# Project - Advanced Databases, Neo4j

Michał Ratajewski - 115727

## Database of deaths in the United States related to the abuse of narcotic substances along with information about the deceased and place of death.

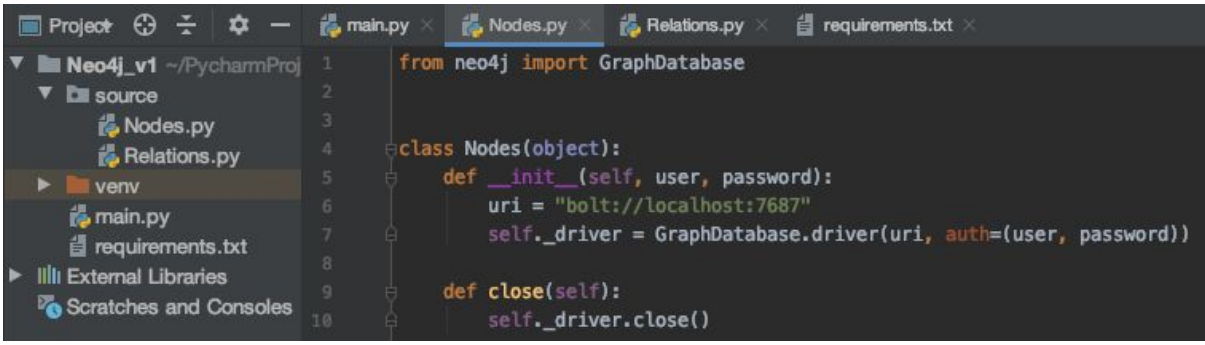
1. Implemented with graph database Neo4j and Python script.
2. Justification for the above choice:
    a. A single death case is associated with an irregular amount of intoxicants,
    b. Place of residence, person, substances detected in the body, place of death form an irregular network,
    c. Bilateral relationships occur when a person dies in their place of residence (recursion),
    d. Relationships are directed and may have properties (e.g. the relationship "died in:" {"reason": "overdose"} -> "Bristol"),
    e. Some of the data mentioned in point 'b' may be unknown / undisclosed at the time of creating the database and be added in the future, which allows Neo4j.
3. Specific properties to be used in the project:
    a. Importing data to the database from a CSV file,
    b. Using conditional functions (FOREACH),
    c. Automatic creation of indexes (CREATE CONSTRAINT),
    d. Creating a Python program that allows you to add relationships and nodes that match the template.
4. To create a project without interrupting local environment I used virtual environment:
    a. virtualenv cypher-app
    b. source cypher-app/bin/activate
    c. pip install -r /Users/michrata/cypher-app/requirements.txt
        i. This file contains the following content:

```
neo4j==1.7.2
neobolt==1.7.9
```

neotime==1.7.4

d. service neo4j restart
e. Going to http://localhost:7474 opens a new created database.

5. A structure of the python part of the project:
   a. Source directory contains Nodes.py and Relations.py files, that allow to create new nodes and relations in database.




6. To enforce data integrity neo4j documentation suggest the use of constraints. *"Constraints can be applied to either nodes or relationships. Unique node property constraints can be created, along with node and relationship property existence constraints, and Node Keys, which guarantee both existence and uniqueness."* I have applied constraint on my 3 different types of nodes: Person, City and Drug.
   a. CREATE CONSTRAINT ON (p:Person) ASSERT p.uID IS UNIQUE
   b. CREATE CONSTRAINT ON (c:City) ASSERT c.ID IS UNIQUE
   c. CREATE CONSTRAINT ON (d:Drug) ASSERT d.ID IS UNIQUE
   d. To validate call: `CALL db.constraints`
7. **# Init - creating people**
   LOAD CSV WITH HEADERS FROM "file:///Accidental_Drug_Related_Deaths.csv" as line
   CREATE (p:Person {ID: line.ID, Age: line.Age, Sex: line.Sex, Race: line.Race})
   FOREACH(ignoreMe IN CASE WHEN trim(line.ID) <> "" THEN [1] ELSE [] END | SET p.ID = line.ID)
   FOREACH(ignoreMe IN CASE WHEN trim(line.Sex) <> "" THEN [1] ELSE [] END | SET p.Sex = line.Sex)
   FOREACH(ignoreMe IN CASE WHEN trim(line.Race) <> "" THEN [1] ELSE [] END | SET p.Race = line.Race)
   RETURN p
8. Check what was created in a table format:
   MATCH (p:Person) RETURN p.ID, p.Age, p.Race, p.Sex
9. **# Init - creating cities - Issue, some people and deaths were not related to any city**
   LOAD CSV WITH HEADERS FROM "file:///Accidental_Drug_Related_Deaths.csv" as line
   WITH line

```
WHERE line.ResidenceCity IS NOT NULL
MERGE (c:City {City_Name: line.ResidenceCity})
ON CREATE SET c.City_Country = line.ResidenceCounty
FOREACH(ignoreMe IN CASE WHEN trim(line.ResidenceCity) <> "" THEN [1] ELSE
[] END | SET c.City_Name = line.ResidenceCity)
FOREACH(ignoreMe IN CASE WHEN trim(line.ResidenceCounty) <> "" THEN [1]
ELSE [] END | SET c.City_Country = line.ResidenceCounty)
RETURN c
---------------------------------
LOAD CSV WITH HEADERS FROM "file:///Accidental_Drug_Related_Deaths.csv" as
line
WITH line
WHERE line.DeathCity IS NOT NULL
MERGE (c:City {City_Name: line.DeathCity})
ON CREATE SET c.City_Country = line.DeathCounty
FOREACH(ignoreMe IN CASE WHEN trim(line.DeathCity) <> "" THEN [1] ELSE []
END | SET c.City_Name = line.DeathCity)
FOREACH(ignoreMe IN CASE WHEN trim(line.DeathCounty) <> "" THEN [1] ELSE []
END | SET c.City_Country = line.DeathCounty)
RETURN c
```

10. Check what was created in a table format, ordered by city name:
```
MATCH (c:City)
RETURN c.City_Name, c.City_Country
ORDER BY c.City_Name
```
11. **# Init - creating nodes representing drugs - it is necessary thus point no 17.**
```
CREATE (HEROINE:Drug {name:'Heroine'}),
(COCAINE:Drug {name:'Cocaine'}),
(FENTANY:Drug {name: 'Fentany'}),
(`FENTANY|ANALOGUE`:Drug {name: 'Fentany|Analogue'}),
(OXYCODONE:Drug {name: 'Oxycodone'}),
(OXYMORPHONE:Drug {name: 'Oxymorphone'}),
(ETHANOL:Drug {name: 'Ethanol'}),
(HYDROCODONE:Drug {name: 'Hydrocodone'}),
(BENZODIAZEPINE:Drug {name: 'Benzodiazepine'}),
(METHADONE:Drug {name:'Methadone'}),
(AMPHET:Drug {name: 'Amphet'}),
(TRAMAD:Drug {name: 'Tramad'}),
(MORPHINE_NOTHEROIN:Drug {name: 'Morphine, not heroine'}),
(HYDROMOTPHONE:Drug {name: 'Hydromotphone'}),
(OTHER:Drug {name: 'Other'}),
(OPIATENOS:Drug {name: 'Opiatenos'}),
(`ANYOPIOID`:Drug {name: 'Anyopioid'});
```
12. Check what was created by previous query:
```
MATCH (d:Drug)
RETURN d
```
13. **# Create relations:**

a. **Lived in:**

```
LOAD CSV WITH HEADERS FROM
"file:///Accidental_Drug_Related_Deaths.csv" as line
WITH line
WHERE line.ResidenceCity IS NOT NULL
MATCH (p:Person {ID: line.ID})
MATCH (c:City {City_Name: line.ResidenceCity})
MERGE (p)-[r:LIVED_IN]->(c)
RETURN count(*)
```

Check results with:

```
MATCH (p:Person)-[r]-(c:City)
        RETURN p, r, c
        LIMIT 1000
```

b. **Died in:**

```
LOAD CSV WITH HEADERS FROM
"file:///Accidental_Drug_Related_Deaths.csv" as line
WITH line
WHERE line.DeathCity IS NOT NULL
            MATCH (p:Person {ID: line.ID})
            MATCH (c:City {City_Name: line.DeathCity})
            MERGE (p)-[r:DIED_IN]->(c)
ON CREATE SET r.Location = line.Location
FOREACH(ignoreMe IN CASE WHEN trim(line.Location) <> "" THEN [1]
ELSE [] END | SET r.Location = line.Location)
RETURN count(*)
```

Check with:

```
MATCH (p:Person {Race:"White"})-[r]->(c:City)
Return p,r,c
LIMIT 500
```

14. **Used a drug:**

```
USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM "file:///Accidental_Drug_Related_Deaths.csv" as
line
MERGE (p:Person {ID: line.ID})
WITH p, line
UNWIND KEYS (line) AS x
WITH p, x
WHERE x IN ['Heroin', 'Cocaine', 'Fentanyl', 'FentanylAnalogue', 'Oxycodone',
'Oxymorphone', 'Ethanol', 'Hydrocodone', 'Benzodiazepine', 'Methadone', 'Amphet',
'Tramad', 'Morphine_NotHeroin', 'Hydromorphone, 'OpiateNOS', 'AnyOpioid'] AND
line[x] = 'Y'
MATCH (d:Drug {name: x})
MERGE (d)-[r:DETECTED_IN]->(p)
RETURN d,r,p
```

15. Class Nodes is useful to add new nodes like Person, City or to run any query you like.

```
    @classmethod
    def create_person(cls, tx, id, age, sex, race):
        tx.run("CREATE (p:Person {id:$id, age=$age, sex=$sex, race=$race}) "
               "RETURN id(p)", id=id, age=age, sex=age, race=race)

    @classmethod
    def create_city(cls, tx, city_name, city_country):
        tx.run("CREATE (c:City {city_country:$city_country}) "
               "RETURN id(c)", city_name=city_name, city_country=city_country)

    @classmethod
    def any_query(cls, tx, query):
        tx.run("$query", query=query)

    def add_person(self, id, age, sex, race):
        with self._driver.session() as session_a:
            session_a.write_transaction(self.create_person, id, age, sex, race)

    def add_city(self, city_name, city_country):
        with self._driver.session() as session_c:
            session_c.write_transaction(self.create_city, city_name, city_country)

    def run_any_query(self, any_query):
        with self._driver.session() as session_q:
            session_q.write_transaction(self.any_query, any_query)
```

16. Class Relations is useful to create new relations like LIVED_IN and DIED_IN.

```
class Relations(object):
    def __init__(self, user, password):
        uri = "bolt://localhost:7687"
        self._driver = GraphDatabase.driver(uri, auth=(user, password))

    def close(self):
        self._driver.close()

    @classmethod
    def died(cls, tx, person_id, dead_city_name, location):
        tx.run("MATCH (p:Person {ID=$id})"
               "MATCH (c:City {City_Name=$city_name})"
               "MERGE (p)-[r:DIED_IN {Location=$Location}]->(c)",
               id=person_id, city_name = dead_city_name, Location=location)

    @classmethod
    def lived(cls, tx, person_id, home_city_name, home_city_state):
        tx.run("MATCH (p:Person {ID=$id})"
               "MATCH (c:City {City_Name=$home_city_name, City_Country=$home_city_stat
               "MERGE (p)-[r:LIVED_IN]->(c)",
               id=person_id, home_city_name=home_city_name, home_city_state=home_city_

    @classmethod
    def overdosed(cls, tx, person_id, drug_name):
        tx.run("MATCH (p:Person {ID=$id}"
               "MATCH (d:Drug {name=$drug_name}"
               "MERGE (d)-[r:DETECTED_IN]->(p)",
               id=person_id, drug_name=drug_name)
```

```
    @classmethod
    def lived(cls, tx, person_id, home_city_name, home_city_state):
        tx.run("MATCH (p:Person {ID=$id})"
               "MATCH (c:City {City_Name=$home_city_name, City_Country=$home_city_state})"
               "MERGE (p)-[r:LIVED_IN]->(c)",
               id=person_id, home_city_name=home_city_name, home_city_state=home_city_state)

    @classmethod
    def overdosed(cls, tx, person_id, drug_name):
        tx.run("MATCH (p:Person {ID=$id}"
               "MATCH (d:Drug {name=$drug_name}"
               "MERGE (d)-[r:DETECTED_IN]->(p)",
               id=person_id, drug_name=drug_name)

    def add_dead(self, person_id, dead_city_name, location):
        with self._driver.session() as session_d:
            session_d.write_transaction(self.died, person_id, dead_city_name, location)

    def add_dead(self, person_id, home_city_name, home_city_country):
        with self._driver.session() as session_d:
            session_d.write_transaction(self.lived, person_id, home_city_name, home_city_country)

    def add_overdosing(self, person_id, drug_name):
        with self._driver.session() as session_e:
            session_e.write_transaction(self.overdosed, person_id, drug_name)
```

17. An example of using Nodes Class:

```python
from source.Nodes import Nodes
from source.Relations import Relations


if __name__ == '__main__':
    a = Nodes("neo4j", "grafowe01")
    a.add_person(8888-88, 99, 'male', 'black')
    a.add_city("Poznan", "Greater Poland")
```