Universitat Politecnica de Catalunya

Semantic Data Management

LAB Assignment[1]

Practice 1 - Property Graphs

Team AlbuquerqueFernandez	Under the guidance of
Filipe Albuquerque Ito Russo **Lucia Victoria Fernandez Sanchez**	Prof. Anna Queralt

2025

[1] GitHub Repository

Contents

Chapter 1

Introduction

1.1 Introduction to Graph-Based Data Management

Modern graph databases present a paradigm shift from conventional relational systems by natively representing interconnected data [1]. These systems organize information as [2]:

- **Nodes**: Entities with typed labels and customizable attributes
- **Edges**: Directed associations between entities carrying their own metadata
- **Properties**: Key-value pairs attached to both nodes and relationships

This structure proves particularly effective for domain models with rich interdependencies like biological networks, supply chains, and digital recommendation engines [**?**].

1.1.1 Neo4j's Advantages

As the graph platform [3] delivers:

- **Cypher**: An intuitive query language, similar to SQL
- **Storage optimization**: Native graph processing engine avoiding join operations

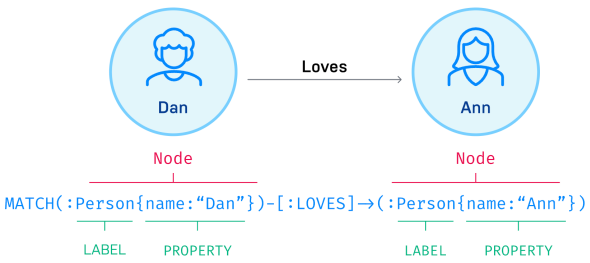

Figure 1.1: A visual way of matching patterns and relationships using Cypher [4].

Chapter 2

A Modeling, Loading, Evolving

Our project manages the Graph database in the context of using Academic paper with different properties like: type, keywords, authors, edition, etc. In the picture 2.1, we can see the schema. In total, we have 7 nodes with 9 relationships.

2.1 Schema - Modeling considerations

Our graph model follows a simple efficient design for representing academic publications. The main element is the Paper node, which contains key attributes such as title, DOI, and publication type (Journal, Conference, or Workshop). Each paper maintains a reference to its main author while supporting multiple contributors through relationships.

Authors are connected to papers via WRITTEN_BY relationship, which include a position attribute to keep the authorship order. The primary author always holds the first position. The publication type determines additional elements in the graph.

For Journal papers, the model creates two supporting nodes: a Journal node containing publication details and a Volume node storing edition-specific metadata. On the other hand, Conference and Workshop papers, a unified Conference node captures event details, including name, edition, venue (city), and year. This approach efficiently organizes academic data while maintaining clarity and flexibility.

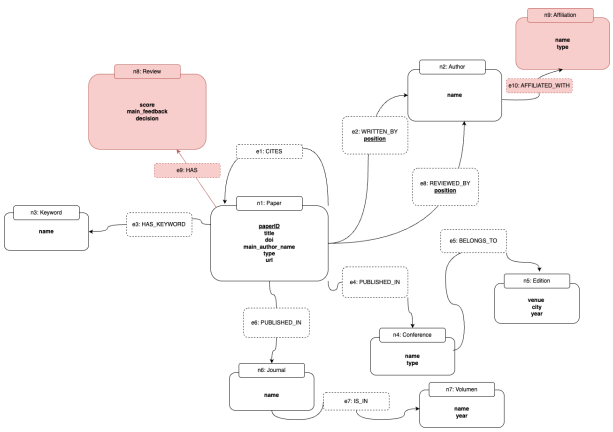

Figure 2.1: Schema with the nodes and relationships

2.2 Instantiating

The project uses two different sources of Data: Synthetic data created with the following characteristics:

Name	Description	Declaration Name
`Keywords`	A curated list of 20 terms related to Computer Science, including those explicitly mentioned in the paper.	`keyword_list`
`Document`	Based on the API response for each paper, we enrich the data with details from real-world European academic events.	`conferences journals workshops`
`Affiliations`	This data, used in Part A.3 of the exercise, simulates a complete API response by including evolving information.	`affiliations`
`Reviews`	The dataset exclusively includes **accepted** papers, each with a review score of 6 or higher.	`paper_reviews`

Table 2.1: Summary of the synthetic data

We employ the DBLP Computer Science Bibliography API[1] for our data collection, selected for its user-friendly and easy implementation.

The API fetches multiple attributes from various publications, with one defining trait: All papers are exclusively within **the Computer Science** domain.

2.3 Loading

For the loading part, we have a script that contains all the nodes and relationships to load onto the neo4j Desktop.

For proper configuration, we explain the details in detail in the repository page[2].

Our data model follows strict naming conventions to maintain consistency:

- **Node Labels**: PascalCase (e.g., `Paper`, `Author`, `Journal`)
- **Relationships**: SCREAMING_SNAKE_CASE (e.g., `CITES`, `HAS_KEYWORD`)
- **Attributes**: lowercase with underscores (e.g., `title`, `title`, `doi`)

2.4 Evolving the graph

In this phase, we modified our graph database with two key nodes. First, we introduced the `Affiliation` node containing two attributes: `name` (string) and `type` (restricted to either `university` or `company`). Each `Author` node maintains a strict one-to-one relationship with an affiliation through the `AFFILIATED_WITH` relationship.

Second, we implemented the `Review` node with three attributes: a numerical `score`, textual `main_feedback`, and a fixed `decision` value (always set to `accepted`). These review nodes connect to `Paper` nodes via `HAS_REVIEW` relationships, completing our academic publication modeling framework.In the Figure 2.1 the relationships and nodes in color red are the evolved information.

[1] Source
[2] Repository

Chapter 3

Querying

3.1 Approach

Our primary objective in the querying section is to maximize efficiency, ensuring optimal processing speed when handling datasets of 1,000 papers or more. Each query is carefully designed to minimize computational time while delivering accurate results.

3.2 Top-Cited Papers

This query specifically targets conference and workshop papers, excluding journals through a conditional filter. It calculates citation counts between papers and sorts the output in descending order based on citation volume, highlighting the most influential works in these categories.

3.3 Conference Communities

To identify established conference communities, we implement an edition threshold requiring participation in at least four different editions (as specified in the project guidelines). This ensures we only analyze conferences with sustained academic activity over time.

3.4 Journal Impact Factor

For journal evaluation, we match journals with their respective papers while having quality:

- Excluding journals with no published papers
- Calculating precise impact factors using the `ToFloat` function to handle division operations

3.5 H-Index

The h-index query employs the following approach:

1. Mapping citation networks between papers
2. Associating publications with their authors
3. Applying the formal h-index calculation (N papers with $\geq N$ citations each)

Chapter 4

Recommender

4.1 Overview

We developed a foundational recommender system with the following components:

4.1.1 Community Definition

The system identifies relevant papers by querying for specific technical keywords. Papers containing any of these keywords are included in the target community.

4.1.2 Venue Identification

To code publications by venue type, we implemented an UNION logical statment:

- Combines results from both Journal and Conference/workshop papers
- Resolves venue names despite the Paper nodes only containing a generic type attribute

4.1.3 Community Database (node)

We created a graph structure by:

- Creating a ”Community” node to represent the database domain
- Linking this node to relevant events and their most influential papers

4.1.4 Gurus Identification

We sort out the ”Gurus” by:

- Extending the Author node with expert attributes
- At least two qualifying papers

Throughout implementation, we kept code reusability and logical consistency across all queries to ensure system coherence and performance.

Chapter 5

Graph algorithms

5.1 Algorithm Implementation

For this part of the practice, we decided to implement two algorithms: **Betweenness** and **Node Similarity**.

5.1.1 Betweenness

The algorithm identifies influential nodes by measuring how often they appear on the shortest paths between other nodes in the network. In this implementation, it analyzes the citation graph (with reversed ‘CITES‘ relationships) to find papers that act as bridges between different research areas.

5.1.2 Node Similarity

The algorithm compares the neighborhood of each Paper node (its connected ‘Keyword‘ nodes) with every other paper’s keywords, identifying pairs with overlapping terminology.

We chose to use both algorithms because we have prior experience with their implementation in a different context. Additionally, this exercise served as a valuable refresher for our final project implementation.

Bibliography

[1] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Computing Surveys*, vol. 40, no. 1, pp. 1–39, 2008.

[2] Neo4j, Inc., "Graph database fundamentals," https://neo4j.com/developer/graph-database/, 2023, accessed: 2025-04-04.

[3] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*, 2nd ed. O'Reilly Media, 2015.

[4] Neo4j, Inc., "Graph database structure diagram," https://neo4j.com/docs/getting-started/graph-database/, 2023, screenshot of conceptual diagram; accessed 2025-04-04.