



Sprawozdanie z projektu nr 4

Zaawansowane systemy baz danych

Wybór zbioru danych:

Do projektu wybrałem zbiór danych 100 amerykańskich restauracji z San Francisco z wystawionymi dla nich opiniami na portalu Trip Advisor. Wejściowy zbiór zawiera 100 restauracji, każda restauracji ma wystawionych 100 opinii. Obiekty json'owe są bardzo rozbudowane posiadają dane opisowe, harmonogram restauracji, dodatkowe charakterystyki jak np. dostępność parkingu, miejsc dla niepełnosprawnych etc., dane kontaktowe czy dane geolokalizacyjne. Recenzje są relatywnie prostymi obiektami posiadającymi tytuł, treść, ocenę i datę. Uważam ten zbiór za odpowiedni, ponieważ na samym starcie mamy 100 węzłów restauracji (a przy odpowiednim podziale logicznym i zaprojektowaniu architektury bazy, węzłów będzie znacznie więcej) oraz 14 700 węzłów opinii na samym początku. Co ważne zbiór już posiada dane geolokalizacyjne, co może być wykorzystane z użyciem dostępnych w Neo funkcji geolokalizacyjnych. Odpowiednia złożoność zbioru pozwala mi na wyodrębnienie kilku węzłów i krawędzi tak, by odpowiednio podzielić dane i zaprojektować bazę.

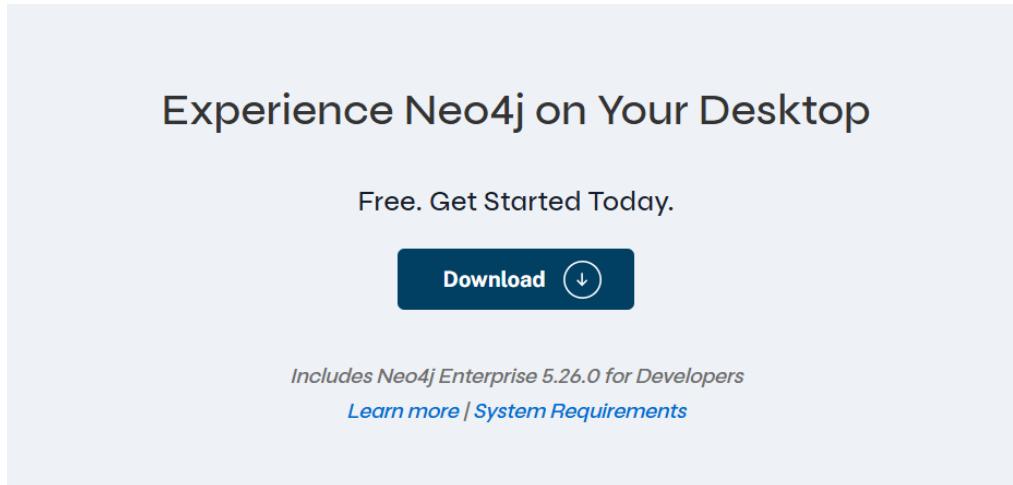
Po namyśle uznałem, że do zbioru warto dodać dane użytkowników i powiązać ich z recenzjami. Kolejnym ciekawym rozszerzeniem może być dodanie szefów kuchni i przypisanie ich jako pracowników do restauracji. Stąd też zdecydowałem się drugi zbiór danych wygenerować syntetycznie (skorzystam z biblioteki, która umożliwia tworzenie danych imitujących rzeczywiste) i powiązać ten zbiór z recenzjami i restauracjami.

Zbiór pierwszy można znaleźć na platformie Kaggle:

<https://www.kaggle.com/datasets/jkgatt/restaurant-data-with-100-trip-advisor-reviews-each>.

Zbiór drugi zamieszczony jest w plikach projektu (katalog `/extractingData/secondDataset`) oraz opisany dokładniej dalej w sekcjach *wstępna architektura bazy danych* oraz *import danych*.

Instalacja i konfiguracja:

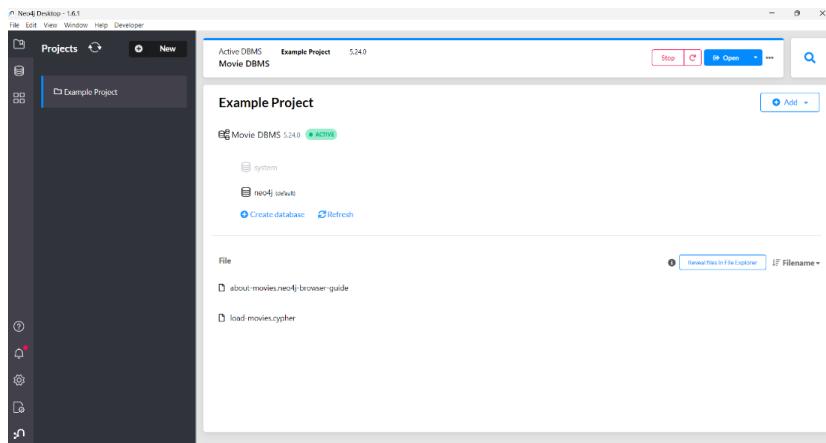


Zrzut ekranu nr 1: pobieranie Neo4j z oficjalnej strony producenta

Abytrzymać dostęp do instalatora należało wypełnić formularz, aby otrzymać klucz aktywacyjny. Następnie standardowo zainstalować krok po kroku.

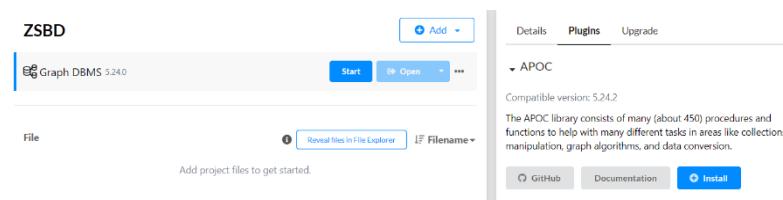
A screenshot of the Neo4j software registration form. It is divided into two main sections: "Software registration" on the left and "Activation" on the right. The "Software registration" section contains fields for "Name" (MichałZdanuk), "Email" (redacted), and "Organization" (Warsaw University of Technology). There is also a link to "Read about our privacy policy." The "Activation" section contains a field for "Software key" with a large redacted area and an "OR" option below it. At the bottom, there are three buttons: "Skip", "Register later", and a blue "Register with Email" button.

Zrzut ekranu nr 2: użycie klucza



Zrzut ekranu nr 3: poprawnie zainstalowana lokalnie baza Neo4j

Następnie stworzyłem bazę ZSBD i doinstalowałem plugin APOC, który wykorzystam do wczytania danych:



Zrzut ekranu nr 4: instalacja APOC

Przy pierwszym uruchomieniu importu otrzymałem błąd. Zatem dodałem w konfiguracji wpis umożliwiający wczytywanie plików przez APOC oraz zrestartowałem serwer.

```
neo4j$ CALL apoc.load.json("file:///restaurants.json") YIELD value AS restaurant MERGE (r:Restaur... ▶ ☆
```

ERROR | Neo.ClientError.Procedure.ProcedureCallFailed

Failed to invoke procedure `apoc.load.json`: Caused by: java.lang.RuntimeException: Import from files not enabled, please set apoc.import.file.enabled=true in your apoc.conf

Zrzut ekranu nr 5: błąd przy wczytywaniu

Rozwiązań mojego problemu znalazłem na forum społeczności neo4j (<https://community.neo4j.com/t/how-to-set-apoc-import-file-enable-true/34726/6>).

Następnie umieściłem wejściowe pliki json w bazowym katalogu importu `/import`. Możliwe jest również zdefiniowanie własnego katalogu i wskazanie ścieżki w pliku konfiguracyjnym.

```
neo4j$ CALL apoc.load.json("file:///restaurants.json") YIELD value RETURN value LIMIT 1;
```

value
{ "id": "ea25d323-0efd-4373-97df-dae0bc565f1", "region": "CA", "name": "21st Amendment Brewery & Restaurant", "locality": "San Francisco", "rating": 4.0, "country": "US" }

Started streaming 1 records after 3 ms and completed after 120 ms

Zrzut ekranu nr 6: weryfikacja poprawności pobierania danych przez APOC

Wstępna architektura bazy danych:

Zbiór wejściowy przetworzę tak by uzyskać odpowiedni podział na kilka rodzajów węzłów, które zostaną połączone krawędziami. Bazę zaprojektowałem tak by utworzyć następujące węzły:

- :Restaurant
- :Desription
- :Contact
- :Review
- :Schedule
- :Cuisine
- :MealOption

Krawędzie łączące węzły:

- **HAS_DESCRIPTION** – łączy restauracje z dodatkowymi charakterystykami
- **HAS_CONTACT** – łączy restauracje z danymi kontaktowymi
- **HAS_SCHEDULE** – łączy restauracje z grafikiem otwarcia
- **REVIEWS** – łączy recenzje z restauracją
- **SERVES_CUISINE** – łączy restaurację i rodzaj kuchni
- **OFFERS** – łączy restaurację i opcje posiłków

Drugi zbiór będzie zbiorem wygenerowanym samodzielnie przy użyciu skryptu pythonowego korzystając z biblioteki faker, aby dane przypominały rzeczywiste. Zdecydowałem się wzbogacić mój projekt o użytkowników (:User) oraz szefów kuchni (:Chef). Użytkowników powiązę z wystawionymi dla restauracji recenzjami poprzez **POSTED_BY**. Szefowie kuchni będą przypisani jako pracownicy restauracji za pomocą **WORKS_AT**.

Import danych:

Zdecydowałem się przygotować skrypt pythonowy, którym obsłużę wejściowy zbiór danych. Na wejście podałem zbiór danych w pliku json i na bazie zbioru wyekstraktowałem zbiory węzłów oraz relacji między węzłami (opisane w sekcji wstępna architektura bazy danych). Dane zagnieżdżone zostały spłaszczone, a zbędne wycięte. Następnie przygotowałem drugi skrypt za pomocą, którego wygeneruję drugi zbiór danych. Pliki ze skryptami: `prepareDataForNeo.py`, `generateSecondDataset.py`. Wygenerowane pliki, którymi zasilam Neo znajdują się w katalogach `/transformedData` oraz `/secondDataset`.

Skrypt tworzyłem przy pomocy narzędzia ChatGPT (wielokrotnie musiałem poprawiać wyniki i dostosowywać pod swoje potrzeby skrypt, ponieważ narzędzie popełniło wiele błędów). Cały projekt architektury oraz pomysł na wydzielenie węzłów i krawędzi jest mój (zdecydowanie nie ufam LLM w kwestii projektowania schematów czy architektury systemu). Wraz z zasilaniem danych zmieniałem skrypt względem początkowych założeń.

```
1 CALL apoc.load.json("file:///restaurants.json") YIELD value AS restaurant
2 MERGE (r:Restaurant {id: restaurant.id})
3 SET r.name = restaurant.name,
4     r.locality = restaurant.locality,
5     r.region = restaurant.region,
6     r.country = restaurant.country,
7     r.rating = restaurant.rating;
```

Added 147 labels, created 147 nodes, set 882 properties, completed after 108 ms.



Zrzut ekranu nr 7: zasilenie węzłów Restaurant

```
1 CALL apoc.load.json("file:///descriptions.json") YIELD value AS description
2 MERGE (d:Description {id: description.id})
3 SET d.parking = description.parking,
4     d.wifi = description.wifi,
5     d.smoking = description.smoking,
6     d.accessible_wheelchair = description.accessible_wheelchair;
```

Added 147 labels, created 147 nodes, set 735 properties, completed after 21 ms.



Zrzut ekranu nr 8: zasilenie węzłów Description

```
1 CALL apoc.load.json("file:///relationships.json") YIELD value AS rel
2 WITH rel WHERE rel.type = "HAS_DESCRIPTION"
3 MATCH (r:Restaurant {id: rel.from}), (d:Description {id: rel.to})
4 MERGE (r)-[:HAS_DESCRIPTION]-(d);
```

Created 147 relationships, completed after 1040 ms.



Zrzut ekranu nr 9: utworzenie krawędzi pomiędzy Restaurant i Description

```
1 CALL apoc.load.json("file:///contacts.json") YIELD value AS contact
2 MERGE (c>Contact {id: contact.id})
3 SET c.tel = contact.tel,
4     c.fax = contact.fax,
5     c.website = contact.website,
6     c.email = contact.email,
7     c.latitude = contact.latitude,
8     c.longitude = contact.longitude;
```

Added 147 labels, created 147 nodes, set 1029 properties, completed after 151 ms.



Zrzut ekranu nr 10: zasilenie węzłów Contact

```
1 CALL apoc.load.json("file:///relationships.json") YIELD value AS rel
2 WITH rel WHERE rel.type = "HAS_CONTACT"
3 MATCH (r:Restaurant {id: rel.from}), (c>Contact {id: rel.to})
4 MERGE (r)-[:HAS_CONTACT]-(c);
```

Created 147 relationships, completed after 589 ms.



Zrzut ekranu nr 11: utworzenie krawędzi pomiędzy Restaurant i Contact

```
1 CALL apoc.load.json("file:///reviews.json") YIELD value AS review
2 MERGE (rev:Review {id: review.id})
3 SET rev.review_website = review.review_website,
4     rev.review_title = review.review_title,
5     rev.review_text = review.review_text,
6     rev.review_rating = review.review_rating,
7     rev.review_date = review.review_date;
```

Added 14700 labels, created 14700 nodes, set 88200 properties, completed after 69584 ms.



Table

Zrzut ekranu nr 12: zasilenie węzłów Review

```
1 CALL apoc.load.json("file:///relationships.json") YIELD value AS rel
2 WITH rel WHERE rel.type = "REVIEWS"
3 MATCH (r:Restaurant {id: rel.to}), (rev:Review {id: rel.from})
4 MERGE (rev)-[:REVIEWS]→(r);
```

Created 14700 relationships, completed after 217885 ms.



Table

Zrzut ekranu nr 13: utworzenie krawędzi pomiędzy Review i Restaurant

```
1 CALL apoc.load.json("file:///schedules.json") YIELD value AS schedule
2 MERGE (s:Schedule {id: schedule.id})
3 SET s.monday = schedule.monday,
4     s.tuesday = schedule.tuesday,
5     s.wednesday = schedule.wednesday,
6     s.thursday = schedule.thursday,
7     s.friday = schedule.friday,
8     s.saturday = schedule.saturday,
9     s.sunday = schedule.sunday;
```

Added 147 labels, created 147 nodes, set 1176 properties, completed after 225 ms.



Table

Zrzut ekranu nr 14: zasilenie węzłów Schedule

```
1 CALL apoc.load.json("file:///relationships.json") YIELD value AS rel
2 WITH rel WHERE rel.type = "HAS_SCHEDULE"
3 MATCH (r:Restaurant {id: rel.from}), (s:Schedule {id: rel.to})
4 MERGE (r)-[:HAS_SCHEDULE]→(s);
```

Created 147 relationships, completed after 1405 ms.



Table

Zrzut ekranu nr 15: utworzenie krawędzi pomiędzy Restaurant i Schedule

```
1 CALL apoc.load.json("file:///cuisines.json") YIELD value AS cuisine
2 MERGE (c:Cuisine {id: cuisine.id})
3 SET c.name = cuisine.name;
```

Added 93 labels, created 93 nodes, set 186 properties, completed after 68 ms.



Table

Zrzut ekranu nr 16: zasilenie węzłów Cuisine

```
1 CALL apoc.load.json("file:///relationships.json") YIELD value AS rel
2 WITH rel WHERE rel.type = "SERVES_CUISINE"
3 MATCH (r:Restaurant {id: rel.from}), (c:Cuisine {id: rel.to})
4 MERGE (r)-[:SERVES_CUISINE]→(c);
```

Created 722 relationships, completed after 662 ms.



Table

Zrzut ekranu nr 17: utworzenie krawędzi pomiędzy Restaurant i Cuisine

```
1 CALL apoc.load.json("file:///mealoptions.json") YIELD value AS mealoption
2 MERGE (m:MealOption {id: mealoption.id})
3 SET m.meal_breakfast = mealoption.meal_breakfast,
4   m.meal_deliver = mealoption.meal_deliver,
5   m.meal_dinner = mealoption.meal_dinner,
6   m.meal_lunch = mealoption.meal_lunch,
7   m.meal_takeout = mealoption.meal_takeout,
8   m.meal_cater = mealoption.meal_cater;
```



Added 147 labels, created 147 nodes, set 1029 properties, completed after 73 ms.

Zrzut ekranu nr 18: zasilenie węzłów MealOption

```
1 CALL apoc.load.json("file:///relationships.json") YIELD value AS rel
2 WITH rel WHERE rel.type = "OFFERS"
3 MATCH (r:Restaurant {id: rel.from}), (m:MealOption {id: rel.to})
4 MERGE (r)-[:OFFERS]→(m);
```



Created 147 relationships, completed after 205 ms.

Zrzut ekranu nr 19: utworzenie krawędzi pomiędzy Restaurant i MealOption

```
1 CALL apoc.load.json("file:///chefs.json") YIELD value AS chef
2 MERGE (c:Chef {id: chef.id})
3 SET c.name = chef.name,
4   c.speciality = chef.speciality;
5
```



Added 100 labels, created 100 nodes, set 200 properties, completed after 53 ms.

Zrzut ekranu nr 20: zasilenie węzłów Chef

```
1 CALL apoc.load.json("file:///chef_restaurant_relationships.json") YIELD value AS rel
2 WITH rel WHERE rel.type = "WORKS_AT"
3 MATCH (r:Restaurant {id: rel.to}), (c:Chef {id: rel.from})
4 MERGE (c)-[:WORKS_AT]→(r);
```



Created 147 relationships, completed after 162 ms.

Zrzut ekranu nr 21: utworzenie krawędzi pomiędzy Chef i Restaurant

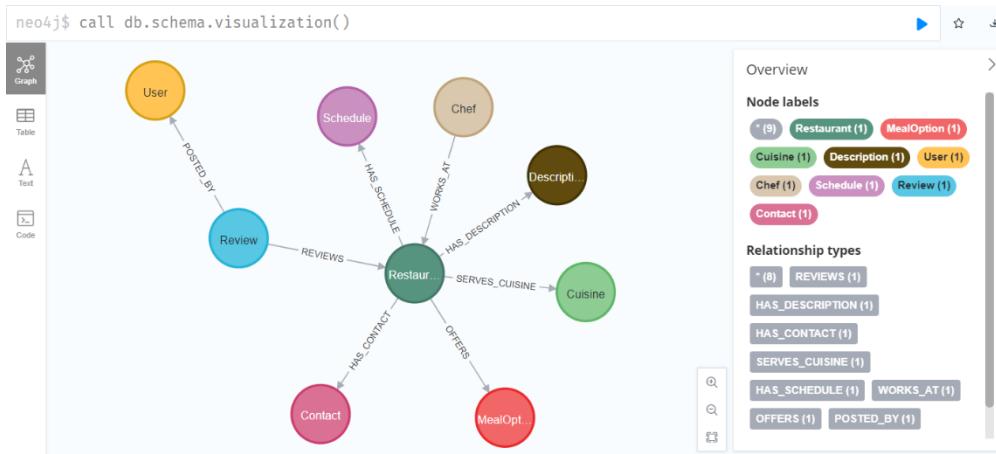
```
1 CALL apoc.load.json("file:///users.json") YIELD value AS user
2 MERGE (u:User {id: user.id})
3 SET u.name = user.name,
4   u.email = user.email,
5   u.address = user.address;
6
```



Added 50 labels, created 50 nodes, set 200 properties, completed after 69 ms.

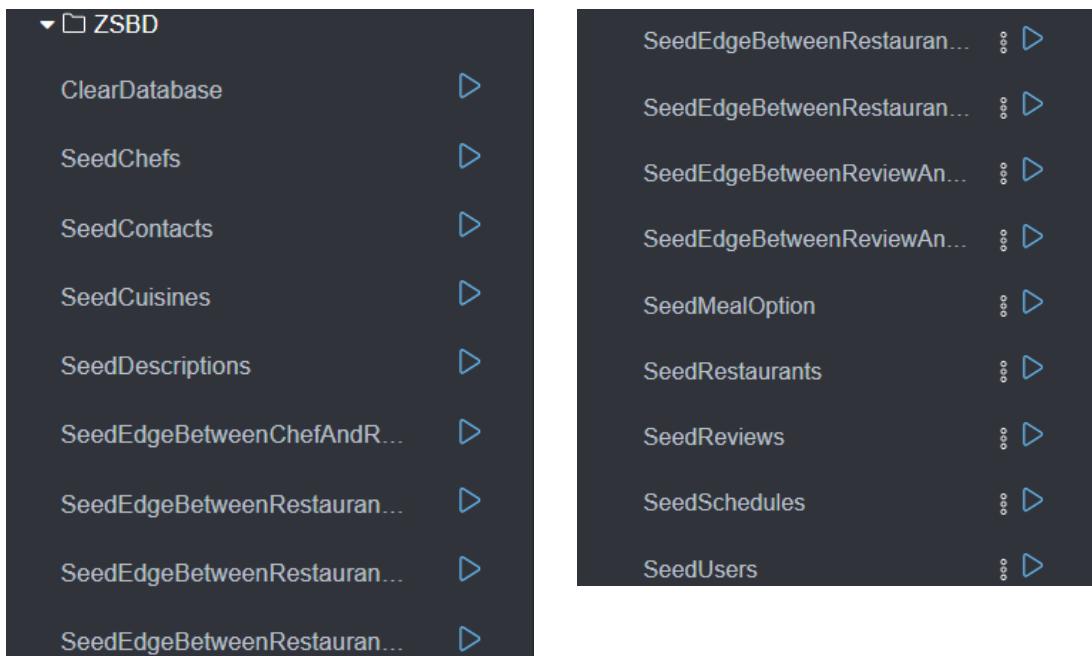
Zrzut ekranu nr 22: zasilenie węzłów User

Poniżej prezentuję schemat bazy po zimportowaniu dwóch zbiorów. W bazie mamy dostęp do 17 628 węzłów. Wykorzystałem wbudowane polecenie: db.schema.visualization()



Zrzut ekranu nr 23: schemat bazy danych (graf)

Cały proces zasilania i obróbki danych był bardzo żmudnym zadaniem, które pochłonęło wiele czasu – głównie przez brak doświadczenia w obsłudze tej bazy (wielokrotnie korygowałem skrypt przygotowujący dane do zasilania, poprawiałem polecenia Cypherowe do załadowania danych, ładowałem i usuwałem dane, gdy błędnie tworzyły się węzły/krawędzie). Mimo wszystko jestem zadowolony z tego, że udało mi się częściowo zautomatyzować proces zasilania i odtwarzania bazy. Wykorzystałem do tego wbudowaną funkcjonalność Neo4j Browser jaką jest zapisywanie do ulubionych zapytań wykonywanych na bazie – zapisałem polecenie do czyszczenia bazy oraz polecenia do seedowania odpowiednich węzłów i tworzenia krawędzi w grafie.



Zrzut ekranu nr 24: zapisane zapytania przyspieszające proces zasilania bazy danych

Zapytania:

Poniżej zademonstruję 5 przykładowych zapytań na mojej bazie danych:

- 1) Zapytanie zwracające 5 najlepiej ocenianych restauracji mających w nazwie człon ‘restaurant’ wraz z podstawowymi danymi kontaktowymi (telefon, strona www, email, położenie geograficzne)
- 2) Restauracje oferujące daną kuchnię (kategoria Cafe) z oceną powyżej określonego progu wraz z dodatkowymi informacjami (dostępność parkingu i przystępność dla osób na wózku) oraz danymi głównego szefa kuchni
- 3) Zliczenie liczby restauracji dla każdego rodzaju kuchni jednocześnie wyliczając średnią ocenę restauracji z kategorii, wyniki posortowane po liczności restauracji, a następnie po średniej ocenie
- 4) Zwrócenie nazw restauracji (z wykorzystaniem UNION), które [mają ocenę powyżej 4.5 i dostępny parking] lub [serwują włoską kuchnią bądź owoce morze]
- 5) Utworzenie recenzji i podpięcie pod nią użytkownika dla restauracji o podanej nazwie (wykorzystanie MERGE)

```

1 MATCH (r:Restaurant)-[:HAS_CONTACT]-(c:Contact)
2 WHERE r.name =~ "(?i).*restaurant.*"
3 RETURN r.name, r.rating, c.tel, c.website, c.email, c.latitude, c.longitude
4 ORDER BY r.rating DESC
5 LIMIT 5
    
```

	r.name	r.rating	c.tel	c.website	c.email	c.latitude	c.longitude
1	"Greens Restaurant"	4.5	"(415) 771-6222"	"http://www.greenarestaurant.com"	"info@greenarestaurant.com"	37.806949	-122.432172
2	"Amber India Restaurant"	4.5	"(415) 777-0500"	"http://www.amber-india.com/SanFrancisco"	"amberindias@gmail.com"	37.785772	-122.404401
3	"Delancey Street Restaurant"	4.5	"(415) 512-5179"	"http://www.delanceystreetfoundation.org"	"raymond@delanceystreetfoundation.org"	37.784486	-122.388432
4	"Delfina Restaurant"	4.5	"(415) 552-4055"	"http://www.delfinasf.com"	"delfina@delfinasf.com"	37.761467	-122.424315
5	"Fattoush Restaurant"	4.5	"(415) 641-0678"	"http://www.fattoush.com"	"amas@fattoush.com"	37.749033	-122.427052

Started streaming 5 records after 1 ms and completed after 7 ms.

Zrzut ekranu nr 25: rezultat zapytania nr 1 (restauracje z pasującym członem nazwy)

```

1 MATCH (r:Restaurant)-[:SERVES_CUISINE]-(c:Cuisine {name: "Cafe"})
2 MATCH (r)-[:HAS_DESCRIPTION]-(d:Description)
3 MATCH (chef:Chef)-[:WORKS_AT]-(r)
4 WHERE r.rating > 4.5
5 MATCH (r)-[:HAS_CONTACT]-(contact:Contact)
6 RETURN r.name AS Restaurant, chef.name AS MainChef, r.rating AS Rating, contact.tel AS Phone, contact.website AS Website, d.parking AS isAvailableParking, d.accessible_wheelchair AS wheelchairAccessible
7 ORDER BY r.rating DESC
    
```

	Restaurant	MainChef	Rating	Phone	Website	isAvailableParking	wheelchairAccessible
1	"Bazaar Cafe"	"Paula Garcia"	5.0	"(415) 831-5620"	"http://bazaarcafe.com"	true	true
2	"L's Caffe"	"Gregory Clark"	5.0	"(415) 206-0274"	"http://www.lscaffe.net/"	true	true
3	"Nanis Coffee"	"John Barber"	5.0	"(415) 928-8817"	"http://www.naniscoffee.com"	true	true

Started streaming 3 records after 29 ms and completed after 32 ms.

Zrzut ekranu nr 26: rezultat zapytania nr 2 (kawiarnie po filtrowane po ocenie)

```

1 MATCH (r:Restaurant)-[:SERVES_CUISINE]-(c:Cuisine)
2 WITH c.name AS cuisine, COUNT(r) AS restaurant_count, AVG(r.rating) AS average_rating
3 RETURN cuisine, restaurant_count, apoc.number.format(average_rating, '0.00') AS
4 average_rating
5 ORDER BY restaurant_count DESC, average_rating DESC

```

cuisine	restaurant_count	average_rating
"American"	91	"3.97"
"Cafe"	50	"4.04"
"Italian"	30	"4.08"
"Sandwiches"	30	"4.07"
"Seafood"	29	"4.00"
"Burgers"	29	"3.88"

Started streaming 93 records after 22 ms and completed after 55 ms.

Zrzut ekranu nr 27: rezultat zapytania nr 3 (posortowane agregacje dla kategorii kuchni)

```

1 MATCH (r:Restaurant)-[:HAS_DESCRIPTION]-(d:Description)
2 WHERE r.rating > 4.5 AND d.parking = true
3 RETURN r.name
4 UNION DISTINCT
5 MATCH (r:Restaurant)-[:SERVES_CUISINE]-(c:Cuisine)
6 WHERE c.name IN ['Italian', 'Seafood']
7 RETURN r.name

```

r.name
"Bazaar Cafe"
"Beretta"
"L's Caffe"
"Nanis Coffee"
"Americano"
"Bacco"

Started streaming 55 records after 29 ms and completed after 33 ms.

Zrzut ekranu nr 28: rezultat zapytania nr 4 (wykorzystanie UNION do zwrócenia zbioru składającego się z dwóch podzbiorów o zadanych kryteriach)

```

1 MERGE (r:Restaurant {name: 'Amber India Restaurant'})
2 MERGE (rev:Review {id: '12345abcdef'})
3 SET rev.review_website = 'website_url',
4 rev.review_title = 'Fantastic Meal',
5 rev.review_text = 'The food was excellent!',
6 rev.review_rating = 5,
7 rev.review_date = '2024-12-15'
8 MERGE (u:User {id: 'userid'})
9 SET u.name = 'User Name',
10 u.email = 'user@example.com',
11 u.address = '123 Main St'
12

```

The diagram shows a graph database interface. On the left, there are four tabs: Graph (selected), Table, Text, and Code. In the center, two nodes are connected by a line labeled "POSTED_BY". The top node is yellow and labeled "User Name". The bottom node is blue and labeled "Amber". A line labeled "REVIEWS" points from the "Amber" node down to the "User Name" node. On the right, a "Node properties" panel is open for the "Review" node. It lists several properties with their values: elementId (4:d92c7d83-add7-4685-af10-02cc32fc1190:17638), id (17638), review_date (2024-12-15), review_rating (5), review_text (The food was excellent!), review_title (Fantastic Meal), review_web (website_url), and site.

Zrzut ekranu nr 29: rezultat zapytania nr 5 (wykorzystanie MERGE do utworzenia recenzji dla restauracji o podanej nazwie)

Indeksy:

Indeksy w Neo4j zdają się przypominać indeksy w bazach relacyjnych. Przy czym tutaj mogą być one zakładane na węzły, relacje bądź właściwości. Dane składowane w indeksie wskazują ścieżkę do pełnych danych w bazie i umożliwiają nam szybsze przeszukiwanie i filtrowanie danych.

Utworzę indeks tekstowy na restaurant.name – często chcemy wyszukiwać restauracji po fragmencie nazwy. Następnie dodam indeks na restaurant.rating – wyszukiwarki praktycznie zawsze mają możliwość filtrowania po ocenie ogólnej placówek. Dalej dodam indeks na cuisine.name oraz indeksy na description.parking, description.accessible_wheelchair – myślę, że mogą to być najpopularniejsze flagi wykorzystywane w zapytaniach.

```
neo4j$ MATCH (r:Restaurant)-[:HAS_DESCRIPTION]→(d:Description) WHERE r.rating > 4.5 AND d.parking = true RETURN r.name UNION DISTINCT ...  
Started streaming 55 records after 2 ms and completed after 14 ms.  
  
neo4j$ MATCH (r:Restaurant)-[:SERVES_CUISINE]→(c:Cuisine) WITH c.name AS cuisine, COUNT(r) AS restaurant_count, AVG(r.rating) AS ave...  
Started streaming 93 records after 3 ms and completed after 68 ms.  
  
neo4j$ MATCH (r:Restaurant)-[:SERVES_CUISINE]→(c:Cuisine {name: "Cafe"}) MATCH (r)-[:HAS_DESCRIPTION]→(d:Description) MATCH (chef:C...  
Started streaming 3 records after 2 ms and completed after 10 ms.  
  
neo4j$ MATCH (r:Restaurant)-[:HAS_CONTACT]→(c>Contact) WHERE toLower(r.name) CONTAINS "restaurant" RETURN r.name, r.rating, c.tel, c...  
Started streaming 5 records after 2 ms and completed after 6 ms.
```

Zrzut ekranu nr 30: czasy zapytań przed dodaniem indeksów

Utworzyłem indeks tekstowy:

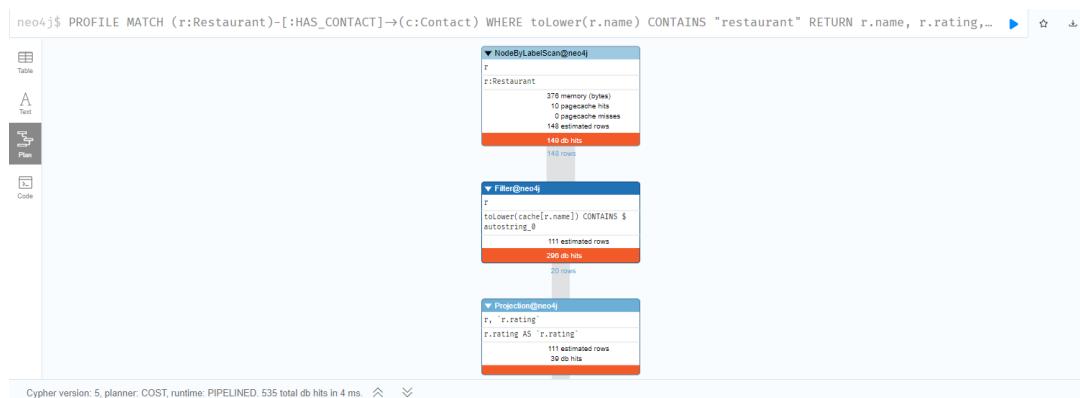
```
neo4j$ CREATE TEXT INDEX restaurant_name_index FOR (r:Restaurant) ON (r.name);  
Added 1 index, completed after 140 ms.
```

Zrzut ekranu nr 31: utworzenie indeksu

Zapytanie, w którym przeszukujemy po nazwie przyspieszyło z 6ms na 4ms. Pozostałe zapytania mają praktycznie te same czasy

```
neo4j$ MATCH (r:Restaurant)-[:HAS_CONTACT]→(c>Contact) WHERE toLower(r.name) CONTAINS "restaurant" RETURN r.name, r.rating, c.tel, ...  
Started streaming 5 records after 1 ms and completed after 4 ms.
```

Zrzut ekranu nr 32: wynik zapytania korzystającego z indeksu



Zrzut ekranu nr 33: rezultat komendy PROFILE do zapytania

```
neo4j$ CREATE INDEX restaurant_rating_index FOR (r:Restaurant) ON (r.rating)
Added 1 index, completed after 10 ms.
```

Zrzut ekranu nr 34: utworzenie indeksu na restaurant.rating

```
neo4j$ MATCH (r:Restaurant)-[:SERVES_CUISINE]→(c:Cuisine {name: "Cafe"}) MATCH (r)-[:HAS_DESCRIPTION]→(d:Description) MATCH (chef:...
```

Started streaming 3 records after 1 ms and completed after 3 ms.

```
neo4j$ MATCH (r:Restaurant)-[:HAS_DESCRIPTION]→(d:Description) WHERE r.rating > 4.5 AND d.parking = true RETURN r.name UNION DISTIN...
```

Started streaming 55 records after 1 ms and completed after 3 ms.

Zrzut ekranu nr 35: wyniki zapytań, które miały filtrowanie po ocenie

Z tego co zauważylem pierwsze z powyższych zapytań mimo wszystko wykorzystuje indeks na cuisine.name i otrzymujemy satysfakcyjny rezultat 3ms. Natomiast drugie zapytanie przyspieszyło z 14ms do 3ms, zatem indeks okazał się przydatny.

Cypher version: 5, planner: COST, runtime: PIPELINED. 219 total db hits in 3 ms. ▲ ▼

Zrzut ekranu nr 36: profiler dla zapytania

```
neo4j$ CREATE INDEX cuisine_name_index FOR (c:Cuisine) ON (c.name)
Added 1 index, completed after 17 ms.
```

Zrzut ekranu nr 37: utworzenie indeksu na cuisine.name

Rezultat poprawił się z 10ms na 3ms. Pozostałe wyniki zostały mniej więcej bez zmian.

```
neo4j$ MATCH (r:Restaurant)-[:SERVES_CUISINE]→(c:Cuisine {name: "Cafe"}) MATCH (r)-[:HAS_DESCRIPTION]→(d:Description) MATCH (chef:...
```

Started streaming 3 records after 1 ms and completed after 3 ms.

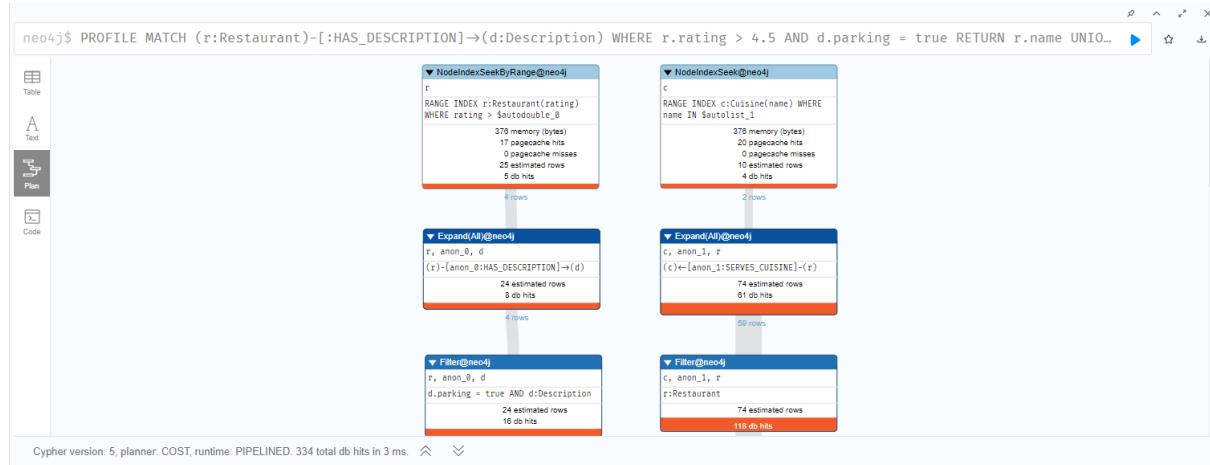
Zrzut ekranu nr 38: wynik zapytania korzystającego z indeksu

```
$ CREATE INDEX description_parking_index FOR (d:Description) ON (d.parking); CREATE INDEX description_accessible_index FOR (d.Descri...
```

neo4j\$ CREATE INDEX description_parking_index FOR (d:Description) ON (d.parking)	<input checked="" type="checkbox"/>
neo4j\$ CREATE INDEX description_accessible_index FOR (d:Description) ON (d.accessible_wheelchair)	<input checked="" type="checkbox"/>

Zrzut ekranu nr 39: utworzenie indeksu dla wybranych flag description

Z tego co zaobserwowałem to w tym konkretnym zapytaniu serwer postanowił nie wykorzystywać indeksów, nie mniej zostawiłbym go – dla zapytań, gdzie tylko ta flaga jest zaznaczona (inne zapytania nie zwolniły, stąd decyzja o zostawieniu indeksu).



Zrzut ekranu nr 40: profiler dla zapytania

SHOW INDEXES											
	id	name	state	populationPercent	type	entityType	labelsOrTypes	properties	indexProvider	owningConstraint	lastRead
2	6	"description_accessible_index"	"ONLINE"	100.0	"RANGE"	"NODE"	["Description"]	["accessible_wheelchair"]	"range-1.0"	null	null
3	5	"description_parking_index"	"ONLINE"	100.0	"RANGE"	"NODE"	["Description"]	["parking"]	"range-1.0"	null	null
4	0	"index_343aff4e"	"ONLINE"	100.0	"LOOKUP"	"NODE"	null	null	"token-lookup-1.0"	null	"2024-12-16T20:26:0"
5	1	"index_f7700477"	"ONLINE"	100.0	"LOOKUP"	"RELATIONSHIP"	null	null	"token-lookup-1.0"	null	"2024-12-15T17:03:5"
6	2	"restaurant_name_index"	"ONLINE"	100.0	"TEXT"	"NODE"	["Restaurant"]	["name"]	"text-2.0"	null	"2024-12-16T20:09:5"
7	4	"restaurant_rating_index"	"ONLINE"	100.0	"RANGE"	"NODE"	["Restaurant"]	["rating"]	"range-1.0"	null	"2024-12-

Started streaming 7 records in less than 1 ms and completed after 1 ms.

Zrzut ekranu nr 41: lista indeksów na serwerze

Poza mechanizmem indeksów możemy optymalizować zapytania na kilka innych sposobów przede wszystkim:

- Odpowiednie modelowanie zapytań.** Zdarza nam się pisać zapytania „na około”, ponieważ skupiliśmy się tylko na końcowym rezultacie – może okazać się, że nasze zapytanie trawersuje niepotrzebne krawędzie i węzły. W tym celu możemy korzystać z funkcjonalności EXPLAIN przed wywołaniem zapytania – Neo4j pokaże nam plan zapytania, a my możemy go przeanalizować i dojść do refleksji, czy możemy sami poprawić zapytanie.
- Wykorzystanie profilera.** Neo daje nam kolejne narzędzie jakim jest PROFILE – daje nam wgląd do dokładniejszych statystyk zapytania – możemy podejrzeć jaka strategia bądź, które indeksy zostały wykorzystane i ile czasu pochłonęły.

3. **Parametryzowane zapytania.** Neo4j pozwala na cache'owanie planów dla parametryzowanych query. Może to nam pomóc poprawić wydajność dla często wywoływanych zapytań z różnymi wartościami często używanego parametru
4. **Wykorzystanie etykiet (ang. labels).** Etykiety wykorzystywane są do grupowania węzłów i polepszają wydajność zapytań. Możemy po analizie pogrupować domenowo węzły i nawet dla istniejącej już bazy dodać etykiety, a następnie wykorzystać je w zapytaniach, do zawężenia przeszukiwań.

Funkcje przestrzenne:

Wybrany przeze mnie zbiór zawierał dane przestrzenne, jednakże początkowo błędnie je przetworzyłem – dodałem bezpośrednio pola latitude oraz longitude. Poprawiłem to poprzez dodanie point do wszystkich węzłów :Contact.

```
1 MATCH (r:Restaurant)-[:HAS_CONTACT]-(c:Contact)
2 WHERE c.latitude IS NOT NULL AND c.longitude IS NOT NULL
3 SET c.location = point({latitude: c.latitude, longitude: c.longitude})
```

Set 147 properties, completed after 118 ms.

Table

Code

Set 147 properties, completed after 118 ms.

Zrzut ekranu nr 41: korekta danych przestrzennych

Wyszukanie 5 najbliższych restauracji względem naszych współrzędnych (powszechna i bardzo ważna funkcjonalności aplikacji):

```
1 MATCH (r:Restaurant)-[:HAS_CONTACT]-(c:Contact)
2 WHERE c.location IS NOT NULL
3 RETURN r.name, point.distance(c.location, point({latitude: 37.7724, longitude: -122.39256})) AS distance
4 ORDER BY distance ASC
5 LIMIT 5;
```

r.name	distance
"Mariowe"	547.4166240267489
"Ironside"	948.0221767884846
"21st Amendment Brewery & Restaurant"	1118.539655517349
"Chez Maman"	1171.6191564774938
"Crossroads Cafe"	1223.1290623311122

Started streaming 5 records after 18 ms and completed after 28 ms.

Zrzut ekranu nr 42: rezultat zapytania

Ciekawym pomysłem na wykorzystanie agregacji może być zwrócenie średniej oceny restauracji znajdujących się z zadanej okolicy (np. w promieniu 1,5km od naszego położenia).

```
1 MATCH (r:Restaurant)-[:HAS_CONTACT]→(c:Contact)
2 WHERE c.location IS NOT NULL AND point.distance(c.location, point({latitude: 37.7724, longitude: -122.39256})) < 1500
3 RETURN avg(r.rating) AS average_rating;
4
```

average_rating
4.11111111111112

Started streaming 1 records after 21 ms and completed after 31 ms.

Zrzut ekranu nr 43: rezultat zapytania z agregacją

Zaletą tych funkcji jest prostota, czytelne API i dostępność na „wyciągnięcie ręki” ciekawych funkcjonalności, które mamy od razu w bazie i nie musimy sami implementować.

Wadą jest konieczność do dostosowania się do podanego standardu zapisu danych. Poza tym musimy posiadać dokładne numeryczne pomiary, aby lokalizacje spełniały tolerancję akceptowalnych przez nas niedokładności.

Ogólnie dane przestrzenne uważam za cenną wiedzę, którą można wykorzystać do wzbogacenia logiki biznesowej i oferowania wielu usprawnień klientom. Dostępność i prostotę użycia funkcji przestrzennych w Neo4j oceniam bardzo pozytywnie. W niemal każdej branży możemy znaleźć potrzebę używania danych przestrzennych, a zatem jest to powszechnie wymagana funkcjonalność – szczególnie ważna w branży transportu i logistyki.

Procedura:

Tworzenie procedury okazało się trudniejszym zadaniem. Wykorzystałem szablon do tworzenia procedur udostępniony przez twórców Neo4j (<https://github.com/neo4j-examples/neo4j-procedure-template>). Uznałem, że wygodną procedurą/utilem będzie własna funkcjonalność, która będzie nam wyliczać dystans pomiędzy dwoma zadanimi wierzchołkami (przydatne do sprawdzenia odległości między restauracjami, gdy planujemy się przemieścić). W tym celu musiałem zainstalować Maven'a. Skorzystałem z instrukcji: <https://phoenixnap.com/kb/install-maven-windows> (wystarczyło pobrać „binarkę” i dodać ścieżkę do zmiennej \$PATH) i zrestartować system. Następnie próbowałem użyć ChatGPT do wygenerowania kodu procedury, jednak kompletnie poległy na tym kroku próbując wcisnąć mi wymyślone metody i nieistniejące API (dwukrotnie) 😊. Poniżej pokażę omamy Chata jako ciekawostkę:

```
Point location1 = (Point) node1.getProperty("location", null);
Point location2 = (Point) node2.getProperty("location", null);

if (location1 == null || location2 == null) {
    log.warn("Brak współrzędnych geograficznych w jednym z węzłów.");
    return Stream.of(new DistanceResult(-1.0)); // Wartość -1 jako błąd
}

// Obliczenie odległości w kilometrach
double distance = calculateHaversineDistance(
    location1.latitude(), location1.longitude(),
    location2.latitude(), location2.longitude()
);
```

Zrzut ekranu nr 43: wyimaginowane metody latitude() i longitude()

2. Nieprawidłowe przypisanie lokalizacji w obliczeniach: W części kodu masz:

```
java
Coordinate cords = location1.getCoordinate();
double[] point1Cords = cords.getCoordinate();

Coordinate cords2 = location1.getCoordinate();
double[] point2Cords = cords.getCoordinate();
```

- `location1` jest użyte dwa razy zamiast `location1` i `location2`. To oznacza, że współrzędne dla obu punktów są takie same.
- Powinno być:

```
java
double lat1 = location1.getY();
double lon1 = location1.getX();

double lat2 = location2.getY(); ↓
double lon2 = location2.getX();
```

Zrzut ekranu nr 44: dalsze wyimaginowane metody getX() i getY()

W tym momencie samemu zająłem się API i klasy Point oraz dalszej Coordinate. Znalazłem wyciąganie koordynatów i tablicy z współrzędnymi.

```
import org.neo4j.annotations.api.PublicApi;
|
/**
 * A point is a geometry described by a single coordinate in space.
 * <p>
 * A call to {@link #getCoordinates()} must return a single element list.
 */
@PublicApi
public interface Point extends Geometry {
    /**
     * Returns the single coordinate in space defining this point.
     *
     * @return The coordinate of this point.
     */
    default Coordinate getCoordinate() {
        return getCoordinates().get(0);
    }

    @Override
    default String getGeometryType() {
        return "Point";
    }
}
```

Zrzut ekranu nr 45: klasa Point zwracająca klasę Coordinate

```
@PublicApi
public final class Coordinate {
    private final double[] coordinate;

    public Coordinate(double... coordinate) {
        if (coordinate.length < 2) {
            throw new IllegalArgumentException("A coordinate must have at least two elements");
        }
        this.coordinate = coordinate;
    }

    public double[] getCoordinate() {
        return coordinate;
    }

    public double[] getCoordinateCopy() {
        return Arrays.copyOf(coordinate, coordinate.length);
    }
}
```

Zrzut ekranu nr 46: klasa Coordinate zwracająca tablicę double z koordynatami

Zweryfikowałem wygenerowaną metodę wyliczania dystansu z koordynat. Wykorzystuje się w tym celu formułę harvesine (<https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128>).

```

public class CalculateDistance {

    @Context
    public Log log;

    @Procedure(name = "example.calculateDistance")
    @Description("Calculates distance between two specified nodes taht contain point property. Result returned in kilometres.")
    public Stream<DistanceResult> calculateDistance(
        @Name("node1") Node node1,
        @Name("node2") Node node2) {

        Point location1 = (Point) node1.getProperty(key:"location", defaultValue:null);
        Point location2 = (Point) node2.getProperty(key:"location", defaultValue:null);

        if (location1 == null || location2 == null) {
            log.warn("Invalid node! Not found geolocation data in nodes.");
            return Stream.of(new DistanceResult(-1.0));
        }

        Coordinate cord1 = location1.getCoordinate();
        Coordinate cord2 = location2.getCoordinate();

        double lon1 = cord1.getCoordinate()[0];
        double lat1 = cord1.getCoordinate()[1];

        double lon2 = cord2.getCoordinate()[0];
        double lat2 = cord2.getCoordinate()[1];

        double distance = calculateHaversineDistance(lat1, lon1, lat2, lon2);
        return Stream.of(new DistanceResult(distance));
    }

    private double calculateHaversineDistance(double lat1, double lon1, double lat2, double lon2) {
        final int R = 6371;

        double latDistance = Math.toRadians(lat2 - lat1);
        double lonDistance = Math.toRadians(lon2 - lon1);

        double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2)
                + Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2))
                * Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2);

        double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
        return R * c; // Wynik w kilometrach
    }

    public static class DistanceResult {
}

```

Zrzut ekranu nr 47: klasa odpowiadająca procedurze

Następnie zgodnie z instrukcją wygenerowałem plik jar, który umieściłem do katalogu /plugins i zrestartowałem serwer. Plik z klasą procedury oraz wynikowy plik .jar zamieszczam w załączonych plikach.

neo4j\$ SHOW PROCEDURES		
	name	description
273	"dbms.showTopologyGraphConfig"	"With this method the configuration of the Topology Graph can be displayed."
274	"dbms.upgrade"	"Upgrade the system database schema if it is not the current schema."
275	"dbms.upgradeStatus"	"Report the current status of the system database sub-graph schema."
276	"example.allnodes"	...
277	"example.calculateDistance"	"Calculates distance between two specified nodes taht contain point property. Result returned in kilometres."
278	"example.getRelationshipTypes"	"Get the different relationships going in and out of a node."

Started streaming 282 records after 93 ms and completed after 114 ms.

Zrzut ekranu nr 48: poprawnie dodana własna procedura

```
1 MATCH (r1:Restaurant)-[:HAS_CONTACT]→(c1:Contact),  
2      (r2:Restaurant)-[:HAS_CONTACT]→(c2:Contact)  
3 WHERE c1.location IS NOT NULL AND c2.location IS NOT NULL  
4       AND r1.name = 'Americano' AND r2.name = 'Bacco'  
5 CALL example.calculateDistance(c1, c2)  
6 YIELD distance  
7 RETURN distance;  
8
```

Table	distance
1	6.036111014015095

Started streaming 1 records after 3 ms and completed after 10 ms.

Zrzut ekranu nr 49: wynik procedury – wyliczony dystans w kilometrach między dwoma zadanimi restauracjami

```
1 MATCH (r1:Restaurant)-[:HAS_CONTACT]→(c1:Contact)  
2 MATCH (u:User{name: 'Allison Hill'})  
3 WHERE r1.name = 'Americano'  
4 CALL example.calculateDistance(c1, u)  
5 YIELD distance  
6 RETURN distance;  
7
```

Table	distance
1	-1.0

Started streaming 1 records after 3 ms and completed after 34 ms.

Zrzut ekranu nr 50: procedura została zabezpieczona przed podaniem błędnych danych, jeśli węzeł nie posiada poprawnie zdefiniowanych koordynatów otrzymamy wynik -1

W Neo4j procedury różnią się pod kilkoma aspektami względem tych, która znamy z baz relacyjnych. Przede wszystkim w Neo4j procedurę musiałem pisać w języku programowania, a nie języku zapytań. Jest to jednocześnie wada i zaleta. Wadą jest wyższy próg wejścia i skomplikowania - procedury pisze się znacznie bardziej, musimy znać paradygmat programowania (tutaj Java i obiektowość). Z drugiej strony mamy więcej możliwości i elastyczność, poza tym do procedury możemy napisać szereg własnych testów i walidować jej działanie w wydzielonym środowisku, a nie na bazie. Myślę, że procedury w Neo4j warto używać jako pakiet małych funkcjonalności, które ułatwiają nam rutynowe czynności – w ten sposób działa APOC, którego widziałem przy listowaniu SHOW PROCEDURES.

Analiza końcowego zbioru:

Zdaje mi się, że wybrany przez mnie zbiór nie ma jasnego sposobu rozłożenia na wielu maszynach. A projektowanie podziału grafu jest bardzo trudnym zadaniem. Jedyne co zdaje mi się być sensowne to:

- Utworzenie głównego podgrupy składającego się z :Restaurant, :Contact i :Chef – które umieściłbym na najstabilniejszej i najmocniejszej maszynie
- Następnie do oddzielnej maszyny wydzieliłbym :Schedule, :MealOption oraz :Description oraz do oddzielnego podgrafa :User i :Chef, dla których być może zajdzie pewne złączenie w przyszłości
- Z perspektywy czasu pozbyłbym się węzła :Cuisine, ponieważ zawiera tylko nazwę. Zamiast tego do istniejących restauracji dodałbym labele odpowiadające tym rodzajom kuchni, które zawierały w surowych danych jako lista. To prawdopodobnie zwiększyłoby wydajność poprzez pogrupowanie restauracji.

Krytycznym mostem jest (:Review)-[:REVIEWS]->(:Restaurant).

Te rozwiązanie nie jest idealne i należałoby zdecydowanie głębiej zastanowić się jak zamodelować lepiej graf. Zaproponowany podział zawiera kilka mostów, co jest minusem i musimy bazować na bezawaryjności systemu.

Biblioteka APOC:

APOC (Awesome Procedures On Cypher) to biblioteka twórców Neo4j rozszerzająca możliwości systemu. Biblioteka daje ogrom funkcjonalności (kilkadziesiąt grup funkcjonalności) i zawiera bardzo rozbudowaną dokumentację co jest zdecydowaną zaletą. Przedstawiam kilka najciekawszych funkcjonalności:

1. **Import danych** (zarówno z pliku JSON jak i CSV). Z importu skorzystałem w tym ćwiczeniu, aby zasilić bazę wybranymi zbiorami danych. Zasilanie jest intuicyjne i relatywnie proste. Iterujemy wczytując kolekcję i sami definiujemy jak mają być przetworzone dane, które załadujemy jako węzły bądź relacje:

```
1 CALL apoc.load.json("file:///restaurants.json") YIELD value AS restaurant
2 MERGE (r:Restaurant {id: restaurant.id})
3 SET r.name = restaurant.name,
4   r.locality = restaurant.locality,
5   r.region = restaurant.region,
6   r.country = restaurant.country,
7   r.rating = restaurant.rating;
```

Added 147 labels, created 147 nodes, set 882 properties, completed after 108 ms.

Zrzut ekranu nr 51: wczytanie pliku restaurants.json i utworzenie węzłów o ile już nie istnieją w bazie

2. **Triggery.** Przydatna funkcjonalność, gdy chcemy wbudować dodatkową logikę biznesową i pilnować spójności danych. Po wglądzie w dokumentację wniosuję, że działają analogicznie do trigerr'ów w bazach relacyjnych. Choć składnia zdaje się mi być mniej intuicyjna niż w SQL (możliwe, że wynika to z mojego braku doświadczenia w pisaniu zapytań w języku Cypher). Konfiguracja jest prosta, należy w pliku konfiguracyjnym dodać dwa wpisy – dozwolenie trigger'ów oraz interwał, po którym następuje weryfikacja replikacji w klastrze.

```
1 CALL apoc.trigger.add(
2   "incrementReviewCount",
3   "
4     UNWIND $createdRelationships AS rel
5     WITH rel
6     WHERE type(rel) = 'REVIEWS'
7     MATCH (restaurant:Restaurant)-[:REVIEWS]-()
8     WITH restaurant, count(*) AS newReviewCount
9     SET restaurant.reviewsCount = newReviewCount
10    "
11  {phase: 'afterAsync'}
12 )
```

Table	name	query	selector
Text	"IncrementReviewCount"	"UNWIND \$createdRelationships AS rel WITH rel WHERE type(rel) = 'REVIEWS' MATCH (restaurant:Restaurant)-[:REVIEWS]-() WITH restaurant, count(*) AS newReviewCount SET restaurant.reviewsCount = newReviewCount " {phase: 'afterAsync'}	{ "phase": "afterAsync" }

Started streaming 1 records after 1 ms and completed after 111 ms.

Zrzut ekranu nr 52: dodanie trigerr'a afterAsync (po dodaniu recenzji trigger będzie inkrementował licznik recenzji dla restauracji)

```

1 MATCH (r:Restaurant {name: 'Test Restaurant'})
2 MERGE (rev:Review {id: '123456f', content: 'Fantastic service!!!'})-[:REVIEWS]→(r)
3

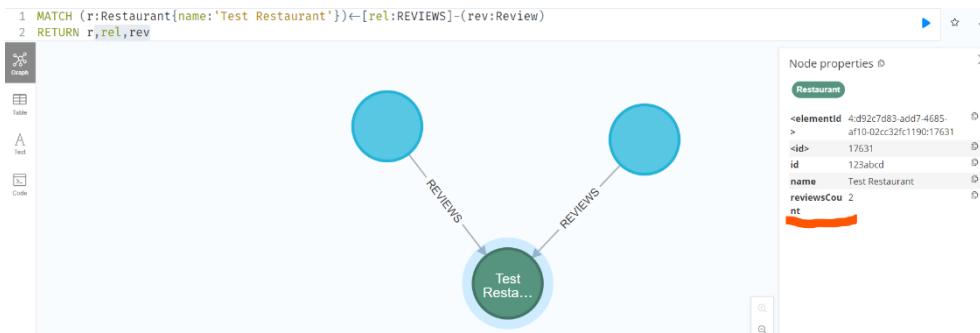
```

	Added 1 label, created 1 node, set 2 properties, created 1 relationship, completed after 25 ms.
--	---

Zrzut ekranu nr 53: utworzenie restauracji z dołączoną recenzją

	neo4j\$ MERGE (rev:Review {id: "123abcde", content: "Great food!"})-[:REVIEWS]→(r:Restaurant {id: "123abcd", name: "Test Restaurant"})
	Added 2 labels, created 2 nodes, set 4 properties, created 1 relationship, completed after 4 ms.

Zrzut ekranu nr 54: dodanie drugiej recenzji



Zrzut ekranu nr 55: weryfikacja działania

3. **Formatowanie dat.** Biblioteka daje wiele możliwości operowania na typie daty. Jako przydatne uznaję możliwość prezentowania daty w różnych formatach w zależności od naszych potrzeb możemy prezentować datę na kilka sposobów.

```

1 MERGE (testNode:TestNode {name: 'test-node', date: '2024-12-15T12:00:00'})
2 RETURN testNode
3

```

Zrzut ekranu nr 56: tworzenie testowego węzła z datą w formacie ISO 8061

```

1 MATCH (testNode:TestNode)
2 WITH testNode,
3     apoc.date.convertFormat(testNode.date, 'yyyy-MM-dd\T\HH:mm:ss', 'MMM dd, yyyy') AS formatted_date_1,
4     apoc.date.convertFormat(testNode.date, 'yyyy-MM-dd\T\HH:mm:ss', 'yyyy-MM-dd HH:mm') AS formatted_date_2
5 RETURN testNode.date AS original_date, formatted_date_1, formatted_date_2
6 LIMIT 1
7

```

	original_date	formatted_date_1	formatted_date_2
	"2024-12-15T12:00:00"	"gru 15, 2024"	"2024-12-15 12:00"

Zrzut ekranu nr 57: prezentowanie sformatowanych dat

4. **Operacje liczbowe.** Dostępnych jest wiele operacji (konwersje, translacje z jednego systemu liczbowego do drugiego, operacje mnożenia/dzielenia, parsowanie etc.). Zaprezentuję konwersję, ponieważ może nam być przydatna do końcowego prezentowania rezultatów odbiorcom (np. wyrównanie i odpowiednie sformatowanie oceny wyrażonej liczbowo).

```

1 MATCH (r:Restaurant)
2 RETURN apoc.number.format(r.rating, '0.00') AS overallRating
3 LIMIT 3

```

The screenshot shows a Neo4j browser interface with a results table titled "overallRating". The table has three rows:

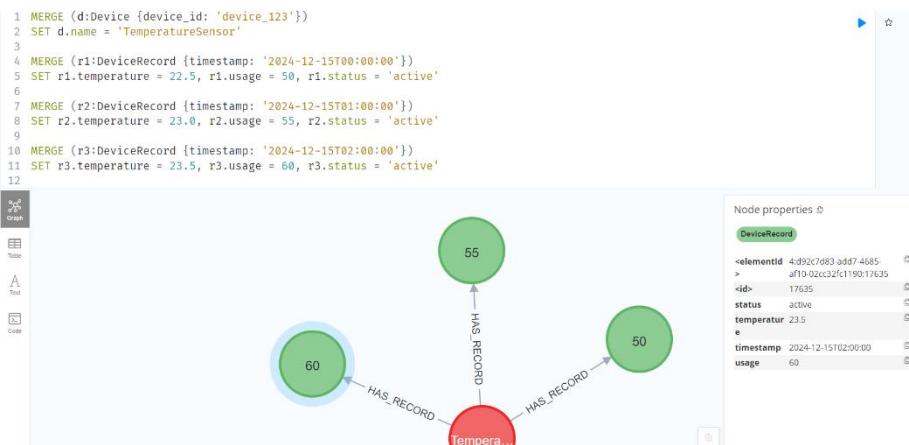
	overallRating
1	"4.00"
2	"4.00"
3	"4.50"

Below the table, a message states: "Started streaming 3 records after 13 ms and completed after 14 ms."

Zrzut ekranu nr 58: jednolicie sformatowane oceny restauracji

5. **Porównanie węzłów.** Może być to bardzo przydatna funkcjonalność w systemach, gdzie monitorujemy czujniki czy urządzenia. Zapisujemy odczyty co stały interwał czasowy i porównujemy dwa ostatnie odczyty. Wówczas możemy bezpośrednio pokazać zmiany, bez dodatkowej potrzeby implementowania tej logiki w wyższych warstwach.

Jako przykład nasze urządzenia co godzinę wysyła payload z informacją o swojej temperaturze i zużyciu podzespołów:



Zrzut ekranu nr 59: dodanie urządzenia i pomiarów

```

1 MATCH (d:Device {device_id: 'device_123'})-[:HAS_RECORD]-(r1:DeviceRecord)
2 WITH d, COLLECT(r1) AS records
3 WITH d, LAST(records) AS latest, LAST(records[0..SIZE(records)-1]) AS previous
4
5 RETURN apoc.diff.nodes(previous, latest) AS differences
6

```

The screenshot shows a Neo4j browser interface with a results table titled "differences". The table contains a single row of JSON data:

```

|differences|
|{|leftOnly: {}, different: {timestamp: {left: "2024-12-15T01:00:00", right: "2024-12-15T02:00:00"}, | |usage: {left: 55, right: 60}, temperature: {left: 23.0, right: 23.5}}, inCommon: {status: "active"} | |, rightOnly: {}|}
|{|}

```

Zrzut ekranu nr 60: porównanie węzłów

Dostajemy bardzo przejrzyste porównanie w wygodnej postaci – jakie właściwości występują tylko w starych/nowym węźle, czym się różnią, jakie właściwości mają takie same.

Wnioski:

Realizacja tego projektu była moim pierwszym kontaktem z bazą grafową. Uznaję to za cenne doświadczenie. Paradymat grafowy jest specyficzny i choć nie jest on raczej wykorzystywany na nie nadaje się do klasycznych (całych) systemów/biznesów to stanowi on bardzo ciekawe rozwiążanie dla problemów z pewnej grupy – rekommendacje, trasy etc. Myślę, że ciekawym pomysłem jest wprowadzenie bazy w systemach modularnych / podzielonych na mikroserwisy, gdzie możemy wydzielić moduł rekommendacji niezależnie i tam zastosować narzędzie dedykowane jak Neo4j – może to nam dać lepsze rezultaty. Samo Neo również oceniam na plus, a w szczególności APOC, które zdaje mi się być narzędziem „must have”, aby pracować z tą bazą.

Źródła:

- <https://neo4j.com/>
- <https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/>
- <https://neo4j.com/docs/cypher-manual/5/styleguide/>
- <https://neo4j.com/labs/apoc/4.1/import/load-json/>
- https://www.youtube.com/watch?v=iyjgOR7nBck&ab_channel=WilliamLyon
- <https://community.neo4j.com/t/how-to-set-apoc-import-file-enable-true/34726/6>
- <https://neo4j.com/docs/browser-manual/current/visual-tour/>
- <https://neo4j.com/docs/cypher-manual/current/indexes/>
- <https://neo4j.com/docs/cypher-manual/current/indexes/search-performance-indexes/managing-indexes/>
- <https://medium.com/@shuv.sdr/query-optimization-in-neo4j-3d2a8b23a8e0>
- <https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128>
- <https://github.com/neo4j-examples/neo4j-procedure-template>
- <https://neo4j.com/labs/apoc/4.1/background-operations/triggers/>
- <https://neo4j.com/labs/apoc/4.3/overview/apoc.date/apoc.date.convertFormat/>
- <https://neo4j.com/labs/apoc/4.3/overview/apoc.diff/apoc.diff.nodes/>