Новосибирский государственный университет
*НАСТОЯЩАЯ НАУКА

Графовая База Данных на Clojure: Clojure-GraphQL

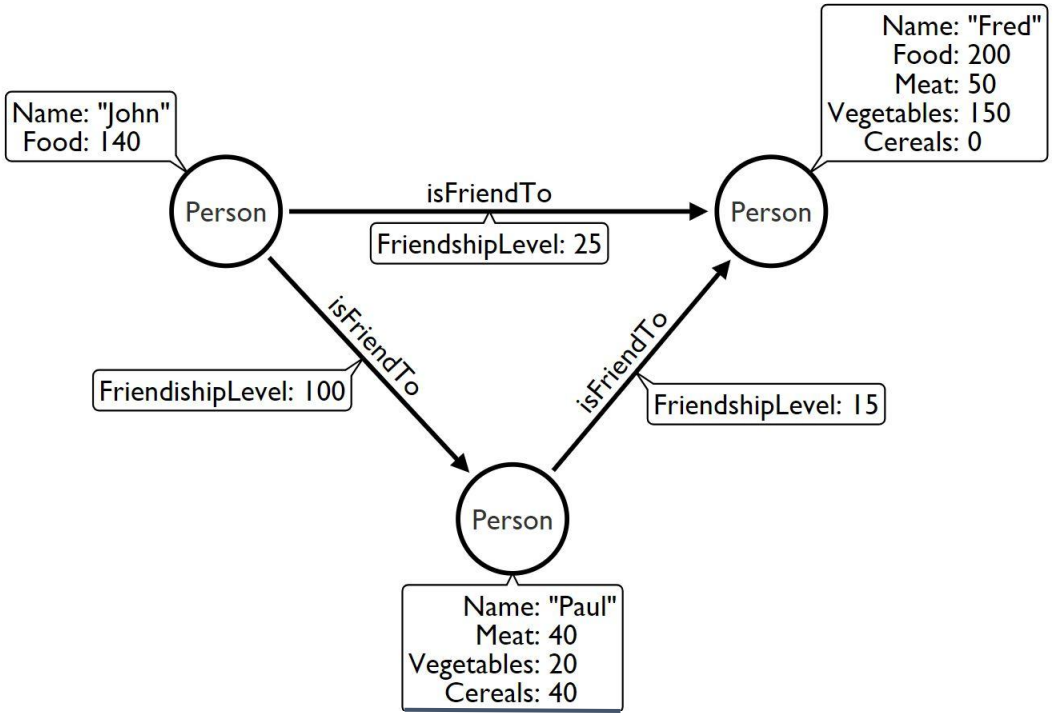
Выполнили студенты гр. 21225.
Вершинин Максим Олегович и Михайлапов Денис Иванович.

2022

Введение: базовые понятия

Граф – Набор **вершин** и набор **рёбер.**

- Вершины соединяются ребрами.
- Рёбра имеют направление (ориентированный граф).
- Вершины и рёбра могут содержать произвольные данные.



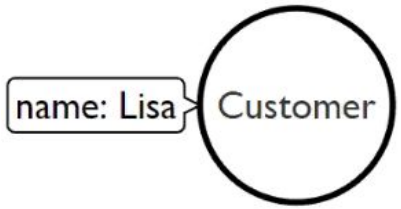
База данных — совокупность данных, организованных в соответствии с концептуальной структурой, описывающей характеристики этих данных и взаимоотношения между ними, которая поддерживает одну или более областей применения

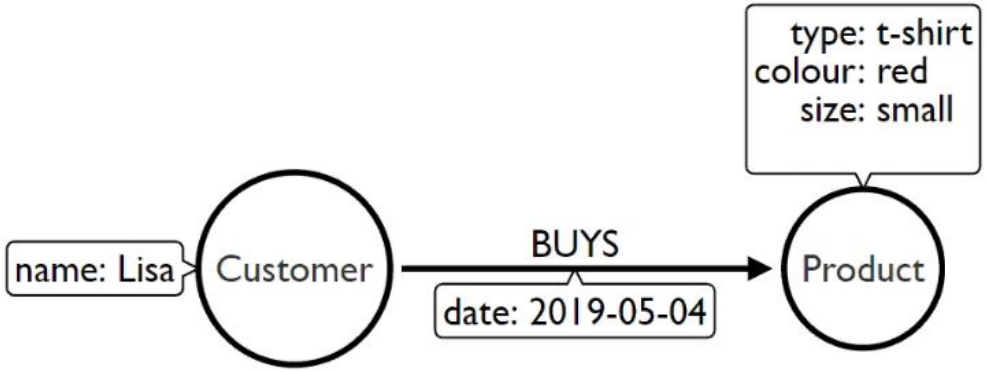
Графовая база данных – разновидность баз данных, информация в которой хранится и обрабатывается в графовой структуре. (NoSQL)

Примеры:
Neo4j, ArangoDB, InfiniteGraph, Amazon Neptune, и др.

Введение: CREATE

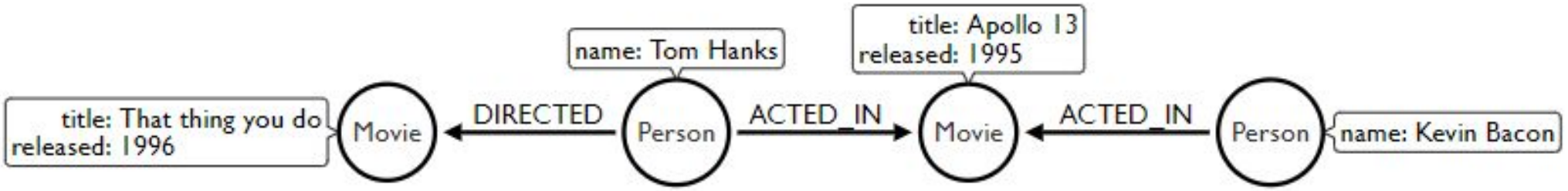
//Create a customer node called “Lisa”
CREATE (p:Customer {name: “Lisa”});





//Create the pattern of “Lisa” BUYS “t-shirt”
CREATE (:Customer {name: “Lisa”})-[:BUYS {data: 2019-05-04}]->(:product {type:"t-shirt", color: “red”, size: “small”});

Введение: MATCH WHERE




```
//Find all the movies Tom Hanks acted in
MATCH (:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m:Movie)
RETURN m.title;
```
—> ["Apollo 13"]

```
//Find all of the co-actors Tom Hanks have worked with
MATCH (:Person {name:"Tom Hanks"})-->(:Movie)<-[:ACTED_IN]-(coActor:Person)
RETURN coActor.name;
```
—> ["Kevin Bacon"]

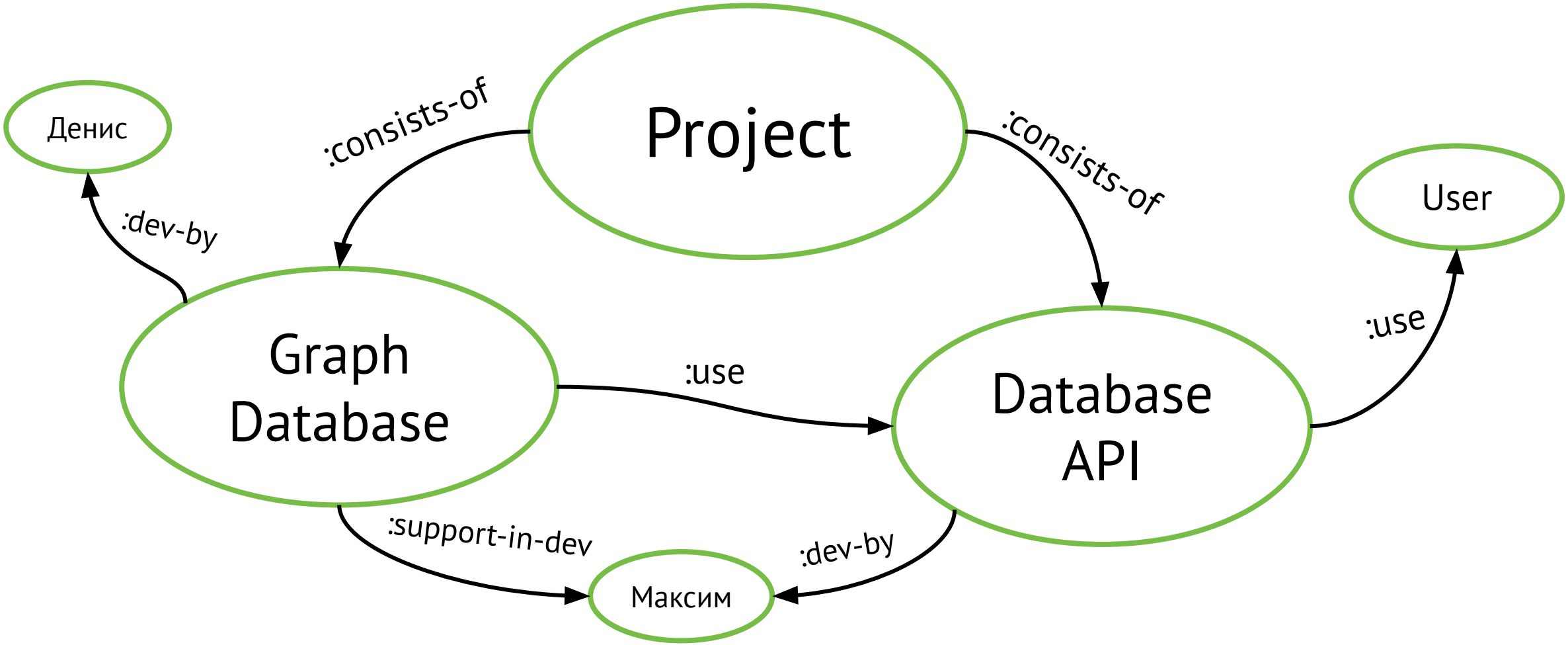
```
//Get the titles of movies released earlier than 2000 year
MATCH (m:Movie) WHERE m.released < 2000
RETURN m.title;
```
—> ["Apollo 13", "That thing you do"]

Введение: применение

- Бизнес-аналитика
- Жизненный цикл предприятий
- Управление сетями и дата-центрами
- Социальные приложения, рекомендательные сервисы и коммерция
- Маркетинг, реклама, PR
- Геосервисы, геоприложения

Подробнее: “Графовый анализ — обзор и области применения” [2] (habr.com)

Проект


Денис
:consists-of
Project
:consists-of
User
:dev-by
Graph
Database
:use
Database
API
:use
:support-in-dev
:dev-by
Максим

Graph Database

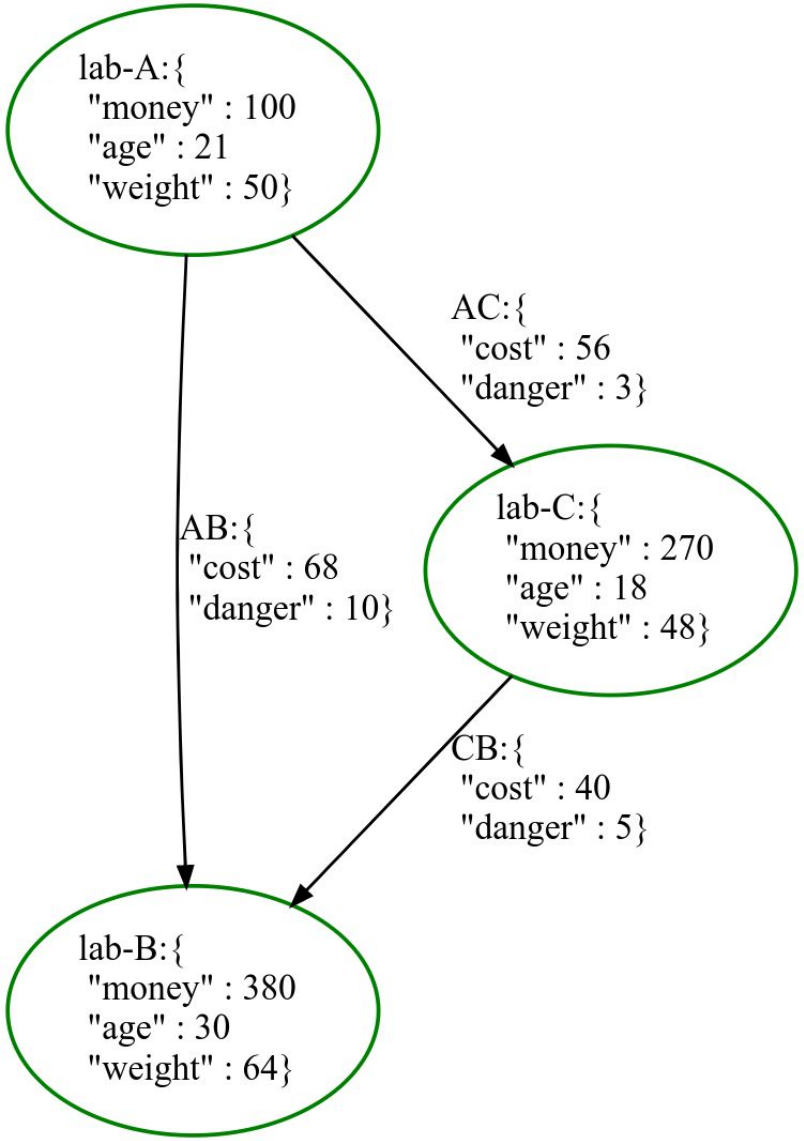
Возможности:

- **Добавление**, **удаление** вершин и рёбер**.**
- **Сохранение** и **загрузка** в/из Json и Byte форматы (средствами библиотеки *jsonista*).
- **Визуализация** графа с помощью *GraphViz* и *dorothy*

А так же

- MATCH<WHERE> (путей и проивол. графа)
- SET
- *Создание связей* между полученными узлами *после MATCH*

Далее подробнее



Graph Database: json graph scheme

```
{:metadata {},
 :adjacency
 {:A
  {:in-edges [],
   :out-edges
   {:B {:labels [:AB], :properties {:cost 68, :danger 10}},
    :C {:labels [:AC], :properties {:cost 56, :danger 3}}},
   :labels [:lab-A],
   :properties {:money 100, :age 21, :weight 50}},
  :B
  {:in-edges [:A :C],
   :out-edges {},
   :labels [:lab-B],
   :properties {:money 380, :age 30, :weight 64}},
  :C
  {:in-edges [:A],
   :out-edges {:B {:labels [:CB], :properties {:cost 40, :danger 5}}},
   :labels [:lab-C],
   :properties {:money 270, :age 18, :weight 48}}}}
```

Модель представления графа: **Adjacency List**.

- В БД зачастую Ноды являются основным объектом внимания, поэтому удобнее иметь вершинноориентированную модель графа.
- Neo4j использует такую модель.

(Так же есть популярная модель основанная на списке ребер json-graph-specification)

- ➢ **:metadata** – хранит служебную информацию о графе.
- ➢ **:adjacency** – список смежностей, непосредственно.
- ➢ **:in-edges** – список вершин входящих дуг
- ➢ **:out-edges** – список исходящих дуг (вместе с данными дуги).
- ➢ **:labels** – список меток (строковая информация)
- ➢ **:properties** – произвольный json с данными

Graph Database: save/load json graph

```clojure
(defn save-graph [graph ^String filename]
  (j/write-value
    (File. (str "./resources/" filename)) graph
    (j/object-mapper {:pretty true :encode-key-fn true})))

(defn decode-str-to-kw [json]
  (let [new-json (transient {})]
    (do
      (doseq [-key (sort (keys json))]
        (let [json-item (json -key)]
          (if (map? json-item) (assoc! new-json (keyword -key) (decode-str-to-kw json-item))
            (if (coll? json-item) (assoc! new-json (keyword -key) (mapv #(keyword %) json-item))
              (assoc! new-json (keyword -key) json-item)))))
      (persistent! new-json))))

(defn load-graph [^String filename]
    (decode-str-to-kw (j/read-value (File. (str "./resources/" filename)))))
```

jsonista: github.com/metosin/jsonista

По данным разработчиков быстрее всех загружает и сохраняет json файлы и форматы, также доступно сохранение в byte формат.

Однако плохо конвертирует string в keywords. Был написан декодер.

Graph Database: visualisation graph

```clojure
(defn save-graphviz
  ([graph filename] (save-graphviz graph filename :png))
  ([graph filename format]
   (println "save to:" (str path-to-images filename))
   (-> (graph2graphviz graph)
       (save! (str path-to-images filename) {:format format}))))

(defn show-graphviz [graph]
  (-> (graph2graphviz graph)
      show!))
```

GraphViz: graphviz.org
dorothy: github.com/daveray/dorothy

Был написан конвертер **graph2graphviz**, который преобразует наш формат графа в представление graphviz.

Для отрисовки и сохранения графа graphviz используется библиотека dorothy.

Graph Database: MATCH ways

```
(defn match-query [graph query-way & [only-ways]]
  (let [ways (get-matched-ways graph query-way)]
    (if (boolean only-ways) ways
     [ways (map #(merge (select-keys (graph :adjacency) %)) ways)])))
```

Определение query-way:

- Одиночная нода (query-node)
- Две ноды с ребром (query-edge)
- Последовательность нод и рёбер (query-edges)

get-matched-ways:

➔ query-node → ways-nodes

➔ query-edge → adjacency list matched edges → ways-edges

➔ query-edges → list of adjacency list matched edges → ways

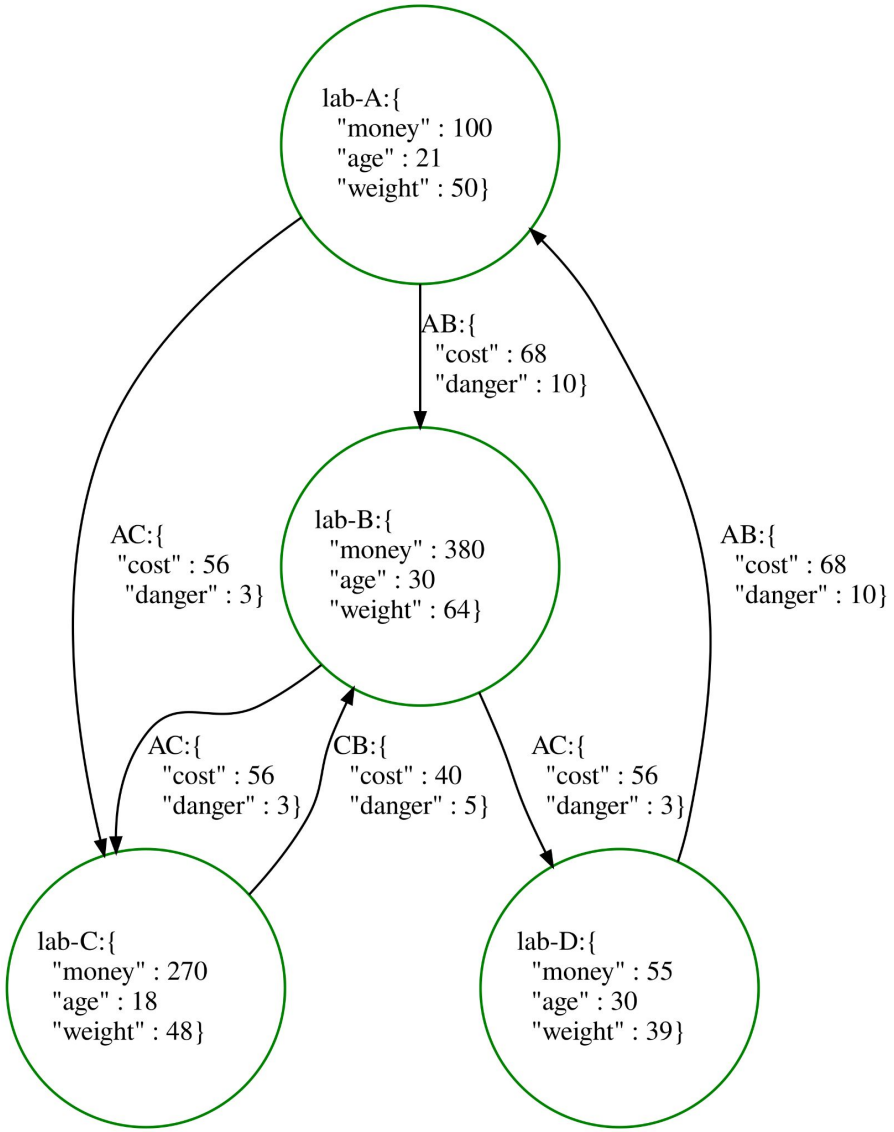
query-edge → adjacency list {:A [:B :C] :B [] ← удалится :C [:A :B]} → ways-edges: ((:A :B) (:A :C) (:C :A) (:C :B))

Match ways – разработка авторов.

- **query-way** имеет ту же структуру, что и граф, однако, когда нам надо указать, что нам подходит любое значение *label* или *property* мы используем "*nil*".
- **query-way** представляет собой последовательность нод соединённых рёбрами (по формату MATCH запроса)

- ways-nodes: ((:A) (:B) (:C))
- ways-edges: ((:A :B) (:A :C) (:C :B))
- ways: ((:A :B :C) (:A :C :B) (:B :A :C))

Graph Database: MATCH (пример)



Примеры query и ответы MATCH:

- query: () => ways: ((:A) (:B) (:C) (:D))
- query: ()-[AC]->() => ways: ((:A :C) (:B :C) (:B :D))
- query: (A)-[AB]->(C) => ways: ()
- query: (D)-[]->(D) => ways: ()
- query: ()-[AB]->()-[AC]->() => ways: ((:A :B :D) (:D :A :C))

Пример query: (:n:)-[AB]->(:k:)-[AC]->(), (:n: и :k: – не ноды, а переменные)

```
{:n
 {:in-edges [],
  :out-edges {:k {:labels [:AB], :properties {:cost 68, :danger 10}}},
  :labels nil,
  :properties nil},
 :k
 {:in-edges [:n],
  :out-edges
  {#uuid "bc1a1d66-1b62-4433-9ef8-11cd28ba6a1a"
   {:labels [:AC], :properties {:cost 56, :danger 3}}},
  :labels nil,
  :properties nil},
 #uuid "bc1a1d66-1b62-4433-9ef8-11cd28ba6a1a"
 {:in-edges [:k], :out-edges {}, :labels nil, :properties nil}}
```

Graph Database: Ullmann Matching*

Есть возможность искать произвольные подграфы

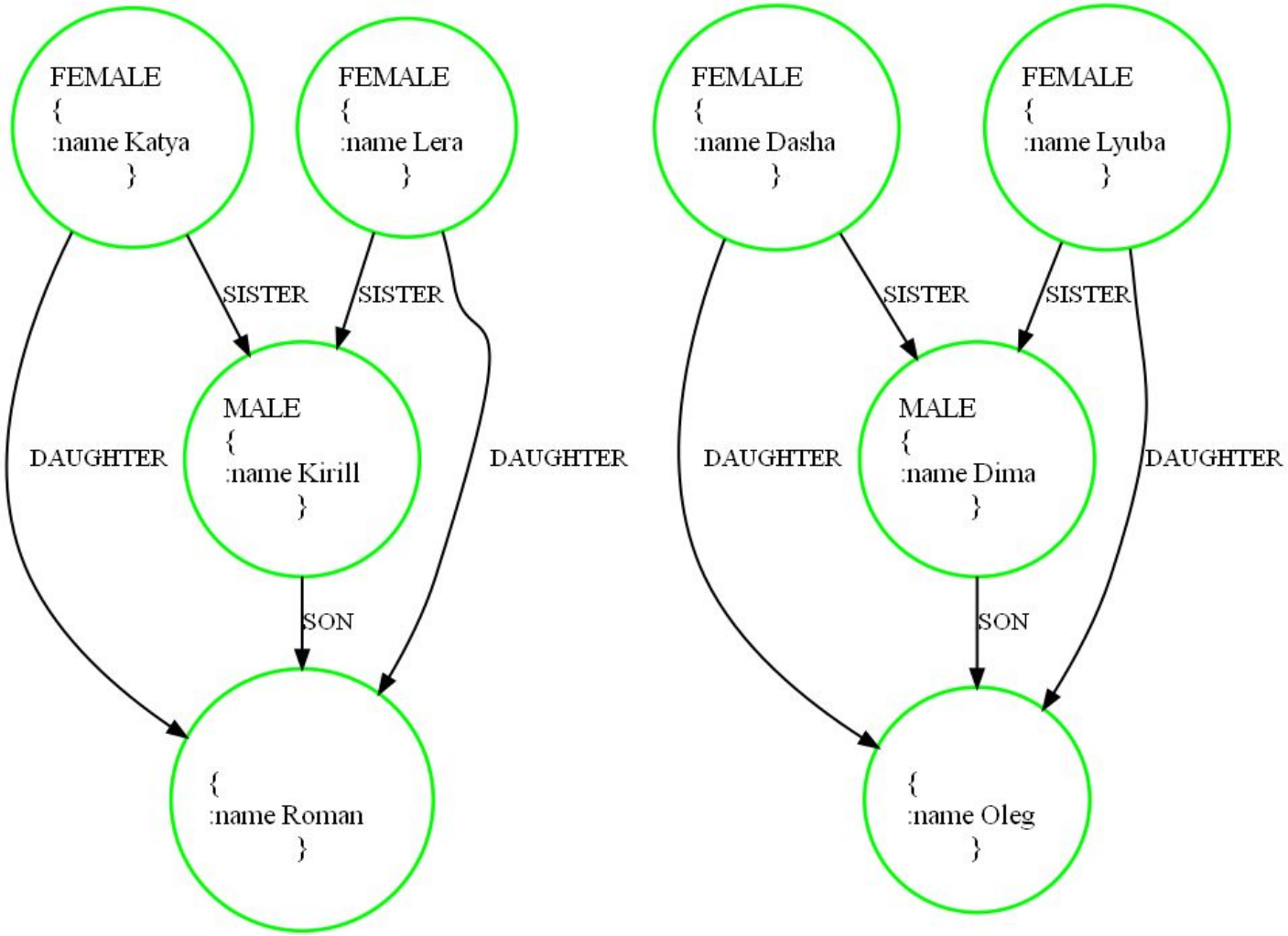
1. Находим кандидатов для каждого узла из запроса;
2. Удаляем невозможных кандидатов, анализируя ребра;
3. Далее обходим граф по каждой вершине в глубину достижимости вершин, убирая неподходящие группы.

*M. Saltz and J. Miller.: “A fast algorithm for subgraph pattern matching on large labeled graphs.” Master’s thesis, UGA, Athens, GA, USA, (2013).

Graph Database: link

- Создание связей между узлами, которые получились в результате *match*.

```
match (c1)-[e1:SON]->(p)<-[e2:DAUGHTER]-(c2),(g:GRANDFATHER)
link (c2)-[:SISTER]->(c1),(g)-[:PARENT]->(p)
```



Graph Database: label INDEX (сортировка по лейблам)

```
{:metadata {},
 :adjacency
 {:I {:in-edges [], :out-edges {}, :labels [], :properties {}},
  :A {:in-edges [], :out-edges {}, :labels [:lab-1], :properties {}},
  :F {:in-edges [], :out-edges {}, :labels [:lab-2], :properties {}},
  :D {:in-edges [], :out-edges {}, :labels [:lab-1], :properties {}},
  :B {:in-edges [], :out-edges {}, :labels [:lab-2], :properties {}},
  :J {:in-edges [], :out-edges {}, :labels [], :properties {}},
  :C {:in-edges [], :out-edges {}, :labels [:lab-3], :properties {}},
  :E {:in-edges [], :out-edges {}, :labels [:lab-1], :properties {}},
  :G {:in-edges [], :out-edges {}, :labels [:lab-3], :properties {}},
  :H {:in-edges [], :out-edges {}, :labels [:lab-3], :properties {}}}}
```

Пусть у нас есть граф состоящий только из нод. У каждой ноды может быть свой лейбл и обычно он трактуется как тип ноды. Хотя не запрещается одной ноде иметь несколько лейблов.

Данная структура называется “index” и находится в “metadata”.

Graph Database: label INDEX (:index)

```
{:metadata
 {:index
  {:lab-1 (:E :D :A),
   :lab-2 (:B :F),
   :lab-3 (:H :G :C),
   #uuid "00000000-0000-0000-0000-000000000000" (:J :I)}},
 :adjacency
 {:I {:in-edges [], :out-edges {}, :labels [], :properties {}},
  :A {:in-edges [], :out-edges {}, :labels [:lab-1], :properties {}},
  :F {:in-edges [], :out-edges {}, :labels [:lab-2], :properties {}},
```

Идея индексации заключается в сортировке нод по лейблам.

Рассмотрим случаи labels у query-node:

➔ labels = [:lab-1] ⟶ берём из индекса список нод “lab-1” и осуществляем поиск только по этим нодам.

➔ labels = [:lab-1 :lab-2 …] ⟶ также берём из индекса списоки нод “lab-1”, “lab-2” и т.д., а затем берём пересечение и осуществляем поиск.

➔ labels = [] ⟶ в этом случае есть зарезервированное место в индексе для “пустого” лейбла (uuid v0)

Graph Database: label INDEX (code)

```
(defn add-labels-in-index [graph labels]
  (let [index-map (get-index-map (graph :adjacency) (wrap labels))
        metadata (graph :metadata)]
    (if (empty? index-map) metadata
      (assoc metadata :index (merge metadata index-map)))))

(defn delete-labels-in-index [metadata labels]
  (if (nil? labels) (dissoc metadata :index)
    (if (empty? labels) metadata
     (let [index-map (metadata :index {})]
          (if (= (disj (keysSet index-map) uuid-v0) (set labels))
            (dissoc metadata :index)     ;uuid-v0 key only
            {:index
             (assoc (delete-items index-map labels) uuid-v0
               (concat (merge-by-keys index-map labels) (index-map uuid-v0)))})))))
```

index-map →

```
{:lab-1 (:E :D :A),
 :lab-2 (:B :F),
 :lab-3 (:H :G :C),
 #uuid "00000000-0000-0000-0000-000000000000" (:J :I)}},
```

Database API

- Язык запросов - аналог Cypher [9] (парсинг посредством **instaparse**);
- База данных - дерево версий графов, в которой есть возможность откатиться до предыдущего варианта с помощью *undo*
- Запросы конструируются и исполняются группами:
  - Каждая группа имеет свой контекст во время исполнения (локальные переменные, переданные параметры, возвращаемое значение);
  - Переменные содержат актуальные элементы из текущей версии графа;
- Реализованные команды:
  - CREATE
  - DELETE
  - MATCH <WHERE>
  - SET
  - UNDO
  - RETURN

instaparse: https://github.com/Engelberg/instaparse

Далее demo

Проект на github:
github.com/max-509/clojure-graphql

Перспективы

Внутренняя логика

- Основательная документация и переосмысление (формализация) концепций.
- Версионность, Транзакционность меняющих состояние операций.
- Параллельность в обработке MATCH запроса (по нодам).

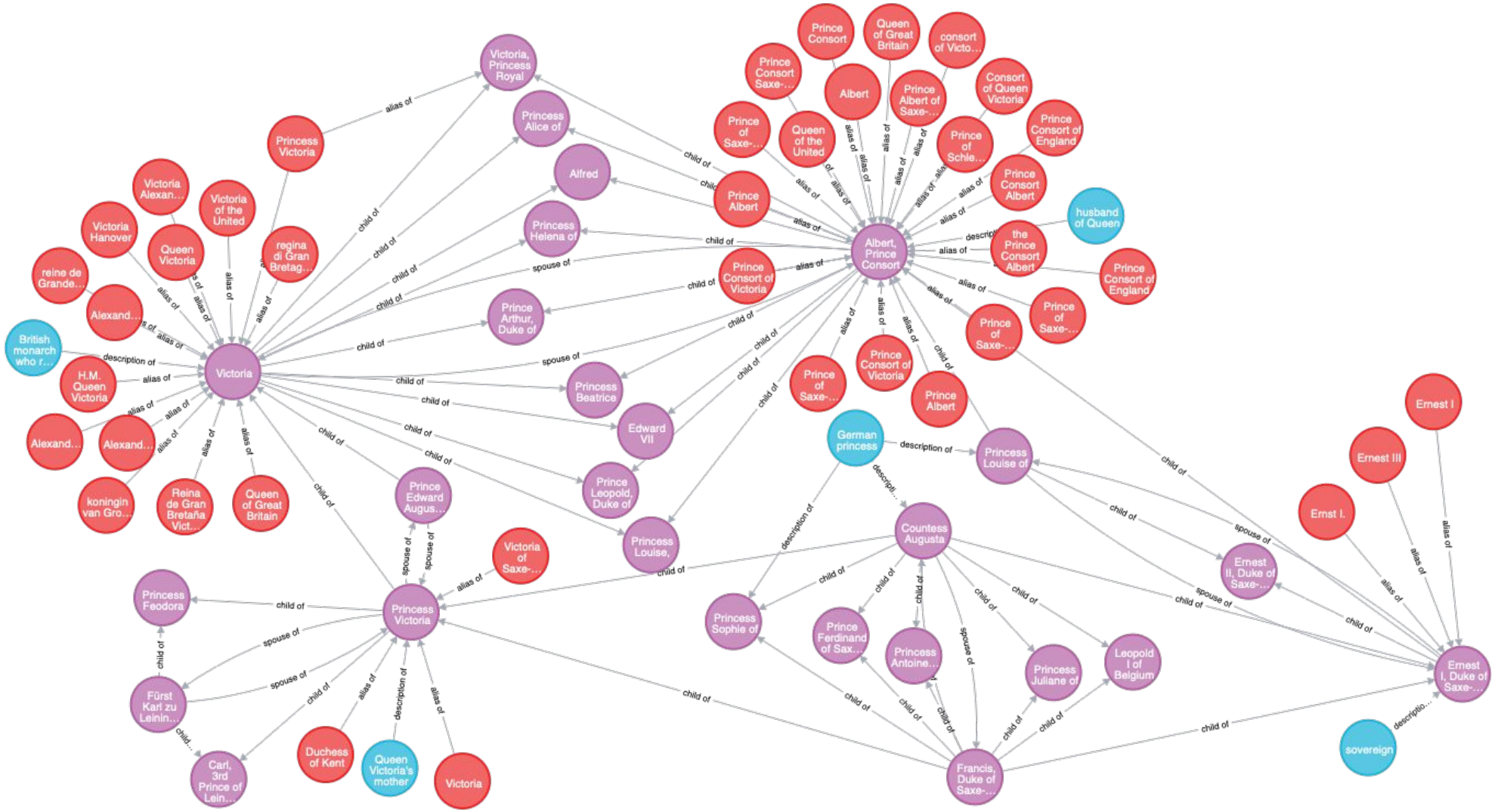
Функциональность

- Вставка подграфа (INSERT).
- Сумма и разница графов (SUM, SUB).
- Команда FOREACH и пользовательские функции обработки информации.
- Поддержка популярных алгоритмов на графах (кратчайший путь, максимальный поток)

Спасибо за внимание

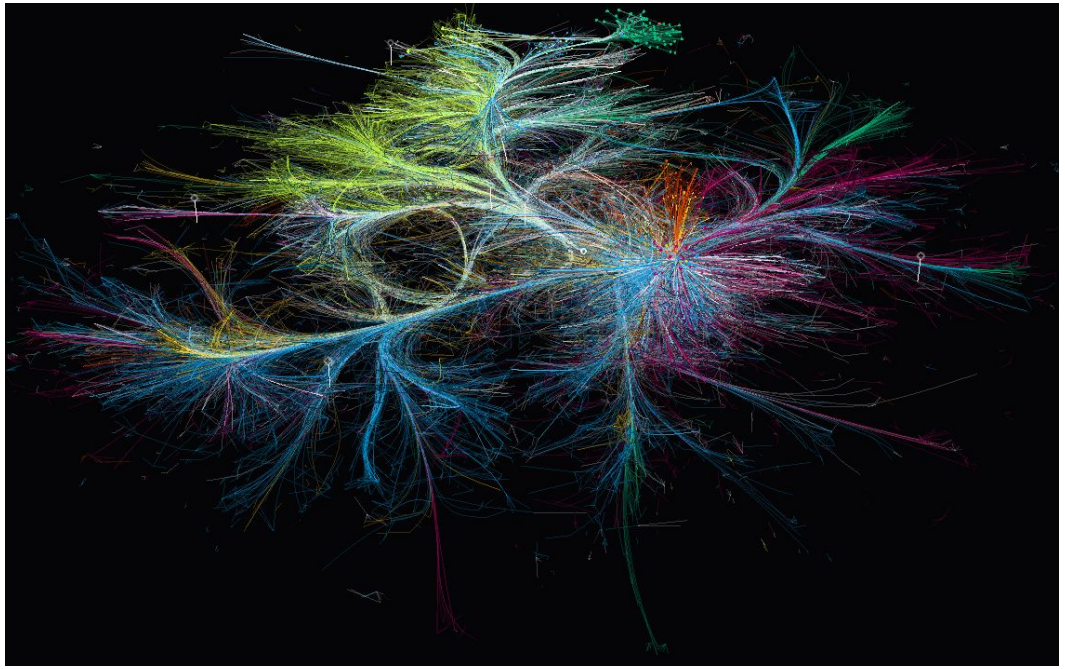
[1] Карта кросс-цитирования журнала Nature: www.nature.com/immersive/d41586-019-03165-4/index.html
[2] “Графовый анализ — обзор и области применения”: habr.com/ru/company/glowbyte/blog/594221/
[3] Язык Cypher: neo4j.com/docs/cypher-manual/current/
[4] Mhedhbi, A., Gupta, P., Khaliq, S., Salihoglu, S.: “A+ Indexes: Tunable and Space-Efficient Adjacency Lists in Graph Database Management Systems.” CoRR arXiv:2004.00130 (2021)
[5] Gala B., Javier T., Antonio V.: “Improving query performance on dynamic graphs” Softw. Syst. Model. 20(4), 1011–1041 (2021). doi:10.1007/s10270-020-00832-3
[6] M. Han, H. Kim, G. Gu, K. Park, and W. Han.: “Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together.” In SIGMOD, pages 1429–1446, (2019)
[7] Sharma, Chandan, and Roopak Sinha.: “A Schema-First Formalism for Labeled Property Graph Databases: Enabling Structured Data Loading and Analytics.” BDCAT ’19, ACM Press, (2019), pp. 71–80 doi:10.1145/3365109.3368782
[8] Fernandes, D., Bernardino, J.: “Graph databases comparison: Allegrograph, arangodb, infinitegraph, neo4j, and orientdb.” In: Data, pp. 373–380 (2018)
[9] N. Francis, A. Green, P. Guagliardo, L. Libkin et al.: “Cypher: An evolving query language for property graphs”(2018).
[10] Saltz et al: "DualIso: An Algorithm for Subgraph Pattern Matching in Very Large Labeled Graphs", IEEE International Congress on Big Data (2014).
[11] M. Saltz and J. Miller.: “A fast algorithm for subgraph pattern matching on large labeled graphs.” Master’s thesis, UGA, Athens, GA, USA, (2013).

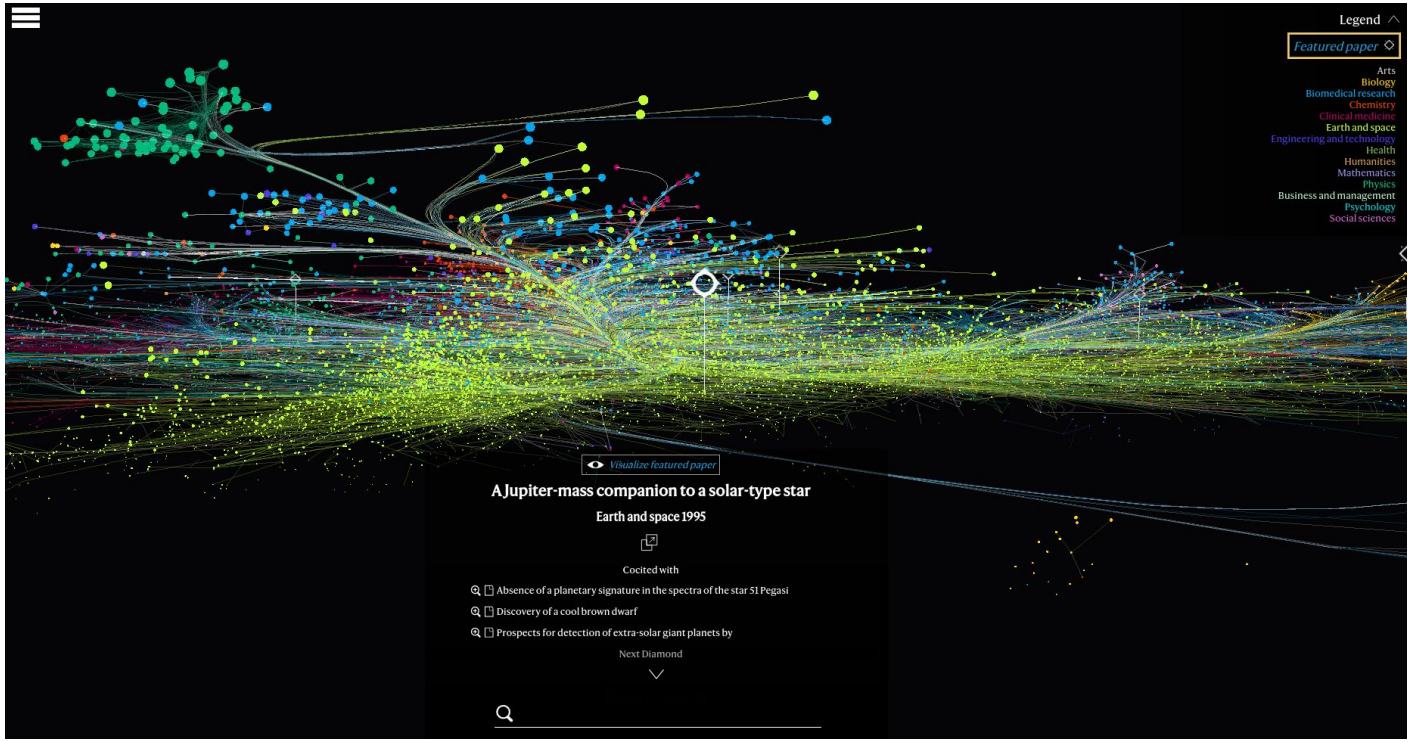
Приложения


Victoria
Victoria, Princess Royal
Princess Victoria
Princess Alice of
Alfred
Princess Helena of
Prince Arthur, Duke of
Princess Beatrice
Edward VII
Prince Leopold, Duke of
Princess Louise,
Prince Edward Augus...
Victoria of Saxe-...
Princess Feodora
Fürst Karl zu Leinin...
Carl, 3rd Prince of Lein...
Duchess of Kent
Queen Victoria's mother
Victoria
Victoria Alexan...
Victoria Hanover
Queen Victoria
Victoria of the United
regina di Gran Bretag...
reine de Grande...
Alexand...
British monarch who r...
H.M. Queen Victoria
Alexand...
Alexand...
koningin van Gro...
Reina de Gran Bretaña Vict...
Queen of Great Britain
Albert, Prince Consort
Prince Albert
Prince Consort of Victoria
Prince Consort
Queen of Great Britain
consort of Victo...
Prince Consort Saxe-...
Albert
Prince Albert of Saxe-...
Prince of Saxe-...
Queen of the United
Consort of Queen Victoria
Prince of Schle...
Prince Consort of England
Prince Consort Albert
husband of Queen
the Prince Consort Albert
Prince Consort of England
Prince of Saxe-...
Prince of Saxe-...
Prince of Saxe-...
Prince Consort of Victoria
Prince Albert
German princess
Princess Louise of
Countess Augusta
Princess Sophie of
Prince Ferdinand of Sax...
Princess Antoine...
Princess Juliane of
Leopold I of Belgium
Francis, Duke of Saxe-...
Ernest I
Ernest III
Ernst I.
Ernest II, Duke of Saxe-...
Ernest I, Duke of Saxe-...
sovereign
child of
spouse of
alias of
description of

Приложения

Графы бывают огромными!





Карта кросс-цитирования журнала Nature [1] (скриншот сделан на сайте https://www.nature.com/immersive/d41586-019-03165-4/index.html)