Raport z wykonania ćwiczenia Neo4j

Jakub Płotnikowski

Styczeń 2020r.

Spis treści

1 Zainstalować serwer neo4j lokalnie

Zainstalowałem Neo4j Desktop Community: `https://neo4j.com/download-center/#community`
Uruchamianie serwera Neo4j:

```
# 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

```
def import_database(tx):
    print("Movies are being imported...")

    tx.run(
        """
        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
        """)

    print("Movies have been successfully imported")

    print("Actors are being imported...")

    tx.run(
        """
        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)
        """)

    print("Actors have been successfully imported")

    print("Directors are being imported...")

    tx.run(
        """
        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)
        """)

    print("Directors have been successfully imported")

# Result

# 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
```

```
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)

```python
def execute_task3(tx):
    title = "Cloud Atlas"
    print(f"\nDirectors of \"{title}\":")

    for record in tx.run("MATCH (person)-[:DIRECTED]-(movie) WHERE movie.title = $title "
                         "RETURN person.name", title=title, ):
        print(f"- {record['person.name']}")

# Result

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

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

4 Stworzyć kilka nowych węzły reprezentującychfilm oraz aktoróww nim występujących, następnie stworzyć relacje ich łączące (np. ACTED_IN)

```python
def execute_task4(tx):
    title = "The Lord of the Rings: The Return of the King"
    tagline = "One Ring To Rule Them All."
    released = 2003

    tx.run("MERGE (:Movie {title: $title, tagline: $tagline, released: $released})",
           title=title, tagline=tagline, released=released)

    Person = namedtuple("Person", "name born")
    people = [
        Person("Viggo Mortensen", 1958),
        Person("Elijah Wood", 1981),
        Person("Ian McKellen", 1939),
        Person("Liv Tyler", 1977),
        Person("Cate Blanchett", 1969),
    ]

    for person in people:
        tx.run("MERGE (:Person {name: $name, born: $born})",
               name=person.name, born=person.born)

        tx.run("MATCH (person:Person {name: $name}), (movie:Movie {title: $title}) "
               "MERGE (person)-[:ACTED_IN]->(movie)",
               name=person.name, title=title)

    print(f"\nActors playing in \"{title}\":")
    for record in tx.run("MATCH (person:Person)-[:ACTED_IN]->(:Movie {title: $title}) "
                         "RETURN person", title=title):
        print(f"- {record['person']['name']}")

# Result

# 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
```

```
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 reprezentującychaktor (np. birthdate oraz birthplace).

```
def execute_task5(tx):
    Person = namedtuple("Person", "name birthdate birthplace")
    people = [
        Person("Viggo Mortensen", "October 20, 1958", "Manhattan, New York"),
        Person("Elijah Wood", "January 28, 1981", "Cedar Rapids, Iowa"),
        Person("Ian McKellen", "May 25, 1939", "Burnley, United Kingdom"),
        Person("Liv Tyler", "July 1, 1977", "New York, New York"),
        Person("Cate Blanchett", "May 14, 1969", "Ivanhoe, Australia"),
    ]

    for person in people:
        tx.run("MATCH (person:Person {name: $name}) "
               "SET person.birthdate = $birthdate, person.birthplace = $birthplace",
               name=person.name,
               birthdate=person.birthdate, birthplace=person.birthplace)

    print("\nBirthplaces and birthdates of actors playing in \"The Lord of The Ring: The Return of the King\":")
    for person in people:
        record = tx.run("MATCH (person:Person {name: $name}) "
                        "RETURN person",
                        name=person.name).single()
        print(f"- {record['person']['name']}: {record['person']['birthplace']}, {record['person']['birthdate']}")

# Result

# Birthplaces and birthdates of actors playing in "The Lord of The Ring: The Return of the King":
# - Viggo Mortensen: Manhattan, New York, October 20, 1958
# - Elijah Wood: Cedar Rapids, Iowa, January 28, 1981
# - Ian McKellen: Burnley, United Kingdom, May 25, 1939
# - Liv Tyler: New York, New York, July 1, 1977
# - Cate Blanchett: Ivanhoe, Australia, May 14, 1969
```

```
Birthplaces and birthdates of actors playing in "The Lord of The Ring: The Return of the King":
- Viggo Mortensen: Manhattan, New York, October 20, 1958
- Elijah Wood: Cedar Rapids, Iowa, January 28, 1981
- Ian McKellen: Burnley, United Kingdom, May 25, 1939
- Liv Tyler: New York, New York, July 1, 1977
- Cate Blanchett: Ivanhoe, Australia, May 14, 1969
```

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

```python
def execute_task6(tx):
    print("\nChanging year of birth from 1972 to 1971 for following actors:")
    for record in tx.run("MATCH (person:Person) "
                         "WHERE person.born = 1972 "
                         "SET person.born = 1971 "
                         "RETURN person.name"):
        print(f"- {record['person.name']}")

    print("\nChanging year of birth from 1971 to 1972 for following actors:")
    for record in tx.run("MATCH (person:Person) "
                         "WHERE person.born = 1971 "
                         "SET person.born = 1972 "
                         "RETURN person.name"):
        print(f"- {record['person.name']}")

# Result

# 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
```

```
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.

```python
def execute_task7(tx):
    print("\nThese actors have played in at least two movies:")
    for record in tx.run("MATCH (person:Person)-[:ACTED_IN]->(movie:Movie) "
                         "WITH person, size(collect(movie)) as movies_played_in "
                         "WHERE movies_played_in >= 2 "
                         "RETURN person.name"):
        print(f"- {record['person.name']}")

    print("\nAverage number of movies that actors have played in "
          "(for actors that have played in at least three movies)",
          end=" ")
    record = tx.run("MATCH (person:Person)-[:ACTED_IN]->(movie:Movie) "
                    "WITH person, size(collect(movie)) as movies "
                    "WHERE movies >= 3 "
                    "RETURN avg(movies) as average_number_of_movies").single()
    print(record["average_number_of_movies"])

# Result

# 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
```

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

8

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

```python
def execute_task9(tx):
    actor1 = "Val Kilmer"
    actor2 = "Al Pacino"

    nodes = []

    print(f"\nActors on the shortest path between \"{actor1}\" and \"{actor2}\":")
    for record in tx.run("MATCH (a:Person {name: $actor1}), (b:Person {name: $actor2}), "
                         "path = shortestPath((a)-[*..25]-(b)) "
                         "RETURN path", actor1=actor1, actor2=actor2):
        for node in record["path"].nodes:
            if "Person" in node.labels:
                nodes.append(node.id)
                print(f"- {node['name']}")

    print(f"\nAdded is_special parameter for people on the shortest path between \"{actor1}\" and \"{actor2}\":")
    for record in tx.run("MATCH (person:Person) "
                         "WHERE id(person) in $nodes "
                         "SET person.is_special = true "
                         "RETURN person", nodes=nodes):
        print(f"- name: {record['person']['name']}, is_special: {record['person']['is_special']}")

# Result

# 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
```

```
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.

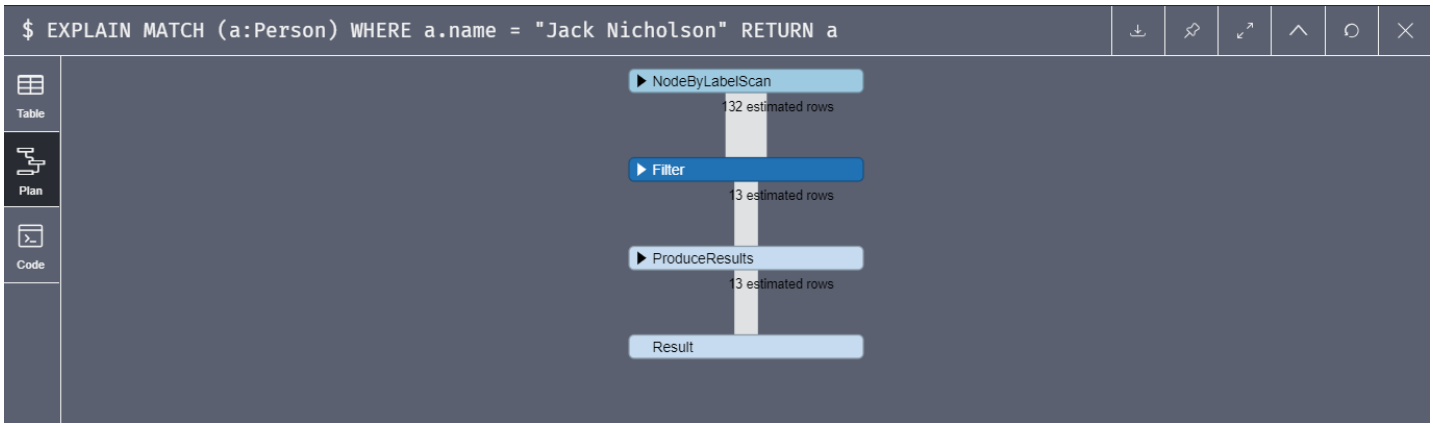
```python
def execute_task10(tx):
    print(f"\nNodes on the second place on path of size 4 between actor: \"Jack Nicholson\" and movie: \"Hoffa\":")

    for record in tx.run(
            """
            MATCH p = (:Person {name: "Jack Nicholson"})-[*3]-(:Movie {title: "Hoffa"})
            RETURN nodes(p)[1] as node
            """):
        print(f"- {record['node']}")

# Result
#
# Nodes on the second place on path of size 4 between actor: "Jack Nicholson" and movie: "Hoffa":
# - <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}>
# - <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}>
# - <Node id=16 labels={'Movie'} properties={'title': 'Hoffa', 'tagline': "He didn't want law. He wanted justice.", 'released': 1992}>
# - <Node id=16 labels={'Movie'} properties={'title': 'Hoffa', 'tagline': "He didn't want law. He wanted justice.", 'released': 1992}>
# - <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}>
```

```
Nodes on the second place on path of size 4 between actor: "Jack Nicholson" and movie: "Hoffa":
- <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}>
- <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}>
- <Node id=16 labels={'Movie'} properties={'title': 'Hoffa', 'tagline': "He didn't want law. He wanted justice.", 'released': 1992}>
- <Node id=16 labels={'Movie'} properties={'title': 'Hoffa', 'tagline': "He didn't want law. He wanted justice.", 'released': 1992}>
- <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).

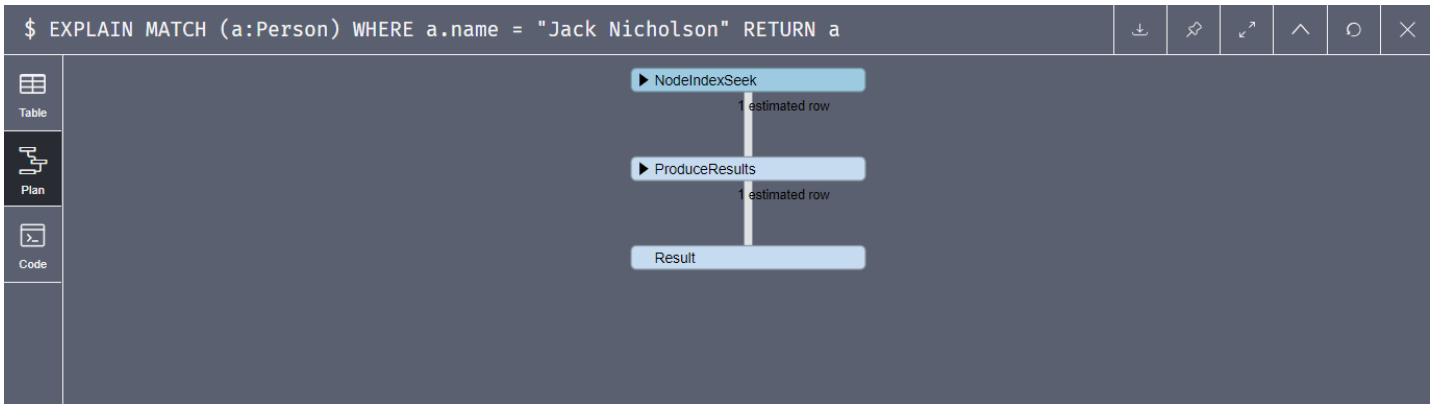
Poniższe 2 screeny przedstawiają wykonanie zapytania EXPLAIN - o aktora Jacka Nicholsona.

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. Dla zapytania z zadania czwartego:
#
# "MATCH (person:Person)-[:ACTED_IN]->(:Movie {title: $title}) "
# "RETURN person"
#
# mozna wykonac optymalizacje w postaci zwracania tylko potrzebnej wartosci z wezla:
# "MATCH (person:Person)-[:ACTED_IN]->(:Movie {title: $title}) "
# "RETURN person.name"
#
# 2. Dla zapytania z zadania trzeciego:
# "MATCH (person)-[:DIRECTED]-(movie) WHERE movie.title = $title "
# "RETURN person.name"
#
# mozna wykonac optymalizacje w postaci okreslenia typu wezla (etykiety):
# "MATCH (person:Person)-[:DIRECTED]-(movie) WHERE movie.title = $title "
# "RETURN person.name"
```

13 Cały kod źródłowy programu

```
from collections import namedtuple

from neo4j import GraphDatabase


def import_database(tx):
    print("Movies are being imported...")

    tx.run(
        """
        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
        """)

    print("Movies have been successfully imported")

    print("Actors are being imported...")

    tx.run(
        """
        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)
        """)

    print("Actors have been successfully imported")

    print("Directors are being imported...")

    tx.run(
        """
        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)
        """)

    print("Directors have been successfully imported")


def execute_task3(tx):
    title = "Cloud Atlas"
    print(f"\nDirectors of \"{title}\":")

    for record in tx.run("MATCH (person)-[:DIRECTED]-(movie) WHERE movie.title = $title "
                         "RETURN person.name", title=title, ):
        print(f"- {record['person.name']}")


def execute_task4(tx):
    title = "The Lord of the Rings: The Return of the King"
    tagline = "One Ring To Rule Them All."
    released = 2003

    tx.run("MERGE (:Movie {title: $title, tagline: $tagline, released: $released})",
           title=title, tagline=tagline, released=released)

    Person = namedtuple("Person", "name born")
    people = [
        Person("Viggo Mortensen", 1958),
```

```
        Person("Elijah Wood", 1981),
        Person("Ian McKellen", 1939),
        Person("Liv Tyler", 1977),
        Person("Cate Blanchett", 1969),
    ]

    for person in people:
        tx.run("MERGE (:Person {name: $name, born: $born})",
               name=person.name, born=person.born)

        tx.run("MATCH (person:Person {name: $name}), (movie:Movie {title: $title}) "
               "MERGE (person)-[:ACTED_IN]->(movie)",
               name=person.name, title=title)

    print(f"\nActors playing in \"{title}\":")
    for record in tx.run("MATCH (person:Person)-[:ACTED_IN]->(:Movie {title: $title}) "
                         "RETURN person", title=title):
        print(f"- {record['person']['name']}")


def execute_task5(tx):
    Person = namedtuple("Person", "name birthdate birthplace")
    people = [
        Person("Viggo Mortensen", "October 20, 1958", "Manhattan, New York"),
        Person("Elijah Wood", "January 28, 1981", "Cedar Rapids, Iowa"),
        Person("Ian McKellen", "May 25, 1939", "Burnley, United Kingdom"),
        Person("Liv Tyler", "July 1, 1977", "New York, New York"),
        Person("Cate Blanchett", "May 14, 1969", "Ivanhoe, Australia"),
    ]

    for person in people:
        tx.run("MATCH (person:Person {name: $name}) "
               "SET person.birthdate = $birthdate, person.birthplace = $birthplace",
               name=person.name,
               birthdate=person.birthdate, birthplace=person.birthplace)

    print("\nBirthplaces and birthdates of actors playing in \"The Lord of The Ring: The Return
     of the King\":")
    for person in people:
        record = tx.run("MATCH (person:Person {name: $name}) "
                        "RETURN person",
                        name=person.name).single()
        print(f"- {record['person']['name']}: {record['person']['birthplace']}, {record['person
    ']['birthdate']}")


def execute_task6(tx):
    print("\nChanging year of birth from 1972 to 1971 for following actors:")
    for record in tx.run("MATCH (person:Person) "
                         "WHERE person.born = 1972 "
                         "SET person.born = 1971 "
                         "RETURN person.name"):
        print(f"- {record['person.name']}")

    print("\nChanging year of birth from 1971 to 1972 for following actors:")
    for record in tx.run("MATCH (person:Person) "
                         "WHERE person.born = 1971 "
                         "SET person.born = 1972 "
                         "RETURN person.name"):
        print(f"- {record['person.name']}")


def execute_task7(tx):
    print("\nThese actors have played in at least two movies:")
    for record in tx.run("MATCH (person:Person)-[:ACTED_IN]->(movie:Movie) "
                         "WITH person, size(collect(movie)) as movies_played_in "
                         "WHERE movies_played_in >= 2 "
                         "RETURN person.name"):
        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}\"
    and \"{actor2}\":")
162     for record in tx.run("MATCH (person:Person) "
163                          "WHERE id(person) in $nodes "
164                          "SET person.is_special = true "
165                          "RETURN person", nodes=nodes):
166         print(f"- name: {record['person']['name']}, is_special: {record['person']['is_special
    ']}")
167
168
169 def execute_task10(tx):
170     print(f"\nNodes on the second place on path of size 4 between actor: \"Jack Nicholson\" and
     movie: \"Hoffa\":")
171
172     for record in tx.run(
173             """
174             MATCH p = (:Person {name: "Jack Nicholson"})-[*3]-(:Movie {title: "Hoffa"})
175             RETURN nodes(p)[1] as node
176             """):
177         print(f"- {record['node']}")
178
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()
```