Project Work Part 2 – Data Design and Modelling

David Alarcórn Sepehr Beheshti Claudio Maggioni
Lorenzo Martingnetti

December 21, 2022

Contents

1 Introduction

This project is about storing the data from the open access (OA) CC-BY articles from across Elsevier’s journals repository[1] in a documental database implemented using MongoDB. To achieve this result, we have to understand the business domain of a publication repository, translate that knowledge in an document model, and finally implement a MongoDB database. This consists of downloading the data in the database, design a way to import this data, define a document Model

[1] `https://elsevier.digitalcommonsdata.com/datasets/zm33cdndxs/2`

representing the application domain, and finally define some useful *MongoDB* commands to both query and update the data.

2 Requirements specification

The model must take into consideration the internal structure of a scientific paper. The tracked data are the following:

- **Title**: the title of the paper;
- **Abstract**: for the purpose of this assignment, we consider an abstract without any formatting on the text (i.e., just a string);
- **Authors**: authors are tracked both as self-standing entities (storing author-related characteristics), as well as authors mentioned on a specific paper (keeping track of the relevant information about the state of an author when the paper was published);
- **Keywords**: metadata used to tag the paper;
- **Publication details**: on which *journal* the paper is published, in which volume, reference number, on which date and the range of pages containing the paper;
- **Sections**: the internal structure of a paper is divided in sections, each one with a title and a content, which in turn may be some text, a subsection, a reference to another paper, or a figure.
- **Figures**: each figure has a label, a caption, the number of the page containing the figure, and an optional image URL;
- **References**: each reference is defined by its reference number, the title and (optional) id of the referenced paper, the name of the authors involved, the journal, its (optional) id, volume and number on which the paper is published.

Note: a figure may be a table in general, but to model a table is outside the scope of this assignment.

3 The Document Model

To represent the document model, we use *TypeScript*[2] syntax to provide an easy to read and typed definition of collections and sub documents. The document model we define is showed in listing 1.

```
// not collections, just type definitions
type Section = { title: string, content: Content[] }
type Content = Section /* a sub section */
               | string /* text, paragraph content */
               | { reference: number } /* bibliography reference */
               | { label: string } /* figure reference */

interface Paper {
    _id: ObjectID,
    title: string,
    abstract: string,
    authors: [{
        affiliation: string, // The university name
        email: string,
        name: string,
        authorId: ObjectID
```

[2]https://www.typescriptlang.org/

```
    }],
    keywords: string[],
    publicationDetails: {
        journalRef: ObjectID,
        journal: string,
        volume: string,
        number: string,
        date: Date,
        pages: {
            start: number,
            end: number
        },
    },
    content: Section[],
    figures: [{
        page: number,
        label: string,
        caption: string,
        imageURL?: string
    }],
    references: [{
        referenceNumber: number,
        paperId?: ObjectID,
        title: string,
        author: [{
            name: string,
            authorId?: ObjectID
        }],
        journal: string,
        journalId?: ObjectID,
        volume: string,
        number: string
    }]
}

interface Author {
    _id: ObjectID,
    name: string,
    email: string,
    affiliation: string,
    bio: string
}

interface Journal {
    _id: ObjectID,
    name: string,
    volumes: [{
        name: string,
        papers: ObjectID[]
    }]
}
```

Listing 1: Document model used to represent the application domain. The *ObjectID* type refers to the MongoDB document references. "?:" denotes an optional property.

Each TypeScript interface is translated into a MongoDB collection.

The attributes related to **authors** stored inside the *Paper* interface reflect the data at the time the paper was published. On the other hand, the data stored in the *Author* interface reflect the most up to date information available. The *affiliation* of an author is the university associated to the author when the paper was published.

Inside the **publication details**, the *volume* is some edition published by a journal, while the *number* is a reference for the paper inside the volume, useful to find it instead of using the page number. The *pages* field is composed of the first and the last page occupied (even partially) by the paper.

The **content** of a paper is a sequence of *Section* elements: each element can be a production of the grammar defined at the top of the model with *type Section* and *type Content*. Again, possible realizations are text, references, figures, or a subsection.

For **figures**, we assume that a single figure is contained is a single page: the *page* field is a simple number.

Regarding the **references**, both *paperID* and *authorID* are optional. This allows the model to store there information about paper and authors which are not stored in the database (the ID is related to the database implementation).

Journals information are duplicated in a self-standing collection in order to speed up queries. Here, a journal (with name and ID) publishes a sequence of volumes, each one having a name and containing a sequence of papers identified by their ID. This implies that if the journal is registered in the *Journal* collection inside the database, we have all the contained papers stored inside the database as well.

4 Data import

In order to import the Elsevier's journal repository dataset we develop a Python import script. The script loads each paper JSON file in the dataset, it converts its data structure to make it comply with the document model we define, and it writes all the data in a MongoDB database.

The script populates a database named `ddm` and in it the three collections *papers*, `authors` and `journals`. Author and Journal data is extracted from the metadata of each paper. The script makes use of a data faking library named *Faker* to generate author bios, journal names (as only the ISSN of them is included in the dataset) and volumes, author affiliation, and publication date. Additionally, figure metadata and references to figures and bibliography in the paper content were not provided by the dataset and are generated randomly too. The library *PyMongo* is used to interact with MongoDB and save the dataset.

Below are listed the sources of the script. The sources include comments providing additional insight in how the import process actually works.

```python
import json
import sys
import random
import os
from faker import Faker
from pymongo import MongoClient

# this script requires the following 'pip' dependencies to be installed:
# - faker
# - pymongo
```

```python
fake = Faker()

# For affiliations, we generate 500 university names by concatenating the string
# 'University of' with a random locality name provided by the Faker library.
# Affiliations are assigned at random to authors, we chose 500 since we felt it
# provided sufficient overlap in the random assignment.
fake_unis = []
for i in range(0, 500):
    (_, _, loc, _, _) = fake.location_on_land()
    fake_unis.append("University of " + loc)

# Here we cache authors referenced in each paper for later insertion in a
# separate collection
authors = {}
def fake_author(name: str, email: str) -> str:
    if name not in authors:
        # We use the author name, for lack of a better field in the dataset,
        # as a unique identifier for authors during the import process. This
        # means that two papers with an author with the same name will be
        # referenced to the same author document in the MongoDB database.
        # Should another dataset include the ORCID of authors, we could use
        # that in this step to assure the references are correct.
        authors[name] = {
            "name": name,
            "email": email,
            "affiliation": fake_unis[random.randint(0, len(fake_unis) - 1)],
            "bio": fake.paragraph(nb_sentences=5)
        }
    return authors[name]

# Here we cache journals referenced in each paper for later insertion in a
# separate collection
journals = {}
def fake_journal(issn: str, volume: str, paper_doi: str) -> str:
    if issn not in journals:
        # For journal names, we select a random company mission slogan using the
        # Faker 'bs' generator and then we extract the last two words. As we have
        # the ISSN of each journal from the paper metadata, we do not need to
        # constrain the number of them as the authors but we simply generate fake
        # metadata for each new ISSN found.
        journals[issn] = {
            "issn": issn,
            "name": " ".join(fake.bs().split(" ")[1:]).capitalize(),
            "volumes": [{
                "name": volume,
                "papers": [paper_doi]
            }]
        }
    else:
        # When the metadata for the journal with this paper's ISSN was already
        # generated, we need to update references to volumes and papers in
        # volumes to include this paper too.
        not_found = True
        for v in journals[issn]["volumes"]:
            if v["name"] == volume:
                not_found = False
                v["papers"].append(paper_doi)
                break
        if not_found:
            journals[issn]["volumes"].append({
                "name": volume,
                "papers": [paper_doi]
            })

    return journals[issn]

# This function, for a str of "a.b.c", will return the value of
# obj["a"]["b"]["c"] or the default value provided.
def getProp(obj: dict, props: str, default = None):
    target = obj
    if target is None:
        return default
    for prop in props.split("."):
        if prop not in target:
            return default
        target = target[prop]
    return target

# The following two functions allow us to convert between the data structure
# for the document body provided by the dataset and the one defined in our
# document model. The dataset provides a list of sentences in the paper, by
# also providing their position as a character offset and the sections they
# are contained in (their 'parents' as per the dataset schema). Additionally
# all section 'ancestors' are provided for each sentence. E.g. if the sentence
# is placed in sub-sub-section 1.2.3, data for section 1, sub-section 1.2 and
# sub-sub-section 1.2.3 is provided. We convert this 'flat' data structure
# in the recursive one specified in the document model, allowing sections to
# directly contain sub-section documents or paragraphs of text.

def save_sentence(body: dict, parents: [dict], sentence: str):
    target = body
    for p in parents:
        if p["id"] not in target["sections"]:
            target["sections"][p["id"]] = {
                "title": p["title"],
                "content": "",
                "sections": {},
            }
        target = target["sections"][p["id"]]
    target["content"] += sentence + " "

def transform_section(sec: dict, figures: [dict], references: [dict]) -> dict:
    arr = []
    ks = []
    for k in sec["sections"].keys():
        ks.append(k)
    ks.sort()
    for k in ks:
        arr.append(transform_section(sec["sections"][k], figures, references))
    if "title" not in sec:
        return arr

    # Here we randomly add references to figures and bibliography to the paper's
    # content in the format described in the document model.
    content = []
    if random.randint(0, 4) == 0 and len(figures) > 0:
        content += [{
            "label": figures[random.randint(0, len(figures)-1)]["label"]
        }]
    if "content" in sec and sec["content"] != "":
        content += [sec["content"]]
    if random.randint(0, 4) == 0 and len(references) > 0:
        content += [{ "reference": random.randint(1, len(references)) }]

    content += arr

    if len(content) > 0 and isinstance(content[-1], list) and \
        len(content[-1]) == 0:
        del content[-1]

    return {
        "title": sec["title"],
        "content": content
    }

def get_author_name(author: dict) -> str:
    first = "" if author["first"] is None else author["first"]
    last = "" if author["last"] is None else author["last"]
    return (first + " " + last).strip()

# This function is the entrypoint for the conversion of a paper object coming
# from the dataset to a paper object following our document model.
def json_to_paper(filename: str, jsonObj: dict) -> dict:
    paper = {}

    paper["title"] = getProp(jsonObj, "metadata.title")
    paper["abstract"] = getProp(jsonObj, "abstract")
    paper["doi"] = getProp(jsonObj, "metadata.doi")

    authors = []
    for author in getProp(jsonObj, "metadata.authors", []):
        email = getProp(author, "email")

        author = fake_author(get_author_name(author), email)
        authors.append({
            "email": author["email"],
            "name": author["name"],
            "affiliation": author["affiliation"]
        })

    paper["authors"] = authors
    paper["keywords"] = getProp(jsonObj, "metadata.keywords")

    publicationDetails = {}
    publicationDetails["issn"] = getProp(jsonObj, "metadata.issn")

    # We generate a year-month-day publication date and, when the
    # journal volume metadata is missing, we generate a volume
    # name from the generated publication year.
    date = fake.date_object()
    volume = getProp(jsonObj, "metadata.volume")
    if volume is None:
        volume = str(date.year) + " Issue"

    journal = fake_journal(
        publicationDetails["issn"],
        volume,
        getProp(jsonObj, "metadata.doi")
    )

    publicationDetails["journal"] = journal["name"]
    publicationDetails["volume"] = volume
    publicationDetails["number"] = random.randint(1, 99)
    publicationDetails["date"] = date.isoformat()
    publicationDetails["pages"] = {
        "start": getProp(jsonObj, "metadata.firstpage"),
        "end": getProp(jsonObj, "metadata.lastpage")
    }
    paper["publicationDetails"] = publicationDetails

    # All the figure metadata is generated randomly as it is missing from
    # the dataset. Captions are built using the 'paragraph' Faker generator
    # and image URLs are built using the Faker 'image_url' generator.
    figures = []
    for i in range(0, random.randint(3, 15)):
        figures.append({
            "page": random.randint(1, 10),
            "label": "fig" + str(i),
            "caption": fake.paragraph(nb_sentences=1),
            "imageURL": fake.image_url()
        })
    paper["figures"] = figures

    i = 0
    references = []
    for key, value in getProp(jsonObj, "bib_entries", {}).items():
        if value is None:
            continue

        i += 1
        ref = {
            "referenceNumber": i,
            "doi": getProp(value, "doi"),
            "title": getProp(value, "title"),
            "authors": [],
            "issn": getProp(value, "issn"),
            "volume": getProp(value, "volume"),
            "year": getProp(value, "year")
        }

        for author in getProp(value, "authors", []):
            ref["authors"].append({ "name": get_author_name(author) })
        references.append(ref)
    paper["references"] = references

    body = {
        "sections": {}
    }

    l = getProp(jsonObj, "body_text", [])
    l.sort(key=lambda x: x["startOffset"])
    for e in l:
        parents = []
        for p in getProp(e, "parents", []):
            parents.append(p)

        parents.append({ "id": e["secId"], "title": e["title"] })
        save_sentence(body, parents, e["sentence"])

    paper["content"] = transform_section(body, figures, references)
    return paper

# Here the connection string to MongoDB is defined. We all use local
# MongoDB instances with anonymous access for the scopes of this
# assignment.
mongo_conn_str = "mongodb://localhost:27017"

# To invoke the script two command line parameters are defined:
# - (required) path to the folder where JSON files of articles from
# %*\hspace{2.2cm}*) the dataset are stored;
# - (optional) a limit on the number of papers to import. -1 imports
# %*\hspace{2.2cm}*) all the dataset and is the default.
def main():
    source_folder: str = sys.argv[1]
    if len(sys.argv) > 2:
        limit: int = int(sys.argv[2])
    else:
        limit: int = -1
```

```python
mongo = MongoClient(mongo_conn_str)
db = mongo["ddm"]

db["papers"].drop()
db["authors"].drop()
db["journals"].drop()

# During the import process we cache each object's MongoDB '_id' field
# to later be able to perform updates inserting document references.
# For the first insertion, when '_id's are still unavailable, one
# field of each document that is guaranteed to be unique is used.
# This fields are:
#
# - DOI for papers;
# - ISSN for journals;
# - name for authors (for a lack of a better field in the dataset).
#
# The following variables are dictionaries mapping the values of these
# fields to the MongoDB '_id' values used to insert their respective
# document, and are later used to insert a proper reference.
paper_ids: dict[str, ID] = {}
author_ids: dict[str, ID] = {}
journal_ids: dict[str, ID] = {}

i = 0
j = 0
for filename in os.listdir(source_folder):
    if filename.endswith(".json"):
        jsonObj = {}
        with open(source_folder + "/" + filename, 'r') as jsonFile:
            jsonStr = "".join(jsonFile.readlines())
            d = json.JSONDecoder()
            jsonObj = d.decode(jsonStr)

        if getProp(jsonObj, "metadata.issn") is None or \
            getProp(jsonObj, "metadata.doi") is None:
            # We defensively skip papers with no journal ISSN or
            # paper DOI. We later determined that no papers in the
            # dataset match this conditions by analyzing the
            # import script output.
            j += 1
            continue

        paper = json_to_paper(filename, jsonObj)

        x = db["papers"].insert_one(paper)
        paper_ids[paper["doi"]] = x.inserted_id

        i += 1
        if i % 100 == 0:
            print("Papers processed: ", i)
        if j % 100 == 0 and j > 0:
            print("Papers skipped: ", j)
        if limit > 0 and i == limit:
            break

print("Papers skipped: ", j)

# As authors and journals are collections of documents derived from
# paper metadata, we accumulate values when parsing each paper in
# global variables defined in the first lines of the script. Here we
# save the gathered data in MongoDB.

i = 0
for name, author in authors.items():
    x = db["authors"].insert_one(author)
    author_ids[name] = x.inserted_id
    i += 1
    if i % 1000 == 0:
        print("Authors processed: ", i)

i = 0
for issn, journal in journals.items():
    x = db["journals"].insert_one(journal)
    journal_ids[issn] = x.inserted_id
    i += 1
    if i % 100 == 0:
        print("Journals processed: ", i)

# The following lines perform a single update operation for each paper, author
# and journal document by replacing the interim 'reference fields' with proper
# '_id' references that were cached during insertion.

i = 0
for paper in db["papers"].find():
    mongo_filter = { "_id": paper["_id"] }
    update = {}
    mongo_update = { "%*\$*)set": update }

    issn = getProp(paper, "publicationDetails.issn", "")
    if issn in journal_ids:
        update["publicationDetails.journalRef"] = journal_ids[issn]

    references = getProp(paper, "references", [])
    for ref in references:
        if ref["doi"] in paper_ids:
            ref["paperId"] = paper_ids[ref["doi"]]

        for author in ref["authors"]:
            name = author["name"]
            if name in author_ids:
                author["authorId"] = author_ids[name]
            if ref["issn"] in journal_ids:
                ref["journalId"] = journal_ids[issn]
        update["references"] = references

        authors_loc = getProp(paper, "authors", [])
        for author in authors_loc:
            name = author["name"]
            if name in author_ids:
                author["authorId"] = author_ids[name]
        update["authors"] = authors_loc

        db["papers"].update_one(mongo_filter, mongo_update)

        i += 1
        if i % 100 == 0:
            print("Papers updated with refs: ", i)

    i = 0
    for journal in db["journals"].find():
        mongo_filter = { "_id": journal["_id"] }
        update = {}
        mongo_update = { "%*\$*)set": update }

        volumes = getProp(journal, "volumes", [])
        for volume in volumes:
            v_papers = []
            for p in volume["papers"]:
                v_papers.append(paper_ids[p])
            volume["papers"] = v_papers
        update["volumes"] = volumes

        db["journals"].update_one(mongo_filter, mongo_update)
        i += 1
        if i % 100 == 0:
            print("Journals updated with refs: ", i)

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

Listing 2: The Python import script.

4.1 Data Imported

As the script described in the previous section is capable of importing the entire Elsevier's journal repository dataset we decide to import it entirely. The resulting MongoDB database has approximately 40.000 documents in the `papers` collection, 206.000 documents in the `authors` collection and 1.800 documents in the `journals` collection.

5 Queries

In this section we define some queries over the database using Python and the PyMongo library. To run these queries on a local database with no credentials, the following preamble is necessary:

```
import time
from pymongo import MongoClient

mongo_conn_str = "mongodb://localhost:27017"
mongo = MongoClient(mongo_conn_str)
db = mongo["ddm"]
```

To check for performance, we measure the time each PyMongo operation takes to execute by defining the following two Python functions implementing a stopwatch of sorts:

```
def start_the_time():
    global start_time
    start_time = time.time()

def end_the_time():
    print("--- %s seconds ---" % (time.time() - start_time))
```

5.1 Add new author to Authors collection

This query inserts a new document to the authors collection that includes: name, email, affiliation and bio of the author.

```
new_author = {
    "name": "Perico de los Palotes",
    "email": "perico.palotes@usi.ch",
    "affiliation": "Università della Svizzera Italiana",
    "bio": "Perico de los Palotes is a renowed scholar in the field of relational" +
        "databases."
}
start_the_time()
new_author_id = db["authors"].insert_one(new_author).inserted_id
end_the_time()
```

The query takes 2.9 ms to execute.

5.2 Add new Journal to the Journals Collection

This query inserts a new document in the *journals* collection by including: ISSN, name and an empty list of volumes.

```
new_journal = {
    'issn': '89012388',
    'name': 'Advanced Topics on Databases',
    'volumes': []
}
start_the_time()
new_journal_id = db["journals"].insert_one(new_journal).inserted_id
end_the_time()
```

The query takes 1.1 ms to execute.

5.3 Add an Author to a Document

This query modifies the paper with DOI *10.1016/j.nicl.2018.08.028* by adding “John Doe” as an author.

```
query = {
    "doi": "10.1016/j.nicl.2018.08.028",
}

update = {
    "$push": {
        "authors": {
            "name": "John Doe",
            "email": "john.doe@usi.ch",
            "affiliation": "Università della Svizzera Italiana",
        }
    }
}

start_the_time()
db["papers"].update_one(query, update)
end_the_time()
```

The query takes 2.6 ms to execute.

5.4 Add a Flag to Authors Informing They Have USI Credentials

This query adds an additional field for documents in the authors collection affiliated with “Università della Svizzera Italiana” informing that this users have USI login credentials.

```
query = {
    "affiliation": "Università della Svizzera Italiana"
}

update = {
    "$set": {
        "hasUSILogin": True
    }
}

start_the_time()
db["authors"].update_many(query, update)
end_the_time()
```

The query takes 104.5 ms to execute.

5.5 Add a Keyword Based on Paper Abstract

This query adds the tag “AI” to the list of keywords of papers that contain the string “Machine Learning” in their abstract.

```
query = {
    "abstract": { "$regex": "Machine Learning" }
}

update = {
    "$push": {
        "keywords": "AI"
    }
}

start_the_time()
db["papers"].update_many(query, update)
end_the_time()
```

The query takes 126.5 ms to execute.

5.6 Search Papers Affiliated With Author Having USI Email

This query returns a list of paper titles of papers that have at least one author having an email address ending in “@usi.ch”. Additionally, we use query projection to only return the “title” field of each document so as to increase performance.

```
start_the_time()
result = db["papers"].find({
    "authors.email": {"$regex": "@usi\.ch"}
}, {
    'title': 1
})
end_the_time()

titles = [doc['title'] for doc in result]
print(titles)
```

The query takes 0.1 ms to execute.

5.7 Titles of the most recent papers

This query retrieves the names of the papers published on the most recent publication date found in the database.

```
pipeline = [{
    "$group":{
        "_id": "$publicationDetails.date"
    }
},{
    "$sort": {
        "_id": -1
    }
},{
    "$limit": 1
}]

start_the_time()
result = db[PAPERS].aggregate(pipeline)
most_recent_date = list(result)[0]['_id']
```

```
query = {
    "publicationDetails.date": most_recent_date
}

result = db[PAPERS].find(query)
for row in result:
    print(row['title'])

end_the_time()
```

This query takes 2082.6 ms to execute.

5.8 Number of papers about "Image Processing"

This query counts number of papers about Image Processing published with at least one author from UK(email ending to ".uk").

```
pipline = {
    "$and": [
        { "authors.email": { "$regex": "\.uk$" } },
        { "title": {"$regex": "image processing"} }
    ]
}

s_t = time.time()
db["papers"].count_documents(pipline)
e_t = time.time()

print('Execution time:', (e_t-s_t) * 1000, 'Milliseconds')
```

The query takes 243 ms to execute.

5.9 Top 10 Universities by affiliations

This query gets the list of top 10 Universities which have highest number of affiliation by the authors.

```
pipeline = [{
    "$group":{
        "_id": "$affiliation",
        "total_faculty": {
            "$sum": 1
        }
    },
}, {
    "$sort": { "total_faculty": -1 }
}, {
    "$limit": 10
}]

s_t = time.time()
result = db["authors"].aggregate(pipeline)
```

```
e_t = time.time()
print('Execution time:', (e_t-s_t) * 1000, 'Milliseconds')
```

The query takes 97 ms to execute.

5.10 Top 10 Journals by Number of Papers

This query returns the titles of the top 10 journals by number of papers published in them. Each journal is returned with a total paper count, returned in the `paper_number` field.

```
pipeline = [{
    "$group": {
        "_id":"$publicationDetails.journal",
        "paper_number":{
            "$sum":1
        }
    }
},{
    "$sort":{
        "paper_number":-1
    }
},{
    "$limit":10
}]

start_the_time()
result = db["papers"].aggregate(pipeline)
end_the_time()
```

The query takes 95.7 ms to complete.

5.11 Top 3 Cited Authors in 'Strategic info-mediaries'

This query returns the names of the top 3 authors by number of citations in the journal "Strategic info-mediaries". The number of references of each author is returned in the `referenceCount` field.

```
pipeline = [
    {
        "$match": {
            "publicationDetails.journal": "Strategic info-mediaries",
        }
    },
    { "$unwind": "$references" },
    { "$unwind": "$references.authors" },
    {
        "$group": {
            "_id": "$references.authors.name",
            "referenceCount": {
                "$sum": 1
            }
        }
    },
    {
        "$sort": {
            "referenceCount": -1
        }
    },
    {
        "$limit": 3
    }
]

start_the_time()
result = db["papers"].aggregate(pipeline)
end_the_time()
```

The query takes 115.7 ms to complete.

5.12 Successful Topics in 'Vertical e-markets'

This query first finds authors that published a paper in “Vertical e-markets” having the word “successful” (optionally capitalized) in their bio. Then, it computes the sum of occurrences per paper per author of each keyword in papers this authors authored, returning the top 10 counts.

```
pipeline = [
    {
        "$match": {
            "publicationDetails.journal": "Vertical e-markets"
        }
    },
    {
        "$unwind": "$authors"
    },
    {
        "$lookup": {
            "from": "authors",
            "localField": "authors.authorId",
            "foreignField": "_id",
            "as": "authors"
        }
    },
    {
        "$match": {
            "authors.bio": {
                "$regex": "[Ss]uccess"
            }
        }
    },
    {
        "$unwind": "$keywords"
    },
    {
        "$group": {
            "_id": "$keywords",
            "referenceCount": {
                "$sum": 1
            }
        }
    },
    {
        "$sort": {
            "referenceCount": -1
        }
    },
    {
        "$limit": 10
    }
]

start_the_time()
result = db["papers"].aggregate(pipeline)
end_the_time()
```

The query takes 302.9 ms to execute.

5.13 Paper Titles in Journal "Enterprise eyeballs"

This query returns the paper titles for all papers in all volumes in the journal "Enterprise eyeballs". The query starts from the document in the *journals* collection and *unwind*s and *lookup*s to join the matching *papers* documents. Additionally, query projection is used to only return the paper title so as to improve performance.

```
pipeline = [
    {
        "$match": {
            "name": "Enterprise eyeballs"
        }
    },
    {
        "$unwind": "$volumes"
    },
    {
        "$unwind": "$volumes.papers"
    },
    {
        "$lookup": {
            "from": "papers",
            "localField": "volumes.papers",
            "foreignField": "_id",
            "as": "paper"
        }
    },
    {
        "$project": {
            "paper.title": 1,
            "_id": 0
        }
    },
    {
        "$limit": 10
    }
]

start_the_time()
result = db["journals"].aggregate(pipeline)
end_the_time()
```

The query takes 4.3 ms to execute.

5.14 Most popular Journal - Top Level Section Title Combination

This query returns the top 10 most frequent combinations of journal names and section titles across papers.

```
pipeline = [
    {
        "$unwind": "$content"
    }, {
        "$group": {
            "_id": {
                "journal": "$publicationDetails.journal",
                "sectionTitle": "$content.title"
            },
            "sectionCount": {
                "$sum": 1
            }
        }
    }, {
        "$sort": {
            "sectionCount": -1
        }
    }, {
        "$limit":10
    }
]

start_the_time()
result = db["papers"].aggregate(pipeline)
end_the_time()
```

This query takes 1305.8 ms to execute.

5.15 Get the Top Writers of Papers

This query returns a list of authors and the number of their published papers which shows the top authors.

```
pipeline = [
   {
        "$lookup": {
            "from": "papers",
            "localField": "_id",
            "foreignField": "authors.authorId",
            "as": "papers"
        },
    },
    {
        "$project": {
            "name": 1,
            "numberOfPapers": {
                "$cond": {
                    "if": { "$isArray": "$papers" },
                    "then": { "$size": "$papers" },
                    "else": 0
                }
            },
        }
    },
    { "$limit": 100 },
    { "$sort": {"numberOfPapers": -1} }
]

s_t = time.time()
result = db["authors"].aggregate(pipeline)
e_t = time.time()
print('Execution time:', (e_t-s_t) * 1000, 'Milliseconds')
```

This query takes 8.2s to execute.

5.16 Authors with list of their papers

This query returns returns a list of authors and their papers. The query gets the list of authors and then adds the list of papers of each author.

```
pipeline = [{
    "$match": {
        "email": {"$exists": True}
    }
},{
    "$lookup":{
        "from":"papers",
        "localField":"_id",
        "foreignField":"authors.authorId",
        "as":"papers"
    }
}, {"$limit": 10}]


s_t = time.time()

result = db[AUTHORS].aggregate(pipeline)
pd.DataFrame(result)


e_t = time.time()
```

```
print('Execution time:', (e_t-s_t) * 1000, 'Millisecond' )
```

This query takes 2.2s to execute.