Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика, искусственный интеллект и системы управления

КАФЕДРА Системы обработки информации и управления

Методические указания к лабораторным работам по курсу «Постреляционные базы данных»

Лабораторная работа №5
«Создание графовой базы данных и работа с ней на примере СУБД Neo4j»

Виноградова М.В., Королев С.В., Соболева Е.Д.

Под редакцией к.т.н. доц. Виноградовой М.В.

Москва, 2022 г.

ОГЛАВЛЕНИЕ

1. ЗАДАНИЕ....4
Цель работы....4
Средства выполнения....4
Продолжительность работы....4
Пункты задания для выполнения....4
Содержание отчета....5
2. ТЕОРЕТИКО-ПРАКТИЧЕСКИЙ МАТЕРИАЛ....6
Особенности графовой СУБД Neo4j....6
Узлы, свойства и метки....7
Типы данных....7
Основные команды языка Cypher....8
Доступность....10
Согласованность данных....10
Транзакции....10
Масштабирование....11
Запуск и установка Neo4j....13
Настройка СУБД Neo4j с помощью exe-файла Windows....13
Установка и настройка Neo4j на сервере с Ubuntu или Debian....19
Возможности среды Neo4j Data Browser....23
Команды определения структуры данных....26
Создание узла со свойствами....26
Удаление и изменение свойств и меток....27
Отношения между узлами....31
Создание отношения между новыми узлами....36
Создание отношений между несколькими узлами....38
Удаление узлов и связей....40
Запросы к БД на языке Cypher....44
Основные запросы на выборку данных....44
Фильтрация по узлам, отношениям, меткам и связям....50
Условие NOT NULL....53
Операторы AND, OR....54
Сортировка по свойству....55
Условие на направление отношения....56
Параметры отношения....57
Существование шаблона в базе данных....59
Объединение результатов запросов....60
Расширенные запросы к БД....62
Ограничение количества строк в выводе....62
Удаление дубликатов....63
Пропуск узлов при выводе....65
Агрегирование....66
Строковые функции....68
Индексы....70
Просмотр списка индексов....73
Удаление индекса....73
Ограничения....74

Ограничение уникальности свойства....74
Ограничение на существование свойства узла....74
Ограничение на существование свойства отношения....75
Ограничение на существование ключа узла....75
Удаление ограничения....75
Вывод всех ограничений....76
Коллекции или списки....77
Создание списка....77
Работа с коллекциями (списками)....77
Генерация коллекций (списков)....79
3. КОНТРОЛЬНЫЕ ВОПРОСЫ....83
4. СПИСОК ИСТОЧНИКОВ....84

1. ЗАДАНИЕ

Лабораторная работа №5 «Создание графовой базы данных и работа с ней на примере СУБД Neo4j» по курсу «Постреляционные базы данных».

Цель работы

- Изучить модель представления данных и способы работы с графовыми БД;
- Освоить методы создания графовой БД и языки запросов к ней;
- Получить навыки работы с графовой СУБД Neo4j.

Средства выполнения

- СУБД Neo4j [1.];
- Утилита Neo4j Server.

Продолжительность работы

Время выполнения лабораторной работы 4 часа.

Пункты задания для выполнения

Базовое:

1. Создать в среде Neo4j базу данных по теме, выданной преподавателем. Определить набор узлов, задать их свойства и метки.
2. Продемонстрировать (вывести на экран) содержимое БД (узлы и их свойства), используя команды Match/Where/Return.
3. Создать набор узлов БД. Определить для них несколько меток и свойств. Создать отношение между новыми узлами. Создать отношение между существующими узлами.
4. Продемонстрировать удаление узлов и связей. Продемонстрировать удаление и изменение свойств и меток.
5. Продемонстрировать содержимое БД (фильтрация по узлам, отношениям, меткам и связям).
6. Выполнить запросы к базе данных на языке Cypher:

- с условием NOT NULL
- операторами AND, OR
- с сортировкой
- с условием на направление отношения
- с параметрами отношения

Расширенное:

7. Продемонстрировать работу команд LIMIT, SKIP
8. Продемонстрировать работу команд UNION, MERGE.
9. Выполнить запросы к базе данных на языке Cypher:
    - с агрегированием,
    - с встроенными функциями (строковые или иные),
    - с указанием начального узла (точка входа),
    - с шаблонами отношений,
    - с удалением дубликатов.

Дополнительное:

10. Создать индекс. Продемонстрировать его использование.
11. Создать ограничение. Продемонстрировать его использование.
12. Создать коллекцию. Продемонстрировать запрос к ней.

Содержание отчета

- Титульный лист;
- Цель работы;
- Задание;
- Тексты запросов и команд в соответствии с пунктами задания.
- Результаты выполнения запросов (скриншоты);
- Вывод;
- Список используемой литературы.

2. ТЕОРЕТИКО-ПРАКТИЧЕСКИЙ МАТЕРИАЛ

Особенности графовой СУБД Neo4j

Графовая база данных — это такая база данных, которая использует графовые структуры для построения семантических запросов с узлов, ребер и свойств в процессе представления и хранения данных [2.]. Графовые БД используются для хранения, управления и составления запросов к сложным и тесно взаимосвязанным группам данных. Помимо этого, архитектура графовой БД особенно хорошо приспособлена к анализу данных на предмет обнаружения совпадений и аномалий в обширных массивах данных, а также к выгодному использованию заключенных в БД взаимосвязей.

Компоненты графовой базы данных — узлы и ребра. Они могут быть дополнены собственным набором полей. Модель такой БД схематично изображена на Рисунок 1.

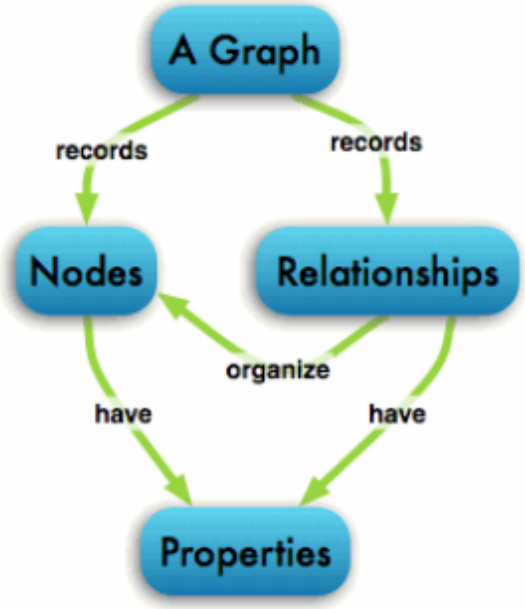


Рисунок 1 – Модель графовой базы данных

В настоящей лабораторной работе будем рассматривать графовые базы данных на примере СУБД Neo4j [3.].

Neo4j – это открытая и самая передовая в мире система управления графовыми базами данных графа, разработана Neo Technology, Inc. Она предназначена для оптимизации поиска, быстрого управления, хранения и обхода узлов и отношений.

Она обеспечивает эффективность и высокую производительность в режиме реального времени, что позволяет разработчикам создавать приложения для удовлетворения возникающих проблем с большими объемами данных.

Узлы, свойства и метки

Основными структурными элементами в базе данных Neo4j являются: Узлы, отношения, метки и свойства.

Узлы (nodes) – используются для представления сущностей, но в зависимости от отношений в графе могут быть использованы для представления связи. Самый простой граф представляет собой одну вершину. Вершина может не иметь значить, либо иметь одно или более именованных значений, которые указываются в виде свойств.

Отношения – это ассоциативные связи между узлами. В модели данных графа свойств отношения должны быть направленными.

Свойства (prosperties) – именованные значения, где имя – это строка. Поддерживаемые значения: числовые, строковые, двоичные, списки предыдущих типов.

Метки (labels) – предоставляют собой графы, которые были сгруппированы в наборы. Все узлы, помеченные одной меткой, принадлежит к одному набору. Упрощают написание запросов к базе. Вершина может быть помечена любым 20 количеством меток. Метки используются для задания ограничений и добавления индексов для свойств.

Типы данных

СУБД Neo4j позволяет для определения свойств использовать следующие типы данных:

- Число, абстрактный тип, содержит подтипы **Integer** и **Float**;
- Строковый тип данных – **String**;
- Логический тип данных – **Boolean**, принимает значения true и false;
- Пространственный тип — **Point** (точка);
- Временные типы: **Date**, **Time**, **LocalTime**, **DateTime**, **LocalDateTime** и **Duration**;
- Структурные типы: **узлы**, **отношения**, **пути** (последовательность узлов и отношений);

- Составные типы: **Lists** (коллекции, каждый элемент которых имеет какой-то определенный тип) и **Maps** (коллекции вида (key: value), где key имеет тип String, а value любого типа).

На Рисунок 2 можно увидеть 7 узлов, помеченных меткой Employee. Свойствами выделенного узла являются id, lastname, firstname, patronymic и table_number.

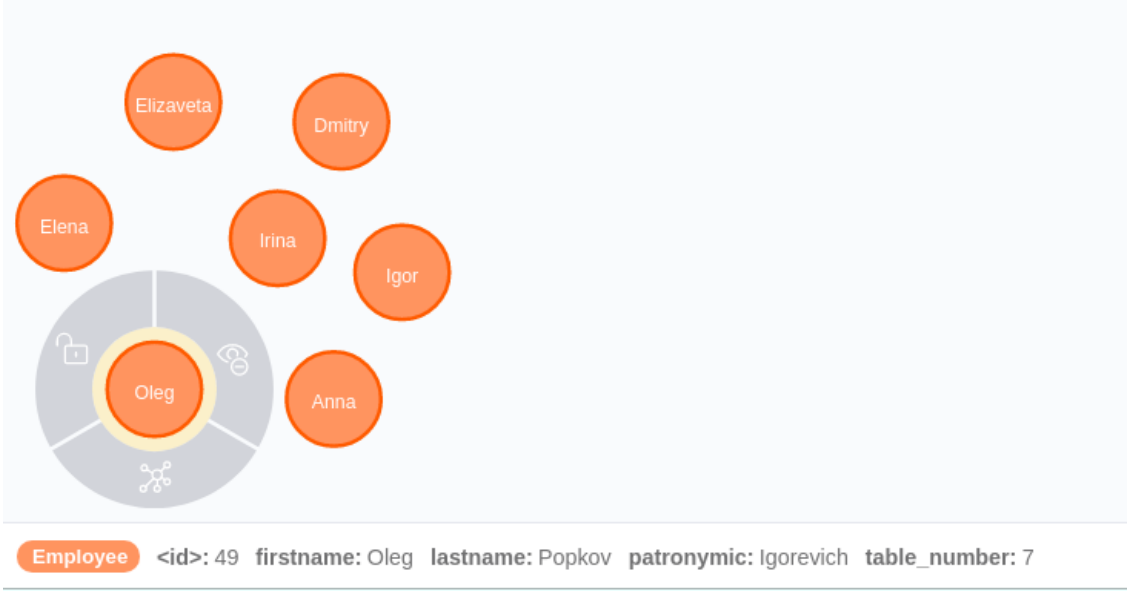


Рисунок 2 – Пример узлов и их свойств в Neo4j Data Browser

Основные команды языка Cypher

В СУБД Neo4j используется собственный язык запросов — Cypher, но запросы можно делать и другими способами, например, напрямую через Java API. Cypher является не только языком запросов, но и языком манипулирования данными, так как предоставляет функции CRUD для графового хранилища.

Cypher – это декларативный язык, основанный на SQL, для визуального описания графиков [4.], в которых используется синтаксис ASCII-Art (англ. ASCII-Графика). Такая реализация позволяет нам указать, что мы хотим выбрать, вставить, обновить или удалить из наших графических данных, не требуя от нас точно описать, как это сделать. Система типа Cypher подробно описана в Cypher Improvement Proposal (CIP) [5.], и содержит следующие типы: узлы, отношения, пути, карты, списки, целые числа, числа с плавающей запятой, логические значения и строки.

Для работы с СУБД Neo4j используется язык **Cypher**. Язык **Cypher** использует следующие команды приведенные в Таблица 1.

Таблица 1 – Основные команды Cypher

Название	Описание
MATCH	Указание шаблона для поиска.
WHERE	Указание условия поиска.
START	Задание начальных точек поиска.
LOAD CSV	Импорт данных из CSV файла.
CREATE	Создание узлов, отношений и свойств.
MERGE	Создание указанного шаблона в графе, если такового не существует.
SET	Обновление меток на узлах, свойств на узлах и отношений.
DELETE	Удаление узлов, отношений или путей из графа.
REMOVE	Удаление свойств и элементов из узлов и отношений.
FOREACH	Используется для обновления данных в списке
CREATE UNIQUE	Используя предложения CREATE и MATCH, вы можете получить уникальный шаблон, сопоставив существующий шаблон и создав недостающий.
RETURN	Определение того, что надо включить в набор результатов запроса.
ORDER BY	Используется для упорядочения вывода запроса по порядку. Используется вместе с RETURN или WITH.
LIMIT	Используется для ограничения строк в результате определенным значением.
SKIP	Пропуск начальных записей результата.
WITH	Объединение частей запроса.
UNION	Объединение результатов нескольких запросов.
CALL	Вызов процедуры.

Рассмотрим основные особенности СУБД Neo4j.

Доступность

Начиная с версии Neo4J 1.8 высокая доступность базы обеспечивается реплицированными ведомыми узлами [6.]. Кроме того, эти ведомые узлы могут выполнять операции записи: свои записи они синхронизируют с текущим ведущим узлом и сначала закрепляют их на ведущем узле, а потом на ведомом. Это обновление в конце концов получают все ведомые узлы. База Neo4J использует программу Apache ZooKeeper для отслеживания идентификаторов последней транзакции на каждом ведомом и текущем ведущем узле. При загрузке сервер связывается с программой ZooKeeper и выясняет, какой узел является ведущим. Если сервер первым присоединяется к кластеру, он становится ведущим узлом; если ведущий узел выходит из строя, то кластер выбирает ведущий узел среди доступных узлов, обеспечивая высокую доступность.

База Neo4J также поддерживает язык запросов Cypher для обхода графа. Помимо этих языков запросов, база Neo4J позволяет запрашивать свойства узлов, обходить граф и перемещаться по отношениям с помощью языковых привязок.

Согласованность данных

В рамках отдельного сервера данные всегда являются согласованными, особенно в базе Neo4J, которая полностью поддерживает транзакции ACID [6.]. Если база Neo4J работает на кластере, запись на ведущий узел синхронизируется с ведомыми узлами, которые всегда доступны для чтения. Операции записи на ведомые узлы всегда доступны и немедленно синхронизируются с ведущим узлом; остальные ведомые узлы не синхронизируются немедленно – они ждут, пока данные не будут распределены с ведущего узла.

Транзакции

База Neo4J поддерживает транзакции ACID [6.]. Прежде чем изменить какой-нибудь узел или добавить какое-то отношение к существующим узлам, необходимо начать транзакцию. Если не упаковать операции в транзакции, то получим исключение **NotinTransactionException**. Операции чтения можно выполнять без создания транзакций.

```
Transaction transaction = database.beginTx();
try {
```

```
        Node node = database.createNode();
        node.setProperty("name", "NoSQL Distilled");
        node.setProperty("published", "2012");
        transaction.success();
} finally {
        transaction.finish();
}
```

В этом коде началась транзакция над базой данных, затем создается узел и его свойства. Транзакция помечена функцией success и закончена функцией finish. Транзакция должна быть помечена функцией success, иначе база Neo4J будет считать, что произошла ошибка, и выполнит откат при вызове функции finish. Вызов функции success без вызова функции finish также не закрепит данные в базе данных. При разработке приложения следует помнить об этой особенности транзакций, которая отличается от транзакций в базе данных RDBMS.

Масштабирование

При масштабировании баз данных NoSQL широко используется фрагментация, в ходе которой данные разделяются и распределяются по разным серверам [6.]. В графовых базах данных фрагментацию сделать трудно, поскольку они ориентированы не на агрегаты, а на отношения. Поскольку любой узел может быть связан отношением с любым другим узлом, хранение связанных узлов на одном и том же сервере позволяет повысить эффективность обхода графа. Обход графа, узлы которого разбросаны по разным компьютерам, имеет низкую эффективность. Зная об этом ограничении графовых баз данных, мы все же можем масштабировать их с помощью общепринятых методов, описанных Джимом Уэббером [Webber Neo4J Scaling].

В принципе существуют три способа масштабирования графовых баз данных. Поскольку в настоящее время компьютеры могут иметь большой объем оперативной памяти, можно добавить на сервер столько микросхем, чтобы работать с множеством узлов и отношений, находящимся целиком в оперативной памяти. Этот метод оказывается полезным только в тех случаях, когда множество данных, с которым мы работаем, действительно может поместиться в оперативной памяти.

Можно улучшить масштабирование чтения, добавив дополнительные ведомые узлы, обеспечивающие только чтение данных, а все операции записи выполнять на ведущем узле. Такой способ, предусматривающий однократную запись данных и их чтение со многих серверов, хорошо зарекомендовал себя на кластерах MySQL и оказался действительно полезным при работе с наборами данных, которые достаточно велики, чтобы не помещаться в оперативной памяти одного компьютера, но достаточно малы, чтобы создать их реплики на разных компьютерах. Ведомые узлы повышают доступность и облегчают масштабирование чтения, поскольку их можно сконфигурировать так, чтобы они никогда не становились ведущими и выполняли только операции чтения. Если набор данных настолько велик, что создавать его реплики нецелесообразно, можно выполнить фрагментацию данных на стороне приложения, используя знания о предметной области. Например, узлы, связанные с Северной Америкой, можно создать на одном сервере, а узлы, связанные с Азией, – на другом. Выполняя такую фрагментацию на стороне приложения, следует понимать, что узлы хранятся в физически разных базах данных (см. Рисунок 3).

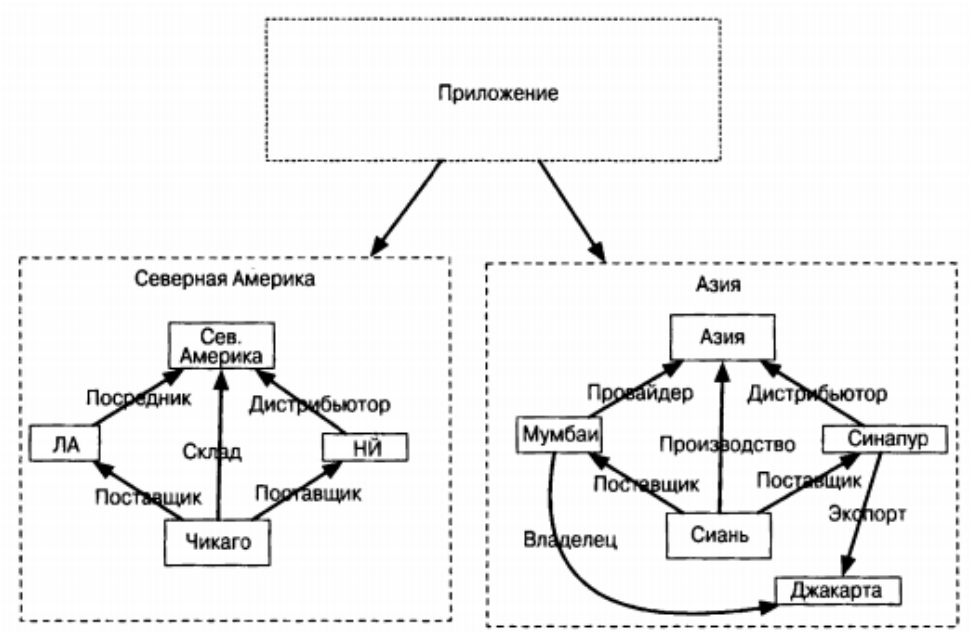


Рисунок 3 – Фрагментация узлов на уровне приложения

Запуск и установка Neo4j

Разберем каким образом произвести установку СУБД Neo4j.

Системные требования, требуемые для качественной работы Neo4j:

1. Для работы СУБД neo4j необходимо минимум 2Gb оперативной памяти, а для стабильной работы рекомендуется 16Gb.
2. В качестве дискового массива рекомендуется использовать SSD-диски.
3. Neo4j написана на Java, поэтому необходима установленная JVM8. О том, как это сделать написано в нашей инструкции (/help/linux/ustanovka-java-na-ubuntu).

Инструкция по установке для macOS подробно описана в материале по URL-ссылке [7.].

Для ОС Windows, Ubuntu или Debian инструкция по установке и запуску представлена в следующих подразделах.

Настройка СУБД Neo4j с помощью exe-файла Windows

В этой главе мы обсудим, как установить Neo4j в вашу систему, используя exe-файл.

Следуйте инструкциям ниже, чтобы загрузить Neo4j в вашу систему.

Шаг 1 – Посетите официальный сайт Neo4j

Перейдите по ссылке https://neo4j.com/ [1.]. В результате вы попадете на домашнюю страницу сайта Neo4j (см. Рисунок 4).

Шаг 2 – Загрузка exe-файла

Необходимо загрузить exe-файл Neo4j, используя кнопку «Загрузить» справа сверху на домашней странице Neo4j (см. Рисунок 4).

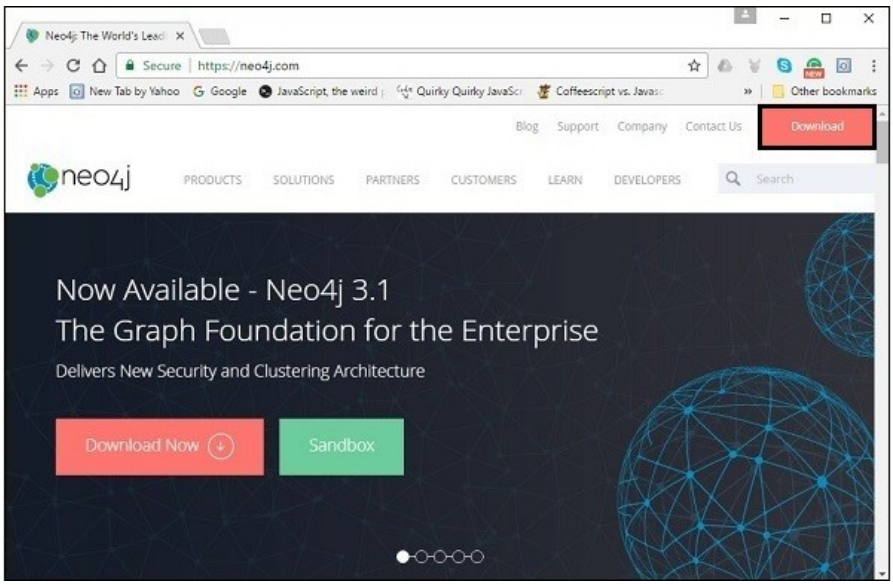


Рисунок 4 – Домашняя страница сайта Neo4j

Шаг 3 – Выбор требуемой версии программного обеспечения

После выполнения Шага 3 откроется страница загрузок, где вы можете скачать версию для сообщества и корпоративную версию Neo4j. Загрузите версию программного обеспечения для сообщества, нажав соответствующую кнопку (см. Рисунок 5).

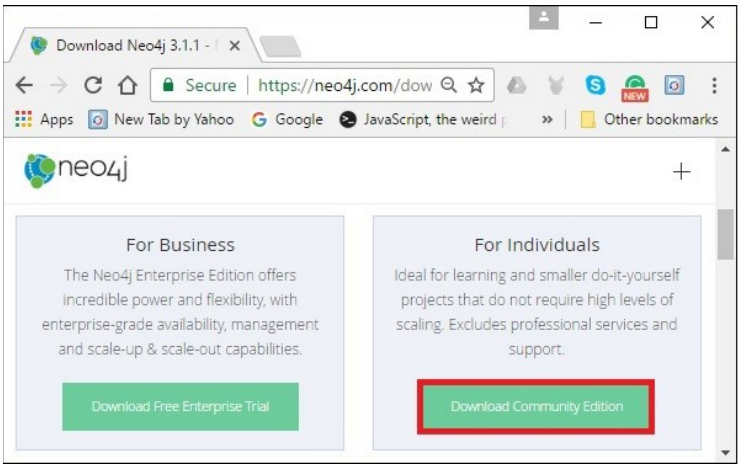


Рисунок 5 – Страница загрузок сайта Neo4j

Шаг 4 – Загрузка Neo4j для Windows

Вы перейдете на страницу, где вы можете скачать версию программного обеспечения Neo4j для сообщества, совместимую с различными операционными системами. Загрузите файл, соответствующий желаемой операционной системе (см. Рисунок 6).

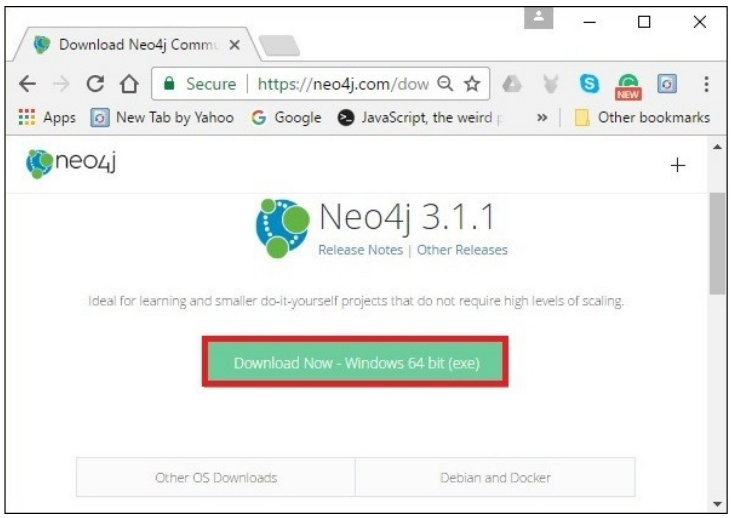


Рисунок 6 – Страница загрузок Neo4j для Windows

Это позволит загрузить файл с именем **neo4j-community_windows-x64_3_1_1.exe** в вашу систему, как показано на следующем снимке экрана (см. Рисунок 7).

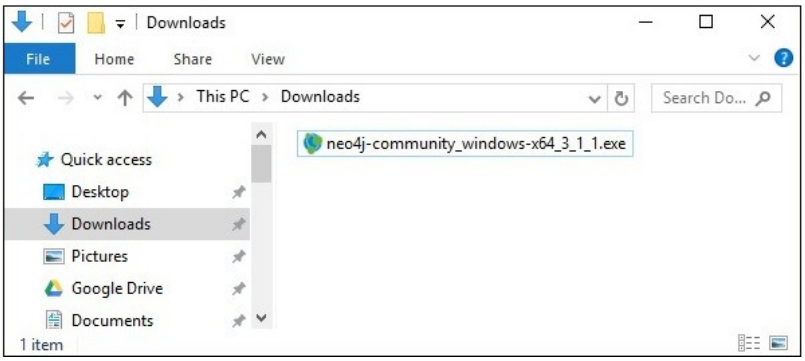


Рисунок 7 – Загруженный exe-файл Neo4j

Шаг 5 – Запуск exe-файла Neo4j

Дважды щелкните exe-файл, чтобы установить Neo4j Server (см. Рисунок 8).

Шаг 6 – Завершение установки Neo4j Server

Примите лицензионное соглашение и продолжите установку.

После завершения процесса вы можете заметить, что Neo4j установлен в вашей системе.

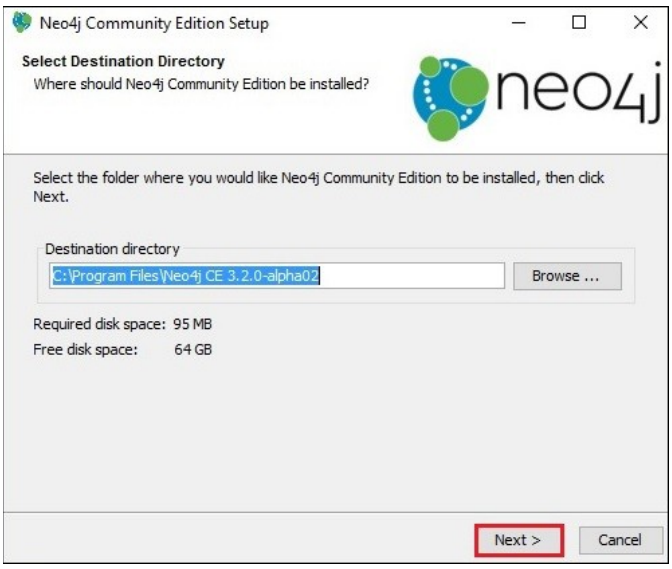


Рисунок 8 – Установка Neo4j Server

Шаг 7 – Запуск Neo4j Community Edition

Щелкните меню запуска Windows и запустите сервер Neo4j, щелкнув ярлык меню «Пуск» для Neo4j (см. Рисунок 9).

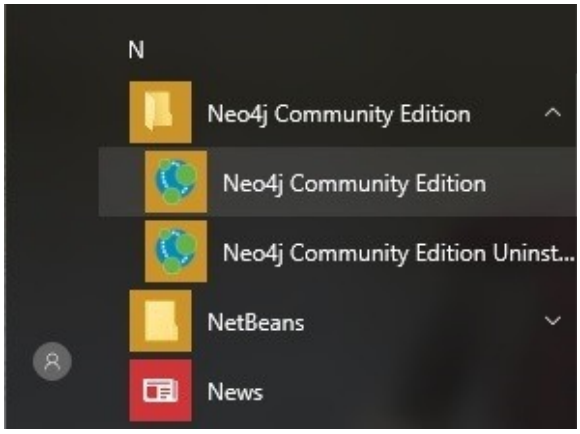


Рисунок 9 – Выбор Neo4j Server из меню Пуск

Шаг 8 – Выбор пути размещения Базы данных

Нажав на ярлык **Neo4j Community Edition**, вы получите окно для Neo4j Community Edition (см. Рисунок 10). По умолчанию он выбирает путь C:\Users\[имя пользователя]\ Documents\Neo4j\ default.graphdb. Если вы хотите, вы можете изменить свой путь к другому каталогу.

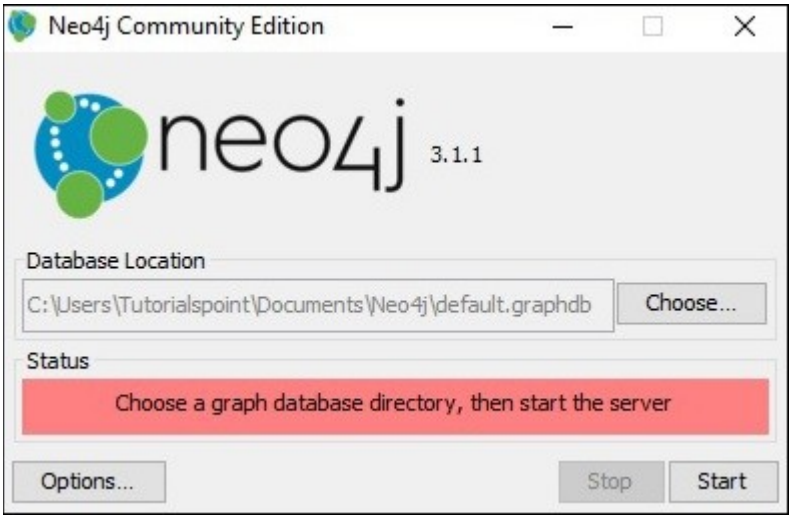


Рисунок 10 – Выбор пути для размещения базы данных Neo4j

Шаг 9 – Запуск Neo4j сервера с базой данных

Нажмите кнопку «Пуск», чтобы запустить сервер Neo4j (см. Рисунок 11).

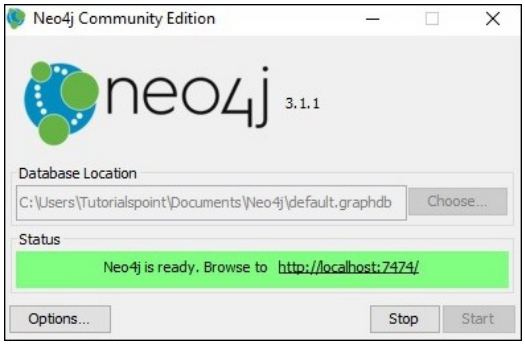


Рисунок 11 – Запуск Neo4j Server

После запуска сервера вы можете заметить, что каталог базы данных заполнен, как показано на следующем снимке экрана (см. Рисунок 12).

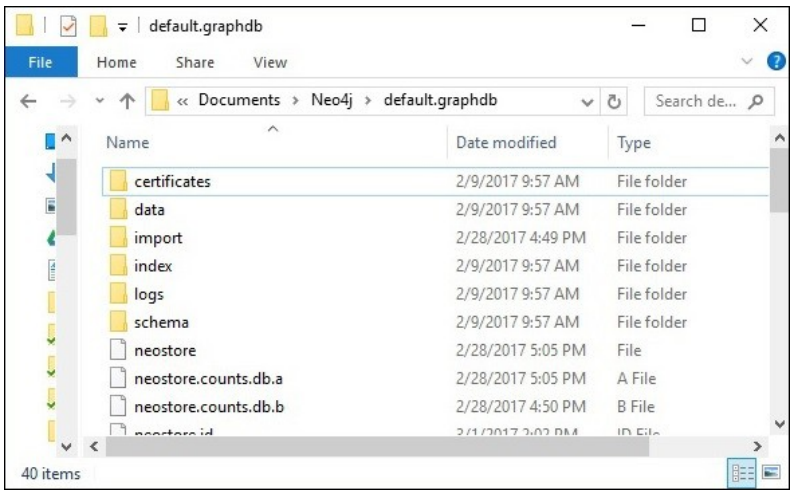


Рисунок 12 – Каталог базы данных

Шаг 10 – Работа с Neo4j

Как обсуждалось в предыдущих главах, СУБД Neo4j предоставляет встроенное приложение для работы с Базой данных.

Вы можете получить доступ к Neo4j Server, используя URL-адрес http://localhost:7474/ .

Для подключения к серверу потребуется ввести логин и пароль, заданные по умолчанию:

Логин: neo4j

Пароль: neo4j

Вы должны увидеть интерфейс Neo4j Browser, представленный на Рисунок 13.

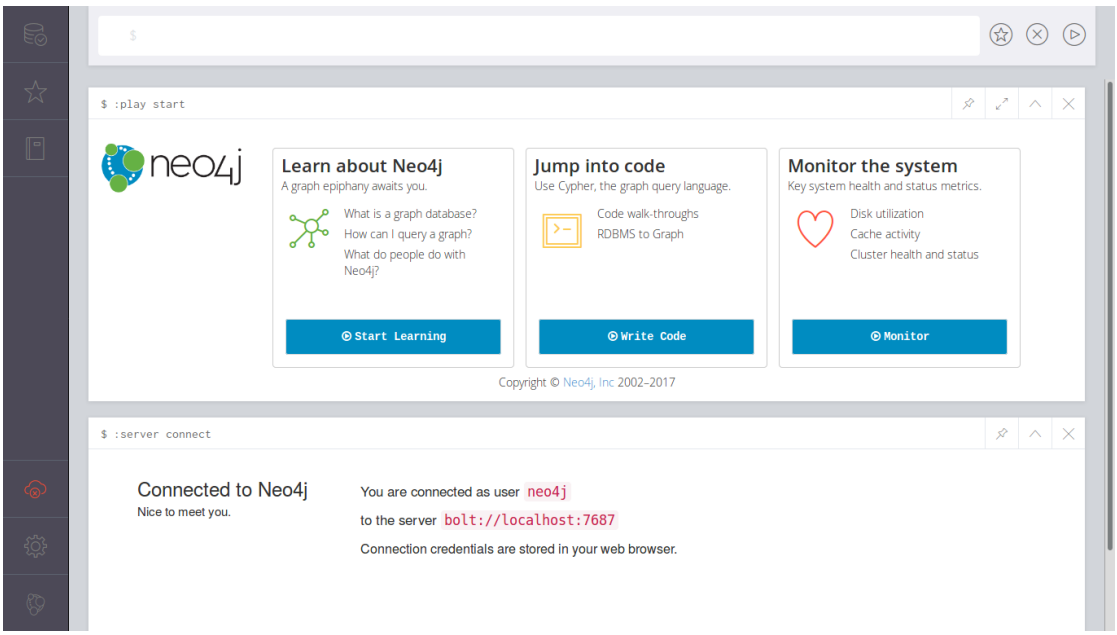


Рисунок 13 – Интерфейс Neo4j Browser для Windows

Установка и настройка Neo4j на сервере с Ubuntu или Debian

В данном разделе описан процесс установки и настройки графовой системы управления базами данных Neo4j на виртуальном сервере с Ubuntu или Debian и подключение к web-интерфейсу.

Шаг 1 – Скачивание GPG-ключа

Скачайте зашифрованный GPG-ключ и добавьте репозиторий в локальный список:

```
wget --no-check-certificate -O -
https://debian.neo4j.org/neotechnology.gpg.key | sudo apt-key add -
echo 'deb http://debian.neo4j.org/repo stable/' >
/etc/apt/sources.list.d/neo4j.list
```

Шаг 2 – Установка СУБД Neo4j

Обновите локальную базу пакетов и установите СУБД Neo4j:

```
apt update apt install neo4j
```

Шаг 3 – Подключение к базе данных

Для того, чтобы подключиться к базе данных с любого IP-адреса, а не только на локальном хосте, в конфигурационном файле необходимо изменить некоторые строки. С помощью текстового редактора откройте файл настроек:

```
vi /etc/neo4j/neo4j.conf
```

Найдите следующие строки, они находятся в разных частях файла:

```
#dbms.connector.http.listen_address=:7474
#dbms.connector.bolt.listen_address=:7687
```

Раскомментируйте их и добавьте IP-адреса, как показано ниже:

```
dbms.connector.http.listen_address=0.0.0.0:7474
dbms.connector.bolt.listen_address=0.0.0.0:7687
```

После сохранения изменений перезапустите neo4j:

```
service neo4j restart
```

Не забудьте настроить firewall для удаленного доступа:

```
iptables -A INPUT -p tcp --dport 7474 -j ACCEPT
iptables -A INPUT -p tcp --dport 7687 -j ACCEPT
```

Шаг 4 – Подключение в браузере

Откройте браузер и подключитесь к СУБД по следующему адресу:

```
<ваш_IP-адрес>:7474
```

Например:

```
111.111.111.111:7474
```

При первом подключении замените значение localhost на ваш IP-адрес для протокола bolt, а в качестве логина и пароля используйте название СУБД (см. Рисунок 14):

```
neo4j
```

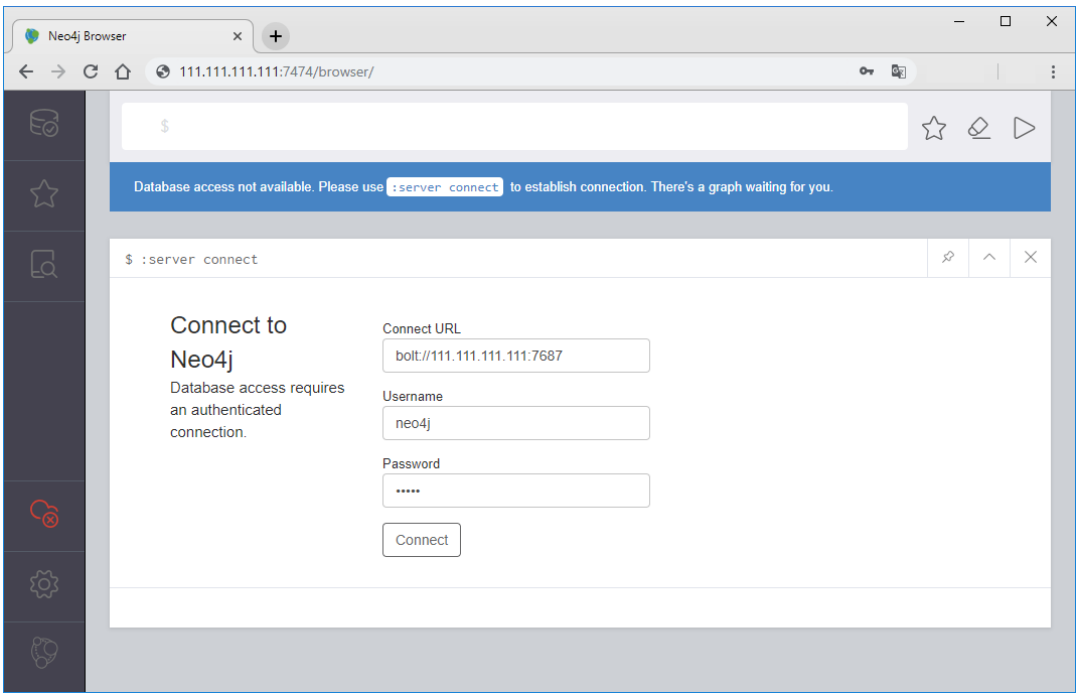


Рисунок 14 – Подключение к Базе данных в браузере

Примечание: сетевой протокол bolt – это высокоэффективный и легкий клиент-серверный протокол, предназначенный для приложений баз данных.

Шаг 5 – Авторизация пользователя Neo4j

Далее укажите и подтвердите новый пароль для пользователя Neo4j (см. Рисунок 15).

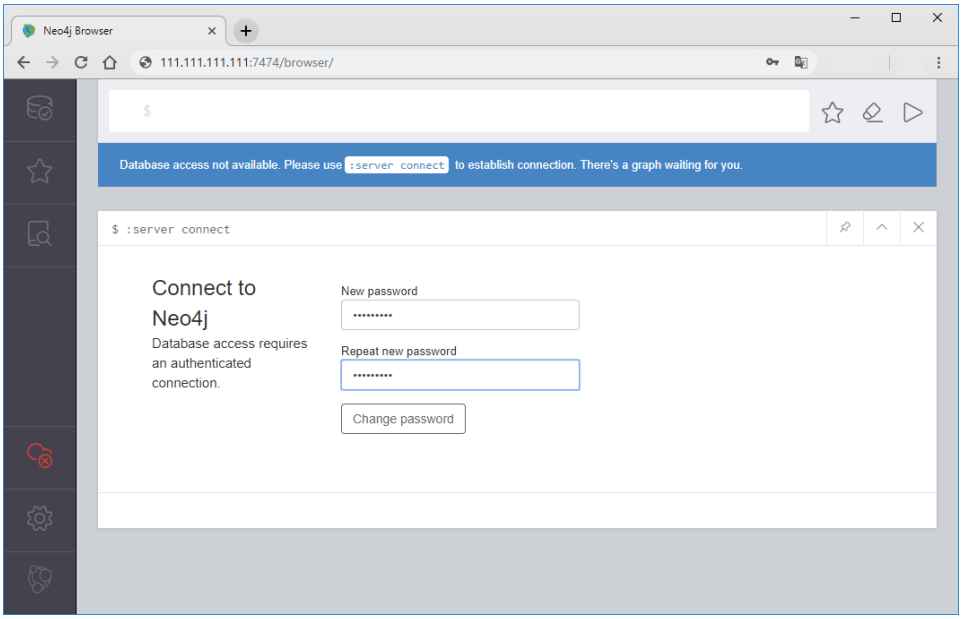


Рисунок 15 – Обновление пароля для базы данных

Теперь вы можете приступить к работе в графовой neo4j (см. Рисунок 16).

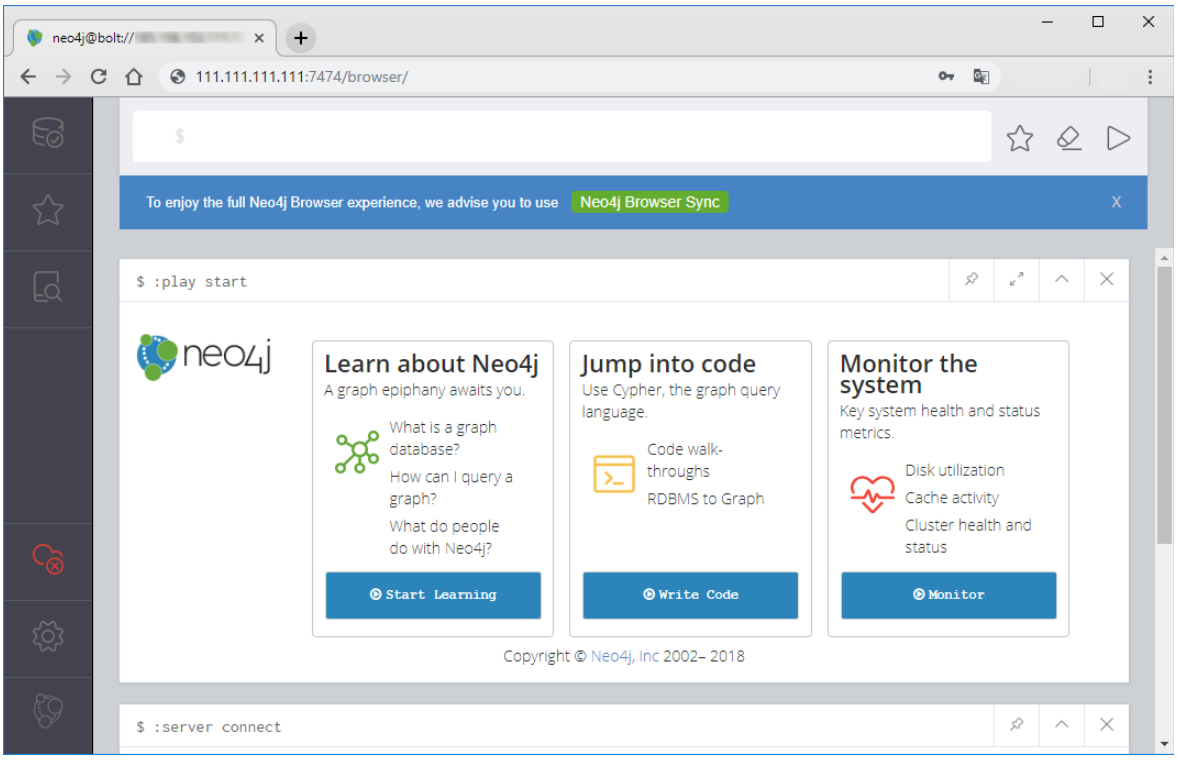


Рисунок 16 – Интерфейс Neo4j в браузере для Ubuntu или Debian

Возможности среды Neo4j Data Browser

Neo4j Data Browser используется для выполнения команд CQL и просмотра выходных данных (см. Рисунок 17). Для работы с Neo4j Data Browser используются следующие кнопки:

- **«$»** – В данном строке описываются все команды CQL, которые необходимо исполнить.
- **«Execute Command»** (англ. Выполнить) – Введите команды после символа доллара и нажмите кнопку для запуска команд. Neo4j Data Browser взаимодействует с сервером базы данных Neo4j, извлекает и отображает результаты прямо под подсказкой доллара.
- **«UI View»** – Используется для просмотра результатов в формате диаграмм.
- **«Grid View»** – Используется, чтобы просмотреть результаты в режиме сетки.
- **«*»** (звездочка) – Используется, чтобы сохранить запрос, добавив его в избранное.

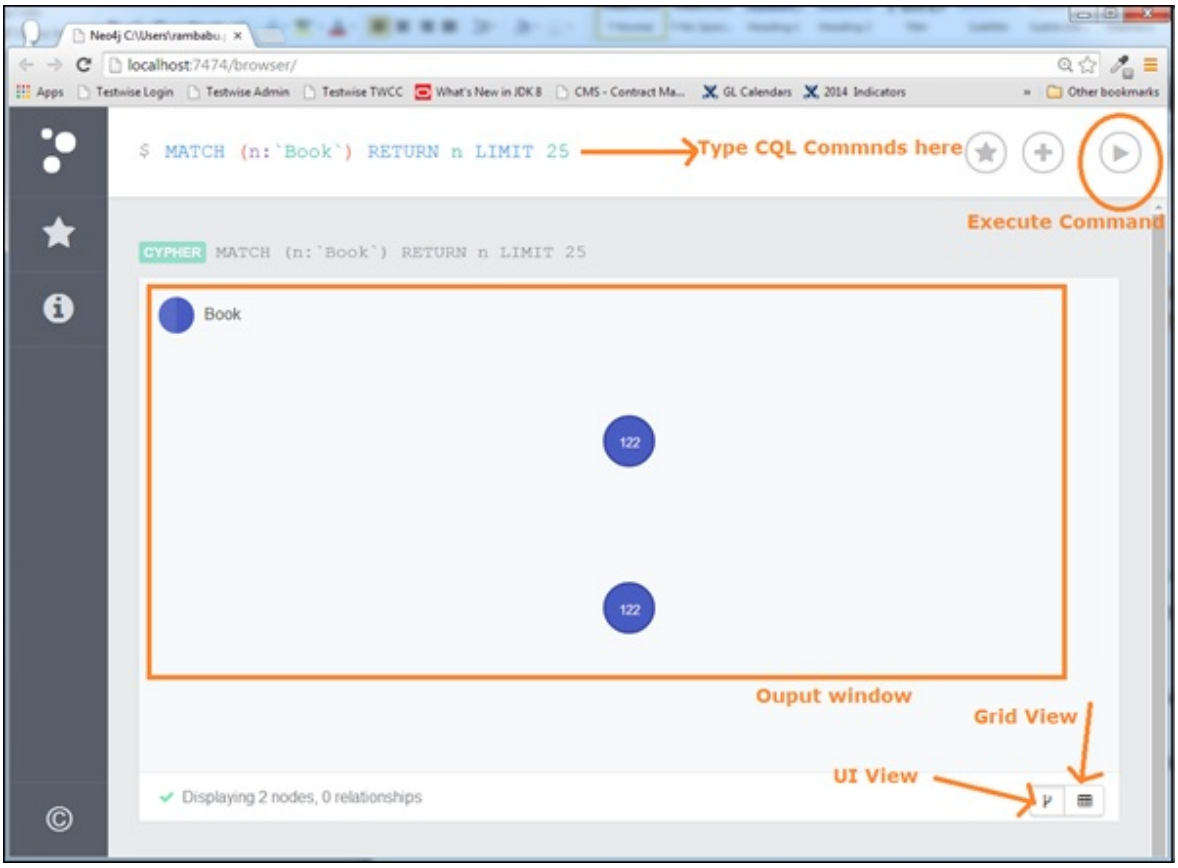


Рисунок 17 – Функциональные области Neo4j Data Browser

Когда мы используем «Вид сетки» для просмотра результатов нашего запроса, мы можем экспортировать их в файл в двух разных форматах (см. Рисунок 18).

При необходимости можно экспортировать полученную базу данных в требуемом формате (см. Рисунок 18):

- Нажмите кнопку «Экспорт CSV», чтобы экспортировать результаты в формате файла CSV.
- Нажмите кнопку «Экспорт JSON», чтобы экспортировать результаты в формате файла JSON.

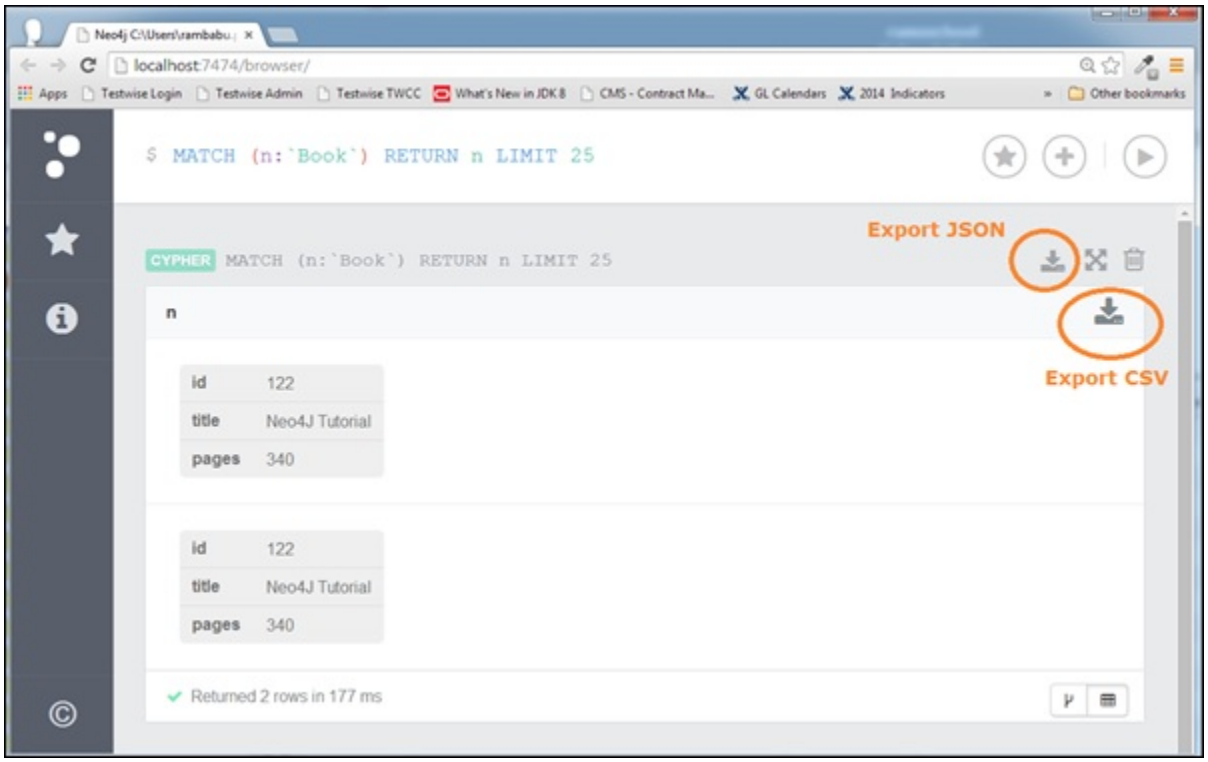


Рисунок 18 – Возможности экспорта файлов в Neo4j Data Browser

Однако, если мы используем «UI View» для просмотра результатов нашего запроса, мы можем экспортировать их в файл только в одном формате: JSON.

После выполнения запроса его обычно можно посмотреть в нескольких вариантах: в виде графа, таблицы, текста и кода. Необходимо кликнуть на необходимый вариант представления в левой части окна с результатом (см. Рисунок 19).

Информацию о базе данных можно просмотреть, кликнув на знак базы данных в левом верхнем углу экрана. Там же можно просмотреть закладки и документацию. В

левом нижнем углу можно настроить облачные сервисы, установить настройки для браузера и прочитать справочную информацию о Neo4j (см. Рисунок 20).

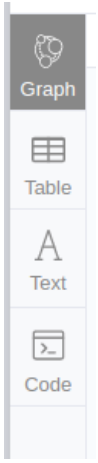


Рисунок 19 – Виды просмотра результата выполненного запроса

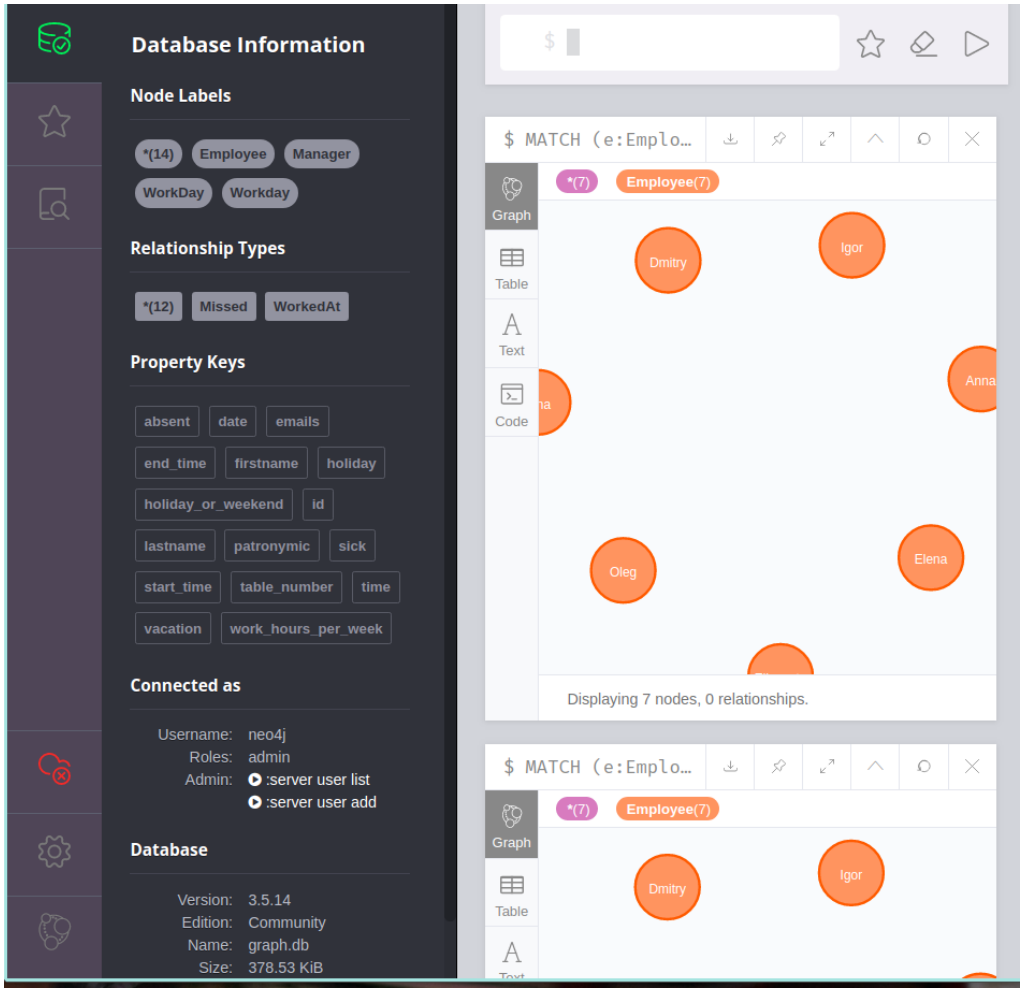


Рисунок 20 – Просмотр информации о созданной Базе данных

Команды определения структуры данных

Создание узла со свойствами

Для создания узла со свойствами используется ключевое слово **CREATE**. Чтобы создать узел со свойствами необходимо выполнить следующий запрос:

```
CREATE (node:label { key1: value1,
                     key2: value2,
                     . . . . . . . . . . })
```

где *node* – узел,

label – метка,

key1, *key2* — название свойства,

value1, *value2* — значение свойства.

Примечание. Обращение к меткам в Neo4j идет через символ двоеточия.

Пример:

Создание узла с меткой Employee (сотрудник) и свойствами: табельный номер, Фамилия, Имя и Отчество (см. Рисунок 21).

```
CREATE (e: Employee { table_number: 7,
                      lastname: 'Popkov',
                      firstname: 'Oleg',
                      patronymic: 'Igorevich'})
```

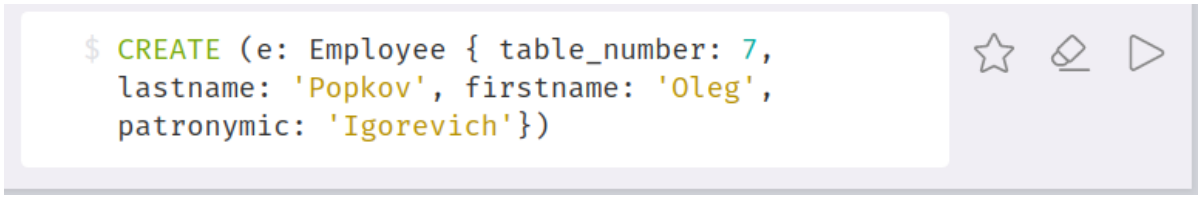


Рисунок 21 – Создание узла с меткой Employee

Для запуска команды необходимо нажать знак треугольника справа поля. В случае успешного выполнения команды появится сообщение, изображенное на Рисунок 22.

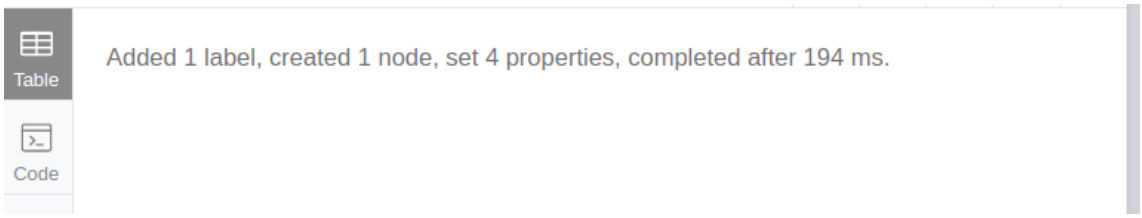


Рисунок 22 – Успешное выполнение создания узла

Подробнее смотрите в [8.].

Удаление и изменение свойств и меток

Изменение свойства узла

Для изменения свойства используется ключевое слово **SET**. Чтобы изменить свойство узла, необходимо выполнить следующий запрос:

```
MATCH (node {attribute1: 'value1'})
      SET node.attribute2='value2'
      RETURN node.attribute1, node.attribute2
```

где *node* – узел,

attribute1,2 – названия свойства узла,

value1, 2 – значения свойства узла.

Пример:

Запрос, устанавливающий количество часов, которые работает сотрудник (узел, у которого свойство табельный номер = 1) в неделю (см. Рисунок 23).

```
MATCH (e) WHERE e.table_number=1
      SET e.work_hours_per_week=35
      RETURN e.lastname, e.firstname, e.patronymic, e.work_hours_per_week
```

e.lastname	e.firstname	e.patronymic	e.work_hours_per_week
"Ivanov"	"Ivan"	"Ivanovich"	35

Рисунок 23 – Изменение свойства узла

Подробнее смотрите в [9.].

Удаление свойства узла

Для удаления свойства узла используется ключевое слово **REMOVE**. Чтобы удалить свойство узла, необходимо выполнить следующий запрос:

```
MATCH (node {attribute1: 'value1'})
      REMOVE node.attribute2
      RETURN node
```

либо, используя ключевое слово **SET**:

```
MATCH (node {attribute1: 'value1'})
      SET node.attribute2=NULL
      RETURN node
```

где *node* – узел,

attribute1, 2 – названия свойства узла,

value1, 2 – значения свойства узла.

Пример:

Запрос, который удалит свойство отчество у узла e с табельным номером 1 (см. Рисунок 24 и Рисунок 25).

```
MATCH (e) WHERE e.table_number=1 REMOVE e.patronymic RETURN e
```

```
{
  "table_number": 1,
  "firstname": "Ivan",
  "work_hours_per_week": 35,
  "lastname": "Ivanov"
}
```

Рисунок 24 – Удаление свойства узла

Рисунок 25 – Графическое изображение узла после удаления свойства

Подробнее смотрите в [10.].

Изменение метки

Для изменения метки необходимо выполнить следующий запрос:

```
MATCH (node: Label1 { attribute: 'value' })
      SET node:Label2
      RETURN node
```

где *node* – узел,

Label1, Label2 – старая и новая метка,

attribute – названия свойства узла,

value – значения свойства узла.

Пример:

Запрос, который удалит метку Employee и установит метку Manager для узла e, у которого свойство табельный номер=1 (см. Рисунок 26).

```
MATCH (e:Employee{table_number:1}) REMOVE e:Employee SET e:Manager RETURN e
```

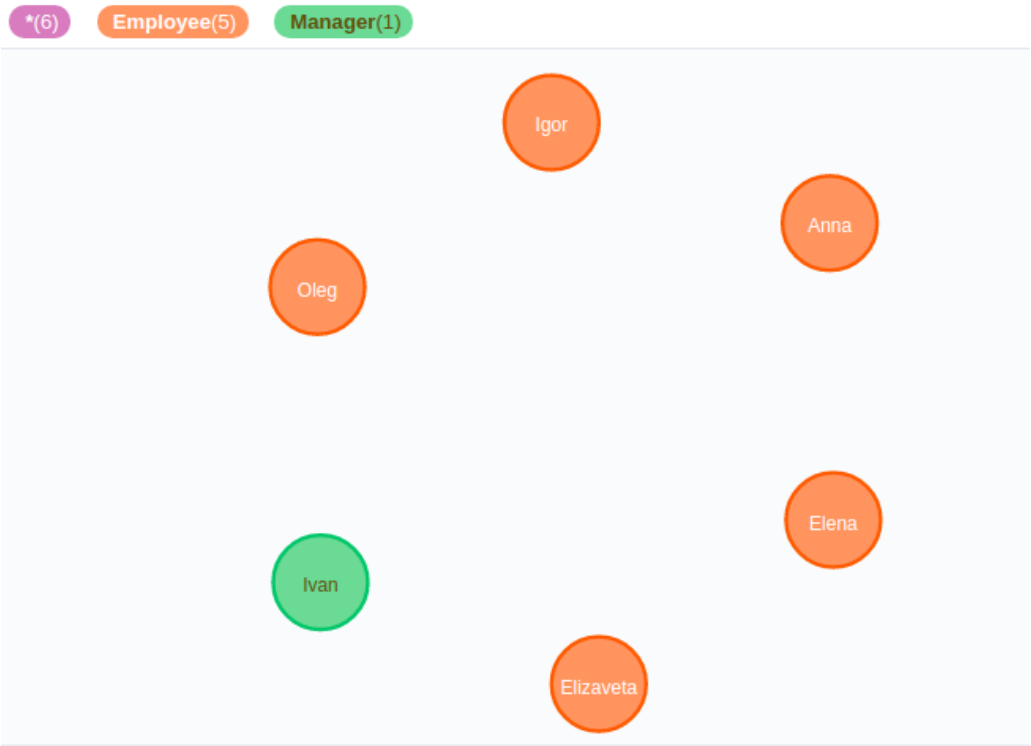


Рисунок 26 – Изменение метки узла

Удаление метки

Для удаления метки узла необходимо выполнить следующий запрос:

```
MATCH (node {attribute: 'value' })
      REMOVE node: Label
      RETURN node
```

где *node* – узел,

Label – метка,

attribute – названия свойства узла,

value – значения свойства узла.

Пример:

Запрос, удаляющий метку Менеджер у узла с меткой Manager с табельным номером 1 (см. Рисунок 27).

```
MATCH (e:Manager{table_number:1}) REMOVE e:Manager RETURN e
```

Рисунок 27 – Удаление метки узла

Отношения между узлами

Основными элементами, которыми оперирует язык Cypher, являются вершины (ноды) и рёбра графа. Рёбра в Neo4j имеют тип и направление, вершины же могут быть помечены одной или более метками, а также могут иметь несколько дополнительных свойств. Для описания шаблонов извлекаемой информации в Cypher используется следующее:

- Вершины записываются в виде пары круглых скобок, внутри которых задаются дополнительные условия, будь то тип выршины или значение какого-либо свойства, например (a:Actor) - вершина типа Actor.

- Рёбра записываются в виде пары квадратных скобок, по аналогии с вершинами внутри скобок задаются доп условия. [e:ACTED] - связь типа ACTED.

Чтобы полностью использовать возможности базы данных графов, необходимо выразить более сложные шаблоны между узлами. Отношения обозначаются как стрелка " -> " между двумя узлами. Дополнительная информация может быть помещена в квадратные скобки внутри стрелки. Такой информацией может быть:

- тип отношений -[:KNOWS|:LIKE]->
- имя переменной -[rel:KNOWS]-> перед двоеточием
- дополнительные свойства -[{since:2010}]->
- структурная информация для путей переменной длины -[:KNOWS*..4]->

Задание отношений:

- Простейший способ описания связи – использование стрелки между двумя узлами. С помощью этой техники можно описать, что связь должна существовать и её направление (a)->(b) или (a)--(b), если не важно направление связи.
- Отношениям также могут быть даны имена. В этом случае используется пара квадратных скобок с идентификатором между ними, которые разбивают стрелку надвое. Например: (a)-[r]->(b)
- Подобно меткам узлов, отношения могут иметь типы. Для описания связи заданного типа можно поступить следующим образом: (a)-[r:REL_TYPE]->(b)
- Связи могут иметь только один тип, но можно задать тип так, чтобы связь имела один тип из набора с помощью разделяющего символа «|»: (a)-[r:TYPE1|TYPE2]->(b)
- Имя связи всегда может быть опущено: (a)-[:REL_TYPE]→(b)

Узлы и выражения отношений являются строительными блоками для более сложных шаблонов. Шаблоны можно писать непрерывно или разделять запятыми. Вы можете ссылаться на ранее объявленные переменные или вводить новые.

Например: user знает друга, который знает еще кого-то

```
(user)-[:KNOWS]-(friend)-[:KNOWS]-(someone)
```

Шаблоны могут использоваться для сравнения и создания данных, но также (для оценки списка путей) в выражениях, предикатах и результатах.

Чтобы создать шаблон отношений, в котором не важна направленность, необходимо использовать следующий запрос:

```
(a)--(b)
```

Чтобы создать шаблон отношений, в котором важна направленность, необходимо использовать следующий запрос:

```
(a)-[r]->(b)
```

Чтобы создать шаблон отношений для указания какого-то конкретного типа отношения необходимо использовать следующий запрос:

```
(a)-[r:REL_TYPE]->(b)
```

где *a, b* – узлы,

r – отношение,

REL_TYPE – метка отношения.

Пример:

Запрос, который выведет все узлы с метками Employee и Workday, которые связаны какой-либо связью направленной от Employee к Workday (см. Рисунок 28).

```
MATCH (e:Employee)-[*]->(w:Workday) RETURN e,w
```

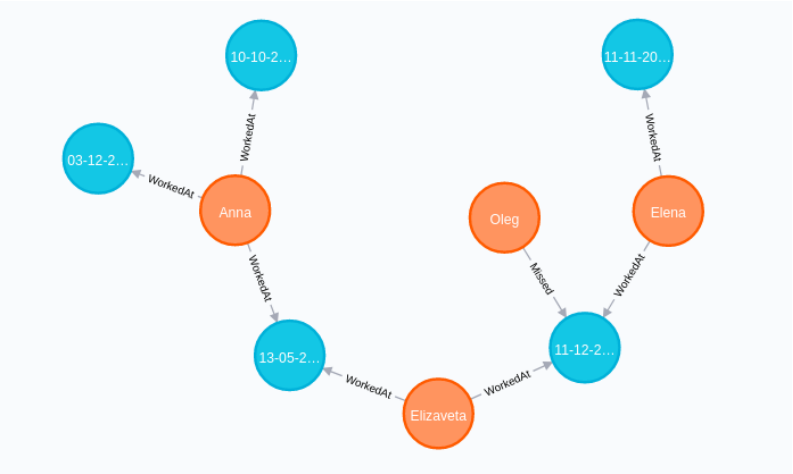


Рисунок 28 – Все узлы, связанные какой-либо связью

Подробнее смотрите в [11.].

Шаблоны отношений переменной длины

Вместо описания длинного пути с использованием последовательности из множества описаний узлов и отношений в шаблоне, многие отношения (и промежуточные узлы) можно описать, указав длину в описании отношений шаблона. Например:

```
(a)-[*2]->(b)
```

Это описывает график из трех узлов и двух отношений, все в одном контуре (путь длиной 2). Это эквивалентно:

```
(a)-->()-->(b)
```

Также может быть указан диапазон длин: такие паттерны отношений называются «отношениями переменной длины». Например:

```
(a)-[*3..5]->(b)
```

Это минимальная длина 3 и максимум 5. Он описывает график либо 4 узлов и 3 отношений, 5 узлов и 4 отношений, либо 6 узлов и 5 отношений, все они связаны вместе в одном пути.

Любая из них может быть опущена. Например, для описания путей длиной 3 и более используйте:

```
(a)-[*3..]->(b)
```

Чтобы описать контуры длиной 5 или меньше, используйте:

```
(a)-[*..5]->(b)
```

Пропуск обеих границ эквивалентен указанию минимум 1, что позволяет описывать пути любой положительной длины:

```
(a)-[*]->(b)
```

Пример:

В качестве простого примера возьмем граф (см. Рисунок 29) и запрос ниже:

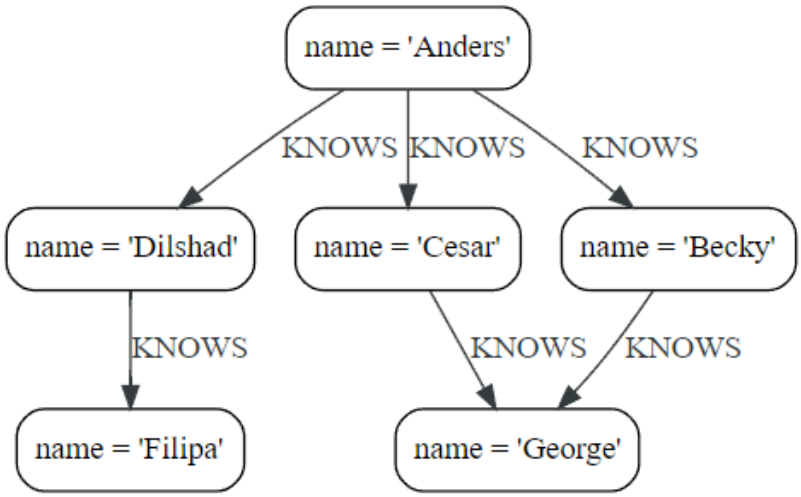


Рисунок 29 – Пример графа для запроса переменной длины

```
MATCH (me)-[:KNOWS*1..2]-(remote_friend)
      WHERE me.name = 'Filipa'
      RETURN remote_friend.name
```

После выполнения запроса можно увидеть следующий результат:

remote_friend.имя
"Dilshad"
"Anders"

Этот запрос находит данные в графе с формой, которая соответствует шаблону: в частности, узел (со свойством name 'Filipa'), а затем связанные узлы, один или два прыжка. Это типичный пример поиска друзей первой и второй степени KNOWS

Обратите внимание, что отношения переменной длины нельзя использовать с CREATE и MERGE

Как описано выше, ряд связанных узлов и связей называется «путем». Cypher позволяет именовать пути с помощью identifer, например:

```
p = (a)-[*3..5]->(b)
```

Создание отношения между новыми узлами

Чтобы создать отношение между новыми узлами, необходимо выполнить следующий запрос:

```
CREATE (node1:Label1 {attribute1: 'value1',  ...})-[rel: RelationName
     {attribute2: 'value2', ...}]->(node2:Label2 {attribute3: 'value3',...})
```

где *node1, node2* – узлы,
Label1, Label2 — метки узлов,
rel – отношение,
RelationName – метка отношения,
attribute1...3 – названия свойств отношений и узлов,
value1...3 – значения свойств отношений и узлов.

Пример:

Создание узла с меткой Employee и свойствами табельный номер и ФИО (Popkov Oleg Igorevich), узла с меткой WorkDay со свойством дата (07-05-2019) и связи с меткой “Missed” (пропустил) с параметрами absent, sick, vacation (пропустил/на больничном/в отпуске) между ними. Вывод сотрудника, рабочего дня и связи между ними (см. Рисунок 30).

То есть данный сотрудник не присутствовал на работе 07-05-2019 по причине болезни.

```
CREATE (e: Employee { table_number: 7,
                      lastname: 'Popkov',
                      firstname: 'Oleg',
                      patronymic: 'Igorevich',
                      work_hours_per_week: 40})-
[worker: Missed {absent:'No',
                sick:'Yes',
                vacation:'No'}]->
(w: Workday {date: '07-05-2019',
             holiday_or_weekend: 'No'}) RETURN e, w
```

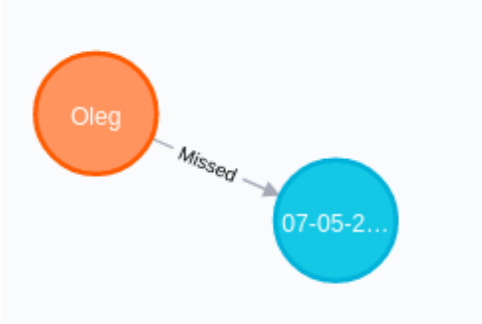


Рисунок 30 – Создание связи между новыми узлами

Создание отношений между несколькими узлами

Связь между существующими узлами можно создать, используя команды **MATCH** и **MERGE**.

Команда **MERGE** является комбинацией команды **CREATE** и команды **MATCH**. Команда **MERGE** ищет заданный шаблон в графе. Если он существует, он возвращает результаты. Если он НЕ существует в графе, то он создает новый узел / отношение и возвращает результаты.

Чтобы создать отношение между уже существующими узлами, необходимо выполнить следующий запрос:

```
MATCH (node1: Label1 {attribute: attribute_value}),
      (node2: Label2 {attribute: attribute_value})
      MERGE (node1)-[rel:RelationName
                              {rel_attribute:rel_attribute_value}]→(node2)
      RETURN node1, node2
```

где *node1*, *node2* – узлы,
Label1, *Label2* — метки узлов,
attribute — название свойства узла,
attribute_value — значение свойства узла,
rel_attribute — название свойства отношения,
rel_attribute_value — значение свойства отношения,
rel – отношение,
RelationName – метка отношения.

Пример 1:

Запрос, который проверит, существует ли связь с меткой Missed и свойствами absent, sick, vacation для узла e с меткой Employee с табельным номером 7 и узла w с меткой WorkDay и датой 11-12-2019, и создаст ее если не существует (см. Рисунок 31).

```
MATCH (e:Employee{table_number:7}),
      (w:Workday{date:'11-12-2019'})
      MERGE (e)-[worker: Missed{absent:'Yes', sick:'No', vacation:'No'}]→(w)
      RETURN e,w
```

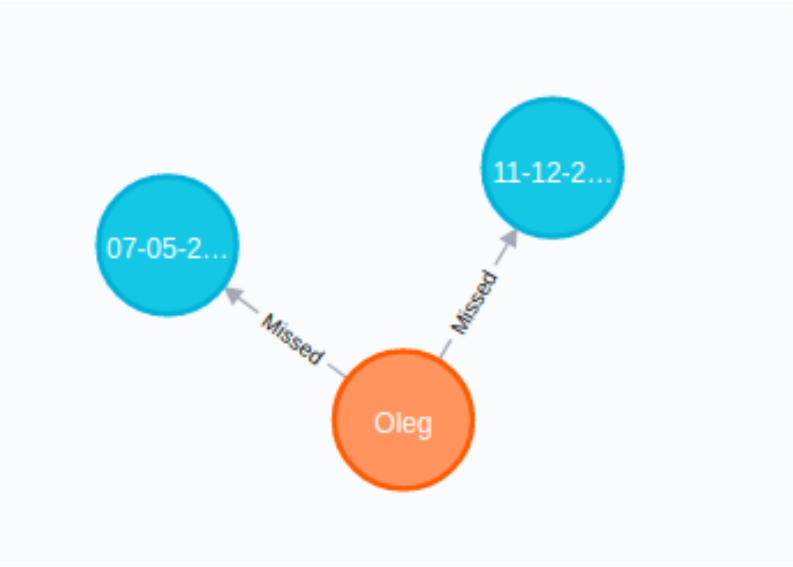


Рисунок 31 – Создание связи между существующими узлами

Пример 2:

Создание связи worked с меткой WorkedAt свойствами start_time, end_time между существующими узлами e с метками Employee (сотрудник) с фамилией Kotova и w – Workday (рабочий день) с датой 07-05-2019. Если в базе не существует сотрудника с заданной фамилией, либо заданного рабочего дня, то neo4j создаст новый узел. Направление связи – от сотрудника к рабочему дню.

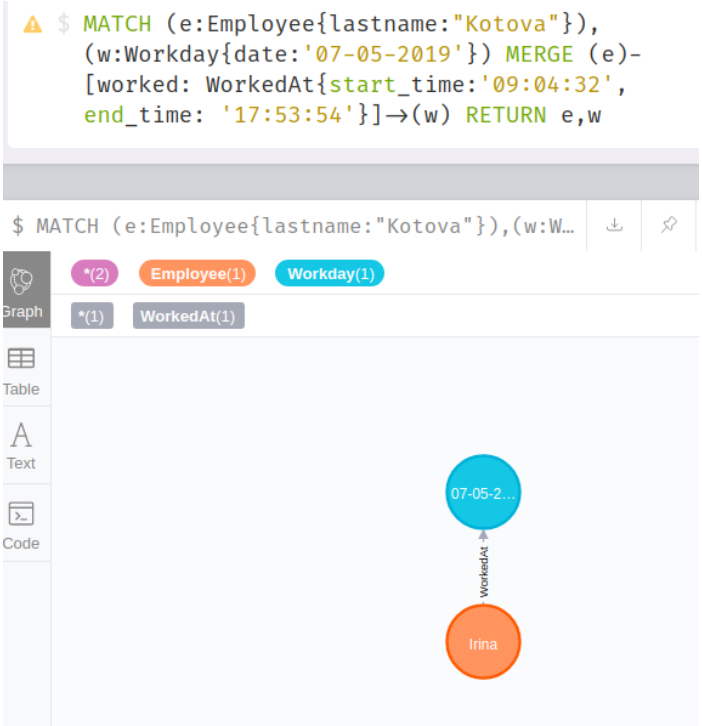


Рисунок 32 – Вывод на экран результата запроса с операторами MATCH и MERGE

Это не единственный вариант создания связей между узлами, более подробно можно найти по ссылке [13.].

Удаление узлов и связей

Ключевое слово **DELETE** может использоваться для удаления узлов или связей из графа.

При этом нельзя удалить узел, не удалив и связи, которые начинаются или заканчиваются в этом узле. Поэтому необходимо использовать команду **DETACH DELETE** или же отдельно удалять все отношения.

Удаление всех узлов и связей:

Чтобы удалить все узлы и связи, необходимо выполнить следующий запрос (результат см. Рисунок 33):

```
MATCH (n) DETACH DELETE n
```

где *n* – узел.

```
(empty result)
```

0 rows, Nodes deleted: 4 Relationships deleted: 2

Рисунок 33 – Удаление всех связей и узлов

Удаление одного узла и всех его связей

Чтобы удалить один узел и все его связи, необходимо выполнить следующий запрос (результат см. Рисунок 34):

```
MATCH (node {attribute: 'value'})
     DETACH DELETE node
```

где *node* – узел,

attribute – названия свойства узла,

value – значения свойства узла.

```
(empty result)
```

0 rows, Nodes deleted: 1 Relationships deleted: 2

Рисунок 34 – Удаление одного узла и всех его связей

Удаление только связей

Чтобы удалить все связи одного узла, необходимо выполнить следующий запрос (результат см. Рисунок 35):

```
MATCH (node {attribute: 'value'}) - [rel: RelationName] → ()
DELETE rel
```

где *node* – узел,

attribute – названия свойства узла,

value – значения свойства узла,

rel – отношение,

RelationName – метка отношения.

```
(empty result)
```

0 rows, Relationships deleted: 2

Рисунок 35 – Удаление только связей узла

Подробнее смотрите в [12.].

Пример удаления связи:

База данных до удаления содержит два узла и связи между ними (см. Рисунок 36).

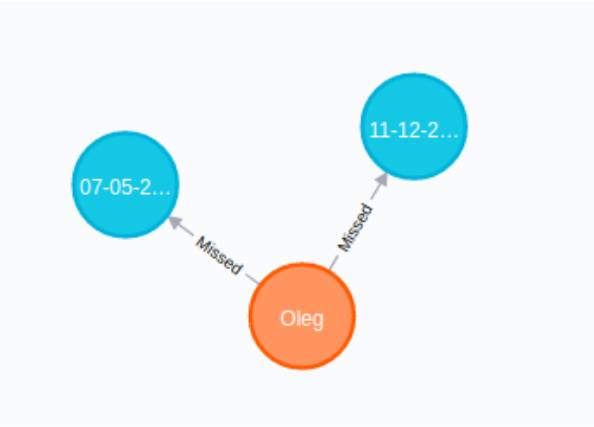


Рисунок 36 – Состояние базы данных до удаления

Удаление связи между узлом **e** с меткой **Employee** с фамилией **Popkov** и узлом **w** с меткой **WorkDay** с датой **07-05-2019**.

```
MATCH (e:Employee)-[m]->(w:Workday) WHERE e.lastname='Popkov' and w.date='07-05-2019' DELETE m
```

Проверим, что связь удалилась, выведем узлы с метками **Employee** и **Workday**, связанные между собой и с фамилией сотрудника **Popkov** (результат см. Рисунок 37):

```
MATCH (e:Employee)-->(w:Workday) WHERE e.lastname ='Popkov'  RETURN e,w
```

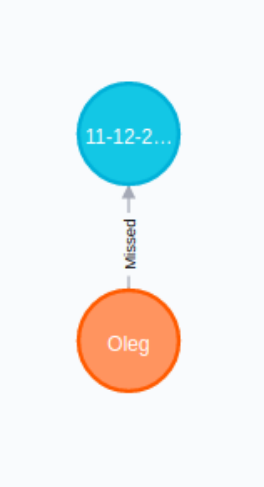


Рисунок 37 – Состояние базы данных после удаления связи между узлами

Пример удаления узла:

Для удаления узла необходимо выполнить следующий запрос (результат см. Рисунок 38):

```
MATCH (e:Employee) WHERE e.firstname='Aleksei' DELETE e
```

Проверка, что узел удален:

```
MATCH (e:Employee) RETURN e
```

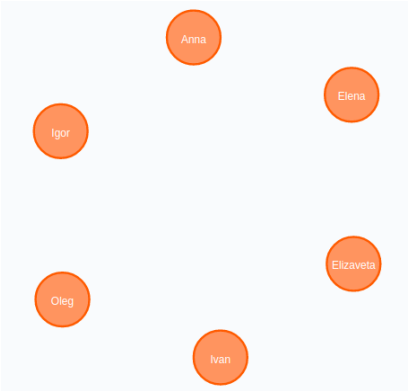


Рисунок 38 – Состояние базы данных после удаления узла

Запросы к БД на языке Cypher

Основные запросы на выборку данных

Оператор MATCH

MATCH – ключевое слово, после которого следует шаблон, описывающий искомую информацию.

Синтаксис запроса, который вернет все узлы в базе данных (см. Рисунок 39):

```
MATCH (n) RETURN n
```

где *n* обозначает узел.

Получение всех узлов под определенной меткой:

```
MATCH (node:Label1)
      RETURN node
```

Получение узлов с несколькими метками:

```
MATCH (node:Label1:Label2)
      RETURN node
```

где *node* – узел,

Label1, Label2 – метки.

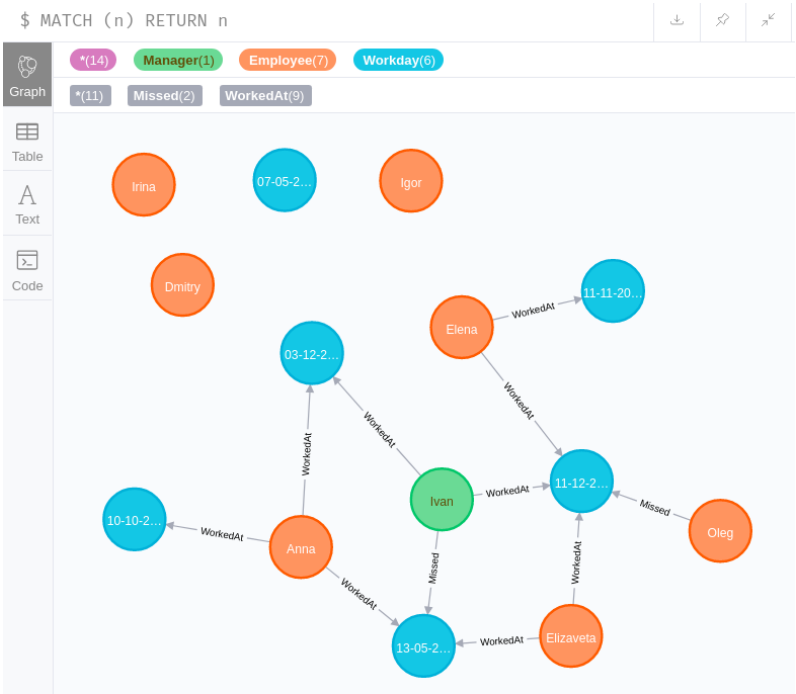


Рисунок 39 – Результат запроса с MATCH

Пример:

Запрос, который выведет все узлы с меткой Employee (см. Рисунок 40):

```
$ MATCH (e:Employee) RETURN e
```

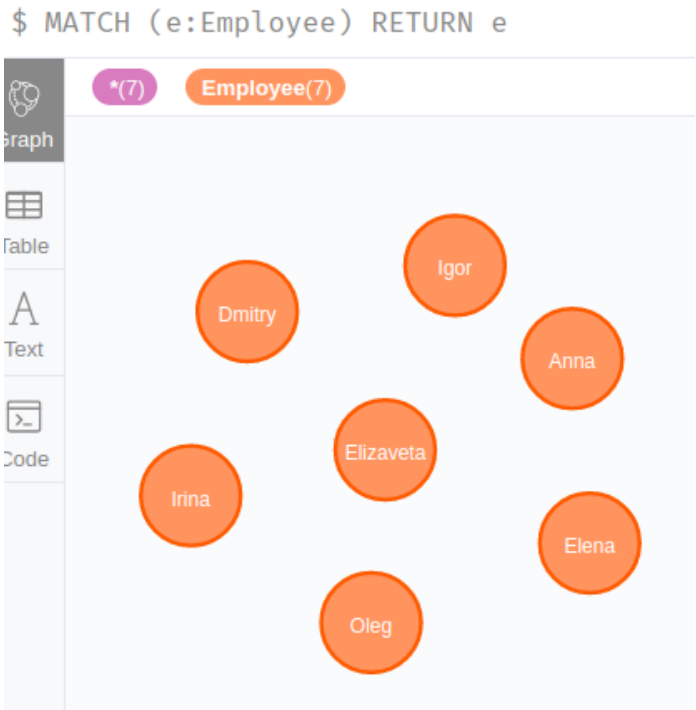


Рисунок 40 – Результат запроса по всем узлам с меткой Employee

Подробнее смотрите в [15.].

Оператор RETURN

Возврат созданного узла. Чтобы посмотреть созданный узел можно воспользоваться командой **RETURN** с **CREATE**.

RETURN – ключевое слово, определяющее, что будет возвращено в результате выполнения запроса.

```
CREATE (node:Label{ key1: value,
key2: value,
. . . . . . . . . }) RETURN node
```

где *node* – узел,

label – метка,

key1, *key2* — название свойства,

value — значение свойства.

Пример:

Данная команда (см. Рисунок 41) создает узел e с меткой Employee и свойствами: табельный номер и ФИО, и выведет его на экран (см. Рисунок 42).

```
$ CREATE (e: Employee { table_number: 7,
lastname: 'Popkov', firstname: 'Oleg',
patronymic: 'Igorevich'}) RETURN e
```

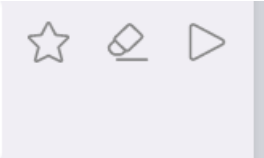


Рисунок 41 – Запрос на создание узла с меткой и свойствами и вывод его на экран

Рисунок 42 – Вывод на экран результата создание узла с меткой и свойствами

В следующем запросе показано, как можно выводить конкретные свойства узлов. В Neo4j можно обращаться к свойствам узлов и отношений через символ точки. Данный запрос выдаст свойства firstname, lastname, patronymic всех узлов с меткой Employee (см. Рисунок 43).

```
$ MATCH (e:Employee) RETURN e.lastname,
  e.firstname, e.patronymic
```

MATCH (e:Employee) RETURN e.lastname, e.fi…

e.lastname	e.firstname	e.patronymic
"Soboleva"	"Elizaveta"	"Denisovna"
"Polyakova"	"Elena"	"Olegovna"
"Kuznecova"	"Anna"	"Egorovna"
"Petrov"	"Igor"	null
"Novikov"	"Dmitry"	null
"Kotova"	"Irina"	null
"Popkov"	"Oleg"	"Igorevich"

Рисунок 43 – Вывод на экран конкретных свойств узла

Подробнее смотрите в [16.].

Оператор WHERE

WHERE – ключевое слово, после которого можно добавить дополнительные ограничения, накладываемые на шаблон, чтобы отфильтровать результаты.

Можно задавать простые условия на значения свойств, на значение меток, а также сложные условия (например, с использованием **AND**, **OR**, **CONTAINS**, регулярных выражений, шаблонов отношений и др. – см. в разделе «Запросы к БД на языке Cypher» и в документации).

Можно фильтровать как узлы, так и отношения.

Фильтрация по значению свойства:

```
MATCH (a) WHERE a.property = "value"  RETURN a
```

Фильтрация по значению метки

```
MATCH (a) WHERE a:Label  RETURN a
```

где, a — узел или отношение;
Label — название метки узла или отношения;
property — название свойства;
value — значение свойства.

Пример:

Запрос, который выведет ФИО сотрудников, которые работают от 30 часов в неделю (см. Рисунок 44). То есть отфильтруются узлы e с меткой Employee, а также в условии фильтрации указано, что значение свойства work_hours_per_week должно быть не меньше 30.

```
$ MATCH (e:Employee) WHERE
  e.work_hours_per_week ≥ 30 RETURN e.lastname,
  e.firstname, e.patronymic
```

$ MATCH (e:Employee) WHERE e.work_hours_per_…

Table | Text | Code

e.lastname	e.firstname	e.patronymic
"Soboleva"	"Elizaveta"	"Denisovna"
"Polyakova"	"Elena"	"Olegovna"
"Popkov"	"Oleg"	"Igorevich"

Рисунок 44 – Вывод на экран результата запроса с фильтрацией

Подробнее смотрите в [17.].

Фильтрация по узлам, отношениям, меткам и связям

Фильтрация по узлам:

Для выполнения фильтрации по узлам необходимо выполнить следующий запрос:

```
MATCH (node)
      WHERE node.attribute=value
      RETURN node
```

где *node* – узел,

attribute — название свойства узла,

value — значение свойства узла.

Пример:

Запрос, выводящий свойства: фамилия, имя - узла e с фамилией Соболева и связанные с данным узлом узлы w, причем отношение должно быть направлено от e к w (см. Рисунок 45).

```
MATCH (e)-->(w)
      WHERE e.lastname='Soboleva'
      RETURN e.lastname,e.firstname, w
```

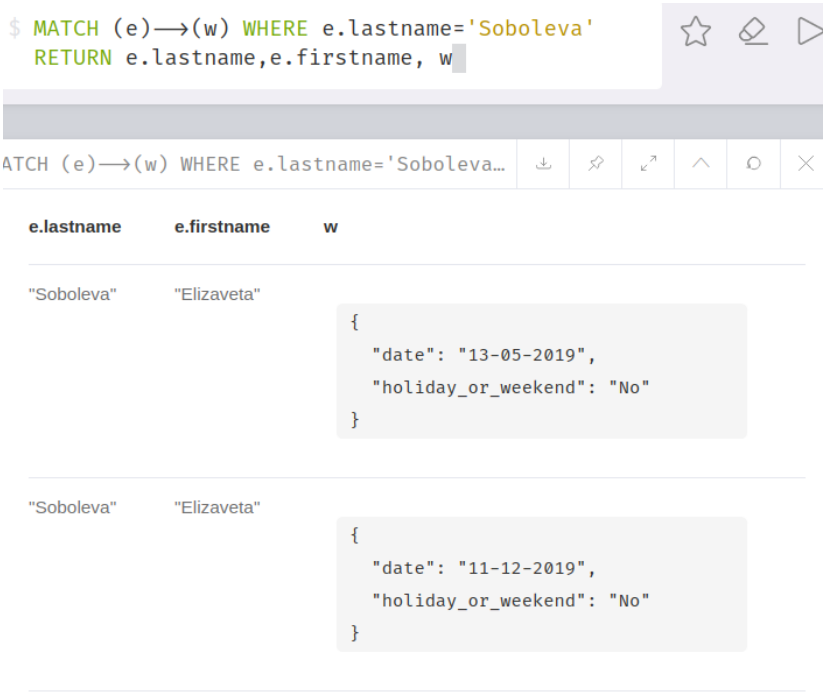


Рисунок 45 – Вывод на экран результата запроса с фильтрацией по узлам

Фильтрация по меткам:

Для выполнения фильтрации по меткам необходимо выполнить следующий запрос:

```
MATCH (node:Label)  RETURN node
```

где *node* – узел,

Label — метка узла.

Пример:

Запрос, который выводит все узлы e с меткой Employee и узлы w, у которых есть свойство – дата 13-05-2019 (см. Рисунок 46).

```
MATCH (e:Employee)-[worker]->(w) WHERE w.date='13-05-2019' RETURN e,w
```

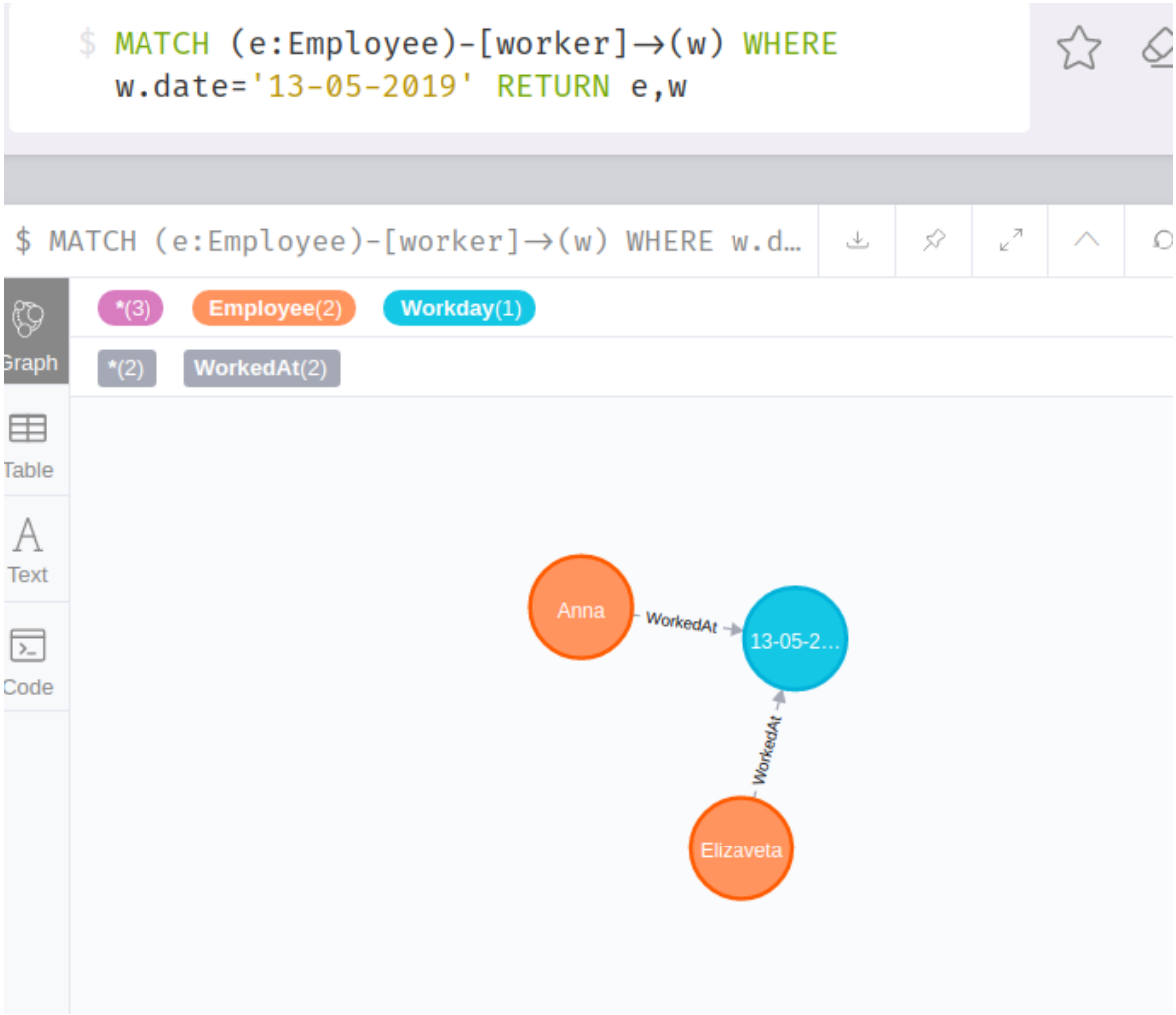


Рисунок 46 – Вывод на экран результата запроса с фильтрацией по меткам

Фильтрация по связям:

Для выполнения фильтрации по связям необходимо выполнить следующий запрос:

```
MATCH (node1)-[rel:RelationName]-(node2)  RETURN node1, node2
```

где *node1, node2* – узлы,

Label1, Label2 — метки узлов,

rel – отношение,

RelationName – метка отношения.

Пример:

Запрос, который выведет все узлы e, w, которые соединены связью с меткой Missed(пропуск) (см. Рисунок 47).

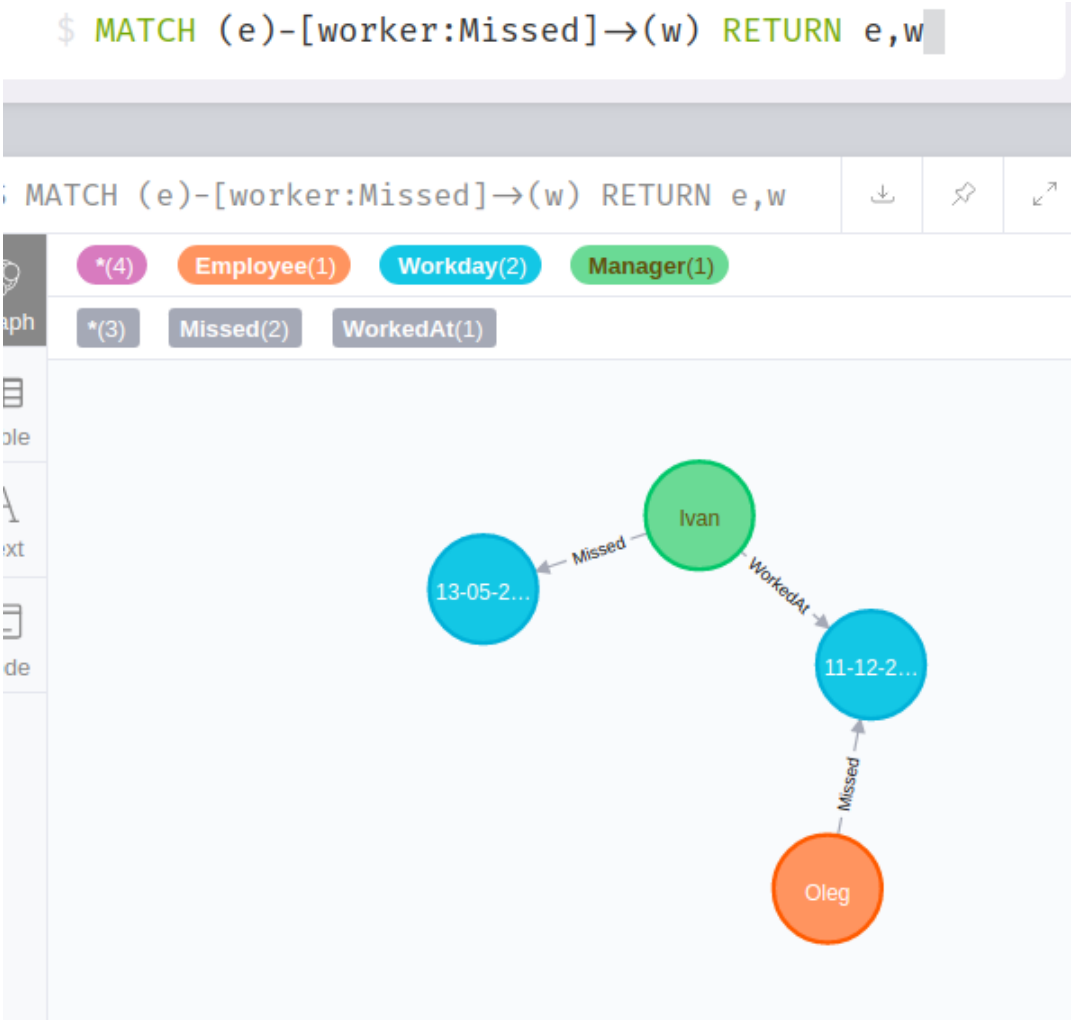


Рисунок 47 – Вывод на экран результата запроса с фильтрацией по связям

Более подробно про фильтрацию о связях можно почитать в [14.].

Условие NOT NULL

Условие **NOT NULL** определяет, что значение свойства задано и не равно **NULL**.

```
MATCH (node) WHERE node.attribute IS NOT NULL RETURN node
```

где *node* – узел,

attribute — название свойства узла.

Пример:

Создадим узел e с меткой Employee и свойствами: табельный номер, фамилия, имя, кол-во рабочих часов в неделю – и не зададим ему свойство «отчество».

```
CREATE (e: Employee { table_number: 6,
                      lastname:'Petrov',
                      firstname:'Igor',
                      work_hours_per_week: 25})
```

Запрос, который выведет все узлы e, у которых свойство отчество не NULL. Как можно заметить, среди выведенных сотрудников нет сотрудников с именем Igor (см. Рисунок 48).

```
MATCH (e)
      WHERE e.patronymic IS NOT NULL RETURN e
```

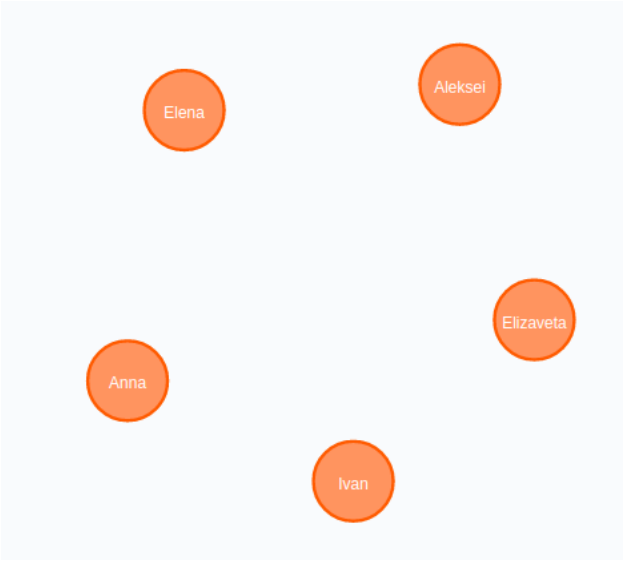


Рисунок 48 – Вывод на экран результата запроса с использование условия NOT NULL

Операторы AND, OR

Приведем пример выполнения запроса с использование оператора **AND**:

```
MATCH (node) WHERE (node.attribute1='value1') AND (node.attribute2='value2')
RETURN node
```

где *node* – узел,

attribute1 attribute2 — название свойств узла,

value1, value2 — значения свойств узла.

Приведем пример выполнения запроса с использование оператора **OR**:

```
MATCH (node) WHERE (node.attribute1='value1') OR (node.attribute2='value2')
RETURN node
```

Пример:

Запрос, выводящий узлы e с меткой Employee и с фамилией Soboleva или Polyakova и связанные с ними узлы w с меткой WorkDay с датой 11-12-2019 или 13-05-2019 (см. Рисунок 49).

MATCH (e:Employee)-->(w:Workday) WHERE (e.lastname='Soboleva' OR e.lastname='Polyakova') AND (w.date='11-12-2019' OR w.date='13-05-2019') RETURN e,w

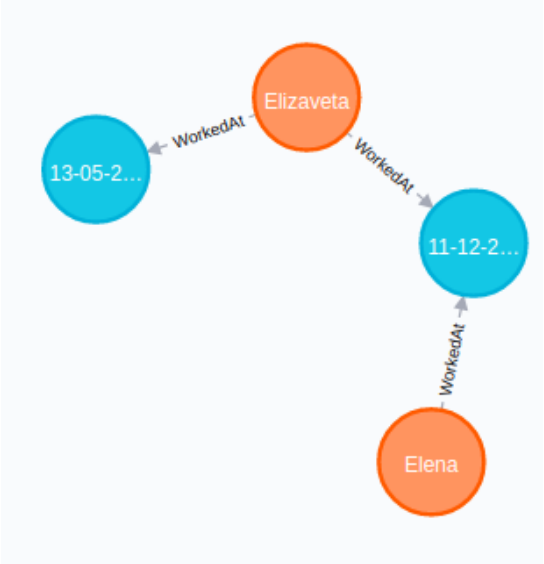


Рисунок 49 – Вывод на экран результата запроса с операторами AND и OR

Подробнее об операторах в [18.].

Сортировка по свойству

Для сортировки по свойству используется оператор **ORDER BY**.

```
MATCH (node) RETURN node.attribute1, node.attribute2 ORDER BY node.attribute1
```

где *node* – узел,

attribute1 attribute2 — название свойств узла,

value1, value2 — значения свойств узла.

Пример:

Запрос, который выведет ФИО сотрудников и отсортирует сначала по фамилии, потом по имени (см. Рисунок 50).

```
MATCH (e:Employee) RETURN e.lastname, e.firstname, e.patronymic ORDER BY
e.lastname, e.firstname
```

e.lastname	e.firstname	e.patronymic
"Ivanov"	"Ivan"	"Ivanovich"
"Kuznecova"	"Anna"	"Egorovna"
"Petrov"	"Igor"	null
"Polyakova"	"Elena"	"Olegovna"
"Popov"	"Aleksei"	"Sergeevich"
"Soboleva"	"Elizaveta"	"Denisovna"

Рисунок 50 – Вывод на экран результата запроса с сортировкой по свойству

Подробнее смотрите в [19.].

Условие на направление отношения

Приведем конструкцию запроса на поиск узлов с направленным отношением между ними:

```
MATCH (node1:Label1)-->(node2:Label2) RETURN node1, node2
```

где *node1, node2* — узлы,

Label1, Label2 – метки.

Пример:

Запрос, выводящий все узлы с меткой Employee со свойствами lastname равными Soboleva или Polyakova и связанные с ними узлы с меткой Workday со свойством date равным 11-12-2019 или 13-05-2019. Направление связи от узлов с меткой Employee к узлам с меткой Workday (см. Рисунок 51).

```
MATCH (e:Employee)-->(w:Workday) WHERE (e.lastname='Soboleva' OR
e.lastname='Polyakova') AND (w.date='11-12-2019' OR w.date='13-05-2019')
RETURN e,w
```

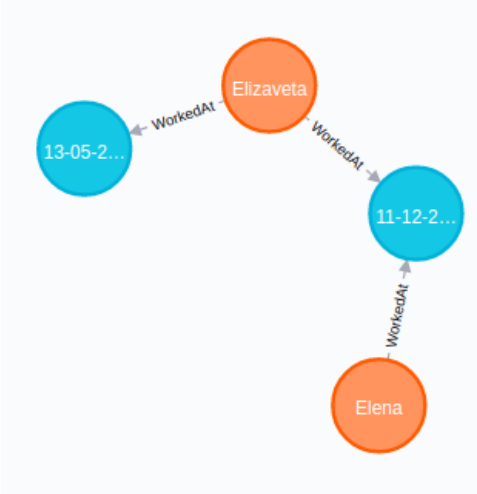


Рисунок 51 – Вывод на экран результата запроса с условием на направление отношения

Тот же запрос, но с обратным направлением связи не выведет ничего (см. Рисунок 52).

```
MATCH (w:Workday)-->(e:Employee) WHERE (e.lastname='Soboleva' OR
e.lastname='Polyakova') AND (w.date='11-12-2019' OR w.date='13-05-2019')
RETURN e,w
```

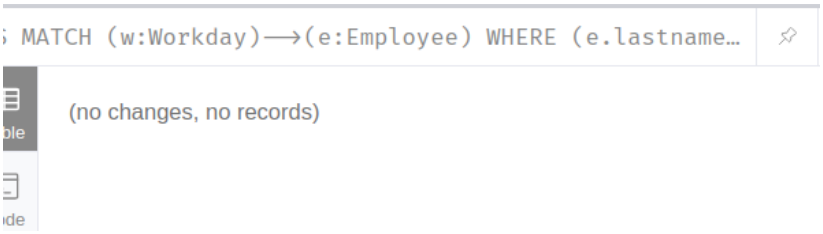


Рисунок 52 – Ошибка при исполнении запроса с условием на направление отношения

Параметры отношения

Приведем конструкцию запроса на поиск узлов с определенными свойствами отношением между ними:

```
MATCH (node1)-[rel:RelationName]->(node2)
      WHERE rel.attribute='value'
      RETURN node1, node2
```

где *node1, node2* – узлы,

Label1, Label2 — метки узлов,

rel – отношение,

RelationName – метка отношения,

attribute – название свойства отношения,

value – значение свойства отношения,

Пример:

Запрос, который вернет только те узлы, которые связаны связью с меткой WorkDay, причем параметр связи end_time равен '18:33:34' или '18:43:24' (см. Рисунок 53).

```
MATCH (e)-[a:WorkedAt]->(w)
      WHERE a.end_time='18:33:34' OR
            a.end_time='18:43:24'
      RETURN e,w
```

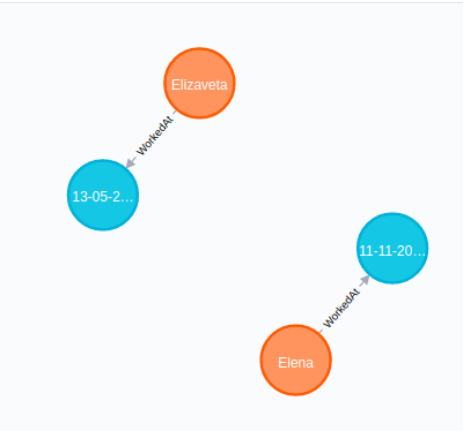


Рисунок 53 – Вывод на экран результата запроса с параметрами отношения

Существование шаблона в базе данных

Для проверки существования паттерна в базе данных используется ключевое слово **MERGE**. Если не существует, то команда создаст его. Чтобы проверить существование паттерна в базе данных необходимо выполнить следующий запрос:

```
MERGE (node:Label{attribute:'value'})
```

где *node* – узел,

Label – метка,

attribute – названия свойства узла,

value – значения свойства узла.

Пример:

Создадим запрос для несуществующего узла с меткой **Employee** и свойствами фамилия, имя и табельный номер.

```
MERGE (e:Employee{lastname:'Novikov',firstname:'Dmitry', table_number:8})
```

В результате создался новый узел (см. Рисунок 54).

Added 1 label, created 1 node, set 3 properties, completed after 1 ms.

Рисунок 54 – Создание узла с проверкой на его существование

Повторим запрос для этого же узла.

```
MERGE (e:Employee{lastname:'Novikov',firstname:'Dmitry', table_number:8})
```

Так как узел уже существует, то новый не создался (см. Рисунок 55).

(no changes, no records)

Рисунок 55 – Отмена создания узла с проверкой на его существование

Подробнее об использовании **MERGE** для отношений смотрите в разделе . В целом об использовании оператора **MERGE** смотрите в [20.].

Объединение результатов запросов

Для объединения результатов нескольких запросов используется ключевое слово **UNION**. Чтобы объединить результаты двух запросов необходимо выполнить следующий запрос:

```
MATCH (node1:Label1) RETURN node1
UNION
MATCH (node2:Label2) RETURN node2
```

где *node1, 2* – узлы,
Label1, 2 – метки**.**

Пример:

Данный запрос состоит из 2 запросов: первый отфильтрует все узлы с меткой Employee, второй отфильтрует узлы с меткой Manager, далее запрос объединит полученные результаты 2 запросов и вернет все узлы Employee и Manager (см. Рисунок 56).

```
MATCH (e:Employee) RETURN e
UNION
MATCH (e:Manager) RETURN e
```

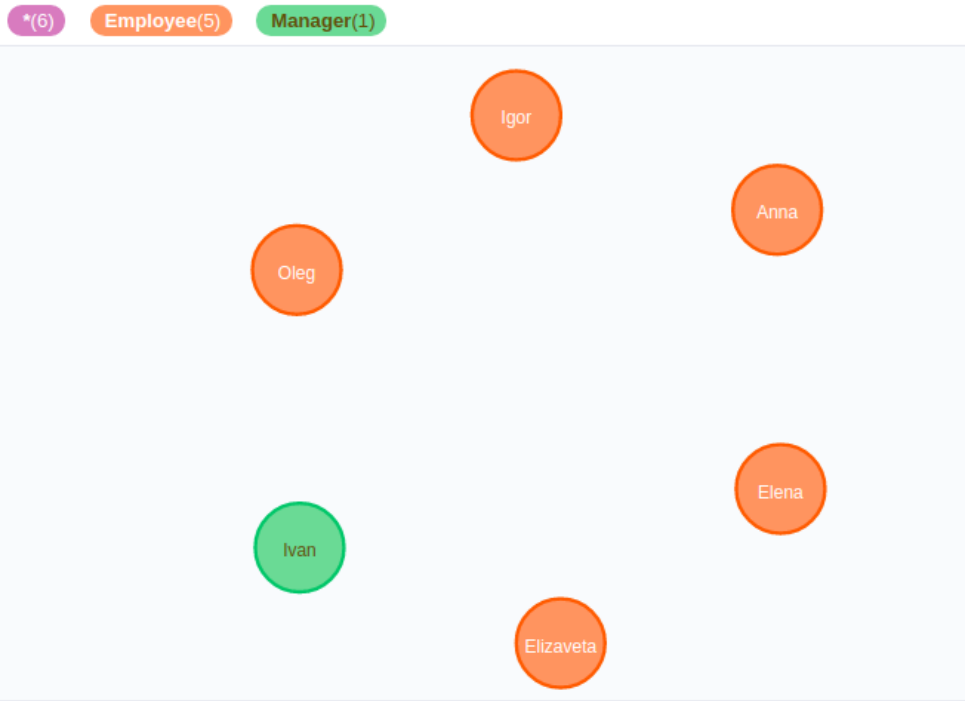


Рисунок 56 – Объединение результатов двух запросов

Подробнее смотрите в [21.].

Расширенные запросы к БД

Ограничение количества строк в выводе

Для ограничения количества строк в выводе используется ключевое слово **LIMIT**. Чтобы просмотреть результат с ограничением количества строк необходимо выполнить следующий запрос:

```
MATCH (n)
      RETURN n LIMIT 10
```

где *node* – узел.

Пример:

Выведем только 5 узлов (см. Рисунок 57).

```
MATCH (w)
      RETURN w LIMIT 5
```

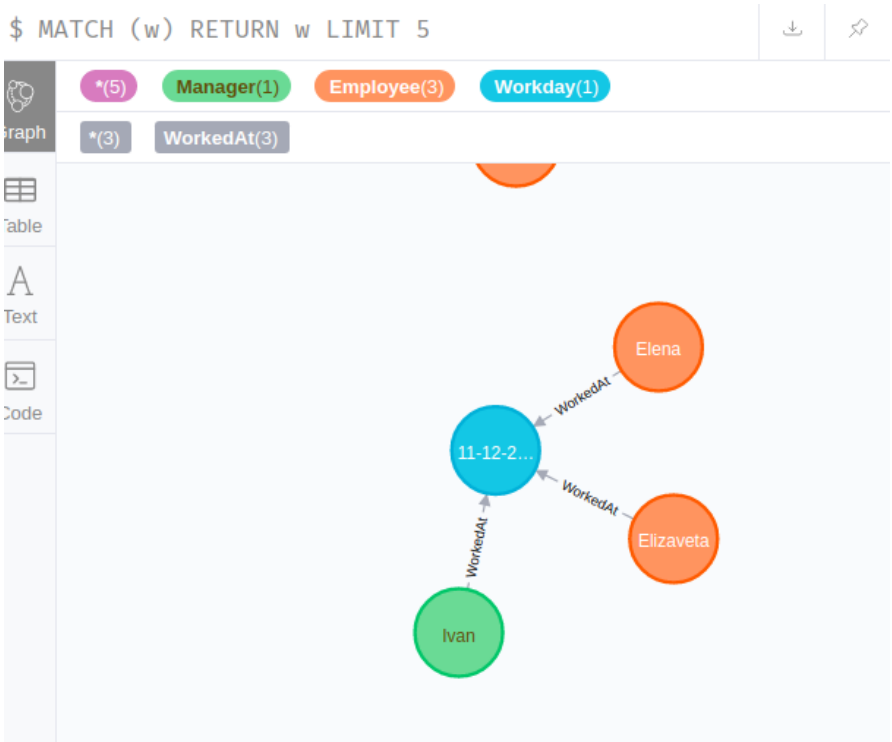


Рисунок 57 – Ограничение количества в выводе

Подробнее смотрите в [22.].

Удаление дубликатов

Для удаления дубликатов используется ключевое слово **DISTINCT**. Чтобы удалить дубликаты, необходимо выполнить следующий запрос:

```
MATCH (node:Label) RETURN DISTINCT node.attribute
```

где *node* – узел,

Label – метка,

attribute – названия свойства узла.

Пример:

Создадим узел с меткой Employee с такими же ФИО как у уже существующего узла.

```
CREATE (e: Employee { table_number: 5,
                      lastname:'Kuznecova',
                      firstname:'Anna',
                      patronymic: 'Egorovna',
                      work_hours_per_week: 25})
```

Выведем фамилии и имена сотрудников.

```
MATCH (e:Employee) RETURN  e.lastname, e.firstname
```

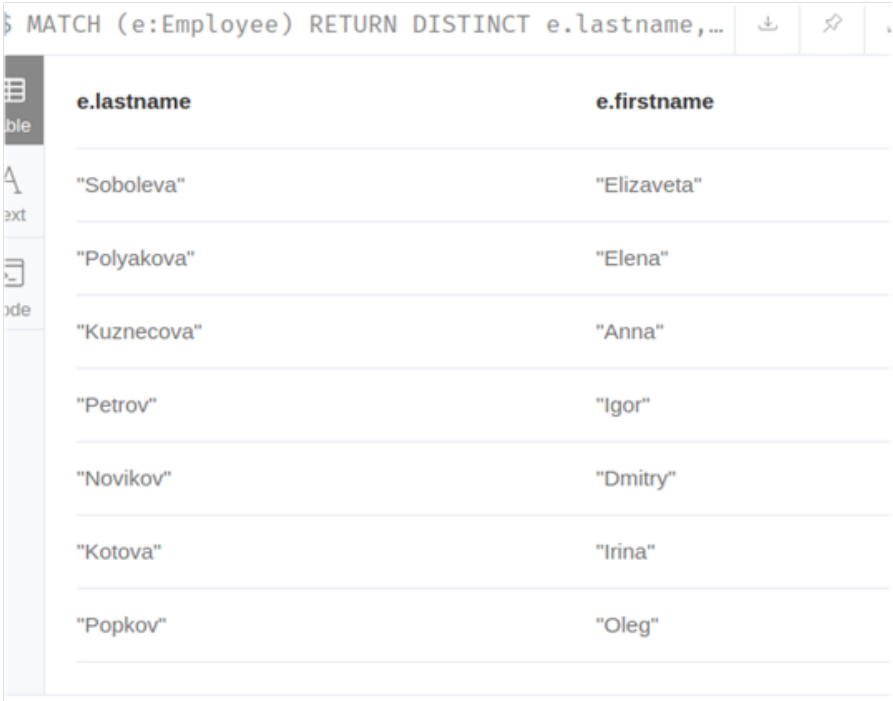
Видим, что запрос вывел 2 Анн Кузнецовых (см. Рисунок 58).

e.lastname	e.firstname
"Soboleva"	"Elizaveta"
"Polyakova"	"Elena"
"Kuznecova"	"Anna"
"Petrov"	"Igor"
"Kuznecova"	"Anna"
"Novikov"	"Dmitry"
"Kotova"	"Irina"
"Popkov"	"Oleg"

Рисунок 58 – Узлы с дубликатами

Выведем только уникальные фамилии и имена. Видим, что теперь вывелась 1 Анна Кузнецова (см. Рисунок 59).

```
MATCH (e:Employee) RETURN DISTINCT e.lastname, e.firstname
```



e.lastname	e.firstname
"Soboleva"	"Elizaveta"
"Polyakova"	"Elena"
"Kuznecova"	"Anna"
"Petrov"	"Igor"
"Novikov"	"Dmitry"
"Kotova"	"Irina"
"Popkov"	"Oleg"

Рисунок 59 – Узлы без дубликатов

Пропуск узлов при выводе

Для того, чтобы определить, сколько узлов необходимо пропустить при выводе, используется ключевое слово **SKIP**. Чтобы просмотреть результат с пропуском узлов необходимо выполнить следующий запрос:

```
MATCH (n)
      RETURN n SKIP x
```

где *node* – узел,

x – количество узлов, которые необходимо пропустить.

Пример:

Запрос, который пропустить первые 5 узлов с меткой Employee и выведет 6-го, 7-го и 8-го сотрудников (см. Рисунок 60).

```
MATCH (e:Employee)
      RETURN e SKIP 5
```

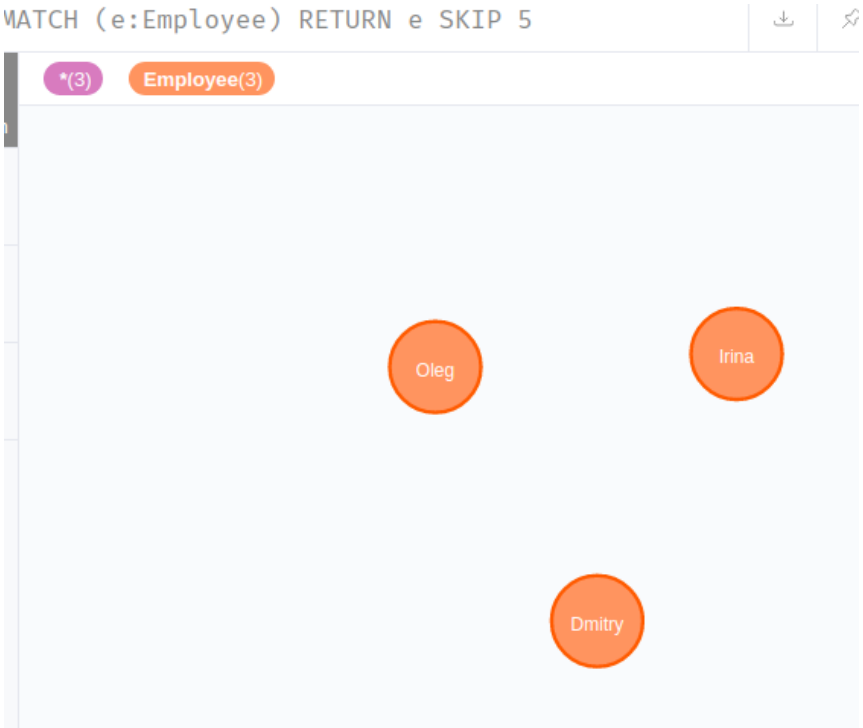


Рисунок 60 – Пропуск узлов при выводе

Подробнее смотрите в [23.].

Агрегирование

Чтобы выполнить агрегированный запрос к базе данных, необходимо использовать следующий синтаксис:

```
MATCH (node: Label)
      RETURN function_name(node.attribute)
```

где *node* – узел,

Label – метка,

attribute – названия свойства узла,

function_name – имя функции.

В Таблица 2 представлены основные агрегатные функции, используемые в Neo4j.

Таблица 2 – Агрегатные функции

Название функции	Описание
avg(expression)	Возвращает среднее значение атрибута
collect(expression)	Возвращает массив из значений атрибута
count(expression)	Возвращает кол-во значений/строк
max(expression)	Возвращает максимальное значение атрибута
min(expression)	Возвращает минимальное значение атрибута
percentileCont(expression, percentile), percentileDisc(expression, percentile)	Возвращает процентиль атрибута (мера, в которой процентное значение общих значений равно этой мере или меньше ее.)
StDev(expression), stDevP(expression)	Возвращают стандартное отклонение значений атрибутов
sum(expression)	Возвращает сумму значений атрибутов

Пример:

Запрос, который выведет количество сотрудников (см. Рисунок 61).

```
MATCH (e:Employee) RETURN count(e) as e_count
```

e_count
7

Рисунок 61 – Агрегированный запрос

Подробнее смотрите в [24.].

Строковые функции

В Таблица 3 приведены строковые функции.

Таблица 3 – Строковые функции

Название функции	Описание
left(original, length)	Возвращает строку, содержащую указанное количество символов в параметре length, слева направо.
lTrim(original)	Удаляет все пробелы слева.
replace(original, search, replace)	Заменяет все выражения указанные в search на выражения, указанные в replace
reverse(origibal)	Возвращает строку с символами, расставленными в обратном порядке.
right(original, length)	Возвращает строку, содержащую указанное количество символов в параметре length, справа налево.
rTrim(original)	Удаляет все пробелы справа.
split(original, splitDelimiter)	Разделяет строку по символам, указанным в splitDelimiter. Возвращает массив.
substring(original, start [, length])	Возвращает подстроку, начиная с символа под номером start, длины length.
toLower(original)	Возвращает строку в нижнем регистре.
toString(original)	Конвертирует integer, float, boolean в string/
toUpper(original)	Возвращает строку в верхнем регистре.
trim(original)	Удаляет пробелы в начале и конце строки.

Синтаксис для функции **substring**:

```
substring(original, start [, length])
```

где *original* — это выражение, возвращающее строку.

Пример:

Запрос, который выведет дату в формате “day «» month «» year «»“ (см. Рисунок 62).

```
MATCH (w:Workday) RETURN ' day '+SUBSTRING(w.date,0,2)+' month
     '+SUBSTRING(w.date,3,2)+ ' year '+SUBSTRING(w.date,6,4) as str_date
```

str_date
" day 11 month 12 year 2019"
" day 11 month 11 year 2019"
" day 10 month 10 year 2019"
" day 03 month 12 year 2019"
" day 13 month 05 year 2019"
" day 07 month 05 year 2019"

Рисунок 62 – Строковые функции

Подробней о строковых функциях смотрите [25.] и обо всех функциях [26.].

Индексы

В Neo4j индекс – это структура данных, которая является избыточной копией некоторых данных и используется для повышения скорости операций поиска данных в базе данных. Это происходит за счет дополнительного пространства для хранения и более медленных операций записи, поэтому решение о том, что индексировать, а что нет, является важной и часто нетривиальной задачей [27.].

Индекс может быть создан для свойства любого узла, которому была присвоена метка. После создания индекса Neo4j будет управлять им и обновлять его при каждом изменении базы данных.

Особенности *индексов*:

1. Лучше указывать имя индекса при его создании, иначе он получит автоматически сгенерированное имя.
2. Имя индекса должно быть уникальным как для индексов, так и для ограничений.
3. Создание индекса не идемпотентно. Будет выдана ошибка, если вы попытаетесь создать один и тот же индекс дважды.

Каждый запрос описывает образец, и в этом образце можно иметь несколько стартовых точек. Стартовой точкой является связь или узел, к которому привязывается образец. Вы можете ввести стартовые точки либо по поиску **id**, либо по поиску в индексе. Если явно не задать стартовые точки, Cypher попытается выявить стартовые точки из запроса. Это делается на основе меток узла или предикатов, содержащихся в запросе.

SQL начинает с результата, который вы хотите получить — то есть мы ВЫБИРАЕМ (**SELECT**) то, что нам нужно, а затем описываем источники данных. В Cypher, предложение START имеет отличную концепцию, которая определяет стартовые точки в графе, откуда должен выполняться запрос. Сам запрос генерируется рекурсивно для каждого объекта в дереве. В процессе выполнения запроса происходит обход узлов, свойств и связей.

С точки зрения SQL, идентификаторы в START подобны именам таблиц, которые указывают на набор узлов или связей. Набор может быть перечислен литерально, передан через параметры или быть определен индексным поиском.

Для создания индекса используется ключевое слово **INDEX**. Чтобы создать индекс для 1 свойства, необходимо выполнить следующий запрос:

```
CREATE INDEX [index_name]
      FOR (node:Label)
      ON (node.attribute)
```

где *node* – узел,

Label – метка,

attribute – название свойства узла,

index_name – название индекса.

Чтобы создать индекс для нескольких свойств, необходимо выполнить следующий запрос:

```
CREATE INDEX [index_name]
      FOR (node:Label)
      ON (node.attribute1,
          node.attribute2,
          …
          n.attribute_n)
```

Пример:

Создание индекса для табельных номеров для узлов с меткой Employee.

```
CREATE INDEX ON :Employee(table_number)
```

Пример:

Запрос, в котором на свойство Табельный номер узлов с меткой Employee накладывается индекс и ищется по нему сотрудник с табельным номером 4 (см. Рисунок 63).

```
MATCH (e:Employee) USING INDEX e:Employee(table_number)
      WHERE e.table_number=4
      RETURN e
```

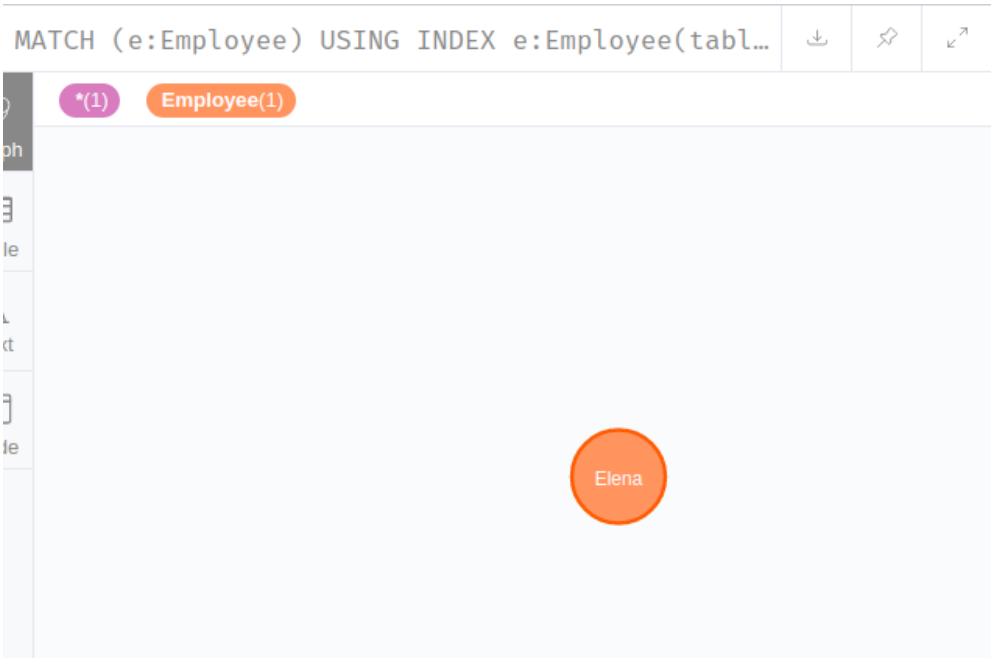


Рисунок 63 – Запрос с использованием индекса

Подробнее про индексы смотрите в [28.].

Просмотр списка индексов

Для просмотра списка индексов используется ключевое слово **CALL**. Чтобы просмотреть список индексов, необходимо выполнить следующий запрос (см. Рисунок 64):

```
CALL db.indexes
```

CALL db.indexes

description	indexName	tokenNames	properties	state	type
"INDEX ON :Employee(table_number)"	"Unnamed index"	["Employee"]	["table_number"]	"ONLINE"	"node_label_p

Рисунок 64 – Просмотр списка индексов

Удаление индекса

Для удаления индекса используется ключевое слово **DROP**. Чтобы удалить индекс, необходимо выполнить следующий запрос:

```
DROP INDEX index_name
```

Удаление индекса одного свойства:

```
DROP INDEX ON :Label(attribute)
```

Удаление индекса нескольких свойств:

```
DROP INDEX ON :Label(node.attribute_1, …, node.attribute_k)
```

где *node* – узел,

Label – метка,

attribute 1... k – название свойства узла,

index_name – название индекса.

Ограничения

В Neo4j ограничения используются, чтобы наложить ограничение на данные, которые могут быть введены для узла или связи. Есть 2 типа ограничений:

1. Ограничение уникальности указывает, что свойство должно содержать уникальное значение (например, табельный номер сотрудника не должен повторяться).
2. Ограничение существования свойства гарантирует, что свойство существует для всех узлов с определенной меткой или для всех связей с определенным типом.

Ограничение уникальности свойства

Для создания ограничения уникальности свойства узла необходимо выполнить следующий запрос:

```
CREATE CONSTRAINT [constraint_name]
      ON (node:Label)
      ASSERT node.attribute IS UNIQUE
```

где *node* – узел,

Label – метка,

attribute – название свойства узла,

constraint_name – название ограничения.

Ограничение на существование свойства узла

Для создания ограничения на существование свойства узла (только для версии Enterprise Edition) необходимо выполнить следующий запрос:

```
CREATE CONSTRAINT [constraint_name]
      ON (node:Label)
      ASSERT EXISTS (node.attribute)
```

Ограничение на существование свойства отношения

Для создания ограничения на существование свойства отношения (только для версии Enterprise Edition) необходимо выполнить следующий запрос:

```
CREATE CONSTRAINT [constraint_name]
      ON ()-[r:RelationName]-()
      ASSERT EXISTS (r.attribute)
```

где *r* – отношение,

RelationName – метка отношения,

attribute – название свойства отношения,

constraint_name – название ограничения.

Ограничение на существование ключа узла

Для создания ограничения на существование ключа узла (только для версии Enterprise Edition) необходимо выполнить следующий запрос:

```
CREATE CONSTRAINT [constraint_name]
      ON (node:Label)
      ASSERT (node.attribute_1,
              node.attribute_2,
              …
              node.attribute_n)
      IS NODE KEY
```

Удаление ограничения

Для удаления ограничения необходимо выполнить следующий запрос:

```
DROP CONSTRAINT constraint_name
```

Вывод всех ограничений

Для вывода всех ограничений необходимо выполнить следующий запрос:

```
CALL db.constraints
```

Пример:

Создание ограничения уникальности для узлов w с меткой Workday для свойства дата.

```
CREATE CONSTRAINT ON (w:Workday) ASSERT w.date is UNIQUE
```

Попытка создать узел w с меткой Workday с уже существующей датой 10-10-2019 (см. Рисунок 65).

```
CREATE (w: Workday{  date: '10-10-2019', holiday_or_weekend: 'No'})
```

ERROR **Neo.ClientError.Schema.ConstraintValidationFailed**

```
Node(36) already exists with label `Workday` and property `date` = '10-10-
2019'
```

Рисунок 65 – Создание узла со повторяющимся уникальным свойством

Подробнее смотрите в [29.].

Коллекции или списки

Коллекция (список) создается при помощи квадратных скобок, внутри которых находятся элементы коллекции, разделенные запятыми.

Создание списка

Для создания списка (см. Рисунок 66) необходимо выполнить следующий запрос:

```
RETURN [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] AS list
```

list
[0,1,2,3,4,5,6,7,8,9]
1 row

Рисунок 66 – Коллекция (список)

Работа с коллекциями (списками)

Для работы со списками можно использовать функцию **RANGE** (диапазон). Она возвращает коллекцию, содержащую все номера между заданными начальным и конечным. Чтобы получить доступ к отдельным элементам в коллекции, также используются квадратные скобки.

Получение элемента списка

Для получения 3-го элемента списка [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] (см. Рисунок 67) необходимо выполнить следующий запрос:

```
RETURN range(0, 10)[3]
```

range(0, 10)[3]
3
1 row

Рисунок 67 – Получение элемента списка

Для отсчета от конца коллекции можно также использовать отрицательные номера:

```
RETURN range(0, 10)[-3]
```

Получение диапазона списка

Для получения диапазона коллекции можно использовать диапазоны внутри квадратных скобок:

```
RETURN range(0, 10)[0..3]
```

Получение размера списка

Для получения размера списка (см. Рисунок 68) необходимо использовать функцию size():

```
size(range(0, 10)[0..3])
```

size(range(0, 10)[0..3])
3
1 row

Рисунок 68 – Размер списка

Генерация коллекций (списков)

Генерация списков (List comprehension) — это конструкция для создания коллекции на основе существующей коллекции, использующая математическую нотацию построения множества вместо функций отображения или фильтрации.

Запрос, который ищет в коллекции [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] четные числа и создает из них список (см. Рисунок 69):

```
RETURN [x IN range(0,10) WHERE x % 2 = 0 ] AS result
```

result
[0,2,4,6,8,10]
1 row

Рисунок 69 – Генерация списка четных чисел

Генерация по шаблону

Генерация по шаблону (Pattern comprehension) — конструкция для создания коллекции на основе шаблона.

Запрос, который создает список годов выпуска фильмов, в которых участвовал Киану Ривз (см. Рисунок 70):

```
MATCH (a:Person { name: 'Keanu Reeves' })
      RETURN [(a)-->(b) WHERE b:Movie | b.released] AS years
```

years
[1997,2000,2003,1999,2003,2003,1995]
1 row

Рисунок 70 – Генерация списка по шаблону

Создание свойства – коллекции

Для создания свойства – коллекции узла необходимо выполнить следующий запрос:

```
CREATE (node:Label { attribute: [value_1, …, value_n] ,
                      . . . . . . . . .  })
```

где *node* – узел,

Label – метка,

attribute – название свойства узла,

value1…n – значения свойства.

Добавление свойства — коллекции в уже существующий узел

Для добавления свойства – коллекции в уже существующий узел необходимо выполнить следующий запрос:

```
MATCH (node:Label { attribute1: value1})
      SET node.attribute2=[value2, …, value_n]
```

где *node* – узел,

Label – метка,

attribute1, 2 – название свойства узла,

value1…n – значения свойства.

Пример:

Для узла e с меткой Employee и свойством фамилия Soboleva зададим коллекцию со списком адресов электронной почты.

```
MATCH (e:Employee)
      WHERE e.lastname='Soboleva' SET e.emails=['lsls@yandex.ru',
                                                'lisobol@gmail.com']
```

Просмотрим информацию о данном сотруднике (см. Рисунок 71):

```
MATCH (e:Employee)
      WHERE e.lastname='Soboleva'
      RETURN e.lastname, e.firstname, e.emails
```

$ MATCH (e:Employee) WHERE e.lastname='Soboleva'…

Table | Text

e.lastname	e.firstname	e.emails
"Soboleva"	"Elizaveta"	["lsls@yandex.ru", "lisobol@gmail.com"]

Рисунок 71 – Просмотр информации об узле со свойством – коллекцией

Выведем второй элемент списка электронных адресов (см. Рисунок 72). Нумерация элементов коллекции начинается с 0.

```
MATCH (e:Employee)
      WHERE e.lastname='Soboleva'
      RETURN e.lastname, e.firstname, e.emails[1]
```

$ MATCH (e:Employee) WHERE e.lastname='Soboleva'…

Table | Text

e.lastname	e.firstname	e.emails[1]
"Soboleva"	"Elizaveta"	"lisobol@gmail.com"

Рисунок 72 – Просмотр информации об узле с выбором одного элемента коллекции

Выведем количество элементов коллекции с помощью функции size() (см. Рисунок 73).

```
MATCH (e:Employee)
      WHERE e.lastname='Soboleva'
      RETURN e.lastname, e.firstname, size(e.emails)
```

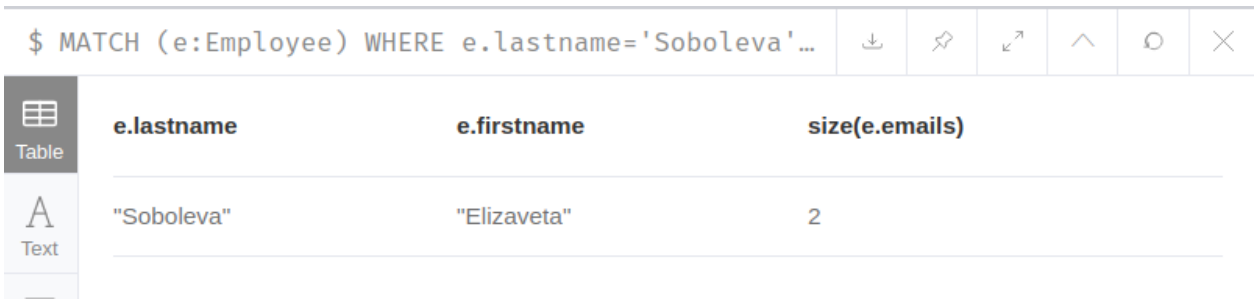


Рисунок 73 – Просмотр информации об узле с количеством элементов коллекции

Проверка наличия заданного адреса электронной почты в коллекции (см. Рисунок 74).

```
MATCH (e:Employee)
      WHERE 'lsls@yandex.ru' IN e.emails
      RETURN e.lastname, e.firstname, e.emails
```

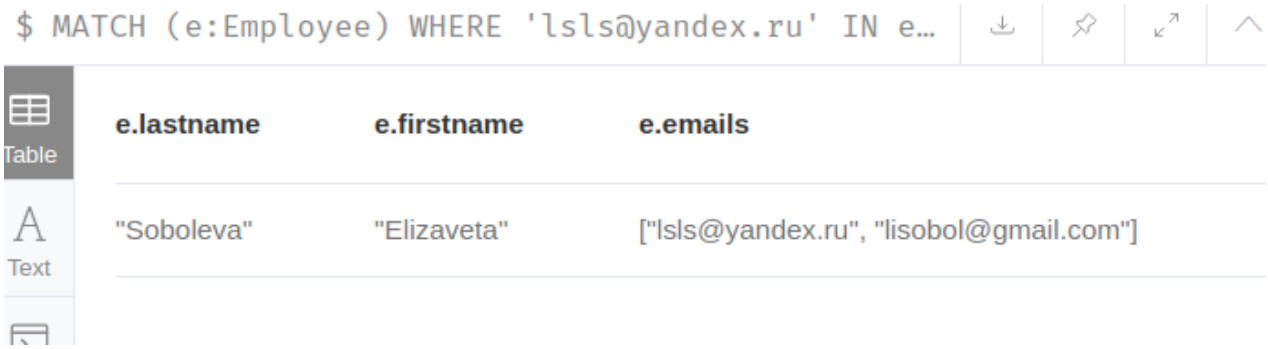


Рисунок 74 – Проверка наличия заданного элемента коллекции

Подробнее смотрите в [30.].

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем особенности графовых СУБД?
2. Модель данных графовой СУБД на примере Neo4j?
3. Что такое узел, отношение, метки и свойства?
4. Как в Neo4j создавать узлы и их связи? Как задавать свойства и метки?
5. Функциональные возможности и языки запросов для СУБД Neo4j?
6. Типы данных, поддерживаемые СУБД Neo4j?
7. Как организована идентификация узлов?
8. Базовая конструкция запросов на выборку данных? Как в запросе указать условие на свойства, метки и шабон отношений?
9. Основные CRUD команды и конструкции запросов языка CQL?
10. Команды CREATE, MERGE. В чем их отличие?
11. Технология репликации и фрагментации в СУБД Neo4j?
12. Поддержка транзакций в СУБД Neo4j?

4. СПИСОК ИСТОЧНИКОВ

1. Neo4j – Текст. Изображение: электронные // Neo4j: [сайт]. – URL: Дистрибутив и документация – https://neo4j.com/ (дата обращения: 15.12.2018)
2. Виноградов В.И., Виноградова М.В. Постреляционные модели данных и языки запросов: Учебное пособие. — М.: Изд-во МГТУ им. Н.Э. Баумана, 2017. – 100с. - ISBN 978-5-7038-4283-6. URL: http://ebooks.bmstu.ru/catalog/254/book1615.html (дата обращения: 26.04.2022). - Текст. Изображение : электронные.
3. Маркин, А. В. Системы графовых баз данных. Neo4j : учебное пособие для вузов / А. В. Маркин. — Москва : Издательство Юрайт, 2021. — 303 с. — (Высшее образование). — ISBN 978-5-534-13996-9. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/467452 (дата обращения: 19.05.2022).
4. Робинсон Ян, Вебер Джим, Эифрем Эмиль. Графовые базы данных: новые возможности для работы со связанными данными / пер. с англ. Р.Н. Рагимова; науч. Ред. А.Н. Кисилев. – 2-е изд. – М.: LVR Пресс, 2016. – 256 с.: ил. ISBN 978-5-97060-201-0
5. OpenSypher – Текст. Изображение: электронные // OpenSypher: [сайт]. – URL: https://opencypher.org/cips/ (дата обращения: 01.05.2022)
6. Фаулер, Мартин, Садаладж, Прамодкумар Дж. NoSQL: новая методология разработки нереляционных баз данных. : Пер. с англ. - М.: ООО "И.Д. Вильямс", 2013г.
7. Neo4j Documentation: macOS installation – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/operations-manual/current/installation/osx/ (дата обращения: 01.05.2022)
8. Neo4j Documentation: CREATE – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/clauses/create/ (дата обращения: 01.05.2022)
9. Neo4j Documentation: SET – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/clauses/set/ (дата обращения: 01.05.2022)

10. Neo4j Documentation: REMOVE – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/clauses/remove/ (дата обращения: 01.05.2022)

11. Neo4j Documentation: Patterns: Patterns for relationships – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/syntax/patterns/#cypher-pattern-relationship (дата обращения: 01.05.2022)

12. Neo4j Documentation: DELETE – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/clauses/delete/ (дата обращения: 01.05.2022)

13. Neo4j Documentation: CREATE: Create relationship – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/clauses/create/#create-relationships (дата обращения: 01.05.2022)

14. Neo4j Documentation: MATCH: Match relationship – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/clauses/match/#relationship-basics (дата обращения: 01.05.2022)

15. Neo4j Documentation: MATCH – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/clauses/match/ (дата обращения: 01.05.2022)

16. Neo4j Documentation: RETURN – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/clauses/return/ (дата обращения: 01.05.2022)

17. Neo4j Documentation: WHERE – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/clauses/where/ (дата обращения: 01.05.2022)

18. Neo4j Documentation: Operators – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/syntax/operators/ (дата обращения: 01.05.2022)

19. Neo4j Documentation: ORDER BY – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/clauses/order-by/ (дата обращения: 01.05.2022)

20. Neo4j Documentation: MERGE – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/clauses/merge/ (дата обращения: 01.05.2022)

21. Neo4j Documentation: UNION – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/clauses/union/ (дата обращения: 01.05.2022)

22. Neo4j Documentation: LIMIT – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/clauses/limit/ (дата обращения: 01.05.2022)

23. Neo4j Documentation: SKIP – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/clauses/skip/ (дата обращения: 01.05.2022)

24. Neo4j Documentation: Aggregating functions– Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/functions/aggregating/ (дата обращения: 01.05.2022)

25. Neo4j Documentation: String functions– Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/functions/string/ (дата обращения: 01.05.2022)

26. Neo4j Documentation: Functions– Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/functions/ (дата обращения: 01.05.2022)

27. CoderLessons.com – Текст. Изображение: электронные // CoderLessons: [сайт]. – URL: https://coderlessons.com/tutorials/bazy-dannykh/uznaite-neo4j/neo4j-indeks (дата обращения: 01.05.2022)

28. Neo4j Documentation: Indexes for search performance – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/indexes-for-search-performance/ (дата обращения: 01.05.2022)

29. Neo4j Documentation: Constraints – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/administration/constraints/ (дата обращения: 01.05.2022)

30. Neo4j Documentation: Lists – Текст. Изображение: электронные // Neo4j Docs: [сайт]. – URL: https://neo4j.com/docs/cypher-manual/current/syntax/lists/ (дата обращения: 01.05.2022)