

Bazy danych – NoSQL

Neo4j – zadania

Autor zadań: Piotr Wróbel

Data laboratorium: 4.12.2019 r.

Data wykonania: 9.01.2020 r.

1. Zaimplementować funkcję – zaimplementowano przykładowe zapytanie MATCH zwracające tytuły filmów zawierające wyraz „You”, kod w Javie przedstawiono poniżej, a wynik wywołania na Rysunku 1:

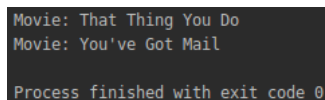
```
public class Main {

    static Connection connection;

    public static void main(String[] args) {
        try {
            connection = DriverManager.getConnection("jdbc:neo4j:http://localhost:7474", "neo4j", "mojeHasło");
            exampleMatch();
            connection.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public static void exampleMatch() throws SQLException {
        String query = "MATCH (m:Movie) WHERE m.title CONTAINS 'You' RETURN m.title";
        PreparedStatement statement = connection.prepareStatement(query);
        ResultSet rs = statement.executeQuery();
        while(rs.next()){
            System.out.println("Movie: " + rs.getString("m.title"));
        }

        statement.close();
        rs.close();
    }
}
```



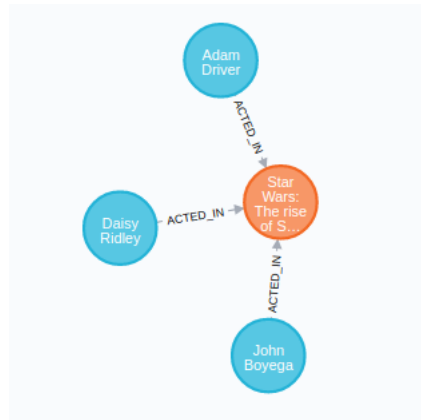
```
Movie: That Thing You Do
Movie: You've Got Mail
Process finished with exit code 0
```

Rysunek 1: Wynik wywołania przykładowego polecenia MATCH z poziomu Javy

2. Stworzyć kilka nowych węzły reprezentujących film oraz aktorów w nim występujących, następnie stworzyć relacje ich łączące (np. ACTED_IN). Utworzono węzeł odpowiadający filmowi i węzły trzech aktorów w nim grających, użyty kod przedstawiono poniżej, a wizualizację wyniku na Rysunku 2

```
CREATE (m:Person { name: 'Daisy Ridley' })
CREATE (m:Person { name: 'Adam Driver' })
CREATE (m:Person { name: 'John Boyega' })
CREATE (m:Movie { title: 'Star Wars: The rise of Skywalker' })
MATCH (a:Person), (m:Movie) WHERE a.name = 'Daisy Ridley' AND m.title = 'Star Wars: The rise of Skywalker' CREATE (a)-[r:ACTED_IN]->(m)
MATCH (a:Person), (m:Movie) WHERE a.name = 'Adam Driver' AND m.title = 'Star Wars: The rise of Skywalker' CREATE (a)-[r:ACTED_IN]->(m)
```

```
MATCH (a:Person), (m:Movie) WHERE a.name = 'John Boyega' AND m.title = 'Star Wars: The rise of Skywalker' CREATE (a)-[r:ACTED_IN]->(m)
```



Rysunek 2: Utworzone nowe węzły bazy

3. Dodać zapytaniem nowe właściwości nowo dodanych węzłów reprezentujących aktor (np. *birthdate* oraz *birthplace*). Zapytania przedstawiono poniżej, a efekt na Rysunku 3

```

MATCH (a:Person) WHERE a.name = 'Daisy Ridley' SET a.birthplace = 'Londyn', a.birthdate = '10-04-1992'
MATCH (a:Person) WHERE a.name = 'Adam Driver' SET a.birthplace = 'San Diego', a.birthdate = '19-11-1983'
MATCH (a:Person) WHERE a.name = 'John Boyega' SET a.birthplace = 'Londyn', a.birthdate = '17-03-1992'

```

```
$ match (m:Movie), (a:Person) where m.title contains "Star Wars" and (a)-[:ACTED_IN]->(m) return a, m
```



Rysunek 3: Efekt dodania wartości atrybutów *birthplace* i *birthdate* dla aktorów.

4. Ułożyć zapytanie, które zmieni wartość atrybutu węzłów danego typu, jeżeli innych atrybut węzła spełnia zadane kryterium. Ułożono zapytanie, które zmienia wartość atrybutu *tagline* na „nowy tagline” dla filmów wydanych w roku 1996 (efekt wywołania przedstawia Rysunek 4):

```
MATCH (m:Movie) WHERE m.released = 1996 SET m.tagline = 'nowy tagline' RETURN m
```



Rysunek 4: Efekt modyfikacji atrybutu *tagline* dla filmów wydanych w 1996 roku.

5. Zapytanie o aktorów którzy grali w conajmniej 2 filmach (użyć collect i length) (Rysunek 5):

```
MATCH (p:Person)-[r:ACTED_IN]->(m:Movie) WITH p, length(collect(r)) AS films WHERE films
>= 2 return p
```



Rysunek 5: Efekt wyszukania aktorów, którzy grali w conajmniej 2 filmach.

i policzyć średnią wystąpień w filmach dla grupy aktorów, którzy wystąpili w conajmniej 3 filmach (Rysunek 6):

```
match (p:Person)-[r:ACTED_IN]->(m:Movie) with p, length(collect(r)) as films where films
>= 3 return avg(films)
```

```
$ match (p:Person)-[r:ACTED_IN]->(m:Movie) with p, length(collect(r)) as films where films ≥ 3 return avg(films)
```

avg(films)
4.333333333333334

Rysunek 6: Średnia ilości ról dla aktorów, którzy grali w conajmniej 3 filmach.

6. Zmienić wartość wybranego atrybutu w węzłach na ścieżce pomiędzy dwoma podanymi węzłami (wynik - Rysunek 7):

```
match (p:Person)-[r:ACTED_IN]->(m:Movie) where p.name = 'Rita Wilson' and m.title contains 'Sleepless' set r.roles = 'zmiana roli' return p, r, m
```

```
$ match (p:Person)-[r:ACTED_IN]->(m:Movie) where p.name = 'Rita Wilson' and m.title contains 'Sleepless' set r.roles = 'zmiana roli' return p, r, m
```

Graph

ACTED_IN <id>: 104 roles: zmiana roli

Rysunek 7: Efekt zmiany wartości atrybutu relacji.

7. Wyświetlić węzły, które znajdują się na 2 miejscu na ścieżkach o długości 4 pomiędzy dwoma wybranymi węzłami (Rysunek 8).

```
match p = (a:Person)-[*4]-(b:Person) where a.name contains 'Keanu Reeves' and b.name contains 'Milos Forman' return nodes(p)[2]
```

```
$ match p = (a:Person)-[*4]-(b:Person) where a.name contains 'Keanu Reeves' and b.name contains 'Milos Forman' return nodes(p)[2]
```

nodes(p)[2]
<pre>{ "name": "Jack Nicholson", "born": 1937 }</pre>

Started streaming 1 records after 1 ms and completed after 3 ms.

Rysunek 8: Drugi element 4-elementowej ścieżki.

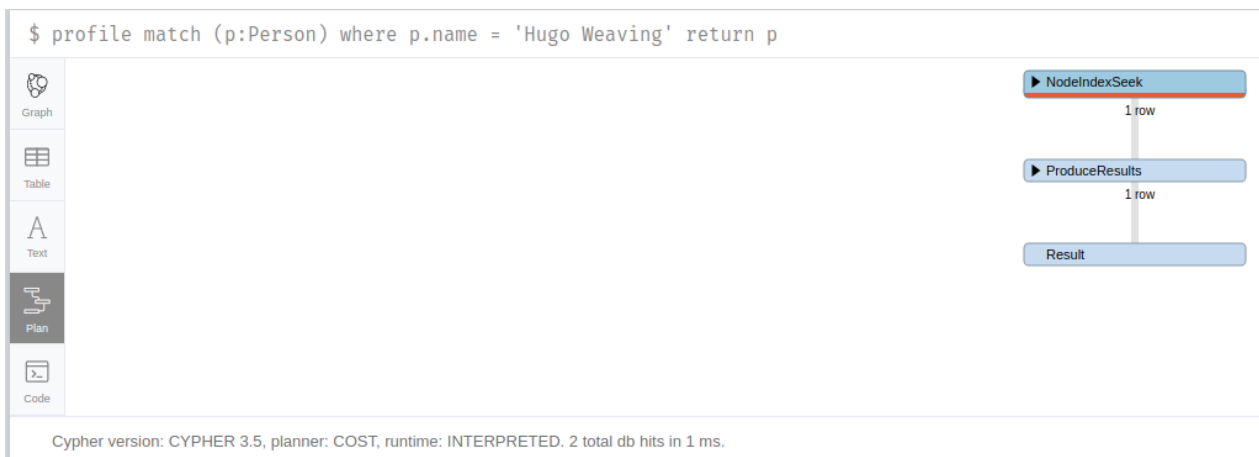
8. Porównać czas wykonania zapytania o wybranego aktora bez oraz z indeksem w bazie nałożonym na atrybut name (DROP INDEX i CREATE INDEX oraz użyć komendy PROFILE/EXPLAIN).

- Wynik szukania bez indeksu – Rysunek 9



Rysunek 9: Wyszukiwanie aktora po imieniu bez założonego indeksu.

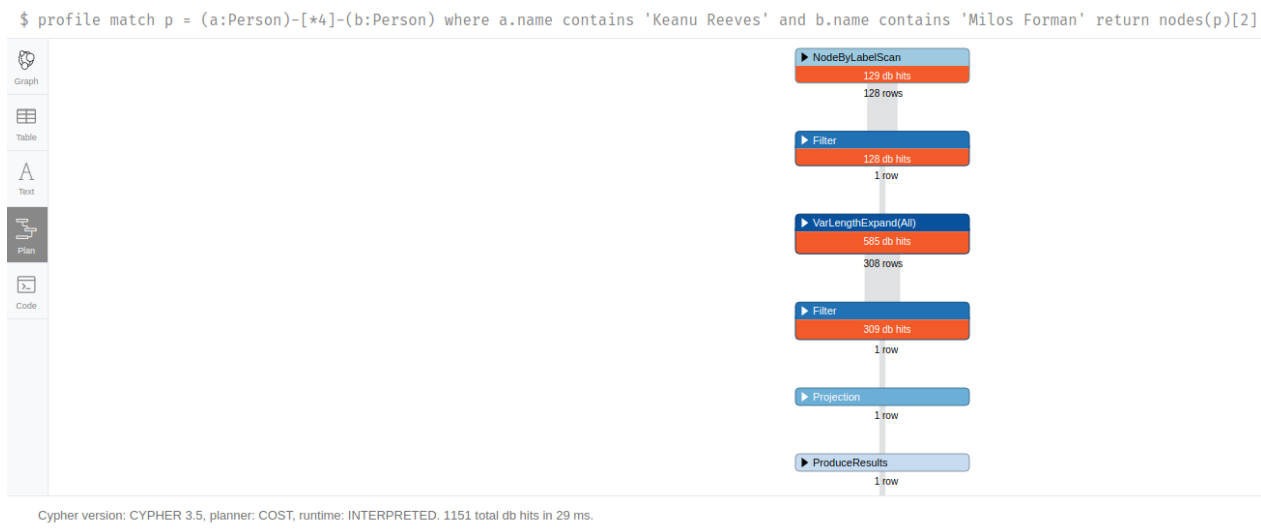
- Wynik szukania z indeksem – Rysunek 10



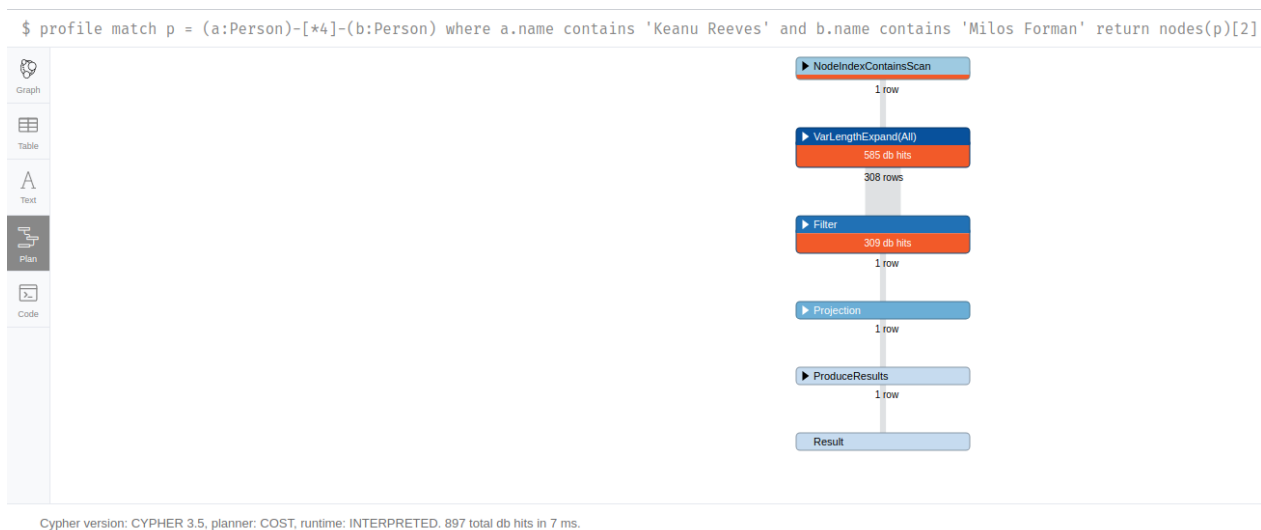
Rysunek 10: Wyszukiwanie aktora po imieniu z założonym indeksem.

9. Spróbować dokonać optymalizacji wybranych dwóch zapytań z poprzednich zadań

- efekt założenia indeksu na atrybut name dla polecenia wyświetlenia węzłów na 2 miejscu ścieżki o długości 4 – wyniki Rysunek 11 i 12



Rysunek 11: Drugi element 4-elementowej ścieżki bez założonego indeksu.



Rysunek 12: Drugi element 4-elementowej ścieżki z założonym indeksem.

- efekt założenia indeksu na atrybuty released i tagline filmu dla zapytania z punktu, w którym zmieniano wartość atrybutu węzłów – wyniki Rysunek 13 i 14.

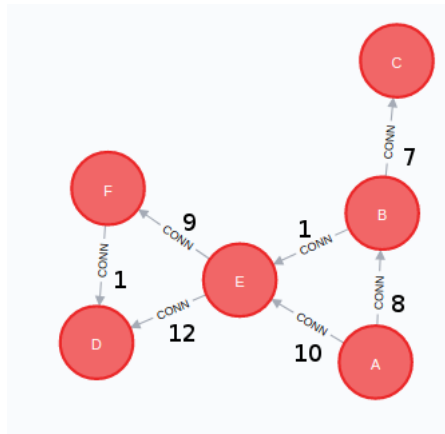


Rysunek 13: Zmiana atrybutu filmów bez indeksu na atrybucie *released*.



Rysunek 14: Zmiana atrybutu filmów z indeksem na atrybucie *released*.

10. Napisać kod, które wygeneruje drzewo rozpinające z bazy



Rysunek 15: Graf użyty w zadaniu, do krawędzi dopisano ich koszty

Wygenerowano przykładowy mały graf skierowany o strukturze przedstawionej na Rysunku 15 (nad krawędziami pokazane są koszty połączeń). W bazie danych każdy wierzchołek ma typ Edge, właściwość label, a każda relacja jest rodzaju CONN i ma właściwości cost i mst (mst jest pomocnicza do wyznaczenia drzewa rozpinającego).

Przykładowe zapytania dodające dane do bazy:

```
CREATE (a:Elem { label = 'A' })
MATCH (a:Elem), (b:Elem) where a.label = 'A' AND b.label = 'B' CREATE (a)-[:CONN]->(b)
match (a:Elem)-[e:CONN]->(b:Elem) where a.label = 'A' AND b.label = 'B' set e.cost = 8
```

W Javie wierzchołki są reprezentowane przez klasę Node:

```
public class Node {
    private String label;
    private Set<Edge> neighbours;

    public Node(String label) {
        this.label = label;
        this.neighbours = new HashSet<>();
    }

    public String getLabel() {
        return label;
    }

    public Set<Edge> getNeighbours() {
        return neighbours;
    }

    public void addEdge(Edge edge) {
        this.neighbours.add(edge);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Node node = (Node) o;
        return label.equals(node.label);
    }
}
```



```

@Override
public int hashCode() {
    return Objects.hash(label);
}
}

```

a krawędzie przez klasę Edge:

```

public class Edge implements Comparable<Edge> {
    private Node start;
    private Node stop;
    private int cost;

    public Edge(Node start, Node stop, int cost) {
        this.start = start;
        this.stop = stop;
        this.cost = cost;

        start.addEdge(this);
    }

    public Node getStart() {
        return start;
    }

    public Node getStop() {
        return stop;
    }

    public int getCost() {
        return cost;
    }

    @Override
    public int compareTo(Edge o) {
        return this.cost - o.getCost();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Edge edge = (Edge) o;
        return cost == edge.cost &&
            start.equals(edge.start) &&
            stop.equals(edge.stop);
    }

    @Override
    public int hashCode() {
        return Objects.hash(start, stop, cost);
    }
}

```

W głównej funkcji programu najpierw czytane są z bazy dane o wierzchołkach i krawędziach, a następnie jest tworzone

minimalne drzewo rozpinające z użyciem algorytmu Prima. Krawędzie należące do drzewa mają zmieniony atrybut mst na wartość 1. Kod głównej klasy:

```
public class Main {

    static Connection connection;

    public static void main(String[] args) {
        try {
            connection = DriverManager.getConnection("jdbc:neo4j:http://localhost:7474", "
                neo4j", "mojeHasło");

            List<Node> nodes = new LinkedList<>();
            readNodes(nodes);
            readEdges(nodes);

            int indexOfA = nodes.indexOf(new Node("A"));
            Node startingNode = nodes.get(indexOfA);
            nodes.remove(startingNode);

            PriorityQueue<Edge> edgesQueue = new PriorityQueue<>(startingNode.getNeighbours
                ());
            while(!nodes.isEmpty()) {
                Edge nextEdge = edgesQueue.poll();
                if(nodes.contains(nextEdge.getStop())) {
                    Node stopNode = nextEdge.getStop();
                    nodes.remove(stopNode);
                    edgesQueue.addAll(stopNode.getNeighbours());
                    markEdge(nextEdge);
                }
            }

            connection.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public static void markEdge(Edge edge) throws SQLException {
        String query = String.format("MATCH (a:Elem)-[r:CONN]->(b:Elem) WHERE a.label = '%s'
            AND b.label = '%s' SET r.mst = 1",
            edge.getStart().getLabel(), edge.getStop().getLabel());
        Statement statement = connection.createStatement();
        statement.execute(query);
    }

    public static void readNodes(List<Node> nodes) throws SQLException {
        String query = "MATCH (e:Elem) RETURN e.label";
        PreparedStatement statement = connection.prepareStatement(query);
        ResultSet rs = statement.executeQuery();

        while(rs.next()) {
            nodes.add(new Node(rs.getString("e.label")));
        }
    }
}
```

```

public static void readEdges(List<Node> nodes) throws SQLException {
    String query = "match (a:Elem)-[r:CONN]->(b:Elem) return a.label, b.label, r.cost";
    PreparedStatement statement = connection.prepareStatement(query);
    ResultSet rs = statement.executeQuery();

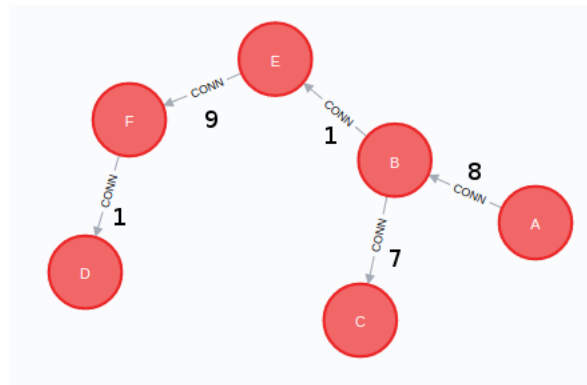
    while(rs.next()) {
        int startIndex = nodes.indexOf(new Node(rs.getString("a.label")));
        Node start = nodes.get(startIndex);

        int stopIndex = nodes.indexOf(new Node(rs.getString("b.label")));
        Node stop = nodes.get(stopIndex);

        Edge edge = new Edge(start, stop, rs.getInt("r.cost"));
    }
}
}

```

W celu wygodnej wizualizacji uzyskanego drzewa, usunięto z bazy krawędzie które do drzewa nie należą i wywołano odpowiednie zapytanie MATCH – wynik przedstawia Rysunek 16.



Rysunek 16: Minimalne drzewo rozpinające, do krawędzi dopisano ich koszty