1 Projekt 4 – Neo4j

Cały kod utworzony przy realizacji zadania dostępny jest w serwisie GitHub pod adresem https://github.com/mycielski/zsbd/tree/main/projekt_4.

1.1 Instalacja

Wykorzystałem oficjalny obraz Neo4j z Dockerhub, który uruchomiłem tą komendą:

```
$ docker run \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/plugins:/plugins \
    --memory=8g \
    --env NEO4J_db_memory_transaction_total_max=4G \
    -e NEO4J_apoc_export_file_enabled=true \
    -e NEO4J_apoc_import_file_enabled=true \
    -e NEO4J_apoc_import_file_use__neo4j__config=true \
    -e NEO4J_PLUGINS='["apoc"]' \
    neo4j
```

Odbyło się to bez żadnych problemów.

1.2 Zbiory danych

Do realizacji zadania wykorzystałem zbiory danych ze strony openflights.org:

- airports
- airlines
- routes
- planes
- countries

Mam zamiar wykorzystać zbiory `airports` i `routes` do stworzenia grafu lotnisk i połączeń między nimi.

Typy węzłów utworzone z tych zbiorów:

- Airline
- City
- Country
- PlaneType
- Terminal
- Timezone

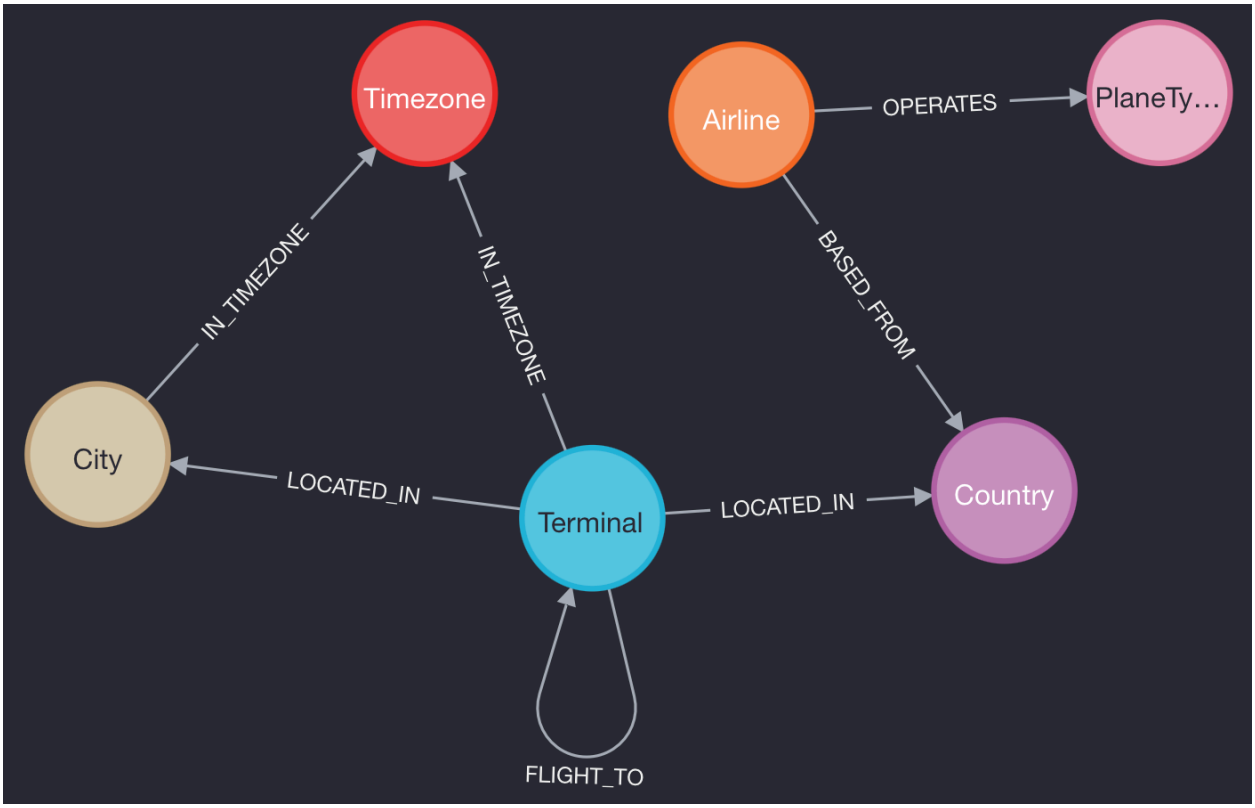
Łącznie 11351 węzłów.

Typy relacji pomiędzy zbiorami:

- BASED_FROM
  - W jakim kraju dana linia lotnicza ma swoją siedzibę.
- IN_TIMEZONE
  - Z której strefy czasowej dane miasto/lotnisko.
- LOCATED_IN
  - Gdzie znajduje się dane miasto/lotnisko.
- OPERATES
  - Jakie samoloty obsługuje dana linia lotnicza.
- FLIGHT_TO
  - Połączenia lotnicze istniejące między lotniskami.

Łącznie 23486 relacji.

1.2.1 Schemat danych



Cały kod wykorzystany do załadowania danych i utworzenia relacji dostępny jest w serwisie GitHub pod adresem https://github.com/mycielski/zsbd/blob/main/projekt_4/import.ipynb.

1.3 Przykład tworzenia węzłów i połączeń

```
with driver.session() as session:
    for _, row in tqdm(terminals_df.iterrows(), total=len(terminals_df)):
        session.run(
            "CREATE (t:Terminal {airport_id: $airport_id, name: $name, iata: $iata, icao: $icao, latitu
            airport_id=row["airport_id"],
            name=row["name"],
            iata=row["iata"],
            icao=row["icao"],
            latitude=row["latitude"],
            longitude=row["longitude"],
            altitude=row["altitude"],
            timezone=row["timezone"],
            type=row["type"],
            source=row["source"],
        )
        # create timezone node
        session.run("MERGE (tz:Timezone {name: $name})", name=row["timezone"])
        # create city node
        session.run("MERGE (c:City {name: $city})", city=row["city"])
```

```
# match city with timezone
session.run(
    """
    MATCH (c:City {name: $city}),
          (tz:Timezone {name: $timezone})
    MERGE (c)-[:IN_TIMEZONE]->(tz)
    """,
    city=row["city"],
    timezone=row["timezone"],
)
# match terminal with timezone
session.run(
    """
    MATCH (t:Terminal {iata: $iata}),
          (tz:Timezone {name: $timezone})
    MERGE (t)-[:IN_TIMEZONE]->(tz)
    """,
    iata=row["iata"],
    timezone=row["timezone"],
)
# match terminal with city
session.run(
    """
    MATCH (t:Terminal {iata: $iata}),
          (c:City {name: $city})
    MERGE (t)-[:LOCATED_IN]->(c)
    """,
    iata=row["iata"],
    city=row["city"],
)
# match terminal with country
session.run(
    """
    MATCH (t:Terminal {iata: $iata}),
          (c:Country {name: $country})
    MERGE (t)-[:LOCATED_IN]->(c)
    """,
    iata=row["iata"],
    country=row["country"],
)
# match city with country
session.run(
    """
    MATCH (c:City {name: $city}),
          (c:Country {name: $country})
    MERGE (c)-[:LOCATED_IN]->(c)
    """,
    city=row["city"],
    country=row["country"],
)
```

1.4 Zapytania

1. Które popularne lotniska (minimum 30 połączeń) nie są ze sobą połączone?

```
MATCH (t1:Terminal)-[r:FLIGHT_TO]->(t2:Terminal)
WITH t1, count(r) as routes_count
WHERE routes_count > 30
WITH collect(t1) as major_airports
UNWIND major_airports as airport1
UNWIND major_airports as airport2
WITH airport1, airport2
WHERE airport1 <> airport2
AND NOT (airport1)-[:FLIGHT_TO]->(airport2)
RETURN
    airport1.name as Airport1,
    airport1.iata as IATA1,
    airport2.name as Airport2,
    airport2.iata as IATA2,
    point.distance(airport1.location, airport2.location)/1000 as Distance_KM
ORDER BY Distance_KM
LIMIT 10
```

Airport1	IATA1	Airport2	IATA2	Distance_KM
"Narita International Airport"	"NRT"	"OR Tambo International Airport"	"JNB"	13598.58493382201
"OR Tambo International Airport"	"JNB"	"Narita International Airport"	"NRT"	13598.58493382201

To zapytanie sugeruje, że między tymi dwoma lotniskami warto otworzyć nowe połączenie.

2. Które państwa są najlepiej skomunikowane, czyli które państwa mają najwięcej połączeń międzynarodowych?

```
MATCH (c1:Country)<-[:LOCATED_IN]-(t1:Terminal)-[r:FLIGHT_TO]->(t2:Terminal)-[:LOCATED_IN]->(c2:Country)
WHERE c1 <> c2
WITH c1.name as country, count(DISTINCT c2) as connections
RETURN country, connections
ORDER BY connections DESC
LIMIT 3
```

country	connections
"Switzerland"	28
"United Arab Emirates"	26
"Belgium"	23

3. Które linie lotnicze mają najbardziej zróżnicowaną flotę samolotów?

```
MATCH (a:Airline)-[op:OPERATES]->(p:PlaneType)
WITH
    a.name as airline_name,
    a.iata as airline_code,
    COUNT(DISTINCT p) as plane_type_count,
    COLLECT(DISTINCT p.name) as fleet
RETURN
```

```
    airline_name as Airline,
    airline_code as IATA,
    plane_type_count as Number_of_Plane_Types,
    fleet as Fleet_List
ORDER BY plane_type_count DESC
LIMIT 5
```

Airline	IATA	Number_of_Plane_Types	Fleet_List
"All Nippon Airways"	"NH"	21	["McDonnell Douglas MD-90", "Boeing 777-300ER" ing 767-300", "Boeing 737-700", "Airbus A321", avilland Canada DHC-8-200 Dash 8 / 8Q", "Airbu -400", "Boeing 777-200LR", "Boeing 787-8", "Ai 800", "Airbus A320", "Boeing 777-300", "Airbus -600", "Boeing 737-400", "Canadair Regional Je "]
"Scandinavian Airlines System"	"SK"	17	["Embraer 170", "Airbus A320", "Boeing 717", " 8-300 Dash 8", "Boeing 767-300", "Boeing 777-2 Jet 200", "Canadair Regional Jet 900", "De Hav Dash 8Q", "Boeing 737-800", "Aerospatiale/Alen /B", "Saab 2000", "Boeing 757-200", "British A 330-300", "Boeing 787-8"]
"Qantas"	"QF"	15	["Boeing 767-300", "Boeing 737-800", "Boeing 7 "Airbus A319", "Airbus A330-200", "De Havilla 8", "Airbus A380-800", "Embraer 170", "Airbus ", "De Havilland Canada DHC-8-400 Dash 8Q", "F "Boeing 777-300ER"]
"Japan Airlines"	"JL"	15	["Airbus A340-300", "Boeing 767-300", "Boeing 0", "Boeing 777-300", "Boeing 747-400", "Airbu 0-300", "Boeing 737-900", "Airbus A319", "Airb "Embraer 190", "Boeing 737-400", "Embraer 170
"South African Airways"	"SA"	14	["Airbus A319", "Boeing 737-800", "Boeing 747- "De Havilland Canada DHC-8-400 Dash 8Q", "Cana "British Aerospace Jetstream 41", "Canadair Re 777-300ER", "Avro RJ85", "Embraer RJ135", "Em ng 777-200LR", "Boeing 767-300"]

4. Lista lotów z podziałem na krajowe i międzynarodowe dla danego państwa.

W moim przykładzie dla Norwegii. **To zapytanie wykorzystuje UNION**.

```
// Domestic
MATCH (t1:Terminal)-[r:FLIGHT_TO]->(t2:Terminal),
      (t1)-[:LOCATED_IN]->(c:Country),
      (t2)-[:LOCATED_IN]->(c)
WHERE c.name = "Norway"
RETURN
t1.iata as From,
t2.iata as To,
'Domestic' as Flight_Type
```

```
UNION

// International
MATCH (t1:Terminal)-[r:FLIGHT_TO]->(t2:Terminal),
      (t1)-[:LOCATED_IN]->(c1:Country),
      (t2)-[:LOCATED_IN]->(c2:Country)
WHERE c1 <> c2 AND c1.name = "Norway"
RETURN
t1.iata as From,
t2.iata as To,
'International' as Flight_Type
```

From	To	Flight_Type
"TRD"	"TRF"	"Domestic"
"TRF"	"SVG"	"Domestic"
"BGO"	"FRA"	"International"
"BGO"	"ABZ"	"International"
"KRS"	"CPH"	"International"
"OSL"	"TLL"	"International"
"OSL"	"SVO"	"International"
"OSL"	"RIX"	"International"
"OSL"	"HEL"	"International"
"OSL"	"GOT"	"International"
"HAU"	"CPH"	"International"
"TRF"	"CPH"	"International"
"TRD"	"TLL"	"International"
"SVG"	"NCL"	"International"
"SVG"	"FRA"	"International"
"SVG"	"ARN"	"International"

5. Połączenie `MERGE` miasta ze strefą czasową.

```
MATCH (c:City {name: $city}),
      (tz:Timezone {name: $timezone})
MERGE (c)-[:IN_TIMEZONE]->(tz)
```

1.5 Funkcje przestrzenne

W moim zbiorze danych terminals zawarte są informacje o współrzędnych geograficznych lotniska. Dzięki temu można wykorzystać funkcje przestrzenne dostępne w Neo4j.

Wykorzystałem je do obliczania długości lotu między dwoma lotniskami w relacji `FLIGHT_TO`:

```
with driver.session() as session:
    for _, row in tqdm(routes_df.iterrows(), total=len(routes_df)):
        if row["stops"] > 1:
            continue
        session.run(
            """
            MATCH (source:Terminal {iata: $source_airport}),
                  (destination:Terminal {iata: $destination_airport}),
                  (airline:Airline {iata: $airline})
            MERGE (source)-[:FLIGHT_TO {iata: $airline, distance: point.distance(source.location, desti
            """,
            source_airport=row["source_airport"],
            destination_airport=row["destination_airport"],
            airline=row["airline"],
            equipment=row["equipment"],
        )
```

1.6 Zapytanie z agregacją `reduce()` i najkrótszą ścieżką

Ile kilometrów musimy przelecieć aby dostać się z lotniska Longreach do lotniska Perth International?

```
MATCH p = SHORTEST 1 (a:Terminal)-[:FLIGHT_TO*]->(b:Terminal)
WHERE a.iata = "LRE" AND b.iata = "PER"
RETURN reduce(total = 0, r IN relationships(p) | total + r.distance)/1000 AS result_km
```

```
result_km

22035.34801551281
```

Domyślnie Neo4j używa szybkiego dwukierunkowego algorytmu BFS ($O(V + E)$) do znalezienia najkrótszej ścieżki1, są natomiast dostępne inne algorytmy, takie jak Dijkstra czy A*.

Istotną zaletą wykorzystania danych przestrzennych jest możliwość wykorzystania funkcji Neo4j do obliczania odległości między węzłami. Nie musimy implementować logiki samodzielnie, wystarczy wywołać wbudowaną funkcję `point.distance`.

1.7 Indeksy

Neo4j wspiera dwa typy indeksów – Search-performance i Semantic2. Zdecydowałem się na wykorzystanie indeksów search-performance aby usprawnić wyszukiwanie węzłów na podstawie ich atrybutów.

1.7.1 Które popularne lotniska (minimum 30 połączeń) nie są ze sobą połączone?

Utworzone indeksy:

```
with driver.session() as session:
    session.run("""
    CREATE INDEX terminal_details IF NOT EXISTS
    FOR (t:Terminal)
```

```
    ON (t.name, t.iata, t.location)
    """)
    session.run("""
    CREATE INDEX flight_source IF NOT EXISTS
    FOR ()-[r:FLIGHT_TO]->()
    ON (r.source)
    """)
    session.run("""
    CREATE INDEX flight_destination IF NOT EXISTS
    FOR ()-[r:FLIGHT_TO]->()
    ON (r.destination)
    """)
```

- Przed stworzeniem indeksów: 0.003 sekundy na zapytanie
- Po stworzeniu indeksów: 0.0018 sekundy na zapytanie

1.7.2 Które linie lotnicze mają najbardziej zróżnicowane floty?

Utworzone indeksy:

```
with driver.session() as session:
    session.run("""
CREATE INDEX airline_details IF NOT EXISTS
FOR (a:Airline)
ON (a.name, a.iata)
""")
    session.run("""
CREATE INDEX plane_type_name IF NOT EXISTS
FOR (p:PlaneType)
ON (p.name)
""")
```

- Przed stworzeniem indeksów: 0.005 sekundy na zapytanie
- Po stworzeniu indeksów: 0.002 sekundy na zapytanie

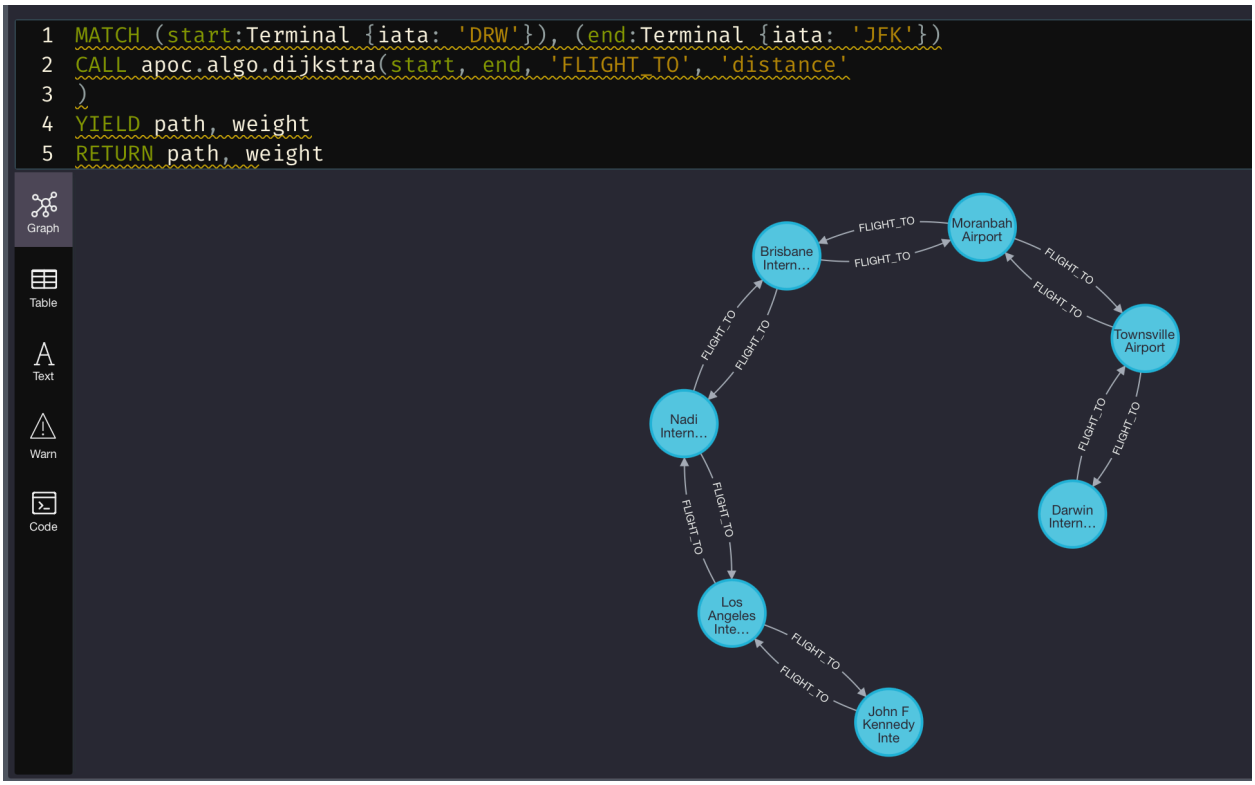
Kod wykorzystany do benchmarkowania zapytań dostępny jest w serwisie GitHub pod adresem https://github.com/mycielski/zsbd/blob/main/projekt_4/bench.ipynb.

Utworzone indeksy są typu range. Indeks jest kopią danych z bazy, wraz ze ścieżką dostępu do oryginalnych danych3.

Na moich zbiorach danych warto byłoby stworzyć także indeksy na kodach IATA i ICAO, ponieważ te często są wykorzystywane do jednoznacznej identyfikacji samolotów, linii i lotnisk.

1.8 APOC

Wykorzystałem algorytm Dijkstry dostępny w APOC do znalezienia najkrótszej ścieżki między dwoma lotniskami.



1.9 Stored procedure

Aby zademnostrować wykorzystanie stored procedure w Neo4j, stworzyłem prostą procedurę, która zwraca listę linii lotniczych.

```
package org.example;

import org.neo4j.graphdb.Label;
import org.neo4j.graphdb.Transaction;
import org.neo4j.procedure.Context;
import org.neo4j.procedure.Description;
import org.neo4j.procedure.Procedure;

import java.util.stream.Stream;

public class GetAirlines {
    @Context
    public Transaction txn;

    public static class JourneyResult {
        public String alias;

        public JourneyResult(String alias) {
            this.alias = alias;
        }

    }

    @Procedure(name="org.example.getAirlines")
```

```
    @Description("Returns airlines from the database")
    public Stream<JourneyResult> getAirlines() {
        return txn.findNodes(Label.label("Airline")).stream()
                .map(node -> new JourneyResult((String) node.getProperty("alias")));
    }

}
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>stored_proc</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.neo4j.driver</groupId>
            <artifactId>neo4j-java-driver</artifactId>
            <version>5.26.0</version>
        </dependency>

        <dependency>
            <groupId>org.neo4j</groupId>
            <artifactId>neo4j</artifactId>
            <version>5.26.0</version>
        </dependency>

        <dependency>
            <groupId>org.neo4j</groupId>
            <artifactId>procedure-compiler</artifactId>
            <version>5.26.0</version>
        </dependency>

    </dependencies>

    <build>
        <pluginManagement>
            <plugins>

                <plugin>
                    <artifactId>maven-shade-plugin</artifactId>
                    <version>3.2.2</version>
                    <executions>
```

```
                        <execution>
                            <phase>package</phase>
                            <goals>
                                <goal>shade</goal>
                            </goals>
                        </execution>
                    </executions>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>

</project>
```

Można ją wywołać w ten sposób:

```
CALL org.example.getAirlines;
```

```
 alias

 "ANA All Nippon Airways"

 "Air Asia"

 "Pulkovo Aviation Enterprise"

 "bmi British Midland"

 "SN Brussels Airlines"

 "Contactair"

 "CSA Czech Airlines"

 "Emirates Airlines"

 "EasyJet Airline"

 "FlyAsianXpress"

 "TACA"

 "Horizon Airlines"

 "JAL Japan Airlines"

 "Pacific Airlines"

 "PIA Pakistan International"

 "Qantas Airways"

 "SAA South African Airways"
```

"SAS Scandinavian Airlines"

"Sibir Airlines"

"SkyWork"

"Swiss Airlines"

"TAP Air Portugal"

"Thai Air Asia"

"Turkmenhovayollary"

"TWA"

"Varig"

"SkyExpress"

"now Jetairlfy"

"Braathens SAFE"

"Avialinii 400"

"Epic Holidays"

"Dennis Sky Holding"

"WEA"

"BRAZIL AIR"

"Kreta Sky"

"SOCHI"

"Tom\\'s air"

"slowbird"

"lionXpress"

"Domenican"

"Russian. Yours Air Lines "

"Baikal Airlines"

"MARYSYA AIRLINES"

"ZABAIKAL "

"Fly Brasil"

"Himalaya"

"Indya1"

"Air Canada Express"

"Moskva-air"

"Air luch"

"Mongol Air "

"NEXT"

"Sovet Air "

"USky"

"Marusya Air"

"RussianConector"

"Indus Airlines Pak"

"Samurai Airlines (DUMMY)"

"AirOne Continental"

"AirOne Polska"

"Javi"

"Zenith"

"Orbit Azerbaijan"

"Royal Inc."

"CheapFlying"

"Royal Southern"

"Sochi Air "

"Tramm Airlines"

"Maryland"

"Apache"

"Jettor"

```
"XPTO  "

"Air Lituanica"

"Rainbow Air (RAI)"

"Rainbow Air CAN"

"Rainbow Air POL"

"Rainbow Air EU"

"Rainbow Air US"

"    "

"Aero Spike"

"ADLER EXPRESS"

"Encore"

"All America BOPY"

"Russian Commuter "
```

Najważniejsza różnica między procedurami w bazach SQL a Neo4j, to że w Neo4j piszemy je w Javie, a nie w języku używanym do tworzenia zapytań (SQL lub Cypher). Ponadto, po napisaniu procedury dla Neo4j należy ją skompilować do pliku `jar` a następnie umieścić w katalogu `plugins`. Neo4j musi wtedy zostać zrestartowany, aby zobaczył nową procedurę i mógł z niej skorzystać. Zastosowania procedur w Neo4j są nieskończone, ogranicza nas jedynie nasza wyobraźnia i umiejętności programowania w Javie.

1.10 Końcowa analiza zbioru danych

W produkcyjnej bazie przechowującej informacje o lotach warto byłoby rozłożyć bazę na kilka maszyn. Na jednej maszynie wówczas znalazłyby się na przykład loty dotyczące jednej linii lotniczej.