System wspomagający organizację transportu pacjentów pomiędzy szpitalami

Projektowanie baz danych

Projekt nierelacyjnej bazy danych

Zespół:
Monika Galińska
Agnieszka Kłobus
Justyna Małuszyńska
Aleksandra Stecka

Etap 10.

Bazy NoSQL są bardzo przydatne w przypadku danych o dużym wolumenie, przede wszystkim dlatego, że łatwo jest skalować je horyzontalnie. W przypadku zbyt dużego ruchu na serwerze proste jest dodanie nowego serwera, który przechowuje kopię bazy danych i obsługuje część ruchu.
Zgodnie z teorią CAP baza danych może mieć co najwyżej dwie spośród trzech cech:

- spójność,
- dostępność,
- tolerancja partycjonowania.

Bazy SQL są spójne i dostępne. W bazach NoSQL postawiono na dostępność i tolerancję partycjonowania, przez co zazwyczaj nie ma w nich wsparcia dla transakcji.

Wady baz danych NoSQL:

- brak mechanizmów transakcyjnych,
- SQL jest szeroko znany,
- relacyjne bazy danych mają większą integralność i spójność danych,
- normalizacja w bazach relacyjnych pozwala na uniknięcie redundancji danych.

Zalety baz danych NoSQL:

- tańsze i prostsze w utrzymaniu,
- łatwo skalowalne horyzontalnie,
- pozwalają na przetwarzanie danych niestrukturalnych,
- brak predefiniowanych schema pozwala na większą elastyczność,
- bardziej przystosowane do przetwarzania Big Data, czyli dużych, zmiennych i różnorodnych zbiorów danych.

Rodzaje baz danych NoSQL:

- **bazy typu klucz-wartość (key-values stores)** – opierają się na kolekcji słowników, składających się z encji. W słownikach z kluczem powiązane są wartości różnych atrybutów dla różnych encji. W tym rodzaju baz stosowane są funkcje haszujące w celu przyspieszenia odczytu, więc stosujemy je głównie tam, gdzie dane często się odczytuje – **np. Redis, DynamoDB, Project Voldemort**
- **bazy kolumnowe (column stores)** – inwersja dla zapisu wierszowego, tzn. dane z tej samej kolumny zapisywane są obok siebie. Często są stosowane do przechowywania dużych ilości danych – **np. Cassandra, Hypertable**
- **bazy dokumentowe (document stores)** – dane przechowywane są w dokumentach zamiast w wierszach, a w każdym dokumencie obowiązuje struktura klucz-wartość. Są bardzo elastyczne, dzięki temu można wiernie odtwarzać dane rzeczywiste – **np. MongoDB, CouchDB, Orient DB**
- **bazy grafowe (graph stores)** – są oparte na teorii grafów. Każdy obiekt jest opisany węzłem w grafie, a relacje pomiędzy obiektami krawędziami w grafie. Dzięki takiej reprezentacji danych możliwe jest szybkie wyszukiwanie danych – **np. Neo4j, FlockDB, HyperGraphDB**

- **bazy obiektowe (object databases)** – ich struktura przypomina model wykorzystywany w obiektowych językach programowania wysokiego poziomu, np. Java, Python. Nie istnieje konieczność mapowanie z modelu relacyjnego na obiektowy, co znacznie skraca czas dostępu do danych – **np. Versant, Objectivity, EyeDB**

Źródła:
https://mfiles.pl/pl/index.php/Baza_NoSQL
https://mansfeld.pl/bazy-danych/bazy-danych-nosql-zalety-wady/

MongoDB

Zdecydowano się na wykorzystanie MongoDB jako nierelacyjnej bazy danych.

Zalety:
- wykorzystywanie plików BSON, które mają podobną strukturę do plików JSON,
- szybka obsługa dużych ilości zapytań i zapisów – istotna w opracowywanej bazie danych, ponieważ planowane jest częste odczytywanie, modyfikowanie i zapisywanie danych,
- skalowalność – istotna w opracowywanej bazie danych, ponieważ planowane jest przechowywanie dużej ilości danych,
- obsługa operacji CRUD – wiele opcji ułatwiających pracę na danych,
- brak schematów bazy danych w oddzielnych plikach,
- popularność MongoDB oznacza powszechną dostępność materiałów i dokumentacji – jest to bardzo przydatne dla początkujących projektantów baz nierelacyjnych,
- prostota formułowania zapytań,
- możliwość linkowania między dokumentami – można w ten sposób zaimplementować "relacje" – nie ma kluczy obcych, więc należy manualnie zatroszczyć się o poprawność danych,
- elastyczność i prostota implementacji,
- darmowe oprogramowanie.

Wady:
- bardzo duża ilość pamięci wykorzystywanej na przechowywanie danych,
- trudne opracowywanie indeksów – błędy w implementacji indeksów powodują drastyczne spowolnienie wyszukiwania danych, a znajdowanie błędów może być czasochłonne,
- brak wsparcia dla funkcji Join,
- ograniczone wsparcie dla transakcji.
- wielkość dokumentu nie większa niż 16MB,
- redundancja danych.

Źródła:
https://acodez.in/mongodb-nosql-database/
https://www.knowledgenile.com/blogs/pros-and-cons-of-mongodb/#MongoDBLimitedDataSizeandNesting
https://code.tutsplus.com/articles/mapping-relational-databases-and-sql-to-mongodb--net-35650

Inne bazy NoSQL

Cassandra:

- przydatna kiedy występuje więcej operacji zapisywania niż odczytywania,
- przydatna jeśli istotniejsza jest dostępność niż spójność danych,
- gorzej radzi sobie w przypadkach gdy potrzebna jest duża ilość agregacji oraz joinów.

Ze względu na konieczność odczytywania danych częściej niż zapisywania ich, oraz konieczność zapewnienia spójności danych, Cassandra nie została wybrana do realizacji projektu.

ElasticSearch:

- szczególnie przydatna przy wyszukiwaniu full-text,
- obsługuje wyszukiwanie rozmyte (fuzzy matching),
- przydatne przy przechowywaniu danych z logów.

W naszej bazie nie występuje konieczność wyszukiwania full-text, zatem ElasticSearch nie zostało wybrane.

Amazon DynamoDB:

- przydatna kiedy potrzebna jest wysoka spójność danych w bazie,
- dobrze radzi sobie kiedy przechowywane są proste wartości key-value, których ilość jest bardzo duża,
- nie obsługuje żadnej metody implementacji połączeń join,
- tworzenie kwerend jest bardzo trudne.

Ze względu na konieczność mapowania połączeń między tabelami naszego modelu danych, a także trudność tworzenia kwerend Amazon DynamoDB nie został wybrany do realizacji projektu.

HBase:

- aby zauważyć jej przydatność konieczne jest przechowywanie co najmniej petabajtów danych,
- przydatna przy przechowywaniu dużej ilości danych zmieniających się w real-time.

Ze względu na brak przewidywań co do przechowywania tak znaczącej ilości danych, HBase nie zostało wybrane.

RavenDB:

- szczególnie przydatny przy projektach używających cache,
- potrafi przyjąć na raz duże ilości danych.

Przez brak potrzeby przechowywania cache w bazie, RavenDb nie zostało wybrane w naszym projekcie.

Weryfikacja przyjętych założeń i ograniczeń

Zweryfikowano przyjęte założenia i ograniczenia – nie zmieniają się one. Ponownie baza uwzględniać ma jedynie osoby, które korzystały lub będą korzystać z przewozów - za wyjątkiem pacjentów, którzy zorganizowali transport we własnym zakresie – a także działać działać tylko dla Polski i uwzględniać jedynie placówki publiczne. Ponownie nie przewiduje się usuwania danych.
Ze względu na wykorzystanie MongoDB i możliwość linkowania pomiędzy dokumentami nie zmienia się model domenowy zaprojektowany w części pierwszej projektu.

Poprawki do etapu 10

W etapie 10 podjęto złą decyzję dotyczącą wyboru systemu zarządzania nierelacyjną bazą danych. Wynikała ona z braku wiedzy zespołu na temat baz nierelacyjnych – ta część projektu jest pierwszą stycznością z bazami NoSQL dla całego zespołu.

MongoDB jest bazą dokumentową. Nie wspiera funkcji Join, ale umożliwia linkowanie pomiędzy dokumentami, które należy obsługiwać programowo. Linkowanie między dokumentami można wykorzystać jako alternatywę dla relacji 1 do wiele w bazach relacyjnych. W przypadku relacji 1 do 1 MongoDB zakłada redundancję danych – jeden dokument przechowywany jest jako atrybut w innym dokumencie.


```
{
    "_id": {
        "$oid": "61caecaadba9c9f98afe719d"
    },
    "Name": "Szpital im. Jana Pawła II",
    "Location": {
        "City": "Wrocław",
        "Address": "Marcinkowska 123b",
        "Postal_code": "23453"
    },
    "Id_user": "76caecaadba9c923148afe719d"
}
```


Aby zadbać o poprawność linkowania między dokumentami konieczne jest programowe dbanie o zgodność id. Ze względu na znaczną ilość relacji w bazie, nie jest to optymalne rozwiązanie dla tego projektu.

Z uwagi na liczbę relacji w projekcie relacyjnej bazy danych należało wybrać bazę grafową. W bazie grafowej obiekt jest reprezentowany przez węzeł grafu, a relacje pomiędzy obiektami – przez krawędzie grafu. Bazy grafowe są więc lepiej przystosowane do realizacji baz danych, w których pomiędzy danymi istnieje wiele relacji.

Z tego względu zdecydowano się na wykorzystanie Neo4j, zaproponowanego przez prowadzącego, który jest grafowym systemem zarządzania bazą danych.

Neo4j

Zalety:

- wydajniejsze niż w bazach relacyjnych wyszukiwanie danych w przypadku gdy danych jest dużo i są mocno połączone,
- nie wymaga skomplikowanych struktur Join ani indeksów do szybkiego wyszukiwania połączonych danych,
- możliwość definiowania zoptymalizowanych ścieżek, które mogą być wykorzystywane w konkretnych przypadkach wyszukiwania danych,

- bardzo szybkie odczytywanie danych,
- elastyczność wynikająca z grafowego modelu danych,
- pełne wsparcie dla transakcji ze względu na spełnianie paradygmatu ACID,
- prosty i silny język definiowania zapytań Cypher,
- wysoka jakość dokumentacji,
- oprogramowanie open-source,
- darmowe oprogramowanie.

Wady:

- ograniczone wsparcie dla sharding – jeśli graf można łatwo podzielić, fragmenty bazy danych mogą być przechowywane na osobnych serwerach, w przeciwnym wypadku przechowywanie bazy na kilku serwerach jest bardzo trudne, a nawet niemożliwe,
- zapisywanie danych zawsze odbywa się na węźle master – Neo4j operuje na architekturze master-slave,
- ograniczony rozmiar grafu – górny limit to około 34 miliardy węzłów i relacji oraz około 274 miliardy własności – nie jest to problemem w opracowywanej bazie danych, ale to ograniczenie może być zbyt małe dla dużych firm operujących na dużych ilościach danych,
- brak wbudowanej obsługi dat.

Etap 11.

W bazie Neo4j tworzenie kolejnych węzłów jest równoznaczne z definiowaniem tabel – w związku z tym dodawanie nowych encji jest uproszczone, definiowanie ich na początku pracy nie jest konieczne. Zaczynamy więc od utworzenia bazy z przykładowymi danymi, co jednocześnie tworzy kolejne encje.

Tworzenie węzła:

```
CREATE (nazwa: label {atrybut: wartość,  atrybut: wartość, …})
```

Tworzenie relacji między obiektami:

```
MATCH (a:label),(b:label)
WHERE a.atrybut = wartość AND b.atrybut = wartość
CREATE (a)-[r:nazwa_relacji {atrybut: wartość,  atrybut: wartość, …}]->(b)
```

Usuwanie obiektu spełniającego jakiś warunek:

```
MATCH (n: label {atrybut: wartość}) DETACH DELETE n
```

Usuwanie wszystkich obiektów:

```
MATCH (n) DETACH DELETE n
```

Usuwanie właściwości z węzła:

```
MATCH (a {atrybut: wartość}) REMOVE a.wł
```

Usuwanie relacji po id:

```
MATCH ()-[r]-() WHERE id(r) = wartość DELETE r
```

Dodanie nowego atrybutu / edycja atrybutu relacji:

```
MATCH ()-[r]-() WHERE id(r) = wartość SET r.atrybut = wartość
```

Dodanie nowego atrybutu / edycja wartości atrybutu węzła:

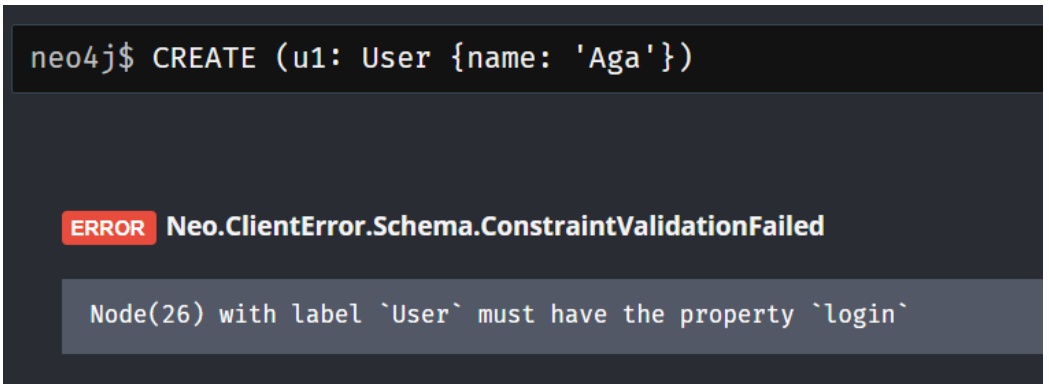
```
MATCH (n) WHERE id(n) = wartość SET n.atrybut = wartość
```

Aby zapewnić spójność bazy (mieć pewność że potrzebne dane są tworzone w poszczególnych węzłach oraz są unikatowe) tworzone są Constraints. Dla każdego pola musi być on utworzony osobno.

```
CREATE CONSTRAINT
FOR (n:User)
REQUIRE n.login IS NOT NULL
```

```
CREATE CONSTRAINT
FOR (n:User)
REQUIRE n.login IS UNIQUE
```

Analogicznie definiowane są ograniczenia NOT NULL oraz UNIQUE dla reszty pól w bazie.



Ograniczenia nałożone na label User nie umożliwiają dodania użytkownika bez loginu

Nie jest możliwe ograniczenie ilości relacji między węzłami – zapewnienie, że nie zostanie utworzona więcej niż jedna relacja jeśli chcemy osiągnąć połączenie jeden-do-jednego nie jest możliwe.

Przechowywanie historii

W bazie relacyjnej historia pasażerów oraz transportów była przechowywana w osobnej tabeli, aby zwiększyć czytelność. Neo4j jest skupione na przechowywaniu relacji raczej niż danych. Uznano, że w bazie Neo4j bardziej czytelne będzie przechowywanie jako atrybuty relacji lub węzłów daty od kiedy i do kiedy dane były aktualne.

Tworzenie bazy z przykładowymi danymi

```
CREATE (:User:Administrator {login: 'jankowalski_1', password: 'pa55w0rd',
name: 'Jan', surname: 'Kowalski'})

CREATE (:User:Supervisor {login: 'ulawisniewska_1', password: 'pa55w0rd',
name: 'Ula', surname: 'Wiśniewska'})
CREATE (:User:Supervisor {login: 'zuzalewandowska_1', password: 'pa55w0rd',
name: 'Zuzanna', surname: 'Lewandowska'})
CREATE (:User:Supervisor {login: 'karolwisniewski_1', password: 'pa55w0rd',
name: 'Karol', surname: 'Wiśniewski'})

CREATE (:User:Doctor {login: 'oleknowak_1', password: 'pa55w0rd', name:
'Aleksander', surname: 'Nowak'})
CREATE (:User:Doctor {login: 'antekwozniak_1', password: 'pa55w0rd', name:
'Antoni', surname: 'Woźniak'})
```

```
CREATE (:User:Doctor {login: 'kubakowalczyk_1', password: 'pa55w0rd', name: 'Jakub', surname: 'Kowalczyk'})

CREATE (:User:Doctor:Medic {login: 'alicjanowak_1', password: 'pa55w0rd', name: 'Alicja', surname: 'Nowak'})

CREATE (:User:Medic {login: 'elakowalczyk_1', password: 'pa55w0rd', name: 'Elżbieta', surname: 'Kowalczyk'})
CREATE (:User:Medic {login: 'elakowalczyk_1', password: 'pa55w0rd', name: 'Elżbieta', surname: 'Kowalczyk'})
CREATE (:User:Medic {login: 'elakowalczyk_1', password: 'pa55w0rd', name: 'Elżbieta', surname: 'Kowalczyk'})

CREATE (:User:Driver {login: 'staskowalski_1', password: 'pa55w0rd', name: 'Stanisław', surname: 'Kowalski'})
CREATE (:User:Driver {login: 'ewaokon_1', password: 'pa55w0rd', name: 'Ewa', surname: 'Okoń'})
CREATE (:User:Driver {login: 'marcinabacki_1', password: 'pa55w0rd', name: 'Marcin', surname: 'Abacki'})

CREATE (:User:AdmEmployee {login: 'makssikora_1', password: 'pa55w0rd', name: 'Masymilian', surname: 'Sikora'})
CREATE (:User:AdmEmployee {login: 'robertmichalik_1', password: 'pa55w0rd', name: 'Robert', surname: 'Michalik'})
CREATE (:User:AdmEmployee {login: 'justynapietrzak_1', password: 'pa55w0rd', name: 'Justyna', surname: 'Pietrzak'})

CREATE (:Patient {name: 'Maksymilian', surname: 'Okoń'})
CREATE (:Patient {name: 'Anna', surname: 'Kowalska'})
CREATE (:Patient {name: 'Joanna', surname: 'Bury'})

CREATE (:MedicalFacility {name: "Uniwersytecki Szpital Kliniczny"})
CREATE (:MedicalFacility {name: "Szpital Wojewódzki w Sieradzu"})
CREATE (:MedicalFacility {name: "Szpital im. Jana Pawła II"})
CREATE (:MedicalFacility {name: "Praktyka Lekarza Rodzinnego Doktor Beata Stecka"})

CREATE (:Location {city: "Wrocław", address: "Fiołkowa 12/3", postalCode: "50-231"})
CREATE (:Location {city: "Sieradz", address: "Sieradzka 58/13", postalCode: "98-200"})
CREATE (:Location {city: "Radom", address: "Warcka 8a", postalCode: "67-342"})

CREATE (:HospitalWard {name: 'Oddział Psychiatryczny', places: '128'})
CREATE (:HospitalWard {name: 'Oddział Kardiologiczny', places: '535'})
CREATE (:HospitalWard {name: 'Oddział Dziecięcy', places: '2137'})

CREATE (:Vehicle {brand: 'Audi', model: 'Q8', licensePlate: "D0 MEDYK", seats: 8})
CREATE (:Vehicle {brand: 'Fiat', model: '126p', licensePlate: "D1 MEDYK", seats: 18})
```

```
CREATE (:Vehicle {brand: 'Kia', model: 'Rio', licensePlate: "E0 MEDYK", seats: 6})

CREATE (:MedicalExam {name: 'Rezonans'})
CREATE (:MedicalExam {name: 'Teleporada'})
CREATE (:MedicalExam {name: 'Badanie krwi'})

CREATE (:Transfer {startTime: "18:00", startDate: "01-01-2022", status: "Scheduled", validSince: "12-12-2021 12:56"})
CREATE (:Transfer {startTime: "11:15", startDate: "01-12-2021", status: "Scheduled", validSince: "01-03-2021 9:30", validUntil: "12-03-2021 08:12"})
CREATE (:Transfer {startTime: "11:15", startDate: "01-12-2021", status: "Realised", validSince: "12-03-2021 08:12"})
CREATE (:Transfer {startTime: "12:05", startDate: "15-03-2021", status: "Scheduled", validSince: "01-02-2021 09:09", validUntil: "03-02-2021 09:09"})
CREATE (:Transfer {startTime: "12:05", startDate: "15-03-2021", status: "Cancelled", validSince: "03-02-2021 09:09"})

CREATE (:Referral {Referral Date: '22-03-2021', BodyPart: 'Ear'})
CREATE (:Referral {Referral Date: '10-02-2021', BodyPart: 'Leg'})
CREATE (:Referral {Referral Date: '13-12-2021', BodyPart: 'Eye'})

MATCH (a:Administrator),(b:Supervisor)
WHERE id(a) = 0 AND id(b) = 1
CREATE (a)-[:CREATES]->(b)
MATCH (a:Administrator),(b:Supervisor)
WHERE id(a) = 0 AND id(b) = 2
CREATE (a)-[:CREATES]->(b)
MATCH (a:Administrator),(b:Supervisor)
WHERE id(a) = 0 AND id(b) = 3
CREATE (a)-[:CREATES]->(b)

MATCH (a:Supervisor),(b:MedicalFacility)
WHERE id(a) = 3 AND id(b) = 12
CREATE (a)-[:MANAGES]->(b)
MATCH (a:Supervisor),(b:MedicalFacility)
WHERE id(a) = 1 AND id(b) = 11
CREATE (a)-[:MANAGES]->(b)
MATCH (a:Supervisor),(b:MedicalFacility)
WHERE id(a) = 3 AND id(b) = 28
CREATE (a)-[:MANAGES]->(b)

MATCH (a:User),(b:MedicalFacility)
WHERE id(a) = 21 AND id(b) = 12
CREATE (a)-[:WORKS]->(b)
MATCH (a:User),(b:MedicalFacility)
WHERE id(a) = 19 AND id(b) = 12
CREATE (a)-[:WORKS]->(b)
MATCH (a:User),(b:MedicalFacility)
WHERE id(a) = 20 AND id(b) = 13
CREATE (a)-[:WORKS]->(b)
MATCH (a:User),(b:MedicalFacility)
WHERE id(a) = 4 AND id(b) = 13
```

```
CREATE (a)-[:WORKS]->(b)
MATCH (a:User),(b:MedicalFacility)
WHERE id(a) = 5 AND id(b) = 28
CREATE (a)-[:WORKS]->(b)
MATCH (a:User),(b:MedicalFacility)
WHERE id(a) = 6 AND id(b) = 11
CREATE (a)-[:WORKS]->(b)
MATCH (a:User),(b:MedicalFacility)
WHERE id(a) = 18 AND id(b) = 13
CREATE (a)-[:WORKS]->(b)
MATCH (a:User),(b:MedicalFacility)
WHERE id(a) = 17 AND id(b) = 28
CREATE (a)-[:WORKS]->(b)
MATCH (a:User),(b:MedicalFacility)
WHERE id(a) = 16 AND id(b) = 11
CREATE (a)-[:WORKS]->(b)
MATCH (a:User),(b:MedicalFacility)
WHERE id(a) = 10 AND id(b) = 12
CREATE (a)-[:WORKS]->(b)
MATCH (a:User),(b:MedicalFacility)
WHERE id(a) = 9 AND id(b) = 12
CREATE (a)-[:WORKS]->(b)
MATCH (a:User),(b:MedicalFacility)
WHERE id(a) = 8 AND id(b) = 11
CREATE (a)-[:WORKS]->(b)
MATCH (a:User),(b:MedicalFacility)
WHERE id(a) = 7 AND id(b) = 28
CREATE (a)-[:WORKS]->(b)

MATCH (a:MedicalFacility),(b:Location)
WHERE id(a) = 13 AND id(b) = 22
CREATE (a)-[:IS_IN]->(b)
MATCH (a:MedicalFacility),(b:Location)
WHERE id(a) = 12 AND id(b) = 14
CREATE (a)-[:IS_IN]->(b)
MATCH (a:MedicalFacility),(b:Location)
WHERE id(a) = 11 AND id(b) = 15
CREATE (a)-[:IS_IN]->(b)
MATCH (a:MedicalFacility),(b:Location)
WHERE id(a) = 28 AND id(b) = 15
CREATE (a)-[:IS_IN]->(b)

MATCH (a:MedicalFacility),(b:HospitalWard)
WHERE id(a) = 13 AND id(b) = 30
CREATE (a)-[:OWNS]->(b)
MATCH (a:MedicalFacility),(b:HospitalWard)
WHERE id(a) = 13 AND id(b) = 27
CREATE (a)-[:OWNS]->(b)
MATCH (a:MedicalFacility),(b:HospitalWard)
WHERE id(a) = 11 AND id(b) = 29
CREATE (a)-[:OWNS]->(b)

MATCH (a:Vehicle),(b:MedicalFacility)
```

```
WHERE id(a) = 32 AND id(b) = 28
CREATE (a)-[:BELONGS_TO]->(b)

MATCH (a:Vehicle),(b:MedicalFacility)
WHERE id(a) = 32 AND id(b) = 28
CREATE (a)<-[:HAS]-(b)

MATCH (a:Vehicle),(b:MedicalFacility)
WHERE id(a) = 31 AND id(b) = 28
CREATE (a)-[:BELONGS_TO]->(b)
MATCH (a:Vehicle),(b:MedicalFacility)
WHERE id(a) = 31 AND id(b) = 28
CREATE (a)<-[:HAS]-(b)

MATCH (a:Vehicle),(b:MedicalFacility)
WHERE id(a) = 33 AND id(b) = 13
CREATE (a)-[:BELONGS_TO]->(b)

MATCH (a:Vehicle),(b:MedicalFacility)
WHERE id(a) = 33 AND id(b) = 13
CREATE (a)<-[:HAS]-(b)

MATCH (a:MedicalFacility),(b:MedicalExam)
WHERE id(a) = 11 AND id(b) = 34
CREATE (a)-[:OFFERS {limit: 20}]->(b)
MATCH (a:MedicalFacility),(b:MedicalExam)
WHERE id(a) = 11 AND id(b) = 35
CREATE (a)-[:OFFERS {limit: 25}]->(b)
MATCH (a:MedicalFacility),(b:MedicalExam)
WHERE id(a) = 11 AND id(b) = 36
CREATE (a)-[:OFFERS {limit: 30}]->(b)
MATCH (a:MedicalFacility),(b:MedicalExam)
WHERE id(a) = 12 AND id(b) = 34
CREATE (a)-[:OFFERS {limit: 40}]->(b)
MATCH (a:MedicalFacility),(b:MedicalExam)
WHERE id(a) = 13 AND id(b) = 36
CREATE (a)-[:OFFERS {limit: 20}]->(b)

MATCH (a:MedicalFacility),(b:MedicalExam)
WHERE id(a) = 28 AND id(b) = 34
CREATE (a)-[:OFFERS {limit: 15}]->(b)
MATCH (a:MedicalFacility),(b:MedicalExam)
WHERE id(a) = 28 AND id(b) = 36
CREATE (a)-[:OFFERS {limit: 25}]->(b)

MATCH (a:Transfer),(b:Vehicle)
WHERE id(a) = 38 AND id(b) = 31
CREATE (a)-[:BOOKS]->(b)
MATCH (a:Transfer),(b:Vehicle)
WHERE id(a) = 39 AND id(b) = 31
CREATE (a)-[:BOOKS]->(b)
MATCH (a:Transfer),(b:Vehicle)
WHERE id(a) = 37 AND id(b) = 32
```

```
CREATE (a)-[:BOOKS]->(b)

MATCH (a:Transfer),(b:Vehicle)
WHERE id(a) = 43 AND id(b) = 31
CREATE (a)-[:BOOKS]->(b)

MATCH (a:Transfer),(b:Vehicle)
WHERE id(a) = 26 AND id(b) = 31
CREATE (a)-[:BOOKS]->(b)

MATCH (a:Transfer),(b:Driver)
WHERE id(a) = 37 AND id(b) = 16
CREATE (a)-[:EXECUTED_BY]->(b)
MATCH (a:Transfer),(b:Driver)
WHERE id(a) = 38 AND id(b) = 16
CREATE (a)-[:EXECUTED_BY]->(b)
MATCH (a:Transfer),(b:Driver)
WHERE id(a) = 39 AND id(b) = 17
CREATE (a)-[:EXECUTED_BY]->(b)

MATCH (a:Transfer),(b:Driver)
WHERE id(a) = 26 AND id(b) = 16
CREATE (a)-[:EXECUTED_BY]->(b)

MATCH (a:Transfer),(b:Driver)
WHERE id(a) = 43 AND id(b) = 17
CREATE (a)-[:EXECUTED_BY]->(b)

MATCH (a:Patient),(b:HospitalWard)
WHERE id(a) = 25 AND id(b) = 30
CREATE (a)-[:OCCUPIES_PLACE {admissionDate: '01-01-2021', dischargeDate:
'11-02-2021'}]->(b)
MATCH (a:Patient),(b:HospitalWard)
WHERE id(a) = 25 AND id(b) = 30
CREATE (a)-[:OCCUPIES_PLACE {admissionDate: '23-03-2021'}]->(b)
MATCH (a:Patient),(b:HospitalWard)
WHERE id(a) = 24 AND id(b) = 29
CREATE (a)-[:OCCUPIES_PLACE {admissionDate: '21-06-2021', dischargeDate:
'11-07-2021'}]->(b)
MATCH (a:Patient),(b:HospitalWard)
WHERE id(a) = 23 AND id(b) = 27
CREATE (a)-[:OCCUPIES_PLACE {admissionDate: '11-01-2021', dischargeDate:
'11-02-2021'}]->(b)
MATCH (a:Patient),(b:HospitalWard)
WHERE id(a) = 23 AND id(b) = 27
CREATE (a)-[:OCCUPIES_PLACE {admissionDate: '19-03-2021'}]->(b)

MATCH (a:Transfer),(b:MedicalFacility)
WHERE id(a) = 37 AND id(b) = 28
CREATE (a)-[:IS_FROM]->(b)
MATCH (a:Transfer),(b:MedicalFacility)
WHERE id(a) = 38 AND id(b) = 13
CREATE (a)-[:IS_FROM]->(b)
```

```
MATCH (a:Transfer),(b:MedicalFacility)
WHERE id(a) = 39 AND id(b) = 13
CREATE (a)-[:IS_FROM]->(b)
MATCH (a:Transfer),(b:MedicalFacility)
WHERE id(a) = 26 AND id(b) = 13
CREATE (a)-[:IS_FROM]->(b)
MATCH (a:Transfer),(b:MedicalFacility)
WHERE id(a) = 43 AND id(b) = 13
CREATE (a)-[:IS_FROM]->(b)

MATCH (a:Transfer),(b:MedicalFacility)
WHERE id(a) = 37 AND id(b) = 12
CREATE (a)-[:IS_TO]->(b)
MATCH (a:Transfer),(b:MedicalFacility)
WHERE id(a) = 38 AND id(b) = 28
CREATE (a)-[:IS_TO]->(b)
MATCH (a:Transfer),(b:MedicalFacility)
WHERE id(a) = 39 AND id(b) = 11
CREATE (a)-[:IS_TO]->(b)

MATCH (a:MedicalExam),(b:Patient)
WHERE id(a) = 34 AND id(b) = 23
CREATE (a)-[:IS_RESERVED {startTime: "18:00", startDate:"01-01-2021"}]->(b)
MATCH (a:MedicalExam),(b:Patient)
WHERE id(a) = 35 AND id(b) = 24
CREATE (a)-[:IS_RESERVED {startTime: "17:00", startDate:"01-01-2021"}]->(b)
MATCH (a:MedicalExam),(b:Patient)
WHERE id(a) = 36 AND id(b) = 25
CREATE (a)-[:IS_RESERVED {startTime: "14:20", startDate:"11-02-2021"}]->(b)
MATCH (a:MedicalExam),(b:Patient)
WHERE id(a) = 35 AND id(b) = 23
CREATE (a)-[:IS_RESERVED {startTime: "11:10", startDate:"21-01-2022"}]->(b)
MATCH (a:MedicalExam),(b:Patient)
WHERE id(a) = 36 AND id(b) = 23
CREATE (a)-[:IS_RESERVED {startTime: "12:00", startDate:"01-02-2022"}]->(b)
MATCH (a:MedicalExam),(b:Patient)
WHERE id(a) = 36 AND id(b) = 25
CREATE (a)-[:IS_RESERVED {startTime: "09:00", startDate:"23-02-2022"}]->(b)

MATCH (a:Referral),(b:Patient)
WHERE id(a) = 42 AND id(b) = 24
CREATE (a)-[:GETS]->(b)
MATCH (a:Referral),(b:Patient)
WHERE id(a) = 40 AND id(b) = 23
CREATE (a)-[:GETS]->(b)
MATCH (a:Referral),(b:Patient)
WHERE id(a) = 41 AND id(b) = 23
CREATE (a)-[:GETS]->(b)

MATCH (a:Referral),(b:Doctor)
WHERE id(a) = 40 AND id(b) = 5
CREATE (a)-[:ISSUED_BY]->(b)
MATCH (a:Referral),(b:Doctor)
```

```
WHERE id(a) = 41 AND id(b) = 4
CREATE (a)-[:ISSUED_BY]->(b)
MATCH (a:Referral),(b:Doctor)
WHERE id(a) = 42 AND id(b) = 6
CREATE (a)-[:ISSUED_BY]->(b)

MATCH (a:Referral),(b:MedicalExam)
WHERE id(a) = 42 AND id(b) = 34
CREATE (b)-[:CONCERNS]->(a)
MATCH (a:Referral),(b:MedicalExam)
WHERE id(a) = 40 AND id(b) = 35
CREATE (b)-[:CONCERNS]->(a)

MATCH (a:Transfer),(b:Medic)
WHERE id(a) = 39 AND id(b) = 7
CREATE (a)-[:OVERSEEN_BY]->(b)
MATCH (a:Transfer),(b:Medic)
WHERE id(a) = 39 AND id(b) = 10
CREATE (a)-[:OVERSEEN_BY]->(b)
MATCH (a:Transfer),(b:Medic)
WHERE id(a) = 38 AND id(b) = 7
CREATE (a)-[:OVERSEEN_BY]->(b)
MATCH (a:Transfer),(b:Medic)
WHERE id(a) = 26 AND id(b) = 7
CREATE (a)-[:OVERSEEN_BY]->(b)
MATCH (a:Transfer),(b:Medic)
WHERE id(a) = 43 AND id(b) = 10
CREATE (a)-[:OVERSEEN_BY]->(b)
MATCH (a:Transfer),(b:Medic)
WHERE id(a) = 43 AND id(b) = 7
CREATE (a)-[:OVERSEEN_BY]->(b)

MATCH (a:Patient),(b:Transfer)
WHERE id(a) = 25 AND id(b) = 39
CREATE (a)-[:PARTICIPATES {status: "lying"]->(b)
MATCH (a:Patient),(b:Transfer)
WHERE id(a) = 24 AND id(b) = 37
CREATE (a)-[:PARTICIPATES {status: "walking"]->(b)
MATCH (a:Patient),(b:Transfer)
WHERE id(a) = 23 AND id(b) = 26
CREATE (a)-[:PARTICIPATES {status: "sitting"]->(b)
MATCH (a:Patient),(b:Transfer)
WHERE id(a) = 23 AND id(b) = 38
CREATE (a)-[:PARTICIPATES {status: "lying"]->(b)

MATCH (a:Transfer),(b:Transfer)
WHERE id(a) = 39 AND id(b) = 43
CREATE (a)<-[:PREVIOUS]-(b)
MATCH (a:Transfer),(b:Transfer)
WHERE id(a) = 26 AND id(b) = 38
CREATE (a)<-[:PREVIOUS]-(b)
```

Definiowane związków pomiędzy przechowywanymi danymi w Neo4j

Neo4j przechowuje dane w węzłach (nodes). Między nimi można zdefiniować związki (relationships).

Model bazy danych grafów właściwości Neo4j składa się z:

- Węzły opisują encje (obiekty dyskretne) domeny.
- Węzły mogą mieć zero lub więcej etykiet do definiowania (klasyfikowania) rodzaju węzłów.
- Relacje opisują połączenie między węzłem źródłowym a węzłem docelowym.
- Relacje zawsze mają kierunek (jeden kierunek).
- Relacje muszą mieć typ (jeden typ), aby zdefiniować (zaklasyfikować) typ relacji.
- Węzły i relacje mogą mieć właściwości (pary klucz-wartość), które dodatkowo je opisują.

Mechanizmy zapewnienia spójności przez Neo4j

Neo4j korzysta z modelowania ACID, co samo w sobie zapewnia spójność bazy (C in ACID stands for Consistency).

- Wszystkie operacje na bazie danych, które uzyskują dostęp do grafu, indeksów lub schematu, muszą zostać wykonane w transakcji. Transakcje są jednowątkowe, ograniczone i niezależne. W jednym wątku można uruchomić wiele transakcji i będą one od siebie niezależne.
- Transakcje w Neo4j używają poziomu izolacji zatwierdzonego do odczytu (read-committed isolation level), co oznacza, że zobaczą dane zaraz po ich zatwierdzeniu, ale nie zobaczą danych w innych transakcjach, które nie zostały jeszcze zatwierdzone. Ten typ izolacji jest słabszy niż serializacja, ale zapewnia znaczne korzyści w zakresie wydajności, a jednocześnie jest wystarczający w przeważającej większości przypadków.
- Można ręcznie nałożyć blokady zapisu w węzłach i relacjach, aby osiągnąć wyższy poziom izolacji — poziom izolacji serializacji. Neo4j Java API umożliwia jawne blokowanie węzłów i relacji. Korzystanie z blokad daje możliwość symulowania efektów wyższych poziomów izolacji poprzez jawne uzyskiwanie i zwalnianie blokad. Na przykład, jeśli blokada zapisu zostanie nałożona na wspólnym węźle lub relacji, wszystkie transakcje będą serializowane na tej blokadzie — dając efekt poziomu izolacji serializacji.
- Bardzo ważne jest, aby zakończyć każdą transakcję. Transakcja nie zwolni blokad ani nabytej pamięci, dopóki nie zostanie zakończona. Wszystkie modyfikacje dokonane w transakcji są przechowywane w pamięci. Oznacza to, że bardzo duże aktualizacje muszą zostać podzielone na kilka transakcji, aby uniknąć wyczerpania pamięci.

- System blokad:
    1. Podczas dodawania, zmieniania lub usuwania właściwości w węźle lub relacji blokada zapisu zostanie nałożona na określony węzeł lub relację.
    2. Podczas tworzenia lub usuwania węzła zostanie nałożona blokada zapisu dla określonego węzła.
    3. Podczas tworzenia lub usuwania relacji blokada zapisu zostanie nałożona na konkretną relację i oba jej węzły.
    4. Blokady zostaną dodane do transakcji i zwolnione po zakończeniu transakcji.
- Ponieważ stosowane są blokady, możliwe jest wystąpienie zakleszczeń. Neo4j wykryje jednak wszelkie zakleszczenia (spowodowane uzyskaniem blokady) przed ich wystąpieniem i zgłosi wyjątek. Transakcja jest oznaczona do wycofania przed zgłoszeniem wyjątku. Wszystkie blokady nabyte przez transakcję będą nadal utrzymywane, ale zostaną zwolnione po zakończeniu transakcji. Po zwolnieniu blokad inne transakcje, które czekały na blokady utrzymywane przez transakcję powodującą zakleszczenie, mogą być kontynuowane. Praca wykonana przez transakcję powodującą zakleszczenie może następnie zostać ponowiona przez użytkownika w razie potrzeby.
- Podczas usuwania węzła lub relacji wszystkie właściwości tej jednostki zostaną automatycznie usunięte, ale relacje węzła nie zostaną usunięte. Neo4j wymusza ograniczenie (po zatwierdzeniu), że wszystkie relacje muszą mieć prawidłowy węzeł początkowy i węzeł końcowy. W efekcie oznacza to, że próba usunięcia węzła, który nadal ma powiązane z nim relacje, spowoduje zgłoszenie wyjątku przy commicie.
- Semantykę usuwania można podsumować w następujący sposób:
    1. Wszystkie właściwości węzła lub relacji zostaną usunięte po ich usunięciu.
    2. Usunięty węzeł nie może mieć żadnych powiązanych relacji, gdy transakcja zostanie zatwierdzona.
    3. Możliwe jest uzyskanie odniesienia do usuniętej relacji lub węzła, który nie został jeszcze zatwierdzony.
    4. Każda operacja zapisu na węźle lub relacji po jego usunięciu (ale jeszcze nie zatwierdzona) spowoduje zgłoszenie wyjątku.
    5. Próba uzyskania nowego lub pracy ze starym odwołaniem do usuniętego węzła lub relacji po zatwierdzeniu spowoduje zgłoszenie wyjątku.

Czym różni się paradygmat BASE od paradygmatu ACID?

ACID:

- **Atomic:**
  Wszystkie operacje w transakcji powiodły się lub każda operacja została wycofana. Każda transakcja jest albo prawidłowo przeprowadzana, albo proces zostaje zatrzymany, a baza danych powraca do stanu sprzed rozpoczęcia transakcji. Gwarantuje to, że wszystkie dane w bazie danych są prawidłowe.

- **Consistent:**
  Po zakończeniu transakcji baza danych jest strukturalnie poprawna. Przetworzona transakcja nigdy nie zagraża integralności strukturalnej bazy danych.
- **Isolated:**
  Transakcje nie walczą ze sobą. Kontrowersyjny dostęp do danych jest moderowany przez bazę danych, dzięki czemu transakcje wydają się przebiegać sekwencyjnie. Transakcje nie mogą naruszać integralności innych transakcji poprzez interakcję z nimi, gdy są jeszcze w toku.
- **Durable:**
  Skutki zastosowania transakcji są trwałe, nawet w przypadku porażki. Dane związane ze zrealizowaną transakcją zostaną zachowane nawet w przypadku awarii sieci czy zasilania. Jeśli transakcja się nie powiedzie, nie wpłynie to na manipulowane dane.

Z ACID zgodne są relacyjne bazy danych. Należą do nich np. MySQL, PostgreSQL, Oracle, SQLite i Microsoft SQL Server. Jeśli chodzi o technologie NoSQL, większość grafowych baz danych (w tym Neo4j) korzysta z modelu spójności ACID, aby zapewnić bezpieczeństwo i spójne przechowywanie danych.

BASE:

- **Basically Available:**
  Zamiast wymuszać natychmiastową spójność, bazy danych NoSQL modelowane BASE zapewniają dostępność danych poprzez ich rozprzestrzenianie i replikację w węzłach klastra baz danych. Baza danych wydaje się działać poprawnie przez większość czasu.
- **Soft State:**
  Zasoby nie muszą być spójne pod względem zapisu, ani różne repliki nie muszą być przez cały czas wzajemnie spójne. Ze względu na brak natychmiastowej spójności wartości danych mogą się zmieniać w czasie. Model BASE zrywa z koncepcją bazy danych, która wymusza własną spójność, delegując tę odpowiedzialność na deweloperów.
- **Eventually Consistent:**
  Zasoby wykazują spójność w pewnym późniejszym momencie (np. leniwie w czasie odczytu). Fakt, że BASE nie wymusza natychmiastowej spójności, nie oznacza, że nigdy jej nie osiąga. Jednak dopóki tak się nie stanie, odczyty danych są nadal możliwe (nawet jeśli mogą nie odzwierciedlać rzeczywistości).

Podobnie jak bazy danych SQL są prawie jednolicie zgodne z ACID, bazy danych NoSQL mają tendencję do dostosowywania się do zasad BASE. MongoDB, Cassandra i Redis należą do najpopularniejszych rozwiązań NoSQL, obok Amazon DynamoDB i Couchbase.

Neo4j – ACID:
https://neo4j.com/docs/java-reference/current/transaction-management/
https://neo4j.com/docs/getting-started/current/graphdb-concepts/

Przykładowe zapytania

1. Wszystkie przejazdy danego kierowcy

```
MATCH (:Driver {login: 'staskowalski_1'})<--(transfer:Transfer)
RETURN transfer
```

2. Lista placówek oferujących dane badanie

```
MATCH (:MedicalExam {name: 'Rezonans'})<--(facility:MedicalFacility)
RETURN facility
```

3. Wszystkie przejazdy danego samochodu

```
MATCH (transfer:Transfer)-->(:Vehicle{licensePlate: 'D0 MEDYK'})
RETURN transfer
```

Etap 12.

Wykorzystano język Python z uwagi na jego prostotę oraz dostępne biblioteki do generowana pseudolosowych wartości na przykład imion, nazwisk, dat, loginów czy numerów rejestracyjnych, a także kompatybilność z Neo4j oraz możliwość skorzystania z wtyczki do środowiska PyCharm Code With Me.
Wykorzystano bibliotekę Neomodel do nawiązania połączenia z bazą danych.
Pobrano aplikację Neo4j z https://neo4j.com/download/ i utworzono lokalną instancję bazy danych. Aby nawiązać połączenie z lokalną bazą danych wykorzystana została komenda:

```
config.DATABASE_URL = f'bolt://{login}:{password}@localhost:7687'
```

Skrypt w języku Python

Plik constants.py:

```
import datetime

START_DATE = datetime.date(2021, 1, 1)
END_DATE = datetime.date(2021, 11, 1)
EMPLOYMENT_DATE = datetime.date(1950, 1, 1)

SEATS = [1, 4, 5, 6, 7, 8, 9, 10, 13, 17, 18, 19, 20, 24]

ROLES = ["Adm", "Sup", "AEm", "Dri", "Doc", "Med"]

NEEDS_CARE = ["Yes", "No"]

STATUS = ["Lying", "Sitting", "Walking"]

TRANSFER_STATUS = ["Scheduled", "Realised", "Cancelled"]

BODY_PARTS = ["head", "spine", "arm", "hand", "finger", "leg", "knee",
"foot", "toe", "stomach", "shoulder", "eye",
              "ear", "breast", "prostate", "heart", "lung", "hip", "pelvis",
"appendix", "bladder", "liver", "kidney",
              "chest"]

HOSPITAL_WARDS = ["Oddzial Anestezjologii i Intensywnej Terapii", "Oddzial
Chirurgii Ogolnej", "Oddzial Dzieciecy",
                  "Oddzial Chirurgii Ogolnej i Onkologicznej", "Oddzial
Chirurgii Urazowo - Ortopedycznej",
                  "Oddzial Chorob Pluc", "Oddzial Chemioterapii", "Oddzial
Chorob Wewnetrznych", "Blok operacyjny",
                  "Oddzial Kardiologiczny", "Oddzial Nefrologiczny", "Oddzial
Chorob Wewnetrznych",
                  "Oddzial Neonatologiczny", "Oddzial Neurologiczny",
"Oddzial Pediatryczny", "Oddzial Nefrologii",
```

```
                  "Oddzial Polozniczo – Ginekologiczny", "Oddzial
Psychiatryczny", "Oddzial Endokrynologii",
                  "Oddzial Rehabilitacji Neurologicznej", "Oddzial
Rehabilitacji Kardiologicznej",
                  "Oddzial Rehabilitacyjny Ogolny", "Oddzial Udarowy",
"Oddzial Urologiczny", "Stacja Dializ",
                  "Szpitalny Oddzial Ratunkowy", "Oddzial Kardiochirurgii",
"Oddzial Obserwacyjno – Zakazny",
                  "Oddzial Dzienny Psychiatryczny", "Oddzial Chirurgii
Plastycznej", "Oddzial Neurochirurgiczny",
                  "Oddzial Okulistyczny", "Oddzial Psychiatryczny", "Oddzial
Leczenia Uzaleznien",
                  "Oddzial Rehabilitacyjny", "Oddzial Chirurgii Naczyniowej",
"Oddzial Nadcisnienia Tetniczego",
                  "Oddzial Gastroenterologii", "Oddzial Laryngologiczny",
"Oddzial Onkologii Klinicznej",
                  "Oddzial Reumatologiczny", "Oddzial Otolaryngologiczny",
"Oddzial Geriatryczny"]

# users limits
ADM_EMPLOYEE_LIMIT = 500
DOCTORS_LIMIT = 2000
MEDICS_LIMIT = 1000
DRIVERS_LIMIT = 700
SUPERVISORS_LIMIT = 100
ADMINISTRATORS_LIMIT = 100

# other limits
LOCATIONS_LIMIT = 1250
MEDICAL_FACILITIES_LIMIT = 400
HOSPITAL_WARDS_LIMIT = 1500
VEHICLES_LIMIT = 1000
TRANSFERS_LIMIT = 7000
PATIENTS_LIMIT = 7500
REFERRALS_LIMIT = 9000
MEDICAL_EXAMS_LIMIT = 175
IS_OCCUPIED_LIMIT = 10000

# relationships limits
MEDICS_IN_TRANSFER_LIMIT = 4
OFFERS_LIMIT = 3000
WARDS_IN_FACILITY_LIMIT = 10
MEDICAL_FACILITY_HAS_LIMIT = 10
PARTICIPATES_LIMIT = 30
GETS_LIMIT = 20

DAILY_LIMIT_LIMIT = 75
PLACES_IN_WARD_LIMIT = 80
POSTAL_CODE_LOWER_LIMIT = 10000
POSTAL_CODE_UPPER_LIMIT = 99999
```

Plik models.py:

```
from neomodel import *


class User(StructuredNode):
    login = StringProperty()
    name = StringProperty()
    surname = StringProperty()
    password = StringProperty()

    medicalFacility = RelationshipTo('MedicalFacility', "WORKS")


class AdmEmployee(User):
    pass


class Doctor(User):
    pass


class Medic(User):
    pass


class Driver(User):
    pass


class Supervisor(User):
    medicalFacility = RelationshipTo('MedicalFacility', 'MANAGES')


class Administrator(User):
    supervisor = RelationshipTo('Supervisor', 'CREATES')


class Location(StructuredNode):
    adress = StringProperty()
    city = StringProperty()
    postalCode = StringProperty()


class OffersRel(StructuredRel):
    limit = IntegerProperty()


class MedicalFacility(StructuredNode):
    name = StringProperty()

    location = RelationshipTo('Location', 'IS_IN')
    hospitalWard = RelationshipTo('HospitalWard', 'OWNS')
```

```
    vehicle = RelationshipTo('Vehicle', 'HAS')
    medicalExam = RelationshipTo('MedicalExam', 'OFFERS', model=OffersRel)


class IsOccupiedRel(StructuredRel):
    admissionDate = StringProperty()
    dischargeDate = StringProperty()


class HospitalWard(StructuredNode):
    name = StringProperty()
    places = IntegerProperty()

    patient = RelationshipTo('Patient', 'IS_OCCUPIED', model=IsOccupiedRel)


class Vehicle(StructuredNode):
    brand = StringProperty()
    licensePlate = StringProperty()
    model = StringProperty()
    seats = IntegerProperty()

    medicalFacility = RelationshipTo('MedicalFacility', 'BELONGS_TO')


class Transfer(StructuredNode):
    startTime = StringProperty()
    startDate = StringProperty()
    status = StringProperty()
    valid = StringProperty()

    driver = RelationshipTo('Driver', 'EXECUTED_BY')
    vehicle = RelationshipTo('Vehicle', 'BOOKS')
    historyTransfer = RelationshipFrom('Transfer', 'PREVIOUS')
    medicalFacilityFrom = RelationshipTo('MedicalFacility', 'IS_FROM')
    medicalFacilityTo = RelationshipTo('MedicalFacility', 'IS_TO')
    medic = RelationshipTo('Medic', 'OVERSEEN_BY')


class OccupiesPlaceRel(StructuredRel):
    admissionDate = StringProperty()
    dischargeDate = StringProperty()


class ParticipatesRel(StructuredRel):
    status = StringProperty()


class Patient(StructuredNode):
    name = StringProperty()
    surname = StringProperty()
```

```
          hospitalWard   =   RelationshipTo('HospitalWard',   'OCCUPIES_PLACE',
model=OccupiesPlaceRel)
                transfer    =    RelationshipTo('Transfer',    'PARTICIPATES',
model=ParticipatesRel)


class Referral(StructuredNode):
    bodyPart = StringProperty()
    referralDate = StringProperty()

    patient = RelationshipFrom('Patient', 'GETS')
    doctor = RelationshipTo('Doctor', 'ISSUED_BY')
    medicalExam = RelationshipTo('MedicalExam', 'CONCERNS')


class IsReservedRel(StructuredRel):
    startDate = StringProperty()
    startTime = StringProperty()


class MedicalExam(StructuredNode):
    name = StringProperty()

    patient = RelationshipTo('Patient', "IS_RESERVED", model=IsReservedRel)
```

Plik generate.py:

```
from faker import Faker
from random import *

from constants import *
from models import *

fake = Faker()

# users
users = []
adm_employees = []
doctors = []
medics = []
drivers = []
supervisors = []
administrators = []

# other
vehicles = []
locations = []
medical_facilities = []
hospital_wards = []
patients = []
transfers = []
medical_exams = []
referrals = []
```

```
# methods used to check if value is unique

def get_logins():
    all_users = User.nodes.all()
    return [single_user.login for single_user in all_users]


def get_license_plates():
    all_vehicles = Vehicle.nodes.all()
    return [single_vehicle.license_plate for single_vehicle in all_vehicles]


# method to get all users used when generating relationships

def get_users():
    return User.nodes.all()


# nodes

def generate_adm_employees():
    logins = get_logins()
    for i in range(0, ADM_EMPLOYEE_LIMIT):
        login = fake.user_name()
        while logins.count(login) != 0:
            login = fake.user_name()
        logins.append(login)
        new_user = AdmEmployee(
            login=login,
            password=fake.password(),
            name=fake.first_name(),
            surname=fake.last_name())
        new_user.save()
        adm_employees.append(new_user)


def generate_doctors():
    logins = get_logins()
    for i in range(0, DOCTORS_LIMIT):
        login = fake.user_name()
        while logins.count(login) != 0:
            login = fake.user_name()
        logins.append(login)
        new_user = Doctor(
            login=login,
            password=fake.password(),
            name=fake.first_name(),
            surname=fake.last_name())
        new_user.save()
        doctors.append(new_user)
```

```
def generate_medics():
    logins = get_logins()
    for i in range(0, MEDICS_LIMIT):
        login = fake.user_name()
        while logins.count(login) != 0:
            login = fake.user_name()
        logins.append(login)
        new_user = Medic(
            login=login,
            password=fake.password(),
            name=fake.first_name(),
            surname=fake.last_name())
        new_user.save()
        medics.append(new_user)


def generate_drivers():
    logins = get_logins()
    for i in range(0, DRIVERS_LIMIT):
        login = fake.user_name()
        while logins.count(login) != 0:
            login = fake.user_name()
        logins.append(login)
        new_user = Driver(
            login=login,
            password=fake.password(),
            name=fake.first_name(),
            surname=fake.last_name())
        new_user.save()
        drivers.append(new_user)


def generate_supervisors():
    logins = get_logins()
    for i in range(0, SUPERVISORS_LIMIT):
        login = fake.user_name()
        while logins.count(login) != 0:
            login = fake.user_name()
        logins.append(login)
        new_user = Supervisor(
            login=login,
            password=fake.password(),
            name=fake.first_name(),
            surname=fake.last_name())
        new_user.save()
        supervisors.append(new_user)


def generate_administrators():
    logins = get_logins()
    for i in range(0, ADMINISTRATORS_LIMIT):
        login = fake.user_name()
```

```
        while logins.count(login) != 0:
            login = fake.user_name()
        logins.append(login)
        new_user = Administrator(
            login=login,
            password=fake.password(),
            name=fake.first_name(),
            surname=fake.last_name())
        new_user.save()
        administrators.append(new_user)


def generate_locations():
    for i in range(0, LOCATIONS_LIMIT):
        location = Location(
            city=fake.city(),
            address=fake.street_address(),
                                    postalCode=randint(POSTAL_CODE_LOWER_LIMIT,
POSTAL_CODE_UPPER_LIMIT)
        )
        location.save()
        locations.append(location)


def generate_medical_facilities():
    for i in range(0, MEDICAL_FACILITIES_LIMIT):
        medical_facility = MedicalFacility(
            name=fake.pystr()
        )
        medical_facility.save()
        medical_facilities.append(medical_facility)


def generate_hospital_wards():
    for i in range(0, HOSPITAL_WARDS_LIMIT):
        hospital_ward = choice(HOSPITAL_WARDS)
        hospital_ward = HospitalWard(
            name=hospital_ward,
            places=randint(0, PLACES_IN_WARD_LIMIT)
        )
        hospital_ward.save()
        hospital_wards.append(hospital_ward)


def generate_vehicles():
    license_plates = get_license_plates()
    for i in range(0, VEHICLES_LIMIT):
        license_plate = fake.license_plate()
        while license_plates.count(license_plate) != 0:
            license_plate = fake.license_plate()
        license_plates.append(license_plate)
        seats = int(choice(SEATS))
        vehicle = Vehicle(
```

```
            brand=fake.pystr(2, 20),
            model=fake.pystr(3, 20),
            licensePlate=license_plate,
            seats=seats
        )
        vehicle.save()
        vehicles.append(vehicle)


def generate_transfers():
    for i in range(0, TRANSFERS_LIMIT):
        start_date = fake.date_between(start_date=START_DATE, end_date='+1y')
        transfer = Transfer(
            startTime=fake.time(),
            startDate=start_date,
            status=choice(TRANSFER_STATUS),
                                valid=fake.date_between(start_date=START_DATE,
end_date=start_date)
        )
        transfer.save()
        transfers.append(transfer)


def generate_patients():
    for i in range(0, PATIENTS_LIMIT):
        patient = Patient(
            name=fake.first_name(),
            surname=fake.last_name()
        )
        patient.save()
        patients.append(patient)


def generate_referrals():
    for i in range(0, REFERRALS_LIMIT):
        body_parts = None
        if fake.pybool():
            body_parts = choice(BODY_PARTS)
        referral = Referral(
            referralDate=fake.date_between(START_DATE, END_DATE),
            bodyPart=body_parts
        )
        referral.save()
        referrals.append(referral)


def generate_medical_exams():
    for i in range(0, MEDICAL_EXAMS_LIMIT):
        medical_exam = MedicalExam(
            name=fake.pystr(5, 45),
        )
        medical_exam.save()
        medical_exams.append(medical_exam)
```

```
# relationships

def generate_executed_by():
    for transfer in transfers:
        driver = choice(drivers)
        transfer.driver.connect(driver)


def generate_books():
    for transfer in transfers:
        vehicle = choice(vehicles)
        transfer.vehicle.connect(vehicle)


def generate_is_from():
    for transfer in transfers:
        medical_facility = choice(medical_facilities)
        transfer.medicalFacilityFrom.connect(medical_facility)


def generate_is_to():
    for transfer in transfers:
        medical_facility = choice(medical_facilities)
        transfer.medicalFacilityTo.connect(medical_facility)


def generate_overseen_by():
    for transfer in transfers:
        medics_quantity = randint(0, MEDICS_IN_TRANSFER_LIMIT)
        for x in range(medics_quantity):
            medic = choice(medics)
            transfer.medic.connect(medic)


def generate_works():
    for user in get_users():
        medical_facility = choice(medical_facilities)
        user.medicalFacility.connect(medical_facility)


def generate_manages():
    for supervisor in supervisors:
        medical_facility = choice(medical_facilities)
        supervisor.medicalFacility.connect(medical_facility)


def generate_creates():
    for administrator in administrators:
        supervisor = choice(supervisors)
        administrator.supervisor.connect(supervisor)
```

```
def generate_owns():
    for medical_facility in medical_facilities:
        wards_quantity = randint(0, WARDS_IN_FACILITY_LIMIT)
        for x in range(wards_quantity):
            hospital_ward = choice(hospital_wards)
            medical_facility.hospitalWard.connect(hospital_ward)


def generate_offers():
    for medical_facility in medical_facilities:
        medical_exam = choice(medical_exams)
                medical_facility.medicalExam.connect(medical_exam,  {'limit':
randint(0, DAILY_LIMIT_LIMIT)})
    for i in range(OFFERS_LIMIT - MEDICAL_FACILITIES_LIMIT):
        medical_exam = choice(medical_exams)
        medical_facility = choice(medical_facilities)
                medical_facility.medicalExam.connect(medical_exam,  {'limit':
randint(0, DAILY_LIMIT_LIMIT)})


def generate_is_occupied_occupies_place():
    for patient in patients:
                admission_date  =  fake.date_between(start_date=START_DATE,
end_date=END_DATE)
                discharge_date  =  fake.date_between(start_date=START_DATE,
end_date='+1y')
        hospital_ward = choice(hospital_wards)
        if choice([True, False]):
                patient.hospitalWard.connect(hospital_ward, {'admissionDate':
admission_date, 'dischargeDate': discharge_date})
                    hospital_ward.patient.connect(patient,  {'admissionDate':
admission_date, 'dischargeDate': discharge_date})
        else:
                patient.hospitalWard.connect(hospital_ward, {'admissionDate':
admission_date})
                    hospital_ward.patient.connect(patient,  {'admissionDate':
admission_date})

    for i in range(IS_OCCUPIED_LIMIT - PATIENTS_LIMIT):
        patient = choice(patients)
                admission_date  =  fake.date_between(start_date=START_DATE,
end_date=END_DATE)
                discharge_date  =  fake.date_between(start_date=START_DATE,
end_date='+1y')
        hospital_ward = choice(hospital_wards)
        if choice([True, False]):
                patient.hospitalWard.connect(hospital_ward, {'admissionDate':
admission_date, 'dischargeDate': discharge_date})
                    hospital_ward.patient.connect(patient,  {'admissionDate':
admission_date, 'dischargeDate': discharge_date})
        else:
```

```
                patient.hospitalWard.connect(hospital_ward, {'admissionDate':
admission_date})
                    hospital_ward.patient.connect(patient, {'admissionDate':
admission_date})


def generate_belongs_to_has():
    for vehicle in vehicles:
        medical_facility = choice(medical_facilities)
        vehicle.medicalFacility.connect(medical_facility)
        medical_facility.vehicle.connect(vehicle)


def generate_participates():
    for patient in patients:
        transfer_quantity = randint(0, PARTICIPATES_LIMIT)
        for x in range(transfer_quantity):
            transfer = choice(transfers)
            patient.transfer.connect(transfer, {'status': choice(STATUS)})


def generate_gets():
    for referral in referrals:
        patients_quantity = randint(0, GETS_LIMIT)
        for x in range(patients_quantity):
            patient = choice(patients)
            referral.patient.connect(patient)


def generate_issued_by():
    for referral in referrals:
        doctor = choice(doctors)
        referral.doctor.connect(doctor)


def generate_concerns():
    for referral in referrals:
        medical_exam = choice(medical_exams)
        referral.medicalExam.connect(medical_exam)


def generate_is_reserved():
    for patient in patients:
                    start_date   =   fake.date_between(start_date=START_DATE,
end_date=END_DATE)
        start_time = fake.time()
        medical_exam = choice(medical_exams)
             medical_exam.patient.connect(patient, {"startDate": start_date,
"startTime": start_time})


def generate_is_in():
    for medical_facility in medical_facilities:
```

```
        location = choice(locations)
        medical_facility.location.connect(location)
```

Plik main.py:

```
from generate import *
from faker import Faker

login = 'neo4j'
password = 'iXkWvm40MmK1bSVHKUPcK9QZsQuOgwNZCu44h1n4jWc'

config.DATABASE_URL = f'bolt://{login}:{password}@localhost:7687'

fake = Faker()


def generate_all_silent():
    # nodes
    generate_adm_employees()
    generate_drivers()
    generate_doctors()
    generate_medics()
    generate_supervisors()
    generate_locations()
    generate_medical_facilities()
    generate_hospital_wards()
    generate_vehicles()
    generate_transfers()
    generate_patients()
    generate_referrals()
    generate_medical_exams()
    # relationships
    generate_executed_by()
    generate_books()
    generate_is_from()
    generate_is_to()
    generate_overseen_by()
    generate_works()
    generate_manages()
    generate_creates()
    generate_owns()
    generate_belongs_to_has()
    generate_offers()
    generate_is_occupied_occupies_place()
    generate_participates()
    generate_gets()
    generate_issued_by()
    generate_concerns()
    generate_is_reserved()


def generate_all_verbose():
    # nodes
```

```
generate_adm_employees()
print("Generating Administrative Employees finished")
generate_drivers()
print("Generating Drivers finished")
generate_doctors()
print("Generating Doctors finished")
generate_medics()
print("Generating Medics finished")
generate_supervisors()
print("Generating Supervisors finished")
generate_locations()
print("Generating Locations finished")
generate_medical_facilities()
print("Generating Medical Facilities finished")
generate_hospital_wards()
print("Generating Hospital Wards finished")
generate_vehicles()
print("Generating Vehicles finished")
generate_transfers()
print("Generating Transfers finished")
generate_patients()
print("Generating Patients finished")
generate_referrals()
print("Generating Referrals finished")
generate_medical_exams()
print("Generating Medical Exams finished")
# relationships
generate_executed_by()
print("Generating relationship EXECUTED BY finished")
generate_books()
print("Generating relationship BOOKS finished")
generate_is_from()
print("Generating relationship IS FROM finished")
generate_is_to()
print("Generating relationship IS TO finished")
generate_overseen_by()
print("Generating relationship OVERSEEN BY finished")
generate_works()
print("Generating relationship WORKS finished")
generate_manages()
print("Generating relationship MANAGES finished")
generate_creates()
print("Generating relationship CREATES finished")
generate_owns()
print("Generating relationship OWNS finished")
generate_belongs_to_has()
print("Generating relationships BELONGS TO and HAS finished")
generate_offers()
print("Generating relationship OFFERS finished")
generate_is_occupied_occupies_place()
print("Generating relationships IS OCCUPIED and OCCUPIES PLACE finished")
generate_participates()
print("Generating relationship PARTICIPATES finished")
```

```
    generate_gets()
    print("Generating relationship GETS finished")
    generate_issued_by()
    print("Generating relationship ISSUED BY finished")
    generate_concerns()
    print("Generating relationship CONCERNS finished")
    generate_is_reserved()
    print("Generating relationship IS RESERVED finished")
    print()
    print("-------------------------Database generated :)")


if __name__ == '__main__':
    generate_all_verbose()
    # generate_all_silent()
```

Etap 13.

Kwerendy

1. Wyszukiwanie użytkowników (kierowców i opiekunów medycznych) oraz pacjentów jadących danym pojazdem w danym czasie:

```
MATCH (vehicle {licensePlate: '5QY K25'}), (transfer:Transfer)
WHERE transfer.startDate="2021-06-17" AND (transfer)-->(vehicle)
CALL {
            with transfer
            MATCH (transfer)--(user:User)
            RETURN user.name AS name, user.surname AS surname
      UNION
            with transfer
            MATCH (patient:Patient)-[k:PARTICIPATES]->(transfer)
            RETURN patient.name AS name, patient.surname AS surname
}
RETURN name AS imie, surname AS nazwisko
```

2. Lista placówek oferujących dane badanie w danym mieście:

```
MATCH (m: MedicalFacility)-[:IS_IN]->(l: Location), (m:
MedicalFacility)-[:OFFERS]->(e: MedicalExam)
WHERE l.city = "Lake Timothy"
AND e.name = "sbfPMjhivSWZBFjQBNPSRF"
RETURN m.name AS nazwa
```

3. Liczba wolnych i zajętych miejsc na poszczególnych oddziałach w danej placówce:

```
MATCH (m: MedicalFacility)-[:OWNS]->(h: HospitalWard)-[o:IS_OCCUPIED]->(),
(m)-[:IS_IN]->(l: Location)
WHERE NOT EXISTS(o.dischargeDate) AND l.city = "Lake Timothy"
RETURN DISTINCT h.name AS oddzial, h.places AS miejsca,
(h.places-toInteger(count(o))) AS wolne
```

4. Liczba skierowań wypisanych na dane badanie:

```
MATCH (r: Referral)-[:CONCERNS]->(e: MedicalExam)
WHERE e.name = "sbfPMjhivSWZBFjQBNPSRF"
RETURN count(e)as liczba
```

Ze względu na brak obsługi dat przez Neo4j pominięte zostało wyszukiwanie w konkretnym okresie.

5. Lista pacjentów najczęściej korzystających z przejazdów (dziesięciu pacjentów):

```
MATCH (p:Patient)
CALL {
  WITH p
  MATCH (p)-[t:PARTICIPATES]->()
  RETURN count(t) AS przejazdy
}
RETURN p.name AS imie, p.surname AS nazwisko, przejazdy
ORDER BY przejazdy DESC
LIMIT 10
```

6. Lista pracowników zatrudnionych w danej placówce:

```
MATCH (u: User)-[:WORKS]->(f: MedicalFacility)
WHERE f.name = "DgPvVOXAvYOihFgXiMKH"
RETURN u.name AS imie, u.surname AS nazwisko
```

Ze względu na brak obsługi dat przez Neo4j pominięte zostało wyliczanie stażu pracowników.

7. Średnia liczba przejazdów kierowców:

```
MATCH (d: Driver)<-[:EXECUTED_BY]-(t: Transfer)
WITH toInteger(count(t)) AS transfers, d AS drivers
RETURN avg(transfers) AS srednia
```

8. Lista placówek z informacją dotyczącą liczby wykonanych z nich przejazdów w danym okresie:

```
MATCH (m: MedicalFacility)
OPTIONAL MATCH (t: Transfer)-[:IS_TO]->(m)
RETURN m.name AS nazwa, toInteger(count(t)) AS liczba
ORDER BY liczba DESC
LIMIT 100
```

9. Placówka z największym dziennym limitem badań na każde badanie:

```
MATCH (exam: MedicalExam)<-[offer:OFFERS]-(m: MedicalFacility)
RETURN exam.name AS badanie, m.name AS placowka, toInteger(max(offer.limit))
AS limit
```

10. Lista różnych oddziałów w danym mieście:

```
MATCH (l:Location {city:"Lake
Timothy"})<-[:IS_IN]-(m:MedicalFacility)-[:OWNS]->(ward: HospitalWard)
RETURN DISTINCT ward.name AS oddzial, m.name AS placowka
```

Wnioski dotyczące realizacji projektu

- Dla zbiorów danych, w których występuje bardzo dużo relacji pomiędzy obiektami – takim zbiorem danych jest właśnie zbiór danych dotyczących transportu medycznego i zarządzania nim – z podstawowych rodzajów baz NoSQL należy wybrać bazę grafową. Baza grafowa wspiera tworzenie relacji między obiektami jako krawędzie grafu. Duża część innych rodzajów baz NoSQL nie wspiera relacji między obiektami – np. w MongoDB można zrobić relacje między obiektami za pomocą linkowania dokumentów, ale jest to nieoptymalne przy dużej liczbie relacji. W takich sytuacjach bezpiecznym wyborem jest baza grafowa.
- Zespołowi ciężko było przyzwyczaić się do innego spojrzenia na bazy danych – niektórzy członkowie zespołu od wielu lat operują na bazach relacyjnych, a ten projekt był dla nich pierwszym zetknięciem się z bazami nierelacyjnymi. Nie sprawiło to większych problemów, ale faktem jest, że przestawienie myślenia z baz relacyjnych na bazy nierelacyjne wymaga trochę wysiłku. Istotny jest również odpowiedni dobór rodzaju nierelacyjnej bazy danych dla danego zbioru danych, co sprawiło zespołowi najwięcej problemów z powodu braku doświadczenia i pierwszej styczności z tematem.
- Co wiąże się z poprzednim wnioskiem, zespołowi ciężko było przyzwyczaić się do tworzenia obiektów w Neo4j – brakowało definicji jakiegoś wzoru dla obiektów przed przejściem do tworzenia obiektu (czyli brakowało nam po prostu tabeli). Nakładanie ograniczeń na węzły jest również bardzo czasochłonne – nie wynikają one z wzoru dla węzłów, którego oczywiście nie ma, dlatego muszą być nakładane na każdy label. Powoduje to większą elastyczność danych, ale duża elastyczność danych nie była wymagana w projekcie z uwagi na charakter danych. Idąc za przyzwyczajeniem z relacyjnych baz danych planowano nałożyć ścisłe ograniczenia na dane w celu ograniczenia błędów wprowadzania użytkowników już na poziomie bazy danych. W przypadku wykorzystania bazy nierelacyjnej, jeżeli ograniczenia na strukturę węzłów istnieją, należy zdefiniować je na wyższym poziomie.
- Dzięki przechowywaniu bazy Neo4j w chmurze możliwe jest równoczesne operowanie na bazie danych przez wszystkich członków zespołu. Przy testowaniu skryptu generującego bazę danych wykorzystano lokalną bazę danych utworzoną na komputerze jednego z członków zespołu, ale w etapie 11 korzystano z bazy danych w chmurze, co okazało się bardzo wygodne, zwłaszcza przy pracy z jednoczesnym wykorzystaniem platformy Zoom.
- Zespołowi ciężko było przyzwyczaić się do nowego narzędzia. Członków zespołu irytował interfejs (np. że nie ma głównego widoku, gdzie widać wszystkie wykonywane kwerendy, tylko każda kwerenda generuje nowe okienko, a pozostałe przesuwają się w dół), brak możliwości zmiany nazwy bazy danych, ograniczona liczba kolorów, a także brak możliwości usunięcia Property Keys. Być może część z tych problemów wynika z wykorzystania darmowej wersji Neo4j.
- Za wyjątkiem tych bardziej kosmetycznych uwag, praca z Neo4j była dla zespołu bardzo przyjemna. Wykorzystywany język – Cypher – jest bardzo intuicyjny i szybko

można się go nauczyć. Dodatkowo, bardzo proste było podłączenie się do bazy danych z zewnętrznego skryptu w Pythonie. Wtyczek dla Neo4j jest do języka Python bardzo dużo (Neo4j jest oprogramowaniem open-source w swojej edycji społeczności i dostępne wtyczki do języka Python opracowane przez społeczność). Wykorzystano wtyczkę neomodel, która jest bardzo prosta w obsłudze i można szybko się jej nauczyć.

Porównanie projektu bazy relacyjnej z projektem bazy nierelacyjnej

- Generowanie tej samej liczby danych dla bazy relacyjnej i bazy nierelacyjnej zajmowało znacznie więcej czasu dla bazy nierelacyjnej. Dzieje się tak dlatego, że w bazie nierelacyjnej oprócz węzłów (które odpowiadają wierszom w tabelach w bazie relacyjnej) należy wygenerować również relacje dla każdego obiektu. Można określić konkretną liczbę generowanych relacji i dłuższy czas generowania przykładowych danych dla bazy nierelacyjnej może wynikać z faktu, że w przypadku bazy nierelacyjnej generowano prawdopodobnie więcej relacji pomiędzy obiektami
- Napisanie skryptu generującego dane do bazy danych było dla zespołu prostsze w przypadku bazy nierelacyjnej. Wykorzystano wtyczkę neomodel, której wykorzystanie jest bardzo intuicyjne. W przypadku generowania danych do bazy relacyjnej wielokrotnie pojawiały się błędy kompilacji, a także duże trudności sprawiło podłączenie się do bazy danych. Konfiguracja MySql również sprawiła pewne problemy. W przypadku generowania danych do bazy nierelacyjnej błędy wynikały głównie z literówek, jednoczesnej pracy kilku osób nad tą samą metodą czy prostych przeoczeń. Konfiguracja narzędzia Neo4j oraz podłączenie się do bazy danych nie sprawiły żadnych problemów.
- Tabele i relacje pomiędzy tabelami, a więc struktura relacyjnej bazy danych, jest bardziej czytelna. Wystarczy rzut okiem na diagram struktury fizycznej relacyjnej bazy danych aby wiedzieć wszystko o połączeniach pomiędzy tabelami oraz strukturze samych tabel i przechowywanych atrybutach. Węzły w grafowej bazie danych reprezentują pojedyncze obiekty (pojedyncze wiersze w relacyjnej bazie danych), a krawędzie łączą bardziej pojedyncze obiekty niż całe klasy obiektów. Bazy grafowe narzucają również limit maksymalnie wyświetlanych węzłów i relacji, co utrudnia pracę np. podczas pisania kwerend. Z tego względu schematy baz relacyjnych są bardziej czytelne (wystarczy na przykład porównać diagram struktury fizycznej relacyjnej bazy danych z wynikiem kwerendy pokazującej wszystkie elementy w bazie grafowej). Bazy grafowe pozwalają jednak na większą elastyczność przechowywanych danych oraz wygodniejsze przechowywanie informacji w relacjach jako atrybuty relacji. W projekcie duża elastyczność danych nie była wymagana, ale wprowadzenie atrybutów relacji pozwoliło ograniczyć liczbę tabel asocjacyjnych, dlatego bardzo wygodnie pracowało się na bazie grafowej.
- Jednym z największych problemów Neo4j, który zaobserwował zespół, jest jego brak wsparcia dla obsługi dat. Korzystanie z dat w sposób efektywny wymaga

odpowiedniego zaplanowania ich formatu i sposobu zapisu, czego zespół nie wiedział przy rozpoczynaniu prac. Daty w projekcie zostały dodane jako Stringi, co uniemożliwia jednak ich praktyczne wykorzystanie w kwerendach. Bardziej praktycznym byłoby zapisywanie w postaci liczbowej np. timestamp. Nie zostało to jednak uwzględnione podczas implementacji.

- Nie jest możliwe jednoznacznie powiedzieć, czy któraś z rodzajów baz danych – relacyjna czy nierelacyjna – jest bardziej odpowiednia dla projektowanego systemu. Obie technologie pozwoliły na zaprojektowanie poprawnie działającego systemu. Osobistym odczuciem zespołu jest, że lepiej pracowało się nam na bazie grafowej Neo4j ze względu na prostotę połączenia z bazą z Pythona oraz szatę graficzną samego narzędzia Neo4j. Z drugiej strony, pisanie kwerend do bazy grafowej sprawiło więcej problemów z powodu konieczności nauki nowego języka. Oprócz tego, jak wspomniano wyżej, nie wykorzystywane są w pełni możliwości bazy grafowej, ponieważ zbiór danych, na których operuje system, nie spełnia modelu 3V. Pewną trudnością było również przestawienie myślenia z dobrze znanego podejścia baz relacyjnych na nowe dla zespołu podejście baz nierelacyjnych. Podsumowując, wygodniej pracowało się z Neo4j, ale obie technologie pozwoliły na zaprojektowanie poprawnie funkcjonującej bazy danych i żaden członek zespołu nie ma odczucia, aby baza relacyjna była lepsza od bazy nierelacyjnej czy odwrotnie.