Universitat Politecnica de Catalunya

Semantic Data Management

Property Graph

Authors:
Ignacio Lloret Lorente
Marcus Karl Paulsson

Facultat d'Informàtica de Barcelona

4 April 2024

Contents

1 A.1

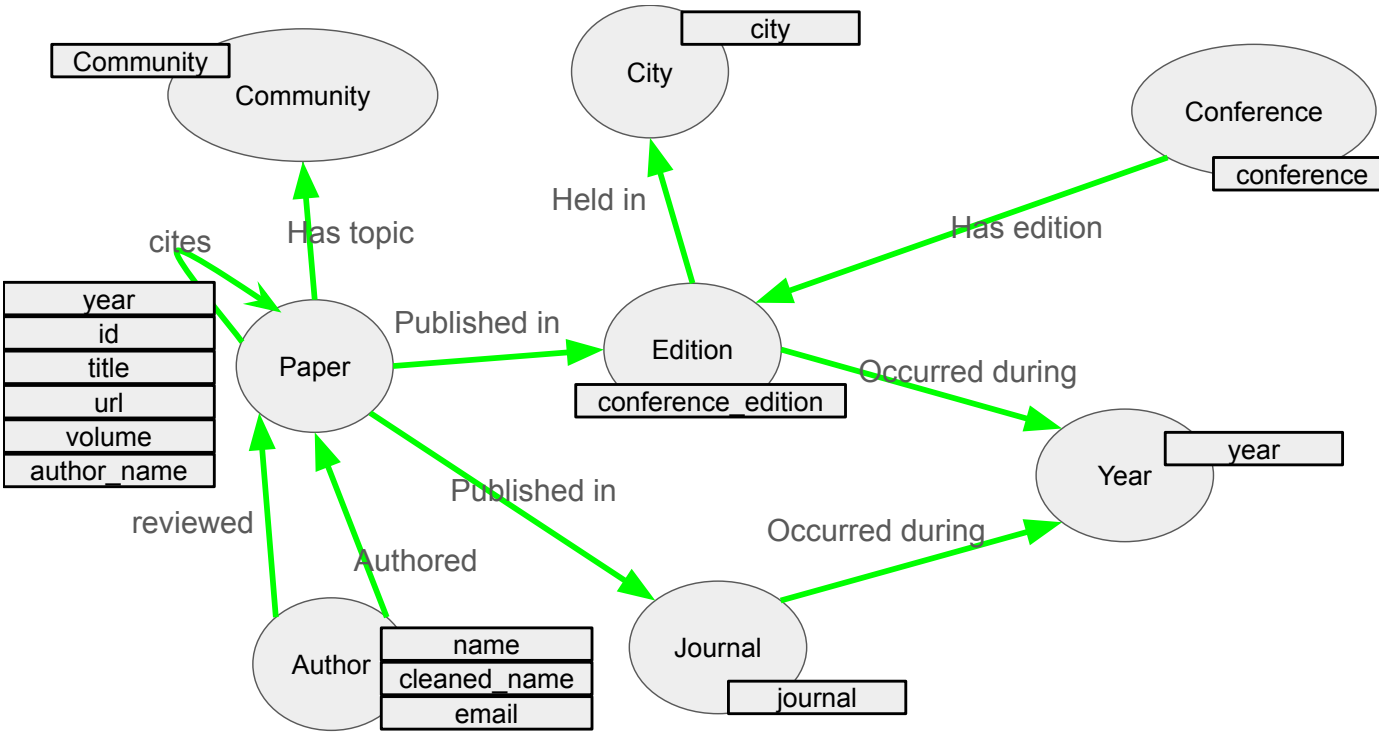

Figure 1: Visual representation of the graph

In terms of maintainability, we have chosen to keep the majority of the label data in the 'Paper' node, as it serves as the central point of the graph model and its metadata is not particularly relevant for any interesting queries. Additionally, by keeping variables like "url" unique to it, we avoid redundant and unnecessary relationship connections. As for the other nodes, they only hold one label each, primarily to facilitate easier maintenance and simpler Cypher queries.

The "Year" node was selected to be a node rather than a label, as it serves both 'Edition' and "Journal". This design choice aims to minimize redundancy and simplify the creation of queries that involve traversing the graph based on the years.

For the "Review" relationship, we opted to establish a direct connection from "Author" to "Paper". This decision was made to circumvent the creation of an additional "review" node, which would contain redundant data. Instead,

multiple edges are utilized, simplifying the process of counting or querying information. Moreover, this approach contributes to a less complex graph structure.

To ensure that this structure is conducive to executing the queries outlined in section B, the "conference" node is directly linked to "Paper" via "Edition". Additionally, since 'Paper' cites other "Papers", traversing from this node is not overly complex, facilitating the retrieval of specific data or statistics on citation counts.

Even if the "Community" node does not have any other connections than the "Paper", we decided to make it its own node as community data is very redundant. This makes the size of query data smaller if we were to extract a whole "Paper" object.

2 A.2

The data used in this graph was sourced from the DBLP database. The raw XML files were converted into CSV files using the provided script from the lab description. The extensive datasets were subsequently filtered using a Python notebook script (file: notebook/sdm project.ipynb) to extract all the labels and metadata necessary for our chosen graph structure. During the filtering process, papers and their metadata were refined to exclude specific characters that could disrupt the parsing ability of the Cypher scripts, one example being multiple quotation marks within string types.

During this preprocessing step, scripts were created to generate synthetic data, such as reviews, in order to meet the requirements outlined in the lab description. Additionally, scripts were developed to map different datasets to each other, serving as a lookup table when establishing relationships within the graph. In cases where the raw data did not align with our graph structure, these scripts were partially synthesized.

3 A.3

For this step, two changes were made to the previous graph implementation. Firstly, we updated the review relationship to include an "approve" label. The "approve" value is derived from randomized data generated during the preprocessing stage. To determine whether a paper is approved or not, we query all the approvals and check if the mean of these approvals exceeds 0.5. If it does, the paper is considered approved.

For the second change, which involved updating the affiliation, all that was required was to create a new connection to the "Author" node; no other aspects were affected by this change. We instantiated a new "affiliation" node and linked the corresponding authors to it.

4 B

```
MATCH (paper:Paper)<-[:CITES]-(citingPaper)
MATCH
    (paper)-[:PRESENTED_IN]->(edition:Edition)-[:HAS_EDITION]-(conference:Conference)
WITH conference, paper, COUNT(citingPaper) AS citationCount

WITH conference, paper.id AS paperID, citationCount
ORDER BY conference.name, citationCount DESC

WITH conference, paperID[0..3] AS TopCitedPapers, citationCount as
    citations
RETURN conference.name, TopCitedPapers, citations
ORDER BY conference.name, citations DESC
```

Top 3 most cited papers of each conference

```
MATCH (author:Author)-[:AUTHORED]->(paper:Paper)-[:PRESENTED_IN]-> \\
   (edition:Edition)<-[:HAS_EDITION]-(conference:Conference)
WITH conference, author, COUNT(DISTINCT edition) AS editionsCount
WHERE editionsCount >= 4
RETURN conference.name AS ConferenceName, author.name, editionsCount
```

For each conference find its community

```
MATCH (j:Journal)<-[:PUBLISHED_IN]-(a:Paper)
WHERE a.year IN [toString(toInteger(date().year) - 2),
    toString(toInteger(date().year) - 1)]
WITH j, count(a) AS articles_count
MATCH (j)<-[:PUBLISHED_IN]-(a:Paper)<-[:CITES]-(c:Paper)
WHERE a.year IN [toString(toInteger(date().year) - 2),
    toString(toInteger(date().year) - 1)]
WITH j, articles_count, count(c) AS citations_count
WHERE articles_count > 0
RETURN j.name AS journal, toFloat(citations_count) / articles_count AS
    impact_factor
ORDER BY impact_factor DESC
LIMIT 10
```

Impact factor

```
MATCH (a:Author)-[:AUTHORED]->(p:Paper)<-[:CITES]-(c:Paper)
WITH a, p, count(c) AS citations
ORDER BY a, citations DESC
WITH a, collect(citations) AS citation_counts
WITH a, [c IN citation_counts WHERE c >= size(range(1, c))] AS
    h_candidates
RETURN a.name AS author, CASE WHEN size(h_candidates) > 0 THEN
    size(h_candidates) ELSE 0 END AS h_index
ORDER BY h_index ASC
LIMIT 10
```

H-INDEX

5 C

```
\\ 1st part
LOAD CSV WITH HEADERS FROM 'file:///communities_data.csv' AS row
CREATE (co:Communities {
   article_id: row.id,
   category: row.category
});

MATCH (p:Paper), (co:Communities)
WHERE p.id = co.article_id
CREATE (p)-[:TOPIC]->(co);
```

```

\\ 2nd part
MATCH (co:Communities)<-[:TOPIC]-(p:Paper)-[:PUBLISHED_IN]->(j:Journal)
WITH j, COUNT(DISTINCT p) AS papersCount
MATCH (co:Communities)<-[:TOPIC]-(p:Paper)-[:PUBLISHED_IN]->(j:Journal)
where co.category = 'database'
WITH j, COUNT(p) as conteo, papersCount
WITH j, (conteo*1.0 / papersCount) as prc
where prc > .9

\\ 3rd part
MATCH (jo:Journal)<-[:PUBLISHED_IN]-(p:Paper)<-[:CITES]-(citingPaper)
MATCH (co:Communities)<-[:TOPIC]-(p:Paper)
where jo.name = j.name AND co.category = 'database'
WITH j.name as journal_name, p, COUNT(citingPaper) AS citationCount
ORDER BY journal_name, citationCount DESC

```

```
// Collect the top 10 papers for each journal
WITH journal_name, citationCount, COLLECT(p)[0..10] AS topPapers

// 4rd part
UNWIND topPapers AS topPaper
MATCH (a:Author)-[:AUTHORED]->(topPaper)
WITH journal_name, topPaper.title AS paper_title, topPaper.id AS
    paper_id, citationCount, COLLECT(a.name) AS author_names

// Collect the results for each journal
RETURN journal_name, citationCount, COLLECT({paper_id: paper_id,
    paper_title: paper_title, authors: author_names}) AS
    top_cited_papers
"""
```

For each journal find the top 10 most cited papaers and its authors

6 D

Node similarity using Jaccard Similarity function:

```
MATCH (n1:Paper {id: '729801'}), (n2:Paper {id: '735026'})
WITH n1, n2,
    [key IN keys(n1) WHERE key IN keys(n2) AND n1[key] = n2[key]] AS
        matching_properties,
    keys(n1) + [key IN keys(n2) WHERE NOT key IN keys(n1)] AS
        all_properties
RETURN n1.id AS node1_id, n2.id AS node2_id,
      matching_properties,
      toFloat(size(matching_properties)) / size(all_properties) AS
          similarity_score
```

Shortest path length using cites as path:

```
MATCH (start:Paper {id: '729351'}), (end:Paper {id: '729801'})
MATCH path = shortestPath((start)-[:CITES*]-(end))
RETURN path, length(path) AS pathLength
```