### Лабораторная работа № 3

# NEO4J язык запросов Cypher (CQL) <https://coderlessons.com/tutorials/bazy-dannykh/uznaite-neo4j/neo4j-cql-vvedenie>

|  |  |  |
| --- | --- | --- |
| **Sr.No** | **RDBMS** | **База данных (графы)** |
| 1 | таблицы | диаграммы |
| 2 | Ряды | Вершины |
| 3 | Столбцы и данные | Свойства и его значения |
| 4 | Ограничения | Отношения |
| 5 | присоединяется | пересечение |

Сравнения (реляционные БД и Graph).

Neo4j Graph Database имеет следующие строительные блоки —

* Вершины
* свойства
* Отношения
* Этикетки
* Браузер данных

**Создание одного узла**

CREATE (sample)

просмотр

MATCH (n) RETURN n

**Создание нескольких узлов**

CREATE (sample1),(sample2)

**Создание узла с меткой (ми)**

CREATE (Dhawan:player)

CREATE (Dhawan:person:player)

**Создать узел со свойствами**

CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"})

RETURN с CREATE можно использовать для просмотра вновь созданного узла.

CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"}) RETURN Dhawan

просмотр всех узлов

MATCH (n) RETURN n

**Создание отношений (синтаксис)**

CREATE (node1)-[:RelationshipType]->(node2)

CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"})

CREATE (Ind:Country {name: "India"})

создайте связь с именем **BATSMAN\_OF** между этими двумя узлами

CREATE (Dhawan)-[r:BATSMAN\_OF]->(Ind)

RETURN Dhawan,Ind

просмотрите все узлы и удалите их

MATCH (n) RETURN n

MATCH (n) DETACH DELETE n

MATCH (n) RETURN n

**Создание связи между существующими узлами**

CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"}) return Dhawan

CREATE (Ind:Country {name: "India"}) return Ind

MATCH (a:player), (b:Country) WHERE a.name = "Shikar Dhawan" AND b.name = "India"

CREATE (a)-[r: BATSMAN\_OF]->(b)

RETURN a,b

**Создание отношений с меткой и свойствами**

MATCH (a:player), (b:Country) WHERE a.name = "Shikar Dhawan" AND b.name = "India"

CREATE (a)-[r:BATSMAN\_OF {Matches:5, Avg:90.75}]->(b)

RETURN a,b

**Создание полного пути (синтаксис)**

CREATE p = (Node1 {properties})-[:Relationship\_Type]->

(Node2 {properties})[:Relationship\_Type]->(Node3 {properties})

RETURN p

CREATE p = (Dhawan {name:"Shikar Dhawan"})-[:TOPSCORRER\_OF]->(Ind{name:

"India"})-[:WINNER\_OF]->(CT2013{name: "Champions Trophy 2013"})

RETURN p

***Команда MERGE является комбинацией команды CREATE и команды MATCH.***

CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"})

CREATE (Ind:Country {name: "India"})

CREATE (Dhawan)-[r:BATSMAN\_OF]->(Ind)

Если существует какой-либо узел с данной меткой, Neo4j возвращает их все.

MERGE (Jadeja:player) RETURN Jadeja

MATCH (n) RETURN n

**Слияние узла со свойствами**

Этот запрос пытается объединить узел с именем «jadeja», используя свойства и метку. Поскольку нет такого узла с точной меткой и свойствами, Neo4j создает его.

MERGE (Jadeja:player {name: "Ravindra Jadeja", YOB: 1988, POB: "NavagamGhed"})

RETURN Jadeja

**OnCreate и OnMatch**

Всякий раз, когда мы выполняем запрос на слияние, узел либо совпадает, либо создается. Используя on create и on match, вы можете установить свойства для указания того, создан ли узел или соответствует.

MERGE (Jadeja:player {name: "Ravindra Jadeja", YOB: 1988, POB: "NavagamGhed"})

ON CREATE SET Jadeja.isCreated = "true"

ON MATCH SET Jadeja.isFound = "true"

RETURN Jadeja

**Объединить отношения**

Как и узлы, вы также можете объединить отношения, используя предложение MERGE.

Поскольку такого отношения не существует, Neo4j создает его.

Create (b:Tournament {name: "ICC Champions Trophy 2013"})

MATCH (a:Country), (b:Tournament)

WHERE a.name = "India" AND b.name = "ICC Champions Trophy 2013"

MERGE (a)-[r:WINNERS\_OF]->(b)

RETURN a, b

**Установка свойства**

Используя предложение SET, вы можете создать новое свойство в узле.

MATCH (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"})

SET Dhawan.highestscore = 187

RETURN Dhawan

**Удаление свойства**

Вы можете удалить существующее свойство, передав ему **значение NULL** .

Create (Jadeja:player {name: "Ravindra Jadeja", YOB: 1988, POB: "NavagamGhed"})

MATCH (Jadeja:player {name: "Ravindra Jadeja", YOB: 1988, POB: "NavagamGhed"})

SET Jadeja.POB = NULL

RETURN Jadeja

**Установка нескольких свойств**

MATCH (Jadeja:player {name:"Ravindra Jadeja",YOB: 1988})

SET Jadeja.POB="NavagamGhed", Jadeja.HS="90"

RETURN Jadeja

**Установка метки на узле**

CREATE (Anderson {name: "James Anderson", YOB: 1982, POB: "Burnely"})

Этот запрос добавляет метку «игрок» к узлу Андерсон и возвращает его.

MATCH (Anderson {name: "James Anderson", YOB: 1982, POB: "Burnely"})

SET Anderson: player

RETURN Anderson

**Установка нескольких меток на узле**

CREATE (Ishant {name: "Ishant Sharma", YOB: 1988, POB: "Delhi"})

MATCH (Ishant {name: "Ishant Sharma", YOB: 1988, POB: "Delhi"})

SET Ishant: player:person

RETURN Ishant

**Удаление всех узлов и отношений**

MATCH (n) DETACH DELETE n

**Удаление определенного узла**

CREATE (Ishant:player {name: "Ishant Sharma", YOB: 1988, POB: "Delhi"})

MATCH (Ishant:player {name: "Ishant Sharma", YOB: 1988, POB: "Delhi"})

DETACH DELETE Ishant

**Удаление свойства**

CREATE (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB: "Ranchi"})

MATCH (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB: "Ranchi"})

REMOVE Dhoni.POB

RETURN Dhoni

**Удаление метки с узла**

MATCH (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB: "Ranchi"})

REMOVE Dhoni:player

RETURN Dhoni

Предложение **FOREACH** используется для обновления данных в списке, будь то компоненты пути или результат агрегации.

 Пример добавляет свойство ко всем узлам на пути, используя предложение FOREACH.

CREATE p = (Dhawan {name:"Shikar Dhawan"})-[:TOPSCORRER\_OF]->(Ind{name:

"India"})-[:WINNER\_OF]->(CT2013{name: "Champions Trophy 2013"})

RETURN p

MATCH p = (Dhawan)-[\*]->(CT2013)

WHERE Dhawan.name = "Shikar Dhawan" AND CT2013.name = "Champions Trophy 2013"

FOREACH (n IN nodes(p)| SET n.marked = TRUE)

MATCH (n) RETURN n

Этот запрос возвращает все узлы в базе данных (мы подробно обсудим этот запрос в следующих главах).

Ниже дан код генерации таблиц EMP в Run SQL Command line

**ЗАДАЧА СОЗДАТЬ АНАЛОГ (реляционной БД) ДАННОЙ СХЕМЫ в виде графа Neo4j**

--

-- Copyright (c) Oracle Corporation 1988, 2000. All Rights Reserved.

--

-- NAME

-- demobld.sql

--

-- DESCRIPTION

-- This script creates the SQL\*Plus demonstration tables in the

-- current schema. It should be STARTed by each user wishing to

-- access the tables. To remove the tables use the demodrop.sql

-- script.

--

-- USAGE

-- From within SQL\*Plus, enter:

-- START demobld.sql

SET TERMOUT ON

PROMPT Building demonstration TABLES. Please wait.

SET TERMOUT OFF

DROP TABLE EMP;

DROP TABLE DEPT;

DROP TABLE DEPT1;

DROP TABLE BONUS;

DROP TABLE SALGRADE;

DROP TABLE DUMMY;

CREATE TABLE EMP

(EMPNO NUMBER(4) NOT NULL,

ENAME VARCHAR2(10),

JOB VARCHAR2(9),

MGR NUMBER(4),

HIREDATE DATE,

SAL NUMBER(7, 2),

COMM NUMBER(7, 2),

DEPTNO NUMBER(2));

INSERT INTO EMP VALUES

(7369, 'SMITH', 'CLERK', 7902,

TO\_DATE('17-12-1980', 'DD-MM-YYYY'), 800, NULL, 20);

INSERT INTO EMP VALUES

(7369, 'SMITH', 'CLERK', 7902,

TO\_DATE('17-12-1980', 'DD-MM-YYYY'), 800, NULL, 20);

INSERT INTO EMP VALUES

(7499, 'ALLEN', 'SALESMAN', 7698,

TO\_DATE('20-02-1981', 'DD-MM-YYYY'), 1600, 300, 30);

INSERT INTO EMP VALUES

(7521, 'WARD', 'SALESMAN', 7698,

TO\_DATE('22-02-1981', 'DD-MM-YYYY'), 1250, 500, 30);

INSERT INTO EMP VALUES

(7566, 'JONES', 'MANAGER', 7839,

TO\_DATE('2-4-1981', 'DD-MM-YYYY'), 2975, NULL, 20);

INSERT INTO EMP VALUES

(7654, 'MARTIN', 'SALESMAN', 7698,

TO\_DATE('28-9-1981', 'DD-MM-YYYY'), 1250, 1400, 30);

INSERT INTO EMP VALUES

(7698, 'BLAKE', 'MANAGER', 7839,

TO\_DATE('1-5-1981', 'DD-MM-YYYY'), 2850, NULL, 30);

INSERT INTO EMP VALUES

(7782, 'CLARK', 'MANAGER', 7839,

TO\_DATE('9-6-1981', 'DD-MM-YYYY'), 2450, NULL, 10);

INSERT INTO EMP VALUES

(7788, 'SCOTT', 'ANALYST', 7566,

TO\_DATE('09-12-1982', 'DD-MM-YYYY'), 3000, NULL, 20);

INSERT INTO EMP VALUES

(7839, 'KING', 'PRESIDENT', NULL,

TO\_DATE('17-11-1981', 'DD-MM-YYYY'), 5000, NULL, 10);

INSERT INTO EMP VALUES

(7844, 'TURNER', 'SALESMAN', 7698,

TO\_DATE('8-9-1981', 'DD-MM-YYYY'), 1500, 0, 30);

INSERT INTO EMP VALUES

(7876, 'ADAMS', 'CLERK', 7788,

TO\_DATE('12-1-1983', 'DD-MM-YYYY'), 1100, NULL, 20);

INSERT INTO EMP VALUES

(7900, 'JAMES', 'CLERK', 7698,

TO\_DATE('3-12-1981', 'DD-MM-YYYY'), 950, NULL, 30);

INSERT INTO EMP VALUES

(7902, 'FORD', 'ANALYST', 7566,

TO\_DATE('3-12-1981', 'DD-MM-YYYY'), 3000, NULL, 20);

INSERT INTO EMP VALUES

(7934, 'MILLER', 'CLERK', 7782,

TO\_DATE('23-1-1982', 'DD-MM-YYYY'), 1300, NULL, 10);

INSERT INTO EMP VALUES

(7934, 'MILLER', 'CLERK', 7782,

TO\_DATE('23-1-1982', 'DD-MM-YYYY'), 1300, NULL, 10);

CREATE TABLE DEPT

(DEPTNO NUMBER(2),

DNAME VARCHAR2(14),

LOC VARCHAR2(13) );

INSERT INTO DEPT VALUES (10, 'ACCOUNTING', 'NEW YORK');

INSERT INTO DEPT VALUES (20, 'RESEARCH', 'DALLAS');

INSERT INTO DEPT VALUES (30, 'SALES', 'CHICAGO');

INSERT INTO DEPT VALUES (40, 'OPERATIONS', 'BOSTON');

CREATE TABLE BONUS

(ENAME VARCHAR2(10),

JOB VARCHAR2(9),

SAL NUMBER,

COMM NUMBER);

CREATE TABLE SALGRADE

(GRADE NUMBER,

LOSAL NUMBER,

HISAL NUMBER);

INSERT INTO SALGRADE VALUES (1, 700, 1200);

INSERT INTO SALGRADE VALUES (2, 1201, 1400);

INSERT INTO SALGRADE VALUES (3, 1401, 2000);

INSERT INTO SALGRADE VALUES (4, 2001, 3000);

INSERT INTO SALGRADE VALUES (5, 3001, 9999);

CREATE TABLE DUMMY

(DUMMY NUMBER);

INSERT INTO DUMMY VALUES (0);

COMMIT;

SET TERMOUT ON

PROMPT DEMONSRTRATION TABLE BUILD IS COMPLETE

Задание . Выполните следующие манипуляции в Neo4j в браузере

**УЗЛЫ**

- создание одного узла CREATE (EMP)

- верификация MATCH (n) RETURN n

- удаление узлов вообще не имеющих свойств: MATCH (n) WHERE keys(n) = [] DETACH DELETE n

- удаление DROP CONSTRAINT ON (node:EMP) ASSERT n.id is UNIQUE

- верификация MATCH (n) RETURN n

- создание двух (нескольких) узлов CREATE (EMP), (DEPT)

- удаление узлов вообще не имеющих свойств: MATCH (n) WHERE keys(n) = [] DETACH DELETE n

- верификация MATCH (n) RETURN n

- cоздание узла EMP с меткой employees CREATE (EMP: employees)

- верификация MATCH (n) RETURN n

- создание второго узла с меткой CREATE (DEPT: departments)

- удаление узлов вообще не имеющих свойств: MATCH (n) WHERE keys(n) = [] DETACH DELETE n

- удалить не связанные ноды MATCH (n) DELETE n

- cоздание узла EMP с несколькими метками e1,e2,… CREATE (EMP: e1:e2)

- cоздание узла со свойствами СИНТАКСИС

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

* CREATE (EMP: employees {EMPNO: 7369, ENAME: ‘Smith’, SAL: 800.1})

INSERT INTO EMP VALUES

(7369, 'SMITH', 'CLERK', 7902,

TO\_DATE('17-12-1980', 'DD-MM-YYYY'), 800, NULL, 20);

- Возврат созданного узла СИНТАКСИС

CREATE (Node:Label{properties. . . . }) RETURN Node

мы использовали запрос MATCH (n) RETURN n для просмотра созданных узлов. Этот запрос возвращает все существующие узлы в базе данных. Вместо этого мы можем использовать предложение RETURN с CREATE для просмотра вновь созданного узла.

- CREATE (EMP: employees {EMPNO: 7369, ENAME: 'Smith', SAL: 800}) return EMP

INSERT INTO EMP VALUES

(7499, 'ALLEN', 'SALESMAN', 7698,

TO\_DATE('20-02-1981', 'DD-MM-YYYY'), 1600, 300, 30);

- CREATE (EMP: employees {EMPNO: 7499, ENAME: ‘Allen’, SAL: 1600}) return EMP

INSERT INTO DEPT VALUES (20, 'RESEARCH', 'DALLAS');

- CREATE (DEPT: departments {DEPNO: 20, DNAME: ‘Research’}) return DEPT

INSERT INTO DEPT VALUES (30, 'SALES', 'CHICAGO');

- CREATE (DEPT: departments {DEPNO: 30, DNAME: “Sales”}) return DEPT

**- Введите еще несколько сотрудников и подразделений !!! По ходу работы сохраняйте/экспортируйте граф в формат .csv !!! ДАННЫЙ ФАЙЛ ЯВЛЯЕТСЯ ОТЧЕТОМ ПО ЛАБОРАТОРНОЙ РАБОТЕ !!!**

|  |  |
| --- | --- |
| **Neo4j**  **n**  **{}**  **{}**  **{"ENAME":Smith,"EMPNO":7369,"SAL":800}**  **{"ENAME":‘Allen,"EMPNO":7499,"SAL":1600}** | **SQL> spool d:/export\_run\_sql.csv**  **SQL> select EMPNO,ENAME,SAL from EMP where EMPNO in (7369,7499)**  **7369 SMITH 800**  **7499 ALLEN 1600**  **SQL> spool off** |

**ОТНОШЕНИЯ**

Ниже приведен синтаксис для создания отношений с использованием предложения CREATE СИНТАКСИС CREATE (node1)-[:RelationshipType]->(node2)

- CREATE (EMP)-[r:WORKS\_FOR]-> (DEPT)

Создание связи между существующими узлами c использованием MATCH

* MATCH (a:employees), (b:departments)

WHERE a.EMPNO = 7369 AND b.DEPNO = 20

CREATE (a)-[: Relation]->(b)

RETURN a,b

* MATCH (a:employees), (b:departments)

WHERE a.EMPNO = 7369 AND b.DEPNO = 20

CREATE (a)-[: works\_for]->(b)

RETURN a,b

Вы можете создать связь с меткой и свойствами, используя предложение CREATE.

* MATCH (a:employees), (b:departments)

WHERE a.EMPNO = 7499 AND b.DEPNO = 30

CREATE (a)-[: works\_for {loc: ‘Chicago’}]->(b)

RETURN a,b

**ОБЪЕДИНЕНИЕ КОМАНД** Команда MERGE является комбинацией команды CREATE и команды MATCH

Слияние узла с меткой

* **MERGE (EMP: employees) RETURN EMP**

**OnCreate и OnMatch**

Всякий раз, когда мы выполняем запрос на слияние, узел либо совпадает, либо создается. Используя on create и on match, вы можете установить свойства для указания того, создан ли узел или соответствует.

* MERGE (EMP: employees {EMPNO: 7369, ENAME: ‘Smith’, SAL: 800.1})

ON CREATE SET EMP.isCreated = "true"

ON MATCH SET EMP.isFound = "true"

RETURN EMP

**Объединить отношения**

- MATCH (a:employees), (b:departments)

WHERE a.EMPNO = 7369 AND b.DEPNO = 20

CREATE (a)-[: look\_for]->(b)

RETURN a,b

Поскольку такого отношения не существует, Neo4j создает его.

**!!! ЗАДАНИЕ (ПРОРАБОТАТЬ СЛЕДУЮЩИЙ МАТЕРИАЛ)**

[**https://coderlessons.com/tutorials/bazy-dannykh/uznaite-neo4j/neo4j-kratkoe-rukovodstvo**](https://coderlessons.com/tutorials/bazy-dannykh/uznaite-neo4j/neo4j-kratkoe-rukovodstvo)

Написание запросов

Разберем простейший синтаксис написания запросов к Neo4j по CRUD.

Чтение данных

Для начала, надо вспомнить что в структуре данных у нас есть:

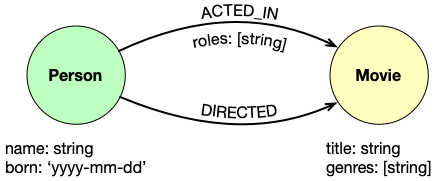
* Nodes — сущности;
* Labels — метки к сущностям;
* Relationships — однонаправленные отношения между сущностями;
* Properties — свойства сущностей или отношений.

Очень упрощенно, основной шаблон запроса для чтения записей можно описать следующим образом:

(<var>:<entity> [{<property>: <value>[, ...]}])<direction>[:<relationship>]<direction>(<var>:<entity>[{<property>: <value>[, ...]}])

Согласен, с примером будет понятнее :)

Вспомним нашу базу данных с фильмами и актерами, и попробуем сделать запрос к ней



// Найти все фильмы

MATCH (m:Movie)

RETURN m

Для получения какой-то выборки используется ключевое слово MATCH (точная копия SELECT из RDBMS).

m:Movie — тут мы присваиваем переменной m все найденные Movies по заданным условиям.

RETURN p — явно указываем *что* должен вернуть наш запрос.

Больше примеров:

// Найти всех актеров с именем Tom Hanks:

MATCH (p:Person {name: 'Tom Hanks'})

RETURN p

{name: 'Tom Hanks'} — условие фильтрации по сущности Person

Но, графовая БД не была бы таковой, если бы в ней не было главной особенности: связей между сущностями.

Давайте найдем все фильмы, в которых играл Tom Hanks:

MATCH (p:Person {name: 'Tom Hanks'})-->(m)

RETURN m.title

m — тут мы присваиваем переменной m все найденные зависимые сущности по заданным условиям.

Отлично! Но неправильно :) Потому что это вернет нам все его связи со всеми зависимыми сущностями по всей базе данных, если таковые будут. Нам это не нужно, мы хотим получить фильмы, в которых он играл. Добавим тип связи ACTED\_IN:

MATCH (p:Person {name: 'Tom Hanks'})-[:ACTED\_IN]->(m)

RETURN m.title

Уже лучше, но тоже неточно. Этот запрос вернет *все* сущности, в которых он играл, не только фильмы. Он же мог играть и в театральных постановках, правильно?

Эти примеры выше работают благодаря так называемомой способности *traverse*, если говорить в терминах Neo4j.

Нода, хоть как-то связанная с другой нодой (даже через другие N нод), соответственно, — *traversable*-нода.

Конкретно в нашей схеме данных других сущностей нет, поэтому получим только Movies. Но лучше не копить техдолг, а явно указать что мы ожидаем только фильмы:

MATCH (p:Person {name: 'Tom Hanks'})-[:ACTED\_IN]->(m:Movie)

RETURN m.title

Теперь отлично!

Больше примеров:

// Найти список всех актеров из "Матрицы"

MATCH (m:Movie {title: 'The Matrix'})<-[:ACTED\_IN]-(p:Person)

RETURN p

// Найти всех режиссеров, родившихся до 1970 года

// Обратите внимание: нам не важно ЧТО именно они режиссировали,

// поэтому тип ноды не указываем:

MATCH (p:Person)-[:DIRECTED]->()

WHERE p.born < 1970

RETURN p

Можно комбинировать связи:

// Найти все фильмы, в которых Tom Hanks

// был одновременно актером ИЛИ режиссеором:

MATCH (m:Movie)<-[:ACTED\_IN|:DIRECTED]-(p:Person {name: 'Tom Hanks'})

RETURN m.title, p.name

В общем случае, ключевое слово WHERE работает почти так-же, как и в RDBMS, поэтому на нем подробно останавливаться не буду. Однако есть интересные применения в контексте проверки связей:

// Найти всех актеров, которые играли в фильмах,

// но не являлись при этом режиссером фильма:

MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WHERE NOT exists( (p)-[:DIRECTED]->(m) )

RETURN p.name, m.title

// Обратиться к атрибуту связи:

// найти имя актера, который играл Нео в Матрице

MATCH (p:Person)-[r:ACTED\_IN]->(m:Movie)

WHERE 'Neo' IN r.roles AND m.title='The Matrix'

RETURN p.name, r.roles

Запись данных

Для добавления данных в Neo4j используется слово MERGE, однако будет большой ошибкой проводить полную аналогию с ключевыми словами CREATE или INSERT из RDBMS.

Дело в том, что MERGE создает *паттерн* в базе данных.

Проще сразу с примерами:

// Создать ноду Person с именем Michael Cain

MERGE (p:Person {name: 'Michael Cain'})

// Создать две ноды (да, это один запрос на три строчки)

MERGE (p:Person {name: 'Katie Holmes'})

MERGE (m:Movie {title: 'The Dark Knight'})

RETURN p, m

Небольшое отступление: можно также использовать ключевое слово CREATE с той лишь разницей, что CREATE не будет проверять по первичному ключу существует ли уже такая нода в БД — это дает ему огромное преимущество в скорости. CREATE чаще всего используют для дампов, чтобы быстро можно было их развернуть на чистой базе. MERGE занимается вставкой данных более аккуратно — с проверкой всех условий, но и более медленно.

Создадим простой паттерн через MERGE:

MATCH (p:Person {name: 'Michael Cain'})

MATCH (m:Movie {title: 'The Dark Knight'})

MERGE (p)-[:ACTED\_IN]->(m)



Ничего не изменилось.

Дело в том, что в этом запросе MATCH говорит нам о том, что мы должны найти записи, а MERGE о том, что мы должны создать связь между ними. Однако, если записи не найдены, то связь и не создастся. Для гарантированного создания сущности или связи можно использовать MERGE для всех трех частей запроса:

MERGE (p:Person {name: 'Michael Cain'})

MERGE (m:Movie {title: 'The Dark Knight'})

MERGE (p)-[:ACTED\_IN]->(m)

// или в одну строку:

MERGE (p:Person {name: 'Michael Cain'})-[:ACTED\_IN]->(m:Movie {title: 'The Dark Knight'})

RETURN p, m



Что тут происходит по логике обработчика запроса:

1. Neo4j пытается найти сущность Person с именем Michael Cain;
2. Если сущность найдена, то Neo4j смотрит все ее связи ACTED\_IN;
3. Затем среди этих связей Neo4j ищет Movie с названием The Dark Knight.

Если хотя-бы на одном из этих этапов не было найдено объектов, то будет создан **весь паттерн**. То есть, если даже фильм The Dark Knight существует, а актер Michael Cain нет, то будет создан новый фильм с таким названием. Это важнейшая особенность, которую стоит учитывать, чтобы избегать дублирования данных.

Попробуем воспроизвести это:

// Выполним трижды запрос на добавление актеров в один и тот-же фильм

MERGE (p:Person {name: 'Yuri Nikulin'})-[:ACTED\_IN]->(m:Movie {title: 'The Diamond Arm'})

RETURN p, m

MERGE (p:Person {name: 'Nina Grebeshkova'})-[:ACTED\_IN]->(m:Movie {title: 'The Diamond Arm'})

RETURN p, m

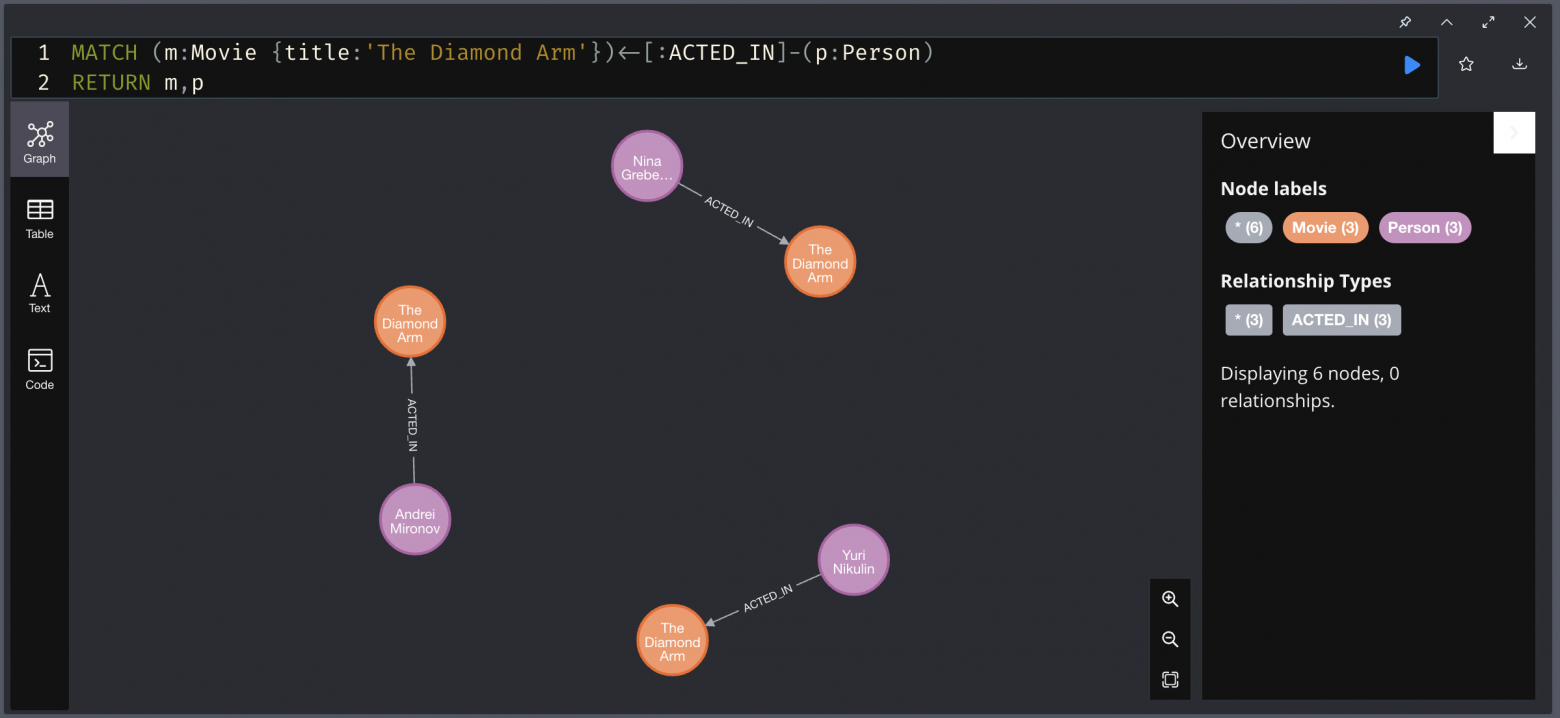
MERGE (p:Person {name: 'Andrei Mironov'})-[:ACTED\_IN]->(m:Movie {title: 'The Diamond Arm'})

RETURN p, m

// Найдем все записи с этим названием фильма

MATCH (m:Movie {title:'The Diamond Arm'})<-[:ACTED\_IN]-(p:Person)

RETURN m,p

Получилось явно не то, что мы хотели

Решением такого кейса может быть создание всех сущностей отдельно и затем отдельно связей между ними. Всё отдельными запросами.

Более сложный пример для создания паттернов:

// В нашей базе Oliver Stone и Rob Reiner никогда не работали вместе,

// то есть у них нет общего фильма.

//

// В этом запросе мы ищем обоих

// и создаем новый отличный безымянный фильм для них :)

MATCH

(oliver:Person {name: 'Oliver Stone'}),

(reiner:Person {name: 'Rob Reiner'})

MERGE (oliver)-[:DIRECTED]->(movie:Movie)<-[:ACTED\_IN]-(reiner)

RETURN movie

Также полезно знать, что для добавления и обновления атрибутов используется ключевое слово SET:

MATCH (p:Person)-[r:ACTED\_IN]->(m:Movie)

WHERE p.name = 'Michael Cain' AND m.title = 'The Dark Knight'

SET r.roles = ['Alfred Penny'], r.year = 2008

RETURN p, r, m

Удаление данных

Это, пожалуй, самый простой раздел, потому что ломать — не строить :)

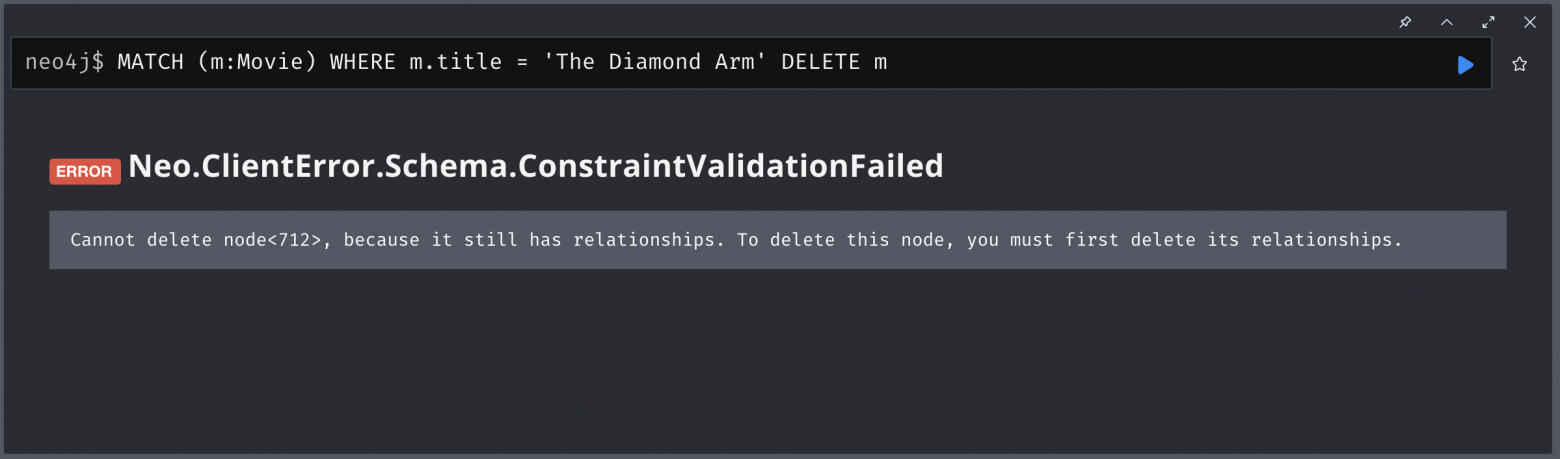
Самое важное знание тут — это то, что нельзя удалить ноду, пока у нее есть хоть одна связь.

Удалим все фильмы с названием The Diamond Arm, которые мы создали ранее:

MATCH (p:Person)

WHERE p.name = 'Jane Doe'

DELETE p

Oops!

Окей, нельзя удалять пока есть связи. Для быстрой починки этого у нас есть ключевая пара слов DETACH DELETE. Но если мы используем такое для удаления фильмов, то никогда потом не найдем актеров из них, которых тоже хотелось бы удалить.

Тогда удалим сначала актеров, потом сами дубликаты фильмов:

// Отцепим (DETACH) и удалим актеров фильмов с названием The Diamond Arm

MATCH (m:Movie {title:'The Diamond Arm'})<-[:ACTED\_IN]-(p:Person)

DETACH DELETE p

// Удалим сами дубликаты фильмов с названием The Diamond Arm.

// Отцеплять (DETACH) уже не нужно, т.к. актеров вместе со связями

// мы удалили только что

MATCH (m:Movie {title:'The Diamond Arm'})

DELETE m

Для удаления атрибутов можно просто присвоить им значение null или явно вызвать ключевое слово REMOVE:

// Вот так

MATCH (p:Person)

WHERE p.name = 'Gene Hackman'

SET p.born = null

RETURN p

// Или так

MATCH (p:Person)-[r:ACTED\_IN]->(m:Movie)

WHERE p.name = 'Michael Cain' AND m.title = 'The Dark Knight'

REMOVE r.roles

RETURN p, r, m

Заключение

Как видно из примеров выше, работать с графовыми БД довольно просто, но в то же время необычно для тех, кто много времени проводит с SQL, нужно немного перестроить мышление.