**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** Козубенко Андрея Алексеевича **группы № 22215 курса 4**

(Ф.И.О. полностью)

**Тема задания**: Разработка драйвера для графовой базы данных Neo4j

Новосибирск 2025

Оглавление

**Введение3**

**Описание реализации4**

**Перечень реализованных методов5**

**Особенности реализации7**

**Тестирование и пример использования7**

**Реализация7**

**Заключение8**

**Список литературы9**

**Введение**

Цель данной лабораторной работы — разработать репозиторий (драйвер/ORM-слой) на языке Python для взаимодействия с графовой базой данных Neo4j, реализующий стандартные CRUD операции для узлов и связей, а также дополнительные вспомогательные функции и возможность выполнения произвольных Cypher-запросов.

**Задачи:**

* Спроектировать API репозитория с методами для создания, чтения, обновления и удаления узлов и дуг.
* Реализовать сериализацию/десериализацию объектов БД в удобный для приложения формат (TNode, TArc).
* Обеспечить вспомогательные функции: генерация URI, трансформация меток и свойств в Cypher.
* Подготовить отчёт с описанием реализованных функций и примерами использования.

**Описание реализации**

Репозиторий реализован в виде класса Neo4jRepository (файл neo4j\_repository.py) с использованием официальной библиотеки neo4j (импорт GraphDatabase). Класс инкапсулирует драйвер и предоставляет методы, перечисленные в задании.

Типы данных:

* TNode — словарь с ключами:
  + id: number — внутренний id в Neo4j (если доступен)
  + uri: string — уникальный строковый идентификатор узла (параметр)
  + title: string
  + description: string
  + arcs?: TArc[] — список исходящих дуг (опционально)
* TArc — словарь с ключами:
  + id: number — внутренний id дуги
  + uri: string — тип дуги (name/type rel)
  + node\_uri\_from: string
  + node\_uri\_to: string

**Перечень реализованных методов**

Ниже дано краткое описание каждого метода (в отчёте нужно описать назначение, входные параметры и возвращаемое значение).

1. get\_all\_nodes() -> List[TNode]
   * Описание: возвращает список всех узлов в базе. Для каждого узла собираются свойства и внутренний id.
   * Вход: нет.
   * Выход: List[TNode].
2. get\_all\_nodes\_and\_arcs() -> List[TNode]
   * Описание: возвращает все узлы и связанные с ними дуги; у узлов в поле arcs присутствуют исходящие дуги в виде TArc.
   * Технически: выполняется MATCH (a)-[r]->(b) RETURN a,r,b, после чего собираются узлы и арки в map по uri.
3. get\_nodes\_by\_labels(labels: List[str]) -> List[TNode]
   * Описание: выбирает узлы по переданному списку меток (label). Поддерживает несколько меток.
4. get\_node\_by\_uri(uri: str) -> Optional[TNode]
   * Описание: ищет узел по uri (свойство узла). Возвращает TNode или None.
5. create\_node(params: Dict[str, Any], labels: Optional[List[str]] = None) -> TNode
   * Описание: создаёт узел с указанными свойствами. Если uri не передан, автоматически генерируется с помощью generate\_random\_string.
   * Возвращает созданный узел.
6. create\_arc(node1\_uri: str, node2\_uri: str, rel\_type: str = "RELATED", props: Optional[Dict[str, Any]] = None) -> Optional[TArc]
   * Описание: создаёт направленную дугу между узлами, найденными по uri. Возвращает объект дуги.
7. delete\_node\_by\_uri(uri: str) -> bool
   * Описание: удаляет узел (и связанные с ним арки) по uri. Возвращает True если удалён хотя бы один узел.
8. delete\_arc\_by\_id(arc\_id: int) -> bool
   * Описание: удаляет дугу по её внутреннему id (функция использует id(r) в Cypher).
9. update\_node(uri: str, props: Dict[str, Any]) -> Optional[TNode]
   * Описание: обновляет/дополняет свойства узла, найденного по uri. Возвращает обновлённый узел.
10. generate\_random\_string(length: int = 12) -> str
    * Описание: генерирует уникальную строку для uri (используется uuid4).
11. run\_custom\_query(query: str, params: Dict[str, Any] = None) -> List[Dict[str, Any]]
    * Описание: выполняет произвольный Cypher-запрос и возвращает список строк результата в виде словарей.
12. collect\_node(node) -> TNode и collect\_arc(arc) -> TArc
    * Описание: вспомогательные функции для трансляции объектов Neo4j (neo4j.Node/neo4j.Relationship) в плоские словари (TNode, TArc) для удобства дальнейшей работы приложения.

**Особенности реализации**

* Для безопасного встраивания свойств в Cypher используются функции transform\_props и transform\_labels. transform\_props использует json.dumps для безопасного представления значений.
* Все операции выполняются в рамках session через with self.driver.session().
* В get\_all\_nodes\_and\_arcs реализовано объединение узлов и агрегация дуг в поле arcs у каждого узла.
* Для генерации uri используется uuid4 (успешно короткий и почти никогда не конфликтует).

**Тестирование и пример использования**

Пример сценария (см. example\_usage.py): создание двух узлов, создание дуги, получение всех узлов с дугами, обновление и удаление.

**Реализация**

[**https://github.com/andrew-kozubenko/Linguistics**](https://github.com/andrew-kozubenko/Linguistics)

Находится в **lab2/neo4j\_repository.py**

**Заключение**

В результате проделанной работы реализован модуль Neo4jRepository, который покрывает все пункты задания: CRUD-операции для узлов и дуг, утилиты по трансформации меток/свойств, генерация uri и выполнение произвольных запросов. API спроектировано так, чтобы его было удобно расширять: collect-функции централизуют логику десериализации, что упростит дальнейшее добавление новых полей и поддерживаемых типов.

**Список литературы**

1. Документация Neo4j — официальный сайт neo4j.com.
2. Python driver for Neo4j — пакет neo4j.