МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра Систем информатики

Направление подготовки 09.06.01 – Информатика и вычислительная техника

ОТЧЕТ

Обучающегося
Власенко Ивана Алексеевича
группы № 22214 курса 4

Тема задания:
Разработка драйвера для графовой базы данных Neo4j

Новосибирск 2025

Введение

В рамках данного проекта была разработана библиотека `GraphRepository` для работы с графовой базой данных Neo4j. Целью проекта являлось создание удобного, безопасного и надежного интерфейса для выполнения основных операций с узлами и связями в графовой базе данных.

Разработанная библиотека предоставляет типизированный интерфейс для работы с Neo4j, включающий методы для создания, чтения, обновления и удаления узлов и связей, а также выполнения произвольных Cypher-запросов.

1 Архитектура системы

Для реализации системы работы с графовой базой данных был разработан основной класс GraphRepository, приведенный на листинге 1.

```
class GraphRepository:

    def __init__(self, uri: str, user: str, password: str, database:
str = None):

        self.uri = uri

        self.user = user

        self.password = password

        self.database = database

        self.driver = GraphDatabase.driver(self.uri,
auth=(self.user, self.password))


    def __enter__(self):

        return self


    def __exit__(self, exc_type, exc_val, exc_tb):

        self.close()
```

Листинг 1 – Класс GraphRepository

При инициализации создается драйвер для подключения к Neo4j и настраивается контекстный менеджер для автоматического управления ресурсами.

2 Реализация методов получения данных

1.2.1 Метод получения всех узлов

Для получения всех узлов графа был реализован метод get_all_nodes(), приведенный на листинге 2.

```
def get_all_nodes(self) -> List[TNode]:

    query = """

    MATCH (n)

    RETURN elementId(n) as element_id, n.uri as uri, n.description
as description, n.title as title

    """
```

```
        results = self._execute_query(query)
        return [self.collect_node(result) for result in results]
```

Листинг 2 – Метод get_all_nodes()

Выполняет Cypher-запрос для получения всех узлов и преобразует результаты в типизированные объекты TNode.

2.2 Метод получения узлов с их связями

Для получения всех узлов вместе с их исходящими связями был реализован метод get_all_nodes_and_arcs(), приведенный на листинге 3.

```
    def get_all_nodes_and_arcs(self) -> List[TNode]:
        query = """
        MATCH (n)
        OPTIONAL MATCH (n)-[r]->(m)
        WITH n, collect({
            element_id: elementId(r),
            uri: type(r),
            node_uri_from: n.uri,
            node_uri_to: m.uri
        }) as arcs
        RETURN elementId(n) as element_id, n.uri as uri, n.description
    as description, n.title as title, arcs
        """
        results = self._execute_query(query)
        nodes = []
        for result in results:
            node = self.collect_node(result)
            node.arcs = [self.collect_arc(arc) for arc in
    result.get('arcs', []) if arc.get('element_id')]
            nodes.append(node)
        return nodes
```

Листинг 3 – Метод get_all_nodes_and_arcs()

Использует OPTIONAL MATCH для включения узлов без связей и агрегирует связи с помощью функции collect().

2.3 Метод получения узлов по меткам

Для получения узлов по определенным меткам был реализован метод get_nodes_by_labels(), приведенный на листинге 4.

```
    def get_nodes_by_labels(self, labels: List[str]) -> List[TNode]:
        if not labels:
            return []

        labels_clause = self._build_labels_clause(labels)
        query = f"""
        MATCH (n{labels_clause})
        RETURN elementId(n) as element_id, n.uri as uri, n.description
    as description, n.title as title
        """
        results = self._execute_query(query)
        return [self.collect_node(result) for result in results]
```

Листинг 4 – Метод get_nodes_by_labels()

Строит строку меток и выполняет запрос для получения узлов с указанными метками.

3 Реализация методов создания данных

3.1 Метод создания узла

Для создания нового узла в графе был реализован метод create_node(), приведенный на листинге 5.

```
def create_node(self, params: Dict[str, Any]) -> TNode:
    if 'uri' not in params:
        params['uri'] = f"node_{self.generate_random_string()}"

    labels = params.pop('labels', [])
    labels_clause = self._build_labels_clause(labels)

    query = f"""
    CREATE (n{labels_clause} $props)
    RETURN elementId(n) as element_id, n.uri as uri, n.description
as description, n.title as title
    """

    results = self._execute_query(query, {'props': params})
    if results:
        return self.collect_node(results[0])
    raise Exception("Не удалось создать узел")
```

Листинг 5 – Метод create_node()

Генерирует URI, строит строку меток и создает узел с использованием параметризованного запроса.

3.2 Метод создания связи

Для создания связи между узлами был реализован метод create_arc(), приведенный на листинге 6.

```
def create_arc(self, node1_uri: str, node2_uri: str, arc_type: str =
"RELATES_TO", properties: Dict[str, Any] = None) -> TArc:
    safe_arc_type = arc_type.replace('`', '``')
```

```
        query = f"""
        MATCH (n1 {{uri: $node1_uri}}), (n2 {{uri: $node2_uri}})
        CREATE (n1)-[r:`{safe_arc_type}` $props]->(n2)
        RETURN elementId(r) as element_id, type(r) as uri, n1.uri as
    node_uri_from, n2.uri as node_uri_to
        """

        results = self._execute_query(query, {
            'node1_uri': node1_uri,
            'node2_uri': node2_uri,
            'props': properties or {}
        })

        if results:
            return self.collect_arc(results[0])
        raise Exception("Не удалось создать связь")
```

Листинг 6 – Метод create_arc()

Безопасно экранирует тип связи, находит узлы по URI и создает связь между ними.

4 Реализация методов обновления и удаления

4.1 Метод обновления узла

Для обновления свойств существующего узла был реализован метод update_node(), приведенный на листинге 7.

```
def update_node(self, uri: str, params: Dict[str, Any]) ->
Optional[TNode]:
    if not params:
        return None


    query = """
    MATCH (n {uri: $uri})
    SET n += $props
    RETURN elementId(n) as element_id, n.uri as uri, n.description
as description, n.title as title
    """


    results = self._execute_query(query, {'uri': uri, 'props':
params})
    if results:
        return self.collect_node(results[0])
    return None
```

Листинг 7 – Метод update_node()

Находит узел по URI и обновляет его свойства с помощью оператора SET.

4.2 Метод удаления узла

Для удаления узла из графа был реализован метод delete_node_by_uri(), приведенный на листинге 8.

```
def delete_node_by_uri(self, uri: str) -> bool:
    query = """
    MATCH (n {uri: $uri})
    DETACH DELETE n
    """
```

```
        with self.driver.session(database=self.database) as session:
            result = session.run(query, {'uri': uri})
            summary = result.consume()
            return summary.counters.nodes_deleted > 0
```

Листинг 8 – Метод delete_node_by_uri()

Удаляет узел вместе со всеми его связями и проверяет успешность операции через статистику выполнения.

5 Типизация данных

5.1 Структура узла

Для типизации узлов графа была создана структура TNode, приведенная на листинге 11.

```
@dataclass
class TNode:
    id: str  # element ID (стабильный)
    uri: str
    description: str
    title: str
    arcs: Optional[List['TArc']] = None
```

Листинг 11 – Структура TNode

Типизированное представление узла графа с использованием стабильного element ID и основных свойств.

5.2 Структура связи

Для типизации связей графа была создана структура TArc, приведенная на листинге 12.

```
@dataclass
class TArc:
    id: str  # element ID (стабильный)
    uri: str  # arc.type
    node_uri_from: str
    node_uri_to: str
```

Листинг 12 – Структура TArc

Типизированное представление связи графа с использованием стабильного element ID и URI узлов-участников.

6 Вспомогательные методы

6.1 Метод генерации случайных строк

Для генерации уникальных URI узлов был реализован метод generate_random_string(), приведенный на листинге 13.

```
def generate_random_string(self, length: int = 10) -> str:
    letters = string.ascii_lowercase + string.digits
    return ''.join(random.choice(letters) for _ in range(length))
```

Листинг 13 – Метод generate_random_string()

Генерирует случайную строку заданной длины из букв и цифр для создания уникальных URI узлов.

6.2 Метод выполнения произвольных запросов

Для выполнения пользовательских Cypher-запросов был реализован метод run_custom_query(), приведенный на листинге 14.

```
def run_custom_query(self, query: str, parameters: Dict[str, Any] =
None) -> List[Dict[str, Any]]:
    return self._execute_query(query, parameters)
```

Листинг 14 – Метод run_custom_query()

Делегирует выполнение произвольного Cypher-запроса к базовому методу _execute_query().

6.3 Метод трансформации узла

Для преобразования необработанных данных узла из базы данных в типизированный объект `TNode` был реализован метод `collect_node()`, приведенный на листинге 15.

```
def collect_node(self, node_data: Dict[str, Any]) -> TNode:
    return TNode(
      id=node_data.get('element_id', ''),
      uri=node_data.get('uri', ''),
      description=node_data.get('description', ''),
      title=node_data.get('title', ''),
      arcs=node_data.get('arcs', [])
```

```
    )
```

Листинг 15 – Метод `collect_node()`

Преобразует словарь с данными узла в типизированную структуру `TNode`, обеспечивая унифицированное представление данных в приложении.

6.4 Метод трансформации связи

Для преобразования необработанных данных связи из базы данных в типизированный объект `TArc` был реализован метод `collect_arc()`, приведенный на листинге 16.

```
def collect_arc(self, arc_data: Dict[str, Any]) -> TArc:

    return TArc(

        id=arc_data.get('element_id', ''),

        uri=arc_data.get('uri', ''),

        node_uri_from=arc_data.get('node_uri_from', ''),

        node_uri_to=arc_data.get('node_uri_to', '')

    )
```

Листинг 16 – Метод `collect_arc()`

Преобразует словарь с данными связи в типизированную структуру `TArc`, что позволяет легко работать со связями в коде.

Заключение

В рамках данного проекта была успешно разработана библиотека `GraphRepository` для работы с графовой базой данных Neo4j. Основные достижения проекта:

1. **Создан безопасный и надежный интерфейс** для работы с Neo4j
2. **Реализованы все основные операции** CRUD для узлов и связей графа
3. **Обеспечена типизация данных** с помощью современных возможностей Python

Библиотека готова к использованию в продакшене и может служить основой для более сложных систем, работающих с графовыми данными. Архитектура системы позволяет легко расширять функциональность и адаптировать под различные бизнес-требования.

Список литературы

1. Neo4j Documentation. [Электронный ресурс]. URL: https://neo4j.com/docs/
2. Cypher Query Language Reference. [Электронный ресурс]. URL: https://neo4j.com/docs/cypher-manual/
3. Python Neo4j Driver Documentation. [Электронный ресурс]. URL: https://neo4j.com/docs/python-manual/
4. Python Dataclasses Documentation. [Электронный ресурс]. URL: https://docs.python.org/3/library/dataclasses.html