CS343: Graph Data Science
Homework 01

Ali Muhammad Asad

3.1 Bibliographic Data Analysis

Model

We can design a simple graph property model based on the provided data as shown:

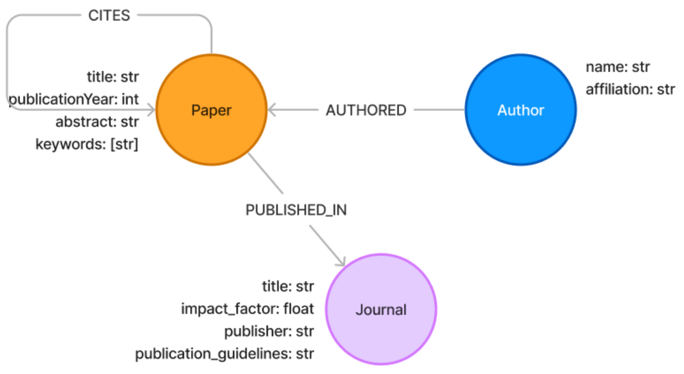

Figure 1: Graph Property Model for Bibliographic Data

The above model can also be represented in the following way:

```
(:Author)-[:AUTHORED]->(:Paper)
(:Paper)-[:PUBLISHED_IN]->(:Journal)
(:Paper)-[:CITES]->(:Paper)
```

In the above graph property model, we have the following nodes and relationships: **Nodes:**

- *Paper* which represents a research paper and has attributes *title*, *publicationYear*, *abstract*, and *keywords*
- *Author* which represents an author and has attributes *name*, *affiliation*
- *Journal* which represents a journal and has attributes *name*, *impactFactor*, *publisher*, *publicaiton_guidelines*

We have the following relationships: **Relationships:**

- *AUTHORED* which is an edge between a paper node, and its author. An author 'authored' a paper.
- *PUBLISHED_IN* which is an edge between a paper node and a journal node. A paper is published in a journal.
- *CITES* which is an edge between two paper nodes. A paper cites another paper.

Rationale

Our rationale behind such a data model stems from the above mentioned nodes and relationships, since this model allows us to easily make connections between various papers, their authors, and journals they were published in. It also allows for efficient search querying and analysis of the data based on the questions provided. An author can have various papers, and a paper can also have multiple authors. Similarly, a paper can be published in a journal, and a journal can have multiple papers. A paper can also cite multiple papers, and can be cited by multiple papers. This model allows for a flexible way to model the relationships between the various entities in the data.

Queries

1. Identify top 10 authors who have collaborated on multiple papers based on the strength of their collaboration based on the number of joint publications.

```
match (a1:Author)-[:AUTHORED]->(p:Paper)<-[:AUTHORED]-(a2:Author)
with a1, a2, count(distinct p) as NumberofJointPublications
where a1 <> a2
return a1.name, a2.name, NumberofJointPublications
order by NumberofJointPublications desc limit 10
```

2. Identify the top 10 most cited papers in the database and the number of citations they have received.

```
match (p:Paper)<-[:CITES]-()
return p.title, count(*) as NumberOfCitations
order by NumberOfCitations desc limit 10
```

3. Identify the top 10 most influential authors based on the number of citations their papers have received.

```
match (a:Author)-[:AUTHORED]->(p:Paper)<-[:CITES]-()
return a.name, count(*) as NumberOfCitations
order by NumberOfCitations desc limit 10
```

4. Identify the top 10 authors with the most publications in the database.

```
match (a:Author)-[:AUTHORED]->(p:Paper)
return a.name, count(distinct p) as NumberOfPublications
order by NumberOfPublications desc limit 10
```

5. Identify the top 10 pairs of papers that are most frequently co-cited together.

```
match (p1:Paper)<-[:CITES]-(p:Paper)-[:CITES]->(p2:Paper)
with p1, p2, count(distinct p) as NumberOfCoCitations
where p1 <> p2
```

```
return p1.title, p2.title, NumberOfCoCitations
order by NumberOfCoCitations desc limit 10
```

6. Calculate the diameter of the citation network, which represents the longest shortest path between any two papers in the network.

```
match (p1:Paper), (p2:Paper) where id(p1) < id(p2)
match p = shortestPath((p1)-[:CITES*]-(p2))
return length(p) as Diameter
order by Diameter desc limit 1
```

7. Calculate the degree distribution of the citation network, which represents the number of citations each paper has received.

```
match (p:Paper)<-[:CITES]-()
return p.title, count(*) as Degree
order by Degree desc
```

8. Calculate the degree distribution of the co-authorship network, which represents the number of co-authors each author has collaborated with.

```
match (a1:Author)-[:AUTHORED]->(p:Paper)<-[:AUTHORED]-(a2:Author)
with a1, count(distinct a2) as Degree
where a1 <> a2
return a1.name, Degree
order by Degree desc
```

9. Identify the top 10 pairs of keywords that most frequently co-occur in the same paper.

```
match (p:Paper)
unwind p.keywords as k1
unwind p.keywords as k2
with k1, k2, count(distinct p) as NumberOfCoOccurences
where k1 < k2
return k1, k2, NumberOfCoOccurences
order by NumberOfCoOccurences desc limit 10
```

10. Calculate the average number of citations per year for top 10 cited paper in the database.

```
match (p:Paper)<-[:CITES]-()
with p, count(*) as NumberOfCitations
return p.title, NumberOfCitations, NumberOfCitations / (2024 - p.publicationYear)
    as AverageCitationsPerYear
order by NumberOfCitations desc limit 10
```

3.2 University Course Registration System

Model

We can design a graph property model as follows:

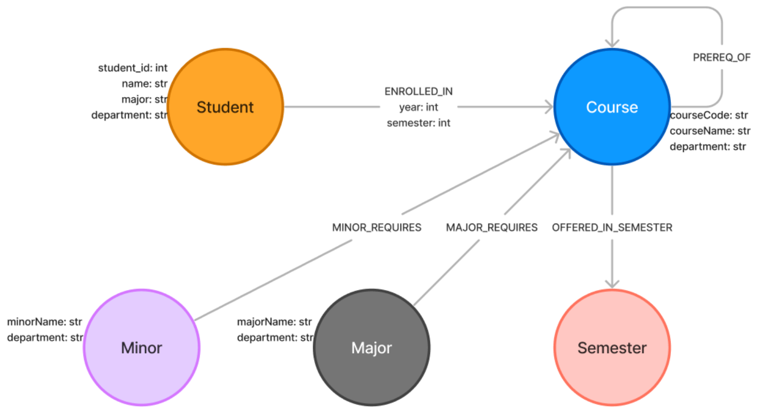

Figure 2: Graph Property Model for University Course Registration System

The above model can also be represented in the following way:

```
(:Student)-[:ENROLLED_IN]->(:Course)
(:Course)-[:OFFERED_IN]->(:Semester)
(:Course)-[:PREREQ_OF]->(:Course)
(:Major)-[:MAJOR_REQUIRES]->(:Course)
(:Minor)-[:MINOR_REQUIRES]->(:Course)
```

In the above graph property model, we have the following nodes and relationships: **Nodes:**

- *Student* which represents a student and has attributes *student_id*, *name*, *major*, *department*
- *Course* which represents a course and has attributes *courseCode*, *courseName*, *department*
- *Semester* which represents a semester that a course is offered in
- *Major* which represents a major and has attributes *majorName*, *department*
- *Minr* which represents a minor and has attributes *minorName*, *department*

We have the following relationships: **Relationships:**

- *ENROLLED_IN* which is an edge between a student node, and a course node. A student is enrolled in a course.
- *OFFERED_IN* which is an edge between a course node and a semester node. A course is offered in a semester.
- *PREREQ_OF* which is an edge between two course nodes. A course is a prerequisite of another course.

- *MAJOR_REQUIRES* which is an edge between a major node and a course node. A major requires a course.
- *MINOR_REQUIRES* which is an edge between a minor node and a course node. A minor requires a course.

Rationale

Our rationale behind the model is that it allows us to easily make connections between various students, courses, and majors/minors. It also allows for efficient search querying and analysis of the data based on the questions provided. A student can be enrolled in multiple courses, and a course can also have multiple students. Similarly, a course can be offered in multiple semesters, and a semester can have multiple courses. A course can also have multiple prerequisites, and can be a prerequisite for multiple courses. A major can require multiple courses, and a course can be required by multiple majors. A minor can require multiple courses, and a course can be required by multiple minors. This model allows for a flexible way to model the relationships between the various entities in the data.

Queries

1. Identify the prerequisites for a course with course code 'CS101'.

```
match (c1:Course {courseCode: "CS101"})<-[:PREREQ_OF]-(c2:Course)
return c2.courseCode as PrerequisiteCourseCode, c2.courseName as
    PrerequisiteCourseName
```

2. List number of courses must be taken to fullfill requirements for each major.

```
match (m:Major)-[:MAJOR_REQUIRES]->(c:Course)
return m.majorName as MajorName, count(distinct c) as NumberofCoursesRequired
```

3. List number of courses must be taken to fullfill requirements for each minor.

```
match (m:Minor)-[:MINOR_REQUIRES]->(c:Course)
return m.minorName as MinorName, count(distinct c) as NumberofCoursesRequired
```

4. List number of courses must be taken before enrolling the course with course code 'CS431'.

```
match path = (c1:Course {courseCode: "CS431"})<-[:PREREQ_OF*]-(c2:Course)
return length(path) as NumberOfCoursesRequired
```

5. Identify the courses that are taken together by at least 10 students. Return the courses and the number of students enrolled in each course.

```
match (s:Student)-[:ENROLLED_IN]->(c:Course)
with c, count(s) as NumberOfStudents
where NumberOfStudents >= 10
return c.courseCode as CourseCode, c.courseName as CourseName, NumberOfStudents
```

6. Identify the courses that are offered in the same semester, indicating potential course scheduling patterns. Return the semester and the courses offered in that semester.

```
match (c:Course)-[:OFFERED_IN]->(s:Semester)
return s as Semester, collect(c.courseCode) as CoursesOffered
```

7. Identify the top 10 most popular courses based on the number of students enrolled.

```
match (s:Student)-[:ENROLLED_IN]->(c:Course)
with c, count(s) as NumberOfStudents
return c.courseCode as CourseCode, NumberOfStudents
order by NumberOfStudents desc limit 10
```

8. Identify the courses which are taken together by students of Computer Science and Electrical Engineering major.

```
match (s:Student {major: "Computer Science"})-[:ENROLLED_IN]->(c:Course)<-[:
    ENROLLED_IN]-(s2:Student {major: "Electrical Engineering"})
return c.courseCode as CourseCode, c.courseName as CourseName
```

9. Identify the sequence of courses that a student with id “1214” has taken, indicating the order in which they have completed the courses.

```
match (s:Student {student_id: "1214"})-[e:ENROLLED_IN]->(c:Course)
return c.courseCode as CourseCode, e.year as Year, e.semester as Semester
order by e.year, e.semester
```

10. List courses with the number of prerequisites assigned, indicating the complexity of the course requirements.

```
match (c1:Course)<-[:PREREQ_OF]-(c2:Course)
return c1.courseCode as CourseCode, count(c2) as NumberOfPrerequisites
```

3.3 KSE 100 Companies

Model

We can design a graph property model as follows:

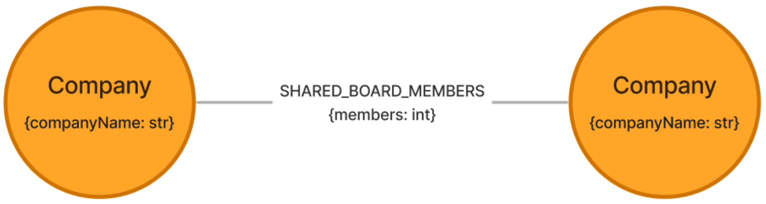

Figure 3: Graph Property Model for KSE 100 Companies

The above model can also be represented in the following way:

```
(:Company {companyName:})-[:SHARED_BOARD_MEMBERS {members:}]->(:Company {
    companyName:})
```

Rationale

For the design of the graph property model, our rationale is as follows:

- **Company** is the node of our graph which represents a company, and has an attribute called *companyName* representing the name of the company
- We have an edge titled *SHARED_BOARD_MEMBERS* which is an edge connecting two comppanies if they have some shared board members.
- The edge has an attribute *members* which is an integer and tells the number of shared members between two company nodes.
- The edge is not specified any direction, as the relationship is symmetric, i.e., if company A has a shared board member with company B, then company B also has a shared board member with company A. Hence, its undirected since it goes both ways.

The above design gives a flexible way to model the relationship as mentioned in the question, also, it allows us to easily add more companies and relationships to the graph model, or allow us to easily add more attributes to the relationships. It also makes it easy to query the graph for answering questions etc.

Queries

1. Provide import statement to load the data into the graph database.

```
load csv with headers from "file:///kse.csv" as row
merge (c1:Company {companyName: row.Company1})
merge (c2:Company {companyName: row.Company2})
merge (c1)-[s:SHARED_BOARD_MEMBERS]->(c2)
set s.members = toInteger(row.weight)
```

2. Determine the number of companies in the dataset

```
match (c:Company)
return count(distinct c.companyName) as ‘Number of Companies‘
```

By the above query, there are 1368 companies in the dataset.

3. Determine the number of edges in the dataset

```
match ()-[s:SHARED_BOARD_MEMBERS]->()
return count(s) as ‘Number of Edges‘
```

By the above query, there are 2658 edges in the dataset.

4. Identify the companies that share the highest number of board members.

```
match (c1:Company)-[s:SHARED_BOARD_MEMBERS]->(c2:Company)
return c1.companyName, c2.companyName, s.members as ‘Shared Board Members‘
order by ‘Shared Board Members‘ desc
limit 1
```

Our companies are: “Maple Leaf Cement Factory Limited”, and “Kohinoor Textile Mills Limited” with 8 shared board members.

5. Determine the average number of shared board member between all pairs of companies.

```
match (c1:Company)-[s:SHARED_BOARD_MEMBERS]->(c2:Company)
return avg(s.members) as ‘Average Shared Board Members‘
```

By the above query, there are 1.431 shared board members on average between all pairs of companies.

6. Find the length of the maximum path between any two companies.

```
match (c1:Company), (c2:Company)
where id(c1) < id(c2)
match path = shortestPath((c1)-[:SHARED_BOARD_MEMBERS]->(c2))
return length(path) as ‘Maximum Path Length‘
order by ‘Maximum Path Length‘ desc
limit 1
```

By the above query, the maximum path length between any two companies is 1. We compare the ids of the two companies to ensure that we don't get the same company on both nodes.

7. Determine companies that have the higest number of board member associations with other companies along with the total number of board members they have.

```
match (c:Company)-[s:SHARED_BOARD_MEMBERS]->()
return c.companyName, count(s) as ‘Number Of Associations’, sum(s.members) as ‘
    Total Board Members’
order by ‘Number Of Associations’ desc, ‘Total Board Members’ desc
limit 1
```

By the above query, the company with the highest number of associations and board members is "HUB Power Company LTD" with 97 associations and 130 board members.

8. A triangle can be formed if a company shared board members with two other companies such that the two companies also share board members with each other. Determine the number of companies that form a triangle.

```
match (c1:Company)-[:SHARED_BOARD_MEMBERS]->(c2:Company)-[:SHARED_BOARD_MEMBERS
    ]->(c3:Company)-[:SHARED_BOARD_MEMBERS]->(c1:Company)
return count(distinct c1) as NumberOfTriangles
```

There are no companies in our dataset that form a triangle.

9. As the CEO of Nestle Milkpak LTD you want to connect with someone who is connected to Shell Pakistan LTD. Find the number of people you need to connect to reach Shell Pakistan LTD.

```
match path = shortestpath((c1:Company {companyName: "Nestle Milkpak Ltd."})
    -[*..15]-(c2:Company {companyName: "Shell Pakistan Ltd"}))
return length(path) as ‘Number of Connection‘
```

By the above query, there are 3 connections needed to reach Shell Pakistan LTD from Nestle Milkpak LTD.