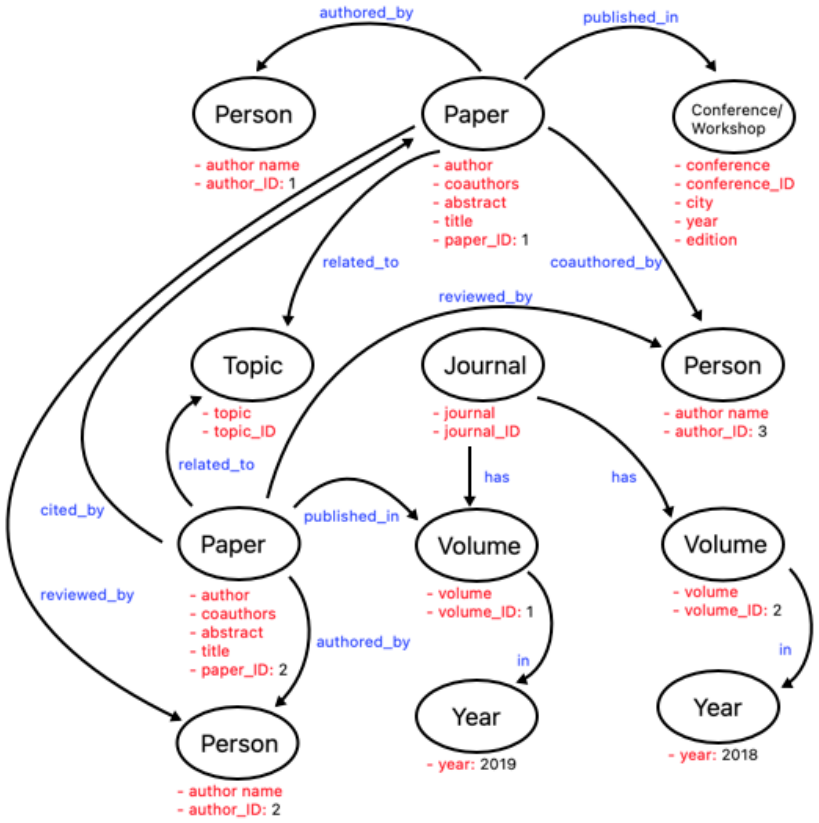
Section A.1: Modeling

In the image below we have the visual representation of how we have created our network:



For this project we have created the following nodes: Paper, Person, Topic, Workshop/Conference, Year, Volume and Journal. As we can see, in blue we have the different relationships that exist between nodes, and in red we have the attributes that exist for each one.

There are some repeated nodes (e.g. *Person*) in the graph above (equal type) specifying with the *id* attribute that they are different instances. This happens for some nodes that can have different relationships (e.g. *Person<-[]-Article*) with the same node, but in case they have one of these relationships (e.g. *authored_by*), they can not have the other one (e.g. *reviewed_by*).

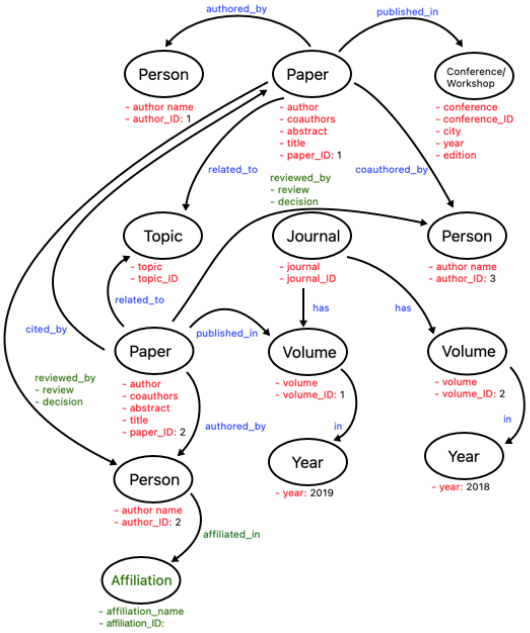
We assume Paper can be published in Journals (in different Volumes of a Journal) or in Conferences but not in both places at the same time. Regarding the Topic nodes, we assume that each article is *related_to* one or more topics.

We have also taken into account the queries of section B of the project statement, for instance, the modeling of the journals is done as follows: a journal has a volume which has been published in a specific year. This is modeled this way in order to ease the task of calculating the papers published and cited for a year in order to answer the query number 3. However, modeling journals as it is proposed, may create difficulties in retrieving information about conferences and journals at the same time since these proceedings are modeled differently.

Section A.2: Instantiating/Loading

The loaded data in our graph database comes from DBLP with some manual modifications and adding new manually generated data.
Before loading the data to Neo4j we had to modify and create more data. We have generated the final CSVs with Python (we attach the .py file to the delivery).
For the Articles table, we have used real data but we have simplified it to have only 300 different articles and removed the columns not used.
Regarding the *Topics*, we have generated the table ourselves by adding some topics, and then we have also created a table of relations between articles and topics to know each article to which topic/s it is related.
We have also generated two tables, one for *Authors* and one for *Co-authors*, from the actual data we already have. And also generated two more tables to create the relationships between the different papers and the authors and co-authors. We assume that each paper has a single author, and may have several co-authors. Therefore, we considered that a paper can have a maximum of one coauthor to keep things simple.
We have decided that of the 300 articles that we had initially: 150 will be published in Journals and 150 in Conferences, to have them equally shared. In the articles, we have taken advantage of a column we had called *Journal*, and we have also used one of the tables directly downloaded initially from DBLP in which there was the list of Journals with their respective IDs. In our case, the 150 articles were finally published in only 3 Journals. In addition, each Journal has one or several volumes, and several volumes can be published in a year. So we wanted to create a table for *Volumes* and another one for *Years* so that each one is a node, in addition to the tables of relations between *Article* and *Volume*, between *Volume* and *Year*, and between *Volume* and *Journal*.
Regarding conferences, we have taken the remaining 150 articles. In this case, the data has been generated by us: each of the conferences, with its city, the edition number and the year. We have done it in such a way that each conference has 4 editions taking into account the queries for the next sections. We have also created the table of relationships between each of the conferences and each of the 150 published articles.
For the reviewers part, we decided to create our own generated relationships between Persons and Articles, taking into account that the author or co-author of an article cannot be a reviewer at the same time of the same article.

Section A.3: Evolving the graph



In this section we were asked to evolve the graph in order to model the review written by the reviewer to a certain paper, the suggested decision of a reviewer and the affiliation which an author or a coauthor belongs to. The changes of the graph model are highlighted in green. To store the review and the suggested decision of the reviewer, we decided to add them as attributes of the edge REVIEWED_BY as we do not need to create them as nodes because the pattern (Paper)-[Reviewed_by]->(Person) it is a relation one to many but each review and decision it is particular for a paper and they don't have to be related with other papers, thus, it can directly added to the edge updating it. Regarding the affiliations of the different authors and coauthors, they are added as nodes. It only was needed to create the nodes of different companies and universities and create a relation one to one with the nodes Person. It was considered to add them as nodes to query data from the different papers and authors of a particular affiliation. For instance, calculate the number of published papers in a certain year of an affiliation (see the example in the Python application).

Section B: Querying

Query 1:

```
MATCH (p2:Paper)-[r1:IS_PUBLISHED_IN]->(c:Conference)
WITH p2, r1, c, size(()-[:CITED_BY]->(p2)) AS counter
WITH c, p2.paper_ID, counter ORDER BY counter DESC
RETURN c, collect(p2.paper_ID)[0..3]
```

query 2.:

```
MATCH
(a:Person)<-[au:AUTHORED_BY]-(paper:Paper)-[p:IS_PUBLISHED_IN]->(c:Conference
) WITH a.author AS author,c.conference as conference,count(DISTINCT
c.edition) as edition WHERE edition>3 with author, conference, edition
ORDER BY edition
RETURN conference, collect(author)
```

Query 3:

```
MATCH
(p1:Paper)-[c:CITED_BY]->(p2:Paper)-[pu:IS_PUBLISHED_IN]->(v:Volume)-[i:IN]->
(y:Year)
CALL { WITH v MATCH (j:Journal)-[h:HAS]-(v1:Volume{volume_ID:v.volume_ID})
RETURN j,v1 }
WITH j.journal as journal, y.year as year, count(p1) as citations
MATCH
(p3:Paper)-[pu1:IS_PUBLISHED_IN]->(v2:Volume)-[i1:IN]->(y1:Year{year:toString
(toInteger(year)-1)})
CALL { WITH v2,journal MATCH
(j:Journal{journal:journal})-[h:HAS]-(v3:Volume{volume_ID:v2.volume_ID})
RETURN j,v3}
WITH citations,y1.year as year1, count(p3) as publications1, journal, year
MATCH
(p4:Paper)-[pu2:IS_PUBLISHED_IN]->(v4:Volume)-[i2:IN]->(y2:Year{year:toString
(toInteger(year1)-1)})
CALL {WITH v4,journal MATCH
(j:Journal{journal:journal})-[h:HAS]-(v5:Volume{volume_ID:v4.volume_ID})
RETURN j,v5}
RETURN journal,year, toFloat(citations)/toFloat(publications1+count(p4)) as
impactfactor
```

Query 4:

```
MATCH (a:Person)<-[au:AUTHORED_BY]-(p1:Paper)-[p:CITED_BY]->(p2:Paper)
WITH a.author as author,p1.title as paper,count(p2) as counter
ORDER BY counter
WITH author, collect(counter) as f with author, [x in range(1,size(f)) where
f[x-1]>=x] as a UNWIND a as b
MATCH author, MAX(b)
```

Section C: Graph algorithms

The first chosen graph algorithm and the written calls that trigger it correctly is the similarity function using the function name `gds.similarity.jaccard`. This algorithm returns the degree of similarity between two nodes. In this way it allows us to compare several nodes and to know which are more similar and which are less similar to each other.
In the case of our call, we wanted to find the list of authors who are most similar to the author named "*Douglas Walton*" taking into account the different topics of the papers written by this author. That is, we looked at the authors who have written similar papers taking into account the topics (keywords).

Here is the code we have used to implement this algorithm in Neo4j:

```
MATCH(p1: Person {author: 'Douglas
Walton'})<-[r1:AUTHORED_BY]-(p:Paper)-[r2:RELATED_TO]->(t1: Topic)
WITH p1, collect(id(t1)) as point1
MATCH(p2: Person)<-[r3:AUTHORED_BY]-(p:Paper)-[r4:RELATED_TO]->(t2: Topic)
WHERE p1 <> p2
WITH p1, p2, point1, collect(id(t2)) as point2
RETURN p1.author AS from, p2.author as to,
gds.alpha.similarity.jaccard(point1, point2) as jaccard
ORDER BY jaccard DESC LIMIT 20;
```

The results obtained are as follows:

"from"	"to"	"jaccard"
"Douglas Walton"	"Jessica Li"	0.6666666666666666
"Douglas Walton"	"Sarah R. Davies"	0.5
"Douglas Walton"	"Brian Bowe"	0.5
"Douglas Walton"	"Ali Shirzadi"	0.5
"Douglas Walton"	"William J. Frey"	0.5

Looking at the results obtained according to Neo4j, we szee that Douglas Walton has written a paper that is related to two topics: Multivariate Analysis and Data Science. We see that the authors who have written more similar Papers considering the Topic are "*Jessica Li*" and "*Sarah R. Davies*".
We have seen that "*Jessica Li*" has written a paper related to 3 topics (Multivariate Analysis, Data Science, Machine Learning), two of them are the same as "*Douglas Walton*". Regarding "*Sarah R. Davies*" has written a paper that is related to the topic of Data Science.
The second algorithm we have chosen to call is PageRank, one of the Centrality Algorithms. This algorithm measures the importance of each node within a graph, based on the number of incoming relationships and the importance of the corresponding source nodes. In our case, we wanted to evaluate how important are each of the different Paper nodes taking into account the "CITED_BY" relationship.

This is the code we have used to call the algorithm to obtain the top 10 most important nodes:

```
CALL gds.graph.create('pagerank', 'Paper', 'CITED_BY')
CALL gds.pageRank.stream('pagerank')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).title AS name, score
ORDER BY score DESC, name limit 10
```

The result obtained is as follows:

name	score
"Defining Nano, Nanotechnology and Nanomedicine: Why Should It Matter?"	5.883425132452689
"The Perverse Effects of Competition on Scientists' Work and Relationships."	5.461463323596753
"Science, Technology and Innovation as Social Goods for Development: Rethinking Research Capacity Building from Sen's Capabilities Approach."	5.019397930563224

The most important node according to our data is the one with the highest score, in our case the paper with the title "*Defining Nano, Nanotechnology and Nanomedicine: Why Should It Matter?*" with a score of 5.88, followed by the paper entitled "*The Perverse Effects of Competition on Scientists*' Work and Relationships." with a score of 5.46. This means that the paper most cited by other papers in our dataset is the one with a score of 5.88.

Section D: Recommender

- Find/define the research communities:
  ```
  CREATE (:Community{name:'Database community'})
  MATCH (t:Topic), (c:Community)
  ```

```
WHERE t.topic IN ['Data Managment', 'Indexing', 'Data Modelling', 'Big
Data',
'Data Processing', 'Data Storage', 'Data Queryng'] AND
c.name='Database   WITH t, c
CREATE (t)-[:is_in]->(c) return t,c
```

- Find the conferences and journals related to the database community:

```
MATCH (co:Conference)<-[pu:IS_PUBLISHED_IN]-(p:Paper) WITH  co as
conference, toFloat(count(p)) as papers_published
MATCH
(co1:Conference)<-[pu1:IS_PUBLISHED_IN]-(p1:Paper)-[r1:RELATED_TO]->(t
1:Topic)-[i1:is_in]->(c1:Community)
WHERE co1.conference_ID=conference.conference_ID
WITH c1, co1, toFloat(count(p1)) as papers_comm, papers_published
WHERE papers_comm/papers_published>0.9
CREATE (co1)-[B:BELONGS_TO]->(c1)
MATCH (j:Journal)-[h:HAS]->(v:Volume)<-[pu:IS_PUBLISHED_IN]-(p:Paper)
WITH  j as journal, toFloat(count(p)) as papers_published
MATCH
(j1:Journal)-[h1:HAS]->(v1:Volume)<-[pu1:IS_PUBLISHED_IN]-(p1:Paper)-[
r1:RELATED_TO]->(t1:Topic)-[i1:is_in]->(c1:Community)
WHERE j1.journal_ID=journal.journal_ID WITH c1, j1, toFloat(count(p1))
as papers_comm, papers_published WHERE
papers_comm/papers_published>0.9
CREATE (j1)-[b:BELONGS_TO]->(c1) return j1,b,c1
```

- Identify the 100 top papers of these conferences/journals:

```
CALL gds.graph.create.cypher(
'pagerank_sectionD',
'MATCH(p:Paper)-[:RELATED_TO]->(t:Topic)-[:is_in]->(c:Community) WHERE
c.name = "Database community" RETURN distinct id(p) AS id',
'MATCH(c1:Community)<-[:is_in]-(t1:Topic)<-[:RELATED_TO]-(p1:Paper)-[:
CITED_BY]->(p2:Paper)-[:RELATED_TO]->(t2:Topic)-[:is_in]->(c2:Communit
y) WHERE c1.name = "Database community" AND c2.name = "Database
community" RETURN id(p1) AS source, id(p2) AS target')
CALL gds.pageRank.stream('pagerank_sectionD')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).title AS name, score
ORDER BY score DESC limit 100
```

- Gurus:

```
CALL gds.pageRank.stream('pagerank_sectionD')
YIELD nodeId, score
WITH gds.util.asNode(nodeId).paper_ID AS paperid_list, score
ORDER BY score DESC limit 100
WITH collect(paperid_list) as list
MATCH (a:Person)<-[:AUTHORED_BY]-(p:Paper)
MATCH (a:Person)<-[:AUTHORED_BY]-(p1:Paper)
WHERE (p.paper_ID IN list) AND (p1.paper_ID IN list) AND
(p.paper_ID<>p1.paper_ID)
RETURN distinct a.author as gurus_name
```