Project 1

Property Graphs

Team: Adria&Victor

Team members: Adrià Casanova Lloveras, Víctor Garcia Pizarro

Course: Semantic Data Management

Program: Data Science Master's Degree

Faculty & School: FIB, UPC - BarcelonaTech

Delivery date: 10/04/2024

A.1 Modeling

The following visual represents the graph we have designed to model the data

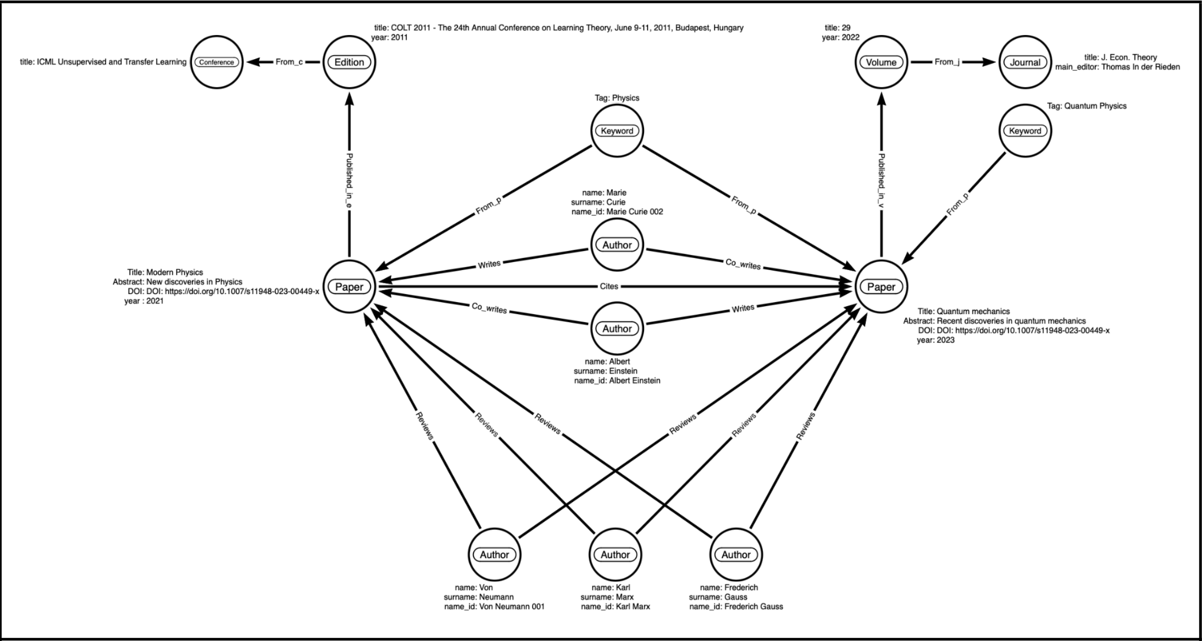

Figure 1: visual representation of the original graph model

We have decided to create the nodes *Paper, Keyword, Author, Edition, Conference, Volume, Journal*. Papers, authors and keywords have the lowest granularity in the graph, so they are the ones with the most relations. For example, an author can *Write, Co_write* or *Review* a paper, so we should not split this node in two or three different ones. A reviewer must have published papers in relevant conferences or journals, a fact which is not represented in figure 1 to make it more comprehensive. Realize that an author writes a paper if he/she is the main author. If not, then he/she co_writes the paper.

As for the publication of papers, realize we distinguish the relationship type between *Published_in_e* and *Published_in_v* depending on whether it will be to an edition or volume. The same happens with *From_c* and *_From_j*.

Maintaining the graph is easy, since there are not many kinds of papers, relationships or properties. Adding or removing a paper or author, however, can be expensive, since it causes many relationships to be added or removed as well. Nevertheless, this is the nature of the reality we are modeling, that is, so there is not much room for improvement.

The graph is reusable as well, since in this form it is simple and only models basic characteristics of research. In the following sections we will expand and modify it, showing its reusability.

There is some redundancy in the graph, but it facilitates and optimizes certain queries. The publication year of papers is both found in the paper and the edition/volume it is published in. This way, queries 3 and 4 on section B are faster. On top of that, authors have a name_id to identify them when their whole name is repeated in the graph, but to ease possible queries on this dimension, we have added the properties name and surname. The rest of the data is certainly non-redundant.

A.2 Instantiating/Loading

We have gathered most of the data from DBLP. We have converted it from XML to CSV with the program provided (dblp-to-csv) and then preprocessed it. This step was important because the original data contains millions of nodes and relationships, which would take very long to load with the Cypher command LOAD CSV. Hence, we have read the CSVs in chunks and kept only the necessary data to get results on the next sections of the project. We have made sure, among other things, to have data from 25 different years and all of the conferences and journals in the database. We have also removed duplicates and columns with unnecessary data or too many null values.

Nevertheless, DBLP did not contain all the properties or relationships we required, so we have generated artificial data. We have generated papers' keywords by applying a NLP algorithm to papers' titles and some more random keywords for those titles that did not produce any relevant result. Citations and reviews are generated completely at random, satisfying the quality rules explained in the project presentation. Abstracts are random strings of characters as well, since we did not need to use them in any query.

That being said, we have merged the original data from articles, proceedings and conferences and the artificial data to later obtain a CSV file for each collection of nodes and relationships. For nodes, each row relates its properties. On the other hand, the rows in the relationships' CSV files contain two columns. One represents the identifying property of the entering node and the other the identifying property of the exiting node.

Even though uniqueness of the identifying properties of each node was assured in the preprocessing step, we have added constraints to the graph database for future maintainability. But first, we clean the database by removing all nodes and relationships and calling the APOC procedure *apoc.schema.assert*, which deletes all constraints.

That being said, we instantiate the graph data with the following Cypher queries:

```
MATCH (n) DETACH DELETE n

CALL apoc.schema.assert({}, {})

CREATE CONSTRAINT Author_name_id IF NOT EXISTS
```

```
FOR (a:Author)
REQUIRE a.name_id IS UNIQUE

CREATE CONSTRAINT Paper_title IF NOT EXISTS
FOR (a:Paper)
REQUIRE a.title IS UNIQUE

CREATE CONSTRAINT Journal_name IF NOT EXISTS
FOR (a:Journal)
REQUIRE a.name IS UNIQUE

CREATE CONSTRAINT Volume_title IF NOT EXISTS
FOR (a:Volume)
REQUIRE a.title IS UNIQUE

CREATE CONSTRAINT Conference_title IF NOT EXISTS
FOR (a:Conference)
REQUIRE a.title IS UNIQUE

CREATE CONSTRAINT Edition_title IF NOT EXISTS
FOR (a:Edition)
REQUIRE a.title IS UNIQUE

CREATE CONSTRAINT Keyword_tag IF NOT EXISTS
FOR (a:Keyword)
REQUIRE a.tag IS UNIQUE


LOAD CSV WITH HEADERS FROM 'file:///author_node.csv' AS row
MERGE (:Author {name_id: row.author_id, name: row.name, surname: row.surname})

LOAD CSV WITH HEADERS FROM 'file:///paper_node.csv' AS row
MERGE (:Paper {title: row.title, DOI: row.DOI, year: toInteger(row.year)
                , abstract: row.abstract})

LOAD CSV WITH HEADERS FROM 'file:///journal_node.csv' AS row
MERGE (:Journal {title: row.journal, main_editor: row.editor})

LOAD CSV WITH HEADERS FROM 'file:///volume_node.csv' AS row
MERGE (:Volume {title: row.volume, year: toInteger(row.year)})

LOAD CSV WITH HEADERS FROM 'file:///conference_node.csv' AS row
MERGE (:Conference {title: row.con_shortname})

LOAD CSV WITH HEADERS FROM 'file:///edition_node.csv' AS row
MERGE (:Edition {title: row.edition_title, year: toInteger(row.edition_year)})

LOAD CSV WITH HEADERS FROM 'file:///keywords_node.csv' AS row
MERGE (:Keyword {tag: row.keyword})


LOAD CSV WITH HEADERS FROM 'file:///writes_edge.csv' AS row
```

```
MATCH (author:Author {name_id: row.main_author})
MATCH (paper:Paper {title: row.paper})
MERGE (author)-[:Writes]->(paper)

LOAD CSV WITH HEADERS FROM 'file:///co_writes_edge.csv' AS row
MATCH (author:Author {name_id: row.co_author})
MATCH (paper:Paper {title: row.paper})
MERGE (author)-[:Co_writes]->(paper)

LOAD CSV WITH HEADERS FROM 'file:///from_c_edge.csv' AS row
MATCH (source:Edition {title: row.edition})
MATCH (target:Conference {title: row.conference})
MERGE (source)-[:From_c]->(target)

LOAD CSV WITH HEADERS FROM 'file:///from_j_edge.csv' AS row
MATCH (v:Volume {title: row.volume})
MATCH (j:Journal {title: row.journal})
MERGE (v)-[:From_j]->(j)

LOAD CSV WITH HEADERS FROM 'file:///published_in_e_edge.csv' AS row
MATCH (source: Paper {title: row.paper })
MATCH (target: Edition {title: row.edition})
MERGE (source)-[:Published_in_e]->(target)

LOAD CSV WITH HEADERS FROM 'file:///published_in_v_edge.csv' AS row
MATCH (source: Paper {title: row.paper })
MATCH (target: Volume {title: row.volume})
MERGE (source)-[: Published_in_v]->(target)

LOAD CSV WITH HEADERS FROM 'file:///reviews_edge.csv' AS row
MATCH (author:Author {name_id: row.reviewer})
MATCH (paper:Paper {title: row.paper})
MERGE (author)-[:Reviews]->(paper)

LOAD CSV WITH HEADERS FROM 'file:///cites_edge.csv' AS row
MATCH (source:Paper {title: row.paper})
MATCH (target:Paper {title: row.cites})
MERGE (source)-[:Cites]->(target)

LOAD CSV WITH HEADERS FROM 'file:///keywords_node.csv' AS row
MATCH (k:Keyword {tag: row.keyword})
MATCH (p:Paper {title: row.paper})
MERGE (k)-[:From_p]->(p)
```

A.3 Evolving the graph

We propose the following modifications to the graph model. Each *Reviews* relation now has two properties: description, a short text, and decision, a binary value. The proportion of positive decisions of the reviews of a paper determines its acceptance, which is a binary property added to these nodes. Moreover, authors now have an affiliation property, whose data has been generated by randomly assigning US colleges and universities to authors. The names of this institutions have been obtained from a CSV available at https://public.opendatasoft.com/explore/dataset/us-colleges-and-universities/table/.

B. Querying

These are the queries we propose:

1. Find the top 3 most cited papers of each conference.

```
MATCH
(conference:Conference)<-[:From_c]-(:Edition)<-[:Published_in_e]-(cited_p:Paper)<-[citation:Cites]-()
WITH conference, cited_p, COUNT(citation) as n_citations
ORDER BY conference, n_citations DESC
RETURN conference.title AS CONFERENCE, COLLECT(cited_p.title)[..3] AS TOP_CITED_PAPERS
ORDER BY CONFERENCE
```

2. For each conference find its community: i.e., those authors that have published papers on that conference in, at least, 4 different editions.

```
CALL {
        MATCH
        (conference:Conference)<-[:From_c]-(edition:Edition)<-[:Published_in_e]-(:Paper)-[:Writes|C
        o_writes]-(author:Author)
        WITH conference.title AS CONFERENCE, author.name_id AS AUTHOR, COUNT(DISTINCT
        edition) AS EDITIONS
        WHERE EDITIONS >= 4
        RETURN CONFERENCE, collect(AUTHOR) AS COMMUNITY
        UNION
        MATCH (conference:Conference)
        RETURN conference.title AS CONFERENCE, null AS COMMUNITY
}
RETURN CONFERENCE, COMMUNITY
ORDER BY COALESCE(size(COMMUNITY), -1) DESC, CONFERENCE
```

3. Find the impact factors of the journals in your graph.

```
MATCH (p:Paper)-[:Published_in_v]->(:Volume)-[:From_j]->(j:Journal)
WHERE p.year IN [date().year-2, date().year-3]
WITH j.title AS JOURNAL, COUNT(p) AS PUBLICATIONS
MATCH (citing_p:Paper)-[:Cites]-(cited_p:Paper)-[:Published_in_v]->(:Volume)-[:From_j]->(j:Journal)
WHERE cited_p.year IN [date().year-2, date().year-3]
AND citing_p.year = date().year-1
WITH JOURNAL, PUBLICATIONS, COUNT(citing_p) AS CITATIONS
RETURN JOURNAL, CITATIONS *1.0 / PUBLICATIONS AS IMPACT_FACTOR
ORDER BY IMPACT_FACTOR DESC, JOURNAL
```

4. Find the h-indexes of the authors in your graph

```
MATCH (a:Author)-[:Writes|Co_writes]->(p:Paper)<-[:Cites]-(:Paper)
WITH a.name_id AS author, p.title as p_title, COUNT(*) AS c
ORDER BY author, c DESC
WITH author, collect({paper:p_title, citations:c}) AS paperCitations, COUNT(p_title) AS n_papers
UNWIND range(0, n_papers-1) AS row_number
WITH author, paperCitations[row_number]['paper'] as paper, paperCitations[row_number]['citations'] AS citations, row_number
WHERE citations >= row_number+1
RETURN author, MAX(row_number)+1 AS h_index
ORDER BY author
```

C. Recommender

We propose the following Cypher statements:

```
# Create the database community
MERGE(r: Research_community {name: 'database'})

# Relate the database keywords to the database community
MATCH(k: Keyword)
WHERE k.tag IN['data management', 'indexing', 'data modeling', 'big data', 'data processing', 'data storage', 'data querying']
MATCH(r: Research_community)
MERGE(k)-[:From_r] -> ®

# Find the database community's conferences and journals and assert the relation
 // Match conferences/journals with papers
MATCH (cj:Conference|Journal)<-[:From_c|From_j]-(:Edition|Volume)<-[:Published_in_e|Published_in_v]-(p:Paper)
WITH cj, COUNT(p) AS n_papers
// Match conferences/journals with database papers
MATCH (cj)<-[:From_c|From_j]-(:Edition|Volume)<-[:Published_in_e|Published_in_v]-(db_p:Paper)<-[:From_p]-(:Keyword)-[:From_r]->(r:Research_community {name: 'database'})
WITH r, cj, n_papers, COUNT(DISTINCT db_p) AS n_db_papers
WITH r, cj, 1.0 * n_db_papers/ n_papers AS percentage_db_papers
WHERE percentage_db_papers >= 0.9
MERGE (cj)-[:Related_to]->(r)

# Find the top-100 papers of the conferences and journals of the database community and assert the relation
MATCH (r:Research_community {name: 'database'})<-[:Related_to]-(:Conference|Journal)<-[:From_c|From_j]-(:Edition|Volume)<-[:Published_in_e|Published_in_v]-(cited_p:Paper)<-[citation:Cites]-(citing_p:Paper)-[:Published_in_e|Published_in_v]->(:Edition|Volume)-[:From_c|From_j]->(:Conference|Journal)-[:Related_to]->(r)
WITH r, cited_p, COUNT(citation) as n_citations
ORDER BY n_citations DESC
WITH r, COLLECT(cited_p)[..100] AS TOP_CITED_PAPERS
UNWIND TOP_CITED_PAPERS AS paper
```

```
MERGE (paper)-[:Is_top_paper_of]->(r)

# Find good matches to review database papers
MATCH(a: Author)-[:Writes | Co_writes] -> (: Paper)-[:Is_top_paper_of] -> (r: Research_community
{name: 'database'})
MERGE(a)-[:Good_match_to_review] -> (r)

# Find database gurus
MATCH (a:Author)-[:Writes|Co_writes]->(p:Paper)-[:Is_top_paper_of]->(r:Research_community
{name: 'database'})
WITH r, a, COUNT(p) AS n_papers
WHERE n_papers >= 2
MERGE (a)-[:Is_a_guru_of]->(r)
```

D. Graph algorithms

We have applied PageRank and Node similarity algorithms in the following way:

```
# pageRank
CALL gds.graph.project('pageRank', 'Paper', 'Cites')
CALL gds.pageRank.stream('pageRank', {nodeLabels: ['Paper'], relationshipTypes: ['Cites']})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).title AS paper, score
ORDER BY score DESC, paper
```

This algorithm returns the relevance of each paper in terms of citations. Let p be a paper. Assuming that each paper that cites p is as important as p, pageRank measures the importance of p based on the number of papers that cite it and their importance. Citations are the basis of the impact factor for journals and h-index for authors, which were measured in section B. With this algorithm, we can now compare the importance of two papers in the same context that we used for the impact factor and h-index, citations.

```
# nodeSimilarity
CALL gds.graph.project('nodeSimilarity', ['Author', 'Paper'], ['Writes', 'Co_writes'])
CALL gds.nodeSimilarity.stream('nodeSimilarity')
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).name_id AS author1, gds.util.asNode(node2).name_id AS author2,
similarity
ORDER BY similarity DESC, author1, author2
```

This algorithm measures the similarity of two authors based on the papers they have written and co-written. The way this works is the following: given the set A of papers written/co-written by an author *a* and the set B of papers written/co-written by an author *b*, the Jaccard Similarity between A and B is calculated and that is the similarity between authors *a* and *b*. That is, nodeSimilarity tells us how much two authors have collaborated between each other.