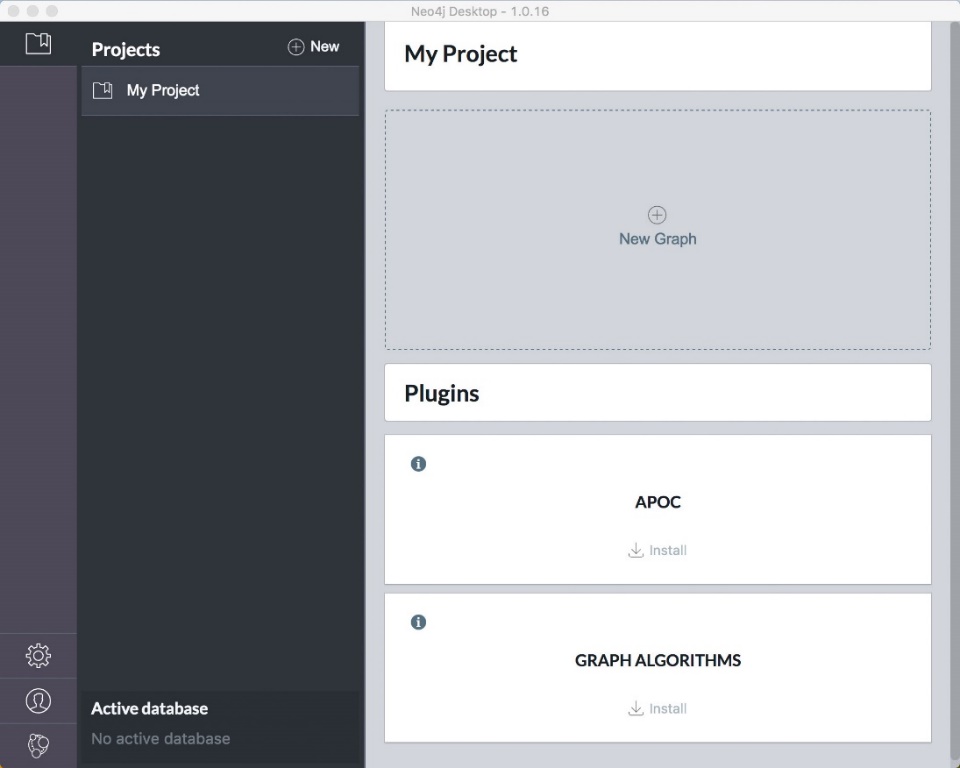
**Лабораторная работа №4**

**Разработка графовой модели в СУБД Neo4j и выполнение запросов на языке Cyther**

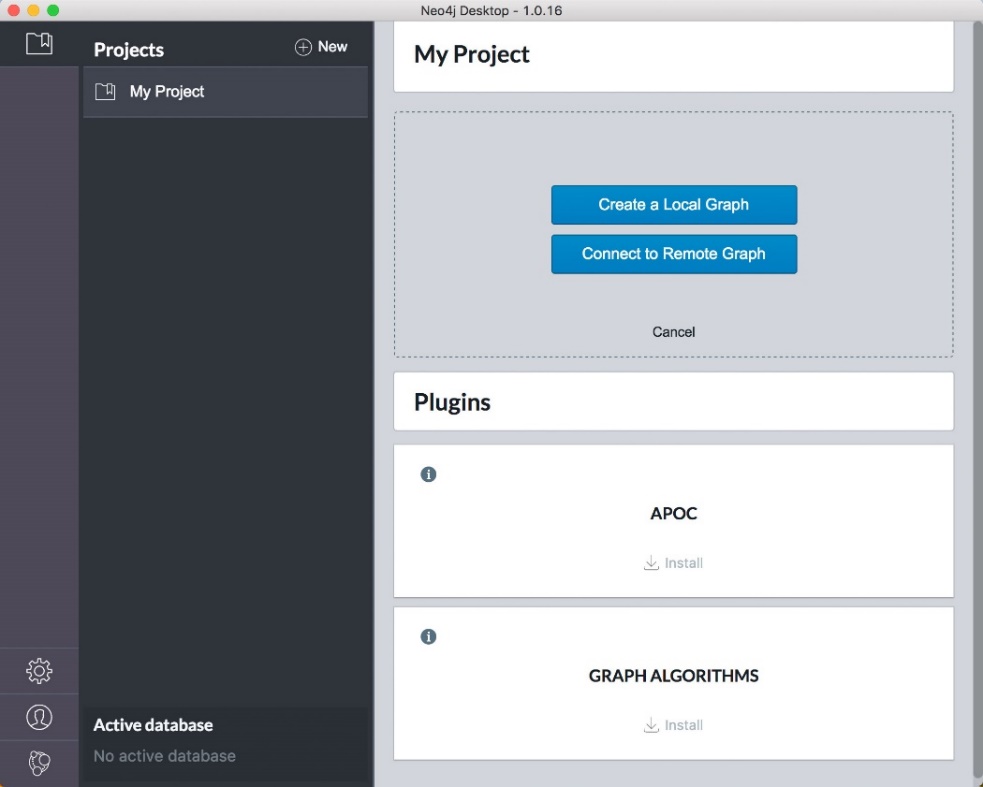
**Теоретическая часть**

Для установки СУБД Neo4j необходимо скачать установочный файл Neo4j с официального сайта [https://neo4j.com](https://neo4j.com/)

Для создания новой базы данных нажмите на кнопку «New Graph».



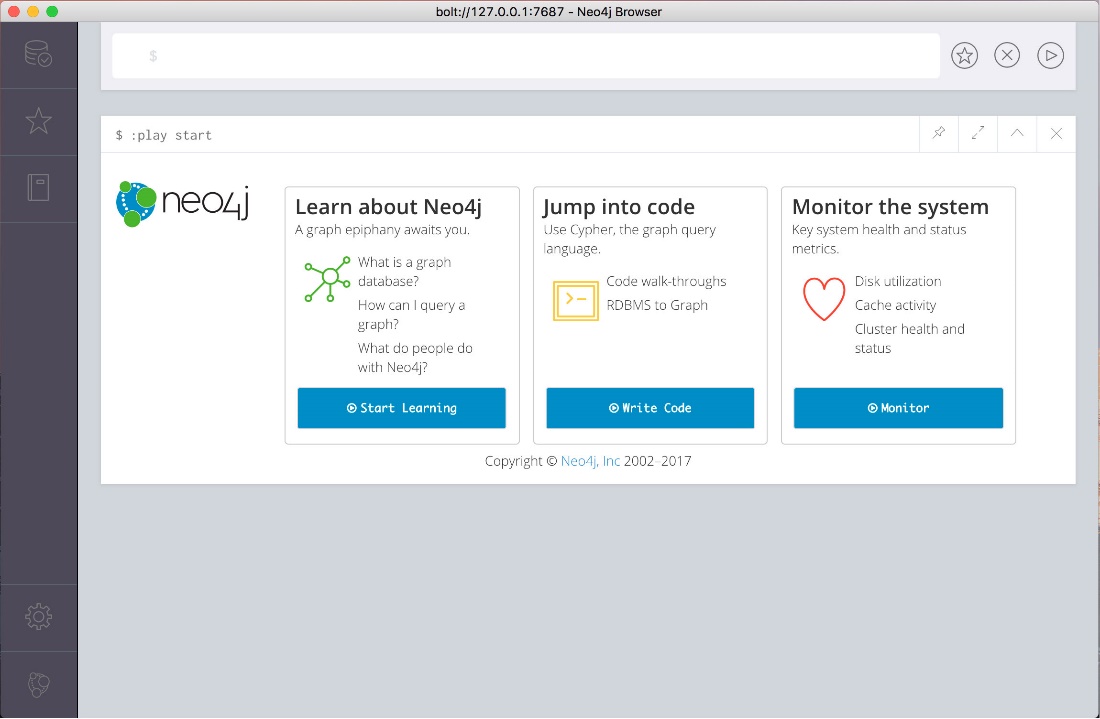
Затем выберите кнопку «Create a Local Graph».



Дайте имя базе данных и установите пароль. Затем нажмите на кнопку «Create»

Для начала работы нажмите кнопку «Start».

Для того, чтобы начать работать с базой данных в браузере нажмите кнопку «Manage», а затем - «Open Browser».



**Cypher** - это декларативный язык, основанный на SQL, для визуального описания графов, в которых используется синтаксис ASCII-Art.

Cypher - относительно простой, но все же очень мощный язык. Очень сложные запросы к базе данных можно легко выразить через Cypher. Разработчики попытались оптимизировать язык для чтения, а не для записи.

Будучи декларативным языком, Cypher фокусируется на ясности выражения того, что нужно извлечь из графа, а не о том, как его получить. Это противоречит императивным языкам (Java), языки сценариев (Gremlin), и привязки (JRuby Neo4j). Такой подход делает оптимизацию запросов вопросом реализации БД, а не пользователя, поэтому необходимо, чтобы происходило обновление всех обходов только потому, что изменилась структура физической базы (новые индексы и т. д.).

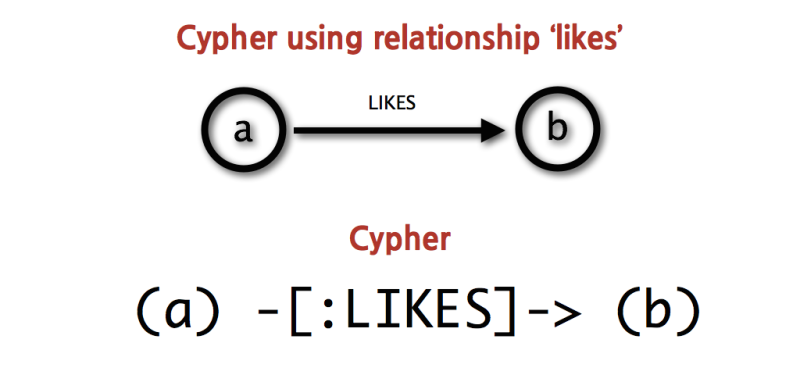
Cypher вдохновлен рядом различных подходов и опирается на установленные практики для выразительного запроса. Большинство ключевых слов, таких как WHERE и ORDER BY, основаны на SQL.

Синтаксис ASCII-Art 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 содержит следующие типы:

* узлы, отношения;
* пути;
* карты;
* списки;
* целые числа;
* числа с плавающей запятой;
* логические значения;
* строки.

Схема отношений между двумя узлами **a** и **b** с ребром-отношением **LIKES** показана ниже**:**



**Nodes (узлы)**

Cypher использует ASCII-Art для представления шаблонов. Мы окружаем узлы круглыми скобками, например: **(node)**. Если позже мы захотим обратиться к узлу, мы дадим ему переменную типа **(p)** для человека или **(t)** для вещи. В реальных запросах мы, вероятно, будем использовать более длинные, более выразительные имена переменных, например **(person)** или **(thing)**. Если узел не имеет отношения к вашему вопросу, вы также можете использовать пустые круглые скобки **()**. Обычно соответствующие метки узла предоставляются для различения сущностей и оптимизации выполнения, например **(p: Person)**.

Пример кода:

MATCH (node:Label) RETURN node.property

MATCH (node1:Label1)-->(node2:Label2)

WHERE node1.propertyA = {value}

RETURN node2.propertyA, node2.propertyB

**Relationships (связи)**

Чтобы полностью использовать возможности базы данных, основанной на графах, необходимо выразить более сложные шаблоны между узлами.

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: **()--()**

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]->**

Чтобы получить доступ к информации о взаимосвязи, мы можем назначить ей переменную для последующей ссылки. Он расположен перед двоеточием **- [rel: KNOWS] ->**или стоит один **- [rel] ->**.

Общий пример синтаксиса:

MATCH (n1:Label1)-[rel:TYPE]->(n2:Label2)

WHERE rel.property > {value}

RETURN rel.property, type(rel)

**Patterns (шаблоны)**

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.

* друг друга **(user)-[:KNOWS]-(friend)-[:KNOWS]-(foaf)**
* кратчайший путь: **path = shortestPath( (user)-[:KNOWS\*..5]-(other) )**
* совместная фильтрация **(user)-[:PURCHASED]->(product)<-[:PURCHASED]-()-[:PURCHASED]->(otherProduct)**
* навигация по дереву **(root)<-[:PARENT\*]-(leaf:Category)-[:ITEM]->(data:Product)**

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

В Cypher поддерживаются следующие форматы запросов:

|  |  |
| --- | --- |
| **Вид запроса** | **Синтаксис запроса** |
| Чтение данных | START [MATCH] [WHERE]  RETURN [ORDER BY] [SKIP] [LIMIT] |
| Запись данных | CREATE [UNIQUE]\* [SET|DELETE|FOREACH]\*  [RETURN [ORDER BY] [SKIP] [LIMIT]] |
| Чтение и запись данных | START [MATCH] [WHERE]  [CREATE [UNIQUE]]\* [SET|DELETE|FOREACH]\*  [RETURN [ORDER BY] [SKIP] [LIMIT]] |

Язык запросов Cypher состоит из следующих предложений:

**START:** начальные точки запроса в графе, полученные с помощью индексов или идентификаторов узлов (ID).

|  |  |
| --- | --- |
| **Пример** | **Семантика** |
| START n=node(\*) | Стартовая точка будет установлена на все узлы. |
| START n=node({ids}) | Стартовая точка будет установлена на один или  более узлов, указанных в списке идентификато- ров. |
| START n=node({id1}), m=node({id2}) | Несколько стартовых точек. |

**MATCH:** графовый паттерн, удовлетворяющий запросу, ограниченный начальными точками из оператора START. Любой паттерн может использоваться в предложении MATCH, за исключением паттернов, содержащих карты свойств.

Пример: MATCH (n)-->(m)

**WHERE:** критерий фильтрации.

Пример: WHERE n.property <> {value}

**RETURN**: возвращаемое значение.

|  |  |
| --- | --- |
| **Пример** | **Семантика** |
| RETURN \* | Вернуть все идентификаторы. |
| RETURN n AS columnName | Использовать псевдоним в качестве имени  колонки. |
| RETURN DISTINCT n | Вернуть уникальные строки. |
| ORDER BY n.property | Отсортировать по свойству. |

|  |  |
| --- | --- |
| **Пример** | **Семантика** |
| ORDER BY n.property DESC | Отсортировать по свойству в убывающем  порядке. |
| SKIP {skip\_number} | Пропустить заданное количество строк в вы-  даче. |
| LIMIT {limit\_number} | Ограничить выдачу заданным количеством  строк. |
| SKIP {skip\_number} LIMIT  {limit\_number} | Пропустить сколько-то строк, а потом огра-  ничить выдачу. |

**CREATE:** создание узлов и связей.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Пример** | | | | **Семантика** |
| CREATE (n {name: {value}}) | | | | Создать узел с заданными свойства-  ми. |
| CREATE | n | = | {map} | Создать узел с заданными свойства-  ми. |
| CREATE n = {collectionOfMaps} | | | | Создать узел с заданными свойства-  ми. |
| CREATE (n)-[r:KNOWS]->(m) | | | | Создать связь заданного типа и на-  правления; присвоить ему идентифи- катор. |
| CREATE (n)-[:LOVES {since: {val- ue}}]->(m) | | | | Создать отношение из заданного ти-  па, направления и набора свойств. |

**DELETE:** удаление узлов, связей и свойств.

Примеры: DELETE n, r - удалить узел или отношение. DELETE n.property - удалить свойство.

**SET:** установка значений свойств.

Примеры: SET n.property={value} - обновить или создать свойство. SET n={map} - задать все свойства (удалит все существующие свойства).

**FOREACH:** выполнение операций обновления для каждого элемента из списка.

**WITH:** разбивает запрос на несколько независимых частей.

При выполнении запросов на языке Cypher необходимо задавать искомые паттерны отношений между узлами. Паттерны составляются по следующим правилам:

|  |  |
| --- | --- |
| **Правило** | **Семантика** |
| (n)-->(m) | Проверка на существование отношения  от n к m. |
| (n)--(m) | Проверка на существование отношения  между n и m (направление игнорируется). |
| (m)<-[:KNOWS]-(n) | Проверка на существование отношения  от n к m типа KNOWS. |
| (n)-[:KNOWS|LOVES]->(m) | Проверка на существование отношения  от n к m типа KNOWS или LOVES. |
| (n)-[r]->(m) | Присвоить идентификатор отношению. |
| (n)-[r?]->(m) | Опциональное отношение. |
| (n)-[\*1..5]->(m) | Переменная для пути отношений между  n и m. |
| (n)-[\*]->(m) | Нет ограничений на глубину отношений. |
| (n)-[:KNOWS]->(m {property:  {value}}) | Сопоставить набор свойств в предложениях CREATE или CREATE UNIQUE. |

В языке Cypher предусмотрены функции для работы с коллекциями и массивами:

|  |  |
| --- | --- |
| **Функция** | **Семантика** |
| NODES(path) | Список узлов в указанном пути. |
| RELATIONSHIPS(path) | Отношения в указанном пути. |
| EXTRACT(x IN collection: x.prop) | Коллекция значений выражения  для каждого элемента коллекции. |
| FILTER(x IN coll: x.prop <> {value}) | Возвращает коллекцию элемен- тов, для которых предикат исти-  нен. |
| TAIL(collection) | Все элементы коллекции, кроме  первого. |
| RANGE({begin}, {end}, {step}) REDUCE(str = "", n IN coll : str + n.prop) | Создать набор чисел с указанным шагом. Вычислить выражение для каждого элемента коллекции и ак-  кумулировать результат. |
| FOREACH (n IN coll : SET n.marked = true) | Выполнить операцию обновления  для каждого элемента в коллек- ции. |

**Вспомогательные функции:**

|  |  |
| --- | --- |
| **Функция** | **Семантика** |
| LENGTH(collection) | Возвращает длину коллекции. |
| TYPE(a\_relationship) | Строковое представление типа отношения. |
| COALESCE(n.property?,  {defaultValue}) | Первое ненулевое выражение. |
| HEAD(collection) | Первый элемент коллекции. |
| LAST(collection) | Последний элемент коллекции. |
| TIMESTAMP() | Количество прошедших миллисекунд от  полуночи 1 января 1970 года по … . |
| ID(node\_or\_relationship) | Внутренний идентификатор отношения или  узла. |

При работе с примитивами языка Cypher используется развитая система

предикатов:

|  |  |
| --- | --- |
| **Предикат** | **Семантика** |
| n.property <> {value} | Использование операторов сравнения. |
| HAS(n.property) AND n.property = {value} | Использование булевых операторов  для комбинирования предикатов. |
| HAS(n.property) | Использование функций. |
| identifier IS NULL | Проверка на null. |
| n.property? = {value} | Возвращает истину, если свойство не  существует. |
| n.property! = {value} | Возвращает ложь, если свойство не  существует. |
| n.property =~ {regex} | Регулярные выражения. |
| (n)-[:KNOWS]->(m) | Убедиться, что паттерн имеет хотя  бы одно совпадение. |
| n.property IN [{val1}, {val2}] | Проверка, что элемент существует в  коллекции. |

В языке Cypher также имеются функции для вычисления предикатов над коллекциями:

|  |  |
| --- | --- |
| **Предикат** | **Семантика** |
| ALL(x IN collection WHERE HAS(x.property)) | Возвращает истину, если предикат выполня-  ется для всех элементов коллекции. |
| ANY(x IN collection WHERE HAS(x.property)) | Возвращает истину, если предикат выполня-  ется хотя бы для одного элемента коллекции. |
| NONE(x IN collection WHERE HAS(x.property)) | Возвращает истину, если предикат не вы-  полняется для всех элементов коллекции. |
| SINGLE(x IN collection WHERE HAS(x.property)) | Возвращает истину, если предикат выполня-  ется строго для одного элемента коллекции. |

**Операторы**:

* математические +, -, \*, /, %;
* сравнения =, <>, <, >, <=, >=,
* булевы AND, OR, NOT;
* строковые +;
* операторы коллекций: +, IN;
* регулярные выражения: =~;
* операторы свойств ?, !.

**Агрегирующие функции:** COUNT, SUM, AVG, MAX, MIN, COLLECT, PERCENTILE\_DISC, PERCENTILE\_CONT.

**Математические функции:** ABS, ROUND, SQRT, SIGN.

**Строковые функции:** STR, REPLACE, SUBSTRING, LEFT, RIGHT, LTRIM, RTRIM, TRIM, LOWER, UPPER.

**Пример**

**1. Создание записи о себе**

CREATE (you:Person {name:"You"})

RETURN you

**CREATE** создает узлы с метками и свойствами.

**2. Добавление базы данных и отношения с ней**

MATCH (you:Person {name:"You"})

CREATE (you)-[like:LIKE]->(neo:Database {name:"Neo4j" })

RETURN you,like,neo

**3. Создание других пользователей**

**CREATE** может создавать отдельные узлы или более сложные структуры.

MATCH (you:Person {name:"You"})

FOREACH (name in ["Johan","Rajesh","Anna","Julia","Andrew"] |

CREATE (you)-[:FRIEND]->(:Person {name:name}))

**4. Поиск других пользователей**

**FOREACH** позволяет выполнять операции обновления для каждого элемента списка.

MATCH (you {name:"You"})-[:FRIEND]->(yourFriends)

RETURN you, yourFriends

**5. Создание взаимоотношений второго уровня**

MATCH (neo:Database {name:"Neo4j"})

MATCH (anna:Person {name:"Anna"})

CREATE (anna)-[:FRIEND]->(:Person:Expert {name:"Amanda"})-[:WORKED\_WITH]->(neo)

**6. Поиск пользователей по различным свойствам**

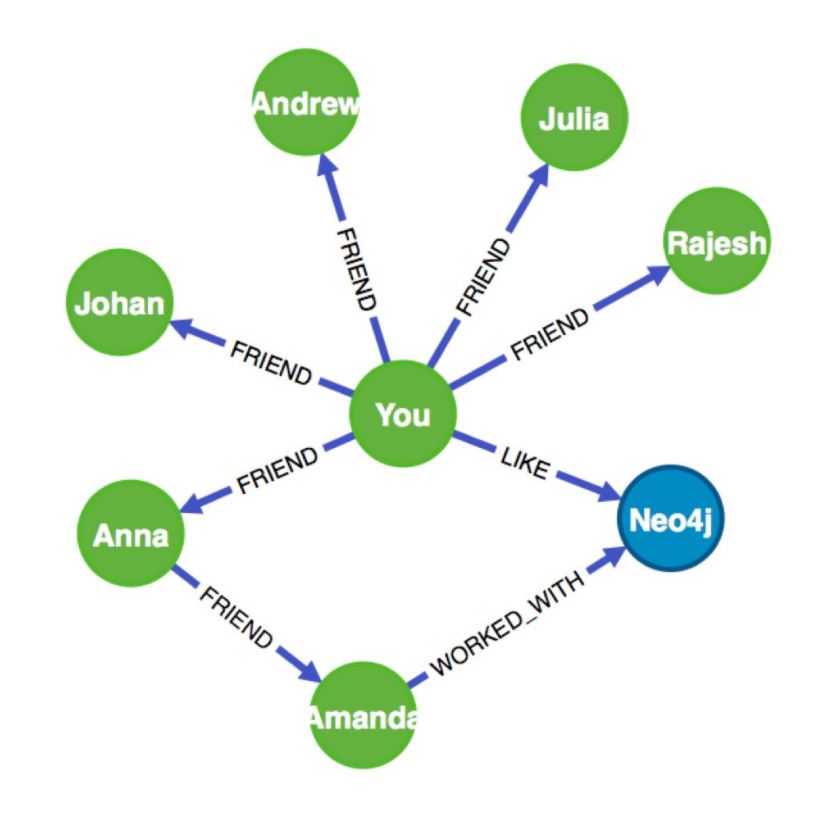
MATCH (you {name:"You"})

MATCH (expert)-[:WORKED\_WITH]->(db:Database {name:"Neo4j"})

MATCH path = shortestPath( (you)-[:FRIEND\*..5]-(expert) )

RETURN db,expert,path

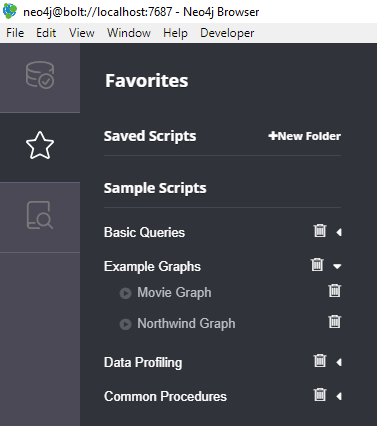
**Конечный результат выполнения всех команд:**

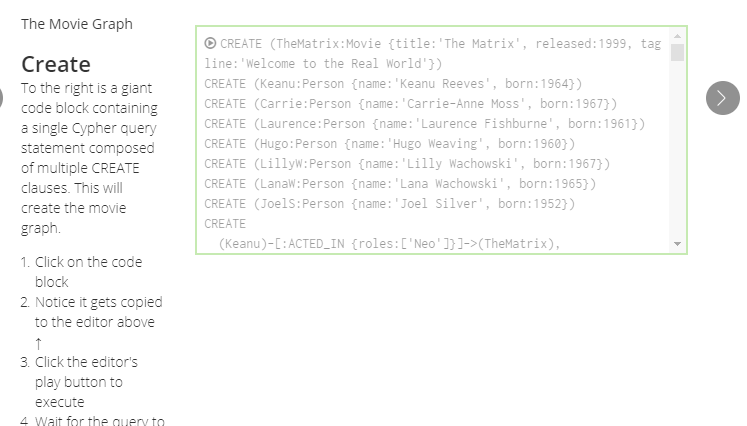


**Задания к лабораторной работе**

**Задание 1**

Загрузите Neo4j и откройте примера графа Movie. Выполните пошаговое руководство, изучите работу графа и особенности основных команд Cyther.





**Задание 2**

Используя графовую СУБД Neo4j, разработайте базу данных модельной социальной сети. Социальная сеть имеет два типа узлов: человек и группа. Узел человек имеет следующие атрибуты: ФИО, пол, возраст, город. Узел группа имеет следующие атрибуты: название (например: спорт, игры, компьютеры). Кроме того узлы имеют атрибут "тип". Семантика связи между узлами – дружеские отношения соответствующих персон («А является другом Б»), участие в группе («А является участником группы Б»). База должна содержать не менее 10 узлов и не менее 15 связей между ними.

Разработайте следующие запросы:

* 1. Выдать упорядоченный список ФИО персон.
  2. Выдать список ФИО мужчин с указанием возраста, упорядоченный по убыванию возраста.
  3. Выдать упорядоченный список ФИО друзей персоны заданными ФИО.
  4. Выдать упорядоченный по алфавиту список ФИО персон, в котором для каждой персоны указано количество друзей.
  5. Выдать упорядоченный список групп социальной сети.
  6. Выдать упорядоченный список групп персоны с заданными ФИО.
  7. Выдать список групп социальной сети с указанием количества членов каждой группы, упорядоченный по убыванию количества членов группы.
  8. Выдать список ФИО персон, в котором для каждой персоны указано количество групп, в которые она входит, упорядоченный по убыванию количества групп.