POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Systems and Methods for Big and Unstructured Data Project

Author(s): **Angelo Capponcelli**

Valentina Abbattista

Ossama El Oukili

Filippo Cinfrignini

Francesca Shmilli

Group Number: **21**

Academic Year: 2022-2023

Contents

1 | Description of the Problem

The aim of the project is to implement a system that manages and queries a bibliography database of scientific articles. Particularly, the ones stored in the DBLP (*Digital Bibliograpy & Library Project*) dataset.
DBLP provides open bibliographic information on major computer science journals and publications. The articles are classified with to the following labels:

Authors
Books written or co-written by recognized authors go into this category, there is the chance to make a search-by-author.

Journals
Articles that have appeared into reviewed journals fall into this category.

Conferences
This category lists papers published in reviewed conferences.

Books
In this category authored monographs, as well as PhD theses, are listed.

2 | Assumptions

The following assumptions are made:

- An *article* is directly and uniquely identified through a DOI (*"Digital Object Identifier"*), which is a unique key-value associated to each article.
- An *article* could include references to other articles.
- An *article* can be published in a *journal* or a *book* and can be presented at a *conference*.
  A set of *articles* can be collected in a *book* or a *journal*.
- An *article* is published inside either one *book* or one *journal*, there are no articles that are in a journal and a book at the same time.
- A *journal* is a collection of *volumes* and a *volume* is a collection of *numbers*.
  Each *volume* contains information about how many *numbers* were published and each *journal* contains information about how many *volumes* it is composed of.
- Every *author* is identified into scientific literature by an alphanumerical key called ORCID (*"Open Researcher and Contributor IDentifier"*).

3 | ER Diagram

The following Entity-Relationship diagram was derived with a bottom-up approach and represent how we perceive objects and concepts relate to each other within the open bibliographic information system. The identified entities of the ER model are further described:

- ARTICLE is the most significant entity, it is the one related with every other entity. It has the following attributes *DOI, Title, Year, Timestamp, BibUrl, BibSource.*
- AUTHOR represents the academic(s) who wrote an article; an academic can write an article, also a group of academics can collaborate and write an article together.
- BOOK represents one or a collection of articles that are published in a certain year by a certain publisher.
- VOLUME is the collection of one or more numbers of a published journal.
- JOURNAL is composed of at least one volume.
- CONFERENCE is an event at which one or more articles are collected to be published inside a book or the volume of a journal.
- MEDIUM is a total generalization of the "means" of publication.

The following relations connect the above entities:

- WRITE connects author with article. An author wrote one or more articles. An article can be written by one or more authors.
- PUBLISHED specifies the means where we can find the article, it can be the volume of a journal but also a book; it also maintains the interval of pages where the article is located.
- PRESENTED binds an article with a conference, an article can be presented at a conference.

- IsPartOf is used to specify the journal in which a volume is stored.
- REFERENCE binds two or more articles between them, if one references another one it will be visible thanks to this relation.

The Entity-Relationship model of the database considered is the following:

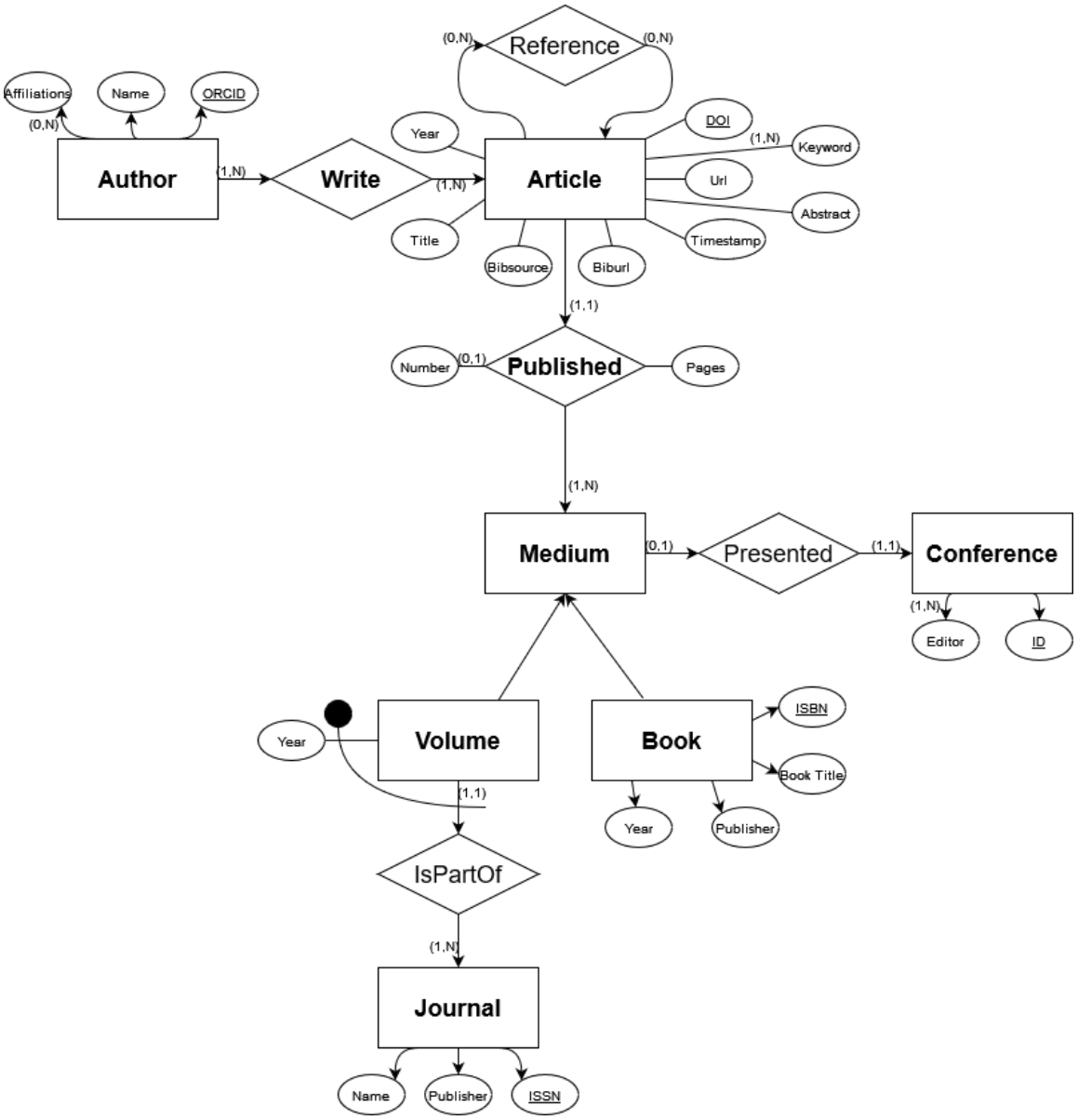


Figure 3.1: ER diagram

4 | Dataset

4.1. Dataset Description

The provided dataset is the one offered by https://dblp.org. It has been downloaded in an .xml format and then converted a into a .csv file through a converter from github. The file was of huge dimension and some rows were incomplete, for example some authors were not correctly linked to the *orcid* identification code.
There was a data cleaning phase to remove all inconsistencies. Then the data were imported into the database from the generated csv files. The preprocessing phase was supported by specific scripts written in python that take as input the csv file containing a particular dataset, and output the csv file with the correctly cleaned data.
Initially, the headers of each column were defined, with the name of the fields in that column. In some cases the fields were already available within the database e.g. in dblp_article.csv the title of the article, in other cases it was necessary to extract a new value from those already present in the database e.g. in dblp_article.csv from the *ee* field, containing the links of an article, it was possible to extract the unique identifier *DOI*. The header was placed at the beginning in the first line of the file, and in the subsequent lines the values of the relevant fields were appended.
Each file was scanned from the beginning until the desired database size was reached, e.g. for the articles file, 3000 rows were selected. Each row, after being read from the file, was analysed to check that all fields were present, that there was consistent data, and to extract further data useful for creating the database.
In some cases, to ensure greater variety within the database and not to lose generality, rows were scanned not in order but rather skipping some, thus exploring the entire dataset available. Each line after processing was appended to the end of the file. Some characters were escaped to avoid errors when importing into the database, e.g. the double quotation marks was replaced with single quotation marks.
Finally, the entire dataset was written to a file and imported into the database.

4.1.1. Pre-processing python script

Script used to preprocess the csv files and reduce the number of rows.

```
1 import re
2 with open("dblp_article.csv", "r", encoding= "ISO-8859-1") as
    file:
3         text = ["ID,author,authoraux,authororcid,booktitle,cdate
    ,cdrom,cite,citelabel,crossref,editor,editororcid,eedoi,ee,
    eetype,i,journal,key,mdate,month,note,notelabel,notetype,
    number,pages,publisher,publnr,publtype,sub,sup,title,
    titlebibtex,tt,url,volume,year,doi"]
4         text.append("\n")
5         x = 0
6         j = 0
7         while x<3000:
8             line = next(file)
9             if "https://doi.org/" in line and j%80==0:
10                splittedString = line.split(";")
11                splittedEE = splittedString[12].split("|")
12                eedoi = splittedEE[0]
13                doi = eedoi.replace("https://doi.org/", "")
14                ee = ''
15                if len(splittedEE)> 1: ee = splittedString[12].
    split("|")[1]
16                splittedString.pop(12);
17                splittedString.insert(12, eedoi)
18                splittedString.insert(13, ee)
19
20                authors = splittedString[1].split("|")
21                orcids = splittedString[3].split("|")
22                if orcids[0] == '':
23                    orcids = []
24                if len(splittedString) < 37 and len(authors) !=
    0 and len(authors) == len(orcids):
25                    line = ';'.join(splittedString)
26                    newLine = line.replace("\n", ";") + doi + "\
    n"
```

```
27                          text.append(newLine.replace(",", "").replace
    (";", ",",))
28                          x +=1
29              j += 1
30
31 text = ''.join([i for i in text])
32
33 # search and replace the contents
34 text = text.replace('"', "'")
35
36
37 # output.csv is the output file opened in write mode
38 x = open("dblp_article_fixed.csv","w", encoding= "ISO-8859-1")
39
40 # all the replaced text is written in the output.csv file
41 x.writelines(text)
42 x.close()
```

Listing 4.1: Python Script for pre-processing

4.2. Dataset Upload

The dataset was imported on Neo4j Desktop following these steps:

1. created a new project
2. created a new local database
3. load the .csv modified files in the import folder: dblp_book.csv, dblp_article.csv, dblp_inproceedings.csv.

Then the following queries were runned to parse our data and generate the graph with the nodes and the relationships:

1. **Import books, articles published in books and relationships PUBLISHED between articles and books from file *dblp_book.csv*:**

```
LOAD CSV WITH HEADERS FROM 'file:///dblp_book.csv' AS line
WITH SPLIT(line.isbn, "|") AS p, line
WHERE line.ee IS NOT NULL
FOREACH (n IN p |
```

```
MERGE(b:Book {isbn: n}) ON CREATE SET
b.booktitle=line.booktitle, b.year=toInteger(line.year)
MERGE(a:Article {doi: line.ee}) ON CREATE SET a.title=line.title,
a.year=toInteger(line.year), a.url=line.url
MERGE(a)-[p:PUBLISHED]->(b) ON CREATE SET p.pages=line.pages);
```

In this query:

- Book nodes are created with properties isbn, booktitle, year. Each Book node has a different ISBN identifier;
- Article nodes are created with properties doi, title, year, url. Each Article node has a different DOI identifier;
- relationships PUBLISHED between books and articles are created with property pages, which indicates the pages of the book where the article is published.

The function SPLIT is used because an article could be published on several books and each book has its own ISBN identifier. The ISBN identifiers for a single article are on the same line and on the same field "line.isbn" and they are separated with the character "|".
Then the clause FOREACH is used to create a new Book node for each ISBN. In this way, each Book node has a different ISBN, thus if there are some books which have the same title but are from different versions, they will be represented by different nodes.

2. **Import authors (that wrote articles published in books) and relationships WRITE between authors and articles (published in books) from file *dblp_book.csv*:**

```
LOAD CSV WITH HEADERS FROM 'file:///dblp_book.csv' AS line
WITH SPLIT(line.author, "|") AS a, SPLIT(line.authororcid, "|") as o, line
WHERE line.ee is not null
FOREACH (i in range(0, size(a)-1) |
    MERGE(n:Author {orcid: o[i]}) ON CREATE SET n.name=a[i]
    MERGE(m:Article {doi: line.ee})
    MERGE(n)-[w:WRITE]->(m));
```

In this query:

- Author nodes are created with properties orcid, name. Each Author node has a different ORCID identifier;

- Article nodes are created with property doi. Each Article node has a different DOI identifier;
- relationships WRITE between authors and articles are created.

As we did before, on the same line there could be multiple authors on the same field "line.author" and so there are multiple ORCID identifiers on the same field "line.authororcid". Both authors and ORCID identifiers are separated by the character "|", thus the function SPLIT can be used.
Then the clause FOREACH is used to create a different node for every author, so every Author node will have a different ORCID identifier.

3. **Import journals from file *dblp_ articles.csv*:**

```
LOAD CSV WITH HEADERS FROM 'file:///dblp_article.csv' AS line
MERGE(j:Journal {journal: line.journal})
ON CREATE SET j.publisher=line.publisher
```

In this query a node for each journal is created with properties journal and publisher.

4. **Import articles published in journals from file *dblp_ articles.csv***

```
LOAD CSV WITH HEADERS FROM 'file:///dblp_article.csv' AS line
WITH line
WHERE line.ee IS NOT NULL
MERGE(a:Article {doi: line.ee})
ON CREATE SET a.title=line.title,
a.year=toInteger(line.year),
a.url=line.url
```

In this query a node for each article is created with the following properties: doi, title, year, url. Each Article node has a different DOI identifier.

5. **Import authors (that wrote articles published in journals) and relationships WRITE between articles and authors from file *dblp_ articles.csv*:**

```
LOAD CSV WITH HEADERS FROM 'file:///dblp_article.csv' AS line
WITH SPLIT(line.author, "|") AS a, SPLIT(line.authororcid, "|") as o,
line
WHERE line.ee IS NOT NULL
FOREACH (i in range(0, size(a)-1) |
    MERGE(n:Author {orcid: o[i]}) ON CREATE SET n.name=a[i]
    MERGE(m:Article {doi: line.ee})
```

```
    MERGE(n)-[w:WRITE]->(m));
```

In this query:

- Author nodes are created with properties orcid, name. Each Author name has a different ORCID identifier;
- relationships WRITE between authors and articles.

The query works as query number 2.

6. **Import of PUBLISHED relationships between articles and journals from file *dblp_ articles.csv*:**

```
LOAD CSV WITH HEADERS FROM 'file:///dblp_article.csv' AS line
MATCH(a:Article {doi: line.ee})
MATCH(j:Journal {journal: line.journal})
MERGE(a)-[p:PUBLISHED]->(j)
ON CREATE SET p.pages=line.pages, p.number=toInteger(line.number)
```

In this query, relationships PUBLISHED between articles and journals are created with properties pages and number.

7. **Import articles presented in conferences from file *dblp_ inproceedings.csv*:**

```
LOAD CSV WITH HEADERS FROM 'file:///dblp_inproceedings.csv' AS line
WITH line
WHERE line.ee IS NOT NULL
MERGE(a:Article {doi: line.ee})
ON CREATE SET a.title=line.title,
a.year=toInteger(line.year),
a.url=line.url
```

In this query Article nodes are created with properties doi, title, year, url. Each Article node has a different ORCID identifier.

8. **Import authors (that wrote articles presented in conferences) and relationships WRITE between authors and articles from file *dblp_ inproceedings.csv*:**

```
LOAD CSV WITH HEADERS FROM 'file:///dblp_inproceedings.csv' AS line
WITH SPLIT(line.author, "|") AS a, SPLIT(line.authororcid, "|") as o, line
WHERE line.ee IS NOT NULL
FOREACH (i in range(0, size(a)-1) |
    MERGE(n:Author {orcid: o[i]}) ON CREATE SET n.name=a[i]
```

```
MERGE(m:Article {doi: line.ee})
MERGE(n)-[w:WRITE]->(m));
```

In this query:

- Author nodes are created with properties orcid and name. Each Author node has a different ORCID identifier;
- relationships WRITE between authors and articles are created.

The query works as queries number 2 and number 5.

9. **Import conferences from file *dblp_ inproceedings.csv*:**

```
LOAD CSV WITH HEADERS FROM 'file:///dblp_inproceedings.csv' AS line
WITH line
WHERE line.booktitle IS NOT NULL
MERGE(n:Conference {name: line.booktitle})
```

In this query Conference nodes are created with property name. We assumed that the field "line.booktitle" of the csv file represents the name of the conference were the article has been presented. We assumed that each Conference node has a different name, which is the identifier.

10. **Import relationships PRESENTED between articles (presented in conferences) and conferences from file *dblp_ inproceedings.csv*:**

```
LOAD CSV WITH HEADERS FROM 'file:///dblp_inproceedings.csv' AS line
WITH line
WHERE line.booktitle IS NOT NULL AND line.ee IS NOT NULL
MATCH(n:Conference {name: line.booktitle})
MATCH(m:Article {doi: line.ee})
MERGE(m)-[p:PRESENTED]->(n)
```

In this query, relationships PRESENTED between articles and conferences is created.

The following images show the resulting graph:

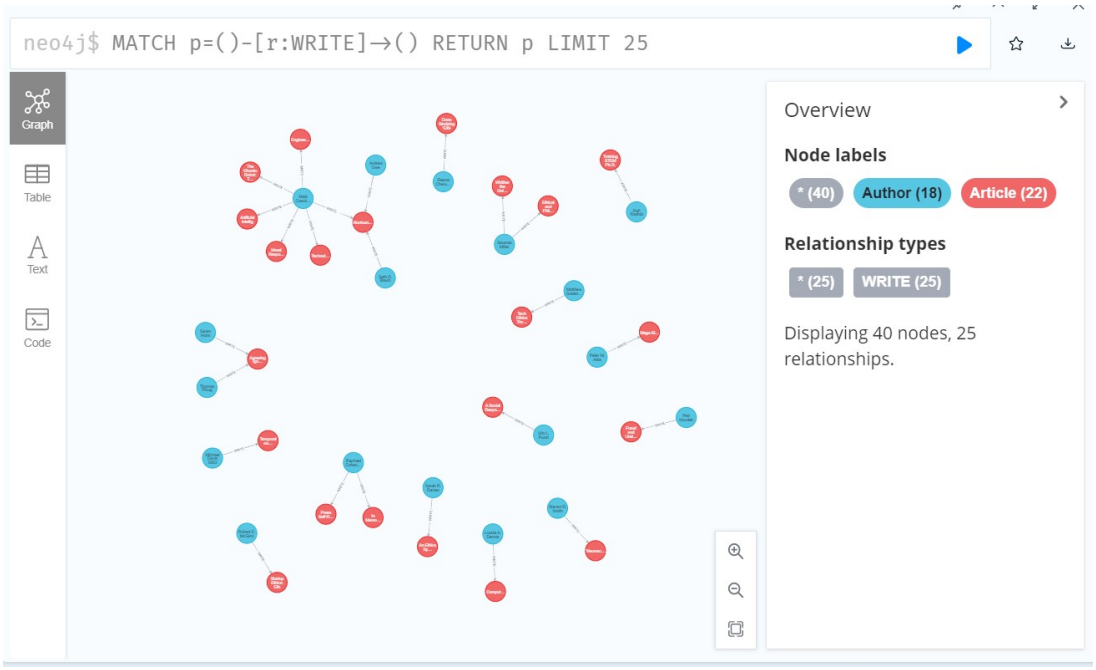


Figure 4.1: Author nodes, Article nodes and relationships WRITE between authors and articles.

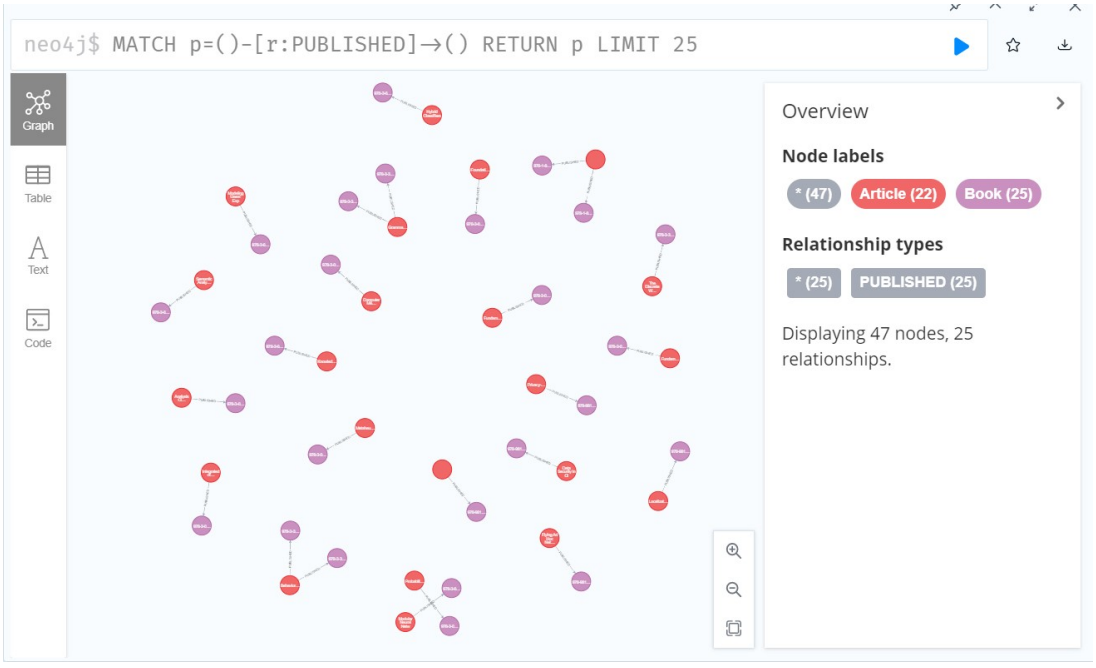


Figure 4.2: Article nodes, Book nodes and relationships PUBLISHED between articles and books.

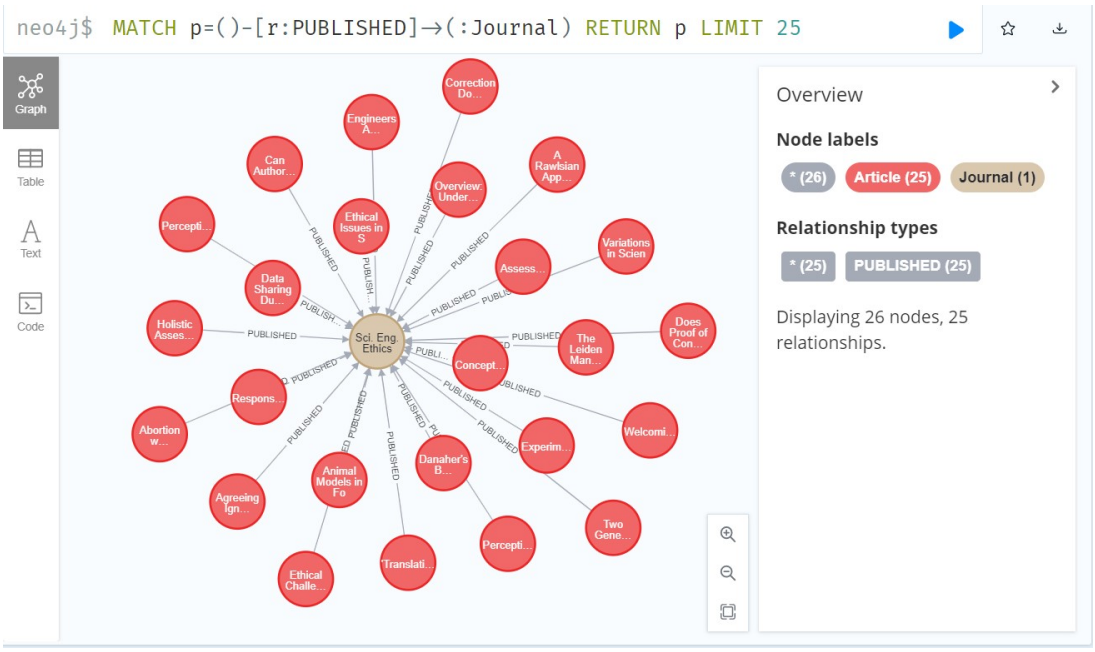


Figure 4.3: Article nodes, Journal nodes and relationships PUBLISHED between articles and journals.

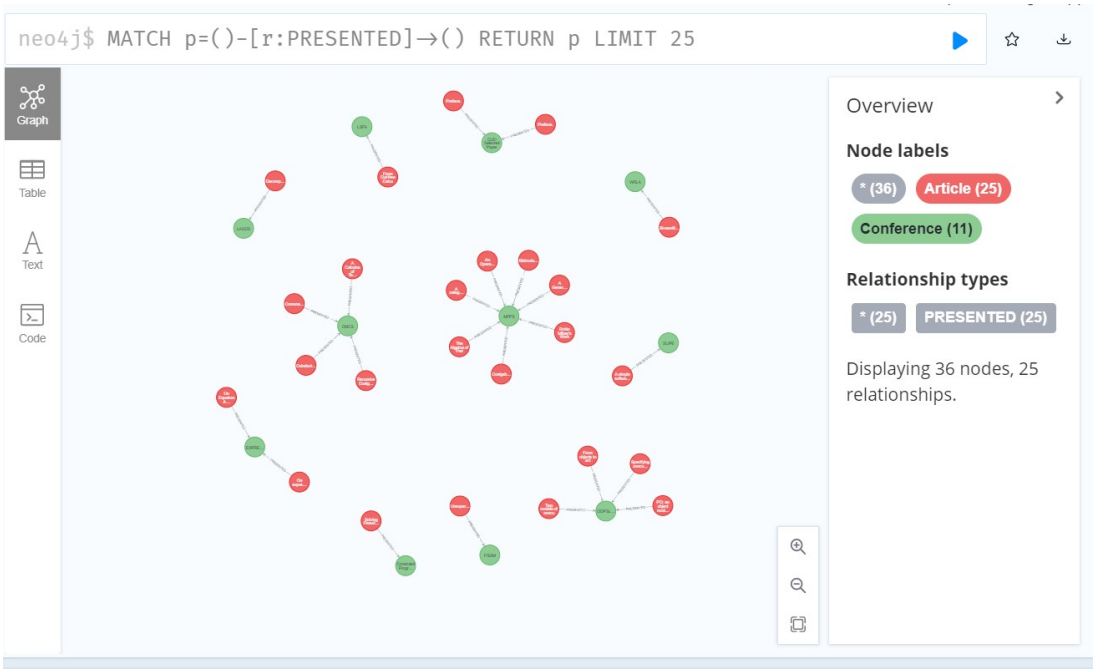


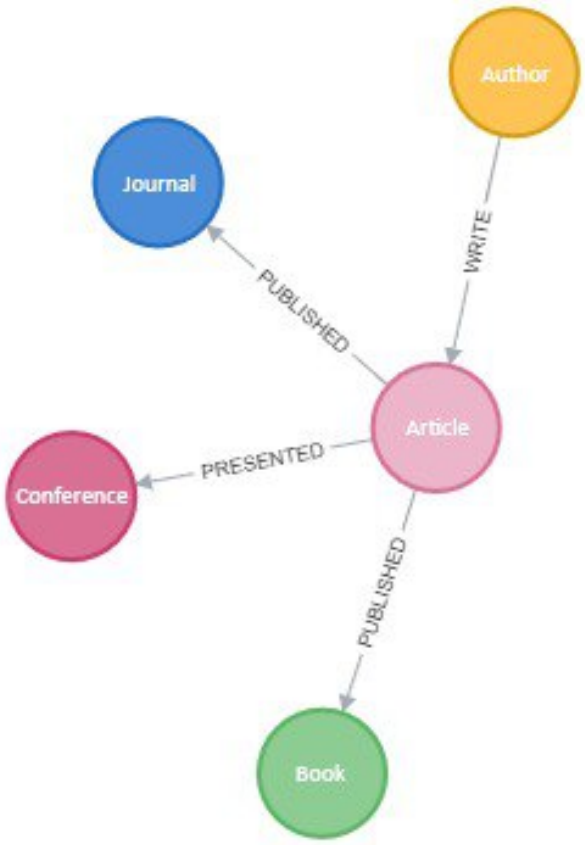
Figure 4.4: Article nodes, Conference nodes and relationships PRESENTED between articles and conferences.

5 | Graph diagram

The graph diagram has a different structure compared to the ER model.
We decided to reduce the number of attributes in the graph diagram to be coherent to the actual and more tangible meaning of the data set. Neo4j provides a graphical visualization of the structure using the following function:

```
call db.schema.visualization
```

This picture shows the result:



After importing data the following nodes and relationships were created:

- **Author** nodes with properties: orcid, name;

- **Article** nodes with properties: doi, title, year, url;
- **Book** nodes with proterties: isbn, booktitle;
- **Conference** nodes with property: name;
- **Journal** nodes with properties: journal, publisher;
- **WRITE** relationships between Author nodes and Article nodes;
- **PUBLISHED** relationships between Article nodes and Book/Journal nodes with properties: pages, number;
- **PRESENTED** relationships between Article nodes and Conference nodes.

The database contains 1491 Article nodes, 2112 Author nodes, 246 Book nodes, 6 Journal nodes and 35 Conference nodes, so the total number of nodes is 3890.
It also contains 271 PRESENTED relationships, 1242 PUBLISHED relationships and 2546 WRITE relationships, so the total number of relationships is 4077.

6 | Query and performance

These queries and commands have been developed to provide an example of usage of the system. The performance time and complexity are reported for each query.

6.1. Create/update commands

In this section some create/update queries are reported; for the other create queries check the **Database upload** section.

1. **Creation of COAUTHOR relationship**

```
MATCH (aut1:Author)-[:WRITE]->(art:Article)<-[:WRITE]-(aut2:Author)
WHERE aut1.orcid <> aut2.orcid
MERGE (aut1)-[:COAUTHOR]->(aut2)
RETURN aut1
```

2. **Update year of an article**

```
MATCH (n:Article {title: "Tech Ethics Through Trust Auditing."})
SET n.year = 2021
RETURN n
```

3. **Update pages**

```
MATCH (n:Article {title: "Engagement Design -
      Designing for Interaction Motivations"})
      -[p:PUBLISHED]->(b:Book)
SET p.pages = "1-158"
RETURN p
```

6.2. Queries

1. **Top 10 authors that wrote the most articles**

```
MATCH ()<-[write:WRITE]-(author)
RETURN author AS Author, count(write) AS NumberOfArticles
ORDER BY NumberOfArticles DESC
LIMIT 10
```

This query selects all the nodes connected through WRITE relationship and for each node of type author computes the aggregate count() function. After doing that we obtain a list which relates each author to the number of written articles. Ordering in a descending way and returning only the first 10 items of the list we obtain the top 10 ranked ones.

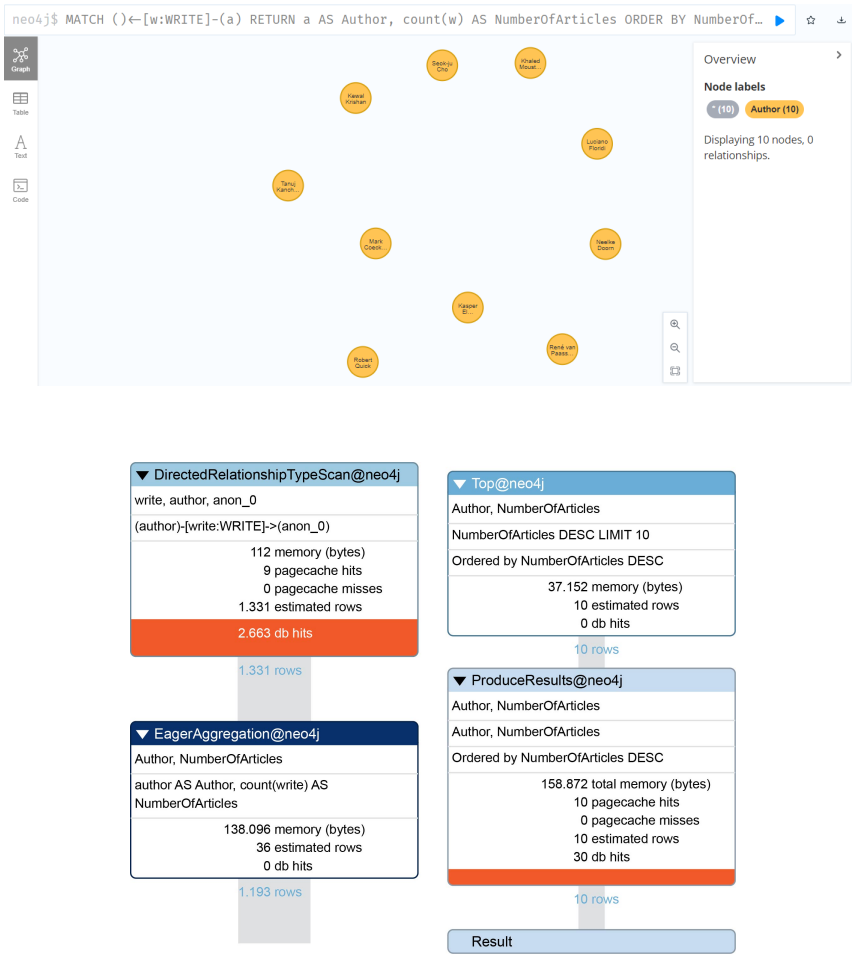


Figure 6.1: Profile query n.1

2. **Top 10 books with the most contributors**

```
MATCH(book)<-[published:PUBLISHED]-(article)<-[write:WRITE]-(author)
RETURN book AS Book , count(write) AS NumberOfContributors
ORDER BY NumberOfContributors DESC
LIMIT 10
```

This query follows the same procedures as the previous one.

"Book"	"NumberOfContributors"
{"journal":"Sci. Eng. Ethics"}	562
{"journal":"J. Econ. Theory"}	483
{"journal":"IEEE Trans. Hum. Mach. Syst."}	479
{"journal":"Comput. Chem. Eng."}	250
{"journal":"Web Intell. Agent Syst."}	19
{"journal":"Web Intell."}	10
{"year":2021,"isbn":"978-981-16-2198-7"}	5
{"isbn":"978-981-16-6095-5"}	5
{"year":2021,"isbn":"978-981-16-4830-4"}	4
{"isbn":"978-3-030-36471-7"}	3

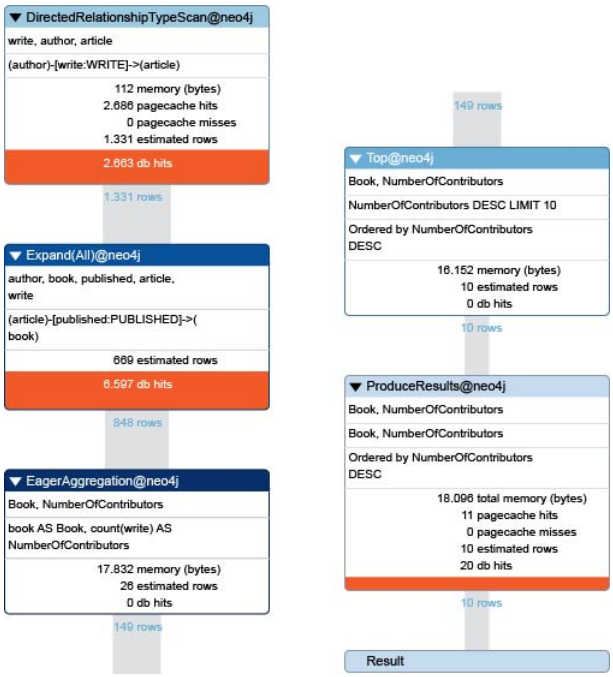


Figure 6.2: Profile query n.2

3. **Average number of authors in a conference**

```
MATCH (c:Conference)<-[p:PRESENTED]-(a:Article)<-[w:WRITE]-(au:Author)
WITH c, count(*) AS na
RETURN avg(na) AS average
```

In this query the number of authors for each conference is computed through count(*) and then all these values are used to compute the average number of authors in a conference.

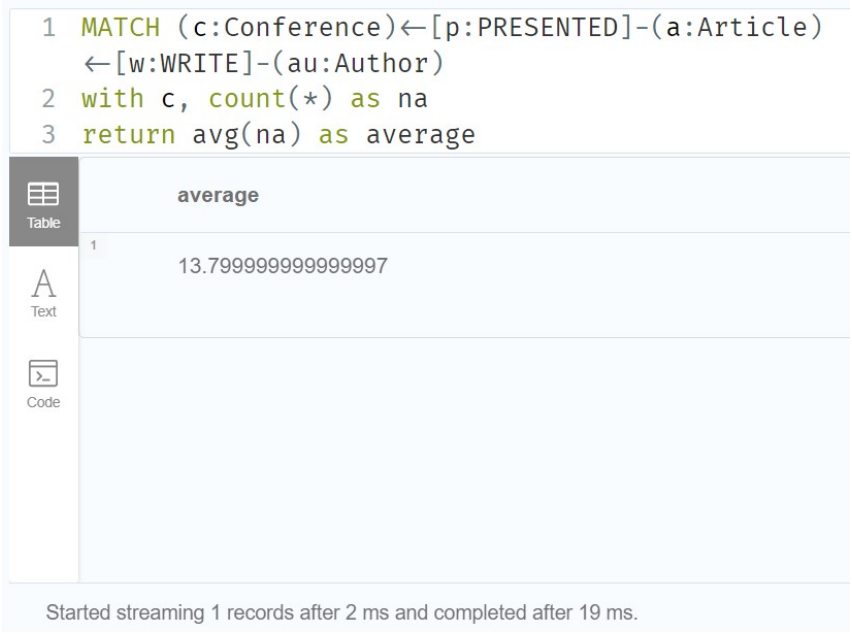


Figure 6.3: Query n.3

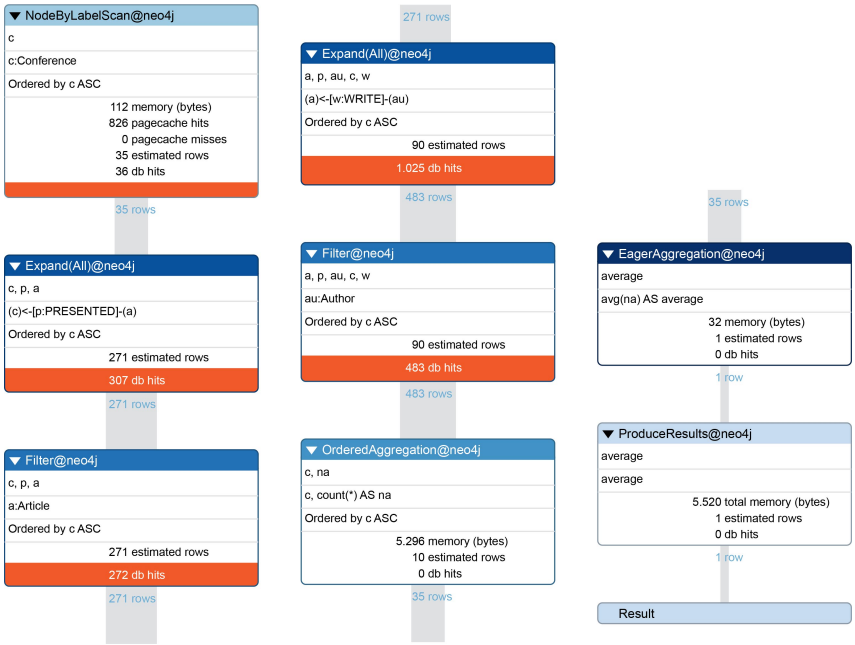


Figure 6.4: Profile query n.3

4. **Number of authors that have published only in conferences**

```
MATCH (au:Author)-[w:WRITE]->(ar:Article)-[p:PRESENTED]->(c:Conference)
WHERE NOT (au)-[:WRITE]->(:Article)-[:PUBLISHED]->(:Book)
AND NOT (au)-[:WRITE]->(:Article)-[:PUBLISHED]->(:Book)
```

```
RETURN count(au)
```

In this query all the authors that wrote an article presented in a conference are taken, and then only the authors that did not write an article published in a book or in a journal are selected through the clause WHERE. At the end the total number of authors that presented an article only in conferences is returned.

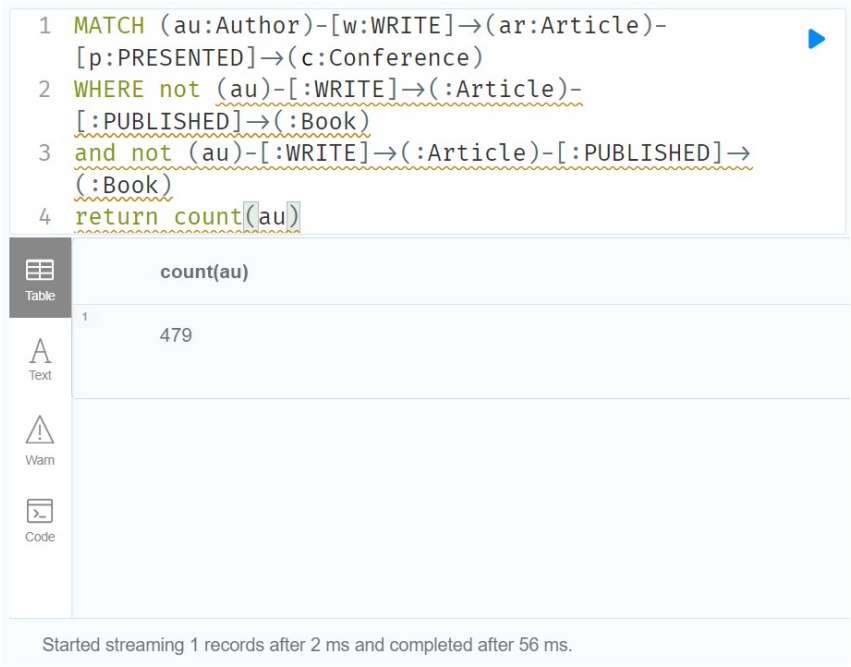


Figure 6.5: Query n.4

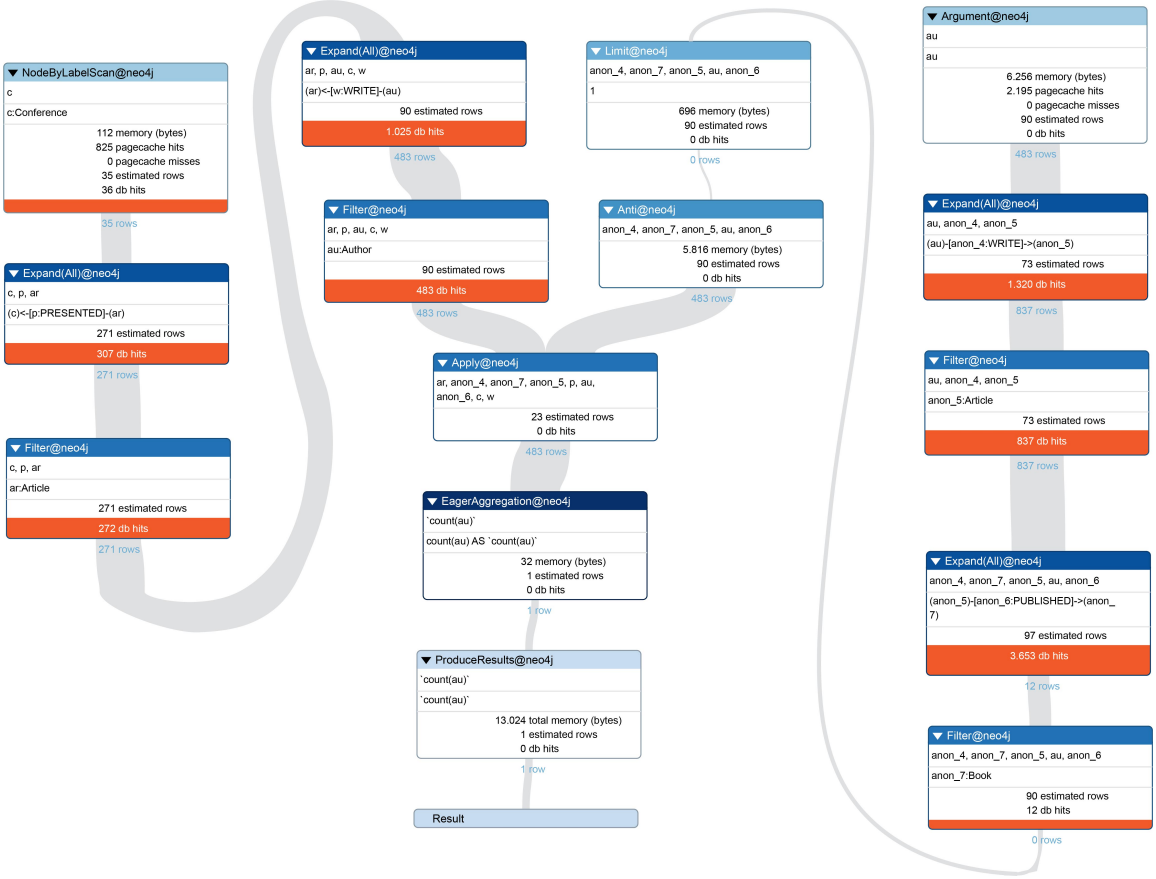


Figure 6.6: Profile query n.4

5. **Shortest path between two authors going through articles only**

```
MATCH   (n:Author {orcid: "0000-0003-0346-1075"}),
(m:Author {orcid: "0000-0001-6657-7871"}),
p = shortestPath((n)-[:WRITE*]-(m))
RETURN length(p)/2 AS mindistance
```

The aim of this query is to compute the collaboration distance between two authors. It is simply the lenght of the shortest path between two authors that passes only through WRITE relationships, divided by 2. The division by 2 is because the shortest path between two collaborators is 2 (there is the Article node in the middle, which is not taken into considerations for our purpose). This can be considered as a kind of Erdos number.

Figure 6.7: Query n.5: result and execution time

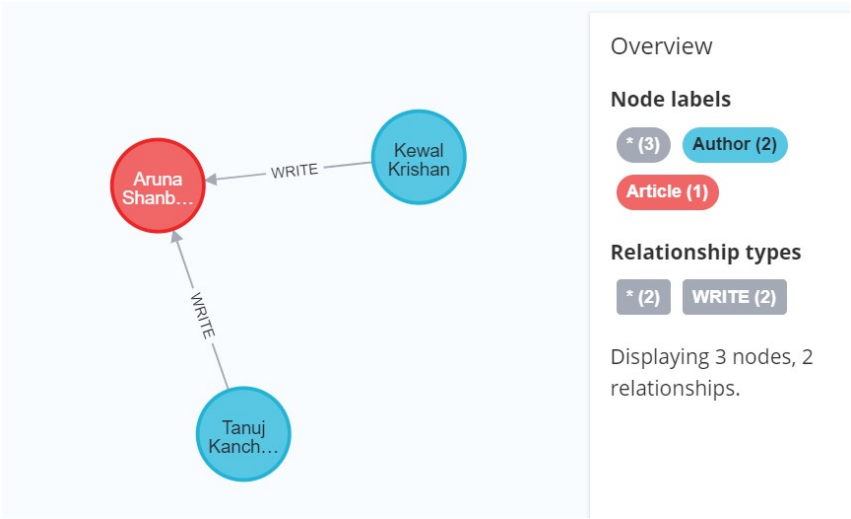


Figure 6.8: Query n.5: path

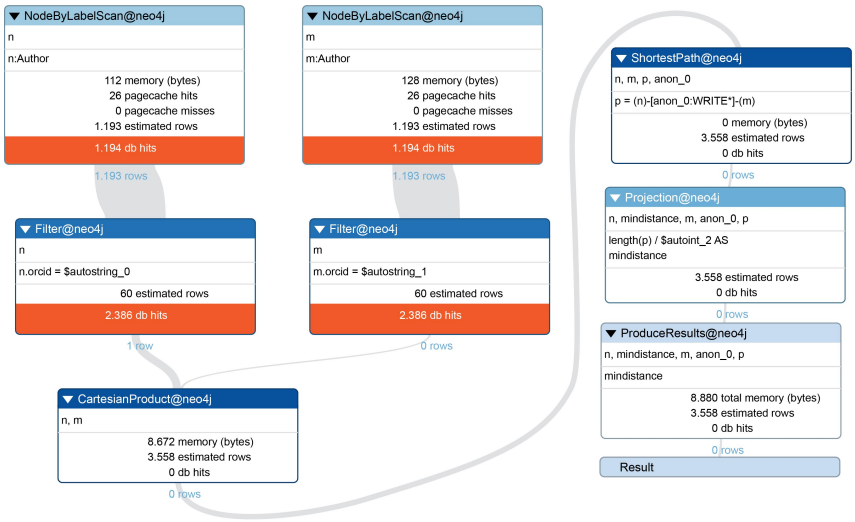


Figure 6.9: Profile query n.5

6. **Authors that published together an article more than one time in the same conference**

```
MATCH (n:Author)-[:WRITE]->(a:Article)-[:PRESENTED]->(c:Conference),
(n1:Author)-[:WRITE]->(a1:Article)-[:PRESENTED]->(c1:Conference)
WHERE c=c1 AND a<>a1 AND n<>n1
AND (n)-[:WRITE]->(a)
AND (n1)-[:WRITE]->(a)
AND (n)-[:WRITE]->(a1)
AND (n1)-[:WRITE]->(a1)
RETURN n,n1
```

In this query, all the authors that collaborated in an article presented in a conference at least twice are taken.

Figure 6.10: Query n.6: result

Figure 6.11: Query n.6: execution time

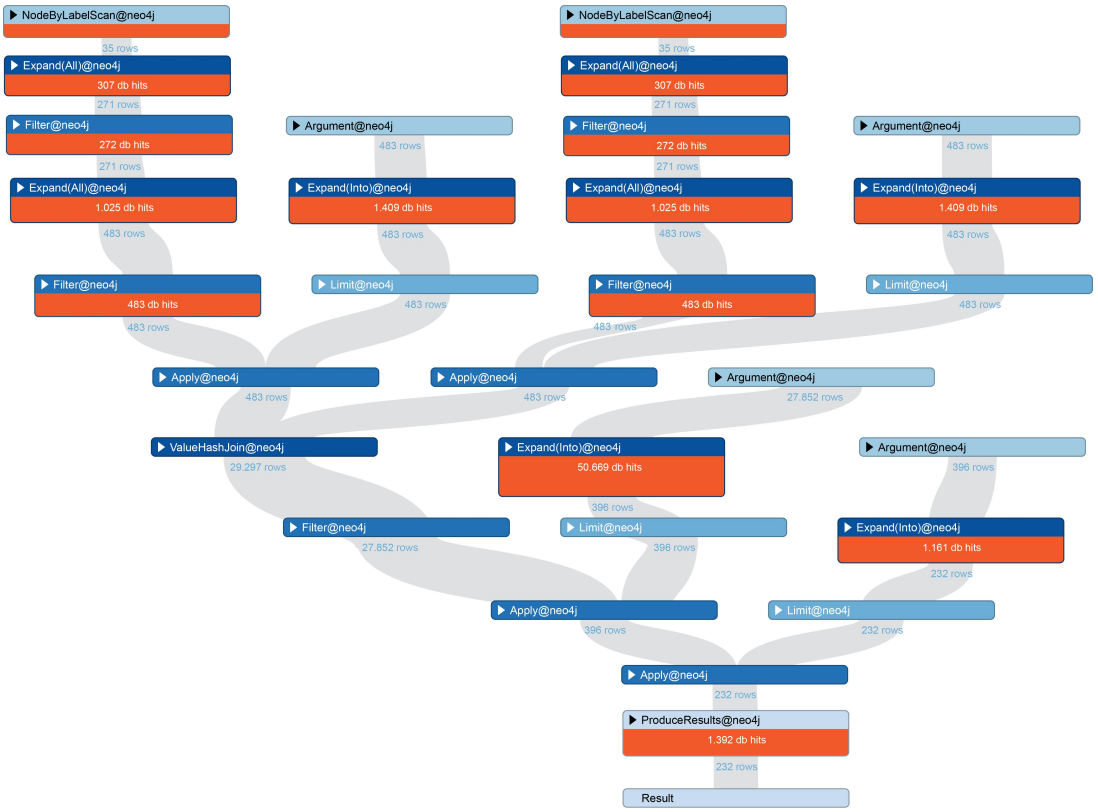


Figure 6.12: Profile query n.6

7. Authors that wrote articles since 2018

```
MATCH (a:Author)-[:WRITE]-(ar:Article)
WHERE ar.year >= 2018
RETURN a
```

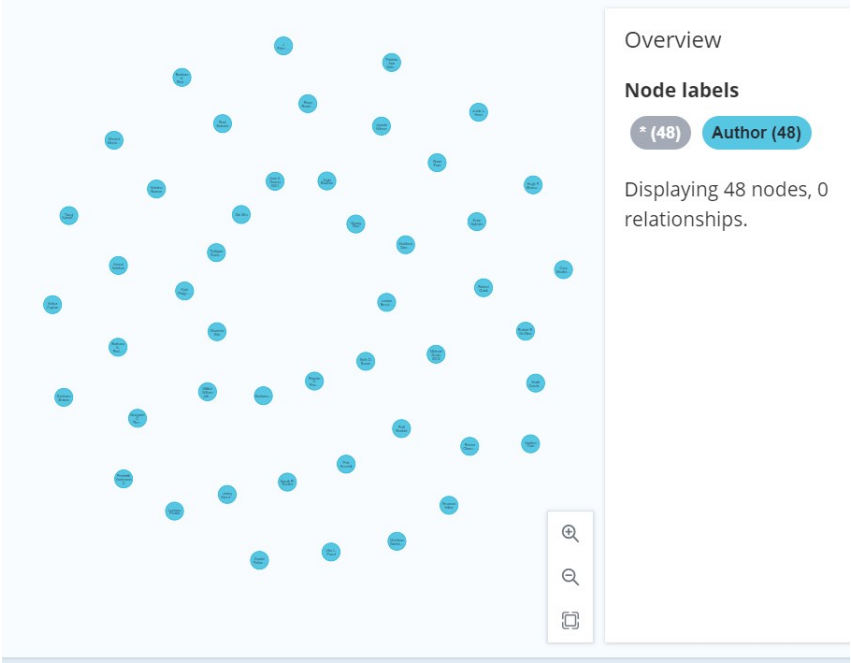


Figure 6.13: Query n.7: partial result

Figure 6.14: Query n.7: execution time

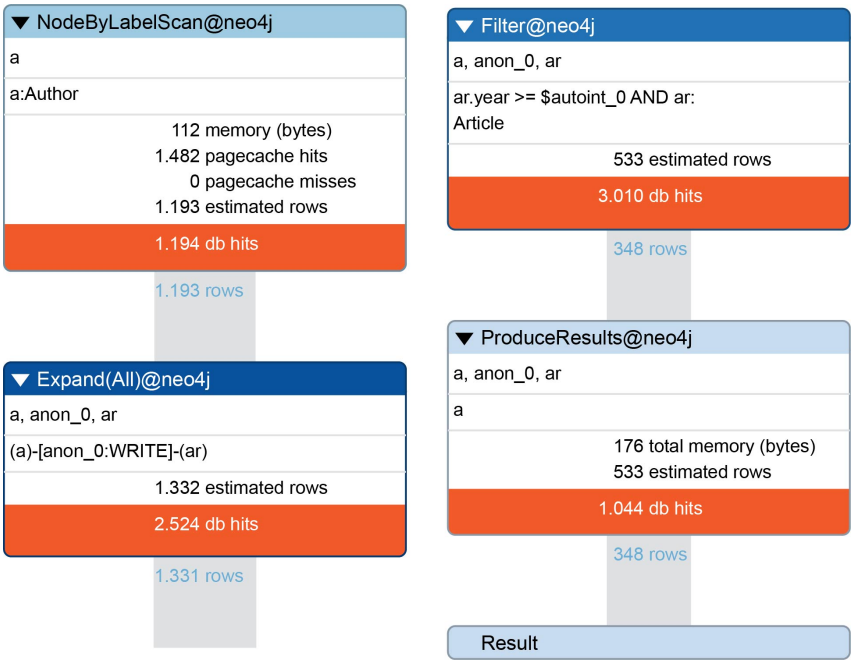


Figure 6.15: Profile query n.7

8. **Given an author, average of how many authors collaborated with him in every article.**

```
MATCH   (au1:Author)-[w1:WRITE]->(a1:Article),
        (au2:Author)-[w2:WRITE]->(a2:Article)
WHERE   au1.orcid = "0000-0003-2125-327X" and
        a1 = a2 and au1.orcid <> au2.orcid
RETURN  count(au2)/count(a1)
```

In this query it is matched the path that goes from the author "Michele Ruta" to every other author who has collaborated with him. Then are counted how many authors have collaborated with him in total, how many articles he has written and then simply calculated the average with the formula: number of authors collaborating divided by number of articles written.

Figure 6.16: Query n.8: results

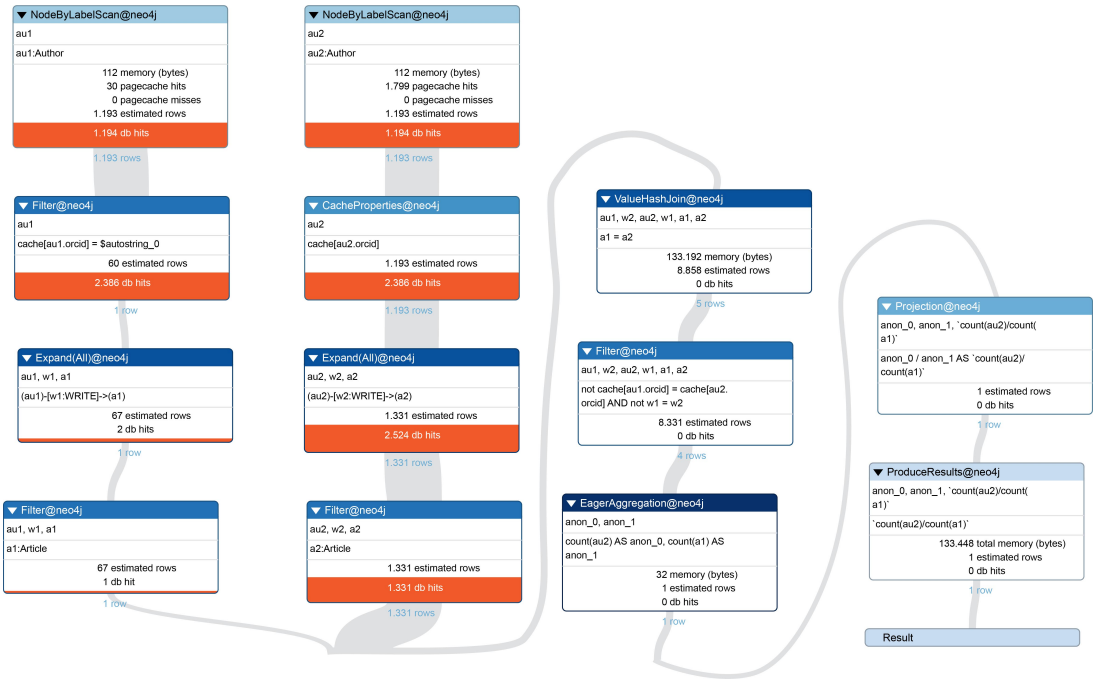


Figure 6.17: Profile query n.8

9. **Return at most 10 journals in which an author published an article**

```
MATCH   (au:Author)-[w:WRITE]->(a:Article)-[p:PUBLISHED]->(j:Journal)
WHERE   au.orcid = "0000-0003-0346-1075"
RETURN DISTINCT j
LIMIT 10
```

This query shows at most 10 distinct journals where an author (in this case "Tanuj Kanchan") has published an article. It is done by matching the path between the nodes Author, Article, Journal and then returning the Journal but limiting the results to 10 and assuring that the results are distinct.

Figure 6.18: Query n.9: results

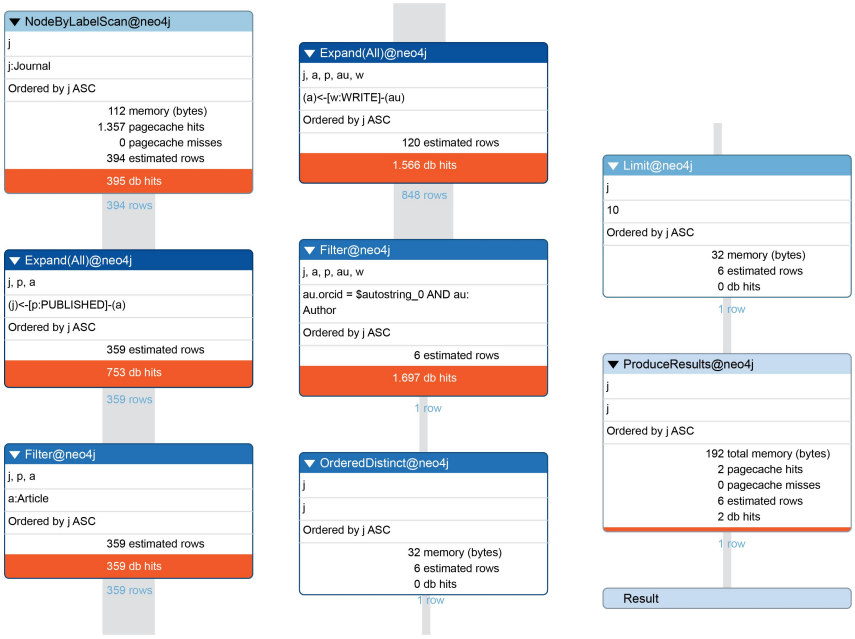


Figure 6.19: Profile query n.9

10. **Return an author that in the same year has published his articles in every possible medium: at least one in a book, one in a journal and one in a book**

```
MATCH   (au1:Author)-[w1:WRITE]->(a1:Article)-[p1:PUBLISHED]->(j:Journal),
        (au2:Author)-[w2:WRITE]->(a2:Article)-[p2:PUBLISHED]->(b:Book),
        (au3:Author)-[w3:WRITE]->(a3:Article)-[p3:PRESENTED]->(c:Conference)
```

```
WHERE  a1.year = a2.year = a3.year and au1.orcid = au2.orcid = au3.orcid
RETURN a1.year, au1.orcid, au1.name
LIMIT 1
```

Match the patterns from an author to each article he has written to the medium where the article is written, check the year to ensure is the same for each of those patterns, if they exist. If found, the query then returns the author and the year of when this happened.

Figure 6.20: Query n.10: results

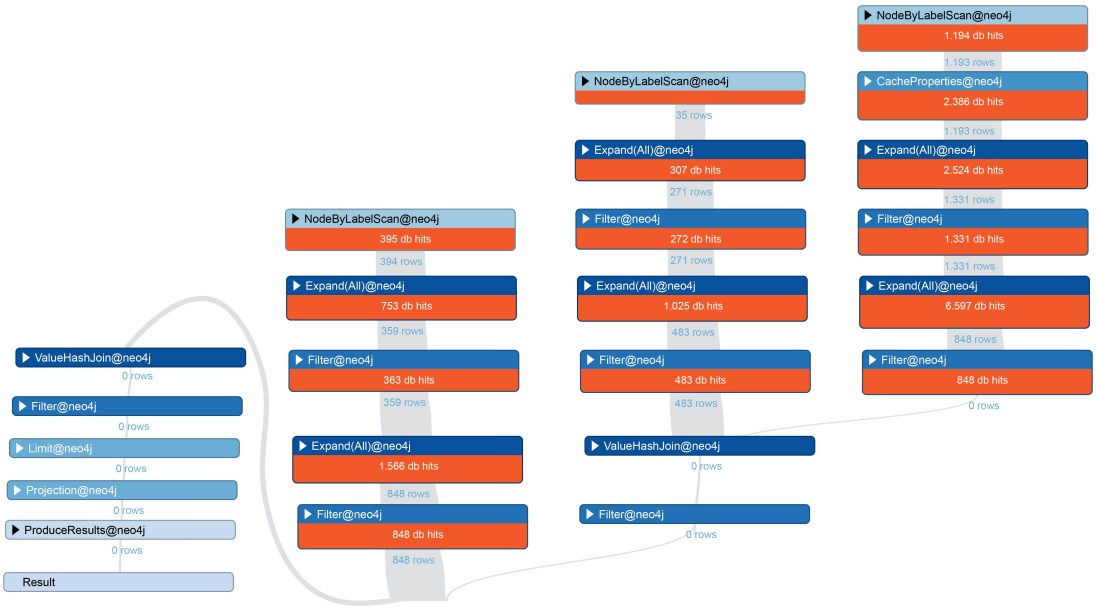


Figure 6.21: Profile query n.10

11. **Which medium does an author use the most used for publishing/presenting an article**

```
MATCH   (au1:Author)-[w1:WRITE]->(a:Article)-[pub1:PUBLISHED]-(b:Book)
WHERE   au1.orcid = "0000-0003-2125-327X"
WITH    count(pub1) AS BookCounter

MATCH   (au2:Author)-[w1:WRITE]->(a:Article)-[pub2:PUBLISHED]-(j:Journal)
WHERE   au2.orcid = "0000-0003-2125-327X"
WITH    count(pub2) AS JournalCounter, BookCounter

MATCH   (au3:Author)-[w1:WRITE]->(a:Article)-[pres:PRESENTED]-(c:Conference)
WHERE   au3.orcid = "0000-0003-2125-327X"
WITH    count(pres) AS ConferenceCounter, BookCounter, JournalCounter

Return BookCounter, JournalCounter,ConferenceCounter
```

The query counts how many articles "Michele Ruta" has published on a book, how many articles he has published on a journal, how many articles he has presented at a conference and then presents them in a table so that it is immediate to see which medium is used the most by "Michele Ruta".

Figure 6.22: Query n.11: results

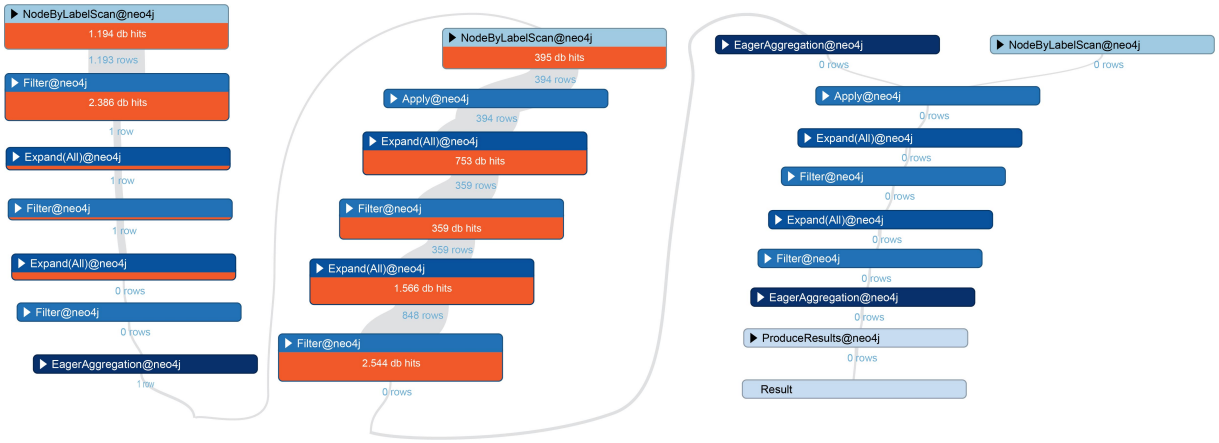


Figure 6.23: Profile query n.11

12. **Authors that have published more than two articles in a book before 2000**

```
MATCH (author:Author)-[WRITE]->(article:Article)-[PUBLISH]->(book:Book)
WHERE book.year < 2000
WITH author, size(collect(distinct book)) AS Books
WHERE Books>= 2
RETURN author.name AS Author, Books AS BookCounter
```

```
ORDER BY BookCounter DESC
```

In this query we first selected all the authors that wrote an article published in a book. After adding the year constraint a list of distinct books is created. Declaring that the list must be bigger than 2 means that an author needs to have published at least two times.

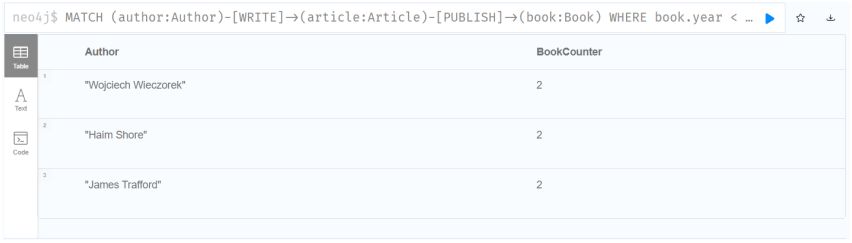


Figure 6.24: Query n.12

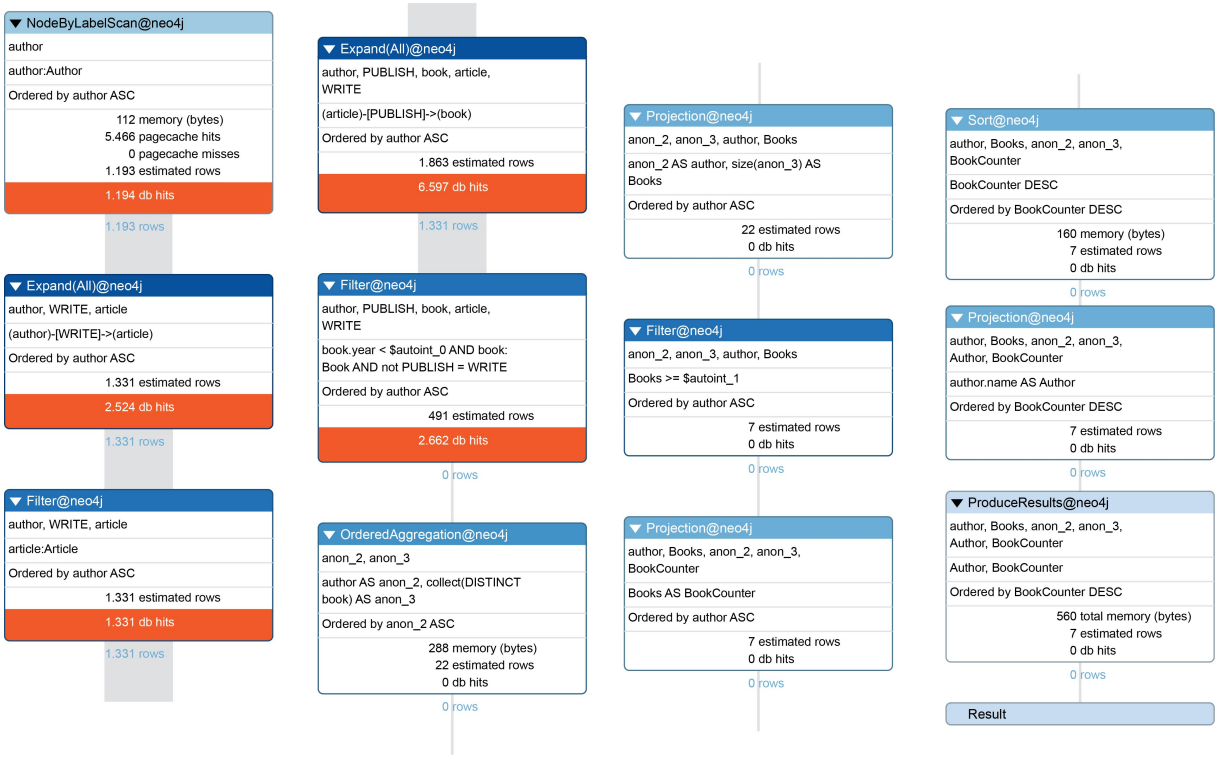


Figure 6.25: Profile query n.12