# Neo4J - Graph DB

## NoSQL baze podataka

Univerzitet u Novom Sadu
Fakultet tehničkih nauka

# Native vs Non-native

Graf baze možemo podeliti u 2 grupe:

1. Native graf baze - skladište i operacije su **osmišljene** da rade sa grafovima
2. Non-native graf baze - skladište i operacije su **prilagođene** radu sa grafovima

# Native vs Non-native

Native graf baze

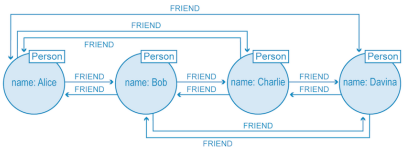
- U skladištu se čuvaju grafovi u izvornom obliku - skup povezanih čvorova
- Čvorovi čuvaju informaciju o svojim susedima - brz pristup na osnovu veza
- Ne zahteva dodatne strukture za nove načine prolaska kroz graf
- Brže izvršavanje, manje zauzeće memorije, **pravljano namenski za grafove**
- Lako skaliranje sa povećanjem podataka

Non-native graf baze

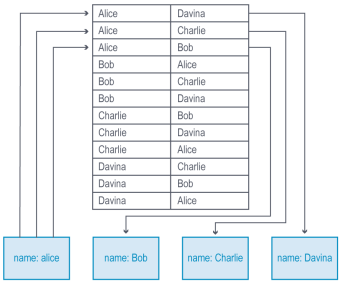
- Skladište je najčešće baza opšte namene (relaciona, wide-column)
- Upotreba dodatnih indeksa za pristup susednim čvorovima
- Zahteva kreiranje novih indeksa za svaki novi način prolaska kroz graf
- Poznata struktura razvojnom timu
- Loše skaliranje sa povećanjem podataka (zahteva 2 do 4 puta više hardverske moći)

# Native vs Non-native - strukture

- Native:




- Non-native:




- Više informacija na *linku*

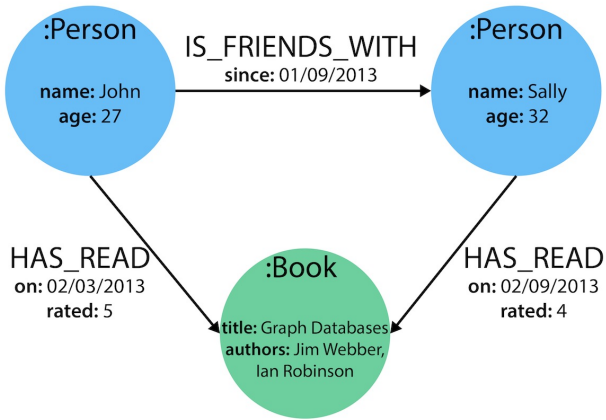
# Neo4J

- *Zvanična dokumentacija*
- *Go driver*

# Osnovne informacije

- Native graf baza podataka
- White-board friendly - logički dijagram se jednostavno preslikava na samu bazu
- Skalabilna
- Podržan ACID
- Jednostavna i raznovsna podrška za upite
- *Shemaless* model podataka - ne zahteva definisanje šeme
- Mogućnost proširenja i dodatka ograničenja nad modelom podataka
- Čvorovi i grane grafa su *top-level entiteti*
- Svojstven upitni jezik - *Cypher*

# Primer grafa



# Terminologija

- **Graf** = povezana struktura čvorova
- **Node** = čvor u grafu
  - predstavlja enitet
  - odgovara torki u relacionoj bazi
  - ima *id*
  - može imati *labele* i *svojstva*
- **Relationship** = grana u grafu
  - predstavlja **usmerenu** vezu između 2 čvora
  - ima *id*
  - može imati *labele* i *svojstva*
- **Label** = labela; identifikuje tip čvora ili veze
- **Properties** = svojstva; ključ-vrednost parovi koji dodatno opisuju čvorove i veze

# Tipovi podataka

- **Svojstva**
  - Number (Integer, Float), String, Boolean
  - *Point* - prostorna tačka
  - Date, Time, Duration, LocalTime, DateTime
- **Strukturalni elementi**
  - Node, Relationship
  - *Path* - putanja između čvorova
- **Složeni tipovi**
  - List, Map

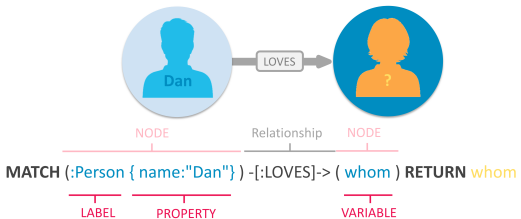
# Model podataka

- Sve informacije potrebne za rad sistema moraju se nalaziti u grafu!
- Da li će podatak biti predstavljen kao čvor ili kao svojstvo?
  - Zavisi od načina na koji ćemo pristupati podatku
- Da li možemo da pratimo bitne veze između podataka kroz graf?
  - Želimo da dođemo od čvora A do čvora B ako postoji smislena veza između njih
- Čvor može imati veći broj labela
  - Korisno ako jedan podatak možemo svrstati u više grupa

# Cypher - Graph Query Language

- Upitni jezik optimizovan za rad sa grafovima
- Jednostavna *slikovita* sintaksa
- Velika raznovrsnost upita i njihovih kombinacija
- *Cheat-sheet sintake*
- **Naredbe su case-insensitive dok su labele i svojstva case-sensitive!**

# Cypher sintaksa




- **()** - zagrade predstavljaju čvorove
- **-[]->** - usmerene linije predstavljaju grane
- **{}** - vitičaste zagrade predstavljaju svojstva
- **:Naziv** - labele se označavaju sa početne :
- **docker exec -it ime_kontejnera cypher-shell -u username -p password** - povezivanje na *Cypher* u okviru pokrenutog Docker kontejnera

# Create

Kreiranje čvora:

```
CREATE (pera:Person {name: "Pera", surname: "Peric", age: 23})
```

Kreiranje grane između 2 čvora:

```
CREATE (pera) -[:IS_FRIENDS {since: "2007-09-03"}]-> (mika)
```

# Read - na osnovu ID-a i svojstva

Pronalazak čvora na osnovu polja uz projekciju svojstava:

```
MATCH (n {name: "Pera", surname: "Peric"})
RETURN n.name, n.surname, n.age
```

Pronalazak čvora na osnovu ID-a:

```
MATCH (n)
WHERE ID(n) = 123
RETURN n
```

# Read - na osnovu grana

Pronalazak svih čvorova sa labelom Person koji su povezani granom sa labelom IS_FRIENDS:

```
MATCH (p1:Person) -[r:IS_FRIENDS]-> (p2:Person)
RETURN p1, r, p2
```

Pronalazak svih čvorova na osnovu svojstva grane:

```
MATCH (p1:Person) -[r:IS_FRIENDS {since: "2007-09-03"}]-> (p2:Person)
RETURN p1, r, p2
```

# Read - rekurzija i agregacije

Pronalazak svih čvorova koji su prijatelji prijatelja do dubine 3:

```
MATCH (p1:Person) -[r:IS_FRIENDS *1..3]-> (p2:Person)
RETURN p1, r, p2
```

Prebrojavanje svih čvorova koji su u vezi sa Perom:

```
MATCH (p:Person {name: "Pera Peric"})<--(x)
RETURN p.name, count(x)
```

# Read - putanje

Pronalazak najkraće putanje između 2 čvora do maksimalne dubine 3:

```
MATCH (pera:Person {name: "Pera Peric"}), (zika:Person {name: "Zika Zikic"})
RETURN shortestPath((pera)-[*..3]-(zika))
```

# Update

Izmena vrednosti svojstva čvora:

```
MATCH (p:Person)
WHERE p.name = "Pera"
SET p.age = 28
```

Dodavanje labele čvoru:

```
MATCH (n)
WHERE ID(n) = 123
SET n:Person
```

# Update - Merge

Podrška za *MERGE* operaciju:

```
MERGE (n:Person {name: "Pera"})
    ON CREATE SET
          n.created = timestamp()
    ON MATCH SET
          n.counter = coalesce(n.counter, 0) + 1,
          n.accessTime = timestamp()
```

- Pronalazi čvor sa labelom Person po imenu
- Ako traženi čvor **ne postoji** = **ON CREATE**, kreira ga i postavlja *created* svojstvo na trenutno vreme
- Ako traženi čvor **postoji** = **ON MATCH**, povećava mu vrednost svojstva *counter* i postavlja *accessTime* svojstvo na trenutno vreme
- Funkcija coalesce vraća prvu *ne-null vrednost* od prosleđenih

# Delete

Brisanje čvora koji nema povezanih grana:

```
MATCH (p:Person)
WHERE p.name = "Pera"
DELETE p
```

Brisanje čvora i povezanih grana:

```
MATCH (n)
WHERE ID(n) = 123
DETACH DELETE n
```

# Ograničenja i indeksi

Kreiranje ograničenja (automatski dodaje indeks po tom polju):

```
CREATE CONSTRAINT FOR (p:Person) REQUIRE (p.name) IS UNIQUE
```

Kreiranje indeksa:

```
CREATE INDEX FOR (p:Person) ON (p.surname)
```

# Primer

U okviru primera *RestNeo4J* upotrebljeni su:

1. *Neo4J* - baza podataka
2. *Cypher skripta* - upiti nad bazom
3. *Docker* - kontejnerizacija rešenja (i "instalacija" baza)
4. *Go* - implementacija primera

# Zadaci

- Proširiti servis tako da podržava dodavanje novog filma
- Proširiti servis tako da podržava dobavljanje svih osoba koje su glumile u filmu i režirale isti taj film
- Proširiti servis tako da podržava dobavljanje prvih *N* najskorijih filmova u kojima je glumio *Keanu Reeves*
- Proširiti servis tako da podržava dobavljanje najkraće putanje između filmova *The Matrix* i *When Harry Met Sally*
- Proširiti servis tako da podržava dobavljanje prvih *N* glumaca sa najvećim brojem uloga u filmovima
- **Bonus**
  - Isprobavajte različite upite, igrajte se sa podacima :)