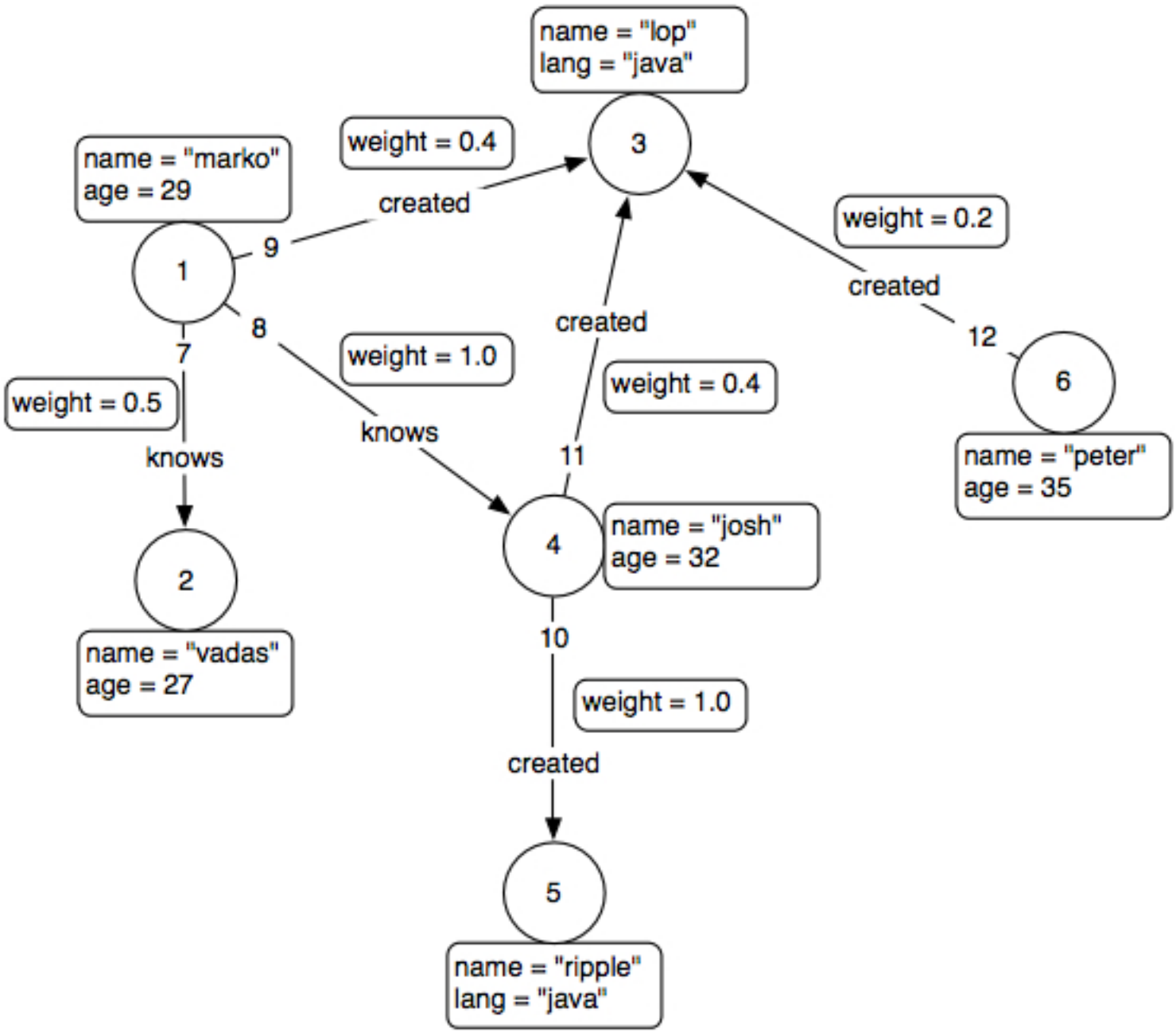
Grafowe języki zapytań

Anna Kosieradzka

Grafowe języki zapytań

- Ogólne:
  - Blueprints
  - Gremlin
  - GraphQL
  - SPARQL
- Związane z jedną bazą danych:
  - SQL (OrientDB)
  - GQL (sones GraphDB)
  - inne API

Graf na dzisiaj



Graf z własnościami *(Property Graph)*

Graf z własnościami składa się z:

- wierzchołków takich, że:
  - każdy ma unikalny identyfikator
  - każdy ma zbiór krawędzi wchodzących
  - każdy ma zbiór krawędzi wychodzących
  - każdy ma zbiór własności (pary klucz, wartość)
- krawędzi takich, że:
  - każda ma unikalny identyfikator
  - każda ma wierzchołek początkowy
  - każda ma wierzchołek końcowy
  - każda ma etykietę opisującą związek między jej wierzchołkami
  - każda ma zbiór własności (pary klucz, wartość)

Blueprints

API w Javie kompatybilne m.in. z:

- grafowe bazy danych:
  - DEX
  - Neo4j
  - OrientDB
- biblioteki:
  - TinkerGraph
  - SAIL (Storage and Inference Layer)
  - Rexster (serwer http udostępniający grafy)
- w przyszłości może również z:
  - InfiniteGraph (prace w trakcie)
  - inne

Blueprints - tworzenie nowego grafu

```
Graph graph = new TinkerGraph();
Vertex a = graph.addVertex(null);
Vertex b = graph.addVertex(null);
a.setProperty("name", "marko");
b.setProperty("name", "peter");
Edge e = graph.addEdge(null, a, b, "knows");
System.out.println(
    e.getOutVertex().getProperty("name")
    + "--" + e.getLabel() + "-->"
    + e.getInVertex().getProperty("name"));
```

Wynik:

```
marko--knows-->peter
```

Blueprints- iterowanie po grafie

Kod:

```
for (Vertex vertex : graph.getVertices()) {
    System.out.println(vertex);
}
```

da wynik:

```
v[3] v[2] v[1] v[6] v[5] v[4]
```

Kod:

```
for (Edge edge : graph.getEdges()) {
    System.out.println(edge);
}
```

da wynik:

```
e[10][4-created->5]
e[7][1-knows->2]
e[9][1-created->3]
```

...

Gremlin

- Język do iterowania po grafach z własnościami
- Składnia przypominająca Groovy
- Korzysta z Pipes
- Ma API w Javie oraz konsolę:

```
marko$ ./gremlin.sh
          \,,,/
          (o o)
-----oOOo-(_)-oOOo-----
gremlin>
```

Pipes

- `Pipe<S,E>` to interfejs w Javie rozszerzający `Iterable<E>` i `Iterator<E>`.
- `Pipe<S,E>` bierze na wejściu `Iterator<S>` lub `Iterable<S>` i zwraca obiekty typu E.

Pipes - przykład

Następujący kod:

```
Pipe<String,Integer> countPipe = new CountPipe();
Pipe<Integer,String> wordPipe = new WordPipe();
Pipeline<String,String> pipeline = new
Pipeline<String,String>(countPipe, wordPipe);

pipeline.setStarts(Arrays.asList("this", "is",
"the", "end."));
while(pipeline.hasNext()) {
  System.out.print(pipeline.next() + " ");
}
```

da na wyjściu:

```
four two three four
```

Gremlin - iterowanie po grafie

Dobry start:

```
gremlin> g = TinkerGraphFactory.
createTinkerGraph()
==> tinkergraph[vertices:6 edges:6]
```

Dostęp do wierzchołka 1:

```
gremlin> v = g.v(1)
==>v[1]
```

Krawędzie wychodzące z wierzchołka 1:

```
gremlin> v.outE
==>e[7][1-knows->2]
==>e[9][1-created->3]
==>e[8][1-knows->4]
```

Gremlin - podstawowe słówka

`_`	emituje obiekt z wejścia (niezmieniony) - IdentityPipe()
`V, E`	Iteratory po odp. wierzchołkach i krawędziach
`id, label`	Identyfikator i etykieta elementu - IdPipe(), LabelPipe()
`outE, inE`	krawędzie wychodzące / wchodzące z / do danego wierzchołka - OutEdgesPipe(), InEdgesPipe()
`bothE`	wszystkie krawędzie od i do tego wierzchołka

`outV, inV`	odp. wierzchołek początkowy i końcowy krawędzi
	
	`gremlin> v.outE.inV`
	`==>v[2]`
	`==>v[3]`
	`==>v[4]`

`bothV`	oba powyższe

`['key'], key`	właściwość elementu - PropertyPipe('key')
	
	`gremlin> v.name`
	`==>marko`

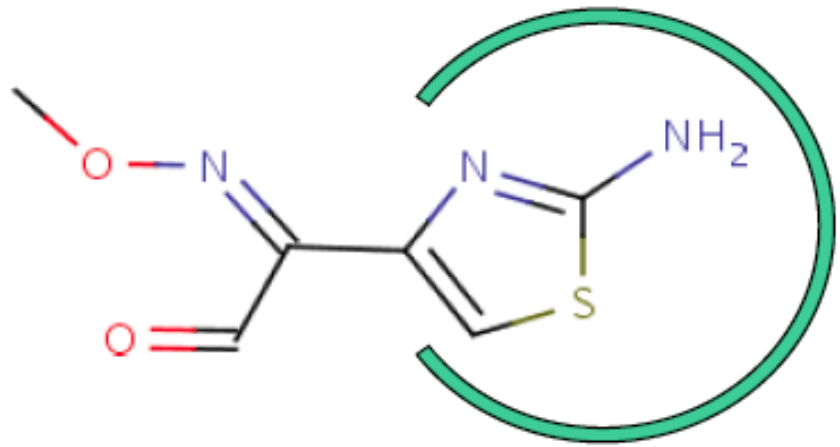
Gremlin - wybrane metody

Graph.v(Object)	Dostęp do wierzchołka po identyfikatorze
Graph.e(Object)	Dostęp do krawędzi po identyfikatorze
Graph.addVertex() Graph.addVertex(Object, Map) Graph.addVertex(Map)	Dodawanie wierzchołków
Graph.addEdge(Object, Vertex, Vertex, String, Map) Graph.addEdge(Vertex, Vertex, String, Map)	Dodawanie krawędzi
Element.key Element[key] = value Element.keys() Element.values() Element.map()	Operacje na mapach własności
Pipe>>n Pipe>>-1	Pobranie n lub wszystkich elemetów z *pipe*.
Pipe>>Collection	Dodaje zawartość *pipe* do `collection`.
Pipe.count()	Zwraca liczbę elementów w *pipe*.

GraphQL

- Teoretyczny: np. operator selekcji to σ
- Pełen opis w: Huahai He; Ambuj K. Singh; Graphs-at-a-time: Query Language and Access Methods for Graph Databases, SIGMOD, June, 2008, pp 405-418. ISBN: 978-1-60558-102-6
- Brak dostępnej implementacji
- Jedno zastosowanie: PharmaMiner firmy Acelot, Inc (premiera: 2. połowa 2011)

```
graph Vieth02 {
    graph aminothiazole as A;
    node C1, O2, N3, C4, C5, O6;
    edge e1 (C1, O2);
    edge e2 (02, N3);
    ...
}
```



SPARQL

- **S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage
- RDF = Resource Description Framework
- Działa z AllegroGraph
- Przykład: zapytanie o wszystkie stolice w Afryce:
-
```
PREFIX abc: <http://example.
com/exampleOntology#>
SELECT ?capital ?country
WHERE {
   ?x abc:cityname ?capital ;
      abc:isCapitalOf ?y .
   ?y abc:countryname ?country ;
      abc:isInContinent abc:Africa .
}
```

GraphDB GQL

- Odchodzimy od koncepcji grafu z własnościami
- Wierzchołki są silnie typowane
- Krawędzie nie mają własności (co najwyżej wagi)
- Pracę zaczynamy od tworzenia typów:

```
CREATE TYPE Person ATTRIBUTES(String name,
Integer age, SET<Project> Projects)

CREATE TYPE Project ATTRIBUTES(String name,
String lang) BACKWARDEDGES (Person.Projects
Authors)
```

- Istnieją `ALTER TYPE, DROP TYPE, DESCRIBE TYPES.`

GraphDB GQL

- `Insert` i `Update` wyglądają znajomo:
  ```
  INSERT INTO Person VALUES (name =  'vadas',
  age = 27)
  UPDATE Person SET(Projects += SETOF(name =
  'lop')) WHERE name = 'marko'
  ```

- Nowością w `Select` jest parametr `Depth`, np.:
  ```
  FROM Website w SELECT w.Tags WHERE
  w.Name = 'xkcd' DEPTH 1
  ```
  jest równoważne:
  ```
  FROM Tag t SELECT t.Name WHERE t.
  TaggedWebsites.Name = 'xkcd'
  ```

OrientDB SQL

- Wierzchołki są słabo typowane
- Każdy wierzchołek ma swój RecordID postaci: [<cluster>:<position>]
- Są 2 rodzaje relacji pomiędzy wierzchołkami: referenced i embedded.
- Referenced oznacza, że jeden rekord przechowywuje wskaźnik do drugiego. Oba rekordy mają własne RID.
  - Do relacji 1-1 i N-1 używa się typu `LINK`.
  - Do relacji 1-N i N-M używa się typów `LINKLIST`, `LINKSET` i `LINKMAP`.
- Embedded oznacza, że jeden rekord zawiera się w drugim - nie ma wówczas własnego RID.
  - Typy to `EMBEDDED[LIST|SET|MAP]`.

OrientDB SQL - CREATE

- Najpierw tworzymy typy:

```
create class Person
create class Project
```

- Następnie tworzymy własności:

```
create property Person.name string
create property Person.age integer
create property Project.name string
create property Project.lang string
create property Person.Projects embeddedmap
Project
```

OrientDB SQL - INSERT i UPDATE

- Zwykłe inserty są standardowe:

```
insert into Person (name, age) values ('marko', 29)
insert into Project (name, lang) values ('lop', 'java')
```

- Gdy pojawiają się relacje, używamy RecordID:

```
insert into Employee (name, boss) values ('jack', 11:99 )
insert into Profile (name, friends) values ('Luca', [10:3,
10:4] )
```

- Podobnie w przypadku update dla kolekcji:

```
update Account add addresses = #12:0
update Account remove addresses = #12:0
```

- oraz dla map:

```
update Account put addresses = 'Luca', #12:0
update Account remove addresses = 'Luca'
```

OrientDB SQL - SELECT

- Tu również można skorzystać z RecordID:

```
select * from [10:3, 10:4, 10:5]
```

- Dla typów zagnieżdżonych mamy operator traverse:

```
select from 11:4 where any() traverse(0,3) (address.city = 'Rome')
```

- Istnieje również FIND REFERENCES:
  find references 5:0
  find references 5:0 [Profile,AnimalType]
  Wynik będzie postaci:
  RESULT: [6:1,8:2,..]

Linki

- Blueprints: https://github.com/tinkerpop/blueprints/wiki/
- Gremlin: https://github.com/tinkerpop/gremlin/wiki/
- GraphQL: http://portal.acm.org/citation.cfm?id=1376660
- SPARQL: http://www.w3.org/TR/rdf-sparql-query/
- OrientDB SQL: http://code.google.com/p/orient/wiki/SQL
- GraphDB GQL: http://developers.sones.de/wiki/doku.php?id=documentation:gql

Pytania?