Bazy Danych - NoSQL

Neo4j - zadania

Dawid Majchrowski
Data Laboratorium : 4.12.2019
Data Wykonania: 28.12.2019

1. Zainstalować serwer neo4j lokalnie.
   - Lokalny serwer instalujemy korzystając z poniższego dockera. (poniżej plik konfiguracyjny docker-compose.yml, zawiera również nazwę użytkownika=neo4j i hasło=123)

docker-compose.yml

```yaml
version: '3'
services:
  neo4j:
    image: neo4j:3.4
    hostname: neo4j
    container_name: neo4j
    ports:
    - "7474:7474"
    - "7687:7687"
    environment:
      NEO4J_AUTH: neo4j/123
      NEO4J_dbms_logs_debug_level: DEBUG
```

2. Wgrać bazę.(Zgodnie z poleceniem wgrywamy bazę w kontenerze dockera korzystając z cypher-shella, tzn wywołując komendy podane w zadaniu)

```
neo4j> LOAD CSV WITH HEADERS FROM 'https://neo4j.com/docs/cypher-manual/3.5/csv/query-tuning/movies.csv' AS
       line
       MERGE (m:Movie { title: line.title })
       ON CREATE SET m.released = toInteger(line.released), m.tagline = line.tagline;
0 rows available after 1230 ms, consumed after another 0 ms
neo4j> LOAD CSV WITH HEADERS FROM 'https://neo4j.com/docs/cypher-manual/3.5/csv/query-tuning/actors.csv' AS
       line
       MATCH (m:Movie { title: line.title })
       MERGE (p:Person { name: line.name })
       ON CREATE SET p.born = toInteger(line.born)
       MERGE (p)-[:ACTED_IN { roles:split(line.roles, ';')}]->(m);
0 rows available after 1502 ms, consumed after another 0 ms
Added 102 nodes, Created 172 relationships, Set 375 properties, Added 102 labels
neo4j> LOAD CSV WITH HEADERS FROM 'https://neo4j.com/docs/cypher-manual/3.5/csv/query-tuning/directors.csv'
       AS line
       MATCH (m:Movie { title: line.title })
       MERGE (p:Person { name: line.name })
       ON CREATE SET p.born = toInteger(line.born)
       MERGE (p)-[:DIRECTED]->(m);
0 rows available after 846 ms, consumed after another 0 ms
Added 23 nodes, Created 44 relationships, Set 46 properties, Added 23 labels
neo4j>
```

   - Na tym etapie decydujemy się na wybór języka python
   - Poniższa konfiguracja będzie stosowana we wszystkich zadaniach

```python
!pip3 install neo4j
from neo4j import GraphDatabase

uri = "bolt://localhost:7687"
driver = GraphDatabase.driver(uri, auth=("neo4j", "123"))
```

3. Zaimplementować funkcję(dowolną, krótką)
   Funkcja pobierająca z bazy podaną ilość filmów.

```
def get_movies(limit):
    with driver.session() as session:
        return session.run("MATCH (n:Movie) "
                           "RETURN n "
                           "LIMIT $limit" , limit=limit)

for item in get_movies(4):
    print(item)
```

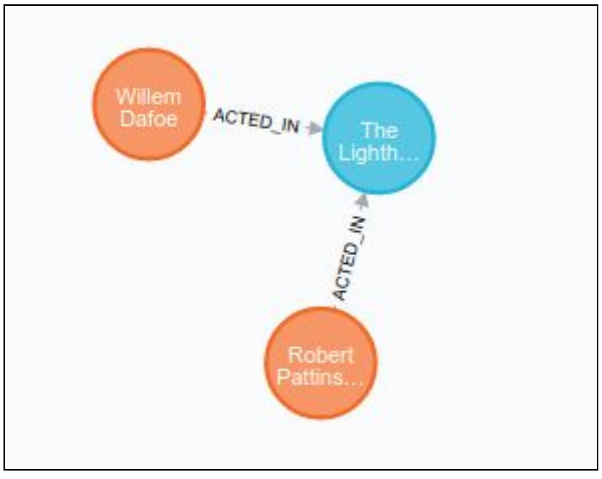
```
<Record n=<Node id=0 labels={'Movie'} properties={'title': "Something's Gotta Give", 'tagline': 'null', 'released
': 1975}>>
<Record n=<Node id=1 labels={'Movie'} properties={'title': 'Johnny Mnemonic', 'tagline': 'The hottest data on eart
h. In the coolest head in town', 'released': 1995}>>
<Record n=<Node id=2 labels={'Movie'} properties={'title': 'The Replacements', 'tagline': 'Pain heals, Chicks dig
scars... Glory lasts forever', 'released': 2000}>>
<Record n=<Node id=3 labels={'Movie'} properties={'title': "The Devil's Advocate", 'tagline': 'Evil has its winnin
g ways', 'released': 1997}>>
```

4. Stworzyć kilka nowych węzły reprezentujących film oraz aktorów w nim występujących, następnie stworzyć relacje ich łączące (np. ACTED_IN)
   - Tworzymy film oraz 2 akuratów w nim grających oraz łączymy ich relacją

```
def create_movie(tx, title, tagline, realeased=2019):
    return tx.run("CREATE (m:Movie {title:$title, tagline:$tagline, realeased:$realeased}) "
                  "RETURN id(m)", title=title, tagline=tagline, realeased=realeased).single().value()

def create_person(tx, name, born):
    return tx.run("CREATE (p:Person {name:$name, born:$born}) "
                  "RETURN id(p)", name=name, born=born).single().value()

def create_relation(tx, title, name):
    return tx.run("MATCH (p:Person), (m:Movie) "
                  "WHERE p.name=$name AND m.title=$title "
                  "CREATE (p)-[r:ACTED_IN]->(m) "
                  "RETURN r", title=title, name=name).single().value()

with driver.session() as session:
    print(session.write_transaction(create_movie, "The Lighthouse", "Light the house"))
    print(session.write_transaction(create_person, "Willem Dafoe", 1955))
    print(session.write_transaction(create_person, "Robert Pattinson", 1986))
    print(session.write_transaction(create_relation, "The Lighthouse", "Willem Dafoe"))
    print(session.write_transaction(create_relation, "The Lighthouse", "Robert Pattinson"))
```

```
187
190
193
<Relationship id=2746 nodes=(<Node id=190 labels=set() properties={}>, <Node id=187 labels=set() properties={}>) t
ype='ACTED_IN' properties={}>
<Relationship id=2747 nodes=(<Node id=193 labels=set() properties={}>, <Node id=187 labels=set() properties={}>) t
ype='ACTED_IN' properties={}>
```

   - Wynik w bazie

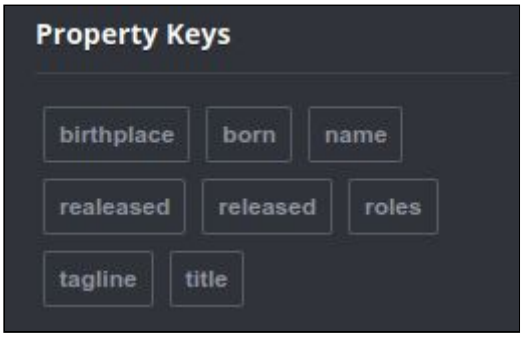


5. Dodać zapytaniem nowe właściwości nowo dodanych węzłów reprezentujących aktor (np.birthdate oraz birthplace).
    - Tworzymy funkcję która, która tworzy nowe pola lub aktualizuje jeżeli, któreś już istnieje.

```
def set_person_born_and_birthplace(name, born, birthplace):
     with driver.session() as session:
        return session.run("MATCH (p:Person) "
                           "WHERE p.name = $name "
                           "SET p += {born: $born, birthplace: $birthplace} "
                           "RETURN id(p)",
                           name=name, born=born, birthplace=birthplace).single().value()

print(set_person_born_and_birthplace("Willem Dafoe", 1956, "Appleton"))
```

```
190
```

Poniżej widzimy, że w bazie danych zostało dodane nowe property “birthplace”



Poniżej widzimy, że Dla Willema, dodane została właściwość birthplace oraz zaktualizowana born.


```
{
  "birthplace": "Appleton",
  "name": "Willem Dafoe",
  "born": 1956
}
```


6. Ułożyć zapytanie, które zmieni wartość atrybutu węzłów danego typu, jeżeli innych atrybut węzła spełnia zadane kryterium
    - Jeżeli tytuł filmu zaczyna się na podany ciąg znaków, to zwiększamy tagline, żeby był pisany dużymi literami

```
def to_upper_if_starts_with(starts_with):
    with driver.session() as session:
        return session.run("MATCH (m:Movie) "
                           "WHERE m.title STARTS WITH $starts_with "
                           "SET m.tagline = toUpper(m.tagline) "
                           "RETURN m",
                           starts_with=starts_with)

for item in to_upper_if_starts_with("The"):
    print(item)
```

```
<Record m=<Node id=2 labels={'Movie'} properties={'title': 'The Replacements', 'tagline': 'PAIN HEALS, CHICKS DIG
SCARS... GLORY LASTS FOREVER', 'released': 2000}>>
<Record m=<Node id=3 labels={'Movie'} properties={'title': "The Devil's Advocate", 'tagline': 'EVIL HAS ITS WINNIN
G WAYS', 'released': 1997}>>
<Record m=<Node id=4 labels={'Movie'} properties={'title': 'The Matrix Revolutions', 'tagline': 'EVERYTHING THAT H
AS A BEGINNING HAS AN END', 'released': 2003}>>
<Record m=<Node id=5 labels={'Movie'} properties={'title': 'The Matrix Reloaded', 'tagline': 'FREE YOUR MIND', 're
leased': 2003}>>
```

7. Zapytanie o aktorów którzy grali w conajmniej 2 filmach (użyć collect i length) i policzyć średnią wystąpień w filmach dla grupy aktorów, którzy wystąpili w conajmniej 3 filmach.

```
def at_least_two_movies():
    with driver.session() as session:
        return session.run("MATCH (p:Person) -[:ACTED_IN] -> (m:Movie) "
                           "WITH p, length(collect(m)) as ct "
                           "WHERE ct > 1 "
                           "RETURN length(collect(p))").single().value()

def calc_avg():
    with driver.session() as session:
        return session.run("MATCH (p:Person) -[:ACTED_IN] -> (m:Movie) "
                           "WITH p, length(collect(m)) as ct "
                           "WHERE ct > 2 "
                           "RETURN avg(ct)").single().value()

print(at_least_two_movies())
print(calc_avg())
```

```
35
4.333333333333333
```

8. (błąd numeracji)
9. Zmienić wartość wybranego atrybutu w węzłach na ścieżce pomiędzy dwoma podanymi węzłami
    - Na najkrótszej ścieżce między 2 osobami, dla każdego węzła dodajemy atrybut keanupath ustawiony na True.
    - Zauważmy, że używamy shortestPath, która zwraca tylko jedną najkrótszą ścieżkę między 2 węzłami

```
def update_nodes(name1, name2):
    with driver.session() as session:
       return session.run("MATCH path = shortestPath((p1:Person)-[*]-(p2:Person)) "
                          "WHERE p1.name=$name1 AND p2.name=$name2 "
                          "UNWIND nodes(path) as n "
                          "SET n.keanupath = True "
                          "RETURN n"
                          ,name1=name1, name2=name2).value()

print(update_nodes("Keanu Reeves", "Tom Hanks"))
```

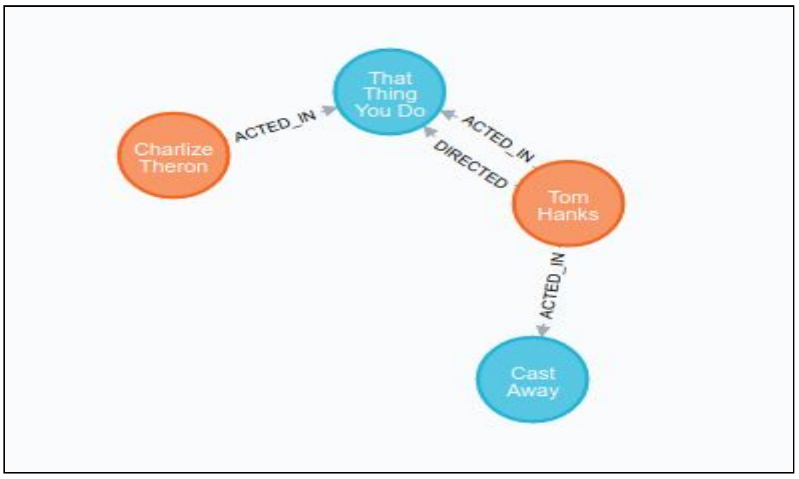
```
<Node id=40 labels={'Person'} properties={'keanupath': True, 'name': 'Keanu Reeves', 'born': 1964}>, <Node id=6 l
abels={'Movie'} properties={'tagline': 'WELCOME TO THE REAL WORLD', 'keanupath': True, 'title': 'The Matrix', 'rel
eased': 1999}>, <Node id=38 labels={'Person'} properties={'keanupath': True, 'name': 'Andy Wachowski', 'born': 196
7}>, <Node id=8 labels={'Movie'} properties={'tagline': 'Everything is connected', 'keanupath': True, 'title': 'Cl
oud Atlas', 'released': 2012}>, <Node id=88 labels={'Person'} properties={'keanupath': True, 'name': 'Tom Hanks',
born': 1956}>]
```

10. Wyświetlić węzły, które znajdują się na 2 miejscu na ścieżkach o długości 4 pomiędzy dwoma wybranymi węzłami.
    - Ścieżka o długości 4 wymaga 2 relacji pomiędzy, zatem (dla naszej bazy) muszą to być 2 różne węzły (person, movie) gdyż węzły mogą występować tylko na przemian
    - Zauważmy, że w naszym zapytaniu zwracamy wszystkie ścieżki o długości 4 między 2 węzłami

In [151]:
```
def find_second(name, title):
     with driver.session() as session:
        return session.run("MATCH path=(p:Person)-[*3]-(m:Movie) "
                           "WHERE p.name=$name  AND m.title=$title "
                           "RETURN nodes(path)[1]", name=name, title=title).value()

find_second("Charlize Theron", "Cast Away")
```

Out[151]:
```
[<Node id=11 labels={'Movie'} properties={'title': 'That Thing You Do', 'tagline': 'In every life there comes a ti
me when that thing you dream becomes that thing you do', 'released': 1996}>,
 <Node id=11 labels={'Movie'} properties={'title': 'That Thing You Do', 'tagline': 'In every life there comes a ti
me when that thing you dream becomes that thing you do', 'released': 1996}>]
```

Sprawdzamy, czy zgadza się to ze ścieżkami w bazie



11. 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).
    - Do mierzenia czasu wykorzystujemy poniższy dekorator

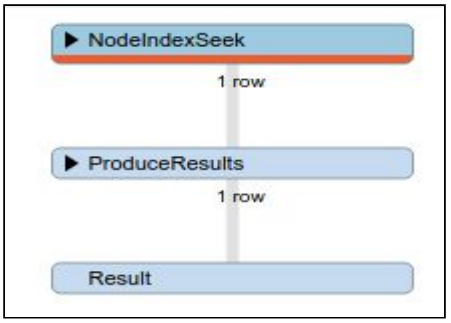
```
from functools import wraps
from time import time

def timing(f):
    @wraps(f)
    def wrap(*args, **kw):
        ts = time()
        result = f(*args, **kw)
        te = time()
        print('func:%r args: %r took: %3.6f ms' % \
          (f.__name__, args, (te-ts)*1000))
        return result
    return wrap
```

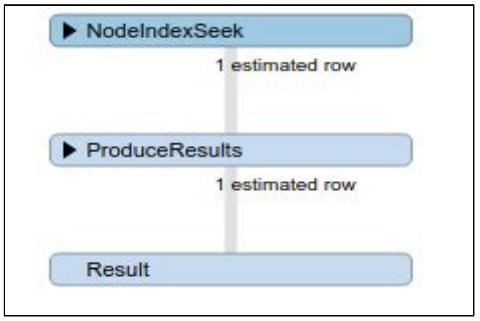
    - Zmierzone czasy wykonania zapytania o aktora w bazie z i bez indexu

```
@timing
def find_name(name):
     with driver.session() as session:
        return session.run("MATCH (p: Person) "
                           "WHERE p.name = $name "
                           "RETURN p" , name=name)

def create_index():
    with driver.session() as session:
        session.run("CREATE INDEX ON:Person(name)")

def drop_index():
    with driver.session() as session:
        session.run("DROP INDEX ON:Person(name)")


create_index()
find_name("Tom Hanks")
drop_index()
find_name("Tom Hanks");
```

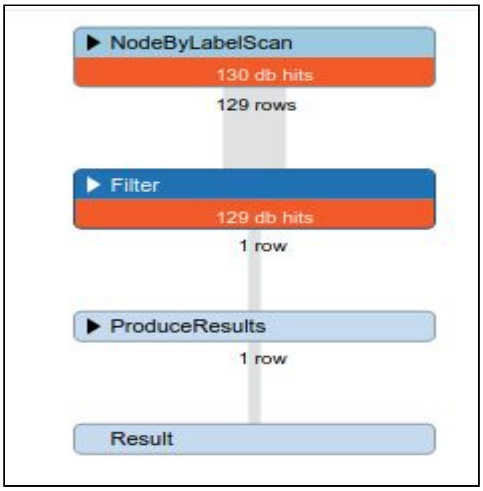
```
func:'find_name' args: ('Tom Hanks',) took: 2.047062 ms
func:'find_name' args: ('Tom Hanks',) took: 17.403364 ms
```

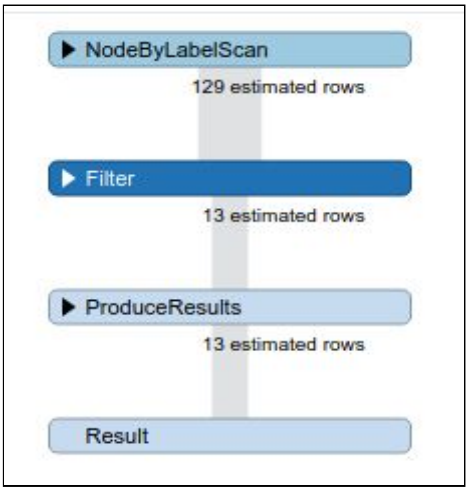
- Profile i explain odpowiednio dla zapytań:
- Z indexem





- Bez indexu



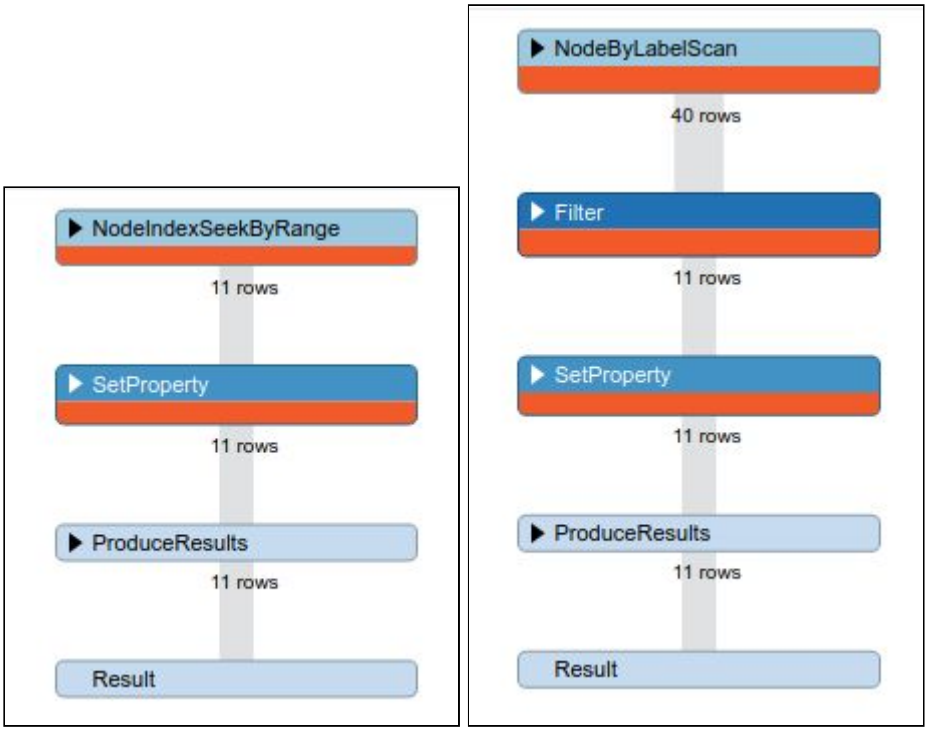


12. Spróbować dokonać optymalizacji wybranych dwóch zapytań z poprzednich zadań (załączyć przykłady w sprawozdaniu).
    a. Pierwsze zapytanie, dotyczące zmiany atrybutu filmów, dla wszystkich tytułów zaczynających się od ciągu znaków

```
@timing
def to_upper_if_starts_with(starts_with):
    with driver.session() as session:
        return session.run("MATCH (m:Movie) "
                           "WHERE m.title STARTS WITH $starts_with "
                           "SET m.tagline = toUpper(m.tagline) "
                           "RETURN m",
                           starts_with=starts_with)

def create_index_movie():
    with driver.session() as session:
        session.run("CREATE INDEX ON:Movie(title)")

def drop_index_movie():
    with driver.session() as session:
        session.run("DROP INDEX ON:Movie(title)")


create_index_movie()
to_upper_if_starts_with("The")
drop_index_movie()
to_upper_if_starts_with("The");
```

```
func:'to_upper_if_starts_with' args: ('The',) took: 2.986908 ms
func:'to_upper_if_starts_with' args: ('The',) took: 9.854078 ms
```

- Optymalizacja w ten sam sposób co w poprzednim zadaniu, wystarczy założyć index na Movie:title. Zobaczmy jeszcze profiler, z indexem i bez niego.

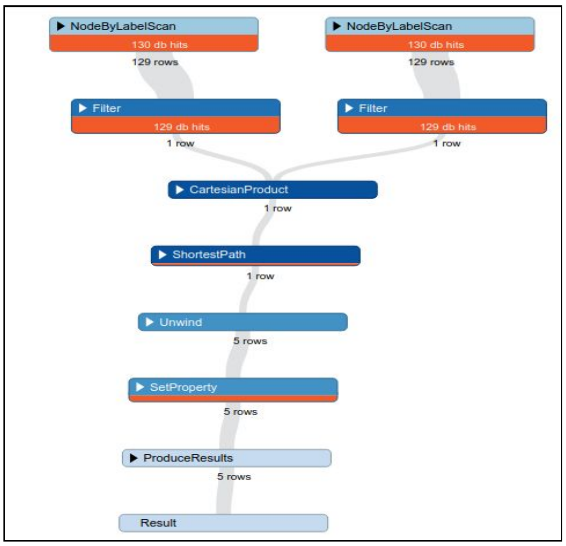


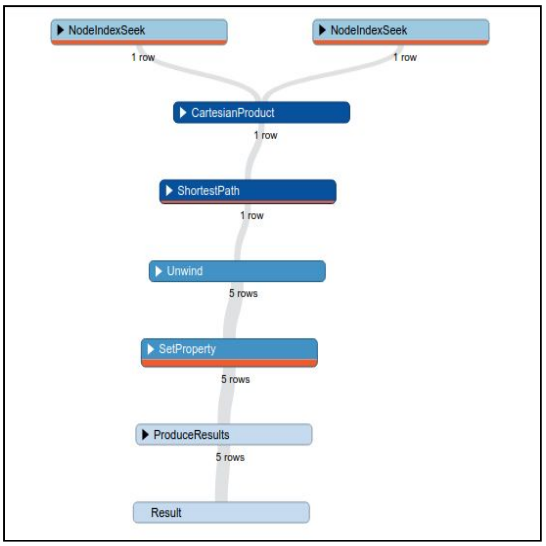
b. Drugie zapytanie, dotyczące aktualizacji atrybutu węzłów na ścieżce między 2 węzłami

```
def update_nodes(name1, name2):
     with driver.session() as session:
        return session.run("MATCH path = shortestPath((p1:Person)-[*]-(p2:Person)) "
                           "WHERE p1.name=$name1 AND p2.name=$name2 "
                           "UNWIND nodes(path) as n "
                           "SET n.keanupath = True "
                           "RETURN n"
                           ,name1=name1, name2=name2).value()
def create_index():
    with driver.session() as session:
        session.run("CREATE INDEX ON:Person(name)")

def drop_index():
    with driver.session() as session:
        session.run("DROP INDEX ON:Person(name)")

create_index()
update_nodes("Keanu Reeves", "Tom Hanks")
drop_index()
update_nodes("Keanu Reeves", "Tom Hanks");
```

```
func:'update_nodes' args: ('Keanu Reeves', 'Tom Hanks') took: 2.200842 ms
func:'update_nodes' args: ('Keanu Reeves', 'Tom Hanks') took: 13.778687 ms
```

- Optymalizacja w ten sam sposób co w poprzednim zadaniu, wystarczy założyć index na Person:name. Zobaczmy jeszcze profiler, z indexem i bez niego.





13. Napisać kod, które wygeneruje drzewo rozpinające z bazy (z poziomu javy lub pythona, nie musi być minimalne) - (można wygenerować własny mały graf do realizacji zadania, zadanie na liczbę punktów powyżej 5.0)

    - Wizualizacja grafu za pomocą biblioteki networkx

```
!pip install numpy
!pip install networkx
!pip install matplotlib

import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import copy

def draw_graph(graph):
    copy_graph = copy.deepcopy(graph)

    G=nx.Graph()
    for node in copy_graph.keys():
        G.add_node(node)

    for node in copy_graph.keys():
        for neighbour in list(copy_graph[node]):
            G.add_edge(node, neighbour)
            copy_graph[neighbour].remove(node)

    pos = nx.shell_layout(G)
    nx.draw(G, pos)
    plt.show()
```

- Drzewo rozpinające generujemy za pomocą algorytmu dfs. Wagi krawędzi są równe, zatem nie ma potrzeby stosować alg. Kruskala/Prima.

```
def dfs(node_id, graph, visited, final_graph):
    visited[node_id] = 1
    for neighbour_id in list(graph[node_id]):
        if(visited[neighbour_id]==0):
            final_graph[node_id].add(neighbour_id)
            final_graph[neighbour_id].add(node_id)
            dfs(neighbour_id, graph, visited, final_graph)

def get_spanning_tree(graph):
    visited = {node_id:0 for node_id in graph.keys()}
    final_graph = {node_id:set() for node_id in graph.keys()}
    for node_id in graph.keys():
        if visited[node_id] == 0:
            dfs(node_id, graph, visited, final_graph)
    return final_graph
```

- Z bazy danych pobieramy wszystkie wierzchołki oraz krawędzie je łączące(u nas max 1 krawędź, w bazie może być multi-graf, ale nie zmienia to mst). W naszym grafie wierzchołki będą odpowiadały ID wierzchołką w grafie, zatem gdyby była potrzeba łatwo można zmapować ID na atrybuty węzła z bazy, analogicznie krawędzie na relacje z bazy.

```
# Fetch database

def get_all_nodes_id():
    with driver.session() as session:
        return session.run("MATCH (n) RETURN id(n)").value()

def get_neighbour(node_id):
    with driver.session() as session:
        return session.run("MATCH (n1)-[*1]-(n2) "
                           "WHERE id(n1)=$node_id "
                           "RETURN id(n2)", node_id=node_id).value()

def number_of_edges(graph):
    return sum([len(list(graph[node_id])) for node_id in graph.keys()])

# fetch graph from database
graph = {node_id:set(get_neighbour(node_id)) for node_id in get_all_nodes_id()}
```

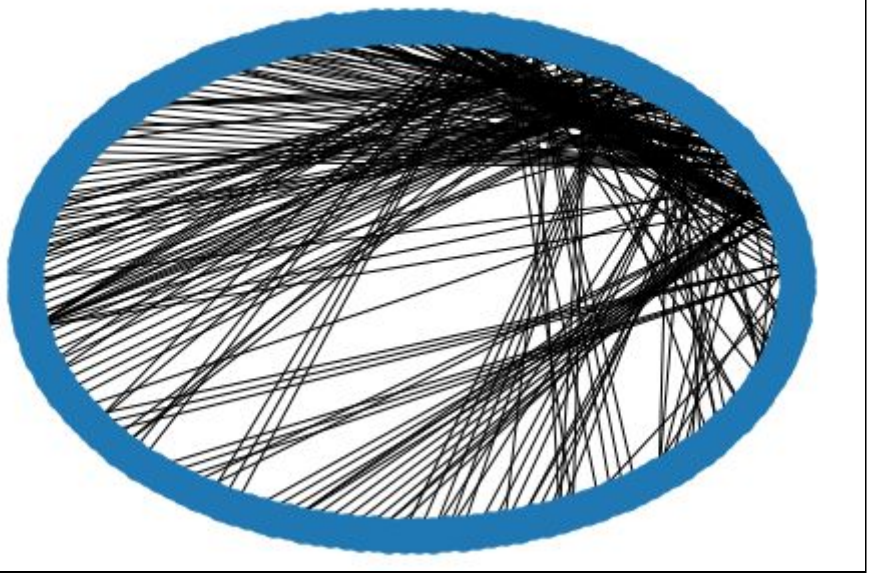
- Poniżej testy dla grafu z bazy danych, liczba krawędzi oraz wygenerowane MST. Następnie przeprowadzimy test dla grafu pełnego na mniejszej ilości wierzchołków, żeby graf był bardziej widoczny.

```
print(number_of_edges(graph))
print(number_of_edges(get_spanning_tree(graph)))
```

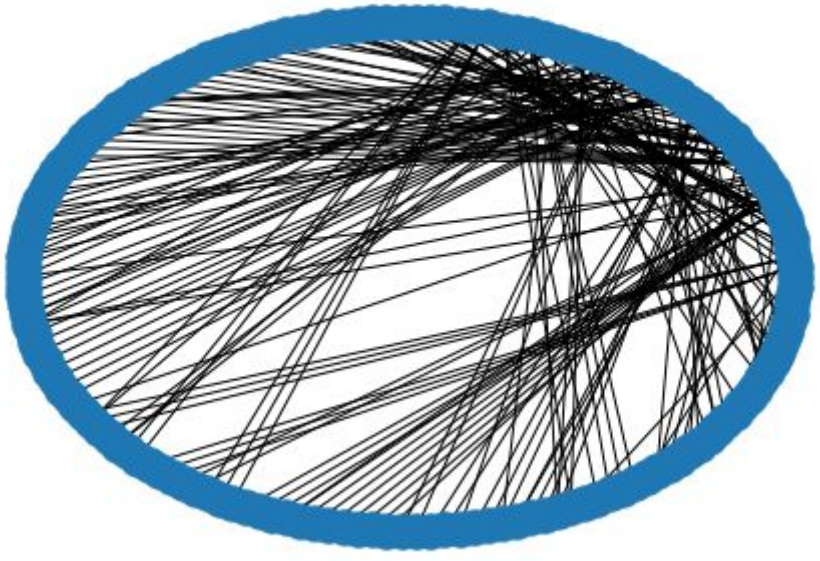
```
215
164
```

- Jak widać graf z bazy danych jest dość rzadki, więc nie będzie zbyt dużej różnicy na rysunkach, do tego jest dużo wierzchołków.

```
draw_graph(graph)
```



```
draw_graph(get_spanning_tree(graph))
```



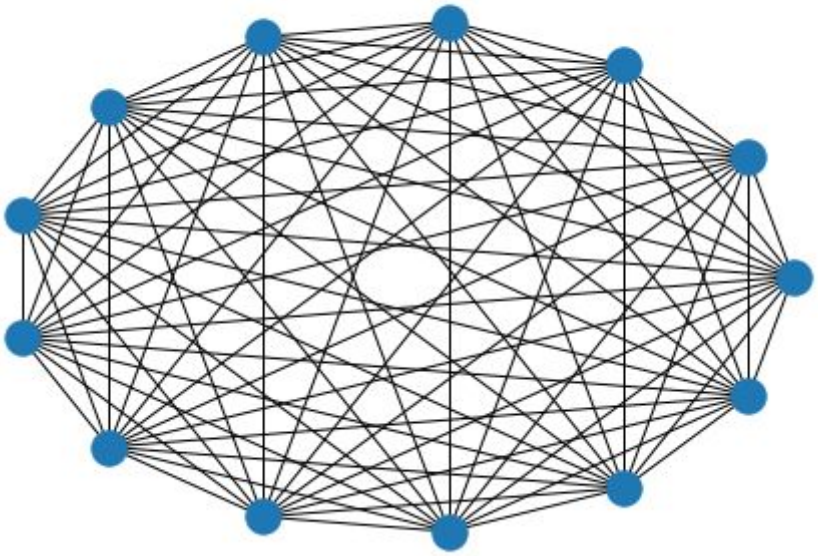
- Jak widać MST zmniejszyło nieznacznie liczbę krawędzi, ale nie jest to zbyt widoczne ze względu na małą ilość cykli, przeprowadzimy jeszcze raz testy dla mniejszej ilości wierzchołków i innego grafu

- Tym razem weźmiemy graf pełny na 13 wierzchołkach

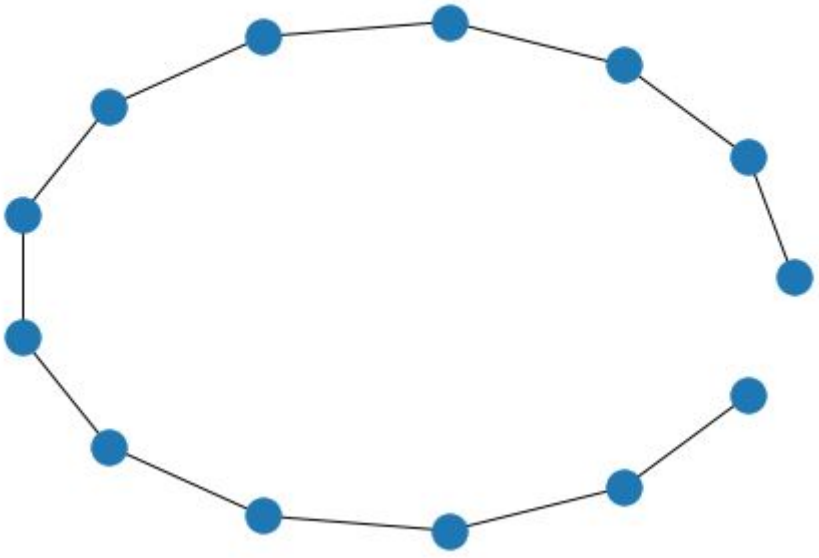
```
graph = {i:{x for x in range(13) if x != i} for i in range(13)}
print(number_of_edges(graph))
print(number_of_edges(get_spanning_tree(graph)))
```

```
78
12
```

```
draw_graph(graph)
```



```
draw_graph(get_spanning_tree(graph))
```



- W tym wypadku MST jest ścieżką, zatem wszystko się zgadza.