NEO4J

WPROWADZENIE

BAZY GRAFOWE

- Graf jako struktura reprezentujące *dane* i *związki* między nimi
  - **węzły** – przechowują wartości
  - **krawędzie** – reprezentują relacje
- „Silnik” bazy zoptymalizowany pod kątem obsługi „relacji”
- Zastosowane technik grafowych (przechodzenie w głąb, wszerz itd.)
- Relacje
  - zdefiniowane na poziomie **pojedynczych danych** (inaczej niż w bazach relacyjnych)
  - **skierowane** (posiadają „kierunek oddziaływania”)

NEO4J

- Otwarty kod źródłowy (Java + Scala), dostępny w serwisie GitHub
  - dostępne komercyjne rozszerzenia (o zamkniętym kodzie)
- Wsparcie dla
  - transakcji (ACID), indeksów i „więzów” (ang. *constraints*)
  - dużych grafów (34 mld węzłów, 34 mld krawędzi i 68 mld atrybutów)
- **Cypher** jako główny język zapytań
- Wiele zastosowań: wszędzie, gdzie graf jest naturalną reprezentacją danych – systemy rekomendacji, bezpieczeństwa, społecznościowe itd.

WĘZŁY W NEO4J

- **Węzeł**
  - może mieć dowolną liczbę **atrybutów**
  - dane reprezentowane przez pary **klucz-wartość**
  - **identity** (identyfikator węzła) jako atrybut *obowiązkowy*
  - może posiadać **etykiety** (służące do klasyfikacji)

RELACJE W NEO4J

- Odpowiadają **krawędziom** w grafie
- Relacja
  - może posiadać **atrybuty** (podobnie jak węzeł)
  - musi mieć **typ**
  - zawsze jest **skierowana**
- Pomiędzy parą węzłów może zachodzić **dowolna liczba** relacji

```
Osoba PRACUJE_DLA  Firma
```

CYPHER – PODSTAWOWE OPERACJE

- Tworzenie węzłów
  - `CREATE (p:Person { firstName: "Tomasz" })`
  - `CREATE (p2:Person { firstName: "Andrzej"}) (p3:Person { firstName: "Agata"}) RETURN p2,p3`
- Operacje na relacjach
  - `MATCH (tomasz:Person { firstName: "Tomasz"})`
  - `MATCH (andrzej: Person { firstName: "Andrzej"})`
  - `CREATE (tomasz)-[rel:WORKING_WITH]->(andrzej)`
- Pobieranie danych
  - `MATCH` <wyrażenie> `RETURN` <zmienne/identyfikatory>

CYPHER – INFORMACJE OGÓLNE

- Podstawowy język zapytań Neo4J (udostępniony na licencji OpenSource)
- Stworzony na podobieństwo SQL-a
- Zapytania strukturą przypominają (proste) zdania w języku naturalnym
- Dla zwiększenia czytelności zapytań używa ASCII-art
  - przykładowo – kierunek relacji można określić za pomocą znaków <, >

```
(pracodawca) - [:ZATRUDNIA] -> (pracownik)
```

CYPHER – TYPY DANYCH

- **Typy proste**
  - Number (posiada podtypy: Integer i Float)
  - String, Boolean, Point
  - Date, Time, LocalTime, DateTime, LocalDateTime, Duration
- Wartości typów prostych mogą być
  - zwracane przez *zapytania*,
  - używane jako *parametry*,
  - przechowywane jako *wartości atrybutów*

CYPHER – TYPY DANYCH

- **Typy strukturalne**
  - węzły
  - związki
  - ścieżki
- Wartości typów strukturalnych
  - mogą być zwracane przez *zapytania*
  - nie mogą być wykorzystywane jako *parametry*
  - nie mogą być przechowywane jako *wartości atrybutów*

CYPHER – TYPY DANYCH

- **Typy złożone**
  - listy
  - odwzorowania (klucze są typu String, a wartości – jakiegokolwiek typu)
- Wartości typów złożonych (podobnie jak strukturalnych)
  - mogą być zwracane przez *zapytania*
  - nie mogą być wykorzystywane jako *parametry*
  - nie mogą być przechowywane jako *wartości atrybutów*

KONWENCJA NAZEWNICZA

- Węzły nazywamy używając *CamelCase*
  - `ItCompany`, `UniversityOfGdańsk`
- Nazwy relacji zapisujemy wielkimi literami, oddzielając ewentualne człony znakiem „_”
  - `PRODUCED_BY`, `ACTED_IN`
- Właściwości (atrybuty) nazywamy stosując *lowerCamelCase*
  - `firstName`, `lastName`, `dateOfBirth`