### Лабораторная работа № 2

Основы работы с Neo4j в браузере

Терминология Neo4j и графовых баз данных в целом.

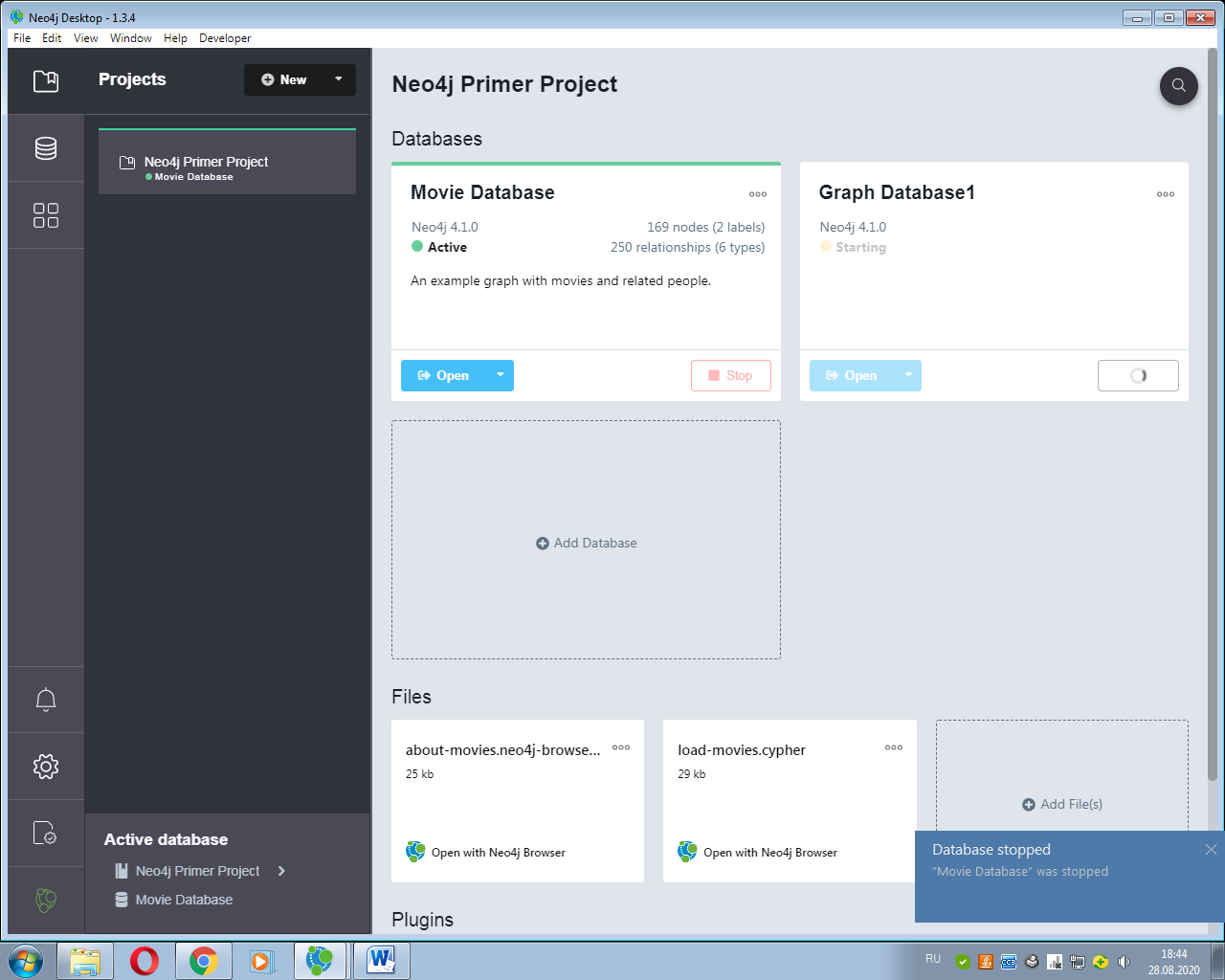
* **graph database**, **[графовая](http://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) база данных** — база данных построенная на графах — узлах и связях между ними
* [**Cypher**](http://www.neo4j.org/learn/cypher) — язык для написания запросов к базе данных Neo4j (примерно, как SQL в Oracle)
* **node**, **нода** — объект в базе данных, узел графа. Количество узлов ограниченно 2 в степени 35 ~ 34 биллиона
* **node label**, **метка ноды** — используется как условный «тип ноды». Например, ноды типа movie могут быть связанны с нодами типа actor. Метки нод **— регистрозависимые**, причем Cypher не выдает ошибок, если набрать не в том регистре название.
* **relation**, **связь** — связь между двумя нодами, ребро графа. Количество связей ограниченно 2 в степени 35 ~ 34 биллиона
* **relation identirfier**, **тип связи** — в Neo4j у связей. Максимальное количество типов связей 32767
* **properties**, **свойства ноды** — набор данных, которые можно назначить ноде. Например, если нода — это товар, то в свойствах ноды можно хранить id товара из базы Oracle
* **node ID**, **ID нода** — уникальный идентификатор ноды. По умолчанию, при просмотрах результата отображается именно этот ID. как его использовать в Cypher запросах
* Cypher - это декларативный язык запросов для neo4j

Основные принципы и возможности Cypher:

* Cypher сопоставляет шаблоны узлов и отношений на графике, чтобы извлечь информацию или изменить данные
* . Cypher имеет концепцию идентификаторов, которые обозначают именованные, связанные элементы и параметры.
* Cypher может создавать, обновлять и удалять узлы, отношения, метки и свойства.
* Cypher управляет индексами и ограничениями.

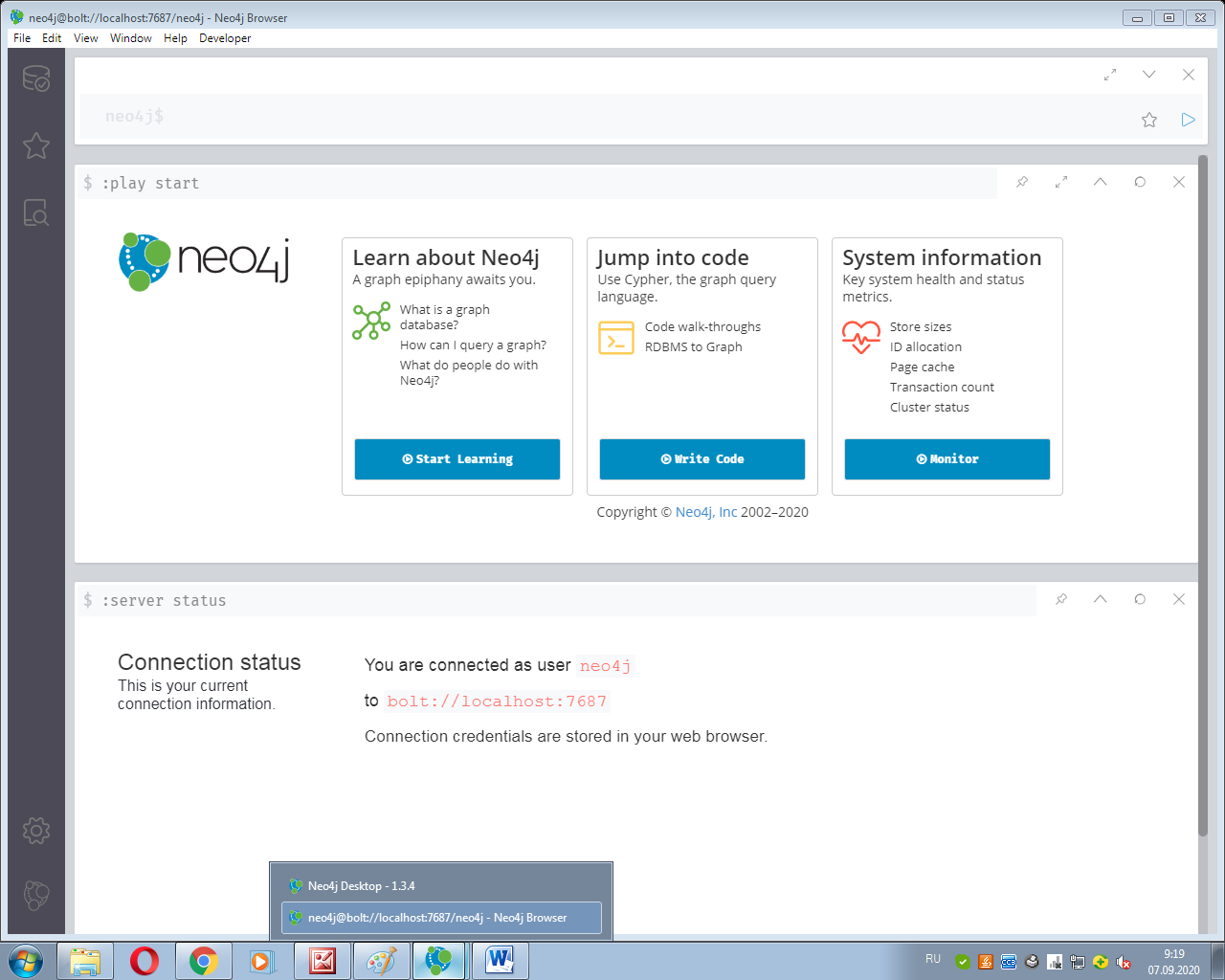
**Задание 1 – дайте команду «Start» БД «Movie Database» (шаблон - An example graph with movies and related people.)**

Познакомьтесь с меню Neo4j Destop



**Задание 2 – дайте команду «Open» БД «Movie Database»**

Познакомьтесь с меню Neo4j Browser



Вершины графа в Neo4j имеют свой тип, в БД **Movie Database** у нас 2 типа вершин:  
\* **Person** (name — имя актера, born — год рождения)  
**\* Movie** (title — наименование фильма, released — год выхода)

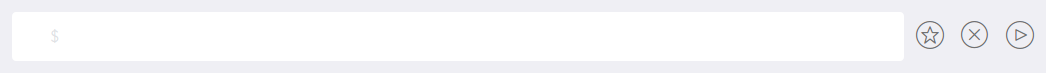
Если проводить аналогию с реляционными базами, то **Person**и **Movie —**это таблицы.

Для работы с БД используется язык запросов **Cypher.**

Cypher — декларативный язык запросов в виде графа, позволяющий получить выразительный и эффективный запрос данных. Язык представлен интуитивно понятным и простым в освоении.

**Редактор**

В верхней части окна Neo4j Browser располагается строка так называемого редактора:



Начиная набор команд с двоеточия, увидим список всех доступных команд с кратким описанием:

**Задание 3 В редакторе наберите команды**

Вызовем команду $ :help нажмите радиокнопку Run

Вызовем команду $ :sysinfo нажмите радиокнопку Run

Вызовем команду $ :play cypher нажмите радиокнопку Run

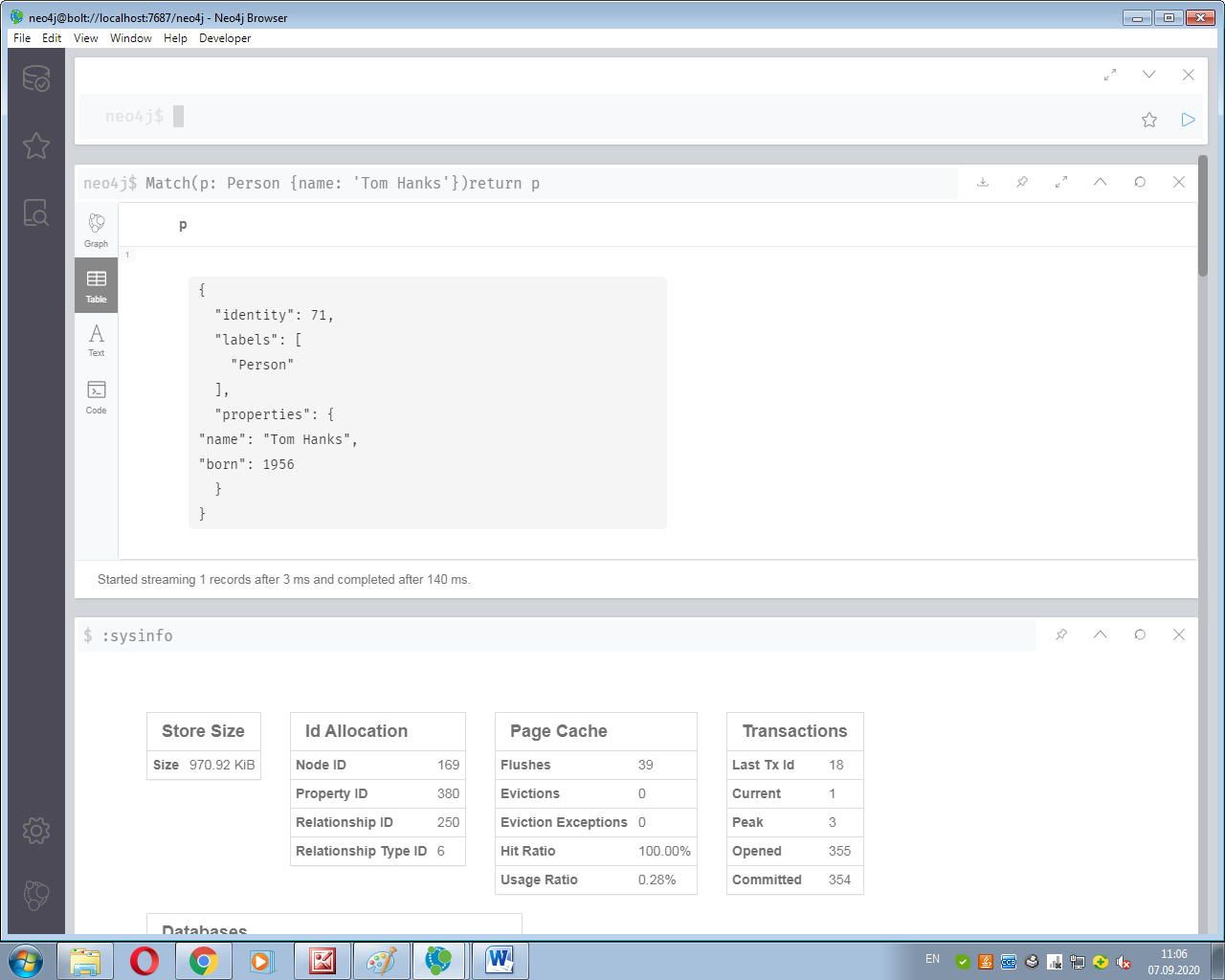
**Задание 4 В редакторе наберите команды (БД Movie Database)**

- Поиск: найдем актера с именем Том Хэнкс

Match(p: Person {name: 'Tom Hanks'})return p

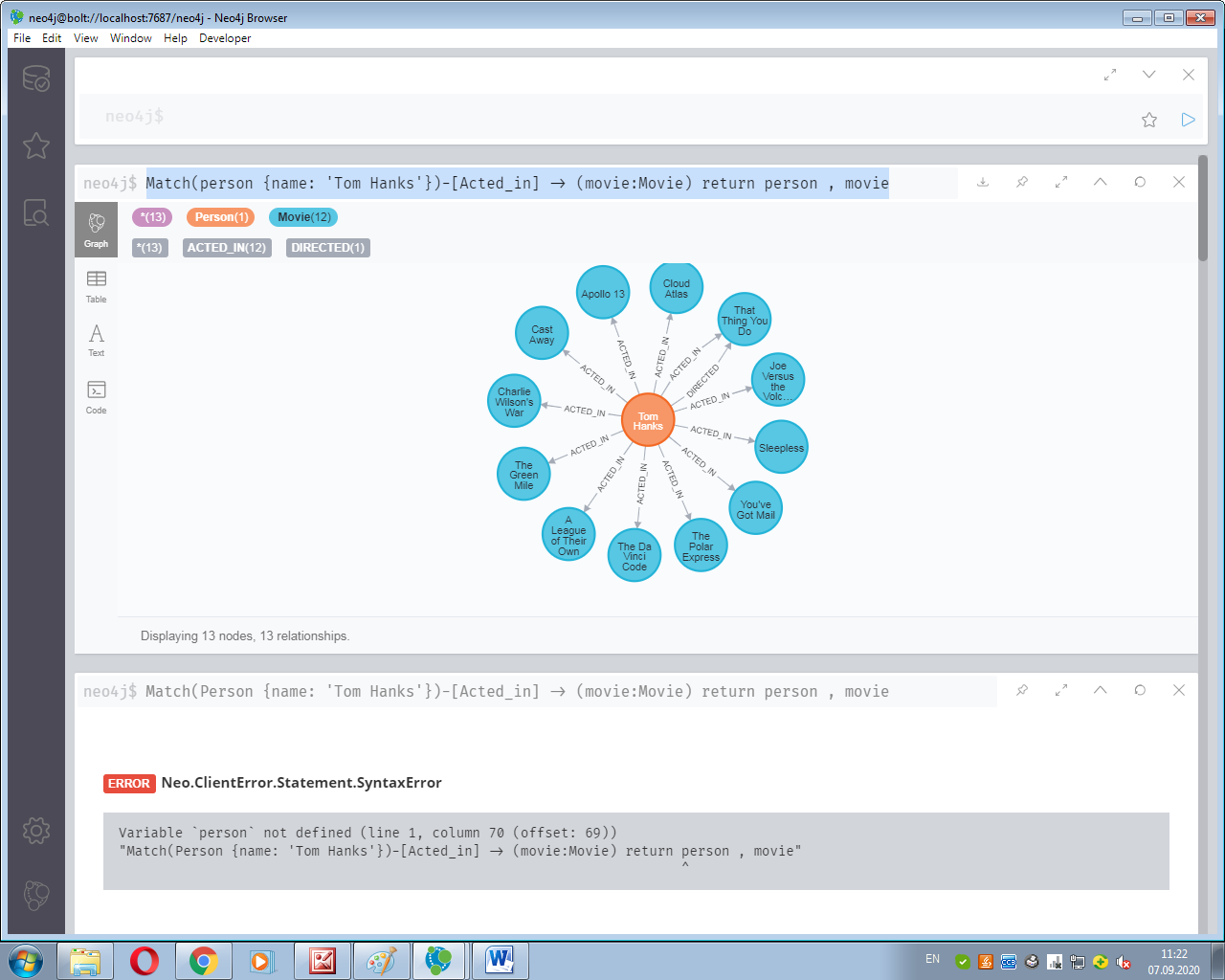
Осуществите навигацию по меню (просмотрите возможности)

* Чтобы указать узел, вы используете круглые скобки: ()
* Чтобы добавить свойства к узлу, используйте скобки: {} скобки



Найдем фильмы, в которых снимался Том Хэнкс:

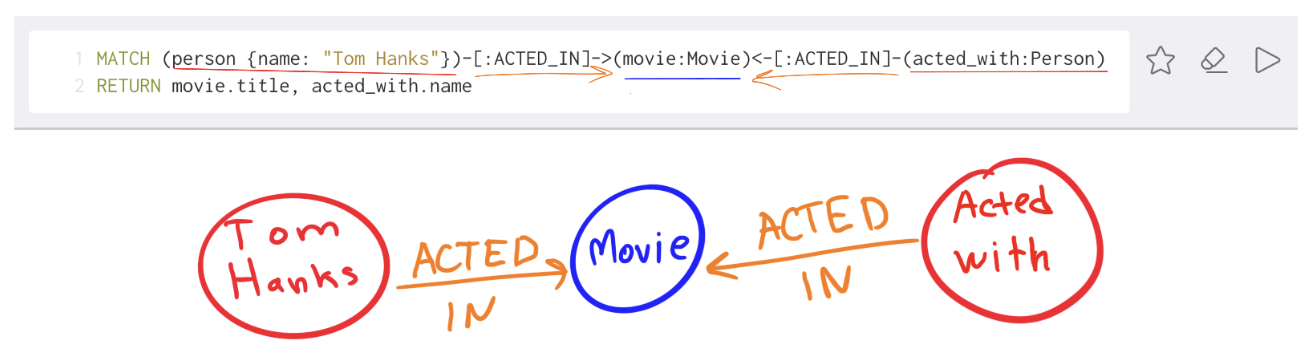
Match(person {name: 'Tom Hanks'})-[:ACTED\_IN] -> (movie:Movie) return person , movie



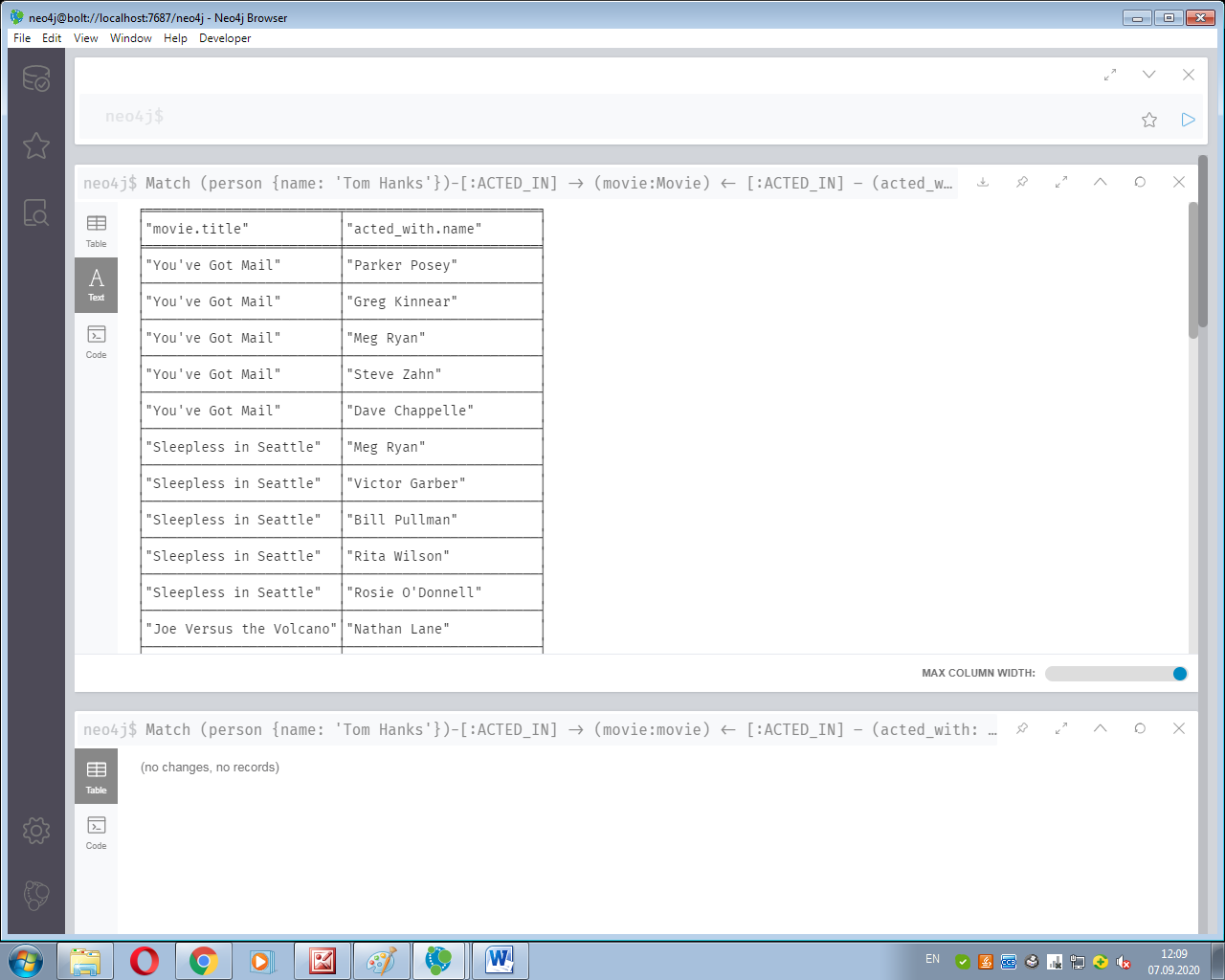
Теперь посмотрим, кто снимался с Томом на одной площадке, и в каком фильме:

Match (person {name: 'Tom Hanks'})-[:ACTED\_IN] -> (movie:Movie) <- [:ACTED\_IN] – (acted\_with: Person) return movie.title, acted\_with.name

Текст запроса приближен к текстовому отображению графа:



От вершины типа Person с именем Tom Hanks по ребру ACTED\_IN находятся все вершины Movie, в которых снимался актер, от вершин типа Movie по связи ACTED\_IN идет поиск всех актеров, снявшихся в каждом найденном фильме соответственно.



**Где и когда использовать графовые СУБД**

Графовые базы уже нашли применение в:

* Социальных сетях
* Системах рекомендаций (с этим товаром часто покупают…)
* Обработка пользовательских данных, корреляция данных из разных источников (информационный след в сети)

Если в вашем приложении планируется много сущностей и связей многие-ко-многим, то это один из признаков, что предпочтительней выбрать графовую СУБД *.* А если у вас уже есть приложение с множеством связей, и обход этих связей занимает много времени и ресурсов — стоит присмотреться в сторону графов, т.к. обход связей в них практически ничего не стоит.

**Выводы**

Графовые базы следует использовать для решения задач, которые решаются только графами.

Например, площадка **medium** использует Neo4j для хранения отношений между различными сущностями ([Стек, который позволил Medium обеспечить чтение на 2.6 тысячелетия](https://habr.com/ru/post/332860/)), а данные хранит в NoSQL БД **dynamoDB**.

Преимущества:

* Графы понятны на интуитивном уровне и являются частью школьной программы, поэтому их можно использовать при обсуждении задач с клиентом;
* Простой язык запросов и наглядный результат, поэтому может использоваться аналитиками;
* Гибкая структура данных, проще вносить изменения в случае изменения требований;
* Возможность создания приближенных к реальной жизни моделей, без низкоуровневых деталей.

Недостатки:

* БД занимает заметно больше места на диске, по отношению к реляционным СУБД;
* На простых запросах производительность ниже, чем в реляционных базах;
* Необходимо обучать разработчиков.

**Задание 5 Самостоятельно создайте дубликат БД** **Movie. Введите в нее свои данные. Выполните выше отработанные запросы.**

+ Add Database

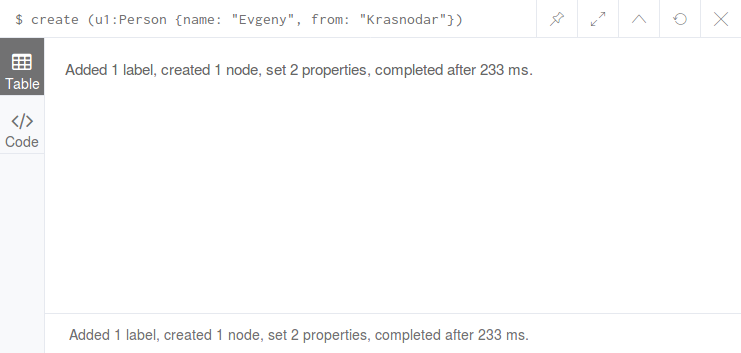
CREATE…

MATCH…

НАПРИМЕР: Создадим небольшой социальный граф. Перейдём в редактор и наберём первую команду на языке Cypher:

CREATE (u1:Person {name: "Evgeny", from: "Krasnodar"})

После выполнения команды Browser сообщит нам результат:

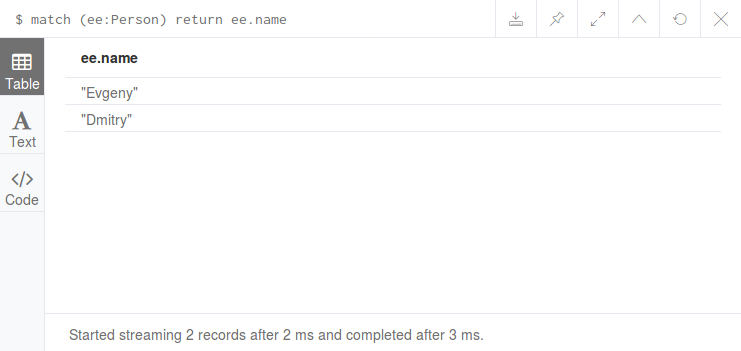


Добавим ещё один узел:

CREATE (u2:Person {name: "Dmitry", from: "Tula"})

Теперь запросим все узлы типа Person и извлечём значения свойства name:

MATCH (ee:Person) RETURN ee.name



*Примечание.* Как и в SQL есть возможность упорядочить извлекаемые данные по какому-либо полю:

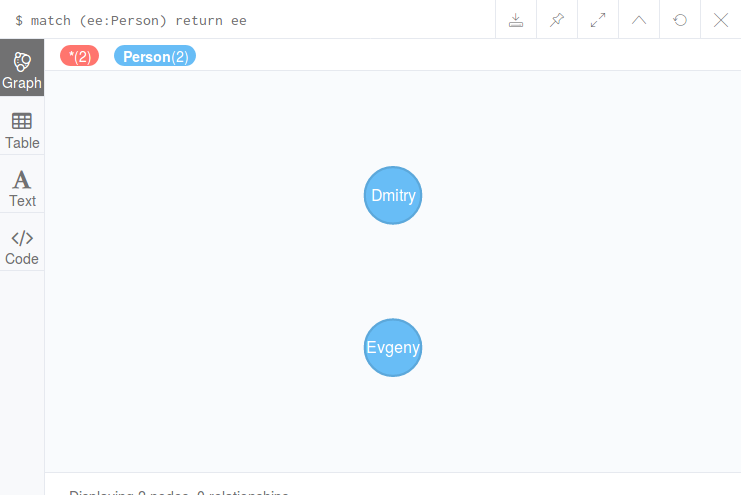
MATCH (ee:Person) RETURN ee.name ORDER BY ee.name

Далее можем запросить все узлы данного типа:

MATCH (ee:Person) RETURN ee



Обратите внимание на появившуюся кнопку **Graph**. Щёлкним на ней и увидим наши узлы в графическом виде:



*Примечание.* В версии 3.4 по-умолчанию как-раз открывается графическое представление. Для получения табличного представления нужно щёлкнуть на кнопку с надписью "Table". Хотя, бывает и наоборот.

Добавим связь между узлами:

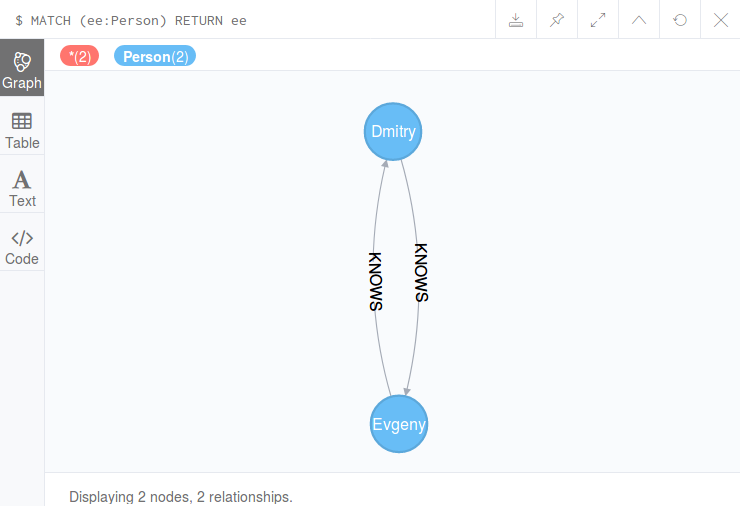
MATCH (e:Person) WHERE e.name = "Evgeny"

MATCH (d:Person) WHERE d.name = "Dmitry"

CREATE (e)-[:KNOWS]->(d),

(d)-[:KNOWS]->(e)

И вновь запросим наш граф:



С помощью Cypher можно также выполнять различные операции над графами, например, запрашивать смежные вершины, друзей друзей в социальном графе, удалять рёбра и вершины и многое другое.

Также можно настраивать Neo4j Browser на различный стиль отображения узлов и связей в зависимости от заданных им меток.