Bazy danych 2 – Neo4j

Opracował: Mateusz Woś

Source code: https://github.com/mwoss/neo4j_lab_task

1. **(1pkt) Należy wymyślić, lub znaleźć prosty graf, który ma przynajmniej 2 węzły i 3 różne krawędzie (+ kilka atrybutów)**.

Graf stworzony przeze mnie składa się z 3 rodzajów wierzchołków:

- Pokemonów
- Regionów
- Trenerów

Rodzaje krawędzi:

- KNOWS
- HAVE
- COMES
- LIVES

Węzły posiadają następujące atrybuty:

- Pokemony: name, type, weight
- Trenerzy: name, last name, age, gender
- Regiony: name, professor, region villain

Do wypisania informacji o moim grafie posłużyłem się następującą funkcją:

```
public void databaseStatistics() {
    System.out.printf(graphDatabase.runCypher("CALL db.schema()"));
    System.out.println(graphDatabase.runCypher("CALL db.labels()"));
    System.out.println(graphDatabase.runCypher("CALL db.relationshipTypes()"));
    System.out.println(graphDatabase.runCypher("MATCH (n)\n" +
            "OPTIONAL MATCH (n)-[r]->(x)\n" +
            "WITH DISTINCT {node1: labels(n), r: type(r), node2: labels(x)}\n" +
            "AS `Relations_between_nodes`\n" +
            "RETURN Relations_between_nodes"));
}
```

Output:

```
+-----------------------------------------------------------------------------------------------------+
| nodes                                | relationships                                                |
+-----------------------------------------------------------------------------------------------------+
| [Node[-1]{},Node[-2]{},Node[-3]{}] | [:LIVES[-1]{},:HAVE[-3]{},:COMES[-2]{},:KNOWS[-4]{}] |
+-----------------------------------------------------------------------------------------------------+
1 row
+-----------+       +------------------+
| label     |       | relationshipType |
+-----------+       +------------------+
| "Region"  |       | "LIVES"          |
| "Trainer" |       | "COMES"          |
| "Pokemon" |       | "HAVE"           |
+-----------+       | "KNOWS"          |
3 rows                +------------------+
                      4 rows
+-------------------------------------------------------------+
| Relations_between_nodes                                     |
+-------------------------------------------------------------+
| {node1 -> ["Region"], r -> <null>, node2 -> <null>}         |
| {node1 -> ["Trainer"], r -> "KNOWS", node2 -> ["Trainer"]} |
| {node1 -> ["Trainer"], r -> "HAVE", node2 -> ["Pokemon"]}  |
| {node1 -> ["Trainer"], r -> "COMES", node2 -> ["Region"]}  |
| {node1 -> ["Pokemon"], r -> "LIVES", node2 -> ["Region"]}  |
+-------------------------------------------------------------+
5 rows
```

2. **(1pkt) Napisać funkcje do tworzenia poszczególnych obiektów i relacji w grafie.**

   Węzły:

```
private Node createPokemon(GraphDatabaseService gdb, String name, String type,
double weight) {
    Node pokemon = gdb.createNode();
    pokemon.addLabel(() -> "Pokemon");
    pokemon.setProperty("name", name);
    pokemon.setProperty("type", type);
    pokemon.setProperty("weight", weight);
    return pokemon;
}

private Node createTrainer(GraphDatabaseService gdb, String name, String
lastName, int age, String gender) {
    Node trainer = gdb.createNode();
    trainer.addLabel(() -> "Trainer");
    trainer.setProperty("name", name);
    trainer.setProperty("lastName", lastName);
    trainer.setProperty("age", age);
    trainer.setProperty("gender", gender);
    return trainer;
}

private Node createRegion(GraphDatabaseService gdb, String name, String
professor, String regionVillains) {
    Node region = gdb.createNode();
    region.addLabel(() -> "Region");
    region.setProperty("name", name);
    region.setProperty("professor", professor);
    region.setProperty("regionVillains", regionVillains);
    return region;
}
```

Relacje:

```
private void createLivesRelationship(Node pokemon, Node region) {
    pokemon.createRelationshipTo(region, RelationshipType.withName("LIVES"));
}

private void createComesRelationship(Node trainer, Node region) {
    trainer.createRelationshipTo(region, RelationshipType.withName("COMES"));
}

private void createHaveRelationship(Node trainer, Node pokemon) {
    trainer.createRelationshipTo(pokemon, RelationshipType.withName("HAVE"));
}

private void createKnowsRelationship(Node trainer1, Node trainer2) {
    trainer1.createRelationshipTo(trainer2, RelationshipType.withName("KNOWS"));
    trainer2.createRelationshipTo(trainer1, RelationshipType.withName("KNOWS"));
}
```

3. **(1pkt) Napisać bardzo prosty populator danych. Może to być zwykły Main czy Unit-test. Może być bardzo prosty, ale tak, aby było po 5 różnych obiektów i 10 relacji każdego typu.**

Tutaj trochę nie przemyślałem tego co robię. Zamiast stworzyć prosty generator pomyślałem, że wypiszę na szybkości po kilka kombinacji ręcznie. Po ponad 60 linijkach kodu okazało się, że potrzeba tego więcej niż „kilka”. Nie kasowałem już tego co napisałem i dopisałem z bólem do końca inserty do bazy. Następnym razem będę tworzył od razu generator do danych.

Trochę tego jest. Przepraszam :(

```
public void createData() {
    GraphDatabaseService graph = graphDatabase.getGraphDatabaseService();
    try (Transaction transaction = graph.beginTx()) {

        //Regions
        Node kanto = createRegion(graph, "Kanto", "Oak", "Team Rocket");
        Node seviiIslands = createRegion(graph, "Sevii Islands", "None", "Team
Rocket");
        Node johto = createRegion(graph, "Johto", "Elm", "Team Rocket");
        Node hoenn = createRegion(graph, "Hoenn", "Birch", "Team Aqua and
Magma");
        Node sinnoh = createRegion(graph, "Sinnoh", "Rowan", "Team Galactic");
        Node unova = createRegion(graph, "Unova", "Juniper", "Team Plasma");
        Node kalos = createRegion(graph, "Kalos", "Sycamore", "Team Flare");
        Node alola = createRegion(graph, "Alola", "Kukui", "Team Skull and
Rainbow");

        //Trainers
        Node caroleH = createTrainer(graph, "Carole", "Hammond", 15, "female");
        Node sethH = createTrainer(graph, "Seth", "Hariss", 17, "female");
        Node dorothyB = createTrainer(graph, "Dorothy", "Barton", 22, "female");
        Node constanceS = createTrainer(graph, "Constance", "Silva", 43,
"female");
        Node danaG = createTrainer(graph, "Dana", "Garner", 19, "female");
        Node brytanB = createTrainer(graph, "Bryant", "Beck", 25, "male");
        Node kyleS = createTrainer(graph, "Kyle", "Sherman", 54, "male");
        Node alanW = createTrainer(graph, "Alan", "Williams", 26, "male");
        Node mauelL = createTrainer(graph, "Manuel", "Larson", 18, "male");
        Node garryH = createTrainer(graph, "Garry", "Holt", 14, "male");
```

```
        //Pokemon
        Node bulbasaur = createPokemon(graph, "Bulbasaur", "grass", 44.2);
        Node charmander = createPokemon(graph, "Charmander", "fire", 55.6);
        Node squirtle = createPokemon(graph, "Squirtle", "water", 52.2);
        Node chikorita = createPokemon(graph, "Chikorita", "grass", 42.5);
        Node totodile = createPokemon(graph, "Totodile", "water", 40.2);
        Node skuntank = createPokemon(graph, "Skuntank", "poison", 22.5);
        Node mudkip = createPokemon(graph, "Mudkip", "water", 32.2);
        Node torchic = createPokemon(graph, "Torchic", "fire", 18.2);
        Node metagross = createPokemon(graph, "Metagross", "rock", 734.3);
        Node gible = createPokemon(graph, "Gible", "rock", 73.3);
        Node oshawoot = createPokemon(graph, "Oshawott", "water", 53.3);
        Node landours = createPokemon(graph, "Landorus", "ground", 152.2);
        Node cottonee = createPokemon(graph, "Cottonee", "fairy", 12.6);
        Node chespin = createPokemon(graph, "Chespin", "grass", 22.2);
        Node fennkein = createPokemon(graph, "Fennekin", "fire", 24.2);
        Node wingull = createPokemon(graph, "Wingull", "flying", 30.0);

        //Relations
        createLivesRelationship(bulbasaur, kanto);
        createLivesRelationship(charmander, kanto);
        createLivesRelationship(squirtle, kanto);
        createLivesRelationship(chikorita, johto);
        createLivesRelationship(totodile, johto);
        createLivesRelationship(skuntank, johto);
        createLivesRelationship(mudkip, hoenn);
        createLivesRelationship(torchic, hoenn);
        createLivesRelationship(metagross, sinnoh);
        createLivesRelationship(gible, sinnoh);
        createLivesRelationship(oshawoot, unova);
        createLivesRelationship(landours, unova);
        createLivesRelationship(cottonee, unova);
        createLivesRelationship(chespin, kalos);
        createLivesRelationship(fennkein, kalos);
        createLivesRelationship(wingull, alola);

        createComesRelationship(caroleH, kanto);
        createComesRelationship(sethH, seviiIslands);
        createComesRelationship(dorothyB, johto);
        createComesRelationship(constanceS, hoenn);
        createComesRelationship(danaG, sinnoh);
        createComesRelationship(brytanB, sinnoh);
        createComesRelationship(kyleS, unova);
        createComesRelationship(alanW, kalos);
        createComesRelationship(mauelL, alola);
        createComesRelationship(garryH, seviiIslands);

        createHaveRelationship(caroleH, fennkein);
        createHaveRelationship(sethH, totodile);
        createHaveRelationship(sethH, mudkip);
        createHaveRelationship(dorothyB, wingull);
        createHaveRelationship(constanceS, oshawoot);
        createHaveRelationship(constanceS, chespin);
        createHaveRelationship(constanceS, metagross);
        createHaveRelationship(danaG, squirtle);
        createHaveRelationship(danaG, skuntank);
        createHaveRelationship(brytanB, cottonee);
        createHaveRelationship(brytanB, bulbasaur);
        createHaveRelationship(kyleS, gible);
        createHaveRelationship(kyleS, charmander);
        createHaveRelationship(alanW, charmander);
        createHaveRelationship(alanW, chikorita);
        createHaveRelationship(alanW, torchic);
        createHaveRelationship(mauelL, landours);
```

```
        createHaveRelationship(mauelL, totodile);
        createHaveRelationship(garryH, chikorita);
        createHaveRelationship(garryH, mudkip);
        createHaveRelationship(garryH, gible);
        createHaveRelationship(garryH, chespin);

        createKnowsRelationship(caroleH, sethH);
        createKnowsRelationship(caroleH, dorothyB);
        createKnowsRelationship(sethH, alanW);
        createKnowsRelationship(dorothyB, alanW);
        createKnowsRelationship(dorothyB, garryH);
        createKnowsRelationship(constanceS, danaG);
        createKnowsRelationship(danaG, kyleS);
        createKnowsRelationship(danaG, mauelL);
        createKnowsRelationship(brytanB, garryH);
        createKnowsRelationship(alanW, caroleH);
        createKnowsRelationship(alanW, brytanB);
        createKnowsRelationship(mauelL, constanceS);
        createKnowsRelationship(mauelL, dorothyB);
        createKnowsRelationship(garryH, mauelL);
        createKnowsRelationship(garryH, danaG);
        createKnowsRelationship(garryH, sethH);

        transaction.success();
    }
}
```

4. **(1pkt) Napisać funkcję do pobrania wszystkich relacji dla danego węzła.**

Stworzyłem dwie funkcje. Jedna szukająca noda po id, druga po nazwie.

```
public String allRelationshipsForNodeByName(String label, String name) {
   return graphDatabase.runCypher(String.format("MATCH (n:%s {name: '%s'})-[r]-(m) RETURN r,n,m", label,
name));
}
public String allRelationshipsForNodeByID(int id) {
   return graphDatabase.runCypher(String.format("MATCH (n)-[r]-(m) WHERE ID(n) = %d RETURN r,n,m",
id));
}
```

Przykładowe wywołanie:

```
System.out.printf(graph.allRelationshipsForNodeByName("Pokemon", "Totodile"));
System.out.printf(graph.allRelationshipsForNodeByID(4));
```

Output:

```
+------------------------------------------------------------------------------------------------------------------------+
| r          | n                                                    | m                                                                    |
+------------------------------------------------------------------------------------------------------------------------+
| :HAVE[43]{} | Node[22]{weight:40.2,name:"Totodile",type:"water"} | Node[16]{age:18,gender:"male",name:"Manuel",lastName:"Larson"}       |
| :HAVE[27]{} | Node[22]{weight:40.2,name:"Totodile",type:"water"} | Node[9]{gender:"female",name:"Seth",lastName:"Hariss",age:17}        |
| :LIVES[4]{} | Node[22]{weight:40.2,name:"Totodile",type:"water"} | Node[2]{regionVillains:"Team Rocket",name:"Johto",professor:"Elm"} |
+------------------------------------------------------------------------------------------------------------------------+
3 rows
+--------------------------------------------------------------------------------------------------------------------------------------------+
| r            | n                                                                    | m                                                                    |
+--------------------------------------------------------------------------------------------------------------------------------------------+
| :COMES[20]{} | Node[4]{regionVillains:"Team Galactic",name:"Sinnoh",professor:"Rowan"} | Node[12]{gender:"female",name:"Dana",lastName:"Garner",age:19} |
| :COMES[21]{} | Node[4]{regionVillains:"Team Galactic",name:"Sinnoh",professor:"Rowan"} | Node[13]{gender:"male",name:"Bryant",lastName:"Beck",age:25}   |
| :LIVES[9]{}  | Node[4]{regionVillains:"Team Galactic",name:"Sinnoh",professor:"Rowan"} | Node[27]{weight:73.3,name:"Gible",type:"rock"}                 |
| :LIVES[8]{}  | Node[4]{regionVillains:"Team Galactic",name:"Sinnoh",professor:"Rowan"} | Node[26]{weight:734.3,name:"Metagross",type:"rock"}            |
+--------------------------------------------------------------------------------------------------------------------------------------------+
4 rows
```

5. **(2pkt) Napisać funkcje do znalezienia ścieżki dla danych dwóch węzłów.**

Stworzyłem dwie funkcje. Jedna szuka ścieżki po nazwie, druga po id nodów.

```
public String pathBetweenNodesByName(String label1, String name1, String label2,
String name2) {
    return graphDatabase.runCypher(String.format("MATCH (n:%s), (m:%s),
p=shortestPath((n)-[*..]-(m)) " +
            "WHERE n.name = \"%s\" AND m.name = \"%s\" " +
            "RETURN p",label1, label2, name1, name2));
}

public String pathBetweenNodesByID(int node1ID, int node2ID) {
    return graphDatabase.runCypher(String.format("MATCH (n), (m),
p=shortestPath((n)-[*..]-(m)) " +
            "WHERE ID(n) = %d AND ID(m) = %d " +
            "RETURN p",node1ID, node2ID));
}
```

Przykładowe wywołanie:

```
System.out.printf(graph.pathBetweenNodesByID(4,5));
System.out.printf(graph.pathBetweenNodesByName("Region", "Sinnoh",
        "Region", "Unova"));
```

Output:

```
+-------------------------------------------------------------------------------------------------
| p
+-------------------------------------------------------------------------------------------------
| [Node[4]{regionVillains:"Team Galactic",name:"Sinnoh",professor:"Rowan"},:COMES[20]{},
+-------------------------------------------------------------------------------------------------
1
 ---------------------------------------------------------------------------
 Node[12]{gender:"female",name:"Dana",lastName:"Garner",age:19},:KNOWS[60]{},
 ---------------------------------------------------------------------------

 ---------------------------------------------------------------------------
 ,Node[14]{gender:"male",name:"Kyle",lastName:"Sherman",age:54}
 ----------------------------------------------------------------------------------------------+
                                                                                               |
 ----------------------------------------------------------------------------------------------+
 ,:COMES[22]{},Node[5]{regionVillains:"Team Plasma",name:"Unova",professor:"Juniper"}] |
 ----------------------------------------------------------------------------------------------+
```