Zaawansowane Systemy Baz Danych – Etap 4 „Neo”

Michał Ankiersztajn 311171

Spis treści

1. Zbiór Danych

Wybieram ten sam zbiór danych co w MongoDB, czyli informacje na temat okazjonalnych cen gier wraz z metadanymi tych gier ze strony https://apidocs.cheapshark.com dodatkowo powiążę te dane z Steam Store Games (Clean dataset) ponieważ posiada on dużo danych na temat gier razem z steam appid, które posiadam również w danych z cheapshark.

Jest to moja pierwsza styczność z bazą grafową, więc wolę pracować na czymś, co już ‘znam’. Może to nie być najoptymalniejszy wybór pod względem bazy grafowej, jednak będzie on bardzo dobry dla porównania baz JSONowych i grafowych.

2. Konfiguracja i import

Szczegóły na temat konfiguracji znajdują się w pliku **docker-compose.yml**

```yaml
services:
  neo4j:
    container_name: neo4j
    image: neo4j:latest
    ports:
      - 7474:7474
      - 7687:7687
    environment:
      - NEO4J_AUTH=neo4j/Admin123!
      - NEO4J_apoc_export_file_enabled=true
      - NEO4J_apoc_import_file_enabled=true
      - NEO4J_apoc_import_file_use__neo4j__config=true
      - NEO4J_PLUGINS=["apoc", "graph-data-science", "apoc-extended]
    volumes:
      - ./neo4j_db/data:/data
      - ./neo4j_db/logs:/logs
      - ./neo4j_db/import:/var/lib/neo4j/import
      - ./neo4j_db/plugins:/plugins
```

Przetworzenie danych:

Początkowe dane nie pozwalały na stworzenie 5 krawędzi i 5 relacji, więc je dodatkowo podzieliłem. Skrypty załączone wspomogłem się przy ich pisaniu ChatemGPT: https://chatgpt.com/share/674acd27-e344-8011-a5ea-efa8412a4de6

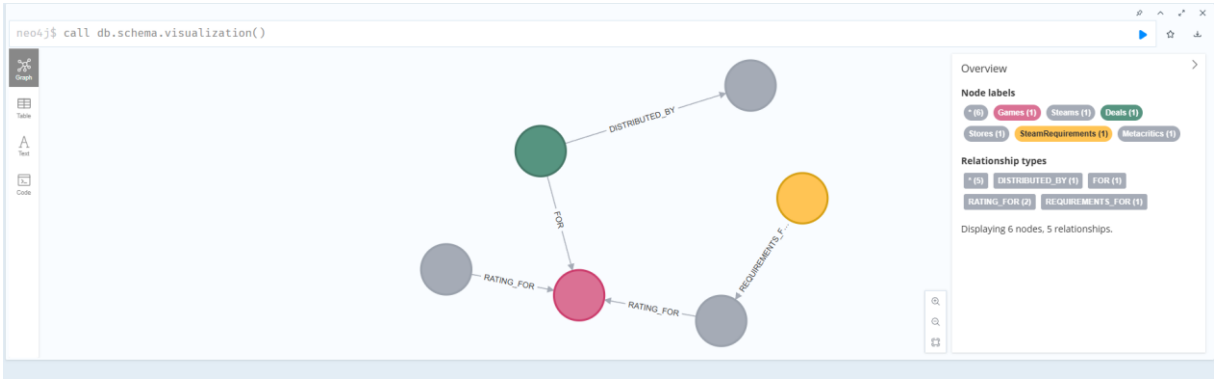
Importowałem zarówno CSV:

```
LOAD CSV WITH HEADERS FROM "file:///steam_requirements_data.csv" AS row
CREATE (n: SteamRequirements {id: row.steam_appid})
set n+= row
RETURN n;
```

Jak i poprzez JSONa:

```
CALL apoc.load.json("file:///games.json")
YIELD value
CREATE (n: Games {id: value.gameID})
set n+= value
RETURN n;
```

Finalna schema (wyeksportowana jako **records.json**), zawarłem w tym punkcie już dodanie kolejnego zbioru danych.



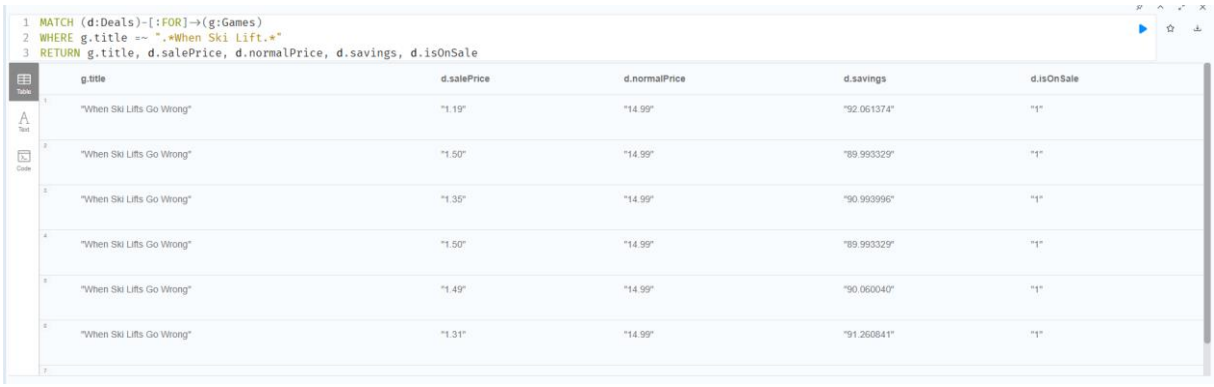
3. Drugi zbiór danych

Drugi zbiór danych jest w postaci CSV i został dodany razem z poprzednim krokiem.

Skorzystam ze zbioru, który został zaproponowany przez Macieja Michalskiego, pasuje on do mojego ponieważ posiadam już dane na temat Steama, jak i gier, a ten dataset ma rozszerzenie danych na temat gier względem tego, co znajduje się w cheapsharku.

Zapytania:

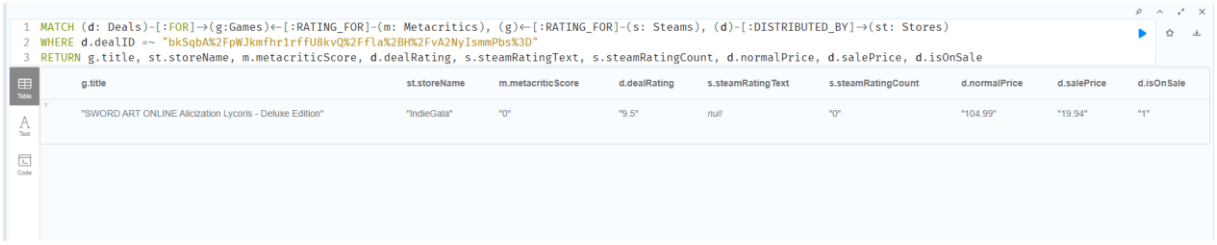
1. Proste wyszukiwanie deali po nazwie gry:



```
MATCH (d:Deals)-[:FOR]→(g:Games)
WHERE g.title =~ ".*When Ski Lift.*"
RETURN g.title, d.salePrice, d.normalPrice, d.savings, d.isOnSale
```

	g.title	d.salePrice	d.normalPrice	d.savings	d.isOnSale
1	"When Ski Lifts Go Wrong"	"1.19"	"14.99"	"92.061374"	"1"
2	"When Ski Lifts Go Wrong"	"1.50"	"14.99"	"89.993329"	"1"
3	"When Ski Lifts Go Wrong"	"1.35"	"14.99"	"90.993996"	"1"
4	"When Ski Lifts Go Wrong"	"1.50"	"14.99"	"89.993329"	"1"
5	"When Ski Lifts Go Wrong"	"1.49"	"14.99"	"90.060040"	"1"
6	"When Ski Lifts Go Wrong"	"1.31"	"14.99"	"91.260841"	"1"

2. Wyświetlanie szczegółów danego deala:



```
MATCH (d: Deals)-[:FOR]→(g:Games)←[:RATING_FOR]-(m: Metacritics), (g)←[:RATING_FOR]-(s: Steams), (d)-[:DISTRIBUTED_BY]→(st: Stores)
WHERE d.dealID =~ "bkSqbA%2FpWJkmfhr1rffU8kvQ%2Ffla%2BH%2FvA2NyIsmmPbs%3D"
RETURN g.title, st.storeName, m.metacriticScore, d.dealRating, s.steamRatingText, s.steamRatingCount, d.normalPrice, d.salePrice, d.isOnSale
```

	g.title	st.storeName	m.metacriticScore	d.dealRating	s.steamRatingText	s.steamRatingCount	d.normalPrice	d.salePrice	d.isOnSale
1	"SWORD ART ONLINE Alicization Lycoris - Deluxe Edition"	"IndieGala"	"0"	"9.5"	null	"0"	"104.99"	"19.94"	"1"

3. Wyświetlenie gier, które są poniżej danej ceny i mają wyższą ocenę niż podana (za pomocą UNION):

```
MATCH (g:Games)←[:RATING_FOR]-(m: Metacritics)
WHERE m.metacriticScore > "90"
RETURN g.title
UNION DISTINCT
MATCH (d: Deals)-[:FOR]→(g:Games)
WHERE d.salePrice < "1.45" AND d.isOnSale = "1"
RETURN g.title
```

	g.title
1	"Disco Elysium - The Final Cut"
2	"System Shock 2"
3	"BioShock Infinite"
4	"Red Dead Redemption 2"
5	"Bzzzt"
6	"Tom Clancy's Splinter Cell Chaos Theory"

4. Wyświetlenie wszystkich dostępnych szczegółów na temat gry:

```
MATCH (g:Games)←[:RATING_FOR]-(m: Metacritics), (g)←[:RATING_FOR]-(s: Steams), (s)←[:REQUIREMENTS_FOR]-(sr:SteamRequirements)
WHERE g.title =~ ".*When Ski Lift.*"
RETURN DISTINCT g.title, m.metacriticScore, s.steamRatingPercent, s.steamRatingText, s.steamRatingCount, sr.minimum, sr.recommended
```

	g.title	m.metacriticScore	s.steamRatingPercent	s.steamRatingText	s.steamRatingCount	sr.minimum
1	"When Ski Lifts Go Wrong"	"80"	"87"	"Very Positive"	"490"	"Requires a 64-bit processor and operating system OS: Windows 7 64bit Processor: 2 GHz Memory: 2 GB RAM Graphics: 1GB Storage: 500 MB a

5. Zapytanie wyświetlające podobne gry (za pomocą MERGE):

```
MATCH (g1:Games {gameID: "194665"})←[:RATING_FOR]-(s1: Steams), (g2: Games)←[:RATING_FOR]-(s2: Steams{steamRatingPercent: s1.steamRatingPercent})
MERGE (g2)-[:SIMILAR_TO]→(g1)
RETURN g1.title AS MainGame, g2.title AS SimilarGames
```

	MainGame	SimilarGames
1	"When Ski Lifts Go Wrong"	"Wreckfest"
2	"When Ski Lifts Go Wrong"	"Cyberdimension Neptunia: 4 Goddesses Online"
3	"When Ski Lifts Go Wrong"	"Underrail"
4	"When Ski Lifts Go Wrong"	"Verdun"
5	"When Ski Lifts Go Wrong"	"Tropico 6"
6	"When Ski Lifts Go Wrong"	"Valfaris"

Zauważyłem ogromny minus neo4j – interfejs graficzny potrafi mocno spowolnić, gdy nie zamyka się komórek z wynikami zapytań.

4. Indeksy

W Neo indeksy są zaimplementowane jako kopie sprecyzowanych danych prymitywnych, takich jak node, relationship, czy properties. Dane przechowywane w indeksie dostarczają ścieżkę do danych w bazie danych i pozwalają na szybsze filtrowanie i przeszukiwanie danych.

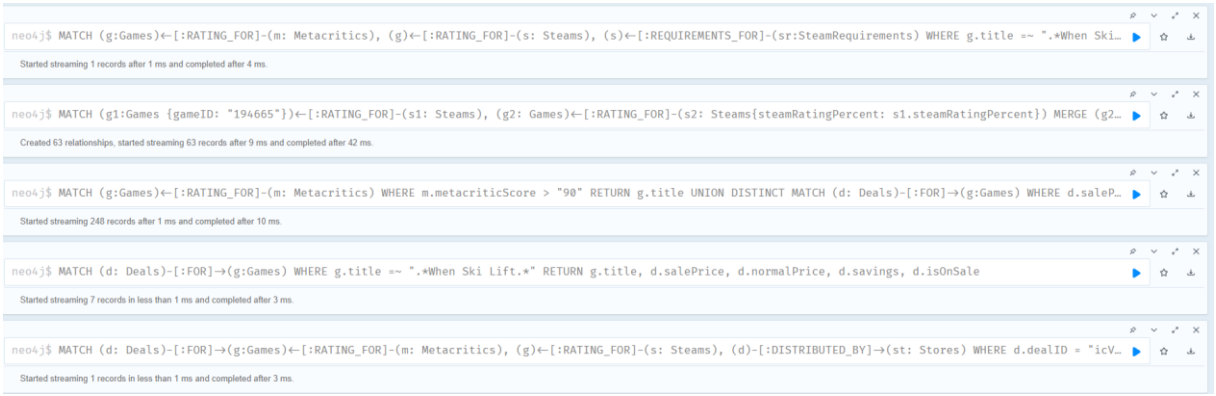
Skorzystam z TEXT INDEXu dla game.title, w tym celu zmienię składnię z =~ „.*nazwa.*” na CONTAINS „nazwa”, aby mieć pewność, że indeksy te zostaną wykorzystane w zapytaniach.

Dalej stworzę indeksy dla metacriticScore i steamRatingPercent, bo są one wykorzystywane przy wyszukiwaniu po gier po ocenie, a także salePrice i isOnSale, które są kluczowe dla aplikacji typu porównywarka cen.

Stworzę też dla dealID i gameID ponieważ są one wykorzystywane w zapytaniach i nie rzadko w tego typu systemie.

Dodatkowo podstawowo baza Neo4J korzysta z Token Lookup Indexów, które są wykorzystywane do odszukiwania relacji pomiędzy węzłami.

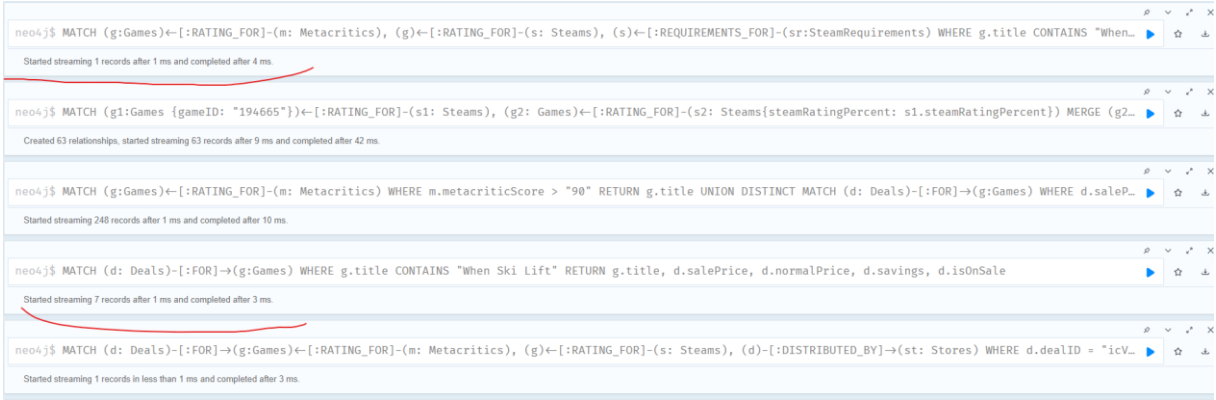
Najpierw sprawdzam czas bez indeksów:



Tworzę indeks tekstowy na tytuł gry:

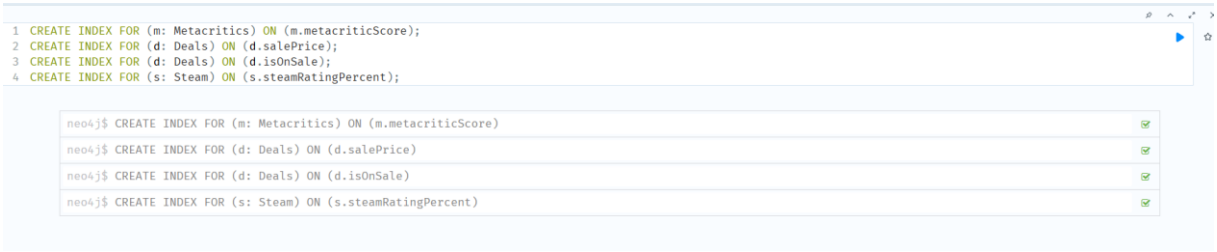
```
neo4j$ CREATE TEXT INDEX FOR (g:Games) ON (g.title)
```

Zapytania korzystające z indeksów:

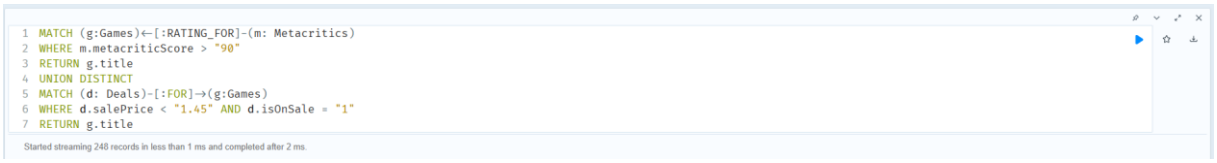


Bez zmian, prawdopodobnie w bazie nie ma wystarczająco dużo danych na temat gier (jest ich parę tyś.), aby zauważyć większą zmianę powinno być ich dużo więcej.

Tworzę indeksy na pricing i rating:

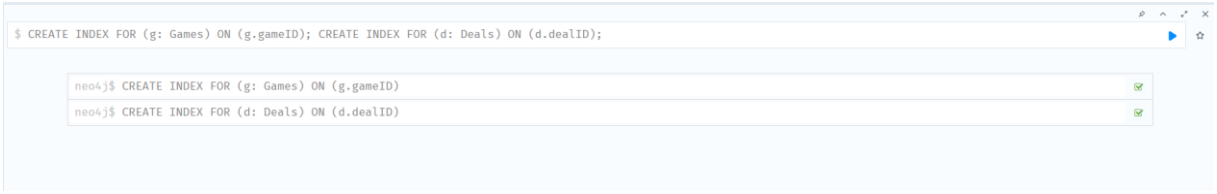


Zapytania korzystające z indeksów:

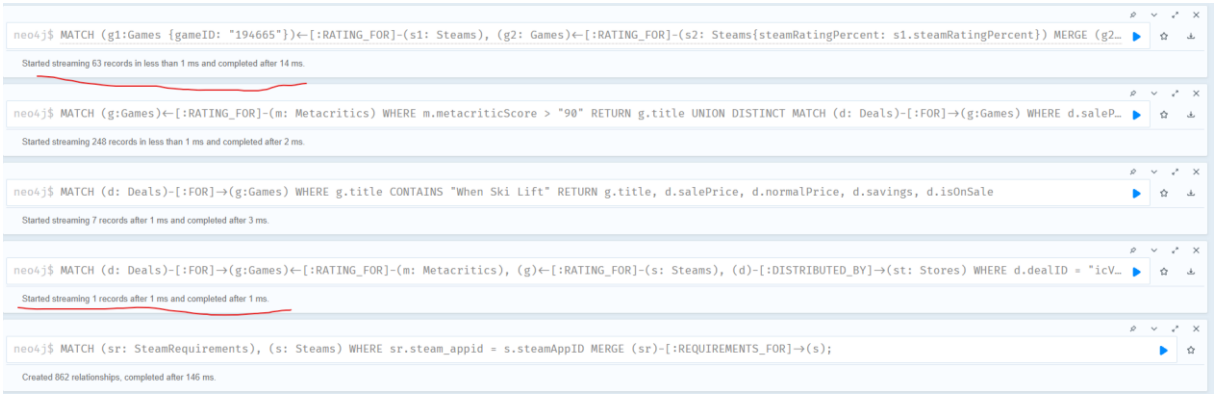


Ponieważ dane są wielokrotnie wykorzystywane dodatkowo z klauzulą UNION to indeksy mają znaczenie i przyspieszają działanie z 10 ms do 2ms, czyli aż o 500%! Zdecydowanie warto indeksować tego typu dane.

Indeksy na ID:

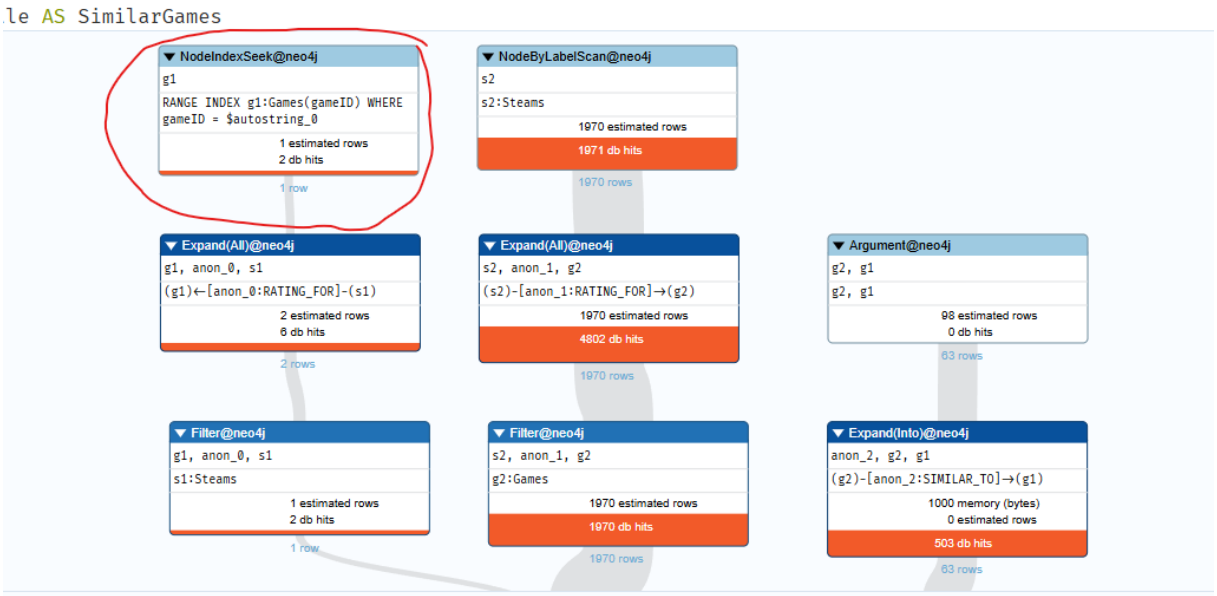


Zapytania korzystające z indeksów:

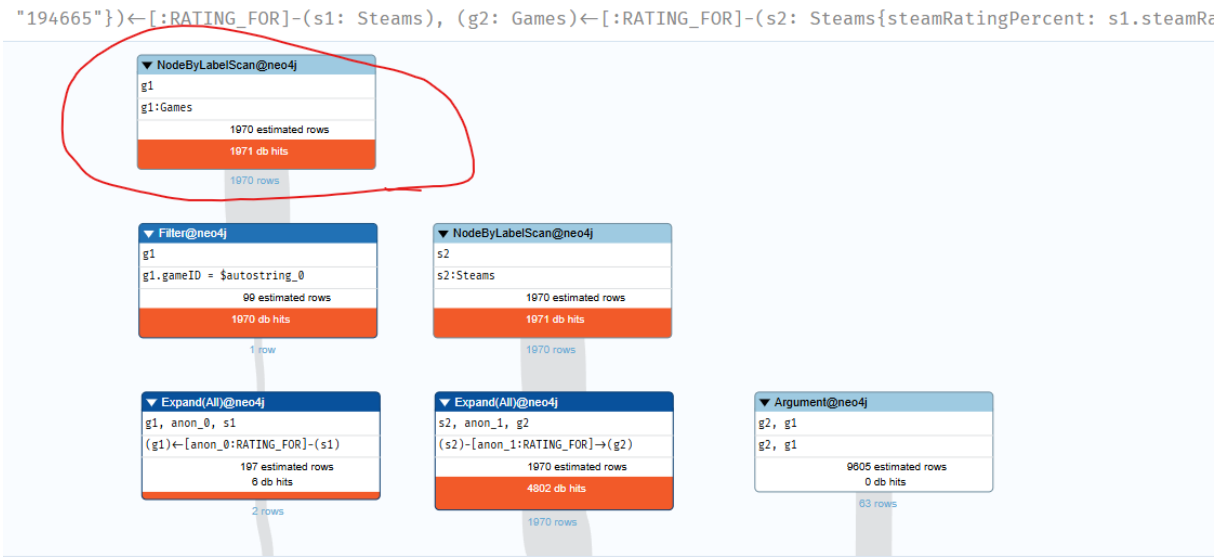


Zapytanie z gameID nie przyspieszyło, a wręcz zwolniło – prawdopodobnie korzystając z {} Neo nie wykorzystuje indeksów, ponieważ drugie zapytanie korzystające z dealID przez WHERE przyspieszyło o 200% z 3ms do 1ms.

Nie jestem pewny, czy to tak działa, a więc decyduję się to sprawdzić za pomocą PROFILE:



Node z game jest znajdywany od razu, czyli indeks powinien działać, sprawdźmy co się stanie, gdy usunę indeks:



Tym razem przeszukane zostało 1970 wierszy, czyli indeks pomógł i to znacznie, bo ograniczyliśmy liczbę wyszukiwani do 1, różnica czasu musiała wynikać z wolniejszego połączenia z bazą danych lub czasu procesów.

Narzędzie PROFILE jest niesamowicie wygodne i przydatne względem tego co jest w MS SQL, MongoDB, czy PostgreSQL. Prawdopodobnie wynika to z możliwości wglądu w planner query i graficznej reprezentacji.

Dodatkowo możemy optymalizować za pomocą:

1. Klauzul WHERE i LIMIT

2. Tworząc optymalne krawędzie, jeśli często potrzebujemy wziąć coś co jest dalej połączone w grafie, a nie jest bezpośrednią relacją między 2 węzłami to warto zastanowić się nad takim połączeniem, aby zoptymalizować liczbę przeszukań
3. Build-in cache – w Neo istnieje wbudowana pamięć podręczna wykorzystywana przy często wykonywanych zapytaniach.
4. Pluginów, jak np. apoc, którego wykorzystałem do ładowania JSONa, można też do załadowania csv z czego finalnie nie skorzystałem.

5. Dane przestrzenne

Dla mojego zestawu danych nie ma danych przestrzennych i bardzo ciężko je wprowadzić, więc korzystam z chat GPT do wygenerowania takich danych:

```
CREATE (warsaw:City {name: 'Warsaw', location: point({latitude: 52.2298, longitude: 21.0118})}),
(krakow:City {name: 'Kraków', location: point({latitude: 50.0647, longitude: 19.9450})}),
(lodz:City {name: 'Łódź', location: point({latitude: 51.7592, longitude: 19.4560})}),
(wroclaw:City {name: 'Wrocław', location: point({latitude: 51.1079, longitude: 17.0385})}),
(poznan:City {name: 'Poznań', location: point({latitude: 52.4084, longitude: 16.9342})}),
(gdansk:City {name: 'Gdańsk', location: point({latitude: 54.3520, longitude: 18.6466})}),
(szczecin:City {name: 'Szczecin', location: point({latitude: 53.4289, longitude: 14.5530})}),
(bydgoszcz:City {name: 'Bydgoszcz', location: point({latitude: 53.1235, longitude: 17.9860})}),
(lublin:City {name: 'Lublin', location: point({latitude: 51.2465, longitude: 22.5685})}),
(katowice:City {name: 'Katowice', location: point({latitude: 50.2649, longitude: 19.0238})}),
(gdynia:City {name: 'Gdynia', location: point({latitude: 54.5186, longitude: 18.5301})}),
(czestochowa:City {name: 'Częstochowa', location: point({latitude: 50.8118, longitude: 19.1242})}),
```

Table

Added 25 labels, created 25 nodes, set 50 properties, completed after 48 ms.

Będzie to mały dataset miast. Zacznijmy od sprofilowania najkrótszej ścieżki wyszukiwania miast, które znajdują się w pewnej odległości od pewnego punktu:

```
PROFILE
MATCH (c: City)
WHERE point.distance(c.location, point({longitude: 52.2298, latitude: 21.0118})) ⩽ 4289238.8604333345
RETURN c.name
```

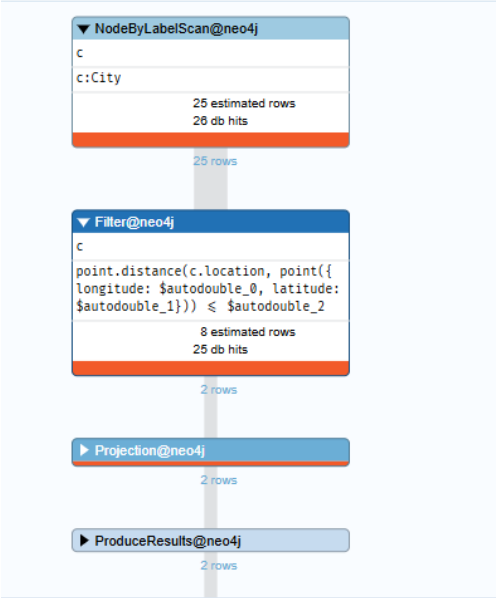
Table

	c.name
1	"Lublin"
2	"Rzeszów"

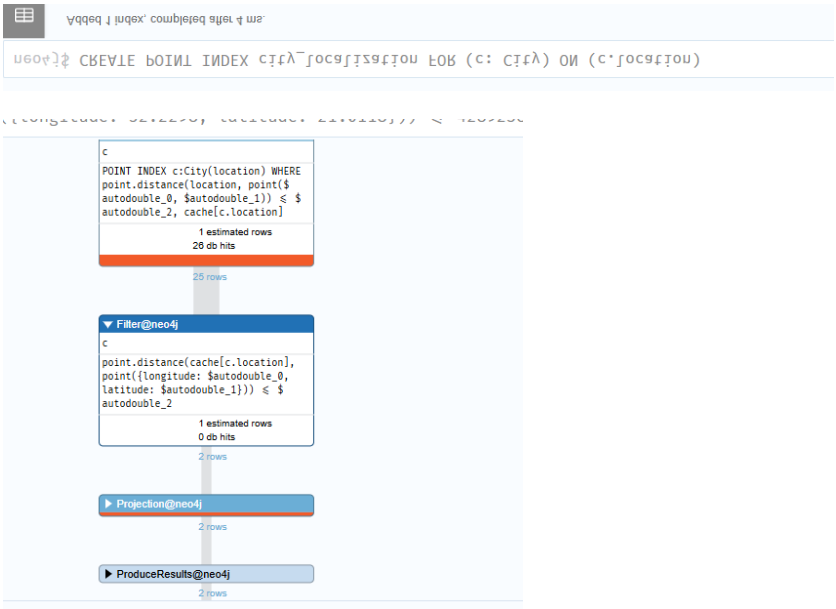
Text

Plan

Planner:



Po dodaniu indeksu:

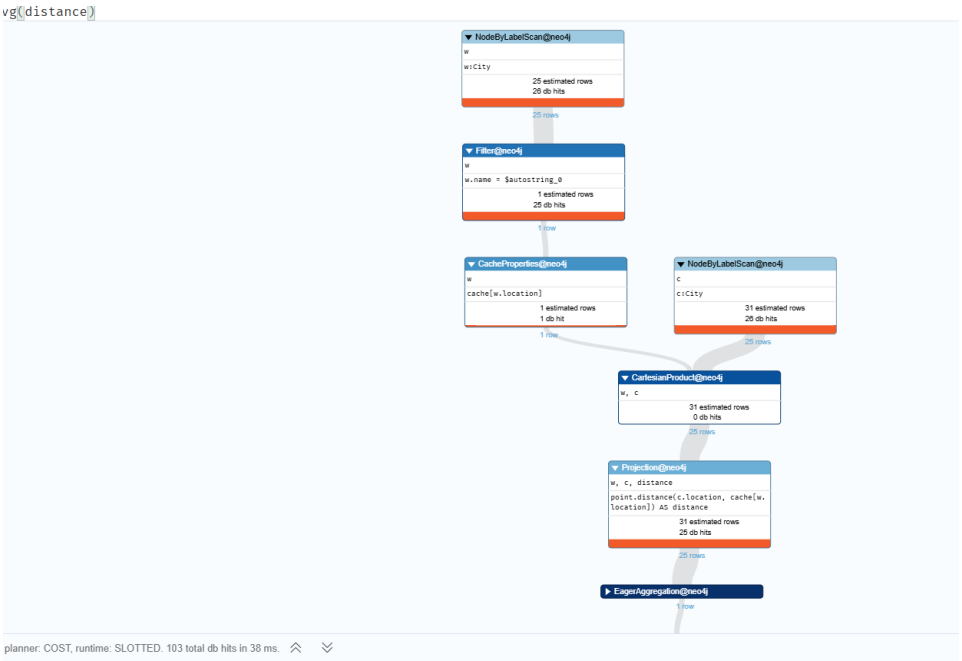


Niestety, w tym przypadku korzyść z indeksu jest zerowa i można powiedzieć, że tylko zaśmieca nam bazę. Sprawdźmy, czy sytuacja jest taka sama w przypadku agregacji:

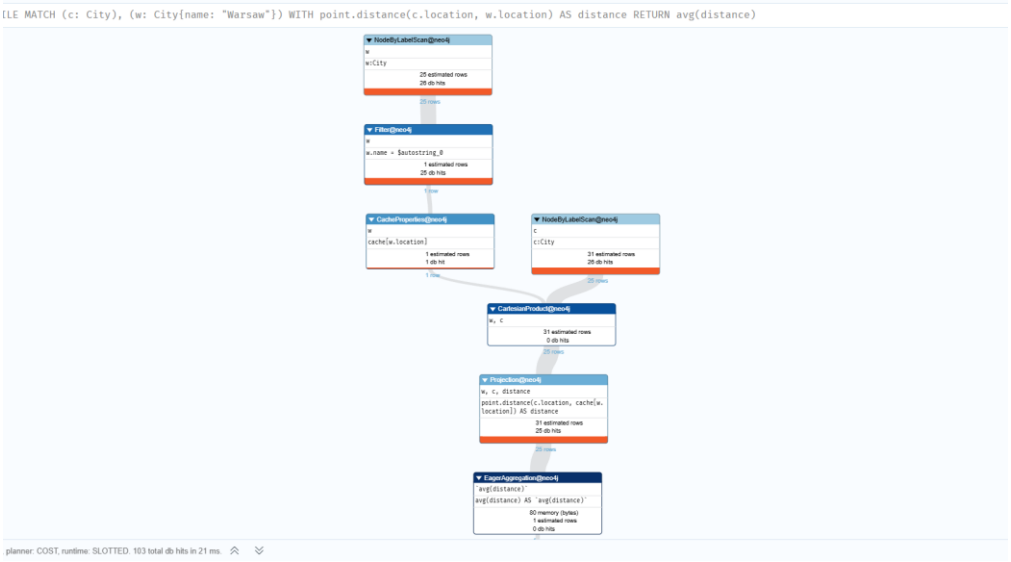
Sprawdzimy średnią odległość, latitude i longitude w bazie (nie ma to za bardzo sensu funkcjonalnie, ale powinno dobrze przetestować agregację):

```
1  PROFILE
2  MATCH (c: City), (w: City{name: "Warsaw"})
3  WITH point.distance(c.location, w.location) AS distance
4  RETURN avg(distance)
```

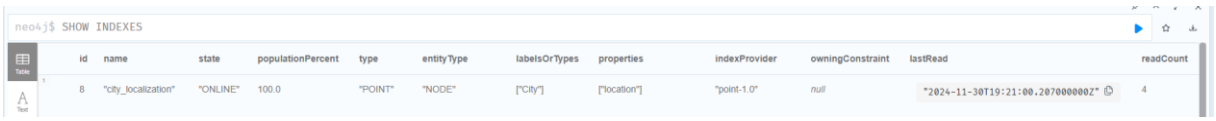
Po usunięciu indeksów odpalam zapytanie i patrzę w planer:



Dodaję indeksy i znowu:



Sytuacja jest identyczna jest tyle samo uderzeń do bazy, wygląda na to, że przy zapytaniu z agregacją indeks typu point nie jest wykorzystywany. Potwierdza się to przy przejrzeniu indeksów:



readCount nie zwiększa się pomimo uruchamiania zapytania z agregacją. Co oznacza, że dla tego typu zapytań indeksy geolokacji nie są wykorzystywane. Ma to sens, ponieważ i tak w tym wypadku chcemy iterować po wszystkich lokacjach miast.

Zalety takich funkcji to prostota, reużywalność i uwspólnione API dla developerów, zmieniając firmę nie zmieniają się funkcje.

Wadą jest to, że jesteśmy zmuszeni do przechowywania w ten sposób danych, aby móc z nich skorzystać, a są sposoby, aby zrobić to inaczej.

Dane geoprzestrzenne najczęściej wykorzystuje się w firmach w których znaczenie mają mapy (np. Google Maps, Apple Maps), trackowanie lokalizacji (np. Uber, Bolt), jak i rozwiązania IOT (np. włączamy ogrzewanie w domu, tylko gdy właściciel jest w promieniu 10km).

6. Procedura

Do stworzenia procedur skorzystam z https://github.com/neo4j-examples/neo4j-procedure-template/, tak abym mógł się zapoznać z strukturą i podjąć decyzję, jakiego typu procedurę chciałbym utworzyć.

Stwierdziłem, że stworzę procedurę, która pokaże nam wszystkie krawędzie powiązane z danym węzłem, jest to dużo szybsze niż szukanie wszystkich krawędzi w schemach:

GetAllRelationships.java

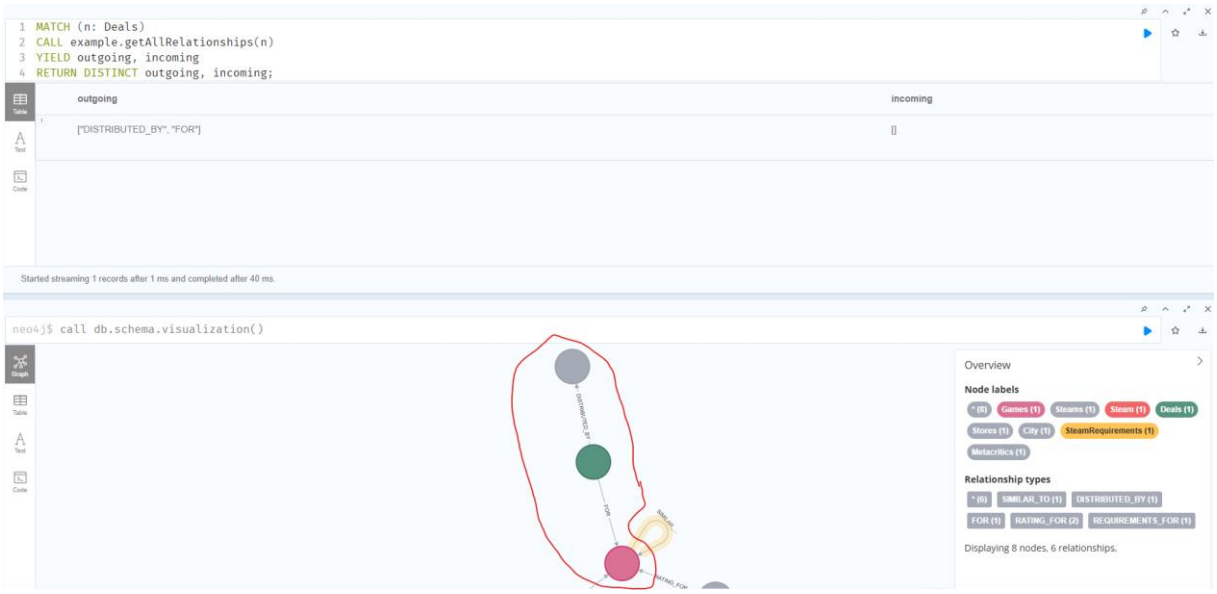
```java
package example;

import org.neo4j.graphdb.Direction;
import org.neo4j.graphdb.Node;
import org.neo4j.procedure.Description;
import org.neo4j.procedure.Name;
import org.neo4j.procedure.Procedure;

import java.util.*;
import java.util.stream.Stream;

public class GetAllRelationships {  no usages  new *
    @Procedure(name = "example.getAllRelationships")  no usages  new *
    @Description("Get the relationships going in and out of a node.")
    public Stream<Relationships> getAllRelationships(@Name("node") Node node) {
        Set<String> outgoing = new HashSet<>();
        node.getRelationships(Direction.OUTGOING).iterator().forEachRemaining(rel -> outgoing.add(rel.getType().name()));

        Set<String> incoming = new HashSet<>();
        node.getRelationships(Direction.INCOMING).iterator().forEachRemaining(rel -> incoming.add(rel.getType().name()));

        return Stream.of(new Relationships(incoming.stream().toList(), outgoing.stream().toList()));
    }
    public static class Relationships {  2 usages  new *
        public List<String> outgoing;  1 usage
        public List<String> incoming;  1 usage

        public Relationships(List<String> incoming, List<String> outgoing) {  1 usage  new *
            this.outgoing = outgoing;
            this.incoming = incoming;
        }
    }
}
```

Tworzę plik jar za pomocą skryptu:

```
PS C:\Users\Michal\IdeaProjects\neo4j-procedure-template> .\mvnw clean package
```

```
[INFO] --- shade:3.5.1:shade (default) @ procedure-template ---
[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing C:\Users\Michal\IdeaProjects\neo4j-procedure-template\target\procedure-template-1.0.0-SNAPSHOT.jar with C:\Users\Michal\IdeaProjects\neo4j-procedure-template\target\procedure-template-1.0.0-SNAPSHOT-shaded.jar
```

I wrzucam go do neo4j_db > plugins, a następnie uruchamiam:



Jak widać procedura działa poprawnie i znalazła relacje należące do Deals (zielona kropka).

Procedury w Neo różnią się od relacyjnych tym, że są pisane w zewnętrznym języku programowania. Daje nam to dużo szersze możliwości pisania bardziej skomplikowanych procedur, minus jest taki, że musimy znać używany do procedur język programowania i można korzystać z wielu języków na raz co znacznie podnosi skomplikowanie systemu plus dużo ciężej jest napisać i od razu korzystać z takiej procedury bo jest parę kroków po drodze. Dodatkowo, ponieważ korzystamy z języka programowania możemy po prostu napisać testy i mieć ‘pewność’, że wszystko jest tak jak powinno.

Bardzo łatwo dorzucić plugin z procedurami, ponieważ jest to prosty plik Jar, więc istnieje możliwość stworzenia util bibliotek, które można wykorzystywać w wielu projektach, jedną z takich bibliotek jest APOC. Z tego też względu procedury w Neo dużo częściej są ‘utilsami’, które pomagają w codziennej pracy.

7. Analiza końcowego zboru danych

Mój zbiór nie ma oczywistego rozłożenia na wielu maszynach fizycznych. To co przychodzi mi do głowy to rozproszenie bazy danych przez :

- Wydzielenie Steams z SteamRequirements do oddzielnego podgrafu SteamRelated, działało by to na oddzielnej maszynie
- Wydzielenie Deals i Stores do oddzielnego podgrafu Distribution
- Dodatkowo w przypadku bardzo dużego rozrośnięcia się bazy dodałbym oddzielna maszynę dla deali z ratingiem >9, ponieważ to te najlepsze deale są najczęściej wyświetlane i szukane, a cała reszta dużo rzadziej i można by to

podzielić na oddzielne maszyny, wtedy jednak dobrze, aby stores były synchronizowane na obydwu maszynach, a nie, aby znajdywały się na oddzielnej maszynie, bo będzie to często odpytywany węzeł przez obydwie maszyny.

Podsumowując:

- Mosty przegubowe to (Deals)-[:FOR]->(Games) oraz (Steams)-[:RATING_FOR]->(Games)
- Węzły przegubowe to Deals, Games oraz Steams.

8. APOC

Po zapoznaniu się z bibliotekom APOC skorzystałem z:

1. Importu danych JSONowych

Bardzo przydatna funkcja zwłaszcza, gdy migrujemy z JSONowej bazy danych jak MongoDB, chcemy skorzystać z danych z publicznego API lub po prostu mamy dane w takim formacie, wykorzystałem to już w rozdziale konfiguracja:

```
CALL apoc.load.json("file:///games.json")
YIELD value
CREATE (n: Games {id: value.gameID})
set n+= value
RETURN n;
```

2. Skorzystam z eksportu do JSONa, aby można było w prostu sposób odtworzyć i załadować moją bazę nie tylko w neo4j, ale i innych rozwiązaniach baz JSONowych:

Plik załączony **all.json**

3. Formatowanie liczb, nierzadko w zależności od kraju lub aplikacji jesteśmy zmuszeni do reprezentacji liczb w inny sposób:

```
MATCH (g:Games)←[:RATING_FOR]-(s:Steams)
RETURN g.title, apoc.number.format(toInteger(s.steamRatingCount), '#,##0;(#,##0)', 'it') as steamRatingCount
LIMIT 100;
```

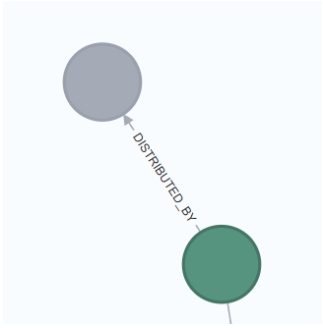
	g.title	steamRatingCount
1	"RoboCop: Rogue City"	"10.123"
2	"Port Royale"	"0"
3	"Cloudpunk"	"12.460"
4	"Master of Orion 3"	"150"
5	"Beat Hazard"	"4.980"
6	"Outshine"	"54"

Started streaming 100 records after 11 ms and completed after 16 ms.

W moim przypadku dobrze, aby steamRatingCount miał jakiś format, bo może być zarówno bardzo duży, jak i bardzo mały

4. Inwersja relacji – bardzo przydatne, gdy się pomylimy albo w trakcie zauważymy, że któryś kierunek ma większy sens:

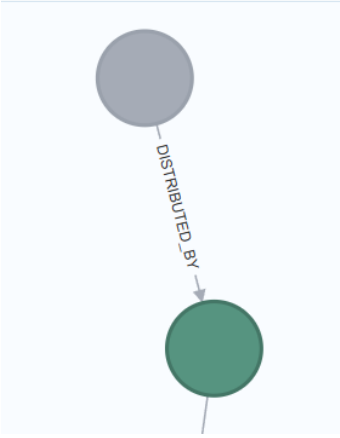
Przed:




```
neo4j$ MATCH (s:Stores)←[rel:DISTRIBUTED_BY]→(d: Deals) CALL apoc.refactor.invert(rel) yield input, output RETURN input, output
```

	input	output
1	7922	{ "identity": 10865, "start": 33229, "end": 30271, "type": "DISTRIBUTED_BY", "properties": { }

Po:



5. Teraz dla odwróconej relacji warto zmienić jej nazwę tak, aby była bardziej opisowa:

```
1 MATCH (s:Stores)-[rel:DISTRIBUTED_BY]→(d: Deals)
2 WITH collect(rel) AS rels
3 CALL apoc.refactor.rename.type("DISTRIBUTED_BY", "DISTRIBUTED_IN", rels)
4 YIELD committedOperations
5 RETURN committedOperations
```

committedOperations
3000

9. Dodatkowe wnioski

W trakcie tworzenia bazy zrozumiałem na czym polega moc baz grafowych i, że dobrany przeze mnie zbiór danych nie był idealny, ponieważ nie posiadałem przykładowo relacja sam-do-siebie, jak to może być w przypadku ludzi (np. Marian jest dzieckiem Ani, Ania jest dzieckiem ... itd.), jak i dwukierunkowości np. w filmie człowiek może być aktorem, jak i producentem i da się to oznaczyć relacjami.

Korzystałem z interfejsu w localhost:7474/browser, problem polegał na tym, że w komórkach nie działa crtl+z, a jest przydatne i przy rozwinięciu komórek z dłuższym wynikiem zapytania przeglądarka się mocno zacinała.

Procedury w neo4j dają dużo więcej możliwości niż w innych bazach i zaskakująco łatwo jest je uruchomić dzięki templatkom i skryptom napisanym przez innych do generowania jarów.

10. Bibliografia

https://medium.com/@matthewghannoum/simple-graph-database-setup-with-neo4j-and-docker-compose-061253593b5a

https://community.neo4j.com/t/create-a-relationship-between-two-already-existing-nodes-by-one-common-property/46914

https://stackoverflow.com/questions/35281066/neo4j-is-it-possible-to-visualise-a-simple-overview-of-my-database

https://stackoverflow.com/questions/37299077/neo4j-importing-local-csv-file

https://neo4j.com/labs/apoc/5/installation/#docker

https://neo4j.com/docs/cypher-manual/current/indexes/

https://neo4j.com/docs/

https://neo4j.com/docs/cypher-manual/current/functions/spatial/