

# **Graflowe bazy danych**

**na podstawie Neo4j**

**Mateusz Piech**

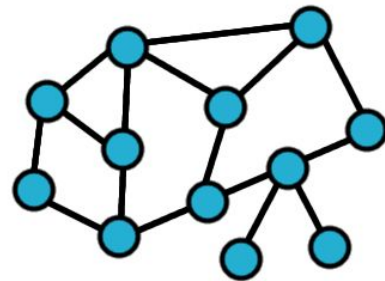
# Agenda

1. Grafy - teoria
2. NoSQL
3. Przechowywanie grafu w relacyjnej bazie danych
4. Grafowe bazy danych
5. Grafowy model danych
6. Graph-Oriented Object Database Model
7. Grafowe bazy danych vs RDBMS
8. Neo4j
9. Języki zapytań:
  - a. Cypher
  - b. Gremlin
10. Neo4j w akcji
11. Laboratorium
12. Podsumowanie



# Grafy

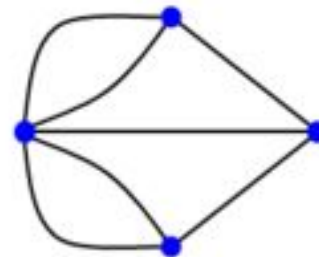
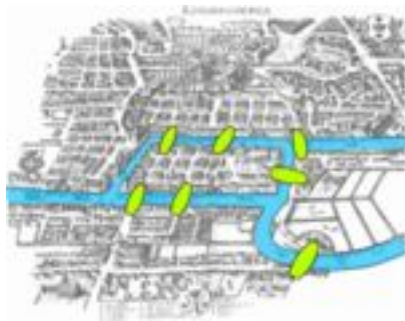
## Czym jest graf?



“Graf to zbiór wierzchołków, które mogą być połączone krawędziami w taki sposób, że każda krawędź kończy się i zaczyna w którymś z wierzchołków”

Graf jest jedną z podstawowych abstrakcji w informatyce

## Zastosowanie



Najstarszy przykład zastosowania grafów w rozwiązaniu problemu mostów królewskich, Leonhard Euler w 1736

## Zastosowanie

- sieci dróg, korytarzy - GPS
- algorytm wyznaczania trasy w internecie
- drzewa
- sieci społecznościowe
- ...

## Podstawowe pojęcia

- cykl - kolekcja połączonych wierzchołków, gdzie pierwszy element jest taki sam jak ostatni
- droga/ścieżka - kolekcja wierzchołków lub krawędzi
- długość drogi
- drzewo spinające graf - drzewo, które zawiera wszystkie wierzchołki grafu  $G$ , zaś zbiór krawędzi drzewa jest podzbiorem zbioru krawędzi grafu.
- gęstość grafu - stosunek ilości wierzchołków do krawędzi
- stopień wierzchołka - ilość krawędzi wychodzących z wierzchołka
- klika - podzbiór wierzchołków, gdzie istnieje połączenie pomiędzy każdym z każdym

## Podział grafów

### Rodzaje:

- nieskierowany
- skierowany
- mieszany
- z wagami
- ...

### Klasy:

- graf spójny
- graf acykliczny
- graf cykliczny
- drzewo
- ...



## Algorytmy / słowa kluczowe

- algorytmy A\*, Dijkstra, Bellman-Ford, Floyd-Warshall,
- przeszukiwania grafu w głąb i wszerz,
- "clusters"
- "connected components"
- "small worlds"



# NoSQL

## Czym jest NoSQL?

- non SQL (not only SQL)
- nie eliminuje klasycznego modelu, a jedynie go ulepsza
- systemy bazodanowe, które przechowują dane w nierelacyjny sposób
- Nie ma ACID, jest BASE
- Możliwość skalowania horyzontalnego i wertykalnego

## Rodzaje baz NoSQL

- Klucz - wartość
- Bazy kolumnowe
- Bazy dokumentowe
- Bazy grafowe
- ....

## Dlaczego?

- eksplozja sieci społecznościowych (Facebook, Twitter) - potrzeba operowania na b. dużej ilości danych
- rozwój technologii cloud-based (cloud computing, cloud storage - Amazon S3)
- wraz z większym wykorzystaniem języków dynamicznych (Ruby, Python, Groovy) zaczęto częściej używać dynamicznie typowanych danych z częstymi zmianami schematu
- open source community

## Zalety i wady

### Zalety:

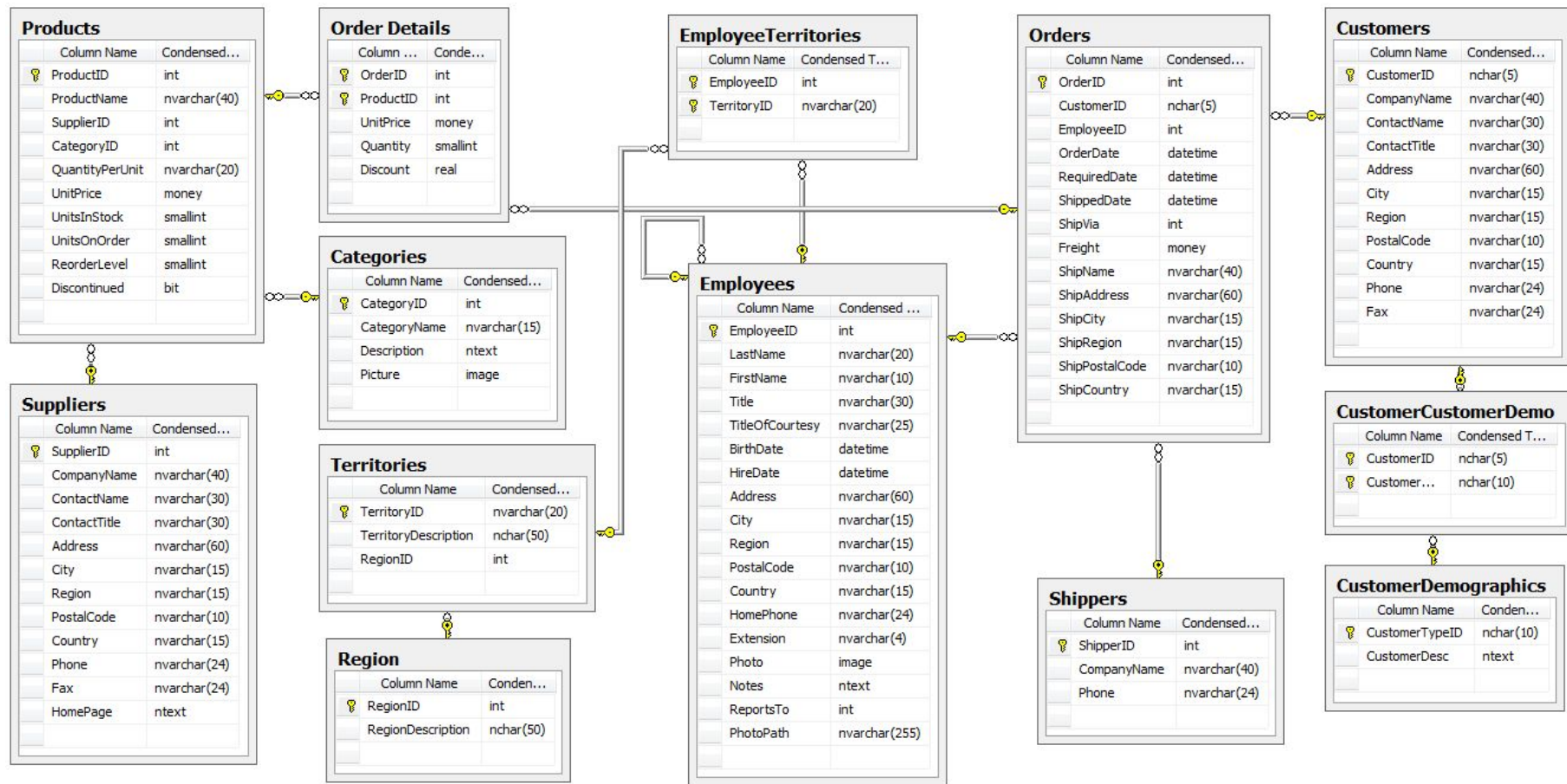
- nie ma schematu
- skalowalność
- open source

### Wady:

- narzędzia ciągle się rozwijają
- duplikacja danych
- brak standardu języka zapytań
- trudne w modelowaniu złożonych struktur



# **Relacyjne bazy danych**



Schemat = graf



## Wady i zalety

### Wady:

- brak skalowalności
- wydajność JOINów
- trudne tworzenie zapytań

### Zalety:

- dowolny schemat danych
- brak redundancji danych
- normalizacja
- ACID



# **Grafowe bazy danych**

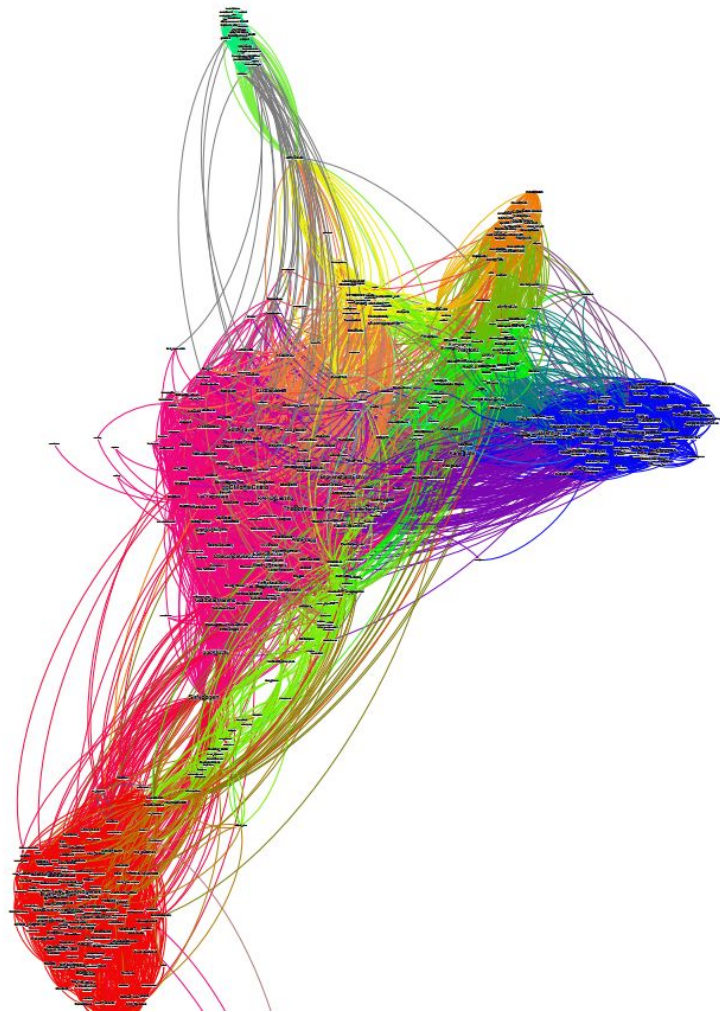
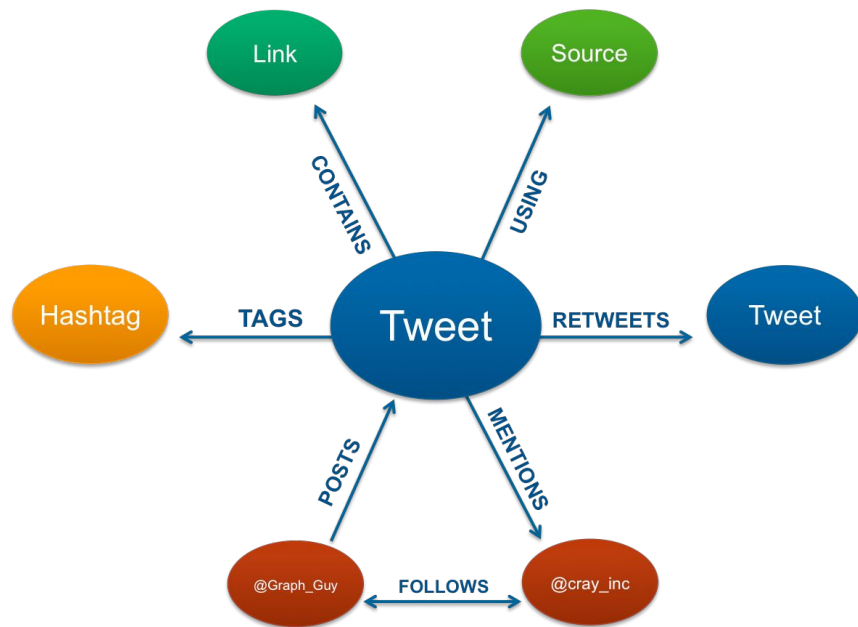
## Grafowa baza danych

“Baza danych wykorzystująca struktury grafów z węzłami, krawędziami i własnościami do przedstawiania i przechowywania danych oraz do obsługi zapytań semantycznych. Bazą taką jest każdy system pamięci masowej, który zapewnia bezindeksowe sąsiedztwo, co oznacza że każdy element bazy zawiera bezpośredni wskaźnik na sąsiadujące elementy i nie jest konieczne wyszukiwanie indeksowe.”

## Geneza

- Zainteresowanie rozwinęło się na początku lat 90
- Później wygasło ze względu na modele semistrukturalne (XML, HTML)
- Rozwój internetu spowodował wzrost danych
- Większa liczba danych = większa liczba relacji między danymi
- Najłatwiej ogarnąć to grafem
- Relacyjne bazy danych potrafią przechować graf, jednak nie jest to naturalne

# Przykłady



## Zalety grafowych baz danych

- skalowalność - możliwość rozproszenia na wielu serwerach
- big data
- dynamiczna struktura
- generyczne typy węzłów i krawędzi
- brak JOINów



“Graf bardziej naturalnie opisuje relacje”



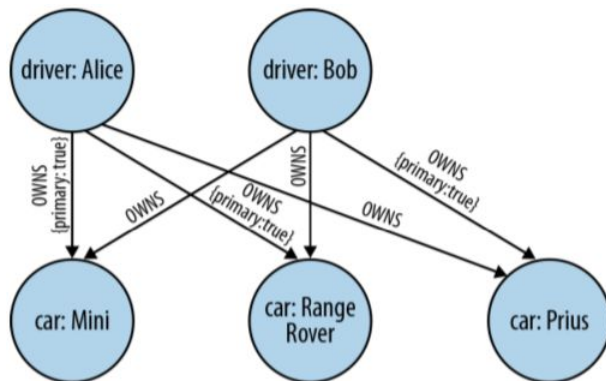
# **Grafowy model danych**



# Struktury danych

## Encja (węzeł, obiekt)

- reprezentuje pojedynczy byt
- własności opisane przez atrybuty



## Relacja (krawędź)

- własność pomiędzy węzłami
- posiada etykietę
- może mieć kierunek
- możliwe wiele relacji pomiędzy tymi samymi obiektami

## Więzy integralności

Więzy integralności są ogólnymi instrukcjami i regułami, które definiują zbiór spójnych stanów bazy danych i/lub zmian stanów.

W przypadku grafowych baz danych wyróżniamy następujące więzy integralności:

- Spójność schema-instancja
- Tożsamość obiektu oraz integralność referencyjna
- Zależności funkcyjne

## Spójność schemat-instancja

Grafowe bazy danych często (ale nie zawsze) definiują schemat bazy (ang. schema), który konkretna instancja bazy danych musi spełniać.

Spójność schemat-instancja w ogólności podporządkowuje się poniższemu wytycznym:

- instancja powinna zawierać tylko encje i relacje o typach zdefiniowanych w schemacie
- encja w instancji może mieć wyłącznie te relacje lub właściwości, które są zdefiniowane dla typu tej encji. Także wszystkie etykiety węzłów i krawędzi muszą wystąpić w schemacie.

### **Oddzielenie schematu i instancji (ang. schema-instance separation)**

Określa stopień, w jakim schemat i instancja są rozróżniane w bazie danych. Model danych może być zaklasyfikowany jako strukturalny (structured) lub niestukturalny (unstructured) w zależności od tego, czy pozwala lub nie pozwala na definicję schematu w celu ograniczenia instancji bazy danych do dobrze określonych typów danych. W większości baz danych występuje rozróżnienie schematu i instancji.

## Tożsamość obiektu oraz integralność referencyjna

W zorientowanych obiektowo bazach danych (jakimi są grafowe bazy danych) tożsamość obiektu jest niezależna od wartości atrybutów i jest osiągnięta przez zapisanie w każdym obiekcie w bazie unikalnego identyfikatora, np. etykiety.

Integralności encji (entity integrity) zapewnia, że każdy węzeł grafu jest unikalną encją identyfikowaną przez jej kontekst. Integralność referencyjna (referential integrity) wymaga istnienia odniesienia tylko do istniejących encji.

Te ograniczenia odpowiadają więzom integralności klucza podstawowego oraz klucza obcego (referencyjnego) w relacyjnych bazach danych.

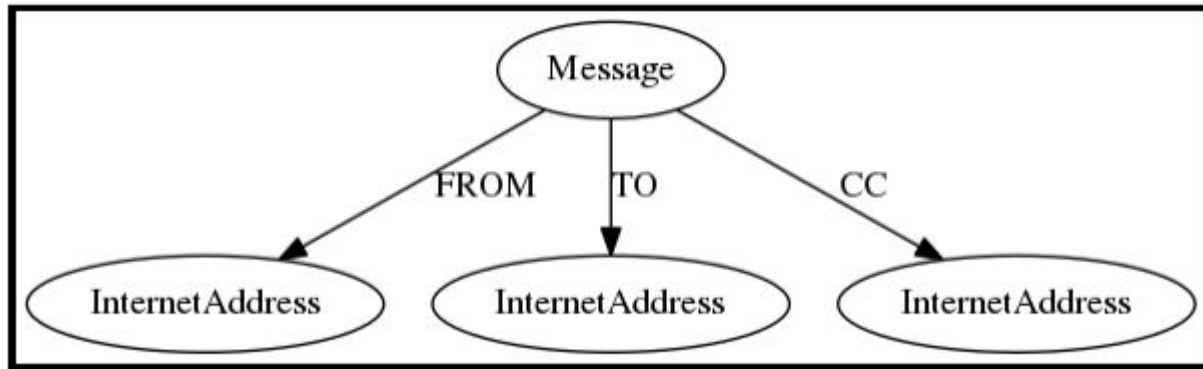
## Zależności funkcyjne

Zależności funkcyjne są zależnościami semantycznymi. Zależność funkcyjna  $A \rightarrow B$ , gdzie  $A$  i  $B$  są zbiorami atrybutów wyraża, że  $A$  określa wartość  $B$  we wszystkich węzłach bazy danych.

Semantyczne więzy integralności na poziomie schematu bazy mogą być reprezentowane przez skierowane krawędzie.

## Jak modelować

- Przygotować diagram relacji
- Zasada: obiekt - węzeł, zdarzenie - krawędź (inaczej duplikacja)
- Implementacja diagramu w bazie



## Języki zapytań

Uwzględniają operacje przechodzenia grafów - graph traversal

Podział na zapytania strukturalne i zapytania o dane.

Zapytania strukturalne:

- Znajdź węzły które nie mają krawędzi
- Znajdź wszystkie węzły zaczynając od A które mają wartość  $Y > Z$

Zapytania o dane:

- Zlicz liczbę węzłów, których atrybut P ma wartość R

## Języki zapytań - przykłady

**Gremlin** - jest to grafowy język programowania. Posiada wbudowane mechanizmy zapytań grafowych, analizy i manipulacji grafami. Jego obsługę posiada m.in. Neo4j.

**SPARQL** - umożliwia wyrażanie zapytań dotyczących różnorodnych źródeł danych, gdzie dane są przechowywane w RDF. Wynikiem zapytań SPARQL jest zbiór grafów RDF. Obsługa np. w AllegroGraph.

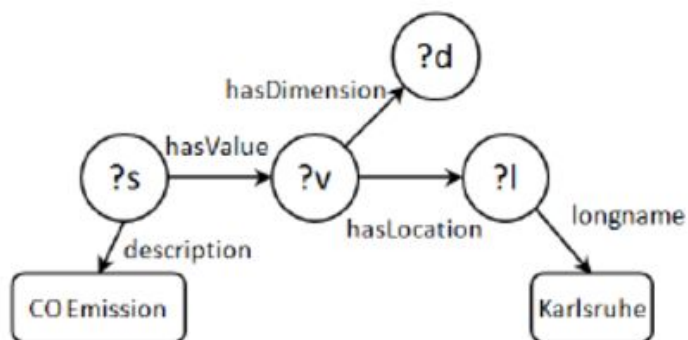
**G, G+, GraphLog** - Języki te umożliwiają tworzenie tzw. grafów zapytań (query graph). Zapytanie grafowe w przypadku języka G wyrażone jest zbiorem etykietowanych skierowanych multigrafów, których węzły mogą być zmiennymi lub stałymi. Wynikiem zapytania jest suma (union) wszystkich grafów zapytań, które dopasowano do podgrafów z instancji grafu (bazy).

**G-Log** - zawiera język deklaratywny do złożonych obiektów. Używa logiczną notację spełnienia reguł (rule satisfaction) do wyznaczenia odpowiedzi na zapytanie.

**Glide** - zapytania są wyrażone z użyciem liniowej notacji składającej się z wyrażień regularnych (etykiet i wild-cards).



## Języki zapytań - przykłady



(a) Query Graph

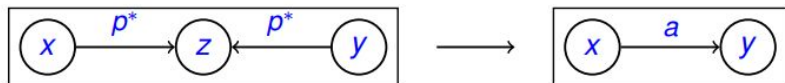
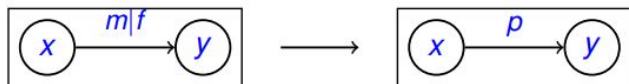
```
select ?s,?v,?d,?l
WHERE {
  ?s ns:description "CO Emission" .
  ?s ns:hasValue ?v .
  ?v ns:hasDimension ?d .
  ?v ns:hasLocation ?l .
  ?l ns:longname "Karlsruhe"
}
```

(b) SPARQL Query

# Języki zapytań - przykłady

## $G, G^+$ Example

- ▶ given a graph
  - ▶ nodes representing people
  - ▶ edges labelled with  $m$  (for *motherOf*) and  $f$  (for *fatherOf*)
- ▶ following query finds parents followed by pairs of people who have a common ancestor



## Języki zapytań - przykłady

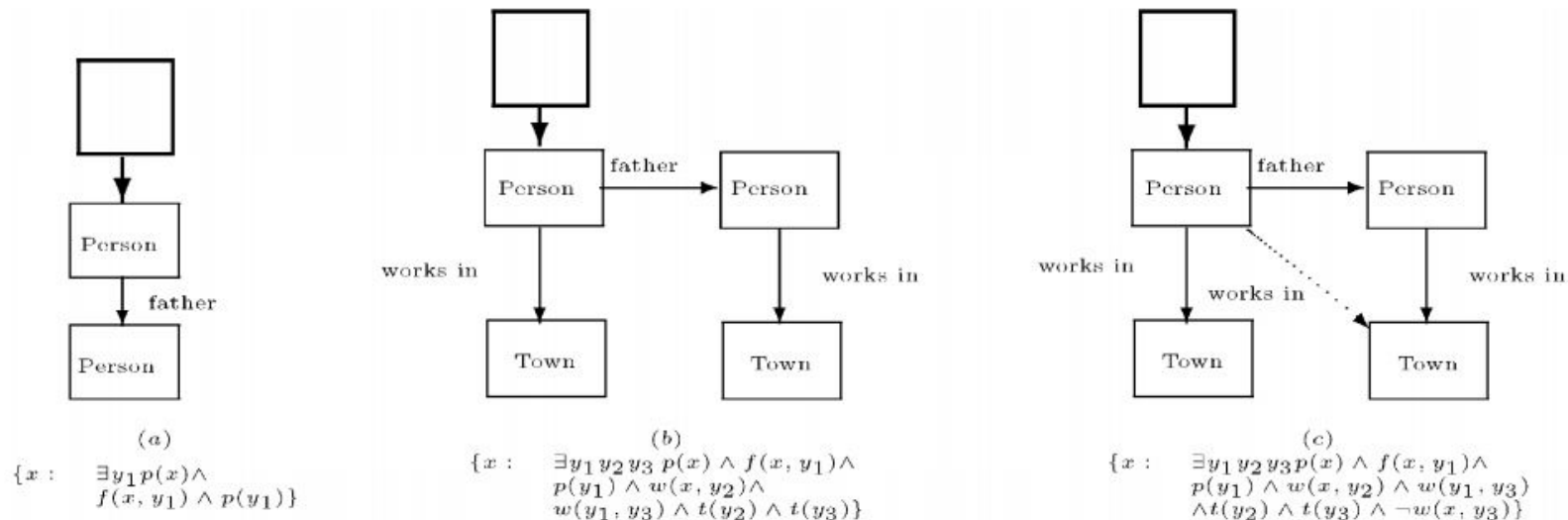


Fig. 1. Sample queries.

## Zastosowania

- uogólnienie klasycznego modelu bazodanowego
- dla bardzo złożonych danych
- reprezentacja wiedzy
- sieci:
  - społecznościowe
  - technologiczne
  - biologiczne

## Rzeczywiste zastosowania



- Twitter używa bazy danych FlockDB do przechowywania grafu społeczności - zawierającego informacje kto śledzi kogo, kto blokuje kogo. W kwietniu 2010 w bazie FlockDB Twittera było ponad 13 bilionów krawędzi, a ilość operacji osiągała 20 tysięcy zapisów/sekundę i 100 tysięcy odczytów/sekundę.
- [\*\*https://neo4j.com/graphgists/\*\*](https://neo4j.com/graphgists/)





# **Graph-Oriented Object Database Model**

## Struktura danych

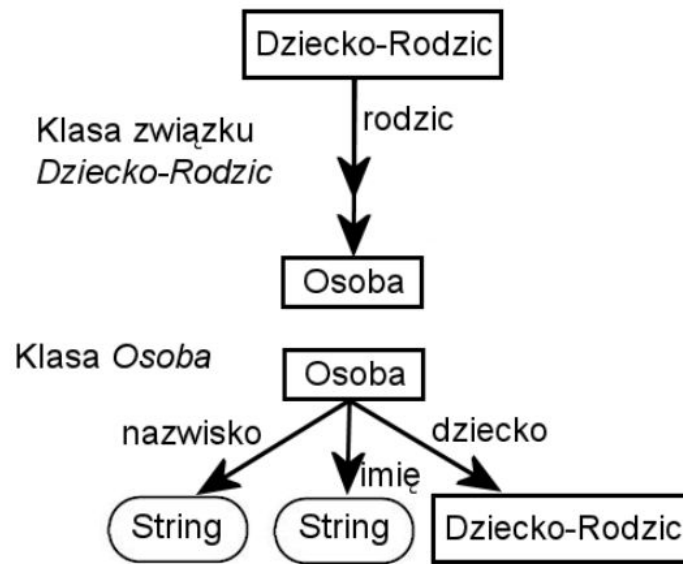
Skierowany graf o dwóch typach wierzchołków:

- „atomowych” (printable), reprezentujących dane liczbowe, stringi, daty, ale także bitmapy oraz pliki dźwiękowe, etykietowanych typem danych, ozn.: 
- „nieatomowych” (non-printable), reprezentujących obiekty abstrakcyjne, etykietowanych klasą obiektu, ozn.: 

oraz dwóch typach krawędzi, etykietowanych nazwą związku lub atrybutu:

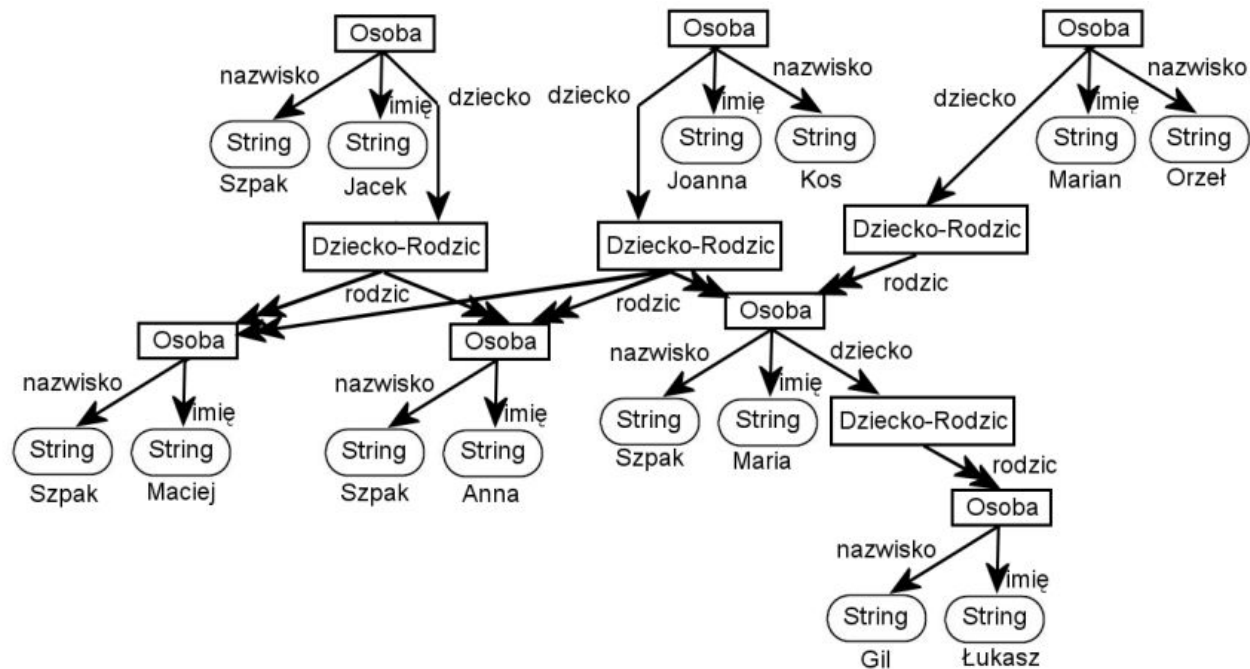
- reprezentujących związki funkcyjne, ozn.: 
- reprezentujących związki jeden-do-wielu, ozn.: 

## Przykładowy schemat BD



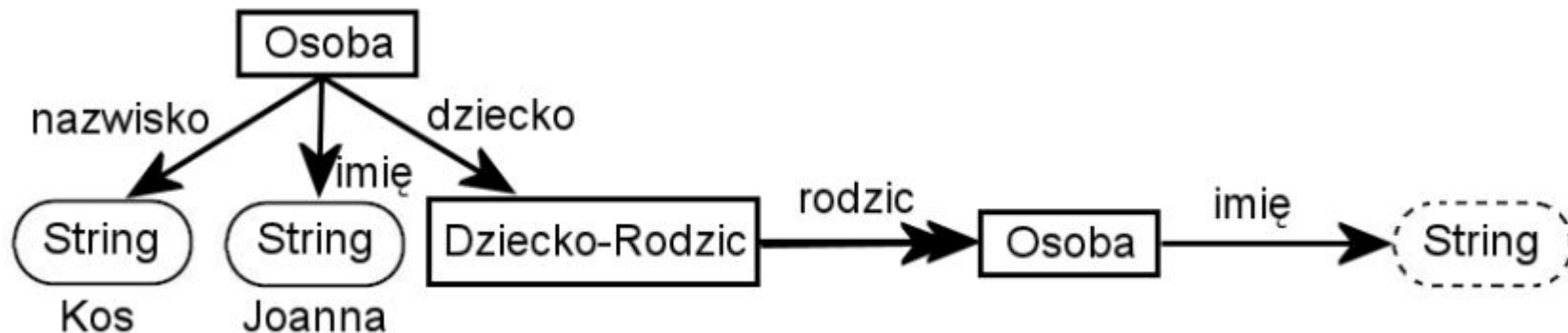


## Przykładowa BD



## Operacje na danych

„Grafowe” operacje na danych - dopasowywanie wzorca (pattern matching) podgrafu. Przykład: zapytanie o imiona dzieci Joanny Kos:

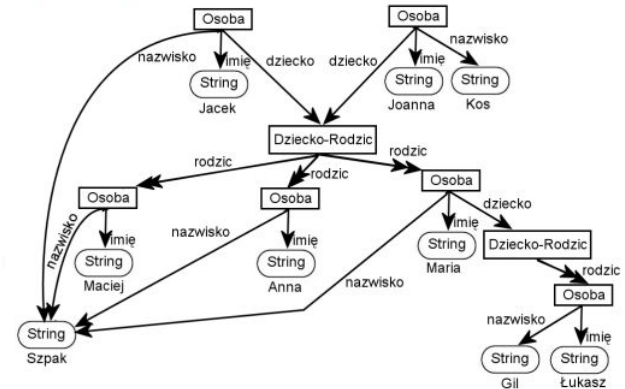


# Operacje na danych

Modyfikacje danych oparte o pattern matching:

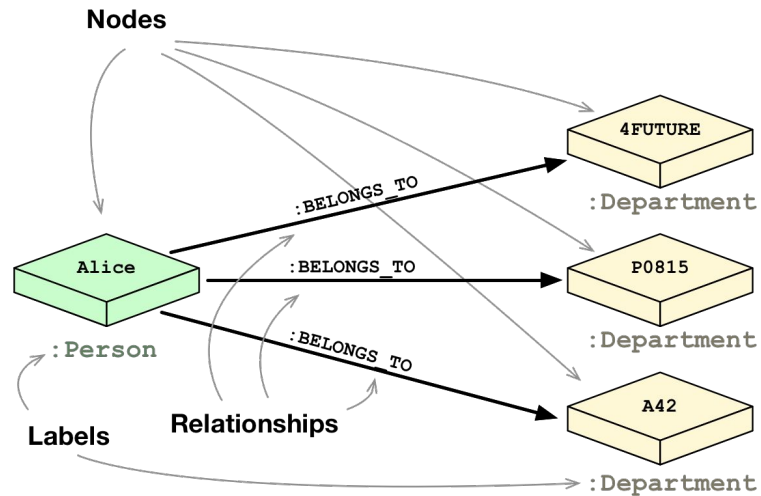
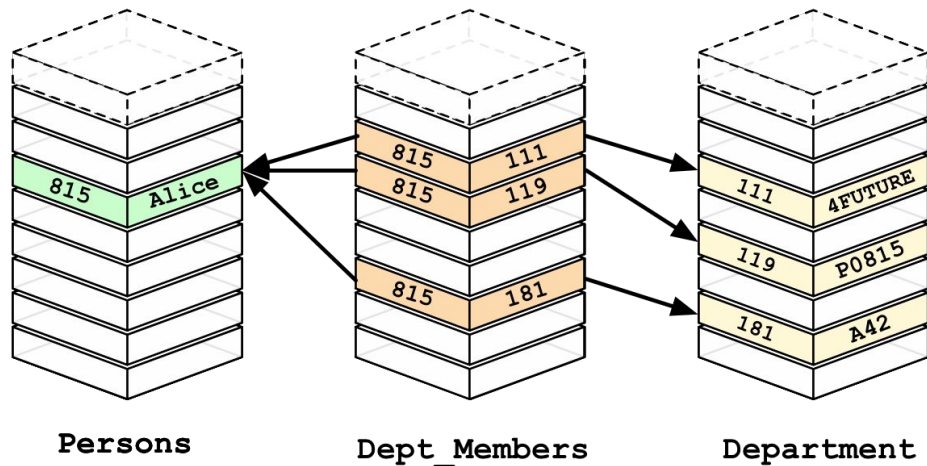
- Dodawanie i kasowanie wierzchołków
- Dodawanie i kasowanie krawędzi
- Abstrakcja

**Abstrakcja - grupowanie duplikatów**

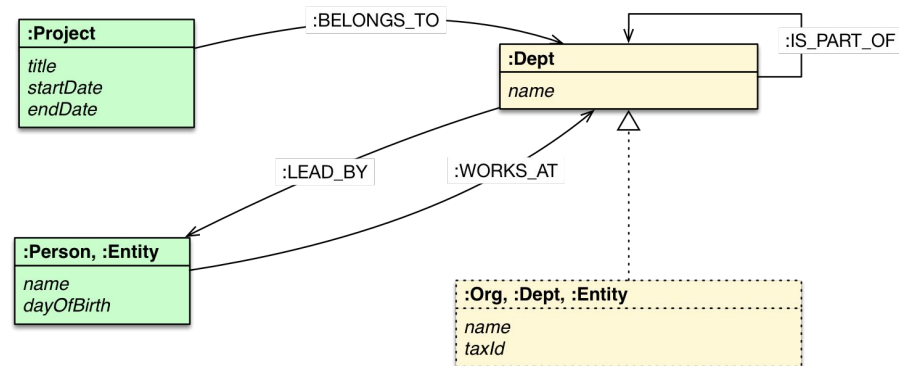
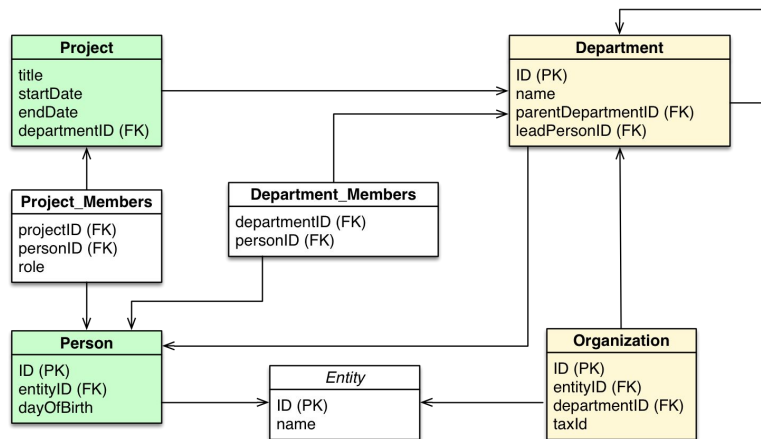


# GDB vs RDBMS

## Porównanie modeli



# Porównanie modeli



## Transformacja modeli

- Każda tabela jest reprezentowana jako etykieta na węzłach
- Każda krotka staje się węzłem
- Kolumny stają się właściwościami węzła
- Usunięcie kluczy głównych
- Zachowanie ograniczeń i indeksów
- Klucze obce zamieniane będą na relacje
- Usuwanie danych z domyślnymi wartościami
- Denormalizacja danych (mogą być powtórzenia danych)
- Tablice złączenia zamieniane są na relacje (krawędzie), kolumny w tych tabelach stają się właściwościami relacji

## Porównanie zapytań

```
SELECT name FROM Person
LEFT JOIN Person_Department
  ON Person.Id =
  Person_Department.PersonId
LEFT JOIN Department
  ON Department.Id =
  Person_Department.DepartmentId
WHERE Department.name = "IT
Department"
```

```
MATCH
(p:Person)<-[:EMPLOYEE]-(d:Depart
ment)
WHERE d.name = "IT Department"
RETURN p.name
```



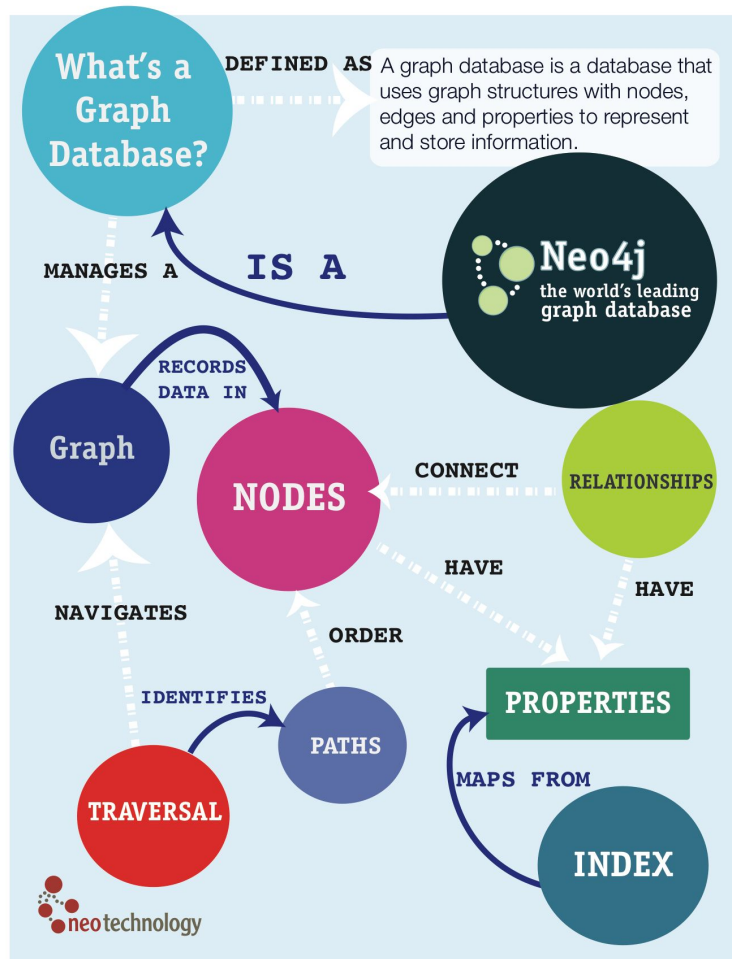
## Wydajność

- prosty graf sieci społecznej
  - około 1000 osób
- średnio 50 znajomych
- $\text{path}(a,b)$  with depth max 4

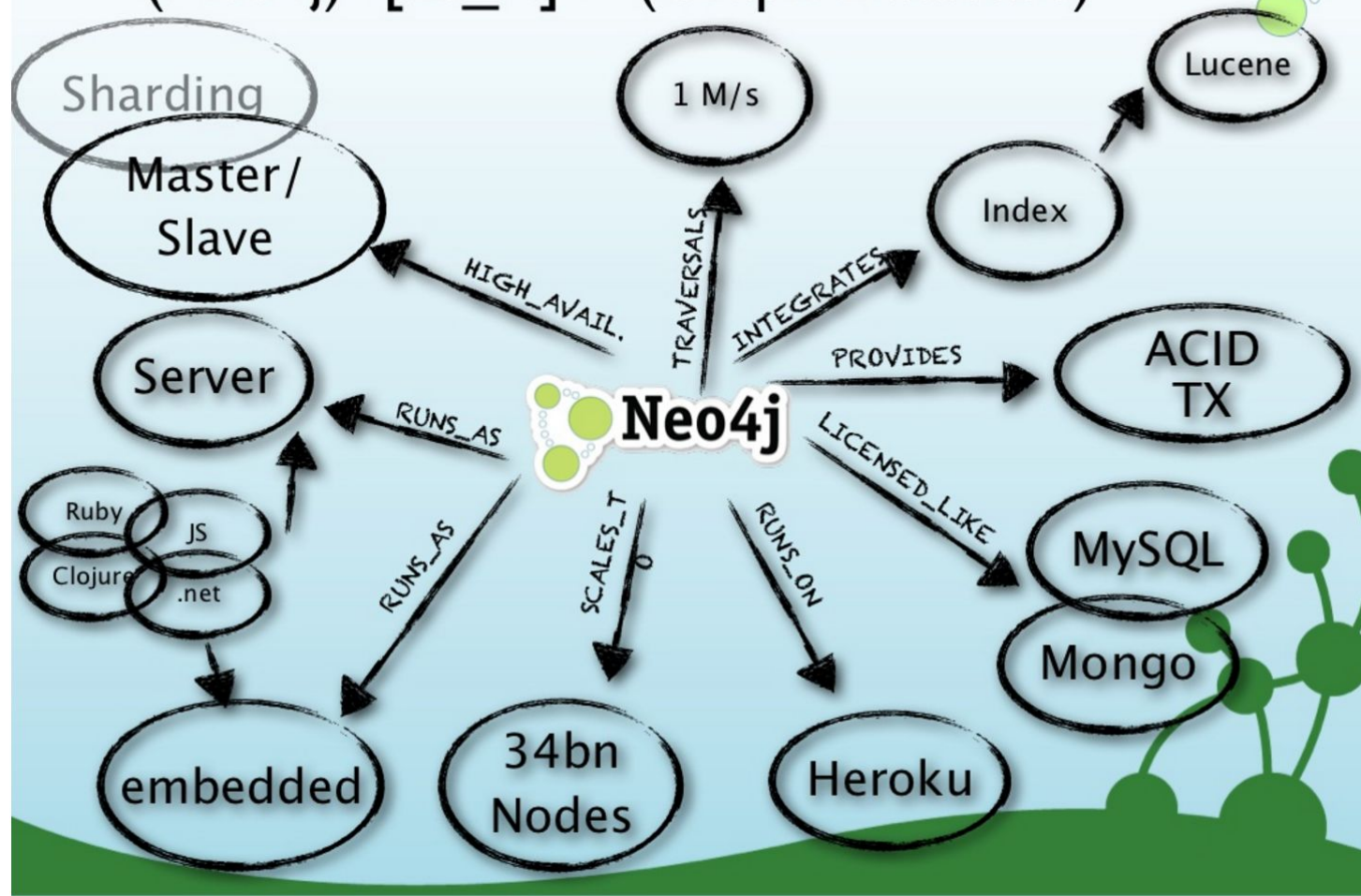
	Liczba osób	Czas zapytania
RDBMS	1000	2000ms
Neo4j	1000	2ms
Neo4j	1000000	2ms



# Neo4j



# (Neo4j) -[:IS\_A]-> (Graph Database)



# Neo4j

## **Grafowa** baza danych:

- graf z właściwościami dla dynamicznego schematu
- idealny dla złożonych, mocno połączonych danych
- zaimplementowane algorytmy grafowe

## Grafowa **baza danych**:

- zapewnia BASE (prawie ACID z eventual consistency)
- mocno skalowalna
- szybka w wędrowaniu po schemacie
- serwer z REST API lub lokalna wersja embedded w JVM
- wysoka wydajność

# Struktura bazy danych

- Node
- Relationship
- Relationship Type
- Property
- Property Index

Nazwa	Data modyfikacji	Typ	Rozmiar
dbms	25.04.2016 19:36	Folder plików	
index	25.04.2016 19:36	Folder plików	
schema	25.04.2016 19:42	Folder plików	
messages	25.04.2016 23:45	Dokument tekstowy	42 KB
neo4j.properties	25.04.2016 19:36	Plik PROPERTIES	2 KB
neostore	25.04.2016 20:08	Plik	8 KB
neostore.counts.db.a	25.04.2016 20:08	Plik A	2 KB
neostore.counts.db.b	25.04.2016 20:03	Plik B	1 KB
neostore.id	25.04.2016 19:36	Plik ID	1 KB
neostore.labeltokenstore	25.04.2016 19:41	Data Base File	8 KB
neostore.labeltokenstore.db.id	25.04.2016 19:36	Plik ID	1 KB
neostore.labeltokenstore.db.names	25.04.2016 19:41	Plik NAMES	8 KB
neostore.labeltokenstore.db.names.id	25.04.2016 19:36	Plik ID	1 KB
neostore.nodestore	25.04.2016 23:45	Data Base File	3 520 KB
neostore.nodestore.db.id	25.04.2016 19:36	Plik ID	1 KB
neostore.nodestore.db.labels	25.04.2016 19:36	Plik LABELS	8 KB
neostore.nodestore.db.labels.id	25.04.2016 19:36	Plik ID	1 KB
neostore.propertystore	25.04.2016 23:45	Data Base File	40 023 KB
neostore.propertystore.db.arrays	25.04.2016 19:36	Plik ARRAYS	8 KB
neostore.propertystore.db.arrays.id	25.04.2016 19:36	Plik ID	1 KB
neostore.propertystore.db.id	25.04.2016 19:36	Plik ID	1 KB
neostore.propertystore.db.index	25.04.2016 20:08	Plik INDEX	8 KB
neostore.propertystore.db.index.id	25.04.2016 19:36	Plik ID	1 KB
neostore.propertystore.db.index.keys	25.04.2016 20:08	Plik KEYS	8 KB
neostore.propertystore.db.index.keys.id	25.04.2016 19:36	Plik ID	1 KB
neostore.propertystore.db.strings	25.04.2016 19:36	Plik STRINGS	8 KB
neostore.propertystore.db.strings.id	25.04.2016 19:36	Plik ID	1 KB
neostore.relationshipgroupstore	25.04.2016 19:36	Data Base File	8 KB
neostore.relationshipgroupstore.db.id	25.04.2016 19:36	Plik ID	1 KB
neostore.relationshipstore	25.04.2016 23:45	Data Base File	20 305 KB
neostore.relationshipstore.db.id	25.04.2016 23:45	Plik ID	9 KB
neostore.relationshiptypestore	25.04.2016 20:08	Data Base File	8 KB
neostore.relationshiptypestore.db.id	25.04.2016 19:36	Plik ID	1 KB
neostore.relationshiptypestore.db.names	25.04.2016 20:08	Plik NAMES	8 KB
neostore.relationshiptypestore.db.names...	25.04.2016 19:36	Plik ID	1 KB
neostore.schemastore	25.04.2016 20:08	Data Base File	8 KB
neostore.schemastore.db.id	25.04.2016 19:36	Plik ID	1 KB
neostore.transaction.db.0	25.04.2016 20:08	Plik 0	182 201 KB
store_lock	25.04.2016 19:36	Plik	0 KB

# Struktura danych

## Node (9 bajtów)

inUse	nextRelId	nextPropId
0	1	5

## Relationship (33 bajty)

inUse	firstNode	secondNode	relationshipType	firstPrevRelId	firstNextRelId	secondPrevRelId	secondNextRelId	nextPropId
0	1	5	9	13	17	21	25	29

## Relationship Type (5 bajtów)

inUse	typeBlockId
0	1

## Property (33 bajty)

inUse	type	keyIndexId	propBlock	nextPropId
0	1	3	5	29

## Property Index (9 bajtów)

inUse	propCount	keyBlockId
0	1	5

## Dynamic Store (125 bajtów)

inUse	next	data
0	1	5

## NeoStore (5 bajtów)

inUse	datum
0	1

## Cache węzła

ID		RelType 1	IN	Rel1	Rel2	Rel3	Rel4	Rel5
Node			OUT	Rel6	Rel7			
		RelType 2	IN	Rel8	Rel9	Rel10		
			OUT	Rel11	Rel12	Rel13	Rel14	
Key1	Val1							
Key2	Val2							
Key3	Val3							
Key4	Val4							



## Przykład kodu w Java

```
1. GraphDatabaseService graphDb = new EmbeddedGraphDatabase("./neo4j");
2. Transaction tx = graphDb.beginTx();
3. try {
4.     Node jan = graphDb.createNode();
5.     Node ala = graphDb.createNode();
6.     jan.setProperty("name", "Jan Kowalski");
7.     ala.setProperty("name", "Alicja Nowak");
8.
9.     Relationship likes = jan.createRelationshipTo(ala, FEELING_TYPE.LIKES);
10.    likes.setProperty("over", 9000);
11.    tx.success();
12. } finally {
13.     tx.finish();
14. }
```

## Przykład modelu w Java

```
@NodeEntity(label = "Company")
public class Company {
    @RelatedToVia(type = "WORKS_IN", direction=INCOMING)
    private Set<Job> jobs;
    private Director director;
}
```

```
@NodeEntity(label = "Worker")
public class Worker {
    @Indexed private String name;
    @RelatedTo(type = "WORKS_IN")
    private Set<Company> companies;
}
```

```
@RelationshipEntity(type = "WORKS_IN")
public class Job {
    @StartNode private Worker worker;
    @EndNode private Company company;
    private Date startDate;
}
```



# Cypher

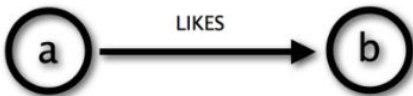
Język zapytań

# Cypher

Cypher jest deklaratywnym, inspirowanym SQL językiem do opisywania wzorców w grafie wykorzystując ascii-art syntax.

Pozwala nam na zdefiniowanie tego co chcemy otrzymać, zaktualizować lub dodać do grafu bez podania kolejnych instrukcji.

Cypher using relationship 'likes'



Cypher

(a) -[:LIKES]-> (b)

## Węzły

Do reprezentacji węzła używane są nawiasy okrągłe "()" które oplatając węzeł tworzą kółko.

Możliwe jest przypisanie do zmiennych, aby później odwoływać się do danego węzła.

```
MATCH (node:Label)  
RETURN node.property
```

```
MATCH  
  (node1:Label1)-->(node2:Label2)  
WHERE node1.propertyA = {value}  
RETURN node2.propertyA,  
       node2.propertyB
```

## Relacje

Do reprezentacji relacji używana jest strzałka "-->" pomiędzy dwoma węzłami.

Możliwe jest podanie dodatkowych informacji w nawiasach kwadratowych:

- typ relacji - [:KNOWS | :LIKE] ->
- nazwa zmiennej - [x:KNOWS] ->
- dodatkowe informacje - [{since:2010}] ->
- długość ścieżki - [:KNOWS\*..4] ->

## Label

:Person  
:Movie

## Relationship

:ACTED\_IN

## Identifier

movie  
role  
actor

```
MATCH (movie:Movie)<-[role:ACTED_IN]-(actor:Person)  
WHERE movie.title="The Matrix"  
RETURN role.roles, actor.name
```

.title  
.roles  
.name

## Property

"The Matrix"

## Value

## Przykład kodu w Java

```
1. Driver driver = GraphDatabase.driver("bolt://localhost", AuthTokens.basic("neo4j", "neo4j"));
2. Session session = driver.session();
3. session.run( "CREATE (a:Person {name:'Arthur', title:'King'})" );
4. StatementResult result = session.run("MATCH (a:Person) WHERE a.name = 'Arthur' RETURN a.name AS name, a.title AS title");
5. while (result.hasNext()) {
6.     Record record = result.next();
7.     System.out.println(record.get("title").asString() + " " + record.get("name").asString());
8. }
9. session.close();
10. driver.close();
```



## Operacije na grafie

### Query:

- START
- MATCH
- WHERE
- RETURN
- ORDER BY
- SKIP/LIMIT

### Update:

- CREATE
- MERGE
- CREATE UNIQUE
- DELETE
- SET
- REMOVE
- FOREACH

## Wzorce dopasowania

Wzorce dopasowania, są wyrażeniami, które zwracają kolekcję węzłów i relacji (ścieżek). Wszędzie tam gdzie można użyć wyrażenia (klauzula 'WHERE' , 'RETURN' , 'WITH' , etc.), można też użyć wzorców dopasowań.

```
MATCH a-[:REL]-b WHERE NOT b-[:REL]-c RETURN b;  
MATCH a-[:REL]-b RETURN a,b-[:REL]-c;  
MATCH a-[:REL]-b WITH a, b-[:REL]-c RETURN a, count(distinct c);
```

## Kolekcje

```
RETURN [0,1,2,3,4,5,6,7,8,9] AS collection
```

```
RETURN range(0,10)[3]
```

```
RETURN length(range(0,10)[0..3])
```

```
RETURN [x IN range(0,10) WHERE x % 2 = 0 | x^3] AS result
```

## Filter

```
RETURN [x IN range(0,10) WHERE x % 2 = 0 | x^3] AS result;
```

```
WHERE a.name='Alice' AND b.name='Daniel' AND  
ALL (x IN nodes(p) WHERE x.age > 30);
```

```
WHERE a.name='Eskil' AND ANY (x IN a.array WHERE x = "one");
```

```
WHERE n.name='Alice' AND NONE (x IN nodes(p) WHERE x.age = 25);
```

```
WHERE n.name='Alice' AND  
SINGLE (var IN nodes(p) WHERE var.eyes = "blue");
```

```
MATCH (n)
```

```
WHERE EXISTS(n.name)
```

```
RETURN n.name AS name, EXISTS((n) - [:MARRIED] ->()) AS is_married
```

## Ścieżki

```
MATCH p = ((a) -- (b)) RETURN nodes(p)
MATCH p = ((a) -- (b)) RETURN relationships(p)
MATCH p = ((a) -[:RAILS] - (b)) UNWIND nodes(p) AS n return DISTINCT n;
```

## Potoki

Klauzula 'WITH' umożliwia przetwarzanie zapytań w potokach, gdzie wynik poprzedniego zapytania jest przekazywany do następnego, myślimy o tym jak o '|' w UNIX.

```
MATCH (david { name: "David" })[-](otherPerson)[-]>()  
WITH otherPerson, count(*) AS foaf  
WHERE foaf > 1  
RETURN otherPerson
```

## Budowanie przykładowych wzorców

- friend-of-a-friend
  - `(user)-[:KNOWS]-(friend)-[:KNOWS]-(foaf)`
- najkrótsza ścieżka
  - `path = shortestPath( (user)-[:KNOWS*..5]-(other) )`
- collaborative filtering
  - `(user)-[:PURCHASED]->(product)<-[:PURCHASED]-()-[:PURCHASED]  
->(otherProduct)`
- nawigacja w drzewie
  - `(root)<-[:PARENT*]-(leaf:Category)-[:ITEM]->(data:Product)`

# Gremlin

Język zapytań





# Gremlin

- Graph Traversal Language
- DSL dla grafów

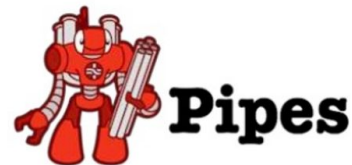
A Graph DSL



A Dynamic Language for the JVM



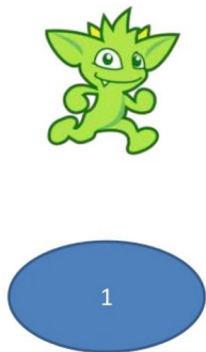
A Data Flow Framework



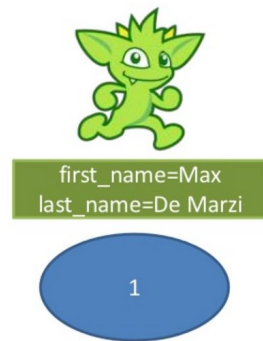
"JDBC" for Graph DBs



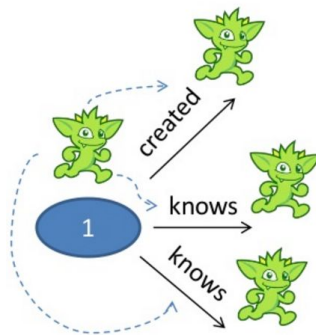
`g.v(1)`



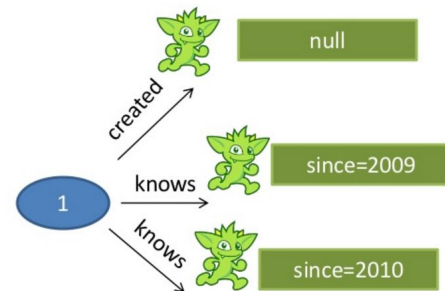
`g.v(1).map()`



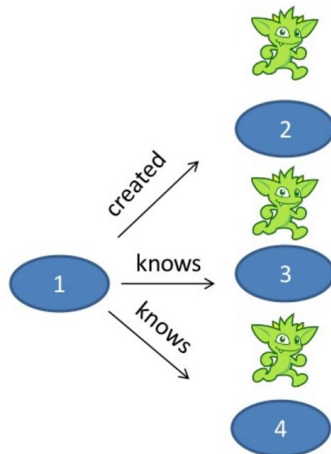
`g.v(1).outE`



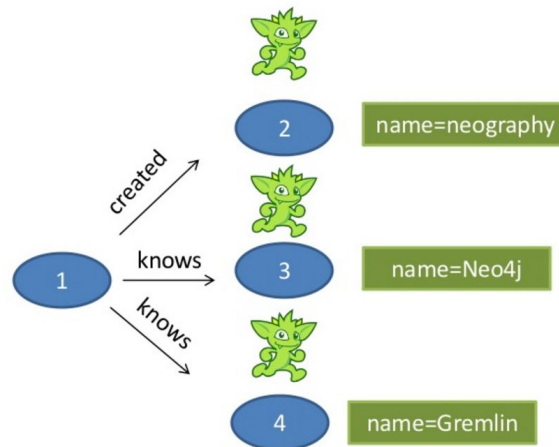
`g.v(1).outE.since`



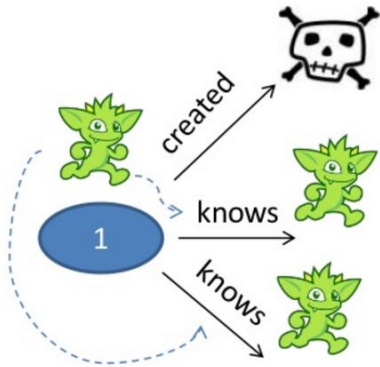
`g.v(1).outE.inV`



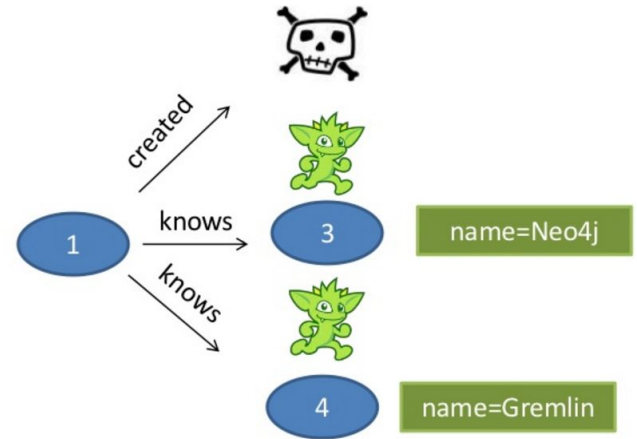
`g.v(1).outE.inV.name`



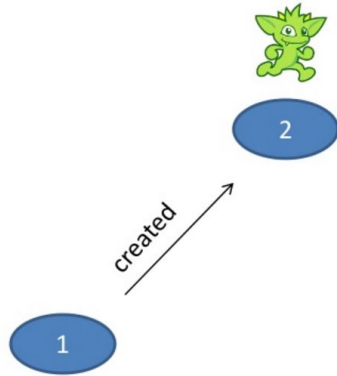
```
g.v(1).outE  
  .filter{it.label=='knows'}
```



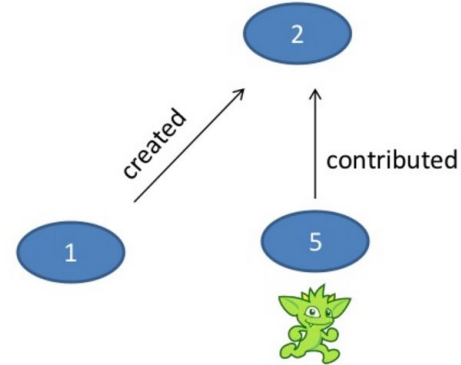
```
g.v(1). out('knows').name
```



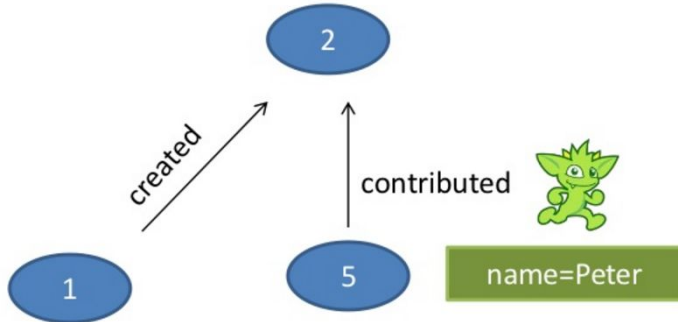
`g.v(1). out('created')`



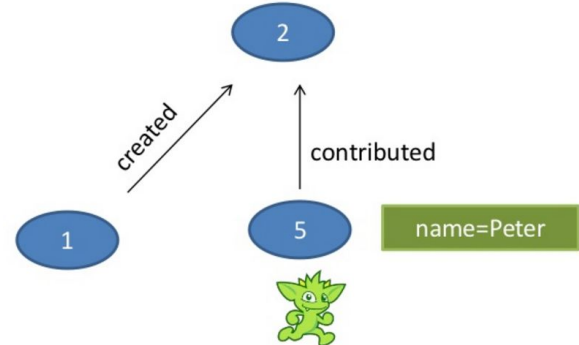
`g.v(1). out('created').in('contributed')`



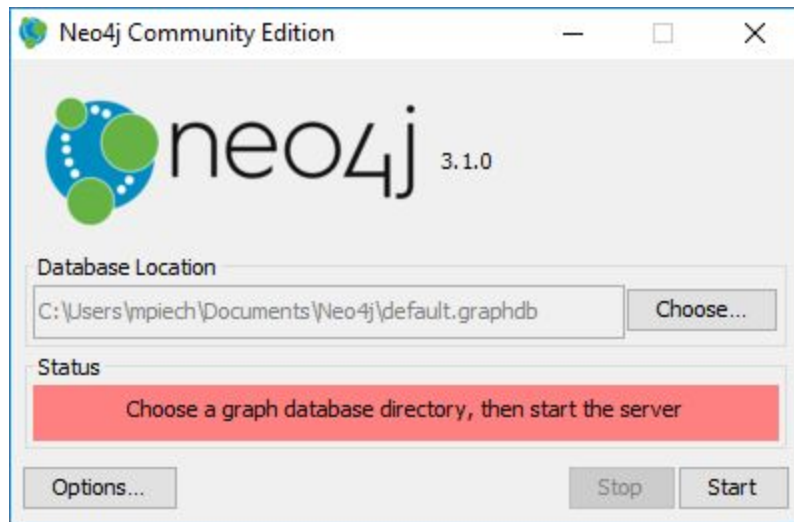
`g.v(1). out('created').in('contributed').name`



`g.v(1). out('created').in('contributed').name.back(1)`



# Neo4j w akcji





## Database Information



### Node labels (2)

Movie Person



### Relationship types (4)

ACTED\_IN DIRECTED

PRODUCED WROTE

### Property keys (8)

episode gender imdbid kind

name season title year

### Connected as

Username: neo4j

Admin: server user list

### Database

Version: 3.1.0

Name: default.graphdb

Size: 4.01 GiB

Information: sysinfo

\$



:play start



### Learn about Neo4j

A graph epiphany awaits you.



What is a graph database?

How can I query a graph?

What do people do with Neo4j?

Start Learning

### Jump into code

Use Cypher, the graph query language.



Code walk-throughs

RDBMS to Graph

Query templates

Write Code

### Monitor the system

Key system health and status metrics.



Disk utilization

Cache activity

Cluster health and status

Monitor

[Free online training, to level-up your Neo4j skills.](#)

Copyright © Neo Technology 2002–2016

```
$ MATCH (p:Person) WHERE p.name = "Emma Watson II" RETURN p
```



Graph

"(1)" Person(1)

Rows

Text

Code

Emma Watson

Displaying 1 node, 0 relationships.

AUTO-COMPLETE ☒









Code

\*(22)    ACTED\_IN(21)    PRODUCED(1)



AUTO-COMPLETE ☒

```
$ MATCH (karolak:Person {name:"Tomasz Karolak"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(emma:Person {name:"Emma Watson II"}) RETURN karolak, ...
```



Graph

\*(5) **Movie(2)** **Person(3)**

\*(4) **ACTED\_IN(4)**



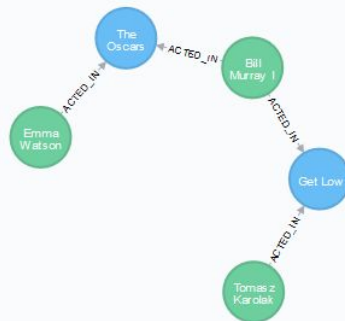
Rows



Text



Code



Displaying 5 nodes, 4 relationships (completed with 4 additional relationships).

AUTO-COMPLETE ☒



# Laboratorium

## Materiał na zajęciach

The IMDb logo, consisting of the letters "IMDb" in a bold, black, sans-serif font, set against a yellow rectangular background with rounded corners.

- Użyjemy bazy danych IMDB
- Dwie części:
  - W pierwszej użyjemy Neo4j Web Interface aby zapoznać się z Cypher
  - W drugiej napiszemy aplikację w Java, która będzie realizować założenia biznesowe

# Podsumowanie

## Podsumowanie

- Grafowe bazy danych mają zastosowanie w modelach, których złożoność przekracza możliwości relacyjnej bazy danych
- Rozwiązania wydajne, skalowalne
- Dynamiczna struktura
- Zastosowanie w wielu sieciach społecznych

IN COMMON WITH AMANDA



## Referencje

- Neo4j - <https://neo4j.com/>
- Grafowe bazy danych by Tom Tom (Prezi)
- Lesson 8: Graph Databases by Bill Vorhies
- Intro to Neo4j presentation -  
<http://www.slideshare.net/jexp/intro-to-neo4j-presentation>
- <https://neo4j.com/docs/cypher-refcard/current/>
- <http://www.slideshare.net/maxdemarzi/introduction-to-gremlin>
- Grafowy model bazy danych na przykładzie GOOD, Marcin Jakubek
- W gąszczu grafów, Jarosław Pałka
- Grafowe bazy danych - przegląd technologii, Daniel Słotwiński