Project Work Part 1 – Data Design and Modelling

David Alarcórn Sepehr Beheshti Claudio Maggioni
Lorenzo Martingnetti

November 9, 2022

Contents

1 Introduction

This project is about storing the data from the *DBLP* publication repository[1] in a graph database implemented using Neo4J and the *Cypher* query language. To achieve this result, we have to understand the business domain of a publication repository, translate that knowledge in an Entity-Relationship model, and finally implement a Neo4J database. This consists of downloading the data in *DBLP*, design a way to import this data, define a Graph Model based on the ER Model we define, and finally to define some useful *Cypher* commands to both query and update the data.

[1] `https://dblp.org`

2 The DBLP Domain

As mentioned, DBLP is a publication repository, i.e. an index of metadata regarding various scientific publication mediums, such as journals and conference articles. DBLP stores titles, authors, references and where to find each publication, but not the publication document itself. Given the broad diversity in the kinds of documents indexed by DBLP, we decided to focus our attention on articles, books, thesis at both the Masters' and PhD level, and on author websites. Each publication record has several attributes representing the metadata of each publication, such as a "metadata version", an record version number internal to DBLP, a list of authors or the journal in which it was published. Different publications have different information attached to it, though all the publications we considered had a year of publication and a title attribute.

3 ER Model

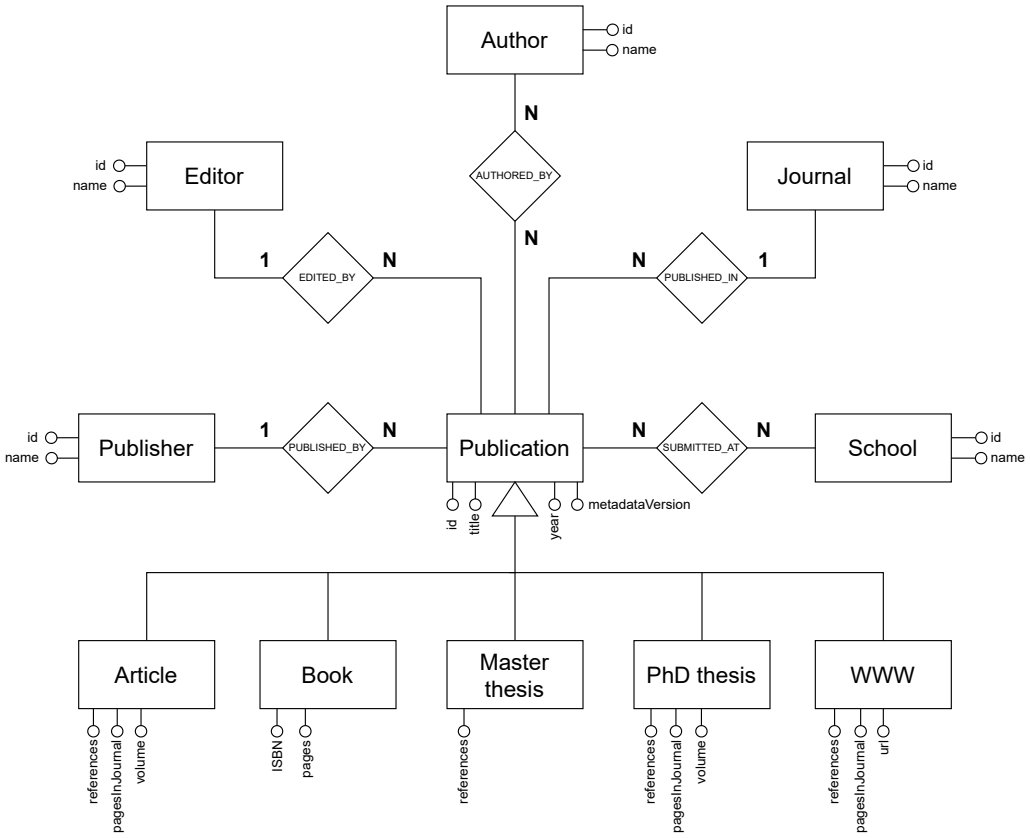

Figure 1: The ER diagram for the DBLP database. Cardinality annotation to be interpreted as look-across, e.g. a *Publication* is related to at most one *Editor*.

The ER model highlights the central concept of the DBLP knowledge: the publication. Each publication registered into the DBLP database is described by its title, year of publication, and metadata version (the version of the metadata stored into the DBLP database). Additionally, each publication can be one of the following:

- **Article**: it may contain *references* to other publications, it is published on a specific range of *pages in a journal*, which in turn may be part of a *volume*.

- **Book**: it is also identified by the *ISBN* code and it has a specific *number of pages*.
- **Master thesis**: it may contain *references* to other publications.
- **PhD thesis**: it may contain *references* to other publications, it may be published on a specific range of *pages in a journal*, which in turn may be part of a *volume*.
- **Website**: it may contain *references* to other publications, it may be published on a specific range of *pages in a journal* (an online journal), and it has a unique *url*.

The other entities, namely **Publisher**, **Editor**, **Author**, **Journal**, **School** are secondary to the DBLP problem specification, which is storing metadata related to publications. They are simply identified by a unique *id* and have a *name* associated. They interact with the publication entity through the relationships defined in the ER above, with the cardinalities specified. Note that no secondary entity has a one to one relationship with *Publication*: this justifies them to be separate entities and not simply attributes of *Publication*.

4 Neo4J Database Creation

4.1 The DBLP XML Data Format

DBLP periodically publishes copies of its entire database as *gzip*-compressed XML files. We decide to download one of such dumps as a data source for our database.

We then decide to use an existing parser for DBLP data, namely the *ThomHurks/dblp-to-csv* project on GitHub[2]. This parser is designed to convert the DBLP xml file into several pairs of CSV files, each of them containing column headers and data for one DBLP article type. A unique integer ID is assigned to each record, which is a feature we exploit when building our Neo4J model. Finally, through the `--relations` command line option, the parser is able to extract some of the publication attributes as separate "entities". Each chosen attribute has its data stored in a separate CSV file, and an additional CSV file is generated containing an ID to ID mapping between the source publication and the extrapolated record.

Listing 1 shows how we downloaded the XML database dump and how we parsed it using the *dblp-to-csv*.

[2] `https://github.com/ThomHurks/dblp-to-csv`

```
# download and decompress the DBLP XML dump
curl -o dblp.xml.gz https://dblp.org/xml/dblp.xml.gz
gzip -d dblp.xml.gz

# download the DTD specification of the DBLP XML format
curl -o dblp.dtd https://dblp.org/xml/dblp.dtd

# invoke the dblp-to-csv utility with the relations specified in the ER diagram
python3 ./XMLToCSV.py --annotate --neo4j dblp.xml dblp.dtd \
    output.csv --relations author:authored_by journal:published_in \
    publisher:published_by school:submitted_at editor:edited_by

# for each entity specified in the ER diagram
for t in article book mastersthesis phdthesis www; do
    # merge the column headers and data CSV files, and perform some substitutions
    # required by the Neo4J CSV parser
    tr ';' '\n' <output_${t}_header.csv | sed 's/:.*//g' | \
        tr '\n' ';' | awk 1 | cat - output_${t}.csv | \
        sed -E 's/\{?\\""\}?/""/g' > ${t}.csv;
done
```

Listing 1: Commands used to generated the CSV files necessary for import in Neo4J.

4.2 Graph Model

Given the generated CSV files and the ER model we define in Section 3, we are able to define a suitable graph diagram. Given the graph database techology we use is Neo4J, the notation we use in our graph diagram reflects the Neo4J notation rules for node and edge labels.

We decide to implement the total disjoint ISA relationship regarding publications by assigning to each publication node both a *:Publication* label and a label representing the specific publication type. This strategy is analogous to the use of a discriminant field in relational databases to define the schema of leaf entities in an ISA relationship, however using labels instead of attributes.

We then define a series of nodes that may be connected to a *:Publication*. These node types map one-to-one with the additional CSV files we extract with the *dblp-to-csv* tool with the `--relations` flag. We then define the edges that connect publications with these auxiliary nodes. Additionally, we define some de-facto cardinalities for each edge kind, that are however informal and not enforceable with a Neo4J constraint definition.

Figure 2 shows the resulting diagram.

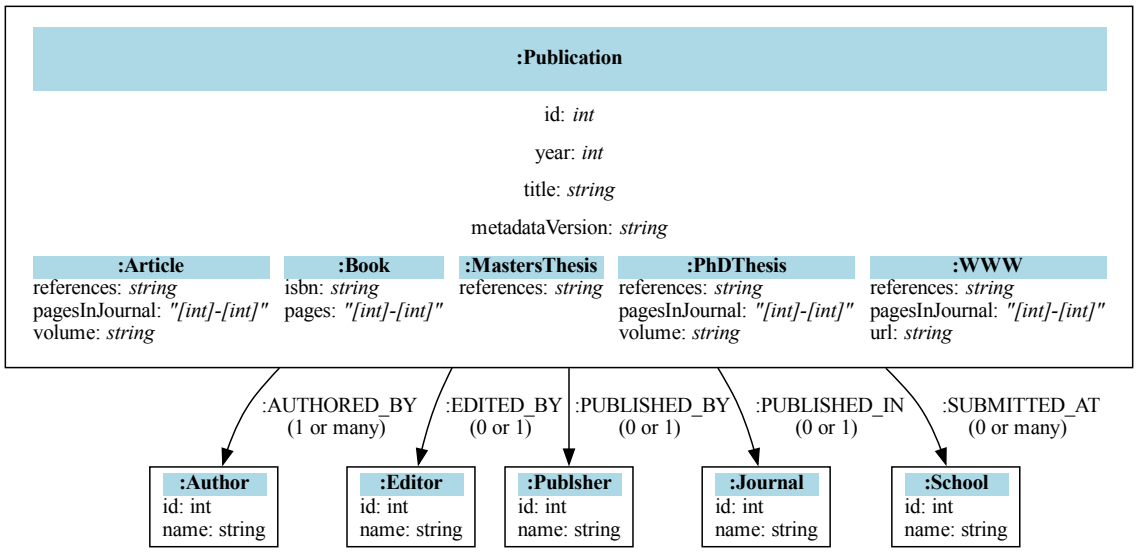

Figure 2: The graph diagram for the DBLP database. Neo4J node labels are highlighted in blue. Below each node label, attributes and their data types are listed. A *:Publication* label is always paired with one and only one of the *:Article*, *:Book*, *:MastersThesis*, *:PhDThesis*, *:WWW* labels. Below each edge label, the number of allowed edges with that label is specified (e.g. a *:Publication* is the source of at least one *:AUTHORED_BY* edge).

4.3 Import Queries

Given the defined Graph model and the generated CSV files, we can write the necessary Cypher queries to import the data in Neo4J.

Cypher is the language that the Neo4J DBMS supports for creating, manipulating and querying data. Cypher is to Neo4J what ANSI SQL is to a relational DBMS like PostgreSQL or MariaDB.

An important constraint we consider when implementing this project is collaboration. Since we have to work in a team while populating the Neo4J database and designing queries we require a way to share a single database instance. To achieve this, we use Neo4J *AuraDB* SaaS database service[3] to create a cloud hosted Neo4J instance and then share its credentials. However, since users cannot access the filesystem of an *AuraDB* instance, we upload the generated CSV files to the *maggioni.xyz* website (owned by Claudio Maggioni) and them instruct Neo4J to query the website and fetch the data.

Listing 2 shows the resulting Cypher import script. When executing the script through the Neo4J web interface, all statements starting with `LOAD CSV` must be prefixed with `:auto` to enable transaction support.

Lines 1-75 of the script contain the commands necessary to create all the publication nodes. As specified in Section 4.2, all publications are labeled with the *:Publication* label and another specific label matching the publication type. All `CREATE` statements are executed in transactions with 1000 record chunks to improve performance.

Lines 77-130 are instead dedicated to importing the "auxiliary" nodes, such as *:Author* and *:Journal* nodes. This lines use the `line['column']` notation for accessing CSV cells instead of the `line.column` notation used in the previous lines due to special characters added by *dblp-to-csv* in the header names.

[3] `https://neo4j.com/cloud/platform/aura-graph-database/`

Lines 139-187 create the edges between the publication and "auxiliary" entities by using the CSV association tables generated by *dblp-to-csv*. All statements are composed by two `MATCH` clauses, which match respectively the publication node and the "auxiliary" node, and a `CREATE` clause, which creates the actual edge and labels it accordingly.

Lines 132-137 define indexes for each publication type based on the `id` attribute, which is guaranteed to have unique values for each record by *dblp-to-csv*. These indexes improve significantly the performance of the edge creation statements, as the `MATCH` clauses are defined just over the `id` attribute and thus Neo4J would set forth an execution plan using the index. This improves the lookup algorithmic complexity from linear to logarithmic, as indexes are by default implemented using BTrees.

This section shows 2 kinds of data creation commands (namely the node creations at lines 1-130 and edge creations at lines 139-187), leaving us to implement 3 data manipulation statements.

```
1  LOAD CSV WITH HEADERS FROM
2  'https://maggioni.xyz/article.csv'
3  AS line FIELDTERMINATOR ';'
4  CALL {
5      WITH line
6      CREATE (:Publication:Article {
7          id: line.article,
8          references: line.ee,
9          metadataVersion: line.mdate,
10         pagesInJournal: line.pages,
11         title: line.title,
12         volume: line.volume,
13         year: line.year
14     })
15 } IN TRANSACTIONS OF 1000 ROWS;
16
17 LOAD CSV WITH HEADERS FROM
18 'https://maggioni.xyz/book.csv'
19 AS line FIELDTERMINATOR ';'
20 CALL {
21     WITH line
22     CREATE (:Publication:Book {
23         id: line.book,
24         title: line.title,
25         isbn: line.isbn,
26         metadataVersion: line.date,
27         pages: line.pages,
28         year: line.year
29     })
30 } IN TRANSACTIONS OF 1000 ROWS;
31
32 LOAD CSV WITH HEADERS FROM
33 'https://maggioni.xyz/mastersthesis.csv'
34 AS line FIELDTERMINATOR ';'
35 CALL {
36     WITH line
37     CREATE (:Publication:MastersThesis {
38         id: line.mastersthesis,
39         title: line.title,
40         references: line.ee,
41         metadataVersion: line.mdate,
42         year: line.year
43     })
44 } IN TRANSACTIONS OF 1000 ROWS;
```

```
45
46  LOAD CSV WITH HEADERS FROM
47  'https://maggioni.xyz/phdthesis.csv'
48  AS line FIELDTERMINATOR ';'
49  CALL {
50      WITH line
51      CREATE (:Publication:PhDThesis {
52          id: line.phdthesis,
53          references: line.ee,
54          metadataVersion: line.mdate,
55          pagesInJournal: line.pages,
56          title: line.title,
57          volume: line.volume,
58          year: line.year
59      })
60  } IN TRANSACTIONS OF 1000 ROWS;
61
62  LOAD CSV WITH HEADERS FROM
63  'https://maggioni.xyz/www.csv'
64  AS line FIELDTERMINATOR ';'
65  CALL {
66      WITH line
67      CREATE (:Publication:WWW {
68          id: line.www,
69          references: line.ee,
70          metadataVersion: line.mdate,
71          title: line.title,
72          year: line.year,
73          url: line.url
74      })
75  } IN TRANSACTIONS OF 1000 ROWS;
76
77  LOAD CSV WITH HEADERS FROM
78  'https://maggioni.xyz/output_author.csv'
79  AS line FIELDTERMINATOR ';'
80  CALL {
81      WITH line
82      CREATE (:Author {
83          id: line[':ID'],
84          name: line['author:string']
85      })
86  } IN TRANSACTIONS OF 1000 ROWS;
87
88  LOAD CSV WITH HEADERS FROM
89  'https://maggioni.xyz/output_editor.csv'
90  AS line FIELDTERMINATOR ';'
91  CALL {
92      WITH line
93      CREATE (:Editor {
94          id: line[':ID'],
95          name: line['editor:string']
96      })
97  } IN TRANSACTIONS OF 1000 ROWS;
98
99  LOAD CSV WITH HEADERS FROM
100 'https://maggioni.xyz/output_journal.csv'
101 AS line FIELDTERMINATOR ';'
102 CALL {
103     WITH line
```

```
    CREATE (:Journal {
        id: line[':ID'],
        name: line['journal:string']
    })
} IN TRANSACTIONS OF 1000 ROWS;

LOAD CSV WITH HEADERS FROM
'https://maggioni.xyz/output_publisher.csv'
AS line FIELDTERMINATOR ';'
CALL {
    WITH line
    CREATE (:Publisher {
        id: line[':ID'],
        name: line['publisher:string']
    })
} IN TRANSACTIONS OF 1000 ROWS;

LOAD CSV WITH HEADERS FROM
'https://maggioni.xyz/output_school.csv'
AS line FIELDTERMINATOR ';'
CALL {
    WITH line
    CREATE (:School {
        id: line[':ID'],
        name: line['school:string']
    })
} IN TRANSACTIONS OF 1000 ROWS;

CREATE INDEX FOR (p:Publication) ON (p.id);
CREATE INDEX FOR (p:Author) ON (p.id);
CREATE INDEX FOR (p:Editor) ON (p.id);
CREATE INDEX FOR (p:Journal) ON (p.id);
CREATE INDEX FOR (p:Publisher) ON (p.id);
CREATE INDEX FOR (p:School) ON (p.id);

LOAD CSV WITH HEADERS FROM
'https://maggioni.xyz/output_author_authored_by.csv'
AS line FIELDTERMINATOR ';'
CALL {
    WITH line
    MATCH (p:Publication { id: line[':START_ID'] })
    MATCH (a:Author { id: line[':END_ID'] })
    CREATE (p)-[:AUTHORED_BY]->(a)
} IN TRANSACTIONS OF 1000 ROWS;

LOAD CSV WITH HEADERS FROM
'https://maggioni.xyz/output_editor_edited_by.csv'
AS line FIELDTERMINATOR ';'
CALL {
    WITH line
    MATCH (p:Publication { id: line[':START_ID'] })
    MATCH (a:Editor { id: line[':END_ID'] })
    CREATE (p)-[:EDITED_BY]->(a)
} IN TRANSACTIONS OF 1000 ROWS;

LOAD CSV WITH HEADERS FROM
'https://maggioni.xyz/output_journal_published_in.csv'
AS line FIELDTERMINATOR ';'
CALL {
```

```
    WITH line
    MATCH (p:Publication { id: line[':START_ID'] })
    MATCH (a:Journal { id: line[':END_ID'] })
    CREATE (p)-[:PUBLISHED_IN]->(a)
} IN TRANSACTIONS OF 1000 ROWS;

LOAD CSV WITH HEADERS FROM
'https://maggioni.xyz/output_publisher_published_by.csv'
AS line FIELDTERMINATOR ';'
CALL {
    WITH line
    MATCH (p:Publication { id: line[':START_ID'] })
    MATCH (a:Publisher { id: line[':END_ID'] })
    CREATE (p)-[:PUBLISHED_BY]->(a)
} IN TRANSACTIONS OF 1000 ROWS;

LOAD CSV WITH HEADERS FROM
'https://maggioni.xyz/output_school_submitted_at.csv'
AS line FIELDTERMINATOR ';'
CALL {
    WITH line
    MATCH (p:Publication { id: line[':START_ID'] })
    MATCH (a:School { id: line[':END_ID'] })
    CREATE (p)-[:SUBMITTED_AT]->(a)
} IN TRANSACTIONS OF 1000 ROWS;
```

Listing 2: The Cypher instructions necessary to define and import the extracted DBLP database dump.

5 Queries

In this section we show the queries we have implemented. For performance reference, all queries are run on a free tier Neo4J AuraDB instance over a subset of the DBLP data composed of 12.150 nodes and 6.216 edges.

5.1 Publications that May Be Hosted on Secure Websites

Query complexity: 3 nodes, conditions, aggregations

This query returns the total number of publications of type *Article* whose author has a HTTPS website.

```
1 MATCH (p:Publication:Article)-[:AUTHORED_BY]->(a:Author)<-[:AUTHORED_BY]-(www:WWW)
2 WHERE www.url STARTS WITH 'https://'
3 RETURN COUNT(DISTINCT p) as publisherCount
```

Listing 3: The Cypher code for the Morgan & Claypool's Journal Relationships query.

By invoking the query with the `PROFILE` clause, Neo4J reports the following execution plan:

- All possible matches for `(www:WWW) WHERE www.url STARTS WITH 'https://'` are found;
- The `:AUTHORED_BY` edge expression finding all candidates for `a`;
- *a* candidates are filtered to match the `:Author` label;
- The `:AUTHORED_BY` edge expression finding all candidates for `p`;
- *p* candidates are filtered to match the `:Publication` and `:Article` labels;
- Publishers are counted;

For the execution of this query, Neo4J reports 2840 total db hits in 89 ms.

5.2 Author Count in Publications

Query complexity: 3 nodes, conditions, aggregations

This query considers the journal "AI Mag." it queries the authors of its publications. The query then returns the author count of author whose name does not contain the letter 'M'. Results are ordered by author count.

```
1 MATCH (j:Journal {name: 'AI Mag.'})<-[:PUBLISHED_IN]-(p:Publication),
2       (p)-[:AUTHORED_BY]->(a:Author)
3 WHERE NOT a.name =~ '.*[mM].*'
4 RETURN p.title, count(a) AS authorCount
5 ORDER BY authorCount DESC
```

Listing 4: The Cypher code for the Author Count in Publications query.

By invoking the query with the `PROFILE` clause, Neo4J reports the following execution plan:

- All possible matches for `(j:Journal name: 'AI Mag.')` are found;
- The `:PUBLISHED_IN` edge expression finding all candidates for `p`;
- *p* candidates are filtered to match the `:Publication` label;
- The `:AUTHORED_BY` edge expression finding all candidates for `a`;
- *a* candidates are filtered to match `(a:Author) WHERE NOT a.name = '.*[mM].*'`, i.e. to find authors whose name does not contain an uppercase or lowercase m;

- Authors are counted for each publication;
- Results are sorted by decreasing author count and then returned.

For the execution of this query, Neo4J reports 1866 total db hits in 84 ms.

5.3 Morgan & Claypool's Journal Relationships

Query complexity: 5 nodes, conditions, aggregations, limits

This query considers the publisher whose name starts with "Morgan & Claypool" (only one publisher matches this criteria in our database instance). We then query all the authors of publications published with this publisher. We then want to find all the publications **not** published with "Morgan & Claypool", and of these publications we want to find journals where they were published in. The top 10 results by author count (i.e. the number of distinct authors involved with both the publisher and the journal) are returned, and each result yields the publisher name, the journal name, and the author count.

```
1 MATCH (p:Publisher)<-[:PUBLISHED_BY]-(pub1:Publication)-[:AUTHORED_BY]->(a:Author),
2       (a)<-[:AUTHORED_BY]-(pub2:Publication)-[:PUBLISHED_IN]->(j:Journal)
3 WHERE p.name STARTS WITH 'Morgan & Claypool'
4 RETURN p.name, j.name, count(DISTINCT a) as authorCount
5 ORDER BY authorCount DESC
6 LIMIT 10
```

Listing 5: The Cypher code for the Morgan & Claypool's Journal Relationships query.

By invoking the query with the `PROFILE` clause, Neo4J reports the following execution plan:

- All possible matches for `(p:Publisher) WHERE STARTS WITH 'Morgan & Claypool'` are found;
- The `:PUBLISHED_BY` edge expression finding all candidates for `pub1`;
- *pub1* candidates are filtered to match the `:Publication` label;
- The `:AUTHORED_BY` edge expression finding all candidates for `a`;
- *a* candidates are filtered to match the `:Author` label;
- The `:PUBLISHED_BY` edge expression finding all candidates for `pub2`;
- *pub2* candidates are filtered to match the `:Publication` label and `pub1 != pub2`;
- The `:PUBLISHED_IN` edge expression finding all candidates for `j`;
- *j* candidates are filtered to match the `:Journal` label;
- The authors are counted;
- Results other than the top 10 by count are discarded, and the remaining matches are sorted by descending author count.

For the execution of this query, Neo4J reports 68 total db hits in 110 ms.

5.4 Author Chains

Query complexity: 5 nodes, conditions, aggregations, limits

This query finds the longest chains between authors whose name contain "Jones" and all other authors. The chain edges must all be of type *AUTHORED_BY* except one, whose type must be *PUBLISHED_IN*. Results are sorted in descending order by the length of the respective chains (more specifically by the total number of *AUTHORED_BY* hops), and the top 10 results are picked. The query returns for each result the names of the two authors, the number of *AUTHORED_BY* hops starting from the "Jones" authors (`startHopCount`) and the number of *AUTHORED_BY*

hops ending on the other author (*endHopCount*).

```
MATCH (a:Author)-[r:AUTHORED_BY*..20]-(p1:Publication)-[:PUBLISHED_IN]->(j:Journal),
      (j)<-[:PUBLISHED_IN]-(p2:Publication)-[s:AUTHORED_BY*..20]-(t:Author)
WHERE a.id <> t.id AND t.name CONTAINS 'Jones'
RETURN a.name, t.name, COUNT(r) as endHopCount, COUNT(s) as startHopCount
ORDER BY startHopCount + endHopCount DESC, a.name ASC
LIMIT 10
```

Listing 6: The Cypher code for the Author Chains query.

By invoking the query with the `PROFILE` clause, Neo4J reports the following execution plan:

- All possible matches for `(p:Author) WHERE t.name CONTAINS 'Jones'` are found;
- The variable length `:AUTHORED_BY` edge expression is expanded to find all candidates for `p2`;
- The `p2` candidates are filtered for the `:Publication` label;
- `:PUBLISHED_BY` edges are traversed to find all candidates for `j`;
- The `j` candidates are filtered for the `:Journal` label;
- `:PUBLISHED_BY` edges are traversed to find all candidates for `p1`;
- The `p1` candidates are filtered for the `:Publication` label and for `p1 <> p2`;
- The variable length `:AUTHORED_BY` edge expression is expanded to find all candidates for `a`;
- `a` is checked to match `(a:Author) WHERE a.id != t.id`;
- `COUNT(r)` and `COUNT(s)` are computed;
- `COUNT(r) + COUNT(s)` is computed, and then the results are sorted and truncated according to the provided clauses.

For the execution of this query, Neo4J reports 34367 total db hits in 112 ms.

5.5 Shortest Author-to-Author Chain

Query complexity: functions

This query finds the shortest path between each author whose name contain "Jones" and each other author. Unlike the Author Chains query, the query does not constrain the edges in the path to be of any particular type. The query returns the top 10 results by path length, yielding for each result the names of both authors, the length of the path, and a list of nodes in the path (each element being a pair of the node's label and the node's name/title).

```
MATCH (t:Author) WHERE t.name CONTAINS 'Jones'
MATCH (a:Author) WHERE a.id <> t.id
MATCH path = shortestPath((a)-[*..20]-(t))
RETURN t.name, a.name, length(path), [
    x IN nodes(path) |
    [reduce(s = '', x IN labels(x) | s + ':' + x), COALESCE(x.name, x.title)]
] AS nodes
ORDER BY length(path) DESC
LIMIT 10
```

Listing 7: The Cypher code for the Shortest Author-to-Author Chain query.

By invoking the query with the `PROFILE` clause, Neo4J reports the following execution plan:

- All `t`s matching `(t:Author) WHERE t.name CONTAINS 'Jones'` are found;

- In parallel with the previous step, all *a*s matching *(a:Author)* are found;
- The Cartesian product of `a`s and `t`s is computed;
- `a`s and `t`s are filtered by `a.id != t.id`;
- The shortest path between each `a` and `t` is computed;
- The length of the path is computed;
- All but the top 10 results by path length are discarded;
- The expression yielding the `nodes` property is computed;
- The resulted are ordered by descending path length.

For the execution of this query, Neo4J reports 100681 total db hits in 1247 ms.

5.6 Number of Authors Who Recently Published With a Publisher

Query complexity: 3 nodes, conditions, aggregations

This query finds the number of authors who recently published with a publisher, and we define recently to mean a publication year greater than 2010.

```
1 MATCH (p:Publisher)<-[pb:PUBLISHED_BY]-(p2:Publication:Book),
2       (a:Author)<-[ab:AUTHORED_BY]-(p2)
3 WHERE p.name = 'Springer' AND toInteger(p2.year) >= 2010
4 RETURN COUNT(a)
```

Listing 8: The Cypher code for the Number of Authors Who Recently Published With a Publisher query.

By invoking the query with the `PROFILE` clause, Neo4J reports the following execution plan:

- All `p2`s matching `(p2:Author) WHERE toInteger(p2.year) >= 2010` are found;
- `:PUBLISHED_BY` edges are traversed to find all candidates for `p`;
- The `p` candidates are filtered for the `:Publication` label and for `p.name = 'Springer'`;
- The `:AUTHORED_BY` edge expression finding all candidates for `a`;
- *a* candidates are filtered to match the `:Author` label;
- Authors are counted.

For the execution of this query, Neo4J reports 86 total db hits in 38 ms.

5.7 Authors of a Journal Whose Publication Metadata Are Not Up to Date

Query complexity: 3 nodes, conditions

```
1 MATCH (a:Author)<-[ab:AUTHORED_BY]-(p:Publication),
2       (j:Journal)<-[pi:PUBLISHED_IN]-(p:Publication)
3 WHERE toInteger(p.year) < date(p.metadataVersion).year AND j.name = 'Theor. Comput. Sci.'
4 RETURN a
```

By invoking the query with the `PROFILE` clause, Neo4J reports the following execution plan:

- All possible matches for `(j:Journal) WHERE j.name = 'Theor. Comput. Sci.'` are found;
- The `:PUBLISHED_IN` edge expression is expanded to find all candidates for `p`;
- The `p` candidates are filtered for the `:Publication` label, and also filtered selecting only those with year lower than that of the metadata version;

- The `:AUTHORED_BY` edge expression is expanded to find all candidates for `a`;
- The `a` candidates are filtered for the `:Author` label;
- The result is ordered by `a.name`.

For the execution of this query, Neo4J reports 1813 total db hits in 98 ms.

5.8 Find a book about Image processing published by Springer

Query complexity: 3 nodes, 3 conditions

This query finds a book about image processing published by Springer after 2000 finds also the author.

```
1 MATCH (p:Publisher)-[r]-(b:Book)-[r2]-(a:Author)
2 WHERE b.year > '2000' and p.name = 'Springer' and b.title contains 'image processing'
3 RETURN p,r,b,r2,a, b.title, a.name
```

Listing 9: The Cypher code for the finding a book about image processing and the author.

By invoking the query with the `PROFILE` clause, Neo4J reports the following execution plan:

- **Cypher version: , planner: COST, runtime: PIPELINED. 93 total db hits in 93 ms**;
- Define the initial candidates by considering all publishers;
- Filter the publishers by considering "Springer" as only interested publisher;
- Expanding the rows by adding the details of the books linked to the publisher;
- Filter the book which are published in 2000 or later and the title are related to image processing;
- Adding the authors data to the books;
- checking that the relationships of publisher and authors are different (relationship with books);
- Prepare and project nodes and links to be visualized;
- Produce results which will be the list of book/books with authors name;

5.9 Find a list and details of publication/thesis related to algorithms

Query complexity: 3 nodes, conditions

This query finds list of publications/thesis related to algorithms that are submited at school before 2009 .

```
1 MATCH (s:School)-[r:SUBMITTED_AT]-(t)-[r2]-(a:Author)
2 WHERE t.title contains 'algorithm' and t.year <='2009'
3 RETURN t.title,a.name,t.year,s,r,t,r2,a
```

Listing 10: The Cypher code for the finding of thesis related to algorithms published before 2009.

By invoking the query with the `PROFILE` clause, Neo4J reports the following execution plan:

- **Cypher version: , planner: COST, runtime: PIPELINED. 304 total db hits in 36 ms**;
- Find the initial candidates by considering all the thesis which are submitted at Schools;
- Filter the thesis that are submitted in 2009 or after and are related to "algorithms";

- Adding the details of authors to the thesis;
- Checking that the relationships of authors be different from relationships of schools (with thesis);
- Project all nodes and attributes which we are interested to visualize;
- Produce final results which is the list of articles, authors and the publication year;

5.10 Find publications about network with higher number of authors

Query complexity: 3 nodes, conditions, aggregation

This query Find number of publications related to network which are published after 2005 and each one has more than 6 authors .

```
1 MATCH (a:Author)-[r0]-(p:Publication)-[r:PUBLISHED_IN]->(j:Journal)
2 WHERE toLower(p.title) contains 'network' and p.year >= '2005'
3 WITH p, count(a) as cnt
4 WHERE cnt > 6
5 RETURN count(p.title) as Nr_publications
```

Listing 11: The Cypher code for finding publications about network with at least 6 authors.

By invoking the query with the `PROFILE` clause, Neo4J reports the following execution plan:

- **Cypher version: , planner: COST, runtime: PIPELINED. 8234 total db hits in 38 ms**;
- Considering the connection of publication to Journals through Published_IN define all possible matches;
- Apply the filter on year of publication and considering that the articles title must include "network";
- Expanding the candidates by adding the nodes of Journals and authors;
- Control that the relationships publications with authors and journal be different;
- Count the number of authors for each publication;
- Filter the publications based on their authors which must bu higher than 6;
- Aggregation: counting total number of articles which have at least 6 authors;

5.11 Find number of authors who have published a book and also an article on journals

Query complexity: 5 nodes, conditions, aggregation

This query Find number of authors who has written a book about user modeling published on ."Morgan & Claypool Publishers" and published also an article on a Journal in the field of artificial intelligence.

```
1 MATCH (a:Author)-[r0]-(b:Book),(b)-[r]-(pb:Publisher)
2 ,(a)-[r2]-(ar:Article)-[r3]-(j:Journal)
3 WHERE toLower(ar.title) contains 'user modeling'
4 and pb.name = 'Morgan & Claypool Publishers' and toLower(j.name) contains 'artif'
5 RETURN count(distinct a.name) as Nr_Author
```

Listing 12: The Cypher code for finding nr of authors who has published a book and also an article.

By invoking the query with the `PROFILE` clause, Neo4J reports the following execution plan:

- **Cypher version: , planner: COST, runtime: PIPELINED. 77 total db hits in 122 ms**;
- All possible matches for pb:Publisher where pb.name = "Morgan & Claypool Publishers" are found;
- Expand the path by linking books to publications;
- Filter books linked to the specific publisher;
- The candidates are filtered that the relationships r and r0 be different;
- The title of article is checked which must contain "user modeling";
- Expand the path by adding Journal node to Article;
- Apply filter of relationship and Article to the Journals;
- Filter the journals by considering toLower(j.name) contains 'artif';
- Count the unique authors who matches all the criteria;

5.12 Update the MetadataVersion of a Publication

Query complexity: data update

This query updates the version of the metadata of a specific publication, for which the title is given.

```
1 MATCH (p:Publication)
2 WHERE p.title = 'On the Combinatorics of Finite Words.'
3 SET p.metadataVersion = '2022-11-07'
```

Listing 13: The Cypher code to update the version of a publication metadata.

By invoking the query with the `PROFILE` clause, Neo4J reports the following execution plan:

- All possible matches for `(p:Publication) WHERE p.title = 'On the Combinatorics of Finite Words.'` are found;
- The `metadataVersion` property is set to '2022-11-07';
- an empty result is displayed, since values are only set;

For the execution of this query, Neo4J reports 7199 total db hits in 19 ms.

5.13 Publishing an Article on a Journal and removing it

Query complexity: data creation/removal

The first query publishes an article on a journal. The second one removes an article from a journal. The inputs of both queries are the name of the journal and the title of the article. Note that only the relationship between the nodes is modified in both cases: this means that the article must be already present in the database upon publication, while it is not removed from the database upon removal of it from the journal.

```
MATCH (j:Journal)
WHERE j.name = 'Theor. Comput. Sci.'
MATCH (p:Publication:Article)
WHERE p.title = 'On the least number of palindromes in two-dimensional words.'
MERGE (j)<-[pi:PUBLISHED_IN]-(p)
```

Listing 14: The Cypher code to publish an article on a journal.

By invoking the query with the `PROFILE` clause, Neo4J reports the following execution plan:

- All possible matches for `(p:Publication) WHERE p.title = 'On the least number of palindromes in two-dimensional words.'` are found;
- All possible matches for `(j:Journal) WHERE j.name = 'Theor. Comput. Sci.'` are found;
- It is performed a cartesian product *(p,j)* with the results of the two queries;
- In parallel, the pattern *(j)¡-[pi:PUBLISHED_IN]-(p)* is created and then applied to the cartesian product to create the new relationship, if this one exists (this is ensured by the LOCK(j,p) statement, where j and p are the previouslyy selected nodes);
- An empty result is created and displayed.

For the execution of this query, Neo4J reports 7487 total db hits in 47 ms.

```
MATCH (j:Journal)<-[pi:PUBLISHED_IN]-(p:Publication:Article)
WHERE j.name = 'Theor. Comput. Sci.'
    AND p.title = 'On the least number of palindromes in two-dimensional words.'
DELETE pi
```

Listing 15: The Cypher code to remove an article from a journal.

By invoking the query with the `PROFILE` clause, Neo4J reports the following execution plan:

- All possible matches for `(j:Journal) WHERE j.name = 'Theor. Comput. Sci.'` are found;
- The `:AUTHORED_BY` edge expression is expanded to find all candidates for `p`;
- The `p` candidates are filtered for the `:Publication` and `:Article` labels, and also filtered for `p.title = 'On the least number of palindromes in two-dimensional words.'`;
- The found set of relationship `pi` is deleted (only one in our case);
- An empty result is created and displayed.

For the execution of this query, Neo4J reports 1675 total db hits in 3 ms.