



Maharashtra Education Society's  
Institute of Management and Career Courses (IMCC), Pune  
AUTONOMOUS



**IT13: ADBMS**

**Chapter 4: Graph Database and Analytics**

**Continuous Assessment 3**

**Using Neo4j**

**1. Library Management:**

Question: "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"

Solution:

// 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'})
```



Maharashtra Education Society's  
Institute of Management and Career Courses (IMCC), Pune  
AUTONOMOUS



```
CREATE (m3:Member {name: 'Sophie', joinDate: '2023-03-10'})
```

```
CREATE (m4:Member {name: 'David', joinDate: '2023-04-05'})
```

```
// Create Borrowing Relationships
```

```
CREATE (m1)-[:BORROWED {date: '2023-05-01'}]->(b1)
```

```
CREATE (m1)-[:BORROWED {date: '2023-06-15'}]->(b3)
```

```
CREATE (m2)-[:BORROWED {date: '2023-05-10'}]->(b2)
```

```
CREATE (m3)-[:BORROWED {date: '2023-07-01'}]->(b1)
```

Execute the following Queries:

```
// 1. Find all books borrowed by Emma
```

```
MATCH (m:Member {name: 'Emma'})-[r:BORROWED]->(b:Book)
```

```
RETURN b.title, r.date
```

```
// 2. Find members who haven't borrowed any books
```

```
MATCH (m:Member)
```

```
WHERE NOT (m)-[:BORROWED]->()
```

```
RETURN m.name
```

```
// 3. Find books that have been borrowed more than once
```

```
MATCH (m:Member)-[:BORROWED]->(b:Book)
```

```
WITH b, COUNT(*) as borrowCount
```

```
WHERE borrowCount > 1
```



Maharashtra Education Society's  
Institute of Management and Career Courses (IMCC), Pune  
AUTONOMOUS



RETURN b.title, borrowCount

// 4. 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



Maharashtra Education Society's  
Institute of Management and Career Courses (IMCC), Pune  
AUTONOMOUS



## 2. Social Media Network :

Question: "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"

### Solution:

// 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



Maharashtra Education Society's  
Institute of Management and Career Courses (IMCC), Pune  
AUTONOMOUS



// Posts ownership

CREATE (u1)-[:POSTED]->(p1)

CREATE (u1)-[:POSTED]->(p2)

CREATE (u2)-[:POSTED]->(p3)

// Comments

CREATE (u3)-[:COMMENTED {

text: 'Great party!',

date: '2023-06-21'

}]->(p2)

// Likes

CREATE (u4)-[:LIKED {date: '2023-05-16'}]->(p1)

CREATE (u4)-[:LIKED {date: '2023-06-22'}]->(p2)

// Sam follows everyone

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)



Maharashtra Education Society's  
Institute of Management and Career Courses (IMCC), Pune  
AUTONOMOUS



**Execute the following Queries:**

**// 1. Find all posts by Alex**

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

**// 2. 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
```

**// 3. Find users who haven't posted anything**

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

**// 4. Find who follows whom and their follow date**

```
MATCH (follower:User)-[r:FOLLOWS]->(followed:User)
RETURN follower.name, followed.name, r.since
```

**// 5. Find posts with comments and their commenters**

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

**Extra Query Challenges (Difficult Level):**

**// 1. Find users who both posted and commented**

```
MATCH (u:User)
```



Maharashtra Education Society's  
Institute of Management and Career Courses (IMCC), Pune  
AUTONOMOUS



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

**// 2. Count the number of likes per post**

```
MATCH (post:Post)<-[:LIKED]-()
RETURN post.title, COUNT(like) as likeCount
ORDER BY likeCount DESC
```

**// 3. 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
```



Maharashtra Education Society's  
Institute of Management and Career Courses (IMCC), Pune  
AUTONOMOUS



### 3. Student Course Management:

Question: "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"

Solution:

```
from neo4j import GraphDatabase
```

```
URI = "your_uri"
```

```
USER = "neo4j"
```

```
PASSWORD = "your_password"
```

```
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
```





Maharashtra Education Society's  
Institute of Management and Career Courses (IMCC), Pune  
AUTONOMOUS



```
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
```

```
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
```

```
def remove_completed_student(student_name):  
    with GraphDatabase.driver(URI, auth=(USER, PASSWORD)) as driver:  
        with driver.session() as session:
```



Maharashtra Education Society's  
Institute of Management and Career Courses (IMCC), Pune  
AUTONOMOUS



```
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"))  
update_student_course("Alice", "Java")  
remove_completed_student("Bob")
```