Московский Государственный Технический Университет имени Н.Э. Баумана
Кафедра "Системы обработки информации и управления" (ИУ5)

Методические указания к лабораторной работе

по теме

«Работа с графовой NOSQL БД на примере Neo4j»

по дисциплине «Постреляционные базы данных»

Составила: Соболева Е.Д.

Москва, 2020 г.

Оглавление

Цель работы:

1. Изучить модель представления данных и способы работы с графовыми БД NoSql.
2. Освоить методы создания графовой БД и языки запросов к ней.
3. Получить навыки работы с графовой БД Neo4j.

Время выполнения:

Время выполнения лабораторной работы 4 часа.

Исходные данные:

1. (локальный ПК) Дистрибутив и документация - https://neo4j.com/ .
2. (доп., кратко)Руководство по установке и началу работы - https://ru.bmstu.wiki/Neo4j
3. Фаулер, Мартин, Садаладж, Прамодкумар Дж. NoSQL: новая методология разработки нереляционных баз данных. : Пер. с англ. - М.: ООО "И.Д. Вильямс", 2013г.

Пункты задания для выполнения:

Задание 1. Создание БД (базовая часть)

Создать в Neo4j базу данных по теме своего ДЗ. Определить набор узлов, задать их свойства и метки.

Продемонстрировать (вывести на экран) содержимое БД (узлы и их свойства), используя команды Match/Where/Return.

Задание 2. Отношения между узлами (базовая часть)

Создать отношения между несколькими узлами (с параметрами).

Продемонстрировать содержимое БД (фильтрация по узлам, отношениям, меткам и связям).

Задание 3. Запросы к БД на языке Cypher **(базовая часть)**

Выполнить запросы к базе данных на языке Cypher:

4. с условием NOT NULL
5. операторами AND, OR
6. с сортировкой
7. с условием на направление отношения
8. с параметрами отношения

Задание 4. CRUD **(хорошо)**

Создать отношение между новыми узлами.
Создать отношение между существующими узлами.
Продемонстрировать удаление узлов и связей.
Продемонстрировать удаление и изменение свойств и меток.
Продемонстрировать работу команд UNION, MERGE.

Задание 5. Расширенные запросы к БД **(хорошо)**

Выполнить запросы к базе данных на языке Cypher:

9. с агрегированием,
10. с встроенными функциями (строковые или иные),
11. с шаблонами отношений,
12. с удалением дубликатов.

Продемонстрировать работу команд LIMIT, SKIP.

Задание 6. Индексы и ограничения (отлично)

Создать индекс. Продемонстрировать его использование.
Создать ограничение. Продемонстрировать его использование.
Создать коллекцию. Продемонстрировать запрос к ней.

图 DBMS 的特点
图数据库是使用图的数据库
用于从流程中的节点、边和属性构建语义查询的结构
数据的呈现和存储。图数据库用于存储、管理和
编译对复杂且密切相关的数据组的查询。除了，
图数据库架构特别适合分析数据
检测大型数据集中的巧合和异常，以及盈利
使用在数据库中得出的关系。
图数据库模型与面向对象一起出现
楷模。这些模型试图限制传统模型。
动机
此类系统的发展已成为一种图数据结构，出现在此类
应用，
作为超文本或地理信息系统，其中之间的关系
数据是一个重要方面。半结构化模型被称为
旨在以灵活的结构存储数据的概念，例如
文档和网页。
形式上，图是一组顶点和边。图表表示连接
作为节点，以及这些对象作为关系相互关联的方式。

Особенности графовых СУБД

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. 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.

Графовые модели баз данных появились наряду с объектно-ориентированными моделями. Эти модели пытаются ограничения, накладываемые традиционными моделями. Мотивацией для
разработки таких систем стала графовая структура данных, появляющихся в таких приложениях,
как гипертекстовые или географические информационные системы, где взаимосвязь между данными является важным аспектом. Полуструктурированными моделями называют такие концепции, которые предназначены для хранения данных с гибкой структурой, например, документов и веб-страниц.

Формально, граф является набором вершин и ребер. Графы представляют собой связи как узлы и как способы, которыми эти объекты связаны друг с другом в качестве отношений.

В общем смысле, такая структура позволяет моделировать все виды сценариев, от строительства космической ракеты до системы дорог. Графы полезны для понимания широкого разнообразия
наборов данных в таких областях как наука, управление или бизнес. В дальнейшем, в концепцию графовой модели базы данных будут включены три основных компонента, а именно:

- структура данных, язык преобразования данных и ограничение их целостности.
- Следовательно, графовые модели базы данных характеризуется следующим образом:
  - данные и/или схемы представлены в виде графов, или с помощью структур
  - данных, которые в итоге сводятся к графам (гиперграфы или гипервершины);
- манипуляция над данными выражается либо в виде преобразовании графов, либо операциями, примитивы которых являются графовые структуры, такие как пути, подграфы, формы графа, связи и прочие;
- согласование данных обеспечивается с помощью ограничений целостности. Эти ограничения могут быть сгруппированы в последовательности схемы, например, идентичности и ссылочной целостности, и функциональные зависимостей включения.

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

在一般意义上，这样的结构允许对各种场景进行建模，从为道路系统建造太空火箭。图表有助于理解各种各样
科学、管理或商业等领域的数据集。后来，在图数据库模型的概念将包括三个主要组件，以及确切地：
? 数据结构、数据转换语言及其完整性的限制。
? 因此，图数据库模型的特点如下：
? 数据和/或模式表示为图形或结构
? 最终归结为图的数据（超图或超顶点）；
? 数据操作被表示为图形的转换，或者其原语是图结构的操作，例如路径、子图、图表形式、连接和其他；
? 使用完整性约束确保数据一致性。这些限制可以分组为模式序列，例如身份和
参照完整性和功能包含依赖性。
图数据库的组件是节点和边。他们可以补充自己的一组字段。这种数据库的模型如图 1 所示

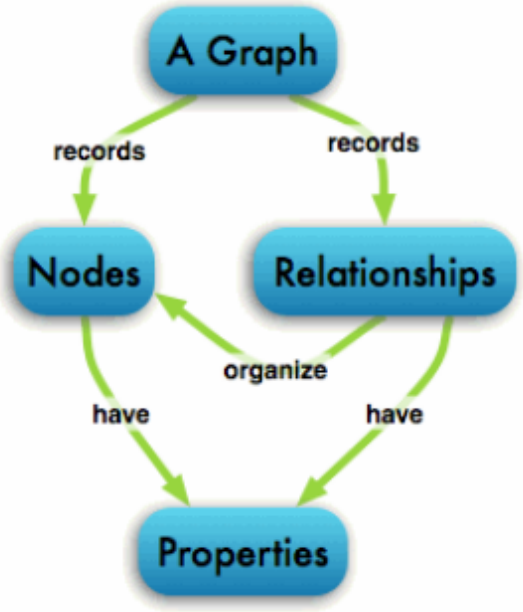


Рисунок 1. Модель графовой базы данных.

Cypher - это декларативный язык, основанный на SQL, для визуального описания графиков, в которых используется синтаксис ascii-art. 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

Cypher 是一种基于 SQL 的用于视觉描述的声明性语言
使用 ascii-art 语法的绘图。 这个实现允许我们
指定我们要从图形中选择、插入、更新或删除的内容
数据，而不需要我们准确描述如何去做。Cypher 类型系统的详细信息

описана в Cypher Improvement Proposal (CIP), и содержит следующие типы: узлы, отношения, пути, карты, списки, целые числа, числа с плавающей запятой, логические значения и строки.
在 Cypher Improvement Proposal (CIP) 中描述，并包含以下类型：节点、关系、路径、映射、列表、整数、浮点数、布尔值和字符串。

Согласованность данных 数据一致性

В рамках отдельного сервера данные всегда являются согласованными, особенно в базе Neo4J, которая полностью поддерживает транзакции ACID. 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.

在单个服务器中，数据始终是一致的，尤其是在数据库中
Neo4J 完全支持 ACID 事务。 如果 Neo4J 基础正在运行
集群，写入主节点与从节点同步，从节点总是可供阅读。 从节点的写入操作始终可用且立即可用
写主节点同步，其他从节点不同步
立即 他们等到数据从领先节点分发。

Транзакции

База Neo4J поддерживает транзакции ACID. 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();
}
```

Neo4J 基础支持 ACID 事务。 在更改任何节点之前
或添加与现有节点的一些关系，您需要启动一个事务。
如果我们不将操作打包到事务中，我们将得到一个 NotinTransactionException 异常。
可以在不创建事务的情况下执行读取操作。

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.

在这段代码中，一个数据库事务已经启动，然后创建了一个节点并且它的特性。 事务用success函数标记，并用finish函数结束。 交易
一定要标注成功功能，否则Neo4J基地会认为已经发生
错误，并在调用完成函数时回滚。 不调用就调用成功函数
完成功能也不会将数据提交到数据库。 在开发应用程序时，
注意事务的这个特性，它不同于数据库中的事务
关系数据库管理系统。

Доступность 可用性

从 Neo4J 1.8 开始，提供了数据库的高可用性复制的从节点。此外，这些从节点可以执行写操作：它们将它们的写操作与当前的 master 同步，并且首先
在主节点上修复它们，然后在从节点上修复它们。毕竟是更新。
Neo4J 基础使用 Apache ZooKeeper 程序
跟踪每个从属和当前事务上的最后一个事务的 ID
主导节点，如果领导者失败，则集群选择领导者
在可用节点之间，提供高可用性。

Начиная с версии Neo4J 1.8 высокая доступность базы обеспечивается реплицированными ведомыми узлами. 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.

Neo4J 基础还支持用于图遍历的 Cypher 查询语言。 除了这些查询语言，Neo4J 库允许您查询节点的属性，遍历图，并使用语言绑定浏览关系。

Масштабирование

При масштабировании баз данных NoSQL широко используется фрагментация, в ходе которой данные разделяются и распределяются по разным серверам. 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].

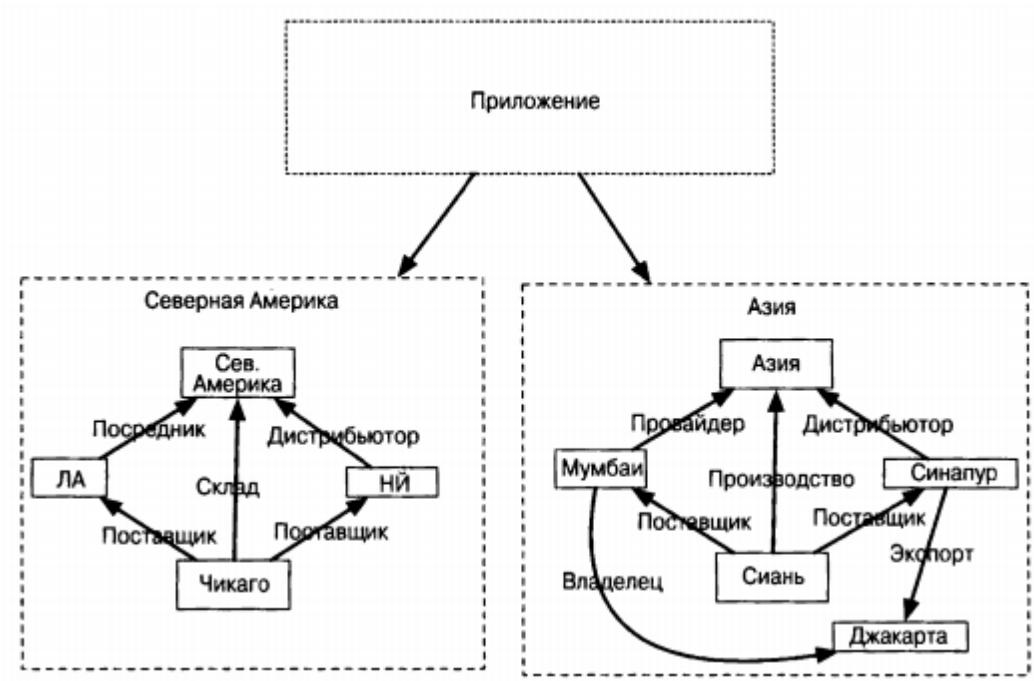
В принципе существуют три способа масштабирования графовых баз данных. Поскольку в настоящее время компьютеры могут иметь большой объем оперативной памяти, можно добавить на сервер столько микросхем, чтобы работать с множеством узлов и отношений, находящимся целиком в оперативной памяти. Этот метод оказывается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

缩放
在扩展 NoSQL 数据库时，碎片被广泛使用，在哪些数据被分离并分布在不同的服务器之间。在图库中数据碎片化很难，因为它们关联在一起，而是关系。由于任何节点都可以与任何其他节点相关联，将相关节点保持在同一台服务器上可提高效率图的遍历。遍历节点数在不同计算机上的图形效率低。知道了图数据库的这个局限性，我们仍然可以使用 Jim Webber 描述的常用方法对它们进行缩放[韦伯 Neo4J 缩放]
原则上，可以通过三种方式来扩展图数据库。
由于现在的计算机可以拥有大量 RAM，
您可以向服务器添加足够的芯片来处理多个节点和完全在 RAM 中的关系。这个方法很有用
只有在我们的数据集确实很大时
适合内存。
您可以通过添加额外的从节点来改进读取扩展，
提供只读数据，并在主导节点上执行所有写操作
这种方法，它涉及一次写入数据并从多个读取它
服务器，已经在 MySQL 集群上很好地证明了自己，结果证明是真的
在处理大到不适合的数据集时很有用
一台计算机的 RAM，但小到足以在其上创建副本
不同的电脑。从节点提高可用性并促进扩展
读取，因为它们可以配置为永远不会成为
领导者并仅执行读取操作。如果数据集太大

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 (рис. 2).

Рисунок 2. Фрагментация узлов на уровне приложения.



创建其副本是不切实际的，您可以在旁边执行数据碎片
使用领域知识的应用程序。 例如，与 North 关联的节点
美国，您可以在一台服务器上创建，而与亚洲相关的节点 - 在另一台服务器上。
在应用端进行这样的分片时，应该理解节点是存储的
在物理上不同的数据库中（图 2）。
图 2. 应用层节点的碎片化

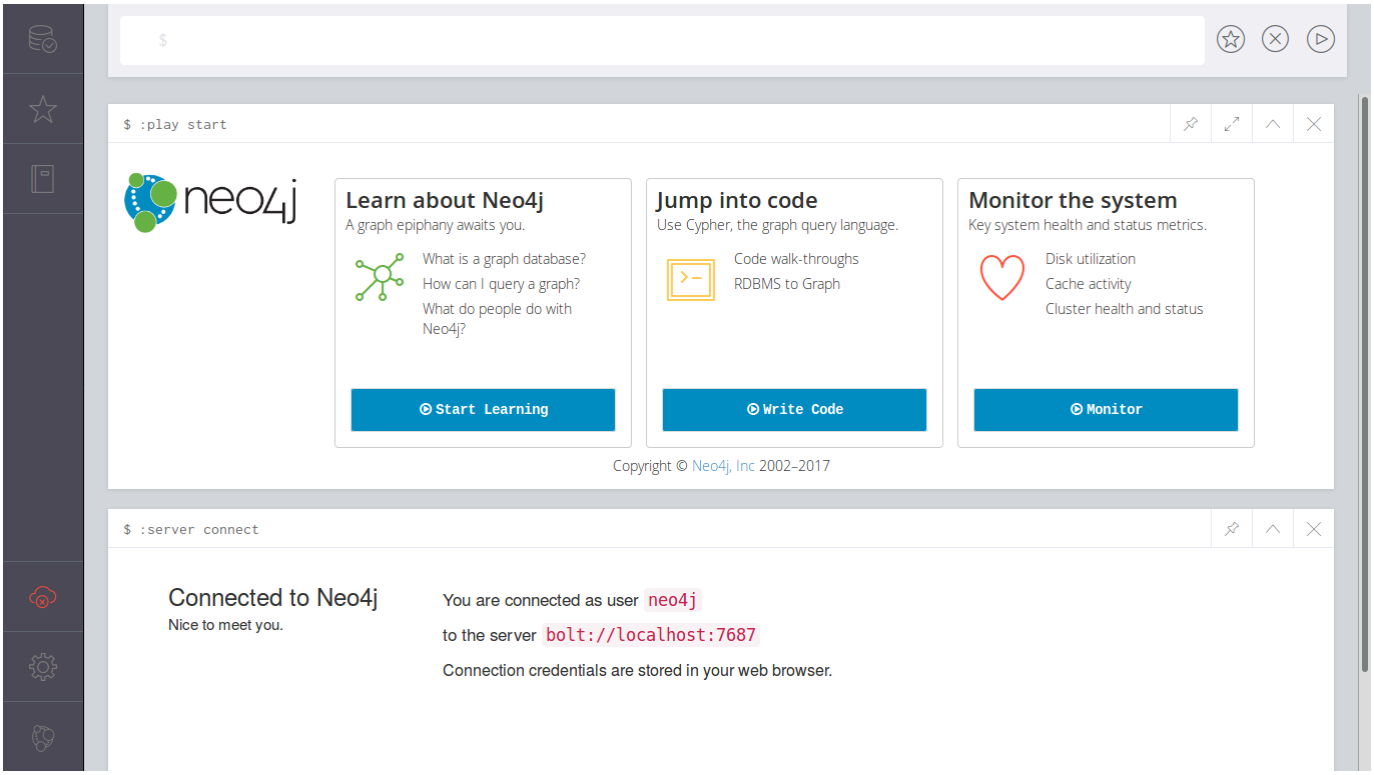
Запуск и установка Neo4j:

Ссылка для скачивания: https://neo4j.com/download-center/
Инструкции по установке и запуску для Windows:
https://coderlessons.com/tutorials/bazy-dannykh/uznaite-neo4j/neo4j-kratkoe-rukovodstvo
https://neo4j.com/docs/operations-manual/current/installation/windows/
Инструкции по установке для Linux:
https://neo4j.com/docs/operations-manual/current/installation/linux/debian/#debian-installation
https://1cloud.ru/help/linux/ustanovka-neo4j-na-ubuntu-debian
Для запуска на Linux введите следующую команду:

```
sudo service neo4j start
```

Инструкция по установке для macOS:
https://neo4j.com/docs/operations-manual/current/installation/osx/

После запуска перейдите по ссылке http://localhost:7474/browser/ . Вы должны увидеть интерфейс Neo4j Browser представленный на изображении .

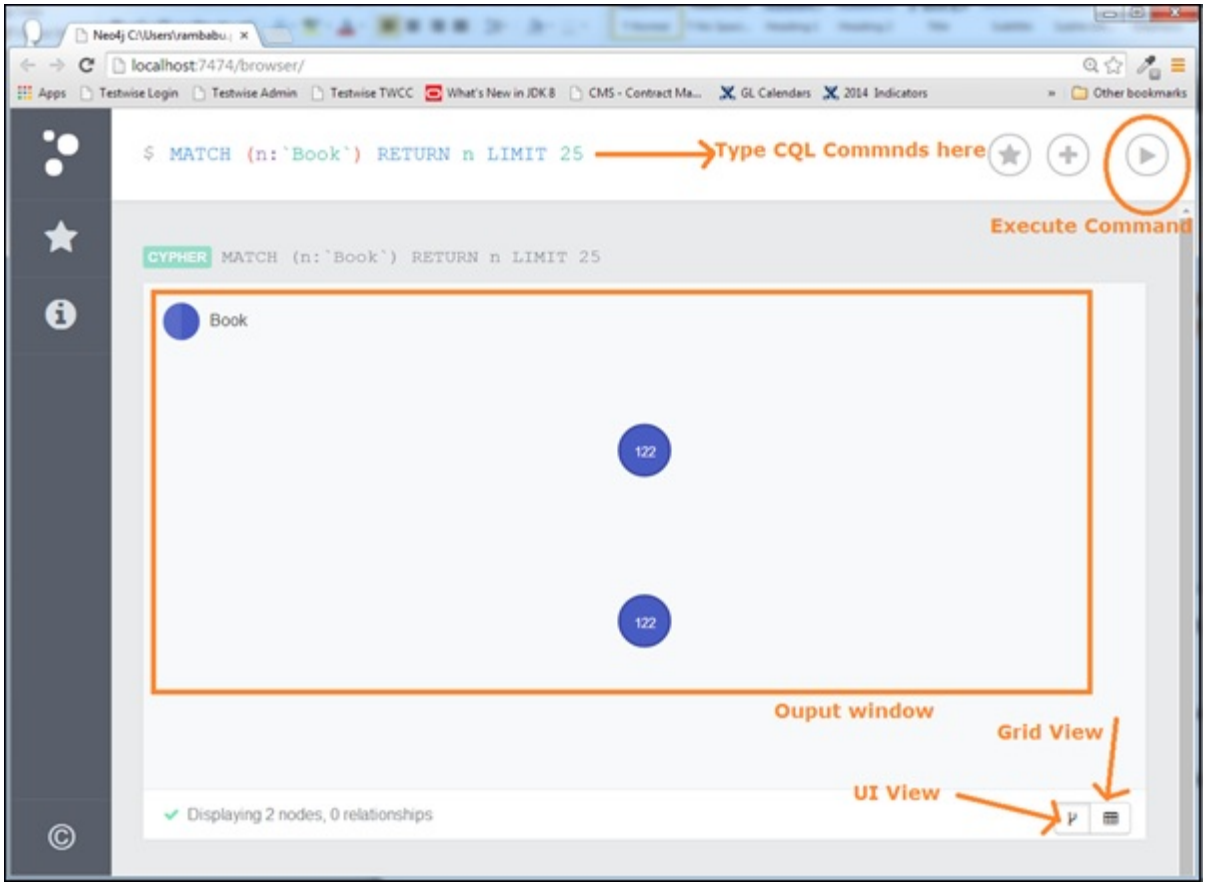


Для подключения к серверу потребуется ввести логин и пароль, заданные по умолчанию.
Логин: neo4j
Пароль: neo4j

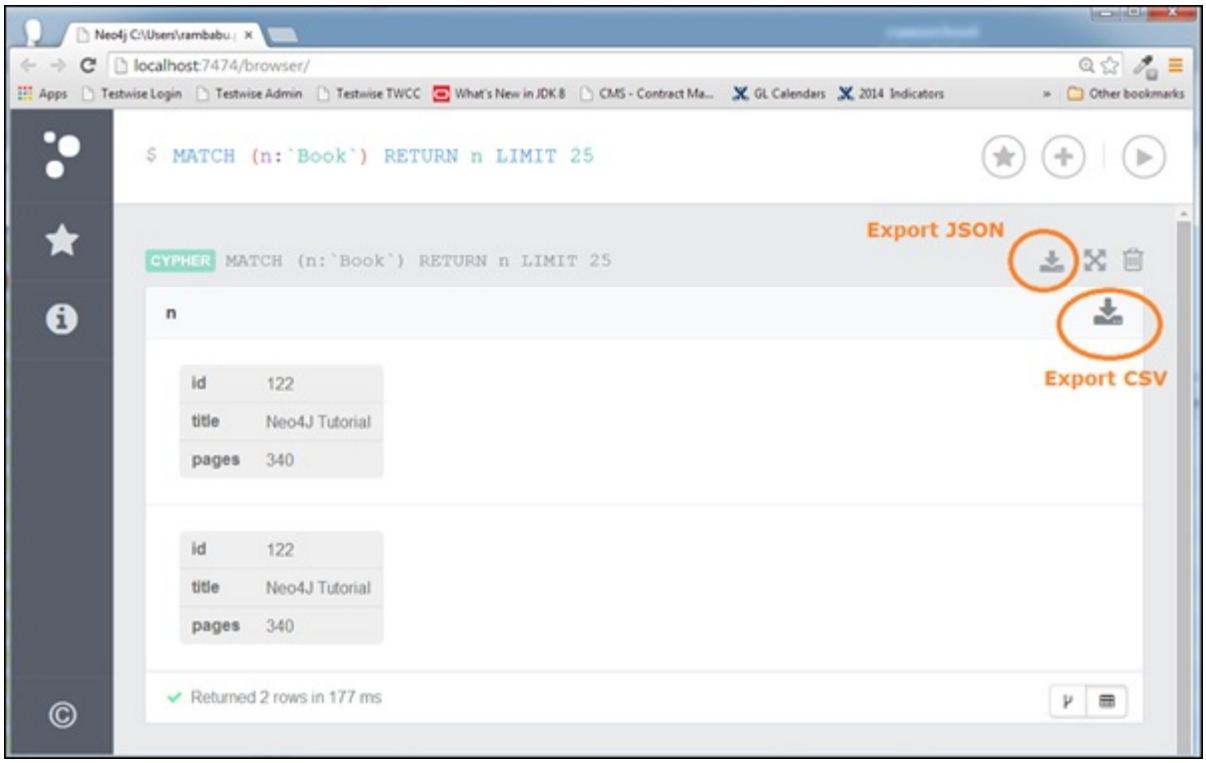
Возможности среды.

Neo4j Data Browser используется для выполнения команд CQL и просмотра выходных данных.

- Здесь нам нужно выполнить все команды CQL по подсказке доллара: «$»
- Введите команды после символа доллара и нажмите кнопку «Выполнить»(Execute Command) для запуска команд. Он взаимодействует с сервером базы данных Neo4j, извлекает и отображает результаты прямо под подсказкой доллара.
- Используйте кнопку «VI View» для просмотра результатов в формате диаграмм. Приведенная выше диаграмма показывает результаты в формате «UI View».
- Используйте кнопку «Вид сетки», чтобы просмотреть результаты в режиме сетки. Следующая диаграмма показывает те же результаты в формате «Grid View».
- Нажав на звездочку можно сохранить запрос, добавив его в избранное.

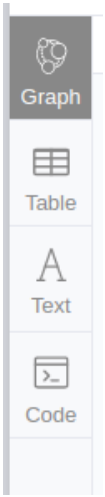


Когда мы используем «Вид сетки» для просмотра результатов нашего запроса, мы можем экспортировать их в файл в двух разных форматах.



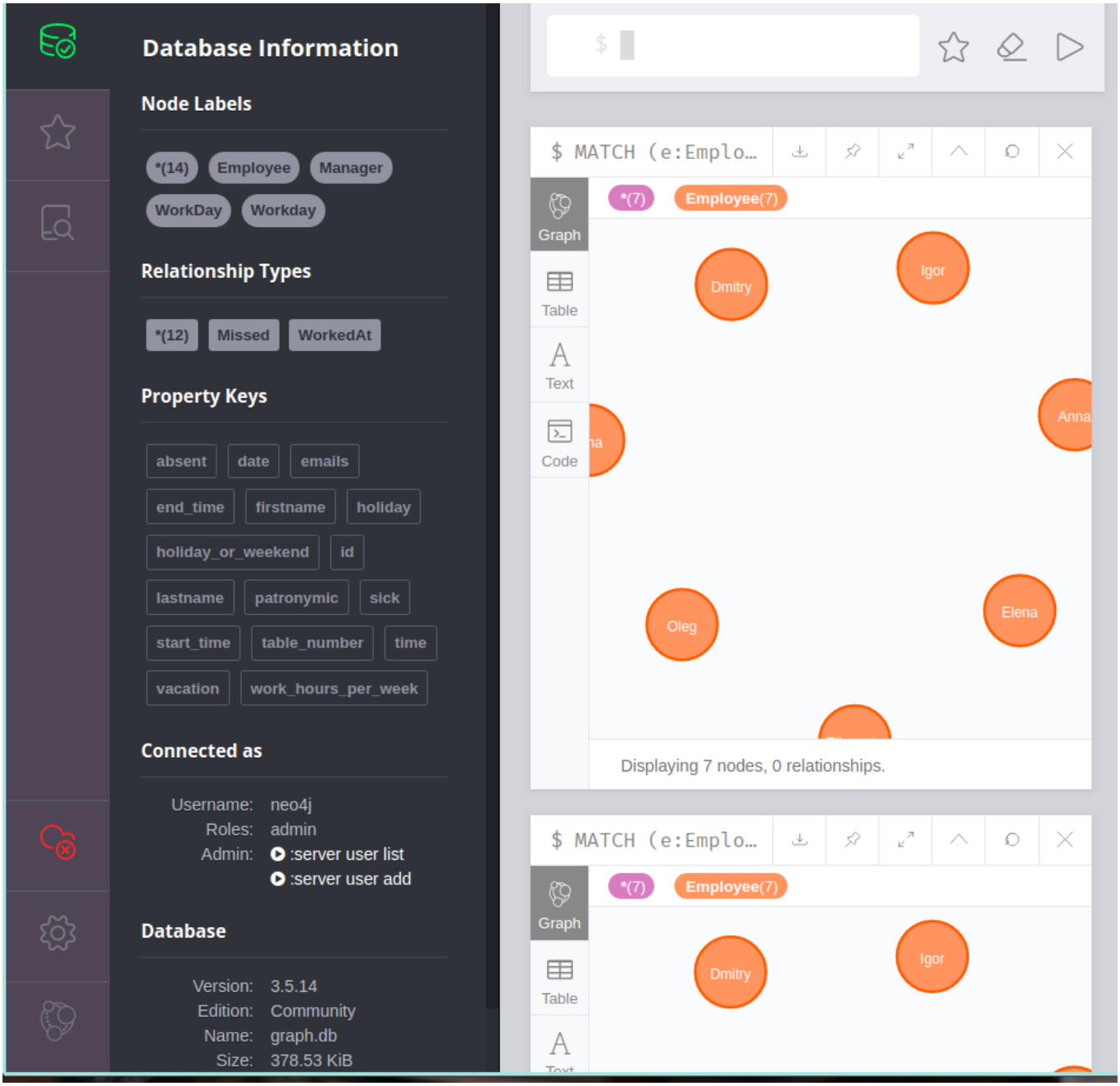
- Нажмите кнопку «Экспорт CSV», чтобы экспортировать результаты в формате файла CSV.
- Нажмите кнопку «Экспорт JSON», чтобы экспортировать результаты в формате файла JSON.
- Однако, если мы используем «UI View» для просмотра результатов нашего запроса, мы можем экспортировать их в файл только в одном 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.



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.



基本 CQL 命令。

Основные команды CQL.

Название	Описание
MATCH	Указание шаблона для поиска. 指定搜索模式
WHERE	Указание условия поиска. 指定搜索条件。
START	Задание начальных точек поиска. 设置搜索的起点。
LOAD CSV	Импорт данных из CSV файла. 从 CSV 文件导入数据。
CREATE	Создание узлов, отношений и свойств. 创建节点、关系和属性。
MERGE	Создание указанного шаблона в графе, если такового не существует. 如果指定的模板不存在，则在图中创建它。
SET	Обновление меток на узлах, свойств на узлах и отношений. 更新节点上的标签、节点上的属性和关系。
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.

Узлы, свойства и метки.

用于表示实体，但取决于图中的关系可以用来表示关系。 最简单的图表代表一个顶点。 一个顶点可能没有意义，也可能有一个或更多指定为属性的命名值。

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.

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

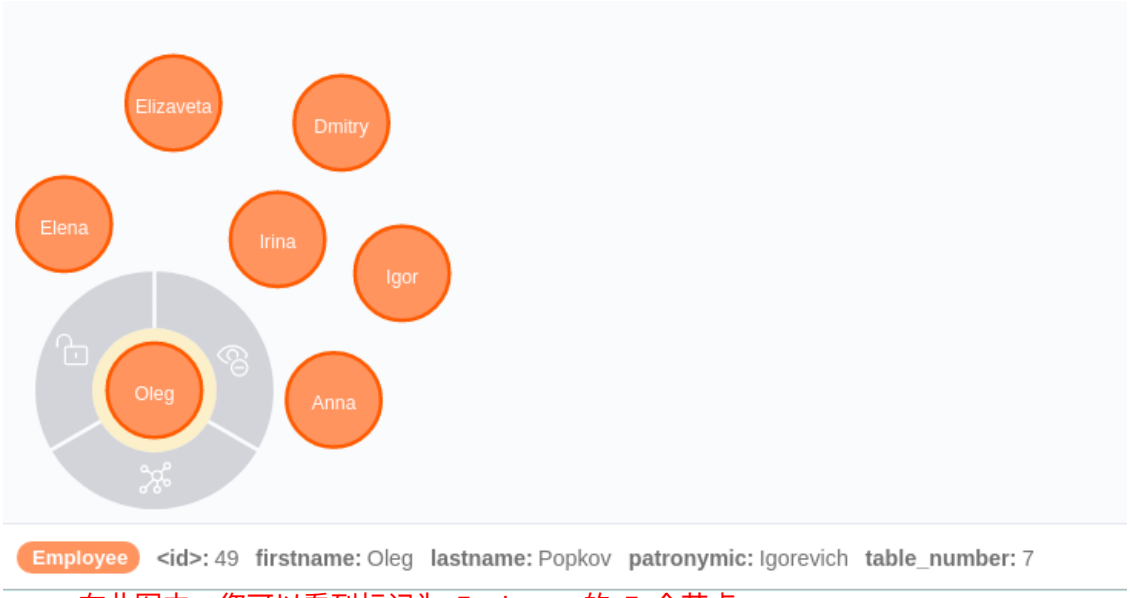
命名值，其中 name 是一个字符串。 支持的值：数字、字符串、二进制、先前类型的列表。

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.

表示已分组为集合的图。 标有相同标签的所有节点都属于同一个集合。 让写作更轻松对数据库的查询。 一个顶点可以用任意 20 个标签进行标注。 标签用于设置限制并为属性添加索引

Типы данных.

- Число, абстрактный тип, содержит подтипы Integer и Float
- String
- Boolean
- 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, a value любого типа).



在此图中，您可以看到标记为 Employee 的 7 个节点。

На данном рисунке можно увидеть 7 узлов, помеченных меткой Employee.

Свойствами выделенного узла являются id, lastname, firstname, patronymic и table_number.

突出显示的节点属性是 id、lastname、firstname、patonymic 和表号。

CREATE

Создание узла со свойствами

Синтаксис:

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

где node - узел

label - метка

key1, key2 — название свойства

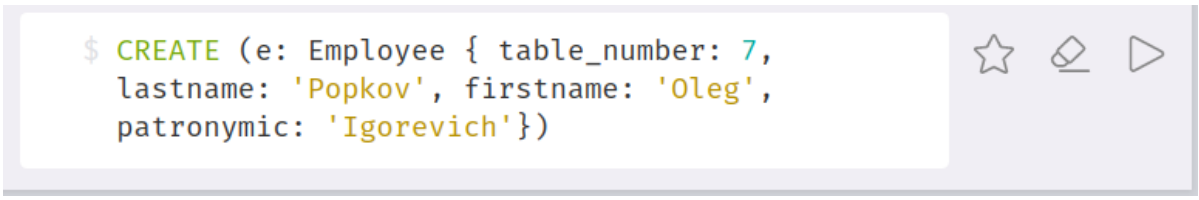
value — значение свойства

Прим. Обращение к меткам в Neo4j идет через символ двоеточия. Neo4j 中的标签通过冒号字符访问。

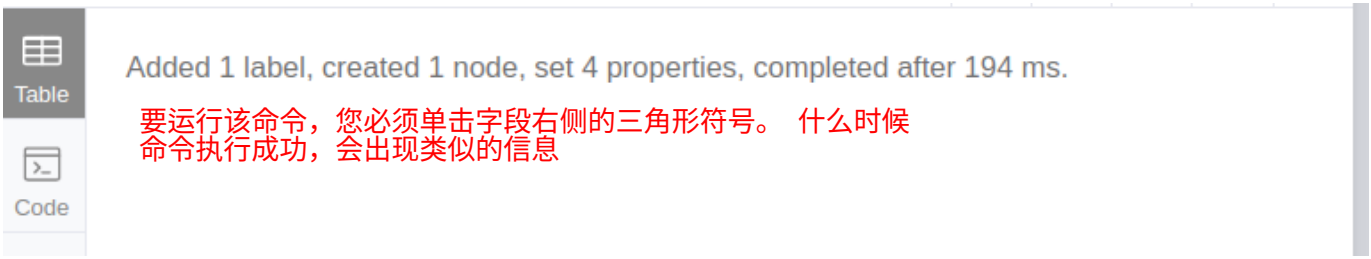
Пример:

Создание узла с меткой Employee(сотрудник) и свойствами: табельный номер и ФИО.

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



Для запуска команды необходимо нажать знак треугольника справа поля. В случае успешного выполнения команды появится подобное сообщение:



Документация**:** https://neo4j.com/docs/cypher-manual/current/clauses/create/

Вывод на экран содержимого БД. Команды Match/Where/Return.

MATCH

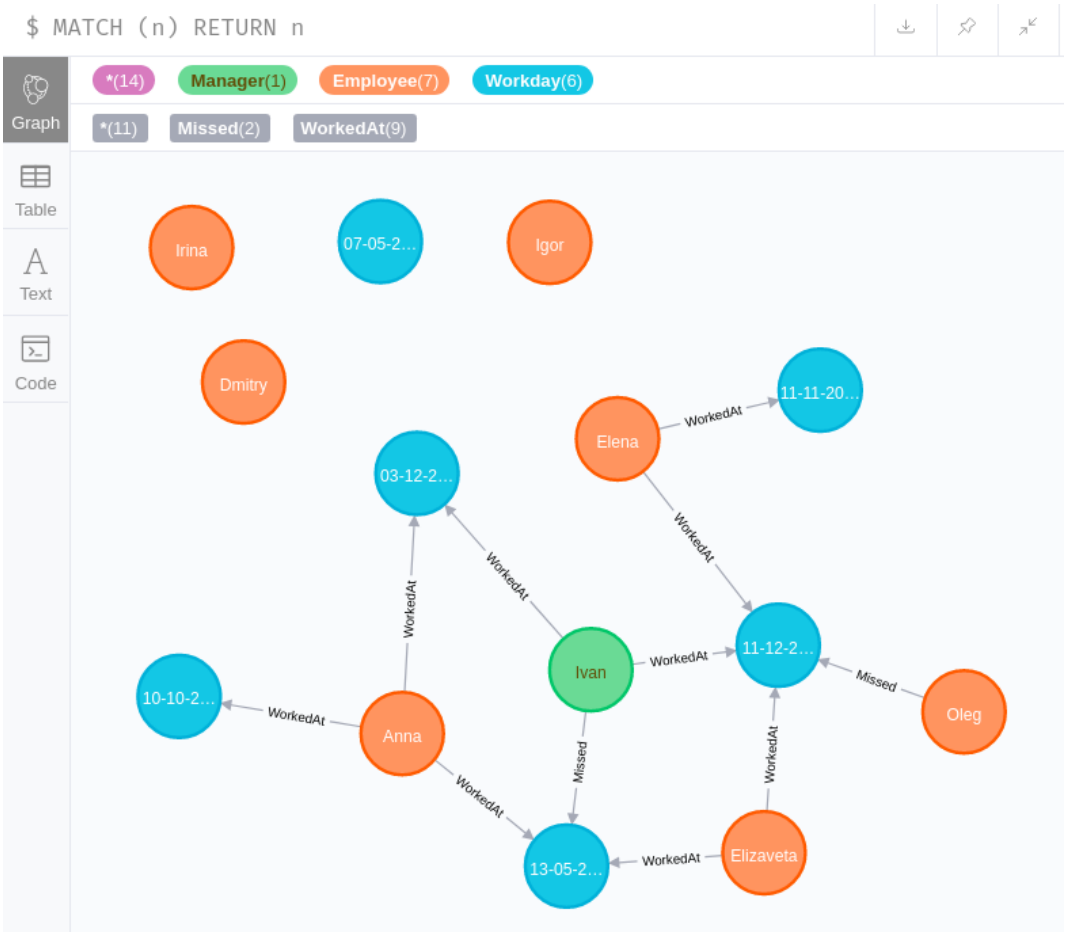
Ключевое слово, после которого следует шаблон, описывающий искомую информацию.

Синтаксис запроса котрый вернет все узлы в базе данных .

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

где n обозначает узел. 其中 n 表示一个节点。

关键字后跟描述正在搜索的信息的模式。
将返回数据库中所有节点的查询语法



Получение всех узлов под определенной меткой

Синтаксис: 获取特定标签下的所有节点
句法：

```
MATCH (node:Label1)

RETURN node
```

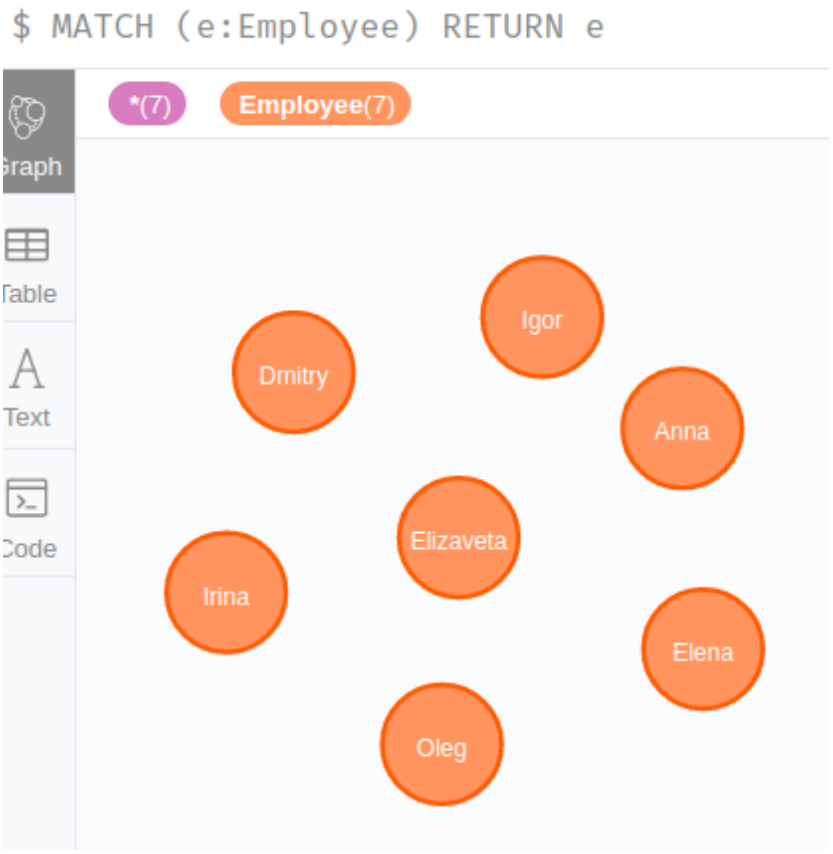
Получение узлов с несколькими метками: 获取具有多个标签的节点：

```
MATCH (node:Label1:Label2)

RETURN node
```

где node - узел
Label1, Label2 - метки

Пример:

Запрос, который выведет все узлы с меткой Employee. 将返回所有标记为 Employee 的节点的查询。



Документация**:** https://neo4j.com/docs/cypher-manual/current/clauses/match/

RETURN.

定义将返回的结果
请求执行

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

Возврат созданного узла.

Чтобы посмотреть созданный узел можно воспользоваться командой RETURN с CREATE.

返回创建的节点。
要查看创建的节点，可以使用 RETURN 命令
创造

Синтаксис:

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

где node - узел
label - метка
key1, key2 — название свойства
value — значение свойства

Пример:

Данная команда создаст узел e с меткой Employee и свойствами: таблельный номер и ФИО, и выведет его на экран.

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

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.

以下查询显示了如何推断特定节点属性。
在 Neo4j 中，您可以通过点字符访问节点和关系的属性。
此查询将返回所有标记为 Employee 的节点的名字、姓氏、父名属性。

```
$ 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"

Документация: https://neo4j.com/docs/cypher-manual/current/clauses/return/

过滤
WHERE 是一个关键字，您可以在其后添加其他
对模板施加的限制，或过滤结果。
您可以对属性值、标签值和
复杂条件（例如，使用 AND、OR、CONTAINS、常规
表达式、关系模板等 - 请参阅“以 Cypher 语言查询数据库”部分和
文档）。
您可以过滤节点和关系。

WHERE

Фильтрация

WHERE - ключевое слово, после которого можно добавить дополнительные ограничения, накладываемые на шаблон, либо 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.

Синтаксис

Фильтрация по значению свойства 按属性值过滤

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

Фильтрация по значению метки 按标签值过滤

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

где a — узел или отношение
Label — название метки узла или отношения
property — название свойства

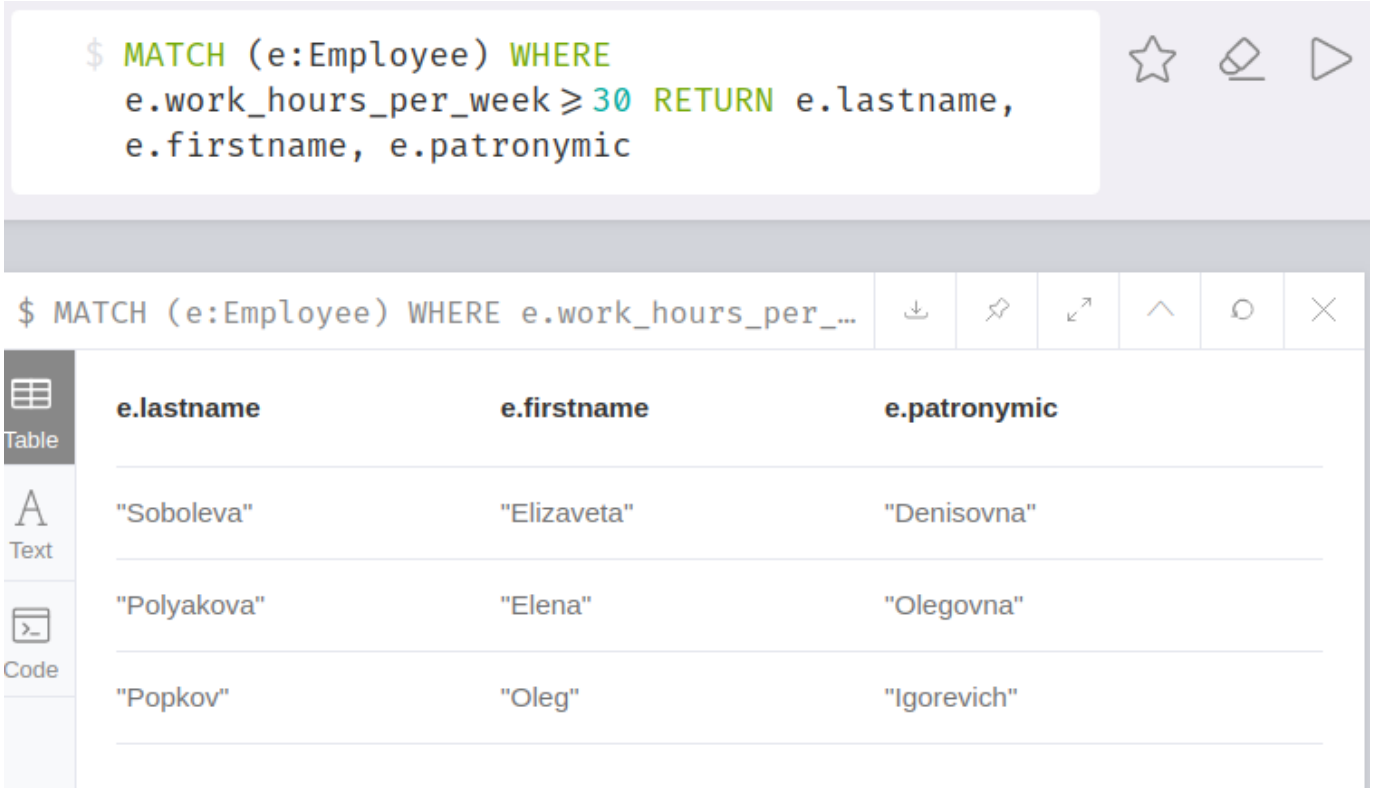
value — значение свойства

将显示每周工作超过 30 小时的员工全名的查询。 那
e 带有 Employee 标签的节点被过滤掉，过滤条件为
work hours per week 属性值必须至少为 30。

Пример:

Запрос, который выведет ФИО сотрудников, которые работают от 30 часов в неделю. 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
```



Документация: https://neo4j.com/docs/cypher-manual/current/clauses/where/

Отношения между узлами

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:

关系类型 -[:KNOWS|:LIKE]->
? 变量名 -[rel:KNOWS]-> 冒号前
? 附加属性 -[{since:2010}]->
? 可变长度路径的结构信息 -[:KNOWS*..4]->
设置关系

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

Задание отношений:

描述连接的最简单方法是在两个节点之间使用箭头。 和
使用这种技术，您可以描述连接必须存在及其方向。
(a)->(b) 或 (a)--(b) 如果链接的方向不重要。

- 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)

关系也可以被命名。 在这种情况下，一对正方形
括号之间带有标识符，将箭头一分为二。 例如：(a)-
[r]->(b)

- Подобно меткам узлов, отношения могут иметь типы. Для описания связи заданного типа можно поступить следующим образом: (a)-[r:REL_TYPE]->(b)

像节点标签一样，关系也可以有类型。 描
述给定类型的连接
你可以这样做：(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)

关系只能有一种类型，但您可以
指定类型以便关系具有一种类型
使用分隔符“|”键入：(a)-[r:
TYPE1|TYPE2]->(b)

- Имя связи всегда может быть опущено: (a)-[:REL_TYPE]→(b)

关联名称总是可以省略：(a)-[:REL TYPE]→(b)

现有节点之间的链接可以使用 MATCH 和
合并。
MERGE 命令是 CREATE 命令和 MATCH 命令的组合。
MERGE 命令在图中搜索给定的模式。 如果存在，则返回
结果。 如果它在图中不存在，那么它会创建一个新的节点/关系并
返回结果。

MATCH, MERGE

Создание отношений между несколькими узлами (с параметрами).

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.

Синтаксис

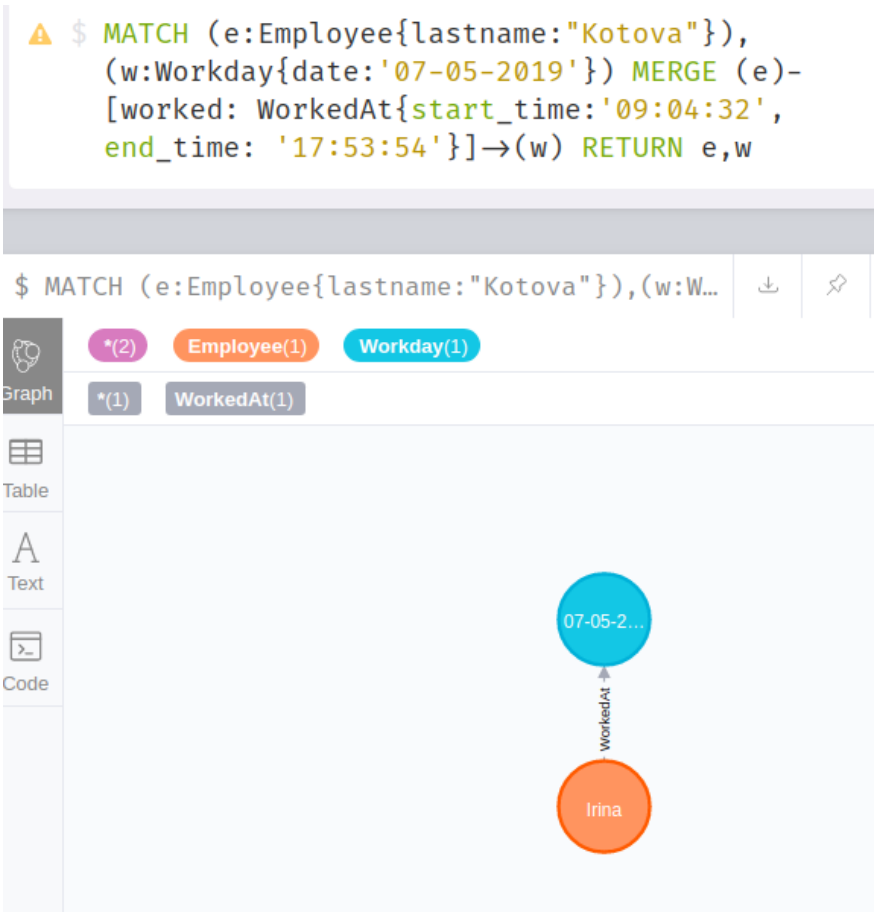
```
MATCH (node1:Label1 {attribute: attribute_value}),(node2: Label2
{attribute: attribute_value}) MERGE (node1)-[rel:RelationName
{rel_attribute:rel_attribute_value}]→(node2) RETURN node1, node2
```

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

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.

创建使用标签 WorkedAt 与属性 start_time、end_time 之间的关系
现有节点 e 带有标签 Employee (employee)，姓氏 Kotova 和 w -
工作日（工作日），日期为 07-05-2019。 如果数据库不包含具有指定
姓氏或指定的工作日，neo4j 将创建一个新节点。 沟通方向——从
工作日的员工。



Это не единственный вариант создания связей между узлами, более подробно можно найти по ссылке: https://neo4j.com/docs/cypher-manual/current/clauses/create/#create-relationships

Фильтрация по узлам, отношениям, меткам и связям. 按节点、关系、标签和链接过滤。

Фильтрация по узлам:

Синтаксис

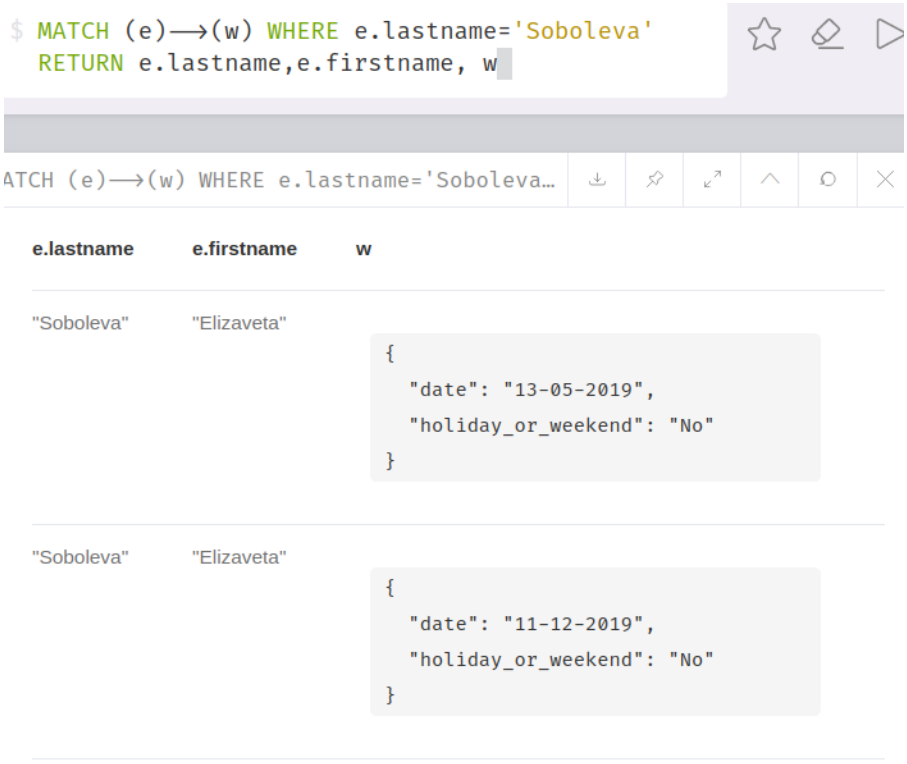
```
MATCH (node) WHERE node.attribute=value RETURN node
```

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.

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



Фильтрация по меткам:

Синтаксис

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

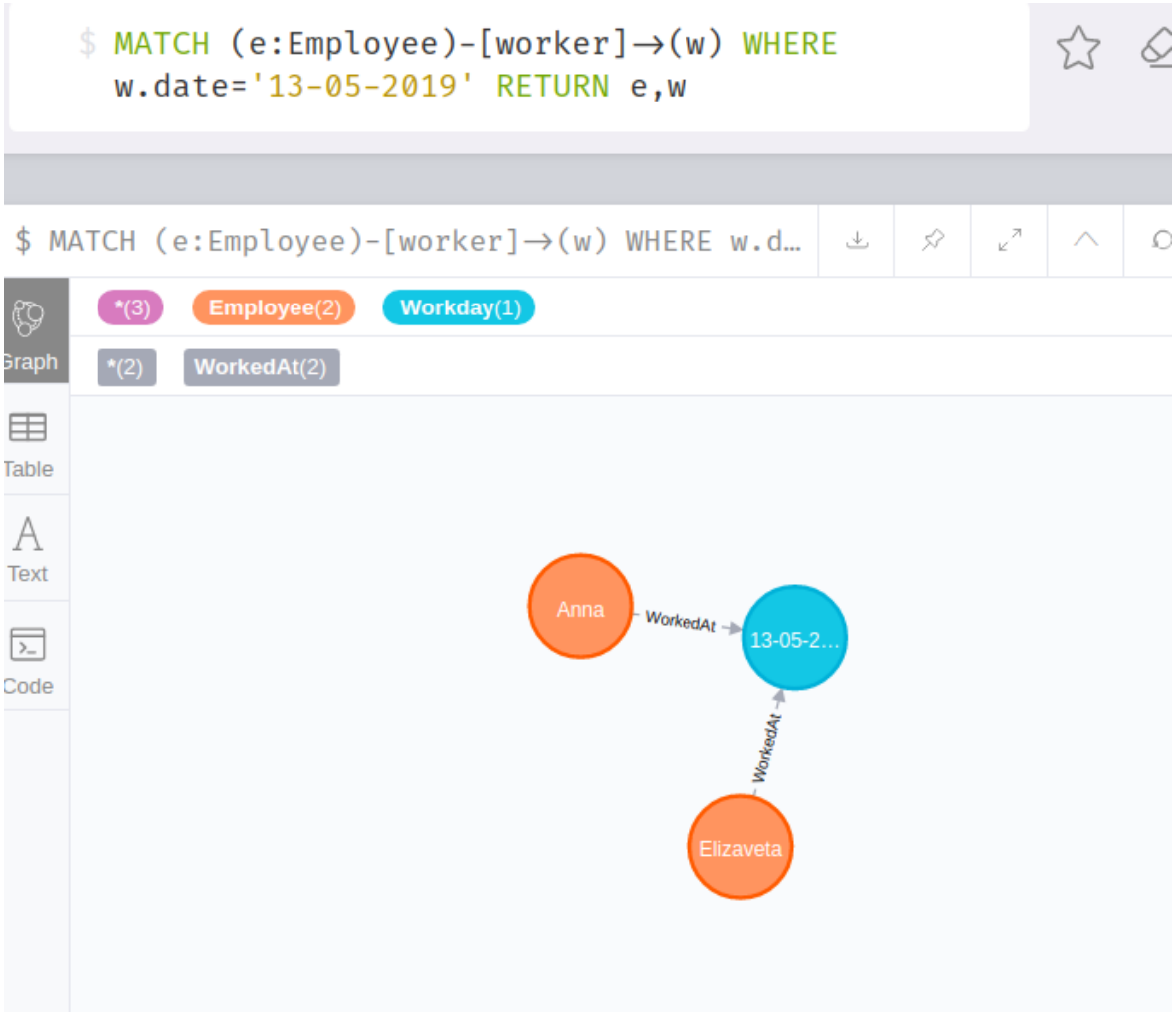
где node – узел

Label— метка узла

Пример:

Пример: Запрос, который выводит все узлы e с меткой Employee и узлы w, у которых есть свойство – дата 13-05-2019.

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



Фильтрация по связям:

Синтаксис

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

где node1, node2 - узлы,

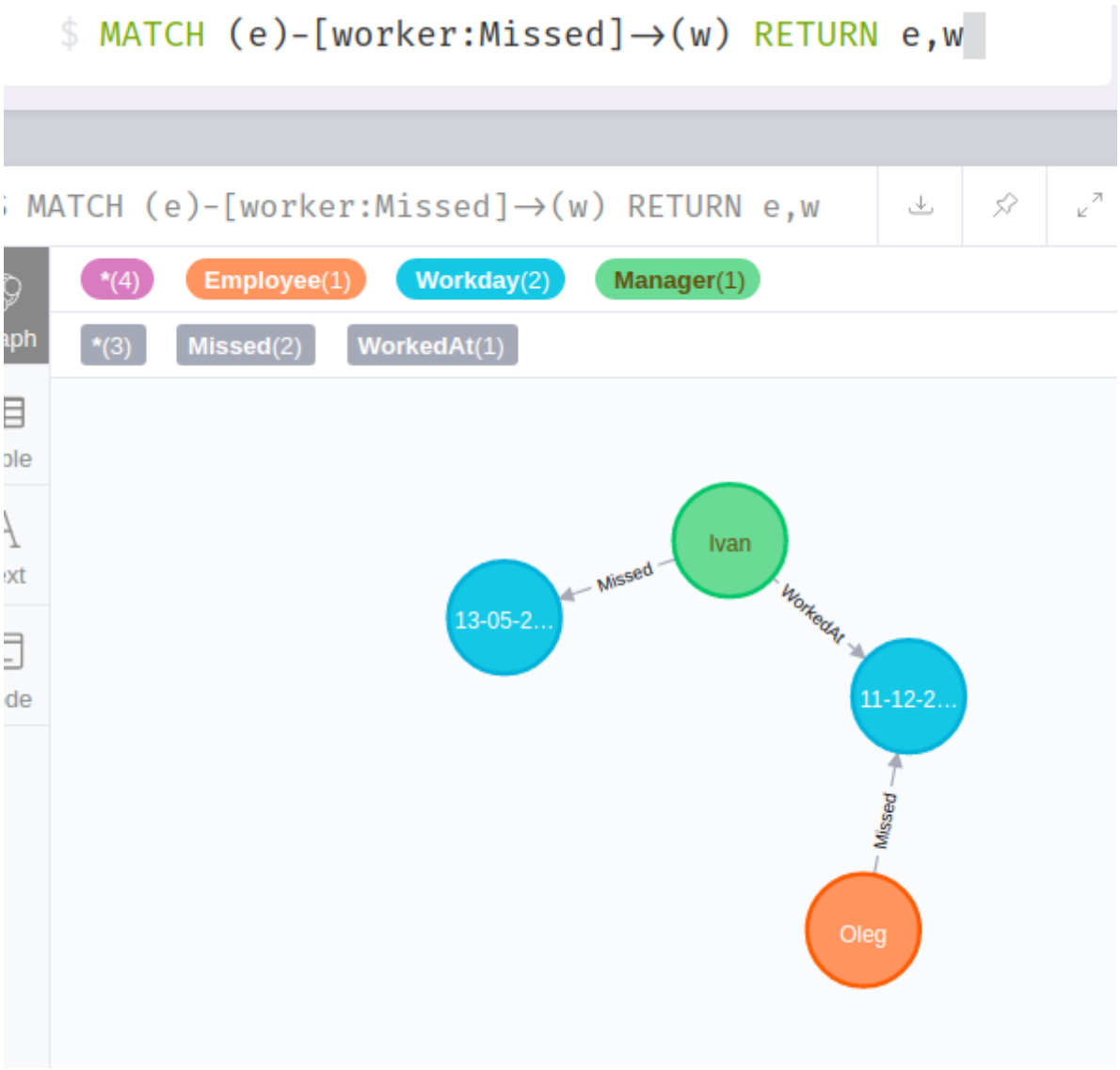
Label1, Label2 — метки узлов

rel – отношение

RelationName – метка отношения

Пример:

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



Более подробно про фильтрацию о связям можно почитать здесь:

https://neo4j.com/docs/cypher-manual/current/clauses/match/#relationship-basics

Запросы к БД на языке Cypher

Условие NOT NULL

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

Синтаксис

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

где node - узел
attribute — название свойства узла

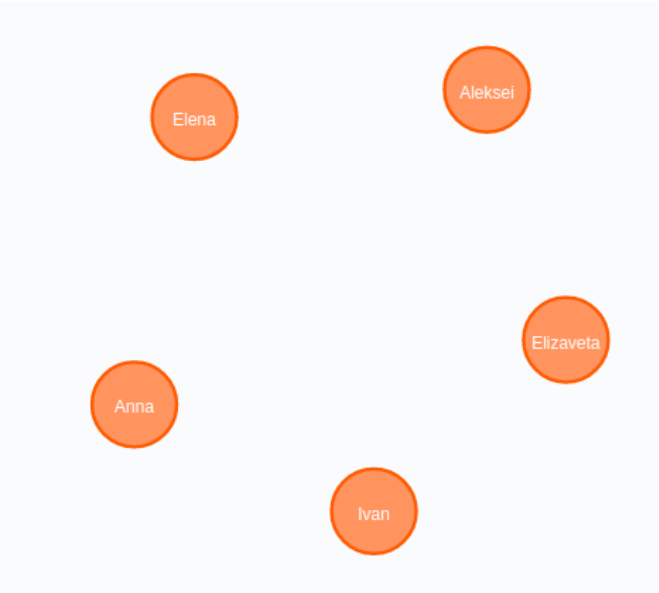
Пример:

Создадим узел e с меткой Employee и свойствами: табельный номер, фамилия, имя, кол-во рабочих часов в неделю и не зададим ему 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 вывеенных сотрудников нет сотрудников с именем Igor.

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



Операторы AND, OR

Синтаксис

AND

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

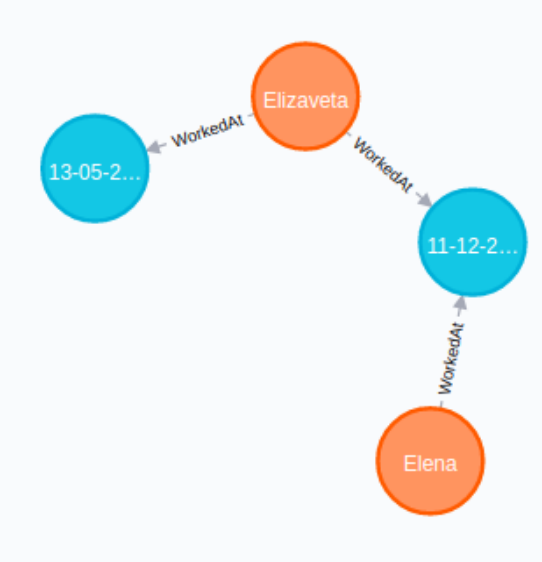
где node - узел,
attribute1 attribute2 — название свойств узла
value1, value2 — значения свойств узла

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.

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



Подробнее об операторах: https://neo4j.com/docs/cypher-manual/current/syntax/operators/

Сортировка ORDER BY

Синтаксис
Сортировка по свойству 按属性排序

```
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.

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"

Подробнее: https://neo4j.com/docs/cypher-manual/current/clauses/order-by/

Условие на направление отношения 关系方向的条件

Синтаксис

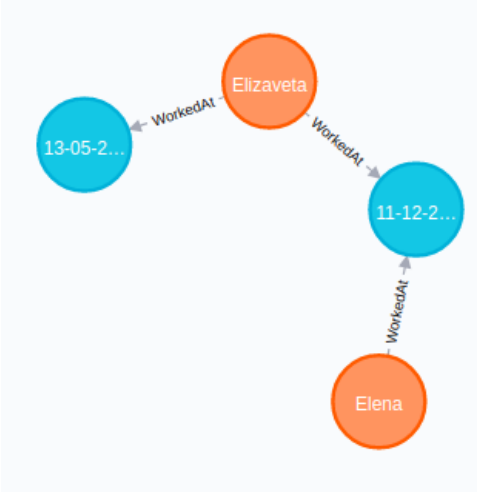
```
MATCH (node1:Label1)-->(node2:Label2) RETURN node1, node2
```

где node1, node2 — узлы
Label1, Label2 - метки

Пример

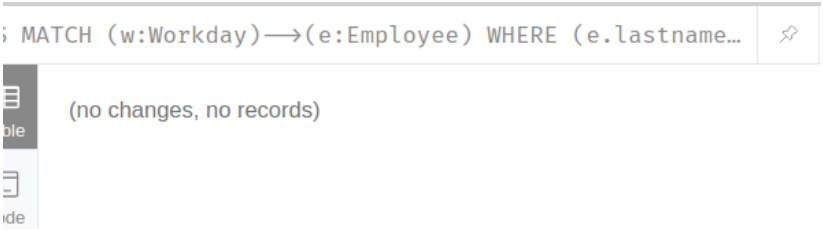
Запрос, выводящий все узлы с меткой Employee со свойствами lastname равными Soboleva или Polyakova и связанные с ними узлы с меткой Workday со свойством date равным 11-12-2019 или 13-05-2019. Направление связи от узлов с меткой Employee к узлам с меткой Workday.

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



Тот же запрос, но с обратным направлением связи не выведет ничего.

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



Параметры отношения

Синтаксис

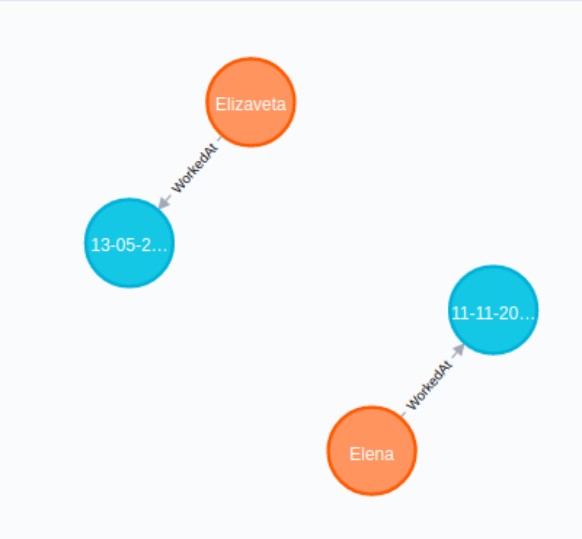
```
MATCH (node1)-[rel:RelationName]->(node2)  WHERE
rel.attribute='value'  RETURN node1, node2
```

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'.

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



Операторы CRUD

Создание отношения между новыми узлами.

Синтаксис

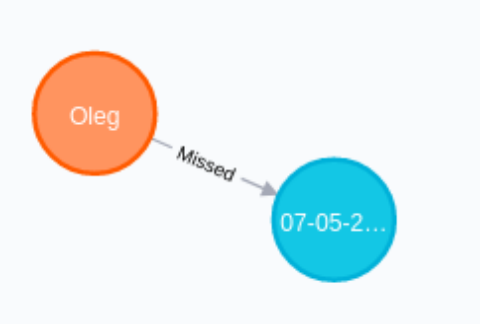
```
CREATE (node1:Label1 {attribute1: 'value1',  ...})-[rel:
RelationName {attribute2: 'value2', ...}]->(node2:Label2
{attribute3: 'value3',...})
```

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.
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



Создание отношения между существующими узлами.

Синтаксис

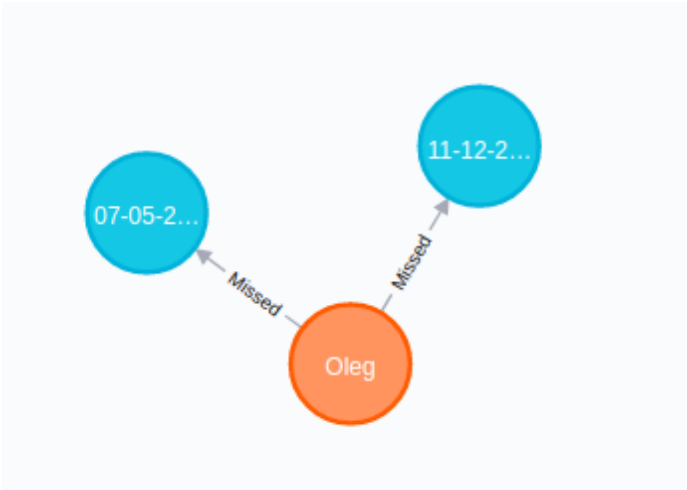
```
MATCH (node1:Label1 {attribute1: value1}),(node2: Label2
{attribute2: value2}) MERGE (node1)-[rel:RelationName
{attribute:value3}]→(node2) RETURN node1, node2
```

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

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.

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



DELETE

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

Синтаксис

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

где n - узел

```
(empty result)
```

```
0 rows, Nodes deleted: 4 Relationships deleted: 2
```

Удаление одного узла и всех его связей 一个节点及其所有链接

Синтаксис:

```
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
```

Удаление только связей

Синтаксис

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

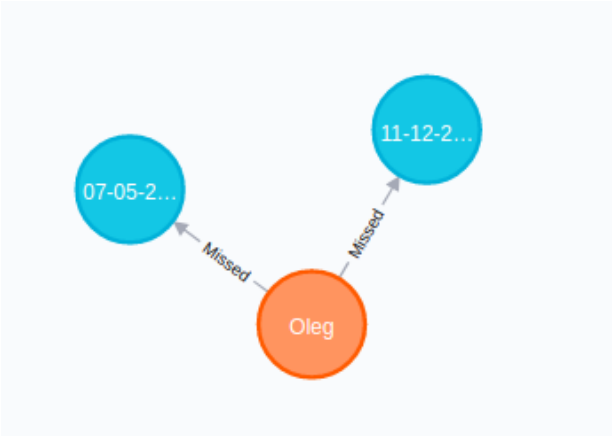
где node – узел
attribute – названия свойства узла
value – значения свойства узла
rel – отношение
RelationName – метка отношения

```
(empty result)
```

```
0 rows, Relationships deleted: 2
```

Подробнее про удаление: https://neo4j.com/docs/cypher-manual/current/clauses/delete/

Пример
До удаления

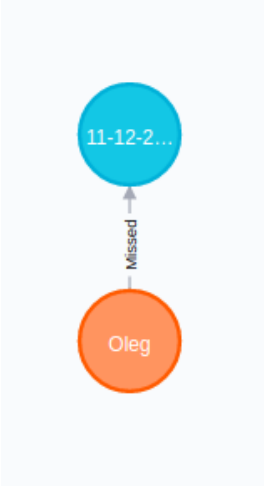


Удаление связи между узлом e с меткой Employee с фамилией Popkov и узлом w с меткой WorkDay с датой 07-05-2019.

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:

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

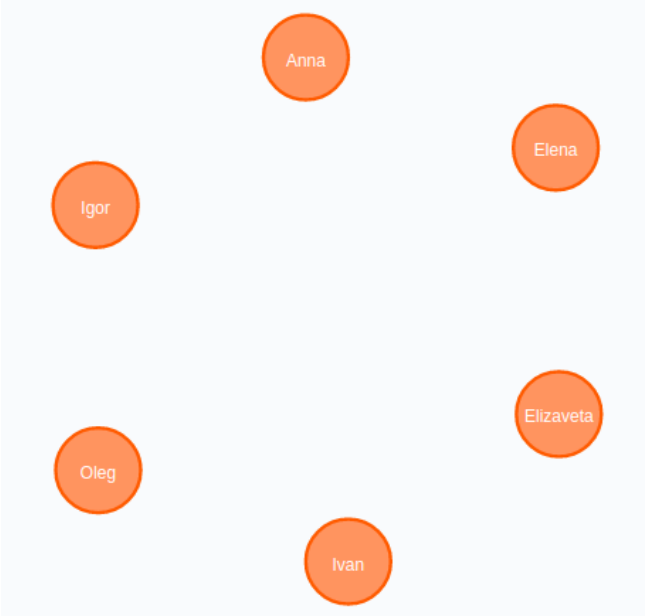


Пример удаления узла:

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

Проверка, что узел удален

MATCH (e:Employee) RETURN e



Удаление и изменение свойств и меток.

SET- Изменение свойства

Синтаксис

```
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.

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

Подробнее: https://neo4j.com/docs/cypher-manual/current/clauses/set/

REMOVE - Удаление свойства

Синтаксис:

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

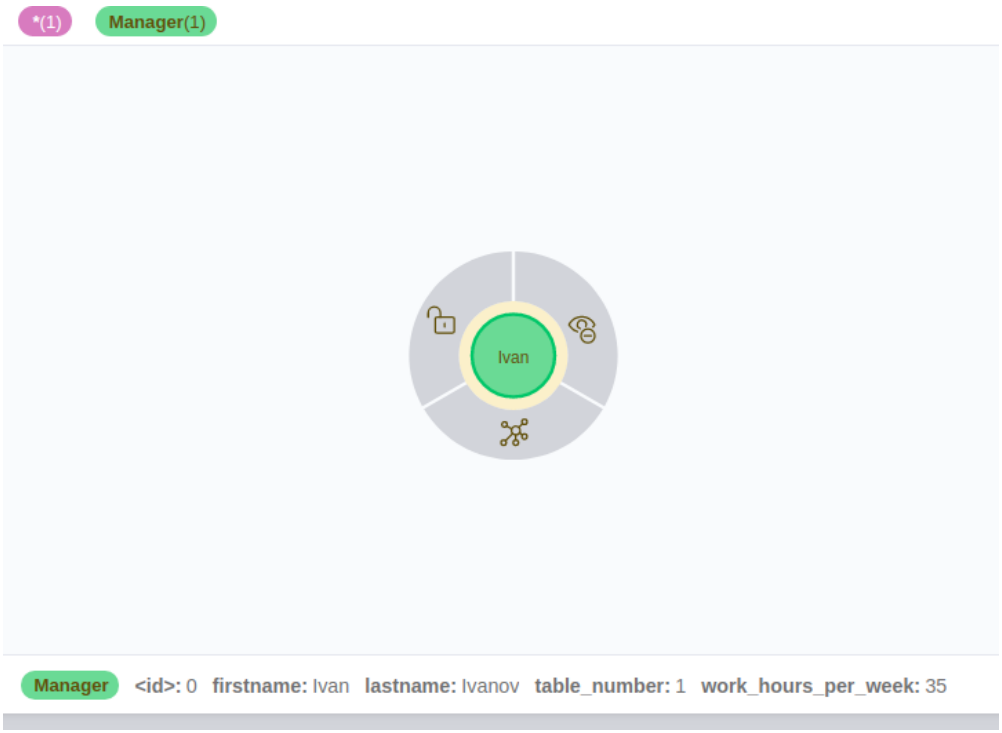
либо

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

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



Подробнее**:** **https://neo4j.com/docs/cypher-manual/current/clauses/remove/**

Изменение метки

Синтаксис

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

SET node:Label2

RETURN node
```

где node – узел

Label1, Label2 – старая и новая метка

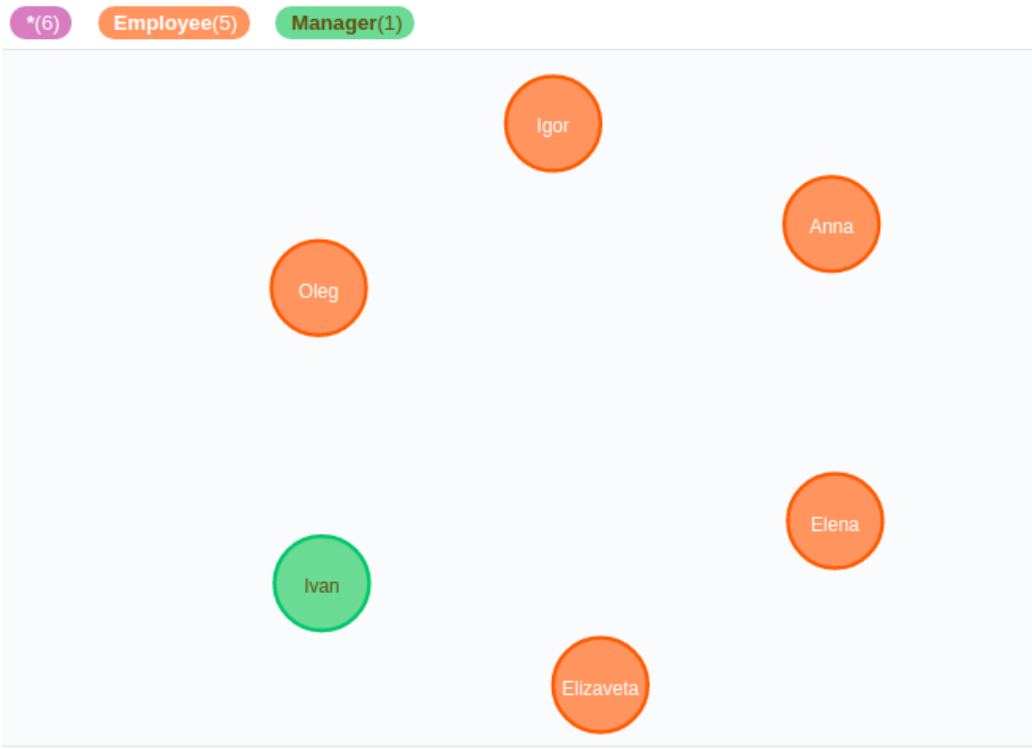
attribute – названия свойства узла

value – значения свойства узла

Пример:

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

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



Удаление метки

Синтаксис

```
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.

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

UNION

Используется для объединения результатов нескольких запросов. 用于组合多个查询的结果。

Синтаксис

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

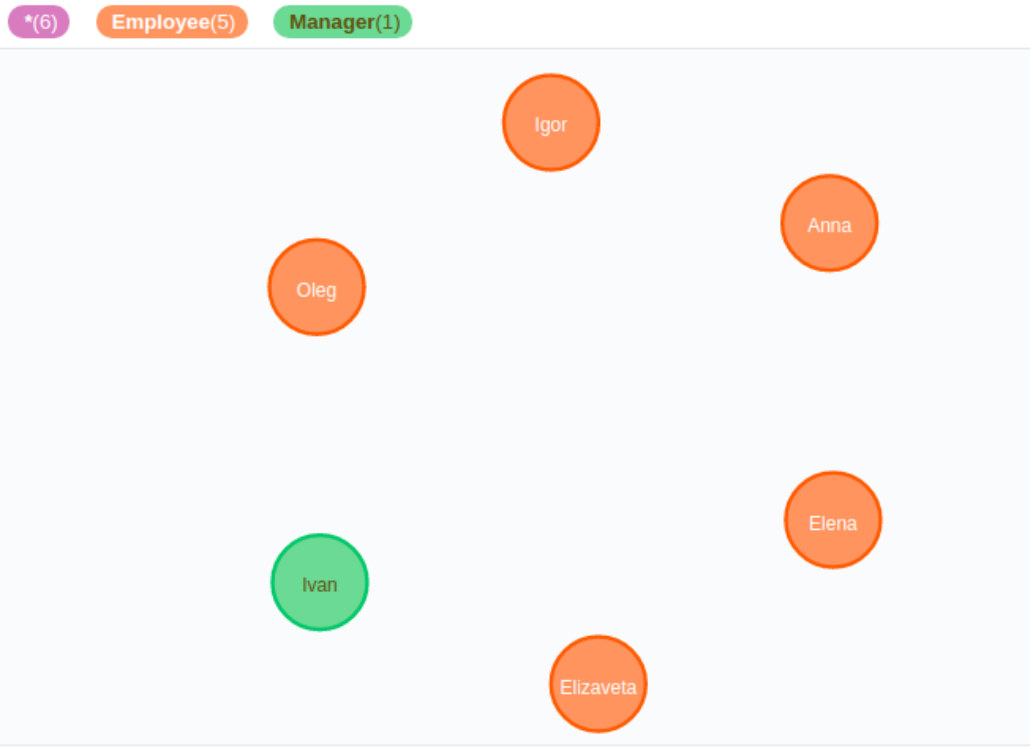
где node1, 2 – узлы
Label1, 2 – метки

Подробнее: https://neo4j.com/docs/cypher-manual/current/clauses/union/

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.

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



MERGE

Проверяет существование паттерна в базе данных. Если не существует, то команда создаст его.

Синтаксис

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

где node – узел
Label – метка
attribute – названия свойства узла
value – значения свойства узла

Подробнее: https://neo4j.com/docs/cypher-manual/current/clauses/merge/

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.

Повторим запрос для этого же узла.

*MER*GE (e:Employee{lastname:'Novikov',firstname:'Dmitry', table_number:8})

Так как узел уже существует, то новый не создался.

(no changes, no records)

Расширенные запросы к БД

Агрегирование.

Синтаксис

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

где node – узел
Label – метка
attribute – названия свойства узла
function_name – имя функции

Подробнее: https://neo4j.com/docs/cypher-manual/current/functions/aggregating/
Пример:
Запрос, который выведет количество сотрудников. 将显示员工人数的查询。
MATCH (e:Employee) RETURN count(e) as e_count

e_count
7

Встроенные функции (строковые или иные). 内置函数（字符串或其他）。聚合函数

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

返回属性的最大值

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

返回属性的百分位数（度量，在
占总值的百分比
等于或小于这个度量。 ）

返回属性的最小值

返回值的标准差
属性

返回属性值的总和

Строковые функции. 字符串函数。

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.

将搜索中指定的所有表达式替换为
替换中指定的表达式

返回一个大写字符串。

trim(original)	Удаляет пробелы в начале и конце строки. 从字符串的开头和结尾删除空格

Подробней о строковых функциях: https://neo4j.com/docs/cypher-manual/current/functions/string/
Обо всех строковых функциях: https://neo4j.com/docs/cypher-manual/current/functions/
Синтаксис для функции substring:

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

где original — это выражение, возвращающее строку.
Пример:
Запрос, который выведет дату в формате „day «» month «» year «»“

```
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"

关系模式。

Шаблоны отношений.

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 так же 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.
- Направление связи задаётся с помощью стрелок следующим образом:
  - ()-[]->()
  - ()<-[]-()
  - ()-[]-() - для случая, когда направление связи не важно
  - В случае, когда нет необходимости накладывать дополнительные ограничения на связи между вершинами, можно опустить квадратные скобки в указании связи: ()--()

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.

Синтаксис:

Если не важна направленность: 如果方向不重要：

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

Если важна направленность.

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

Для указания какого-то конкретного типа отношения. 指示特定类型的关系

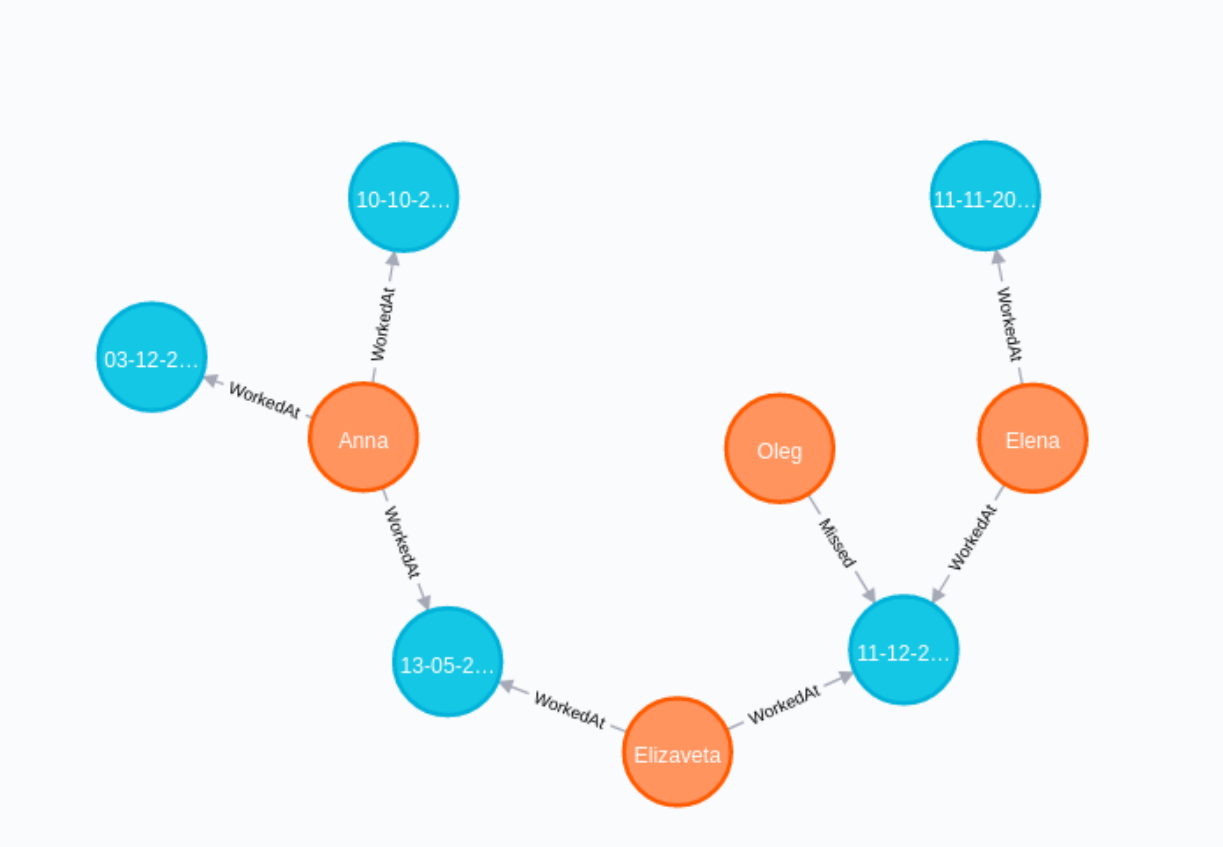
```
(a)-[r:REL_TYPE]->(b)
```

где a, b – узлы
r – отношение
REL_TYPE – метка отношения

Подробнее: https://neo4j.com/docs/cypher-manual/current/syntax/patterns/#cypher-pattern-relationship

Пример:
Запрос, который выведет все узлы с метками Employee и Workday, которые связаны какой-либо связью направленной от Employee к Workday.

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



Удаление дубликатов.

Синтаксис:

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

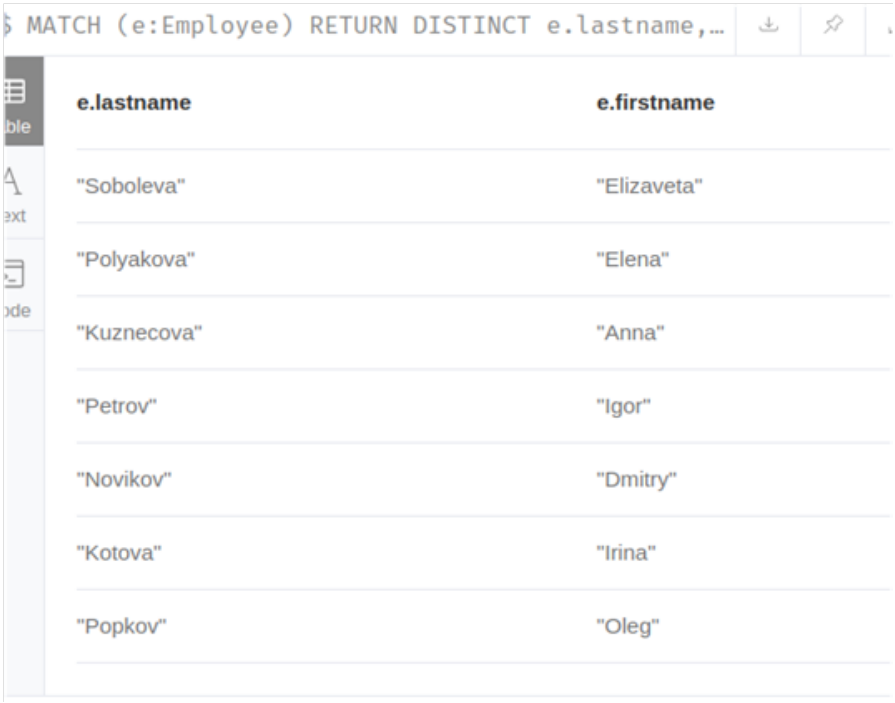
где node – узел
Label – метка
attribute – названия свойства узла

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. Видим, что запрос вывел 2 Анн Кузнецовых.
MATCH (e:Employee) RETURN e.lastname, e.firstname

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

Выведем только уникальные фамилии и имена. Видим, что теперь вывелась 1 Анна Кузнецова.

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



LIMIT

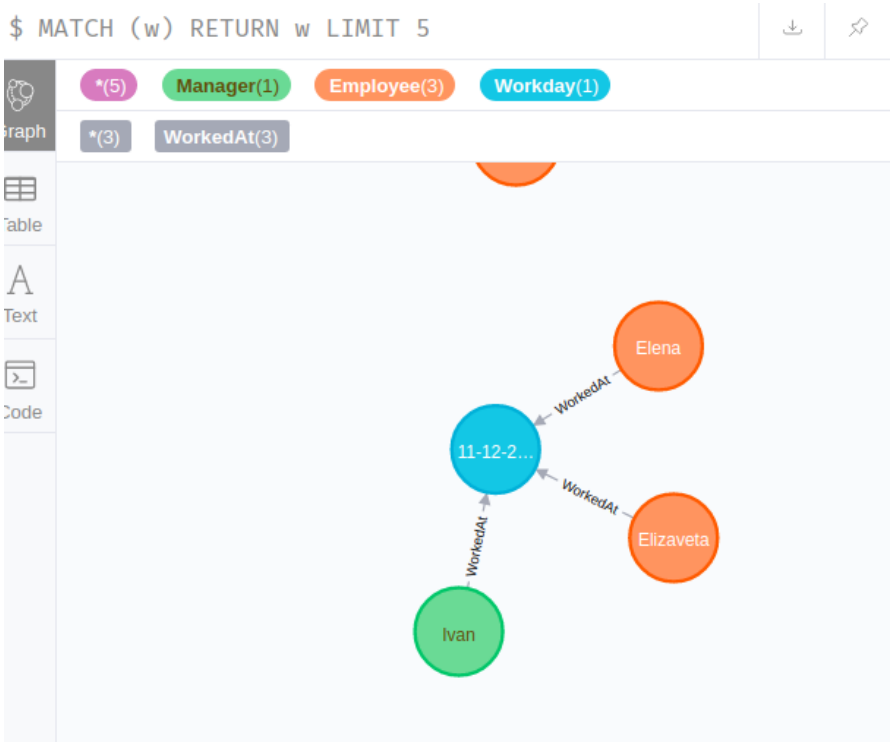
Ограничивает кол-во строк в выводе.

Синтаксис:

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

где node – узел

Подробнее: https://neo4j.com/docs/cypher-manual/current/clauses/limit/
Пример:
Вывод 5 узлов.
MATCH (w) RETURN w LIMIT 5



SKIP

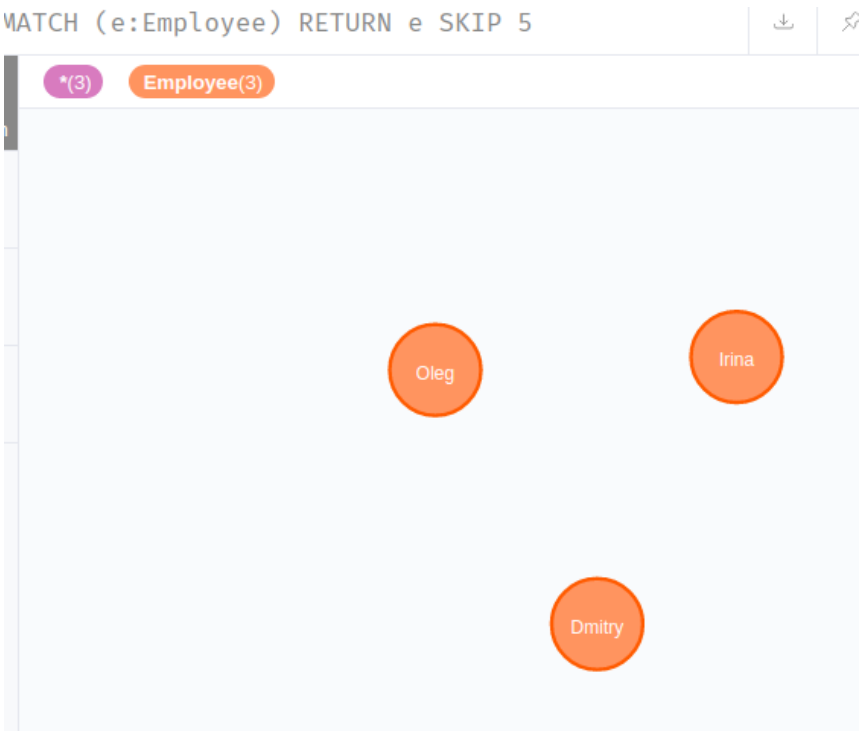
Определяет, сколько узлов необходимо пропустить. 指定要跳过的节点数。
Синтаксис:

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

где node – узел
x – количество узлов, которые необходимо пропустить.

Подробнее: https://neo4j.com/docs/cypher-manual/current/clauses/skip/
Пример:
Запрос, который пропустить первые 5 узлов с меткой Employee и выведет 6, 7 и 8 сотрудников

MATCH (e:Employee) RETURN e SKIP 5



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.

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:

- Лучше указывать имя индекса при его создании, иначе он получит автоматически сгенерированное имя.
- Имя индекса должно быть уникальным как для индексов, так и для ограничений.
- 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 — 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 литрально, передан через параметры или быть определен индексным поиском.

Синтаксис:

Создание индекса для 1 свойства.

```
CREATE INDEX [index_name]

FOR (node:Label)

ON (node.attribute)
```

где node – узел
Label – метка
attribute – название свойства узла
index_name – название индекса

Создание индекса для нескольких свойств.

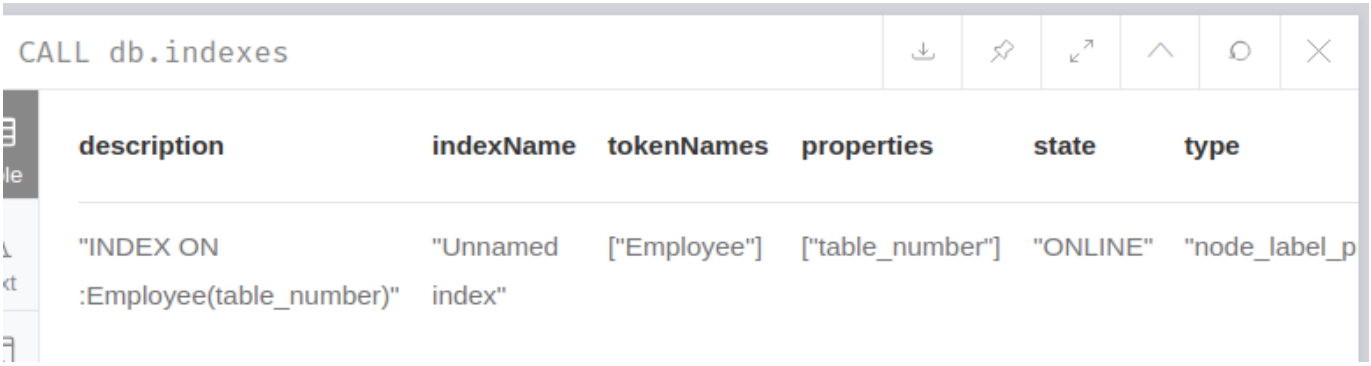
```
CREATE INDEX [index_name]

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)

CALL - Функция просмотра списка индексов

Создание индекса для нескольких свойств.

```
CALL db.indexes
```



DROP - Удаление индекса

```
DROP INDEX index_name
```

Удаление индекса одного свойства

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

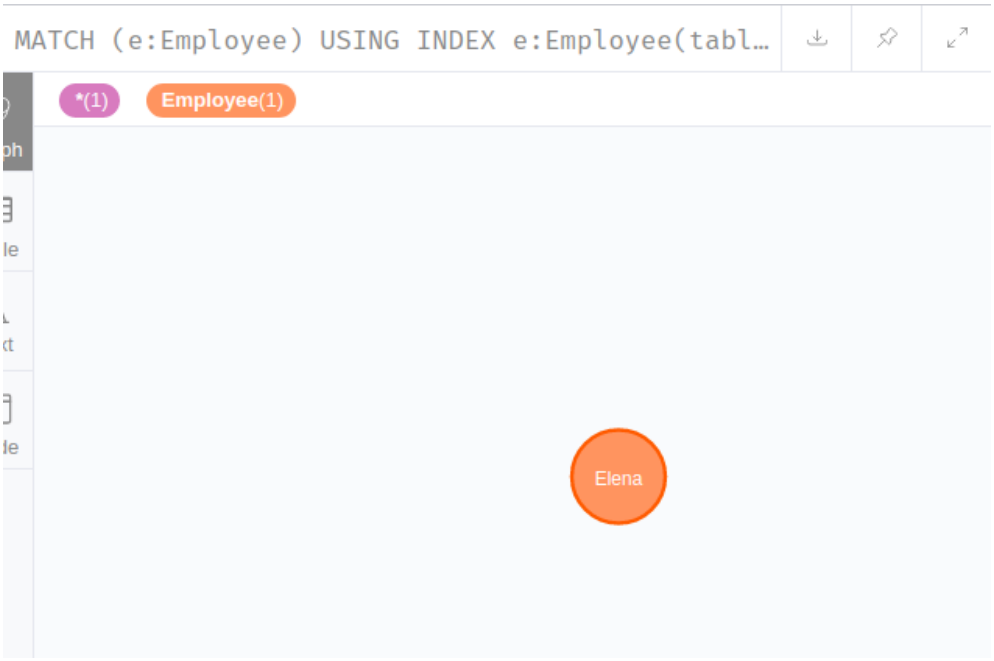
Удаление индекса нескольких свойств

```
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

Подробнее про индексы: *https://neo4j.com/docs/cypher-manual/current/administration/indexes-for-search-performance/*

Пример:
Запрос в котором на свойство Табельный номер узлов с меткой Employee накладывается индекс и ищется по нему сотрудник с табельным номером 4.
MATCH (e:Employee) USING INDEX e:Employee(table_number) WHERE e.table_number=4 RETURN e



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.

Синтаксис:
Ограничение уникальности свойства.

```
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(только для версии Enterprise Edition).

```
CREATE CONSTRAINT [constraint_name]

ON (node:Label)

ASSERT EXISTS (node.attribute)
```

Ограничение на существование свойства отношения(только для версии Enterprise Edition)..

```
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(только для версии Enterprise Edition)..

```
CREATE CONSTRAINT [constraint_name]

ON (node:Label)

ASSERT (node.attribute_1,

node.attribute_2,

…

node.attribute_n)

IS NODE KEY
```

Удаление ограничения.

```
DROP CONSTRAINT constraint_name
```

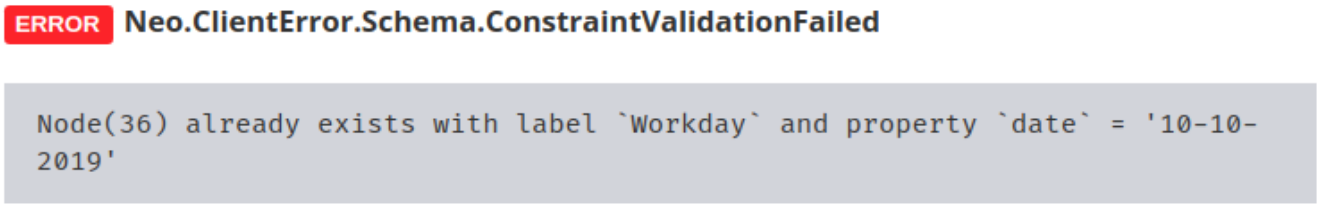
Вывод всех ограничений.

```
CALL db.constraints
```

Подробнее**:** https://neo4j.com/docs/cypher-manual/current/administration/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.
CREATE (w: Workday{ date: '10-10-2019', holiday_or_weekend: 'No'})

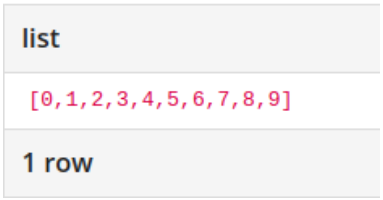


Коллекции.

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

Создание списка.

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



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 3 элемента списка [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

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

Для отсчета от конца коллекции можно также использовать отрицательные номера.

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

Чтобы вернуть диапазоны коллекции можно использовать диапазоны внутри скобок.

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

Можно использовать функцию size() для получения размера списка.

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

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

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.

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

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

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

Запрос, который создает список годов выпуска фильмов, в которых участвовал Киану Ривз.

```
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

Создание свойства — коллекции.

```
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.

```
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

Подробнее: https://neo4j.com/docs/cypher-manual/current/syntax/lists/

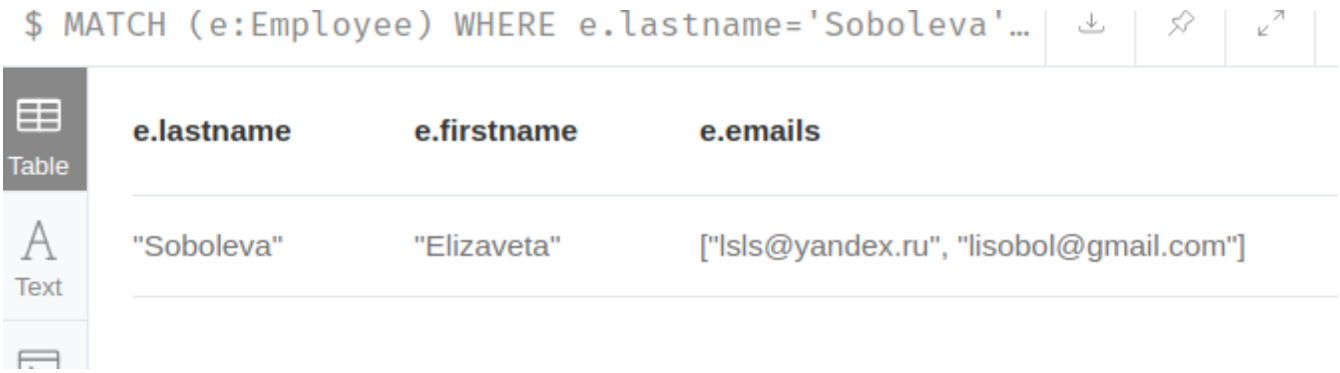
Пример:

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.

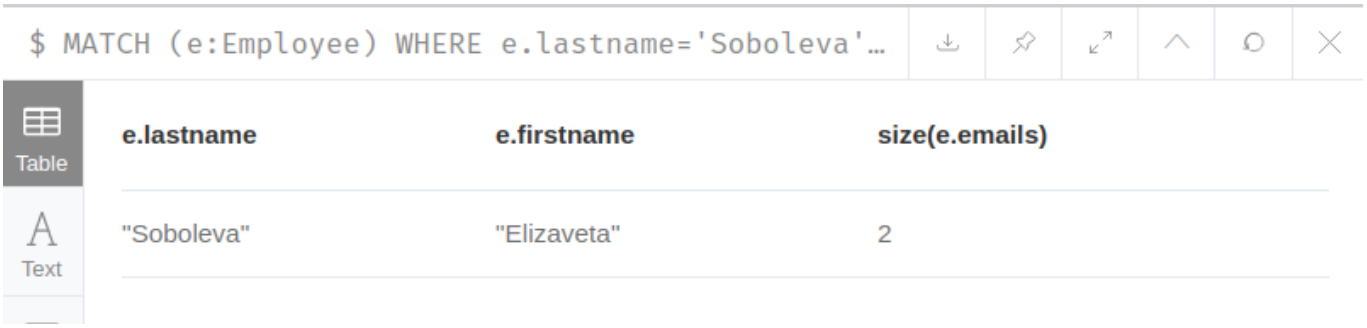
MATCH (e:Employee) WHERE e.lastname='Soboleva' RETURN e.lastname, e.firstname, e.emails



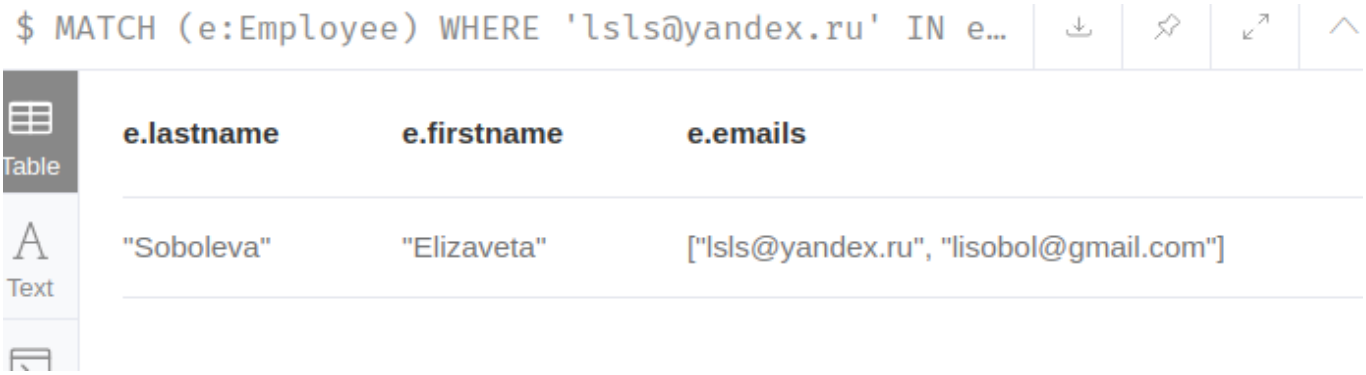
Выведем второй элемент списка электронных адресов. Нумерация элементов коллекции начинается с 0.

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

Выведем количество элементов коллекции с помощью функции size().
MATCH (e:Employee) WHERE e.lastname='Soboleva' RETURN e.lastname, e.firstname, size(e.emails)



Проверка наличия заданного адреса электронной почты в коллекции.
MATCH (e:Employee) WHERE 'lsls@yandex.ru' IN e.emails RETURN e.lastname, e.firstname, e.emails



Вопросы для самопроверки:

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?

Литература:

1. Дистрибутив - https://neo4j.com/ .
2. Документация по языку CQL - https://neo4j.com/docs/cypher-refcard/3.1/.
3. Руководство по установке и началу работы - https://ru.bmstu.wiki/Neo4j
4. Фаулер, Мартин, Садаладж, Прамодкумар Дж. NoSQL: новая методология разработки нереляционных баз данных. : Пер. с англ. - М.: ООО "И.Д. Вильямс", 2013г.
5. Neo4j – Краткое руководство - https://coderlessons.com/tutorials/bazy-dannykh/uznaite-neo4j/neo4j-kratkoe-rukovodstvo
6. Проверка времени исполнения сгенерированных запросов к графовой базе данных - https://cyberleninka.ru/article/n/proverka-vremeni-ispolneniya-sgenerirovannyh-zaprosov-k-grafovoy-baze-dannyh/viewer
7. Язык запросов Cypher (Neo4j) - http://art-in-stamps.ru/development/cypher.shtml