Графови бази от данни

Георги Ангелов

astea SOLUTIONS

Обхождане

- Заявките представляват обхождане на графа
- Много по-голяма производителност от RDBMS или NoSQL – локалност на данните

Подходящи за

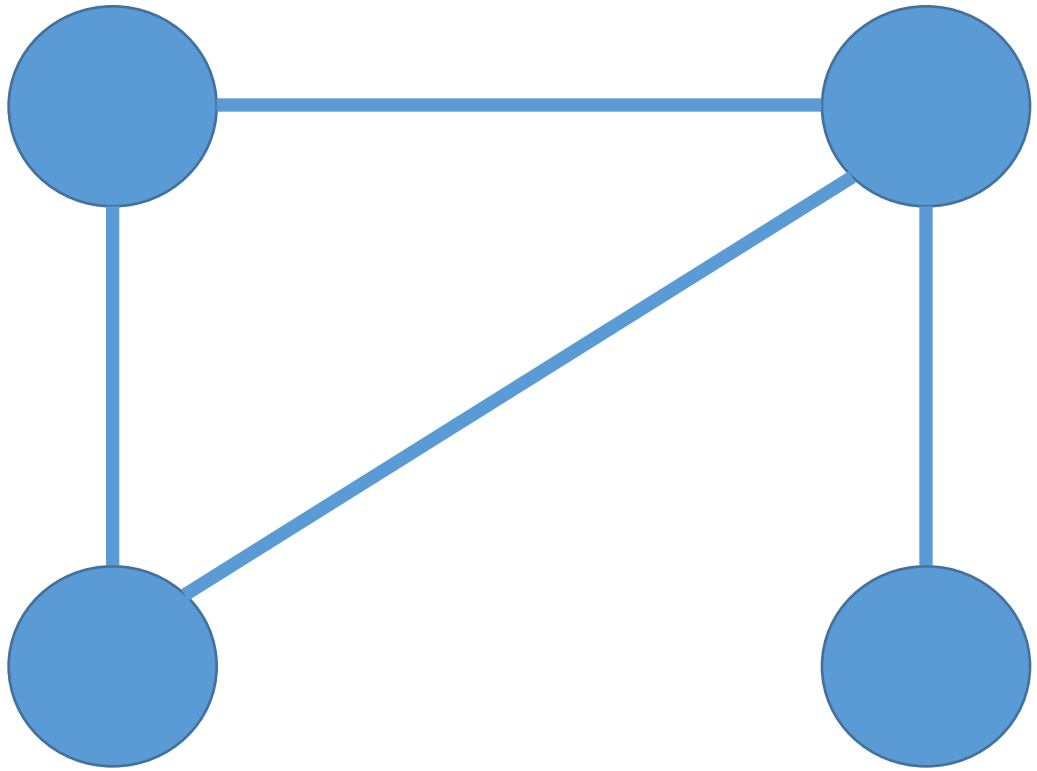
- Социални мрежи
- Препоръчващи системи
- Анализиране на данни
- Търсене на пътища
- Други видове данни с много връзки

Например

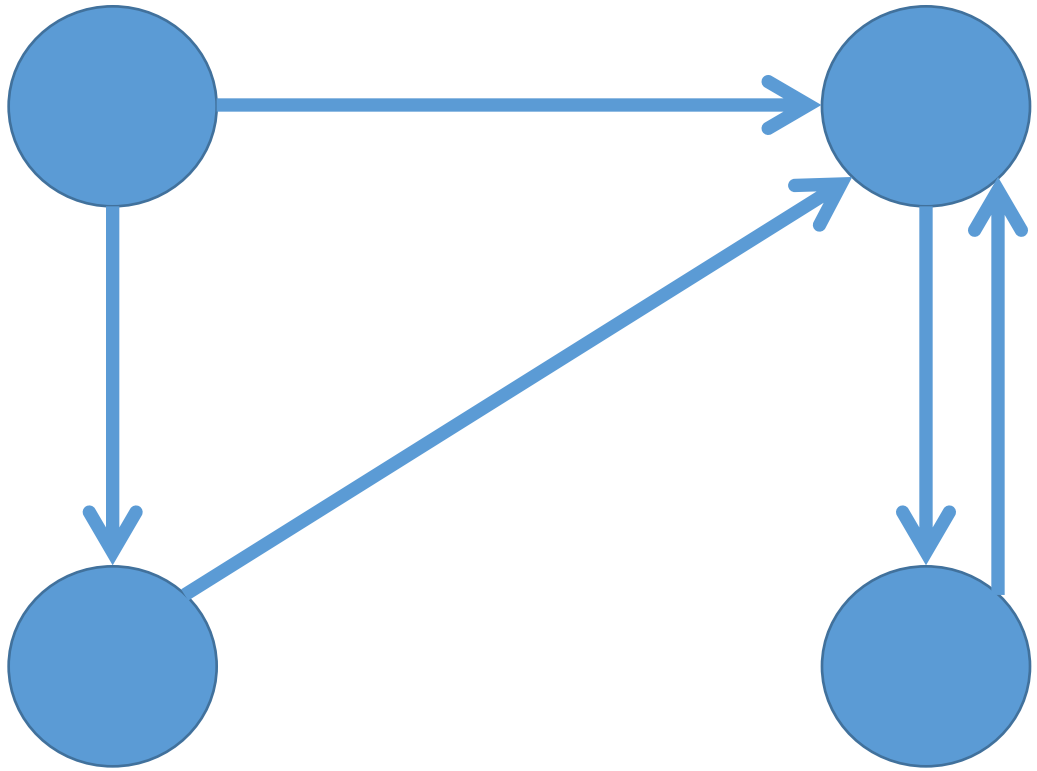
- Социална мрежа
- Сайт за запознанства
- Онлайн магазин
- Сайт и мобилно приложение, агрегиращо онлайн концерти
- Анализ на поведението на потребители в сайт

Графови модели

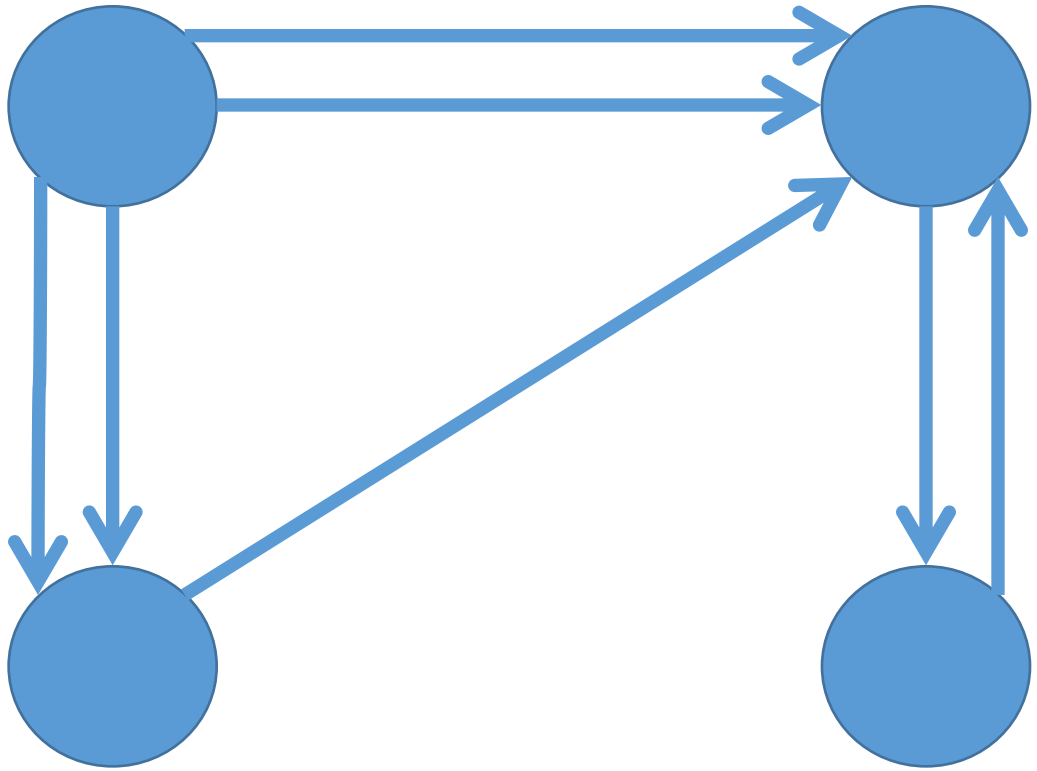
Неориентиран граф



Ориентиран граф

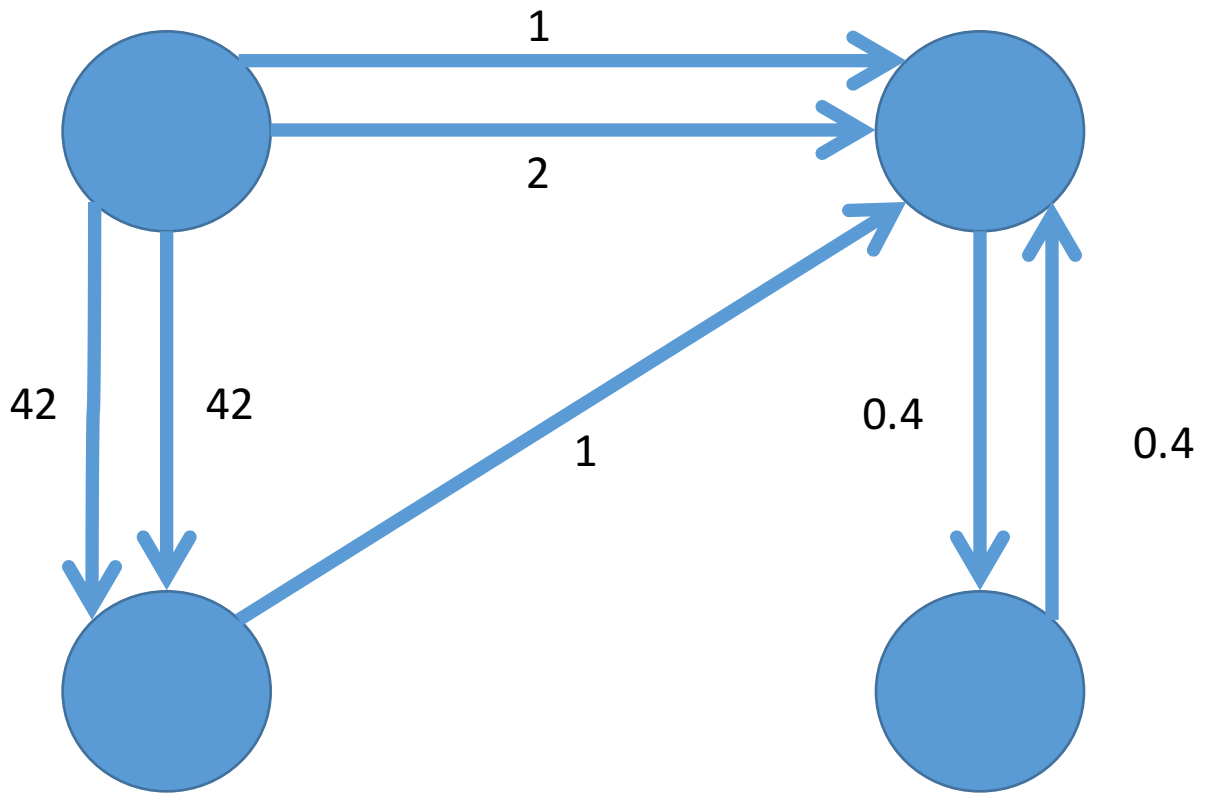


Ориентиран мултиграф

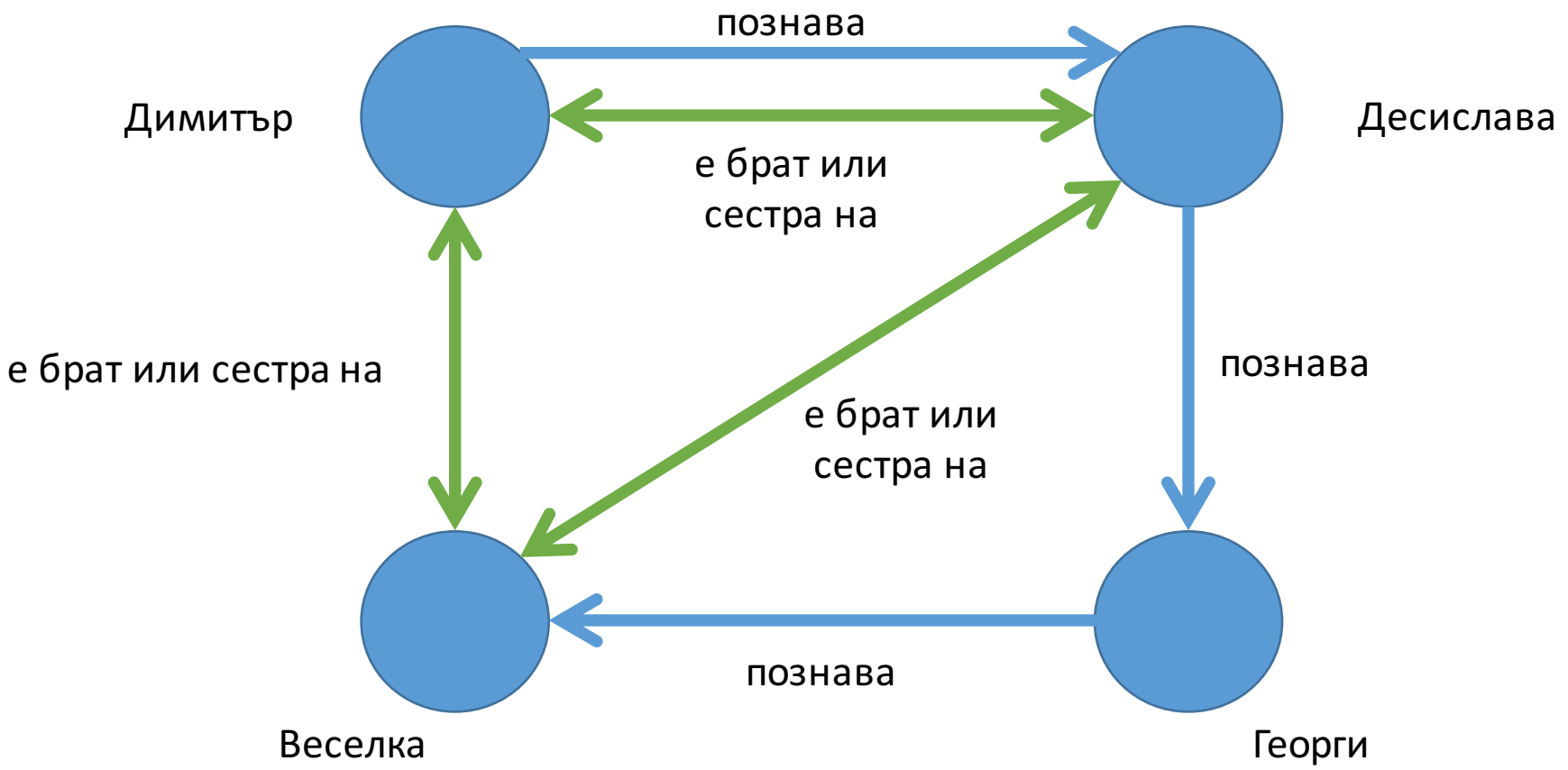


Граф Дракула

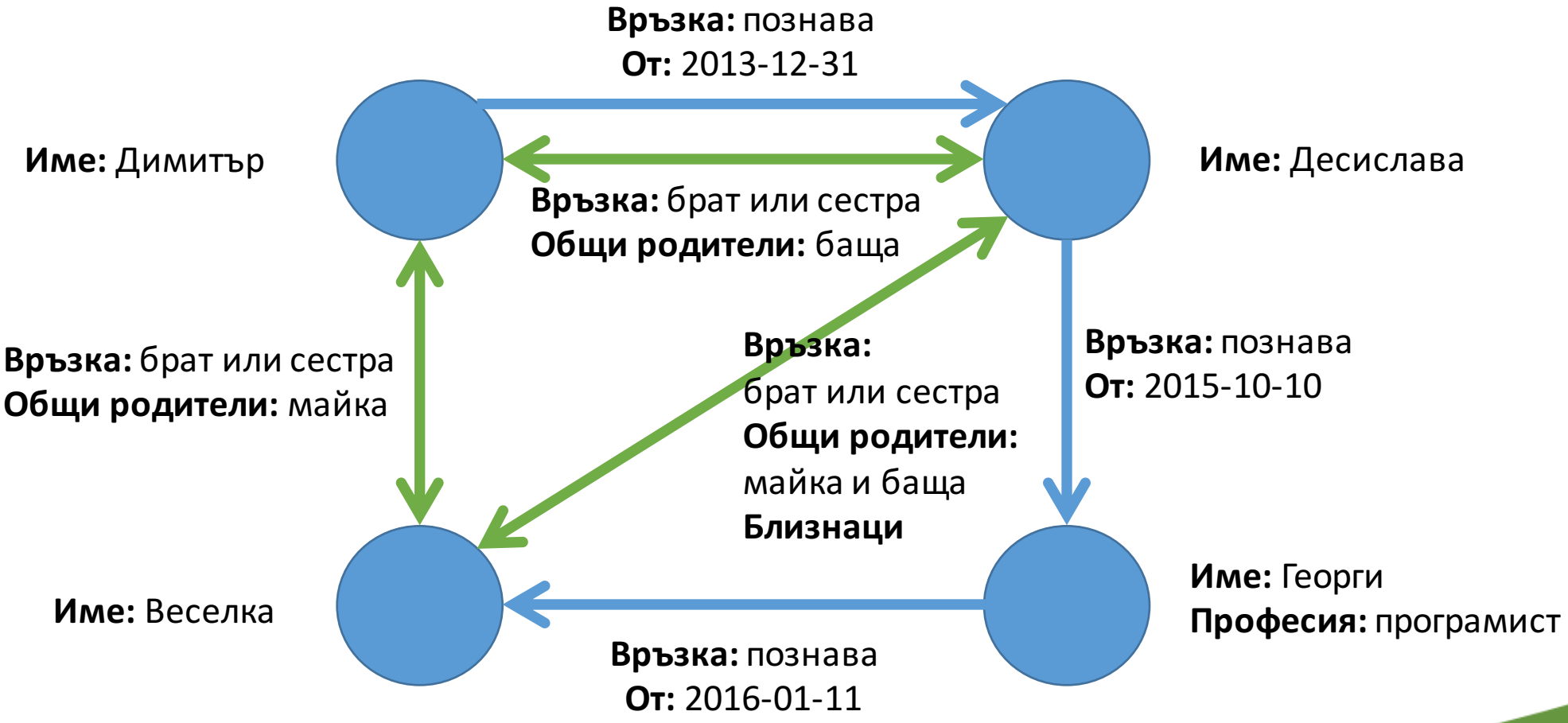
Ориентиран мултиграф с тегла



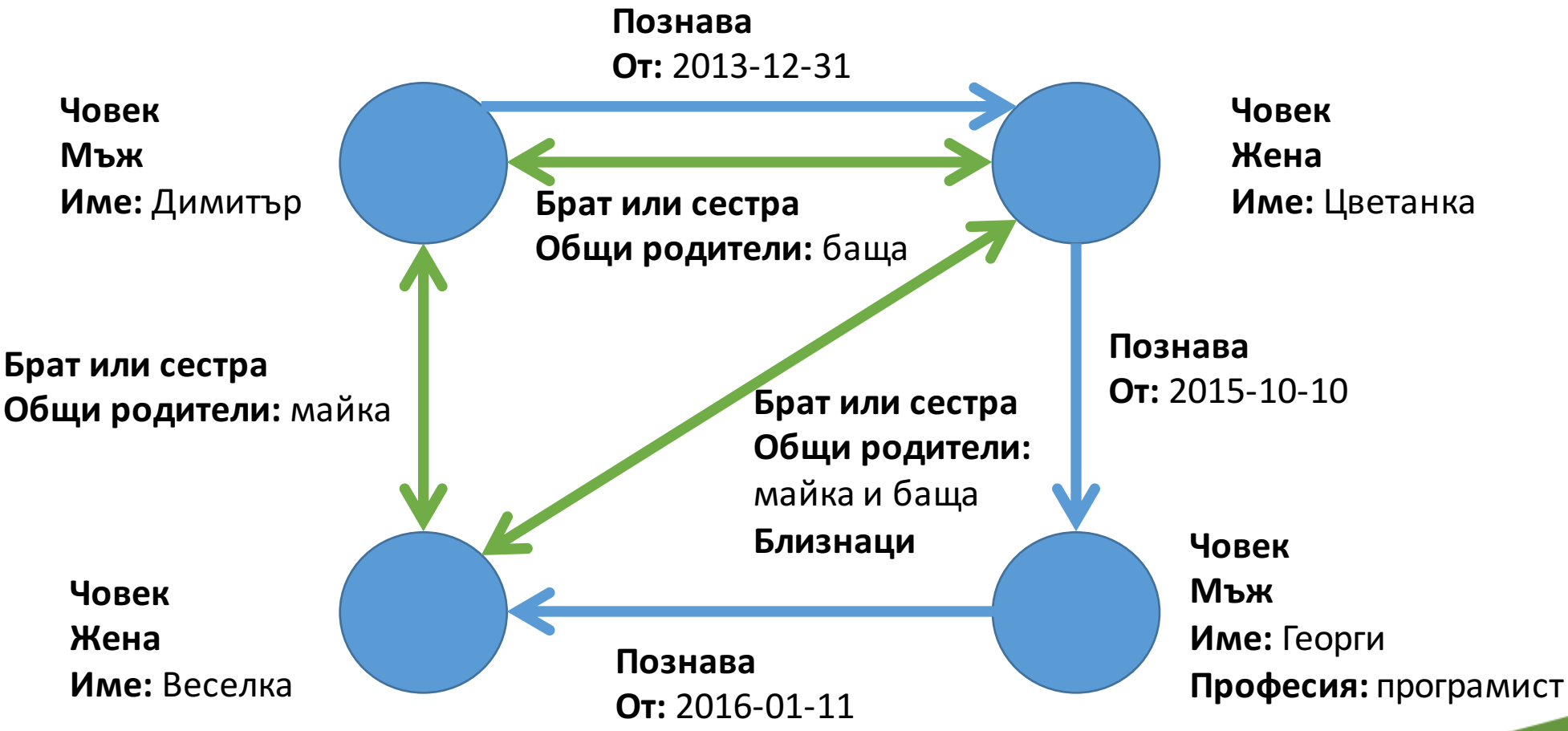
Ориентиран мултиграф с етикети



Ориентиран мултиграф със свойства (property graph)



Ориентиран мултиграф с етикети и свойства (labeled property graph)



(Labeled) property graph

- Модел за представяне на данни
- Използва се от графовите бази от данни
- Ребрата са ориентирани, но могат да се обхождат и в двете посоки
- Всеки връх и ребро може да има нула или повече етикети и нула или повече свойства
- Schema-free


neo4j

Neo4j

- Най-популярната графова база от данни
- Отворен код (GPL)
- Платена за комерсиално използване
- Консистентност
- Транзакции
- Репликация

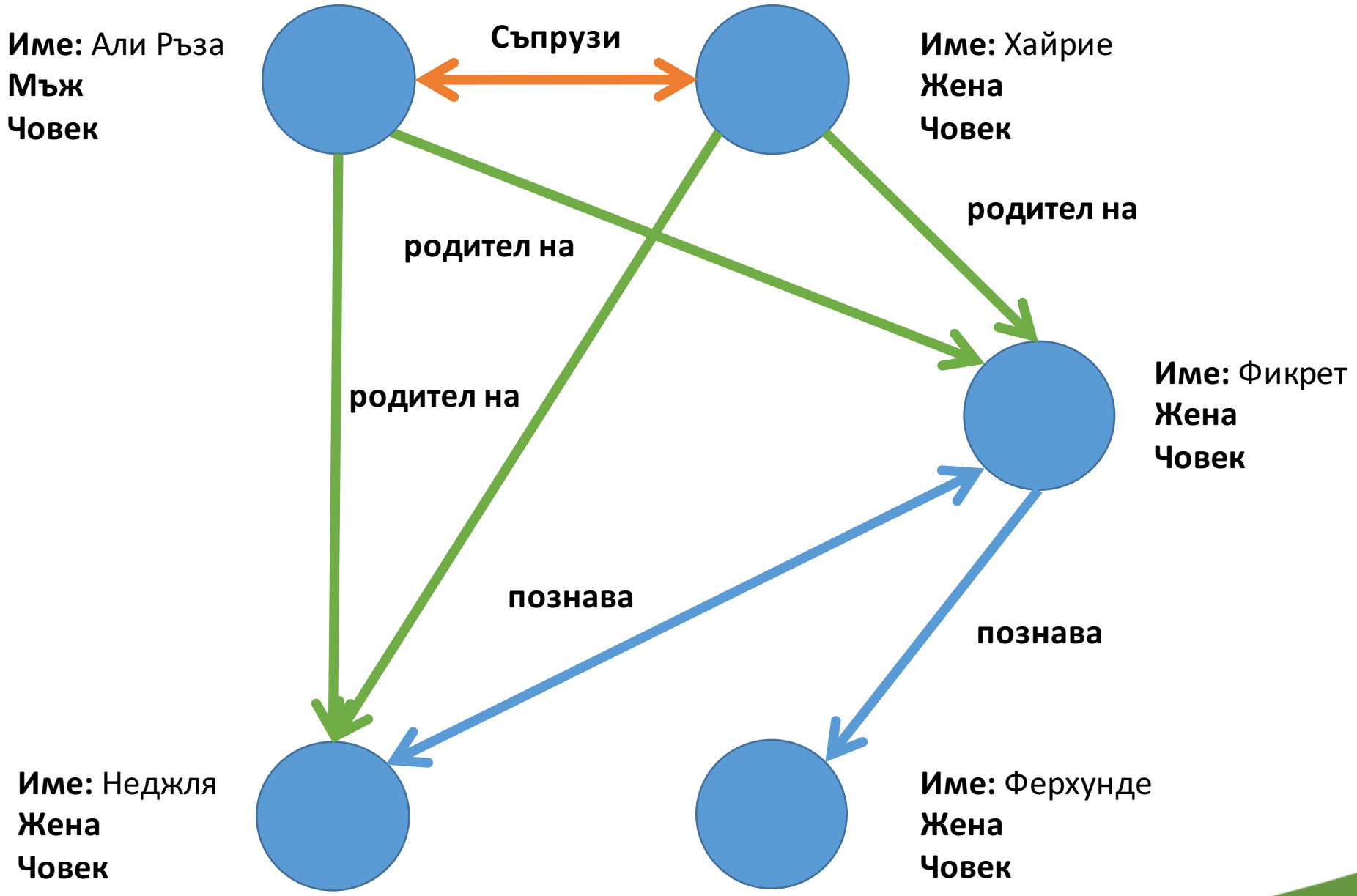
Cypher

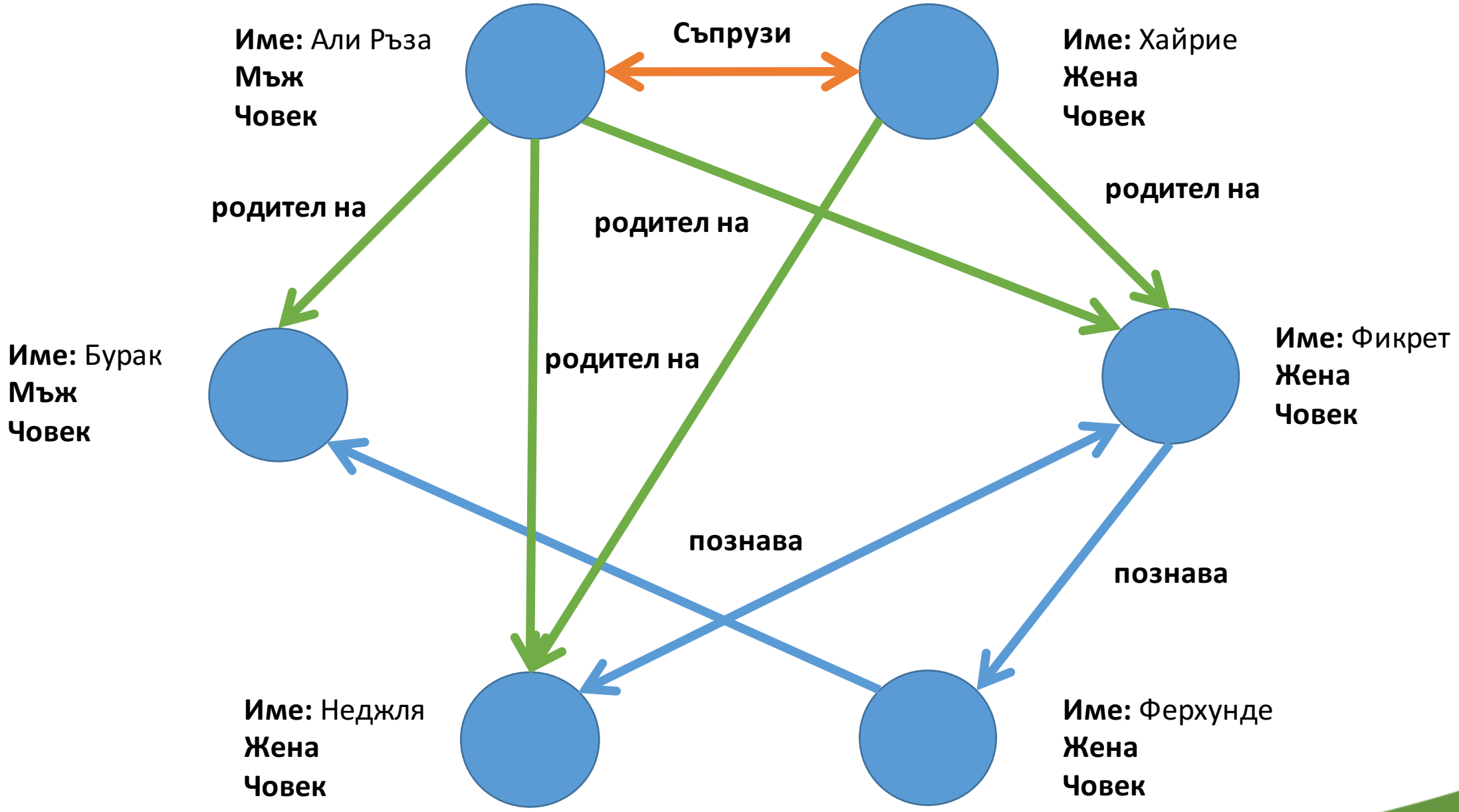
- Език за заявки към графови данни
- Създаден специално за Neo4j
- OpenCypher – млад проект за разпространение на Cypher и към други СУБД
- Декларативен език

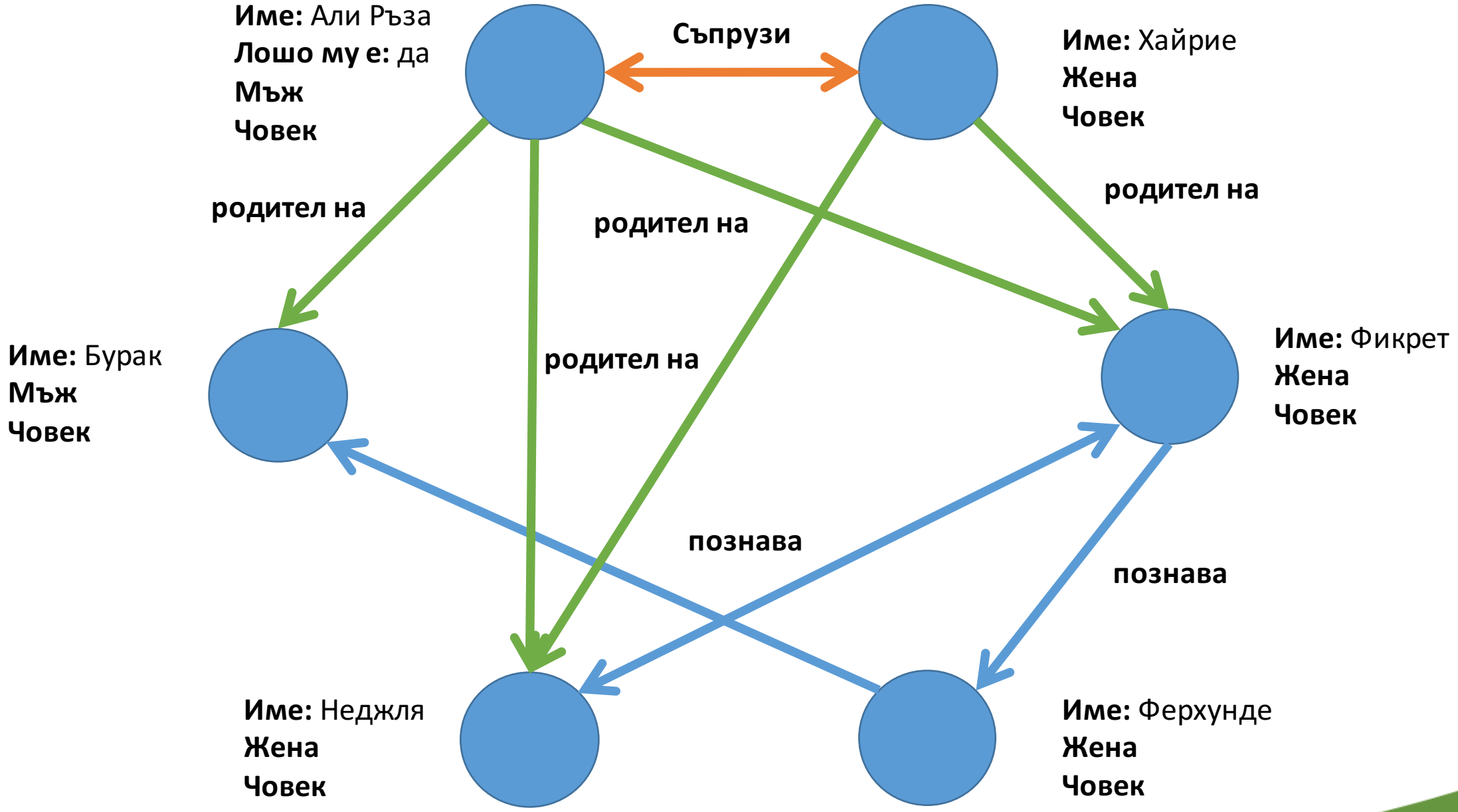
Инсталация

astea SOLUTIONS

Примери с Neo4j


Име: Али Ръза
Мъж
Човек
Съпрузи
Име: Хайрие
Жена
Човек
родител на
родител на
родител на
Име: Фикрет
Жена
Човек
познава
познава
Име: Неджля
Жена
Човек
Име: Ферхунде
Жена
Човек


Име: Али Ръза
Мъж
Човек
Съпрузи
Име: Хайрие
Жена
Човек
родител на
родител на
родител на
Име: Бурак
Мъж
Човек
родител на
Име: Фикрет
Жена
Човек
познава
познава
Име: Неджля
Жена
Човек
Име: Ферхунде
Жена
Човек


Име: Али Ръза
Лошо му е: да
Мъж
Човек
Съпрузи
Име: Хайрие
Жена
Човек
родител на
родител на
родител на
Име: Бурак
Мъж
Човек
родител на
Име: Фикрет
Жена
Човек
познава
познава
Име: Неджля
Жена
Човек
Име: Ферхунде
Жена
Човек

Добавяне на върхове

```
create
  (ali:Person:Man {name: 'Али Ръза', лобо: true}),
  (hayrie:Person:Woman {name: 'Хайрие'}),
  (burak:Person:Man {name: 'Бурак'}),
  (fikret:Person:Woman {name: 'Фикрет'}),
  (nedjlya:Person:Woman {name: 'Неджля'}),
  (ferhunde:Person:Woman {name: 'Ферхунде'})
```

Добавяне на ребра

```
create
  (ali)-[:married_to]->(hayrie),
  (hayrie)-[:married_to]->(ali),

  (ali)-[:parent_of]->(fikret),
  (ali)-[:parent_of]->(burak),
  (ali)-[:parent_of]->(nedjlya),
  (hayrie)-[:parent_of]->(nedjlya),
  (hayrie)-[:parent_of]->(fikret),

  (nedjlya)-[:knows]->(fikret),
  (fikret)-[:knows]->(nedjlya),
  (fikret)-[:knows]->(ferhunde),
  (ferhunde)-[:knows]->(burak)
```

* Разчита се, че ще се изпълни едновременно с предната заявка, иначе ще създаде празни върхове.

Показване на всички върхове и ребра

```
match (vertex) return vertex

match ()-[edge]-() return edge
```

Търсене на върхове по етикети

```
match (p:Person) return p
```

Търсене на върхове със свойство

```
match (p:Person {name: 'Али Ръза'})
return p
```

Добавяне на ребра без "променливите"

```
match (ali:Person {name: 'Али Ръза'})),
      (hayrie:Person {name: 'Хайрие'})
create (ali)-[:married_to]->(hayrie)
```

Добавяне на ребра със свойства

```
match (ali:Person {name: 'Али Ръза'})),
      (hayrie:Person {name: 'Хайрие'})
create (ali)-[:married_to {on: '1980'}]->(hayrie)
```

Проекция

```
match (p:Person {name: 'Али Ръза'})
return p.name
```

Прости обхождания

```
match (p:Person {name: 'Али Ръза'})
      -[:parent_of]->(child:Person)
return child.name
```

```
match (child:Person)
      <-[:parent_of]-
      (p:Person {name: 'Али Ръза'})
return child.name
```

Изтриване на върхове

```
match (v) delete v
```

```
match (p:Person {name: 'Бурак'}) delete p
```

```
match (child:Person)
      <-[:parent_of]-
      (p:Person {name: 'Али Ръза'})
delete child
```

Изтриване на ребра

```
match ()-[r]-() delete r
```

```
match (:Person {name: 'Неджля'})
      <-[relationship:parent_of]-
      (:Person {name: 'Али Ръза'})
delete relationship
```

"Неджля, ти вече не си моя дъщеря! Не стъпвай повече в тази къща!"

- Али Ръза, 2010

Промяна

```
match (ali:Person {name: 'Али Ръза'})
set ali.лобо = false
return ali
```

Прости агрегации

```
match (p:Person)<-[:parent_of]-(child:Person)
return p.name, count(child) as child_count
```

Сортиране

```
match (p:Person)<-[:parent_of]-(child:Person)
return p.name, count(child) as child_count
order by child_count desc
```

Филтриране с where

```
match (p:Person)-[:parent_of]->(child:Person)
where p.name = 'Али Ръза'
return p.name, count(child) as child_count
order by child_count desc
```

Междинни стъпки с with

```
match (p:Person)-[:parent_of]->(child:Person)
with p.name as name, count(child) as child_count
return name, child_count
```

```
match (p:Person)-[:parent_of]->(child:Person)
with p.name as name, count(child) as child_count
where child_count > 2
return name, child_count
order by child_count desc
```

Обхождания с променлива дължина

Всички внуци на Али Ръза

```
match (:Person {name: 'Али Ръза'})-
      [:parent_of*2]->(grandchild:Person)
return grandchild.name
```

```
match (:Person {name: 'Али Ръза'})-
      [path:parent_of*1..10]-(relative:Person)
return relative.name, length(path)
```

Обхождания с променлива дължина

Последното ще върне и него самия – ще го филтрираме с where

```
match (ali:Person {name: 'Али Ръза'})-
      [:parent_of*]-(relative:Person)
where relative <> ali
return relative.name
```

Обхождания с променлива дължина

Всъщност ще върне и жена му и нейните роднини. Ако не ги искаме:

```
match (ali:Person {name: 'Али Ръза'})<-
      [:parent_of*0..]-(:Person)-
      [:parent_of*0..]->(relative:Person)
where relative <> ali
return relative.name
```

match с няколко шаблона

Предната заявка може да се напише и така

```
match (common_parent:Person)-[:parent_of*0..]->
      (ali:Person {name: 'Али Ръза'}),
      (common_parent)-[:parent_of*0..]->
      (relative:Person)
where relative <> ali
return relative.name
```

Всички хора, с които има общ родител. В двата match-а `common_parent` ще е един и същ, защото използваме едно и също име.

Колекции и имена

Фикрет иска някой да я запознае с Бурак:

```
match path=(:Person {name: 'Фикрет'})-[:knows*]
           ->(:Person {name: 'Бурак'})
return extract(person in nodes(path) | person.name)
```

Най-къс път

```
match (fikret:Person {name: 'Фикрет'}),
      (burak:Person {name: 'Бурак'}),

      path=shortestPath((fikret)-[:knows*]->(burak))

return extract(person in nodes(path) | person.name)
```

Въпроси за Neo4j/Cypher?


OrientDB®

OrientDB

- Отворен код (Apache2 license)
- Безплатна дори за комерсиално използване
- Има и enterprise версия
- Консистентност
- Транзакции
- Репликация
- Език за заявки – SQL с разширения или Gremlin

OrientDB SQL

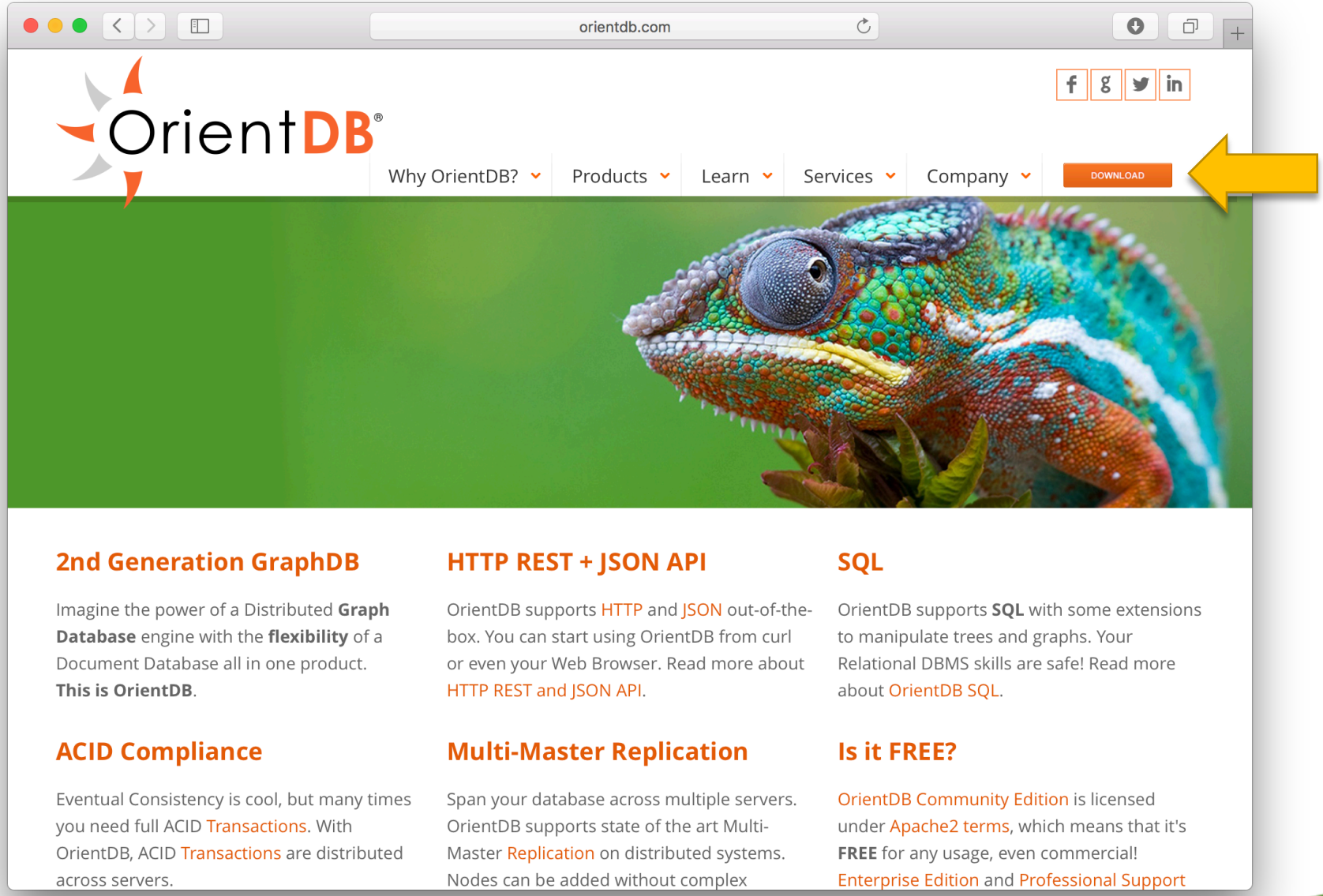
- Прилича на SQL
- Не поддържа част от по-сложните конструкции в SQL
- Има допълнителни функции за работа с графи
- Простите заявки стават лесно
- Сложните заявки са ад и хакерии

Gremlin

$G = (V, E)$

- Отворен език за заявки към графови данни
- Поддържа се от много графови БД, вкл. Neo4j, OrientDB, Titan и т.н.
- Императивен, за разлика от Cypher и OrientDB SQL
- Доста по-сложен от тях
- Но е по-мощен
- И имате повече контрол върху обработката

Инсталация



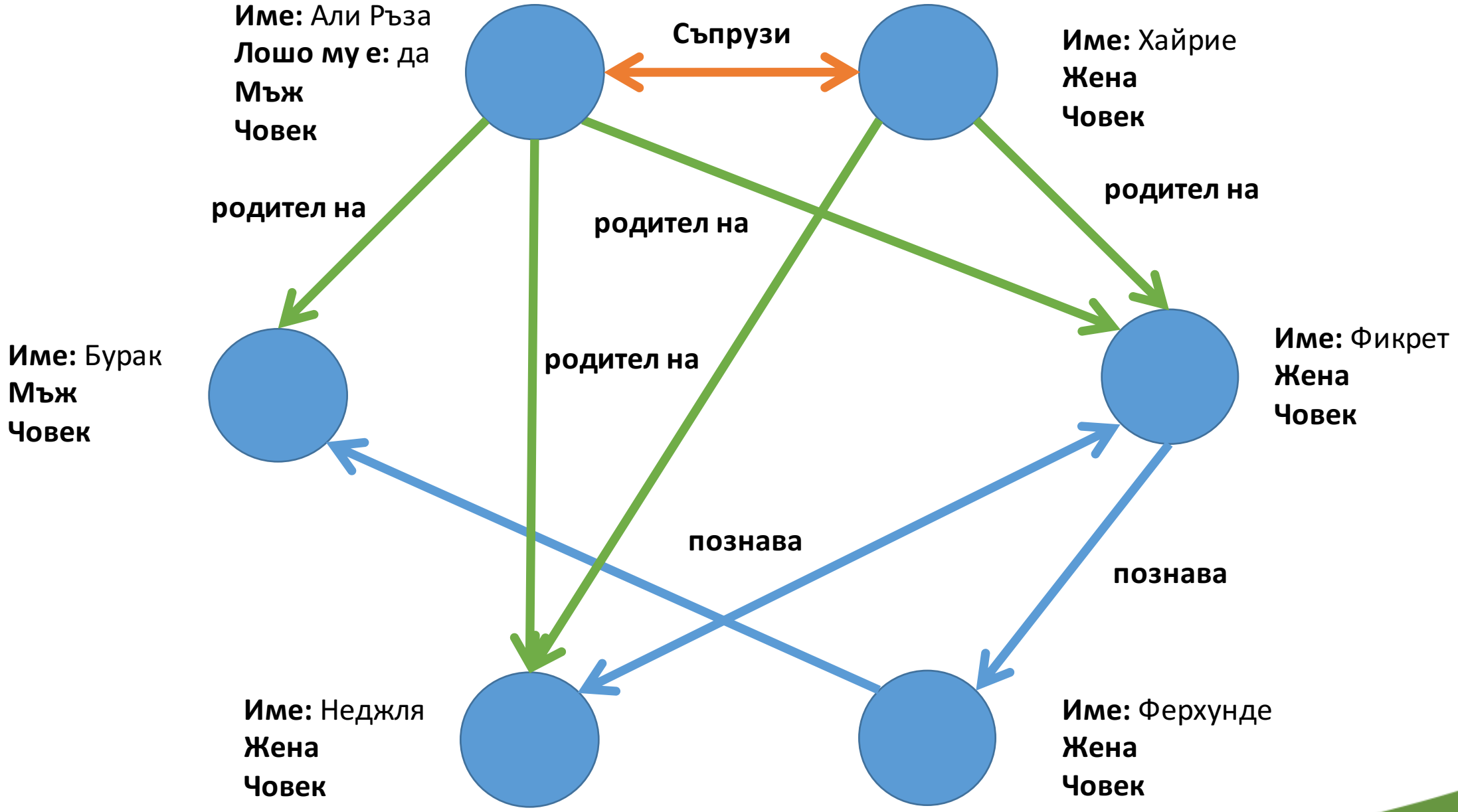
astea SOLUTIONS

Инсталация

След като го свалите…

- Разархивирате някъде
- Пускате bin/server.sh
- Използвате bin/gremlin.sh за Gremlin конзола (http://orientdb.com/docs/2.0/orientdb.wiki/Gremlin.html)
- Или http://localhost:2480 за OrientDB SQL конзола

Примери с Gremlin


Име: Али Ръза
Лошо му е: да
Мъж
Човек
Съпрузи
Име: Хайрие
Жена
Човек
родител на
родител на
родител на
Име: Бурак
Мъж
Човек
родител на
Име: Фикрет
Жена
Човек
познава
познава
Име: Неджля
Жена
Човек
Име: Ферхунде
Жена
Човек

Добавяне на върхове

```
ali      = g.addVertex('person', 'name', 'Али Ръза', 'лобо', true)
hayrie   = g.addVertex('person', 'name', 'Хайрие')
burak    = g.addVertex('person', 'name', 'Бурак')
fikret   = g.addVertex('person', 'name', 'Фикрет')
nedjlya  = g.addVertex('person', 'name', 'Неджля')
ferhunde = g.addVertex('person', 'name', 'Ферхунде')
```

Добавяне на ребра

```
ali    .addEdge('married_to', hayrie)
hayrie.addEdge('married_to', ali)

ali    .addEdge('parent_of', fikret)
ali    .addEdge('parent_of', burak)
ali    .addEdge('parent_of', nedjlya)
hayrie.addEdge('parent_of', nedjlya)
hayrie.addEdge('parent_of', fikret)

nedjlya.addEdge('knows', fikret)
fikret .addEdge('knows', nedjlya)
fikret .addEdge('knows', ferhunde)
fikret .addEdge('knows', burak)
```

* Разчита се, че ще се изпълни заедно с предната заявка.

Показване на всички върхове

```
gremlin> g.V()
==>v[#9:0]
==>v[#9:1]
==>v[#9:2]
==>v[#9:3]
==>v[#9:4]
==>v[#9:5]
```

Показване на всички върхове

```
gremlin> g.V().map
==>{name=Али Ръза, lobo=true}
==>{name=Хайрие}
==>{name=Бурак}
==>{name=Фикрет}
==>{name=Неджля}
==>{name=Ферхунде}
```

Показване на всички ребра

```
gremlin> g.E()
==>e[#11:0][#9:0-married_to->#9:1]
==>e[#11:1][#9:1-married_to->#9:0]
==>e[#12:0][#9:0-parent_of->#9:3]
==>e[#12:1][#9:0-parent_of->#9:2]
==>e[#12:2][#9:0-parent_of->#9:4]
==>e[#12:3][#9:1-parent_of->#9:4]
==>e[#12:4][#9:1-parent_of->#9:3]
==>e[#13:-2][#9:4-knows->#9:3]
==>e[#13:-3][#9:3-knows->#9:4]
==>e[#13:-4][#9:3-knows->#9:5]
==>e[#13:-5][#9:3-knows->#9:2]
```

Търсене на върхове

```
gremlin> ali = g.V().has('name', 'Али Ръза')
==>v[#9:0]

gremlin> ali.name
==>Али Ръза

gremlin> ali.лобо
==>true

gremlin> ali.map
==>{name=Али Ръза, лобо=true}
```

Изтриване

```
vertex = g.V().has(...)
g.removeVertex(vertex)

edge = g.E().has(...)
g.removeEdge(edge)
```

Промяна

```
vertex = g.V().has(...)
vertex.lobo = false
```

Прости обхождания

```
gremlin> ali.out('parent_of').map
==>{name=Фикрет}
==>{name=Бурак}
==>{name=Неджля}
```

```
gremlin> g.V().as('child')
         .in('parent_of').has('name', 'Али Ръза')
         .back('child').map
==>{name=Бурак}
==>{name=Фикрет}
==>{name=Неджля}
```

Прости обхождания

```
gremlin> ali.outE('parent_of')
==>e[#12:0][#9:0-parent_of->#9:3]
==>e[#12:1][#9:0-parent_of->#9:2]
==>e[#12:2][#9:0-parent_of->#9:4]

gremlin> ali.outE('parent_of').inV.map
==>{name=Фикрет}
==>{name=Бурак}
==>{name=Неджля}
```

Обхождане с Gremlin

- Една gremlin заявка представлява поток (pipe)
- Като при функционалното програмиране
- С повече възможности – видяхте "връщане назад" и именувани стъпки
- Поддържа се и разклонение/сливане на потоци
- Поддържат се странични ефекти

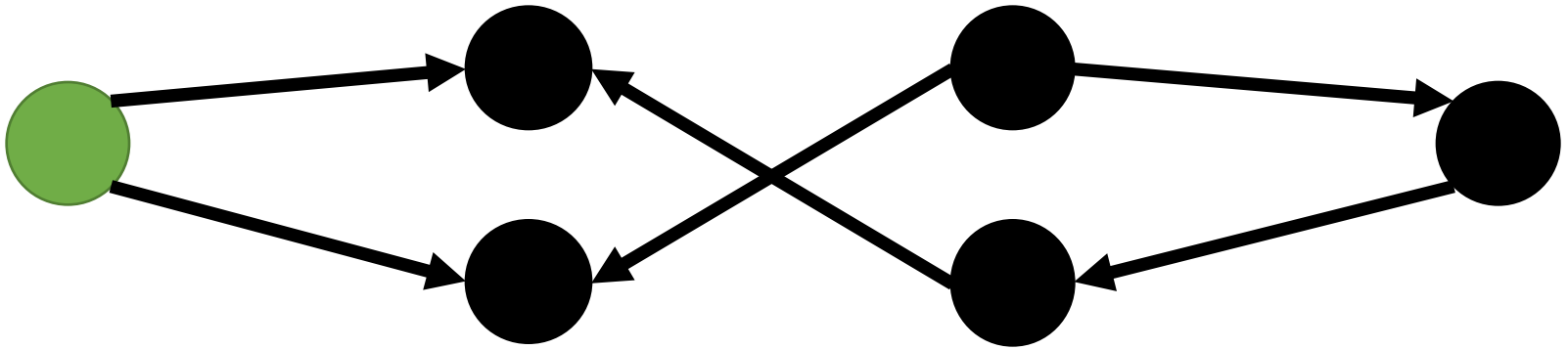
Обхождане с Gremlin

- outE – ребра "излизащи" от текущите върхове
- inE – "влизащи" ребра
- outV – върхове, от които текущото ребро "излиза"
- inV – върхове, в които "влиза"
- out = outE.inV
- in = inE.outV
- Какво прави outE.outV ?

- Всички приемат опционален аргумент – тип на реброто/върха

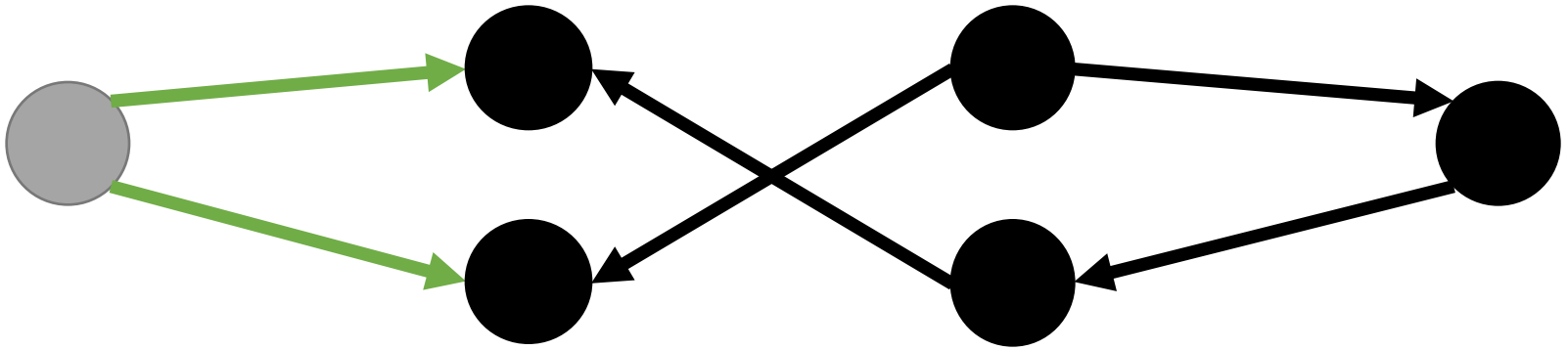
Прости обхождания

ali



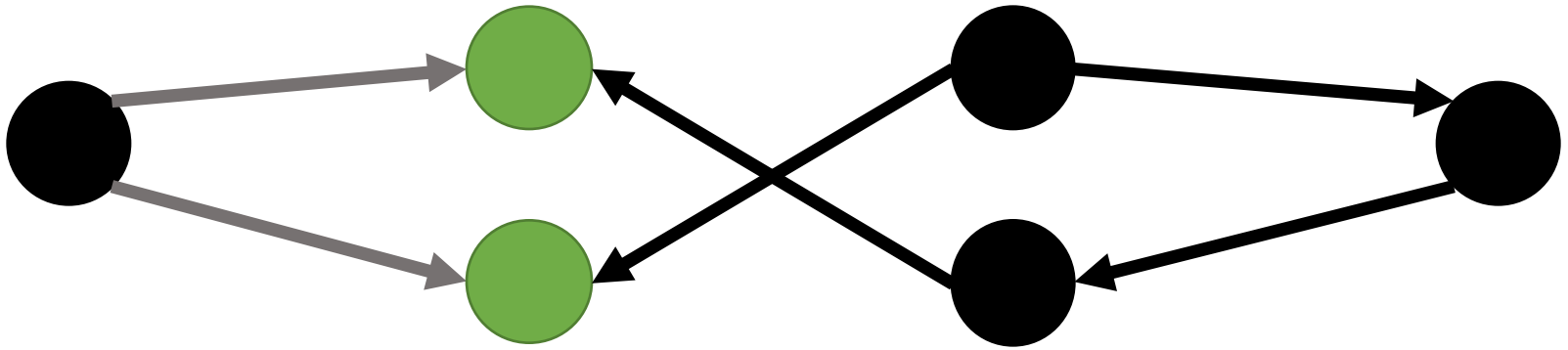
Прости обхождания

```
ali.outE
```



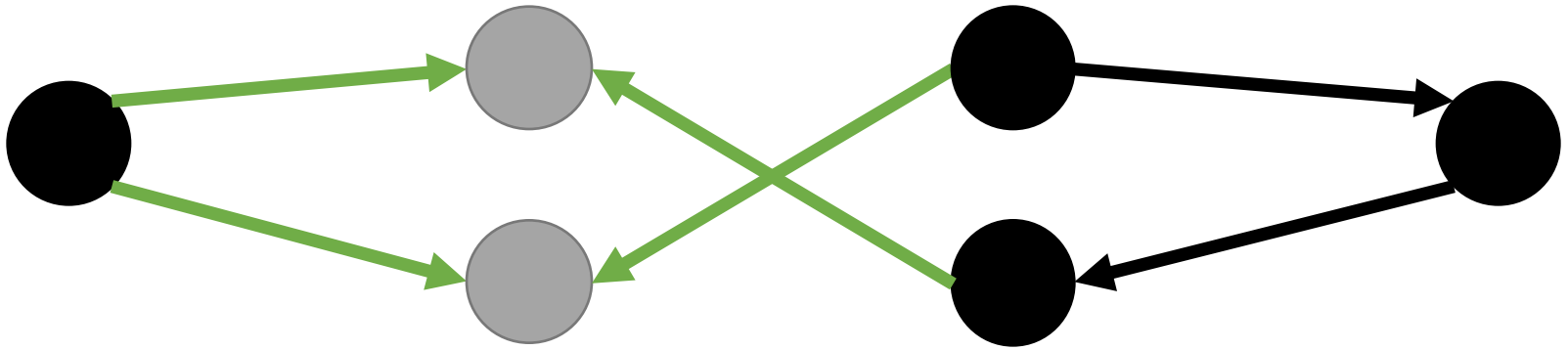
Прости обхождания

```
ali.outE.inV
```



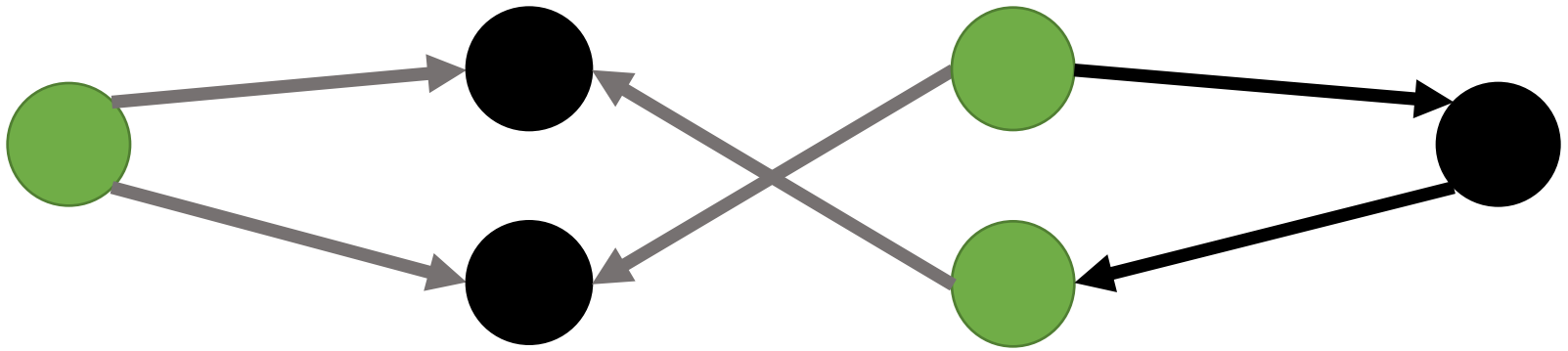
Прости обхождания

```
ali.outE.inV.inE
```



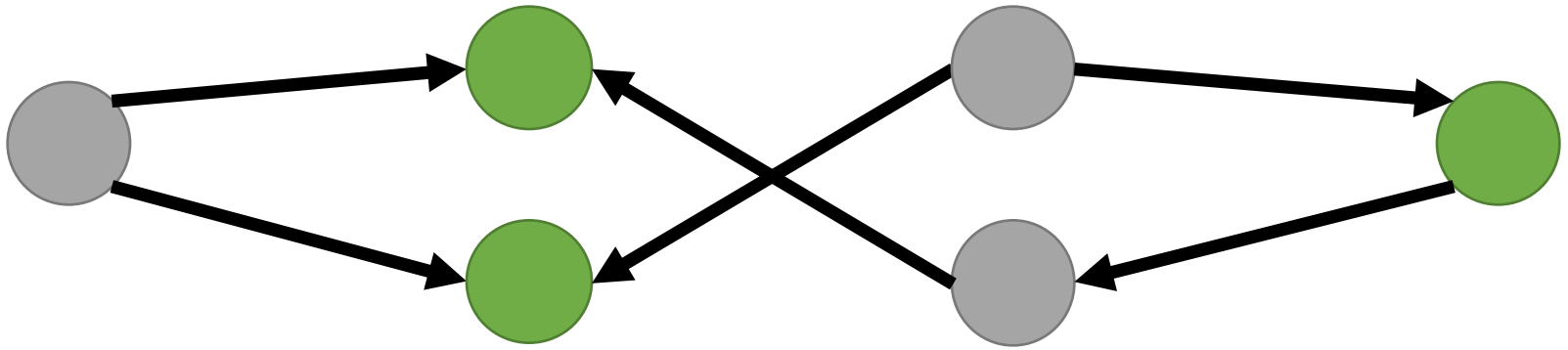
Прости обхождания

```
ali.outE.inV.inE.outV
```



Прости обхождания

```
ali.outE.inV.inE.outV.out
```



Прости обхождания (отново)

```
gremlin> ali.out('parent_of').map
==>{name=Фикрет}
==>{name=Бурак}
==>{name=Неджля}


gremlin> g.V().as('child')
         .in('parent_of').has('name', 'Али Ръза')
         .back('child').map
==>{name=Бурак}
==>{name=Фикрет}
==>{name=Неджля}
```

Прости обхождания (отново)

```
gremlin> ali.outE('parent_of')
==>e[#12:0][#9:0-parent_of->#9:3]
==>e[#12:1][#9:0-parent_of->#9:2]
==>e[#12:2][#9:0-parent_of->#9:4]

gremlin> ali.outE('parent_of').inV.map
==>{name=Фикрет}
==>{name=Бурак}
==>{name=Неджля}
```

Прости агрегации

```
gremlin> g.V().transform {
           [it.name, it.out('parent_of').count()]
         }

==>[Али Ръза, 3]
==>[Хайрие, 2]
==>[Бурак, 0]
==>[Фикрет, 0]
==>[Неджля, 0]
==>[Ферхунде, 0]
```

Прости агрегации

```
gremlin> g.V().transform {
           [it.name, it.out('parent_of').count()]
         }.filter { it[1] > 0 }.sort { -it[1] }

==>[Али Ръза, 3]
==>[Хайрие, 2]
```

Обхождания с променлива дължина

```
parents = ali.as('parents')
    .in('parent_of')
    .loop('parents') { it.loops < 100 } { true }

parents.as('children')
    .out('parent_of')
    .loop('children') { it.loops < 100 } { true }
```

Вървим произволен брой стъпки нагоре и събираме върховете в parents, после произволен брой стъпки надолу, събирайки децата.

Въпроси?