Graph Database – Hands on Assignment

1) <u>Library Management System</u>

<u>Question</u>: Create a graph for a library where:

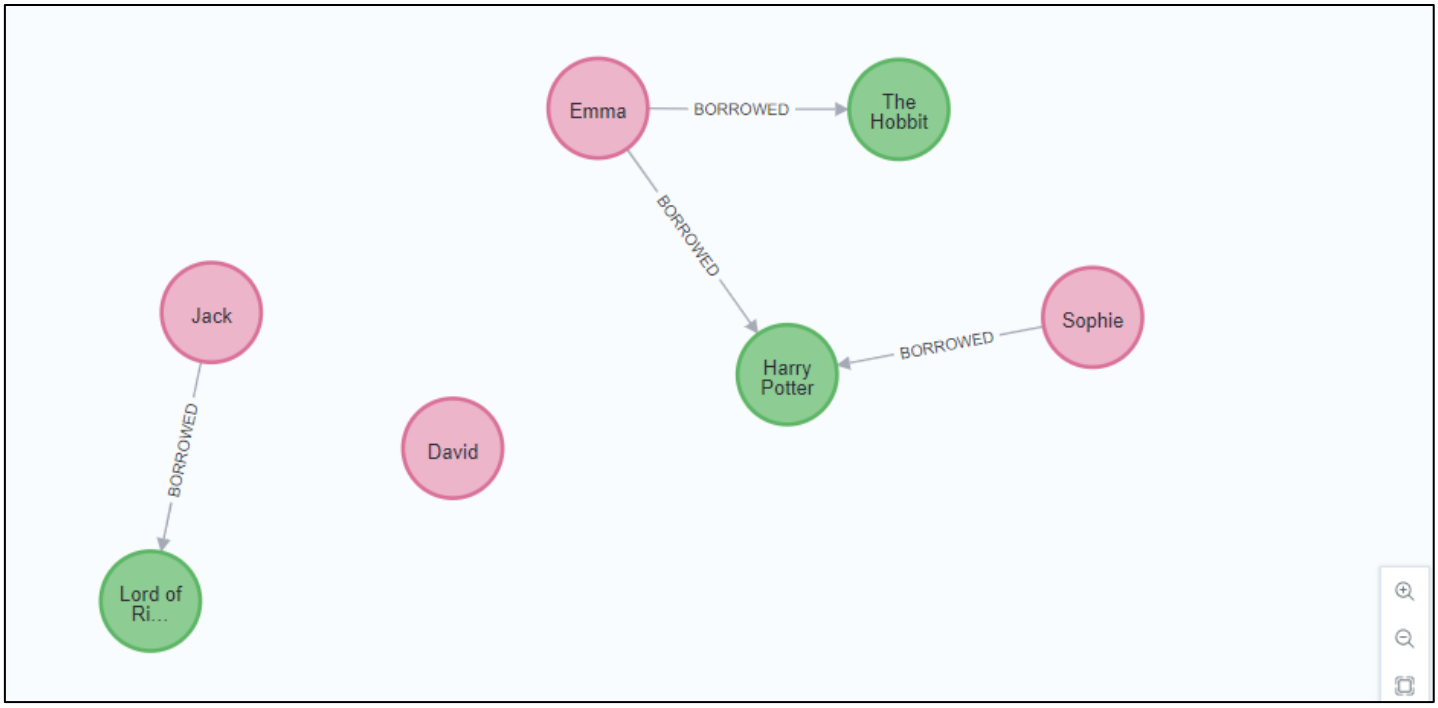
- There are three books: 'Harry Potter', 'Lord of the Rings', and 'The Hobbit'
- There are four members: Emma, Jack, Sophie, and David
- Emma has borrowed 'Harry Potter' and 'The Hobbit'
- Jack has borrowed 'Lord of the Rings'
- Sophie has borrowed 'Harry Potter'
- David hasn't borrowed any books
- Include book genres, member joining dates, and borrowing dates

<u>Solution</u>:

```
// Create Books
CREATE (b1:Book { title: 'Harry Potter', genre: 'Fantasy', publishYear: 1997})
CREATE (b2:Book {title: 'Lord of the Rings', genre: 'Fantasy', publishYear: 1954})
CREATE (b3:Book {title: 'The Hobbit',genre: 'Fantasy',publishYear: 1937})

// Create Members
CREATE (m1:Member {name: 'Emma', joinDate: '2023-01-15'})
CREATE (m2:Member {name: 'Jack', joinDate: '2023-02-20'})
CREATE (m3:Member {name: 'Sophie', joinDate: '2023-03-10'})
CREATE (m4:Member {name: 'David', joinDate: '2023-04-05'})

// Create Borrowing Relationships
MATCH
   (m1:Member{name: 'Emma'}),
   (b1:Book { title: 'Harry Potter'})
CREATE (m1)-[:BORROWED {date: '2023-05-01'}]->(b1)
```

```
MATCH
   (m1:Member{name: 'Emma'}),
   (b3:Book { title: 'The Hobbit'})
CREATE (m1)-[:BORROWED {date: '2023-06-15'}]->(b3)

MATCH
   (m2:Member{name: 'Jack'}),
   (b2:Book { title: 'Lord of the Rings'})
CREATE (m2)-[:BORROWED {date: '2023-05-10'}]->(b2)

MATCH
   (m3:Member{name: 'Sophie'}),
   (b1:Book { title: 'Harry Potter'})
CREATE (m3)-[:BORROWED {date: '2023-07-01'}]->(b1)
```

<u>Output</u>:



Q1) Find all books borrowed by Emma.
-> MATCH (m:Member {name: 'Emma'})-[r:BORROWED]->(b:Book)
RETURN b.title, r.date

Output:

	b.title	r.date
1	"Harry Potter"	"2023-05-01"
2	"The Hobbit"	"2023-05-01"

Q2) Find members who haven't borrowed any books.

```
-> MATCH (m:Member)
WHERE NOT (m)-[:BORROWED]->()
RETURN m.name
```

<u>Output</u>:

	m.name
1	"David"

Q3) Find books that have been borrowed more than once.

```
-> MATCH (m:Member)-[:BORROWED]->(b:Book)
WITH b, COUNT(*) as borrowCount
WHERE borrowCount > 1
RETURN b.title, borrowCount
```

<u>Output</u>:

	b.title	borrowCount
1	"Harry Potter"	2

Q4) Find the most recent borrower for each book.

```
-> MATCH (m:Member)-[r:BORROWED]->(b:Book)
WITH b, m, r
ORDER BY r.date DESC
RETURN b.title, m.name, r.date
```

Output:

	b.title	m.name	r.date
1	"Harry Potter"	"Sophie"	"2023-07-01"
2	"Lord of the Rings"	"Jack"	"2023-05-10"
3	"Harry Potter"	"Emma"	"2023-05-01"
4	"The Hobbit"	"Emma"	"2023-05-01"

2) <u>Social Media Network</u>

<u>Question</u>: Create a social media network where:

- There are five users: Alex, Maya, Ryan, Priya, and Sam
- Alex posts two photos: 'Vacation' and 'Party'
- Maya posts one photo: 'Graduation'
- Ryan comments on Alex's 'Party' photo
- Priya likes both of Alex's photos
- Sam follows everyone but hasn't posted anything
- Include post dates, comment text, and user join dates

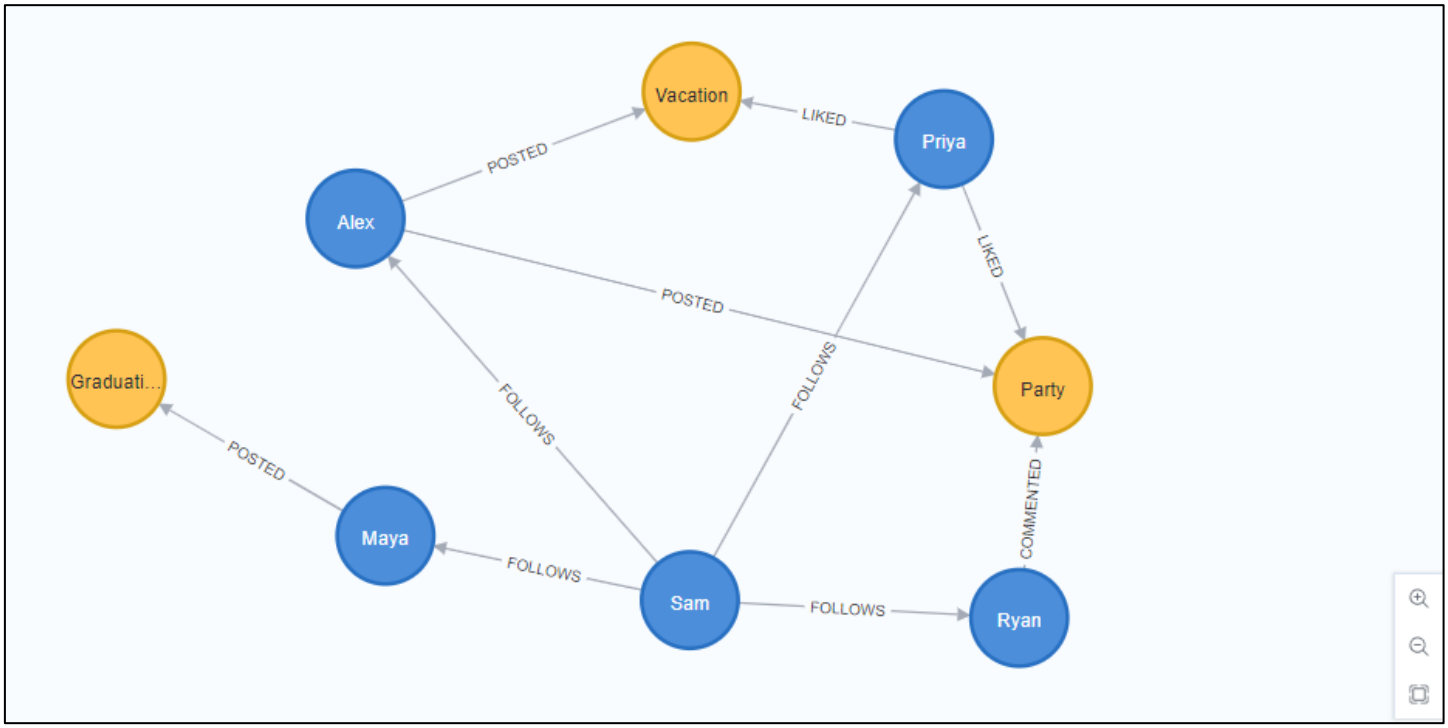
<u>Solution</u>:

```
// Create Users
CREATE (u1:User { name: 'Alex', joinDate: '2023-01-10', email: 'alex@email.com'})
CREATE (u2:User { name: 'Maya', joinDate: '2023-02-
15', email: 'maya@email.com'})
CREATE (u3:User {name: 'Ryan', joinDate: '2023-03-20', email: 'ryan@email.com'})
CREATE (u4:User {name: 'Priya', joinDate: '2023-02-
01', email: 'priya@email.com'})
CREATE (u5:User {name: 'Sam', joinDate: '2023-04-05', email: 'sam@email.com'})

// Create Posts
CREATE (p1:Post { title: 'Vacation', date: '2023-05-15', type: 'photo'})
CREATE (p2:Post { title: 'Party', date: '2023-06-20', type: 'photo'})
CREATE (p3:Post { title: 'Graduation', date: '2023-07-01', type: 'photo'})


// Create Relationships
MATCH
   (u1:User{name:'Alex'}),
  (p1:Post{title:'Vacation'})
CREATE (u1)-[:POSTED]->(p1)
```

```
MATCH
   (u1:User{name:'Alex'}),
   (p2:Post{ title:'Party'})
CREATE (u1)-[:POSTED]->(p2)

MATCH
   (u2:User{name:'Maya'}),
   (p3:Post{ title:'Graduation'})
CREATE (u2)-[:POSTED]->(p3)


// Comments
MATCH

   (u3:User{name:'Ryan'}),
   (p2:Post{title:'Party'})
CREATE (u3)-[:COMMENTED {
text: 'Great party!',
date: '2023-06-21'
}]->(p2)

// Likes
MATCH
   (u4:User{name:'Priya'}),
   (p1:Post{title:'Vacation'})
CREATE (u4)-[:LIKED {date: '2023-05-16'}]->(p1)

MATCH
   (u4:User{name:'Priya'}),
   (p2:Post{title:'Party'})
CREATE (u4)-[:LIKED {date: '2023-06-22'}]->(p2)
```

```
// Sam follows everyone
MATCH
   (u5:User{name:'Sam'}),
   (u1:User{name:'Alex'}),
   (u2:User{name:'Maya'}),
   (u3:User{name:'Ryan'}),
   (u4:User{name:'Priya'})
CREATE (u5)-[:FOLLOWS {since: '2023-04-10'}]->(u1)
CREATE (u5)-[:FOLLOWS {since: '2023-04-10'}]->(u2)
CREATE (u5)-[:FOLLOWS {since: '2023-04-10'}]->(u3)
CREATE (u5)-[:FOLLOWS {since: '2023-04-10'}]->(u4)
```

<u>Output</u>:



Q1) Find all posts by Alex.

-> MATCH (u:User {name: 'Alex'})-[:POSTED]->(p:Post)
RETURN p.title, p.date

Output:

	p.title	p.date
1	"Vacation"	"2023-05-15"
2	"Party"	"2023-06-20"

Q2) Find all users who liked Alex's posts.

```
-> MATCH (u:User)-[:LIKED]->(p:Post)<-[:POSTED]-(poster:User)
WHERE poster.name = 'Alex'
RETURN DISTINCT u.name
```

Output:

	u.name
1	"Priya"

Q3) Find users who haven't posted anything.

```
-> MATCH (u:User)
WHERE NOT (u)-[:POSTED]->()
RETURN u.name
```

Output:

	u.name
1	"Priya"
2	"Sam"
3	"Ryan"

Q4) Find who follows whom and their follow date.
-> MATCH (follower:User)-[r:FOLLOWS]->(followed:User)
RETURN follower.name, followed.name, r.since

<u>Output</u>:

	follower.name	followed.name	r.since
1	"Sam"	"Priya"	"2023-04-10"
2	"Sam"	"Alex"	"2023-04-10"
3	"Sam"	"Maya"	"2023-04-10"
4	"Sam"	"Ryan"	"2023-04-10"

Q5) Find posts with comments and their commenters.

-> MATCH (u:User)-[c:COMMENTED]->(p:Post)
RETURN p.title, u.name, c.text, c.date

<u>Output</u>:

	p.title	u.name	c.text	c.date
1	"Party"	"Ryan"	"Great party!"	"2023-06-21"

Q6) Find users who both posted and commented.

```
-> MATCH (u:User)
WHERE (u)-[:POSTED]->() AND (u)-[:COMMENTED]->()
RETURN u.name
```

Output:

```
(no changes, no records)
```

Q7) Count the number of likes per post.
-> MATCH (post:Post)<-[like:LIKED]-()
RETURN post.title, COUNT(like) as likeCount
ORDER BY likeCount DESC

Output:

	post.title	likeCount
1	"Vacation"	1
2	"Party"	1

Q8) Find the most active user (combining posts, comments, and likes).

```
-> MATCH (u:User)
OPTIONAL MATCH (u)-[:POSTED]->(p:Post)
OPTIONAL MATCH (u)-[:COMMENTED]->(c:Post)
OPTIONAL MATCH (u)-[:LIKED]->(l:Post)
RETURN
u.name,
COUNT(DISTINCT p) as posts,
COUNT(DISTINCT c) as comments,
COUNT(DISTINCT l) as likes,
COUNT(DISTINCT p) + COUNT(DISTINCT c) + COUNT(DISTINCT l) as totalActivity
ORDER BY totalActivity DESC
```

Output:

	u.name	posts	comments	likes	totalActivity
1	"Priya"	0	0	2	2
2	"Alex"	2	0	0	2
3	"Maya"	1	0	0	1
4	"Ryan"	0	1	0	1
5	"Sam"	0	0	0	0

3) <u>Student Course Management</u>

<u>Question</u>: Create Python functions to manage a student course system where you can:

- Add students with name, age, and course
- Get all students in a specific course
- Update student's course
- Delete students who have completed their course"

<u>Solution</u>:

```
from neo4j import GraphDatabase
# connect auradb using driver to get uri
URI = "neo4j+s://9d761d93.databases.neo4j.io"
USER = "neo4j"
# password provided during auradb instance creation
PASSWORD = "dTn-fNT43hNBqGXtlzjxZQx73kJG_Ih7Cj9Jukp47ko"
```

Q1) Function to add students with name, age and course.

```
-> def add_student(name, age, course):
    with GraphDatabase.driver(URI, auth=(USER, PASSWORD)) as driver:
        with driver.session() as session:
            result = session.execute_write(
                lambda tx: tx.run("""
                    CREATE (s:Student {
                        name: $name,
                        age: $age,
                        course: $course
                    }) RETURN s
                    """, name=name, age=age, course=course).data()
            )
            return result
```

Q2) Function to get all students in a specific course.

```
-> def get_course_students(course_name):
    with GraphDatabase.driver(URI, auth=(USER, PASSWORD)) as driver:
        with driver.session() as session:
            result = session.execute_read(
                lambda tx: tx.run("""
                    MATCH (s:Student)
                    WHERE s.course = $course
                    RETURN s.name, s.age
                    """, course=course_name).data()
            )
            return result
```

Q3) Function to update student's course.

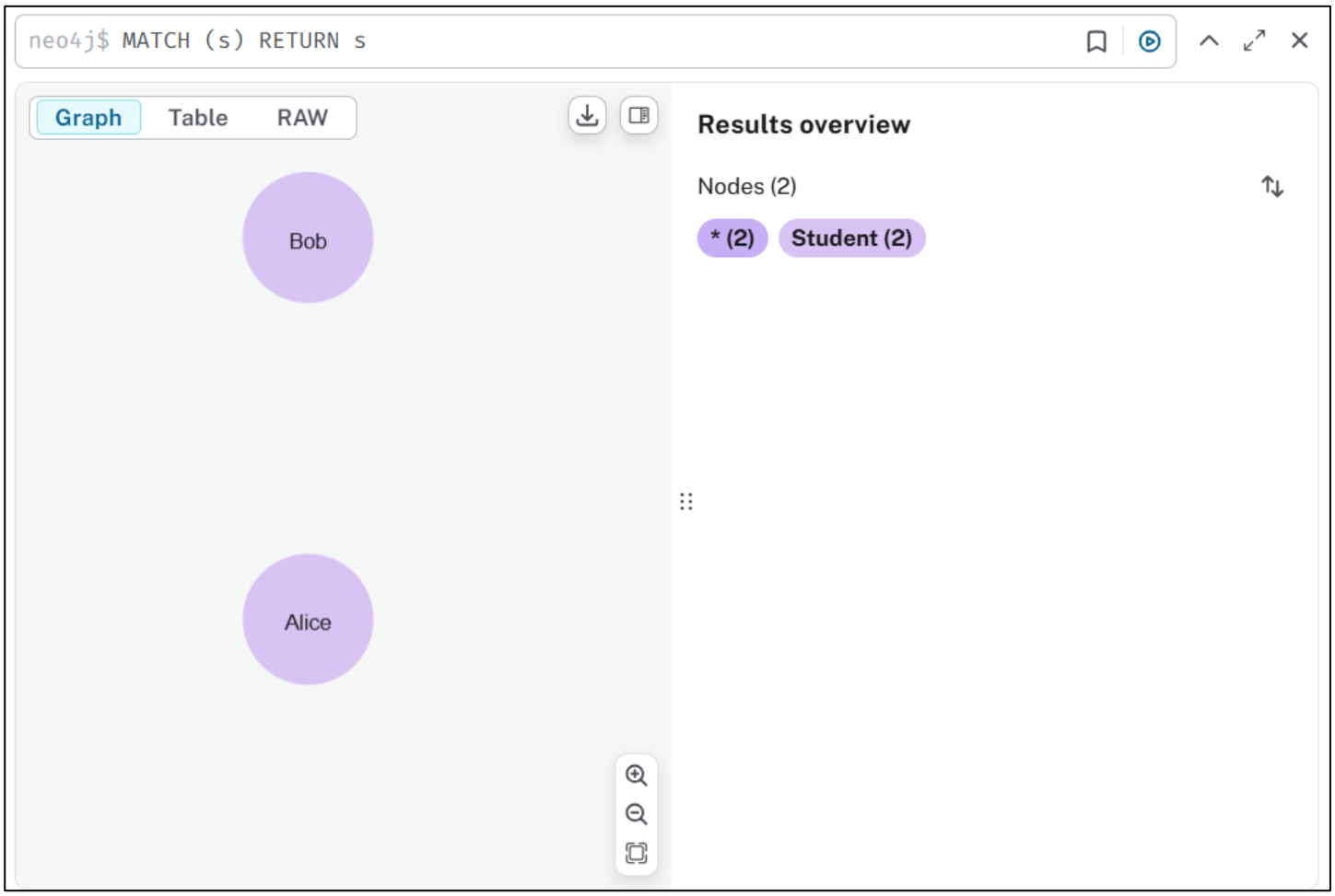
```
-> def update_student_course(student_name, new_course):
    with GraphDatabase.driver(URI, auth=(USER, PASSWORD)) as driver:
        with driver.session() as session:
            result = session.execute_write(
                lambda tx: tx.run("""
                    MATCH (s:Student {name: $name})
                    SET s.course = $course
                    RETURN s """, name=student_name, course=new_course).data()
            )
            return result
```

Q4) Function to delete students who have completed their course.

```
-> def remove_completed_student(student_name):
    with GraphDatabase.driver(URI, auth=(USER, PASSWORD)) as driver:
        with driver.session() as session:
            result = session.execute_write(
                lambda tx: tx.run("""
                    MATCH (s:Student {name: $name})
                    DELETE s """, name=student_name)
            )
            return "Student removed successfully"
```

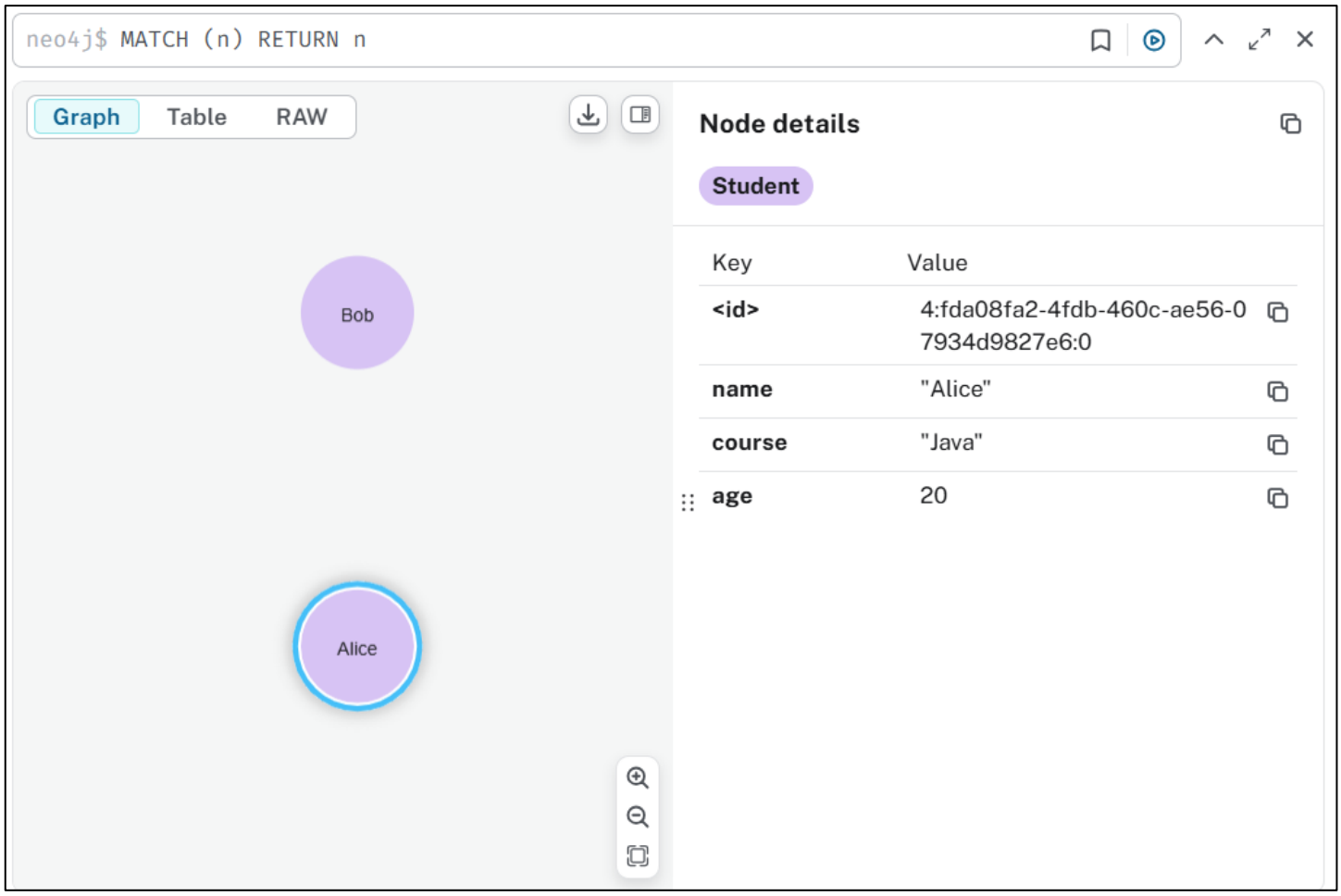
```
// Test the functions
add_student("Alice", 20, "Python")
add_student("Bob", 22, "Java")
print(get_course_students("Python"))
```

<u>Output</u>:



update_student_course("Alice", "Java")

<u>Output</u>:



remove_completed_student("Bob")

Output: