

Raport z wykonania ćwiczenia Neo4j

Jakub Płotnikowski

Styczeń 2020r.

Spis treści

1	Zainstalować serwer neo4j lokalnie	2
2	Wgrać bazę	3
3	Zaimplementować funkcję (wystarczy wykonać jedno zapytanie typu MATCH WHERE i wyświetlić wynik)	4
4	Stworzyć kilka nowych węzły reprezentujących film oraz aktorów w nim występujących, następnie stworzyć relacje ich łączące (np. ACTED_IN)	5
5	Dodać zapytaniem nowe właściwości nowo dodanych węzłów reprezentujących aktor (np. birthdate oraz birthplace).	6
6	Ułożyć zapytanie, które zmieni wartość atrybutu węzłów danego typu, jeżeli innych atrybut węzła spełnia zadane kryterium	7
7	Zapytanie o aktorów którzy grali w conajmniej 2 filmach (użyć collect i length) i policzyć średnią występów w filmach dla grupy aktorów, którzy wystąpili w conajmniej 3 filmach.	8
8		10
9	Zmienić wartość wybranego atrybutu w węzłach na ścieżce pomiędzy dwoma podanymi węzłami	10
10	Wyświetlić węzły, które znajdują się na 2 miejscu na ścieżkach o długości 4 pomiędzy dwoma wybranymi węzłami.	11
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).	12
11.1	Zapytanie bez indeksu	12
11.2	Zapytanie z indeksem	12
12	Spróbować dokonać optymalizacji wybranych dwóch zapytań z poprzednich zadań (załączyć przykłady w sprawozdaniu).	13
13	Cały kod źródłowy programu	14

1 Zainstalować serwer neo4j lokalnie

Zainstalowałem Neo4j Desktop Community: <https://neo4j.com/download-center/#community>

Uruchamianie serwera Neo4j:

```
1 # C:\Neo4j\neo4j-community-3.5.14-windows\neo4j-community-3.5.14\bin> .\neo4j.bat start
```

Serwer dostępny był pod adresem: <http://localhost:7474/browser/>

2 Wgrać bazę

Zadanie wykonałem w Pythonie, do każdego zadania dołączony kod źródłowy oraz rezultat działania

```
1 def import_database(tx):
2     print("Movies are being imported...")
3
4     tx.run(
5         """
6         LOAD CSV WITH HEADERS FROM
7         'https://neo4j.com/docs/cypher-manual/3.5/csv/query-tuning/movies.csv' AS line
8         MERGE (m:Movie { title: line.title })
9         ON CREATE SET m.released = toInteger(line.released), m.tagline = line.tagline
10        """
11
12    print("Movies have been successfully imported")
13
14    print("Actors are being imported...")
15
16    tx.run(
17        """
18        LOAD CSV WITH HEADERS FROM
19        'https://neo4j.com/docs/cypher-manual/3.5/csv/query-tuning/actors.csv' AS line
20        MATCH (m:Movie { title: line.title })
21        MERGE (p:Person { name: line.name })
22        ON CREATE SET p.born = toInteger(line.born)
23        MERGE (p)-[:ACTED_IN { roles:split(line.roles, ';')}]>(m)
24        """
25
26    print("Actors have been successfully imported")
27
28    print("Directors are being imported...")
29
30    tx.run(
31        """
32        LOAD CSV WITH HEADERS FROM
33        'https://neo4j.com/docs/cypher-manual/3.5/csv/query-tuning/directors.csv' AS line
34        MATCH (m:Movie { title: line.title })
35        MERGE (p:Person { name: line.name })
36        ON CREATE SET p.born = toInteger(line.born)
37        MERGE (p)-[:DIRECTED]>(m)
38        """
39
40    print("Directors have been successfully imported")
41
42 # Result
43
44 # Movies are being imported...
45 # Movies have been successfully imported
46 # Actors are being imported...
47 # Actors have been successfully imported
48 # Directors are being imported...
49 # Directors have been successfully imported
```

```
Movies are being imported...
Movies have been successfully imported
Actors are being imported...
Actors have been successfully imported
Directors are being imported...
Directors have been successfully imported
```

3 Zaimplementować funkcję (wystarczy wykonać jedno zapytanie typu MATCH WHERE i wyświetlić wynik)

```
1 def execute_task3(tx):
2     title = "Cloud Atlas"
3     print(f"\nDirectors of \'{title}\':")
4
5     for record in tx.run("MATCH (person)-[:DIRECTED]-(movie) WHERE movie.title = $title "
6                           "RETURN person.name", title=title, ):
7         print(f"- {record['person.name']}")
8
9 # Result
10
11 # Directors of "Cloud Atlas":
12 # - Tom Tykwer
13 # - Andy Wachowski
14 # - Lana Wachowski
```

```
Directors of "Cloud Atlas":
- Tom Tykwer
- Andy Wachowski
- Lana Wachowski
```

4 Stworzyć kilka nowych węzły reprezentujących film oraz aktorów w nim występujących, następnie stworzyć relacje ich łączące (np. ACTED_IN)

```
1 def execute_task4(tx):
2     title = "The Lord of the Rings: The Return of the King"
3     tagline = "One Ring To Rule Them All."
4     released = 2003
5
6     tx.run("MERGE (:Movie {title: $title, tagline: $tagline, released: $released})",
7           title=title, tagline=tagline, released=released)
8
9     Person = namedtuple("Person", "name born")
10    people = [
11        Person("Viggo Mortensen", 1958),
12        Person("Elijah Wood", 1981),
13        Person("Ian McKellen", 1939),
14        Person("Liv Tyler", 1977),
15        Person("Cate Blanchett", 1969),
16    ]
17
18    for person in people:
19        tx.run("MERGE (:Person {name: $name, born: $born})",
20              name=person.name, born=person.born)
21
22        tx.run("MATCH (person:Person {name: $name}), (movie:Movie {title: $title}) "
23              "MERGE (person)-[:ACTED_IN]->(movie)",
24              name=person.name, title=title)
25
26    print(f"\nActors playing in \"{title}\":")
27    for record in tx.run("MATCH (person:Person)-[:ACTED_IN]->(:Movie {title: $title}) "
28                      "RETURN person", title=title):
29        print(f"- {record['person']['name']}")
30
31 # Result
32
33 # Actors playing in "The Lord of the Rings: The Return of the King":
34 # - Elijah Wood
35 # - Cate Blanchett
36 # - Sean Astin
37 # - Orlando Bloom
38 # - John Rhys-Davies
39 # - Ian McKellen
40 # - Hugo Weaving
41 # - Miranda Otto
42 # - Viggo Mortensen
43 # - Liv Tyler
```

```
Actors playing in "The Lord of the Rings: The Return of the King":
- Elijah Wood
- Cate Blanchett
- Sean Astin
- Orlando Bloom
- John Rhys-Davies
- Ian McKellen
- Hugo Weaving
- Miranda Otto
- Viggo Mortensen
- Liv Tyler
```

5 Dodać zapytaniem nowe właściwości nowo dodanych węzłów reprezentujących aktor (np. birthdate oraz birthplace).

```
1 def execute_task5(tx):
2     Person = namedtuple("Person", "name birthdate birthplace")
3     people = [
4         Person("Viggo Mortensen", "October 20, 1958", "Manhattan, New York"),
5         Person("Elijah Wood", "January 28, 1981", "Cedar Rapids, Iowa"),
6         Person("Ian McKellen", "May 25, 1939", "Burnley, United Kingdom"),
7         Person("Liv Tyler", "July 1, 1977", "New York, New York"),
8         Person("Cate Blanchett", "May 14, 1969", "Ivanhoe, Australia"),
9     ]
10
11     for person in people:
12         tx.run("MATCH (person:Person {name: $name}) "
13              "SET person.birthdate = $birthdate, person.birthplace = $birthplace",
14              name=person.name,
15              birthdate=person.birthdate, birthplace=person.birthplace)
16
17     print("\nBirthplaces and birthdates of actors playing in \"The Lord of The Ring: The Return
18         of the King\":")
19     for person in people:
20         record = tx.run("MATCH (person:Person {name: $name}) "
21              "RETURN person",
22              name=person.name).single()
23         print(f"- {record['person']['name']}: {record['person']['birthplace']}, {record['person']
24             ['birthdate']}")
25
26 # Result
27
28 # Birthplaces and birthdates of actors playing in "The Lord of The Ring: The Return of the King
29 # ":
30 # - Viggo Mortensen: Manhattan, New York, October 20, 1958
31 # - Elijah Wood: Cedar Rapids, Iowa, January 28, 1981
32 # - Ian McKellen: Burnley, United Kingdom, May 25, 1939
33 # - Liv Tyler: New York, New York, July 1, 1977
34 # - Cate Blanchett: Ivanhoe, Australia, May 14, 1969
35
36 Birthplaces and birthdates of actors playing in "The Lord of The Ring: The Return of the King":
37 - Viggo Mortensen: Manhattan, New York, October 20, 1958
38 - Elijah Wood: Cedar Rapids, Iowa, January 28, 1981
39 - Ian McKellen: Burnley, United Kingdom, May 25, 1939
40 - Liv Tyler: New York, New York, July 1, 1977
41 - Cate Blanchett: Ivanhoe, Australia, May 14, 1969
```

6 Ułożyć zapytanie, które zmieni wartość atrybutu węzłów danego typu, jeżeli innych atrybut węzła spełnia zadane kryterium

```
1 def execute_task6(tx):
2     print("\nChanging year of birth from 1972 to 1971 for following actors:")
3     for record in tx.run("MATCH (person:Person) "
4                           "WHERE person.born = 1972 "
5                           "SET person.born = 1971 "
6                           "RETURN person.name"):
7         print(f"- {record['person.name']}")
8
9     print("\nChanging year of birth from 1971 to 1972 for following actors:")
10    for record in tx.run("MATCH (person:Person) "
11                          "WHERE person.born = 1971 "
12                          "SET person.born = 1972 "
13                          "RETURN person.name"):
14        print(f"- {record['person.name']}")
15
16 # Result
17
18 # Changing year of birth from 1972 to 1971 for following actors:
19 # - Noah Wyle
20 # - Regina King
21 # - Corey Feldman
22 # - Wil Wheaton
23 # - Rick Yune
24 # - Paul Bettany
25 # - Sean Astin
26 #
27 # Changing year of birth from 1971 to 1972 for following actors:
28 # - Noah Wyle
29 # - Regina King
30 # - Corey Feldman
31 # - Wil Wheaton
32 # - Rick Yune
33 # - Paul Bettany
34 # - Sean Astin
```

```
Changing year of birth from 1972 to 1971 for following actors:
- Noah Wyle
- Regina King
- Corey Feldman
- Wil Wheaton
- Rick Yune
- Paul Bettany
- Sean Astin
```

```
Changing year of birth from 1971 to 1972 for following actors:
- Noah Wyle
- Regina King
- Corey Feldman
- Wil Wheaton
- Rick Yune
- Paul Bettany
- Sean Astin
```

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.

```
1 def execute_task7(tx):
2     print("\nThese actors have played in at least two movies:")
3     for record in tx.run("MATCH (person:Person)-[:ACTED_IN]->(movie:Movie) "
4                           "WITH person, size(collect(movie)) as movies_played_in "
5                           "WHERE movies_played_in >= 2 "
6                           "RETURN person.name"):
7         print(f"- {record['person.name']}")
8
9     print("\nAverage number of movies that actors have played in "
10          "(for actors that have played in at least three movies)",
11          end=" ")
12     record = tx.run("MATCH (person:Person)-[:ACTED_IN]->(movie:Movie) "
13                     "WITH person, size(collect(movie)) as movies "
14                     "WHERE movies >= 3 "
15                     "RETURN avg(movies) as average_number_of_movies").single()
16     print(record["average_number_of_movies"])
17
18 # Result
19
20 # These actors have played in at least two movies:
21 # - Jack Nicholson
22 # - Keanu Reeves
23 # - Gene Hackman
24 # - Charlize Theron
25 # - Hugo Weaving
26 # - Laurence Fishburne
27 # - Carrie-Anne Moss
28 # - Ben Miles
29 # - Tom Hanks
30 # - Rain
31 # - Rick Yune
32 # - Liv Tyler
33 # - Bonnie Hunt
34 # - Jerry O'Connell
35 # - Cuba Gooding Jr.
36 # - Tom Cruise
37 # - Meg Ryan
38 # - Kiefer Sutherland
39 # - Kevin Bacon
40 # - J.T. Walsh
41 # - Danny DeVito
42 # - Helen Hunt
43 # - Greg Kinnear
44 # - Bill Paxton
45 # - Gary Sinise
46 # - Oliver Platt
47 # - Sam Rockwell
48 # - Marshall Bell
49 # - Max von Sydow
50 # - Robin Williams
51 # - Nathan Lane
52 # - Rosie O'Donnell
53 # - Steve Zahn
54 # - James Cromwell
55 # - Zach Grenier
56 # - Philip Seymour Hoffman
57 # - Ian McKellen
58
59 # Average number of movies that actors have played in (for actors that have played in at least
    three movies) 4.4
```


These actors have played in at least two movies:

- Jack Nicholson
- Keanu Reeves
- Gene Hackman
- Charlize Theron
- Hugo Weaving
- Laurence Fishburne
- Carrie-Anne Moss
- Ben Miles
- Tom Hanks
- Rain
- Rick Yune
- Liv Tyler
- Bonnie Hunt
- Jerry O'Connell
- Cuba Gooding Jr.
- Tom Cruise
- Meg Ryan
- Kiefer Sutherland
- Kevin Bacon
- J.T. Walsh
- Danny DeVito
- Helen Hunt
- Greg Kinnear
- Bill Paxton
- Gary Sinise
- Oliver Platt
- Sam Rockwell
- Marshall Bell
- Max von Sydow
- Robin Williams
- Nathan Lane
- Rosie O'Donnell
- Steve Zahn
- James Cromwell
- Zach Grenier
- Philip Seymour Hoffman
- Ian McKellen

Average number of movies that actors have played in (for actors that have played in at least three movies) 4.4

9 Zmienić wartość wybranego atrybutu w węzłachna ścieżce pomiędzy dwoma podanymi węzłami

```

1 def execute_task9(tx):
2     actor1 = "Val Kilmer"
3     actor2 = "Al Pacino"
4
5     nodes = []
6
7     print(f"\nActors on the shortest path between \"{actor1}\" and \"{actor2}\":")
8     for record in tx.run("MATCH (a:Person {name: $actor1}), (b:Person {name: $actor2}), "
9                           "path = shortestPath((a)-[*..25]-(b)) "
10                          "RETURN path", actor1=actor1, actor2=actor2):
11         for node in record["path"].nodes:
12             if "Person" in node.labels:
13                 nodes.append(node.id)
14                 print(f"- {node['name']}")
15
16     print(f"\nAdded is_special parameter for people on the shortest path between \"{actor1}\"
17 and \"{actor2}\":")
18     for record in tx.run("MATCH (person:Person) "
19                           "WHERE id(person) in $nodes "
20                           "SET person.is_special = true "
21                          "RETURN person", nodes=nodes):
22         print(f"- name: {record['person']['name']}, is_special: {record['person']['is_special']}")
23
24 # Result
25
26 # Actors on the shortest path between "Val Kilmer" and "Al Pacino":
27 # - Val Kilmer
28 # - Meg Ryan
29 # - Tom Hanks
30 # - Charlize Theron
31 # - Al Pacino
32
33 # Added is_special parameter for people on the shortest path between "Val Kilmer" and "Al
34 Pacino":
35 # - name: Charlize Theron, is_special: True
36 # - name: Al Pacino, is_special: True
37 # - name: Val Kilmer, is_special: True
38 # - name: Meg Ryan, is_special: True
39 # - name: Tom Hanks, is_special: True

```

Actors on the shortest path between "Val Kilmer" and "Al Pacino":

- Val Kilmer
- Meg Ryan
- Tom Hanks
- Charlize Theron
- Al Pacino

Added is_special parameter for people on the shortest path between "Val Kilmer" and "Al Pacino":

- name: Charlize Theron, is_special: True
- name: Al Pacino, is_special: True
- name: Val Kilmer, is_special: True
- name: Meg Ryan, is_special: True
- name: Tom Hanks, is_special: True

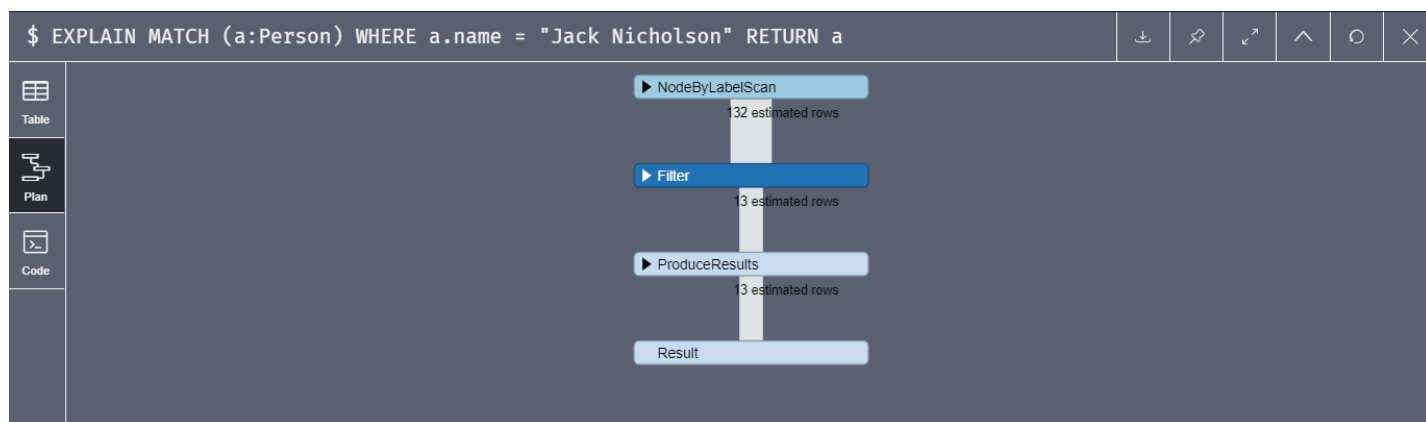
10 Wyświetlić węzły, które znajdują się na 2 miejscu na ścieżkach o długości 4 pomiędzy dwoma wybranymi węzłami.

```
1 def execute_task10(tx):
2     print(f"\nNodes on the second place on path of size 4 between actor: \"Jack Nicholson\" and
3         movie: \"Hoffa\":")
4
5     for record in tx.run(
6         """
7         MATCH p = (:Person {name: \"Jack Nicholson\"})-[*3]-(:Movie {title: \"Hoffa\"})
8         RETURN nodes(p)[1] as node
9         """):
10         print(f"- {record['node']}")
11
12 # Result
13 #
14 # Nodes on the second place on path of size 4 between actor: "Jack Nicholson" and movie: "Hoffa
15 #   ";
16 # - <Node id=14 labels={'Movie'} properties={'title': 'A Few Good Men', 'tagline': "In the
17 #   heart of the nation's capital, in a courthouse of the U.S. government, one man will stop at
18 #   nothing to keep his honor, and one will stop at nothing to find the truth.", 'released':
19 #   1992}>
20 # - <Node id=15 labels={'Movie'} properties={'title': "One Flew Over the Cuckoo's Nest", '
21 #   tagline': "If he's crazy, what does that make you?", 'released': 1975}>
22 # - <Node id=16 labels={'Movie'} properties={'title': 'Hoffa', 'tagline': "He didn't want law.
23 #   He wanted justice.", 'released': 1992}>
24 # - <Node id=16 labels={'Movie'} properties={'title': 'Hoffa', 'tagline': "He didn't want law.
25 #   He wanted justice.", 'released': 1992}>
26 # - <Node id=15 labels={'Movie'} properties={'title': "One Flew Over the Cuckoo's Nest", '
27 #   tagline': "If he's crazy, what does that make you?", 'released': 1975}>
28
29 Nodes on the second place on path of size 4 between actor: "Jack Nicholson" and movie: "Hoffa":
30 - <Node id=14 labels={'Movie'} properties={'title': 'A Few Good Men', 'tagline': "In the heart of the nation's capital, in a courthouse of the U.S. government, one man will stop at nothing to keep his honor, and one will stop at nothing to find the truth.", 'released': 1992}>
31 - <Node id=15 labels={'Movie'} properties={'title': "One Flew Over the Cuckoo's Nest", 'tagline': "If he's crazy, what does that make you?", 'released': 1975}>
32 - <Node id=16 labels={'Movie'} properties={'title': 'Hoffa', 'tagline': "He didn't want law. He wanted justice.", 'released': 1992}>
33 - <Node id=16 labels={'Movie'} properties={'title': 'Hoffa', 'tagline': "He didn't want law. He wanted justice.", 'released': 1992}>
34 - <Node id=15 labels={'Movie'} properties={'title': "One Flew Over the Cuckoo's Nest", 'tagline': "If he's crazy, what does that make you?", 'released': 1975}>
```

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).

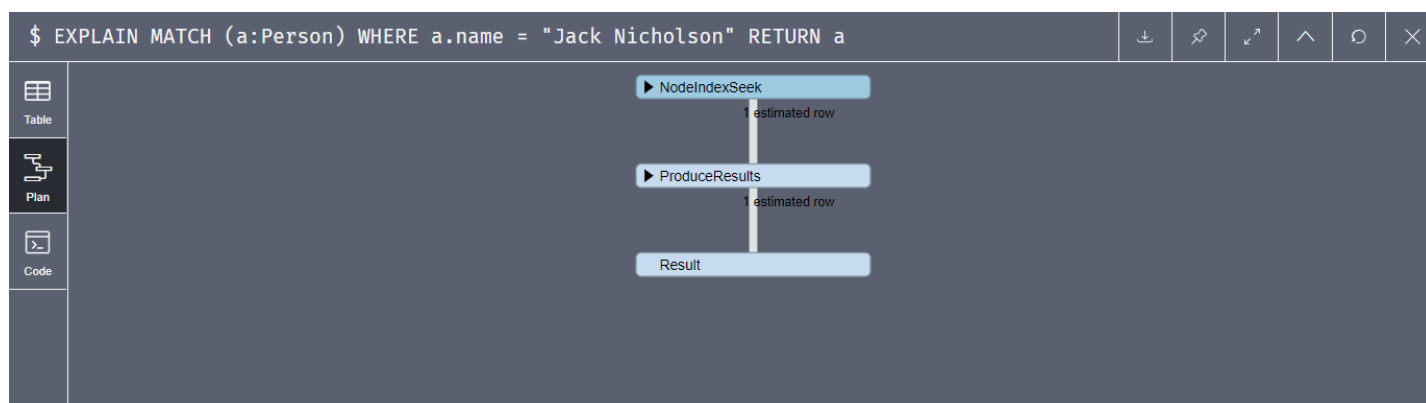
Poniższe 2 screeny przedstawiają wykonanie zapytania EXPLAIN - o aktora Jacka Nicholsona.

11.1 Zapytanie bez indeksu



Completed after 6 ms.

11.2 Zapytanie z indeksem



Completed after 1 ms.

12 Spróbować dokonać optymalizacji wybranych dwóch zapytańz poprzednich zadań(załączyć przykłady w sprawozdaniu).

```
1 # 1. Dla zapytania z zadania czwartego:
2 #
3 # "MATCH (person:Person)-[:ACTED_IN]->(:Movie {title: $title}) "
4 # "RETURN person"
5 #
6 # mozna wykonac optymalizacje w postaci zwracania tylko potrzebnej wartosci z wezla:
7 # "MATCH (person:Person)-[:ACTED_IN]->(:Movie {title: $title}) "
8 # "RETURN person.name"
9 #
10 # 2. Dla zapytania z zadania trzeciego:
11 # "MATCH (person)-[:DIRECTED]-(movie) WHERE movie.title = $title "
12 # "RETURN person.name"
13 #
14 # mozna wykonac optymalizacje w postaci okreslenia typu wezla (etykiety):
15 # "MATCH (person:Person)-[:DIRECTED]-(movie) WHERE movie.title = $title "
16 # "RETURN person.name"
```

13 Cały kod źródłowy programu

```
1 from collections import namedtuple
2
3 from neo4j import GraphDatabase
4
5
6 def import_database(tx):
7     print("Movies are being imported...")
8
9     tx.run(
10         """
11         LOAD CSV WITH HEADERS FROM
12         'https://neo4j.com/docs/cypher-manual/3.5/csv/query-tuning/movies.csv' AS line
13         MERGE (m:Movie { title: line.title })
14         ON CREATE SET m.released = toInteger(line.released), m.tagline = line.tagline
15         """
16
17     print("Movies have been successfully imported")
18
19     print("Actors are being imported...")
20
21     tx.run(
22         """
23         LOAD CSV WITH HEADERS FROM
24         'https://neo4j.com/docs/cypher-manual/3.5/csv/query-tuning/actors.csv' AS line
25         MATCH (m:Movie { title: line.title })
26         MERGE (p:Person { name: line.name })
27         ON CREATE SET p.born = toInteger(line.born)
28         MERGE (p)-[:ACTED_IN { roles:split(line.roles, ';')}]>(m)
29         """
30
31     print("Actors have been successfully imported")
32
33     print("Directors are being imported...")
34
35     tx.run(
36         """
37         LOAD CSV WITH HEADERS FROM
38         'https://neo4j.com/docs/cypher-manual/3.5/csv/query-tuning/directors.csv' AS line
39         MATCH (m:Movie { title: line.title })
40         MERGE (p:Person { name: line.name })
41         ON CREATE SET p.born = toInteger(line.born)
42         MERGE (p)-[:DIRECTED]>(m)
43         """
44
45     print("Directors have been successfully imported")
46
47
48 def execute_task3(tx):
49     title = "Cloud Atlas"
50     print(f"\nDirectors of \"{title}\":")
51
52     for record in tx.run("MATCH (person)-[:DIRECTED]-(movie) WHERE movie.title = $title "
53                          "RETURN person.name", title=title, ):
54         print(f"- {record['person.name']}")
55
56
57 def execute_task4(tx):
58     title = "The Lord of the Rings: The Return of the King"
59     tagline = "One Ring To Rule Them All."
60     released = 2003
61
62     tx.run("MERGE (:Movie {title: $title, tagline: $tagline, released: $released})",
63           title=title, tagline=tagline, released=released)
64
65     Person = namedtuple("Person", "name born")
66     people = [
67         Person("Viggo Mortensen", 1958),
```

```

68     Person("Elijah Wood", 1981),
69     Person("Ian McKellen", 1939),
70     Person("Liv Tyler", 1977),
71     Person("Cate Blanchett", 1969),
72 ]
73
74 for person in people:
75     tx.run("MERGE (:Person {name: $name, born: $born})",
76           name=person.name, born=person.born)
77
78     tx.run("MATCH (person:Person {name: $name}), (movie:Movie {title: $title}) "
79           "MERGE (person)-[:ACTED_IN]->(movie)",
80           name=person.name, title=title)
81
82 print(f"\nActors playing in \"{title}\":")
83 for record in tx.run("MATCH (person:Person)-[:ACTED_IN]->(movie {title: $title}) "
84                     "RETURN person", title=title):
85     print(f"- {record['person']['name']}")
86
87
88 def execute_task5(tx):
89     Person = namedtuple("Person", "name birthdate birthplace")
90     people = [
91         Person("Viggo Mortensen", "October 20, 1958", "Manhattan, New York"),
92         Person("Elijah Wood", "January 28, 1981", "Cedar Rapids, Iowa"),
93         Person("Ian McKellen", "May 25, 1939", "Burnley, United Kingdom"),
94         Person("Liv Tyler", "July 1, 1977", "New York, New York"),
95         Person("Cate Blanchett", "May 14, 1969", "Ivanhoe, Australia"),
96     ]
97
98     for person in people:
99         tx.run("MATCH (person:Person {name: $name}) "
100              "SET person.birthdate = $birthdate, person.birthplace = $birthplace",
101              name=person.name,
102              birthdate=person.birthdate, birthplace=person.birthplace)
103
104     print("\nBirthplaces and birthdates of actors playing in \"The Lord of The Ring: The Return of the King\":")
105     for person in people:
106         record = tx.run("MATCH (person:Person {name: $name}) "
107                         "RETURN person",
108                         name=person.name).single()
109         print(f"- {record['person']['name']}: {record['person']['birthdate']}, {record['person']['birthplace']}")
110
111
112 def execute_task6(tx):
113     print("\nChanging year of birth from 1972 to 1971 for following actors:")
114     for record in tx.run("MATCH (person:Person) "
115                         "WHERE person.born = 1972 "
116                         "SET person.born = 1971 "
117                         "RETURN person.name"):
118         print(f"- {record['person.name']}")
119
120     print("\nChanging year of birth from 1971 to 1972 for following actors:")
121     for record in tx.run("MATCH (person:Person) "
122                         "WHERE person.born = 1971 "
123                         "SET person.born = 1972 "
124                         "RETURN person.name"):
125         print(f"- {record['person.name']}")
126
127
128 def execute_task7(tx):
129     print("\nThese actors have played in at least two movies:")
130     for record in tx.run("MATCH (person:Person)-[:ACTED_IN]->(movie:Movie) "
131                         "WITH person, size(collect(movie)) as movies_played_in "
132                         "WHERE movies_played_in >= 2 "
133                         "RETURN person.name"):
134         print(f"- {record['person.name']}")

```

```

135
136 print("\nAverage number of movies that actors have played in "
137       "(for actors that have played in at least three movies)",
138       end=" ")
139 record = tx.run("MATCH (person:Person)-[:ACTED_IN]->(movie:Movie) "
140               "WITH person, size(collect(movie)) as movies "
141               "WHERE movies >= 3 "
142               "RETURN avg(movies) as average_number_of_movies").single()
143 print(record["average_number_of_movies"])
144
145
146 def execute_task9(tx):
147     actor1 = "Val Kilmer"
148     actor2 = "Al Pacino"
149
150     nodes = []
151
152     print(f"\nActors on the shortest path between \"{actor1}\" and \"{actor2}\":")
153     for record in tx.run("MATCH (a:Person {name: $actor1}), (b:Person {name: $actor2}), "
154                       "path = shortestPath((a)-[*..25]-(b)) "
155                       "RETURN path", actor1=actor1, actor2=actor2):
156         for node in record["path"].nodes:
157             if "Person" in node.labels:
158                 nodes.append(node.id)
159                 print(f"- {node['name']}")
160
161     print(f"\nAdded is_special parameter for people on the shortest path between \"{actor1}\" "
162           "and \"{actor2}\":")
163     for record in tx.run("MATCH (person:Person) "
164                       "WHERE id(person) in $nodes "
165                       "SET person.is_special = true "
166                       "RETURN person", nodes=nodes):
167         print(f"- name: {record['person']['name']}, is_special: {record['person']['is_special']}")
168
169 def execute_task10(tx):
170     print(f"\nNodes on the second place on path of size 4 between actor: \"Jack Nicholson\" and "
171           "movie: \"Hoffa\":")
172
173     for record in tx.run(
174         """
175         MATCH p = (:Person {name: "Jack Nicholson"})-[*3]-(:Movie {title: "Hoffa"})
176         RETURN nodes(p)[1] as node
177         """):
178         print(f"- {record['node']}")
179
180 if __name__ == "__main__":
181     driver = GraphDatabase.driver("bolt://localhost:7687", auth=("neo4j", "essa"))
182
183     with driver.session() as session:
184         session.write_transaction(import_database)
185         session.read_transaction(execute_task3)
186         session.write_transaction(execute_task4)
187         session.write_transaction(execute_task5)
188         session.write_transaction(execute_task6)
189         session.read_transaction(execute_task7)
190         session.read_transaction(execute_task9)
191         session.read_transaction(execute_task10)
192
193     driver.close()

```