of `633 ms` and doesn't reduce the papers to work on. We can also see from the `explain.executionStats totalKeysExamined` that the number of indexes entries is zero because we don't use any indexing on the fields involved.

```
executionStats:
 { executionSuccess: true,
   nReturned: 53302,
   executionTimeMillis: 633,
   totalKeysExamined: 0,
   totalDocsExamined: 53302,
   executionStages:
    { stage: 'PROJECTION_DEFAULT',
      nReturned: 53302,
      executionTimeMillisEstimate: 45,
      works: 53304,
      advanced: 53302,
      needTime: 1,
      needYield: 0,
      saveState: 81,
      restoreState: 81,
      isEOF: 1,
      transformBy:
       { _id: true,
         title: '$title',
         fos: '$fos',
         keywords: '$keywords',
         fosSize: { '$size': [ '$fos' ] },
         keywordsSize: { '$size': [ '$keywords' ] } },
      inputStage:
       { stage: 'COLLSCAN',
         nReturned: 53302,
         executionTimeMillisEstimate: 0,
         works: 53304,
         advanced: 53302,
         needTime: 1,
         needYield: 0,
         saveState: 81,
         restoreState: 81,
         isEOF: 1,
         direction: 'forward',
         docsExamined: 53302 } } } },
```

Only after the scrolling, we execute a `match` that narrows down the papers and reduces the computation. As follows, we can see that this part of the query requires also `610 ms`, given the fact that has to analyze again all the documents, but after the execution only `9` matching papers are returned and shown with the projection.

```
{ '$match': { '$and': [ { keywordsSize: { '$gt': 30 } }, { fosSize: { '$gt': 15 } } ] },
  nReturned: 9,
  executionTimeMillisEstimate: 610 },
{ '$project':
   { _id: true,
     title: '$title',
     intersection: { '$setIntersection': [ '$fos', '$keywords' ] } },
  nReturned: 9,
  executionTimeMillisEstimate: 610 } ],
```

Given the fact that the query examined so far has to scroll over all the documents twice, we thought of the following alternative, that first executes the `match` part and then shows the intersection between the `fos` and the `keywords`, without using two projections.

```
db.bibliography.aggregate( [ {
    $match: { $expr: { $and: [ {
        $gt: [ { $size: "$keywords" }, 30 ] }, {
        $gt: [ { $size: "$fos" }, 15 ] } ] } } } , {
    $project: {
        title: "$title",
        intersection: { $setIntersection: [
            "$fos",
            "$keywords" ] } } } ] ).explain("executionStats")

```

As before the query starts examining all the documents of the collection and we have no indexes entries, not having any index on the involved fields. We can see some important differences from the previous procedure looking at mainly two fields of the `explain.executionStats` reported below. We can understand from the `explain.executionStats.executionTimeMillis` that the execution time is lower, requiring only `246 ms`, doing the filtering of the papers as the first action of the query. Afterward, the projection of the result required only `24 ms`, because the documents were already narrowed down. In fact, we can notice that the `match` examined all the collection, but returned only `9` matches, exactly like before, only now this operation was done before any other, so it improved the overall execution.

```
executionStats:
 { executionSuccess: true,
   nReturned: 9,
   executionTimeMillis: 246,
   totalKeysExamined: 0,
   totalDocsExamined: 53302,
   executionStages:
    { stage: 'PROJECTION_DEFAULT',
      nReturned: 9,
      executionTimeMillisEstimate: 24,
      works: 53304,
      advanced: 9,
      needTime: 53294,
      needYield: 0,
      saveState: 53,
      restoreState: 53,
      isEOF: 1,
      transformBy:
       { _id: true,
         title: '$title',
         intersection: { '$setIntersection': [ '$fos', '$keywords' ] } },
      inputStage:
       { stage: 'COLLSCAN',
         filter:
          { '$expr':
             { '$and':
                [ { '$gt': [ { '$size': [ '$keywords' ] }, { '$const': 30 } ] },
                  { '$gt': [ { '$size': [ '$fos' ] }, { '$const': 15 } ] } ] } },
         nReturned: 9,
         executionTimeMillisEstimate: 24,
         works: 53304,
         advanced: 9,
         needTime: 53294,
         needYield: 0,
         saveState: 53,
         restoreState: 53,
         isEOF: 1,
         direction: 'forward',
         docsExamined: 53302 } } },
```


In conclusion, this alternative query is more efficient than the previous one, requiring less time to compute. From this analysis we noticed that is better to execute the filtering part, the `match`, as soon as possible in the query, in order to reduce the number of documents we have to work on, speeding up the process since the operations are executed like a pipeline.

4. **Organizations that held more conferences in the same location**
   The following query matches all the papers that are of type `Conference`. We select only the events held by the `"IEEE Global Telecommunications Conference (Globecom)"` then for each author involved in the conference, we check the affiliation. After every step of the query, we obtain a new temporary collection of documents. For instance, with the `unwind` command, we deconstruct the array of `authors` from the input papers to output a new document for each element and, in this way, we create a much bigger dataset.

With this query, we are interested in knowing how many times an organization held a conference in a specific location through one of its members. Regrouping the data by conference location and affiliation of the author involved, we can count the number of times the organization took part in an event held there. Finally, the result is sorted by the number of engagements and limited only to the first more active organizations.

```
db.bibliography.aggregate( [ {
    $match: { $and: [ {
        publication_type: { $eq: "Conference" } }, {
        venue: { $eq: "IEEE Global Telecommunications Conference (
    Globecom)" } } ] } }, {
    $unwind: "$authors" }, {
    $group: {
        _id : {
            location: "$location",
            organization: "$authors.org" },
        locationPerOrganization: { $sum: 1 } } }, {
    $project: {
        location: "$location",
        author: "$organization",
        locPerOrgConference: "$locationPerOrganization" } }, {
    $sort: { locPerOrgConference : -1 } }, {
    $limit: 5 } ] )

```

5. **Papers with a pattern structure**

   With this query, we count the number of documents that satisfy a given pattern specified in the `$match` part of the query. In this example, we are searching for papers with four sections, and at least one must have four paragraphs, four figures, and one subsection with just one paragraph. Using `$size`, it's possible to check the

dimension of an array, while using `$elemMatch` is possible to define the criteria that at least one document in an array must satisfy. Finally, through the dot notation, we can access a field of an embedded document.

```
db.bibliography.aggregate( [ {
    $match: { $and: [ {
        sections: { $size: 4 } }, {
        sections: { $elemMatch: {
            paragraphs: { $size: 4 },
            figures: { $size:4 },
            subsections: { $size: 1 },
            "subsections.paragraphs": { $size: 1 } } } } ] } }, {
    $group: {
        _id: true,
        count: { $sum: 1 } } } ] )

```

```
> db.bibliography.aggregate([
  {$match:
    {"$and":
      [{"sections": {$size: 4}},
       {"sections": {$elemMatch: {"paragraphs": {$size: 4},
                                  "figures": {$size:4},
                                  "subsections": {$size: 1},
                                  "subsections.paragraphs": {$size: 1}}
                     }
      }]
    }
  },
  {$group:
    {"_id": true,
     "count": {"$sum": 1}
    }
  }
  ])
< { _id: true, count: 304 }
```

6. **Number of papers published by the current staff of an organization**
   What we want to achieve with this query is to know how many papers have been published by the researchers working at a certain organization. The output list is ordered in descending order, showing which are the organizations whose researchers have published the most. Notice that a paper published by authors from different organizations is counted for all the organizations involved. In the first part of the query we unwind the documents by `authors` and do a filtering of inconsistent data, since the focus of this query is to know which are the organizations that have published the most. Secondly, we proceed by grouping documents by the author

affiliation (`authors.org`) and the paper itself. This operation is needed to filter out the documents referring to the same paper and written by authors that are affiliated with the same organization. If we hadn't done this, the same paper would have been counted multiple times, for all the author who participated in writing it and are part of the same organization, inflating that organization's contribution. Notice that in the result an organization with name `"Corresponding author"` appears, probably this value has been used by the maintainer of the DB to denote independent authors or authors for whom the affiliated organization was not known.

```
db.bibliography.aggregate( [ {
    $unwind: "$authors" }, {
    $match: { $and: [ {
        "authors.org": { $ne: null } }, {
        "authors.org": { $ne: '' } } ] } }, {
    $group: {
        _id: {
        org: "$authors.org",
        p_id: "$_id" } } }, {
    $group: {
        _id: "$_id.org",
        papers: { $sum: 1 } } }, {
    $sort: { papers: -1 } }, {
    $limit: 5 } ] )

```

```
{ organization: 'Corresponding author.', numberOfPapers: 318 }
{ organization: 'IEEE', numberOfPapers: 73 }
{ organization: 'Microsoft Research', numberOfPapers: 61 }
{ organization: 'Carnegie Mellon University, Pittsburgh, PA',
  numberOfPapers: 55 }
{ organization: 'Carnegie Mellon University',
  numberOfPapers: 53 }
```

7. **Number of pages published on Journals by authors affiliated with a certain organization**

   This query is similar to the one above, but here we want to retrieve which organizations have contributed the most in writing papers published in journals. More specifically, we sum the number of pages written in a specific journal by authors who are currently affiliated with that specific organization. This objective is obtained by first unwinding the authors and filtering them to remove some inconsistent data. Secondly, we proceed by grouping documents by the author affiliation (`authors.org`) and the paper itself (the other fields of `venue` and `pages` do not con-

tribute to the grouping because they are a single field of the paper document). This operation achieve the same goal as the one described in the query before. In the second `$group` stage, we group the documents by organization and journal, and we have an accumulator to sum all the pages written by the staff of that organization in that particular journal. Finally, the last stages are only for projection and pretty printing the output.

```
db.bibliography.aggregate( [ {
    $unwind: "$authors" }, {
    $match: { $and: [ {
        "authors.org": { $ne: null } }, {
        "authors.org": { $ne: '' } }, {
        publication_type: { $eq: "Journal" } }, {
        $expr: { $lte: [
            "$page_start",
            "$page_end" ] } } ] } }, {
    $addFields: { pages: { $subtract: [
        "$page_end",
        "$page_start" ] } } }, {
    $group: { _id: {
        org: "$authors.org",
        paperId: "$_id",
        journal: "$venue",
        pages: "$pages" }, } }, {
    $group: {
        _id: {
            org: "$_id.org",
            journal: "$_id.journal" },
        pages: { $sum: "$_id.pages" } } }, {
    $sort: { pages: -1 } }, {
    $project: {
        organization: "$_id.org",
        journal: "$_id.journal",
        numberOfPages: "$pages" } }, {
    $project: { _id: 0 } }, {
    $limit: 5 } ] )

```

```
{ organization: 'Dept. of Commun. Eng., Nat. Central Univ., Taoyuan, Taiwan',
  journal: 'Selected Areas in Communications, IEEE Journal  ',
  numberOfPages: 1519 }
{ organization: 'Dept. of Electr. & Comput. Eng., Univ. of Maryland, College Park, MD, USA#TAB#',
  journal: 'Selected Areas in Communications, IEEE Journal  ',
  numberOfPages: 1519 }
{ organization: 'Corresponding author.',
  journal: 'Applied Mathematics and Computation',
  numberOfPages: 596 }
{ organization: 'university of edinburgh',
  journal: 'Electronic Notes in Theoretical Computer Science',
  numberOfPages: 418 }
{ organization: 'DI École Normale Supérieure, France',
  journal: 'Foundations and Trends® in Computer Graphics and Vision',
  numberOfPages: 200 }
```


8. **Couples of field of study and keyword which appear more frequently within the papers**

   The query returns the association between fields of study and keywords which are more present within the database and how many times they appear together. An initial filtering is done in order to eliminate the papers which doesn't contain any keyword or any field of study. The two clauses for doing the filtering can be modified increasing the required number of keywords or the required number of fields of study. This kind of change can be interesting in order to understand which fields of study and keywords become more relevant when the subset of considered paper is different. The results are ordered by the number of occurrences of the couples `field of study - keyword` by decreasing order. This can be done by the operator `"$sort"`. We filter out the couples of fields of study and keywords with less then 100 in order to get rid of some misleading information due to not coupled elements. The threshold can be adjusted.

```
db.bibliography.aggregate( [ {
    $match: {
        $expr: { $gte: [ {
            $size: "$fos" },
            0 ] } } }, {
    $match: {
        $expr: { $gte: [ {
            $size: "$keywords" },
            0 ] } } }, {
    $unwind: { path: "$fos" } }, {
    $unwind: { path: "$keywords" } }, {
    $group: {
        _id: {
```

```
            fieldOfStudy: "$fos",
            keyword: "$keywords" },
        keywordsCount: { "$sum": 1 } } }, {
    $match: { "keywordsCount": { $gte: 100 } } }, {
    $sort: { keywordsCount: -1 } }, {
    $limit: 5 }, {
    $project: {
        fieldOfStudy: "$fos",
        keyword: "$keywords",
        score: "$keywordsCount" } } ] )

```

```
> db.bibliography.aggregate([
    {"$unwind": {"path": "$fos"}},
    {"$unwind": {"path": "$keywords"}},
    {"$group": {
      "_id": {
        "fos": "$fos",
        "keyword": "$keywords"
      },
      "keywordsCount": {"$sum": 1}
    }},
    {"$sort": {"keywordsCount": -1}},
    {"$limit": 5},
    {"$project": {"fieldOfStudy": "$fos", "keyword": "$keywords", "score": "$keywordsCount"}}
    ])
< { _id: { fos: 'computer science', keyword: 'data mining' },
    score: 1066 }
  { _id: { fos: 'computer science', keyword: 'internet' },
    score: 960 }
  { _id: { fos: 'computer science', keyword: 'computer science' },
    score: 844 }
  { _id: { fos: 'computer science', keyword: 'real time' },
    score: 798 }
  { _id:
     { fos: 'artificial intelligence',
       keyword: 'feature extraction' },
    score: 643 }
```

9. **Papers that reference old papers with same publisher**
   In the beginning, we filter the documents retrieving only those that have a `publisher` then, we perform a join between papers and their references using `$lookup` with the condition that the `_id` of the paper must be in the `references` field of the other one. In the `$let` command we specify the variables to use in the pipeline that are related to the outer document and that are accessed using `$$`. Then we require both papers to have the same publisher and that the referenced article was published at least 10 years before using `$dateDiff`. The `_id`, `title`, `publisher`, and `date` of the referenced papers that satisfy these conditions are added to the field called `referencedPapers` in the outer document. Lastly, we filter documents keeping only those with at least one joined document, we project only a few fields to make the answer more readable and to check that it is correct, and we limit the number of returned papers.

```
db.bibliography.aggregate( [ {
    $match: { publisher: { $exists: true } } }, {
    $lookup: {
        from: "bibliography",
        let: {
            pub: "$publisher",
            ref: "$references",
            d: "$date" },
        pipeline: [ {
            $match: { $expr: { $and: [ {
                $in: [
                    "$_id",
                    "$$ref" ] }, {
            $eq: [
                "$publisher",
                "$$pub" ] }, {
            $gte: [ {
                $dateDiff: {
                    startDate: "$date",
                    endDate: "$$d",
                    unit: "year" } },
                10 ] } ] } } }, {
        $project: {
            _id: 1,
            title: 1,
            publisher: 1,
            date: 1 } } ],
        as: "referencedPapers" } }, {
    $match: { "referencedPapers.0": { $exists: true } } }, {
    $project: {
        _id: 1,
        title: 1,
        publisher: 1,
        date: 1,
        referencedPapers: 1 } }, {
    $limit: 3 } ] )

```

```
{ _id: '53e997ddb7602d9701fd5406',
  title: 'An Asymmetric Edge Adaptive Filter for Depth Generation and Hole Filling in 3DTV',
  publisher: 'Springer London',
  date: 1987-05-11T05:40:26.000Z,
  join:
   [ { _id: '53e99866b7602d97020a0cb7',
       title: 'Depth-image-based rendering for 3DTV service over T-DMB',
       publisher: 'Springer London',
       date: 1959-09-30T21:11:50.000Z } ] }
{ _id: '53e997e3b7602d9701fd8218',
  title: 'A Programming Language for Deriving Hypergraphs',
  publisher: 'Springer New York',
  date: 2016-04-24T22:35:04.000Z,
  join:
   [ { _id: '53e9982cb7602d970204e40f',
       title: 'Programmed Graph Grammars',
       publisher: 'Springer New York',
       date: 1975-08-21T00:47:13.000Z } ] }
{ _id: '53e997e3b7602d9701fd95b3',
  title: 'Constructing and Exploring Composite Items Using Max-valid Bundles',
  publisher: 'Elsevier',
  date: 1993-11-05T15:44:58.000Z,
  join:
   [ { _id: '53e99821b7602d970204096d',
       title: 'Summarizing relational databases',
       publisher: 'Elsevier',
       date: 1945-10-16T20:17:40.000Z } ] }
```


The `$lookup` can be very expensive from a computational point of view and require much time, so to speed the process up it is convenient, when possible, to reduce the input data to the join phase by filtering the documents. This can be done by searching for a specific author, title, publisher, or something meaningful in a `$match` condition before the `$lookup`. This is possible since all the operations in a query are executed on the database as a pipeline.

10. **Recommended books given one book read**

    The query returns the book considered the nearest to the given one, in this case, `"Pattern Recognition Letters"`. One book is suggested if it has a field of study in common with the book `"Pattern Recognition Letters"`. The similarity between books is then evaluated in terms of the number of equal keywords associated with the papers belonging to both considered books.

```
db.bibliography.aggregate( [ {
    $match: { publication_type: "Book" } }, {
    $unwind: { path: "$fos" } }, {
    $unwind: { path: "$keywords" } }, {
```

```
    $lookup: {
        from: "bibliography",
        localField: "fos",
        foreignField: "fos",
        as: "otherPaper" } }, {
    $match: { venue: "Pattern Recognition Letters" } }, {
    $unwind: { path: "$otherPaper" } }, {
    $match: { $expr: { $ne: [
        "$_id",
        "$otherPaper._id" ] } } }, {
    $match: { $expr: { $ne: [
        "$venue",
        "$otherPaper.venue" ] } } }, {
    $unwind: { path: "$otherPaper.keywords" } }, {
    $match: { $expr: { $eq: [
        "$keywords",
        "$otherPaper.keywords" ] } } }, {
    $group: {
        _id: { suggestedBook: "$otherPaper.venue" },
        keywordsCount: { $sum: 1 } } }, {
    $sort: { keywordsCount: -1 } }, {
    $limit: 10 }, {
        $project: {
            title: "$suggestedBook",
            similarity: "$keywordsCount" } } ] )

```

```
{ _id: { suggestedBook: 'Pattern Recognition' },
  similarity: 58 }
{ _id: { suggestedBook: 'ICIP' }, similarity: 28 }
{ _id: { suggestedBook: 'Neurocomputing' }, similarity: 20 }
{ _id: { suggestedBook: 'Expert Syst. Appl.' }, similarity: 17 }
{ _id: { suggestedBook: 'Inf. Sci.' }, similarity: 13 }
{ _id: { suggestedBook: 'ISNN (2)' }, similarity: 13 }
{ _id: { suggestedBook: 'IEEE J. Biomedical and Health Informatics' },
  similarity: 11 }
{ _id: { suggestedBook: 'ICASSP' }, similarity: 10 }
{ _id: { suggestedBook: 'IEEE Trans. Pattern Anal. Mach. Intell.' },
  similarity: 10 }
{ _id: { suggestedBook: 'Neural Computing and Applications' },
  similarity: 10 }
```

11. **Authors who have referenced themselves the most within their papers**
    The query returns the authors who have referenced themselves the most. An author does an auto-reference when the considered paper presents a reference to another paper the author has wrote. The results are the name of the authors and the number

of times they have made a self-reference, the output is ordered using this number.

```
db.bibliography.aggregate( [ {
    $unwind: { path: "$references" } }, {
        $lookup: {
            from: "bibliography",
            localField: "references",
            foreignField: "_id",
            as: "referencedPaper" } }, {
    $match: { $expr: { $gt: [ {
        $size: "$referencedPaper" },
        0 ] } } }, {
    $unwind: { path: "$referencedPaper" } }, {
        $match: { $expr: { $ne: [
            "$_id",
            "$otherPaper._id" ] } } }, {
    $match: { "authors": { $exists: true } } }, {
    $match: { "referencedPaper.authors": { $exists: true } } }, {
    $unwind: { path: "$authors" } }, {
    $unwind: { path: "$referencedPaper.authors" } }, {
    $match: { $expr: { $eq: [
        "$authors",
        "$referencedPaper.authors" ] } } }, {
    $group: {
        _id: {
            authorId: "$authors._id",
            authorName: "$authors.name" },
        aut: { $sum: 1 } } }, {
    $sort: { "aut": -1 } }, {
    $limit: 10 }, {
    $project: {
        "_id": 0,
        "name": "$_id.authorName",
        "autoreference": "$aut" } } ] )

```

```
< { name: 'K. V. S. Prasad', autoreference: 4 }
  { name: 'Cheng-Lin Liu', autoreference: 4 }
  { name: 'Sam Toueg', autoreference: 4 }
  { name: 'Qiu-Feng Wang', autoreference: 4 }
  { name: 'Yingxu Wang', autoreference: 4 }
  { name: 'Kathleen Romanik', autoreference: 4 }
  { name: 'Martín Abadi', autoreference: 3 }
  { name: 'Michal Parnas', autoreference: 3 }
  { name: 'Kazuhisa Makino', autoreference: 3 }
  { name: 'Karel Hrbacek', autoreference: 3 }
```

Part IV

Spark

11 | Introduction to the problem

12 | Dataset structure

13 | Data upload

14 | Commands and queries