Лабораторная работа №5-6, Часть 2: Работа с графовыми базами данных (Neo4j)

Цель работы: Освоить принципы работы с графовыми базами данных на примере Neo4j. Получить практические навыки создания узлов и связей, выполнения запросов на языке Cypher и визуализации графовых структур.

Стек технологий:

- **ОС:** Ubuntu 24.04 LTS
- **Окружение:** Docker, Conda (`mlops-lab`)
- **Библиотеки:** `neo4j`, `pandas`, `matplotlib`
- **База данных:** Neo4j (Docker контейнер)
- **Язык запросов:** Cypher
- **Интерфейс:** Neo4j Browser

Теоретическая часть

1. Графовые базы данных Графовые БД предназначены для хранения и обработки данных в виде графов (узлов и связей). Ключевые особенности:

- **Узлы (Nodes):** Сущности данных (объекты)
- **Связи (Relationships):** Отношения между узлами
- **Свойства (Properties):** Атрибуты узлов и связей
- **Метки (Labels):** Категории узлов

2. Neo4j Ведущая графовая база данных с открытым исходным кодом:

- **Cypher:** Декларативный язык запросов
- **ACID-совместимость:** Поддержка транзакций
- **Визуализация:** Интерактивный просмотр графов

3. Применение в управлении знаниями

- Построение онтологий и знанийых графов
- Анализ социальных сетей
- Рекомендательные системы
- Поиск паттернов и взаимосвязей

Задание на практическую реализацию

Этап 1: Запуск Neo4j в Docker

1. **Запуск Neo4j контейнера:**

```
docker run \
    --name neo4j-graphdb \
    -p 7474:7474 \
    -p 7687:7687 \
    -d \
    -v $(pwd)/neo4j/data:/data \
    -v $(pwd)/neo4j/logs:/logs \
    -v $(pwd)/neo4j/import:/var/lib/neo4j/import \
    --env NEO4J_AUTH=neo4j/password123 \
    neo4j:latest
```

2. **Проверка работы контейнера:**

```
docker ps
docker logs neo4j-graphdb
```

Этап 2: Подключение к Neo4j Browser

1. **Открытие веб-интерфейса:**
   - Откройте браузер и перейдите по адресу: `http://localhost:7474`
   - Логин: `neo4j`
   - Пароль: `password123`
2. **Изменение пароля:**
   - При первом входе измените пароль на `graphdb2024`

Этап 3: Основы языка Cypher

1. **Создание скрипта для работы с Neo4j:**

```
touch neo4j_demo.py
```

2. **Подключение к базе данных:**

```
from neo4j import GraphDatabase
import pandas as pd

# Настройки подключения
URI = "bolt://localhost:7687"
AUTH = ("neo4j", "graphdb2024")

# Создание драйвера
driver = GraphDatabase.driver(URI, auth=AUTH)
```

```
def test_connection():
    with driver.session() as session:
        result = session.run("RETURN 'Connected to Neo4j' AS message")
        return result.single()["message"]

print(test_connection())
```

Этап 4: Создание онтологии предметной области

1. **Создание узлов и связей:**

```
def create_knowledge_graph(tx):
    # Очистка базы данных
    tx.run("MATCH (n) DETACH DELETE n")

    # Создание узлов (сущностей)
    query = """
    CREATE
    (ai:Domain {name: 'Artificial Intelligence'}),
    (ml:Technology {name: 'Machine Learning', type: 'ML'}),
    (dl:Technology {name: 'Deep Learning', type: 'DL'}),
    (nlp:Technology {name: 'NLP', type: 'NLP'}),
    (bert:Model {name: 'BERT', developer: 'Google'}),
    (gpt:Model {name: 'GPT', developer: 'OpenAI'}),
    (python:Language {name: 'Python', paradigm: 'multi-paradigm'}),
    (pytorch:Framework {name: 'PyTorch', language: 'Python'}),
    (tf:Framework {name: 'TensorFlow', language: 'Python'}),

    // Создание связей
    (ai)-[:INCLUDES]->(ml),
    (ai)-[:INCLUDES]->(nlp),
    (ml)-[:CONTAINS]->(dl),
    (nlp)-[:USES]->(bert),
    (nlp)-[:USES]->(gpt),
    (ml)-[:IMPLEMENTED_IN]->(python),
    (dl)-[:FRAMEWORK]->(pytorch),
    (dl)-[:FRAMEWORK]->(tf),
    (bert)-[:BUILT_WITH]->(tf),
    (gpt)-[:BUILT_WITH]->(pytorch),
    (python)-[:HAS_FRAMEWORK]->(pytorch),
    (python)-[:HAS_FRAMEWORK]->(tf)
    """
    tx.run(query)

with driver.session() as session:
    session.execute_write(create_knowledge_graph)
    print("База знаний создана")
```

Этап 5: Выполнение базовых запросов

1. **Поиск всех узлов:**

```
def get_all_nodes(tx):
    result = tx.run("MATCH (n) RETURN n.name AS name, labels(n) AS labels")
    return [{"name": record["name"], "labels": record["labels"]} for record
in result]

print("Все узлы в базе:")
nodes = get_all_nodes(driver.session())
for node in nodes:
    print(f"{node['name']} - {node['labels']}")
```

2. **Поиск связей:**

```
def get_relationships(tx):
    query = """
    MATCH (a)-[r]->(b)
    RETURN a.name AS source, type(r) AS relationship, b.name AS target
    """
    result = tx.run(query)
    return [{"source": record["source"], "relationship":
record["relationship"], "target": record["target"]} for record in result]

print("\nСвязи в графе:")
relationships = get_relationships(driver.session())
for rel in relationships:
    print(f"{rel['source']} --{rel['relationship']}--> {rel['target']}")
```

Этап 6: Сложные запросы и анализ

1. **Поиск путей:**

```
def find_paths(tx, start_node, end_node):
    query = """
    MATCH path = (a {name: $start_node})-[*]->(b {name: $end_node})
    RETURN [node in nodes(path) | node.name] AS path_nodes,
           [rel in relationships(path) | type(rel)] AS relationships
    """
    result = tx.run(query, start_node=start_node, end_node=end_node)
    return [{"nodes": record["path_nodes"], "relationships":
record["relationships"]} for record in result]

print("\nПути от AI до BERT:")
paths = find_paths(driver.session(), "Artificial Intelligence", "BERT")
for path in paths:
    print(f"Путь: {' -> '.join(path['nodes'])}")
```

2. **Анализ степени связности:**

```
def analyze_connectivity(tx):
    query = """
    MATCH (n)
    RETURN n.name AS node,
           size((n)--()) AS degree,
           labels(n)[0] AS type
    ORDER BY degree DESC
    """
    result = tx.run(query)
    return [{"node": record["node"], "degree": record["degree"], "type":
record["type"]} for record in result]

print("\nАнализ связности узлов:")
connectivity = analyze_connectivity(driver.session())
for node in connectivity:
    print(f"{node['node']} ({node['type']}): {node['degree']} связей")
```

Этап 7: Работа с реальными данными

1. **Импорт данных из CSV:**

```
def import_ai_researchers(tx):
    # Создание узлов исследователей
    query = """
    LOAD CSV WITH HEADERS FROM 'file:///ai_researchers.csv' AS row
    CREATE (:Researcher {
        name: row.name,
        affiliation: row.affiliation,
        field: row.field,
        h_index: toInteger(row.h_index)
    })
    """
    tx.run(query)

# Создание CSV файла для импорта
researchers_data = """name,affiliation,field,h_index
Yann LeCun,Facebook AI Research,Computer Vision,180
Andrew Ng,Stanford University,Machine Learning,150
Yoshua Bengio,MILA,Deep Learning,170
Geoffrey Hinton,University of Toronto,Neural Networks,200
Demis Hassabis,Google DeepMind,Reinforcement Learning,80
"""

with open("neo4j/import/ai_researchers.csv", "w") as f:
    f.write(researchers_data)

with driver.session() as session:
```

```
    session.execute_write(import_ai_researchers)
    print("Данные исследователей импортированы")
```

2. **Создание связей с существующей онтологией:**

```
def connect_researchers_to_domains(tx):
    query = """
    MATCH (r:Researcher), (d:Domain {name: 'Artificial Intelligence'})
    CREATE (r)-[:WORKS_IN]->(d)

    MATCH (r:Researcher {name: 'Yann LeCun'}), (dl:Technology {name: 'Deep
Learning'})
    CREATE (r)-[:CONTRIBUTED_TO]->(dl)

    MATCH (r:Researcher {name: 'Andrew Ng'}), (ml:Technology {name: 'Machine
Learning'})
    CREATE (r)-[:CONTRIBUTED_TO]->(ml)
    """
    tx.run(query)

with driver.session() as session:
    session.execute_write(connect_researchers_to_domains)
    print("Связи исследователей созданы")
```

Этап 8: Визуализация и экспорт

1. **Визуализация графа через Neo4j Browser:**
   - Откройте Neo4j Browser (`http://localhost:7474`)
   - Выполните запрос: `MATCH (n) RETURN n LIMIT 25`
   - Нажмите "View in graph" для визуализации

2. **Экспорт данных:**

```
def export_graph_data(tx):
    query = """
    MATCH (n)
    RETURN n.name AS name,
           labels(n) AS labels,
           properties(n) AS properties
    """
    result = tx.run(query)
    df = pd.DataFrame([dict(record) for record in result])
    df.to_csv("graph_export.csv", index=False)
    return df

graph_data = export_graph_data(driver.session())
print("\nЭкспортированные данные:")
print(graph_data.head())
```

3. **Закрытие соединения:**

```
driver.close()
print("Соединение с Neo4j закрыто")
```

4. **Остановка контейнера:**

```
docker stop neo4j-graphdb
docker rm neo4j-graphdb
```

Требования к оформлению и отчету

Критерии оценки для Части 2:

- **Удовлетворительно:** Успешно выполнены Этапы 1-4 (запуск Neo4j, создание базовой онтологии). Скрипт запускается без ошибок.
- **Хорошо:** Дополнительно успешно выполнен Этап 5-6 (реализованы базовые и сложные запросы, анализ связности). Получены осмысленные результаты запросов.
- **Отлично:** Все задания выполнены в полном объеме. Реализованы расширенные функции (Этапы 7-8): импорт реальных данных, визуализация графа, экспорт результатов. Проанализирована структура графа знаний.

Рекомендуемая литература

1. **Neo4j Documentation:** https://neo4j.com/docs/
2. **Cypher Query Language:** https://neo4j.com/developer/cypher/
3. **Graph Databases Book:** https://graphdatabases.com/
4. **Neo4j Python Driver:** https://neo4j.com/docs/python-manual/current/
5. **Knowledge Graphs:** https://www.ontotext.com/knowledgehub/fundamentals/what-is-a-knowledge-graph/