Social network visualization

JavaCro'16

Ivan Senji
Marko Ciglar

IDENTALIA
CONSULTING

Social network visualization

JavaCro'16

Introduction

Why?

How?

Examples

Introduction

JavaCro'16

About us

What we do

Why social networks

Why

Explore our data and gain insights

Experiment with interesting technologies

Starting state

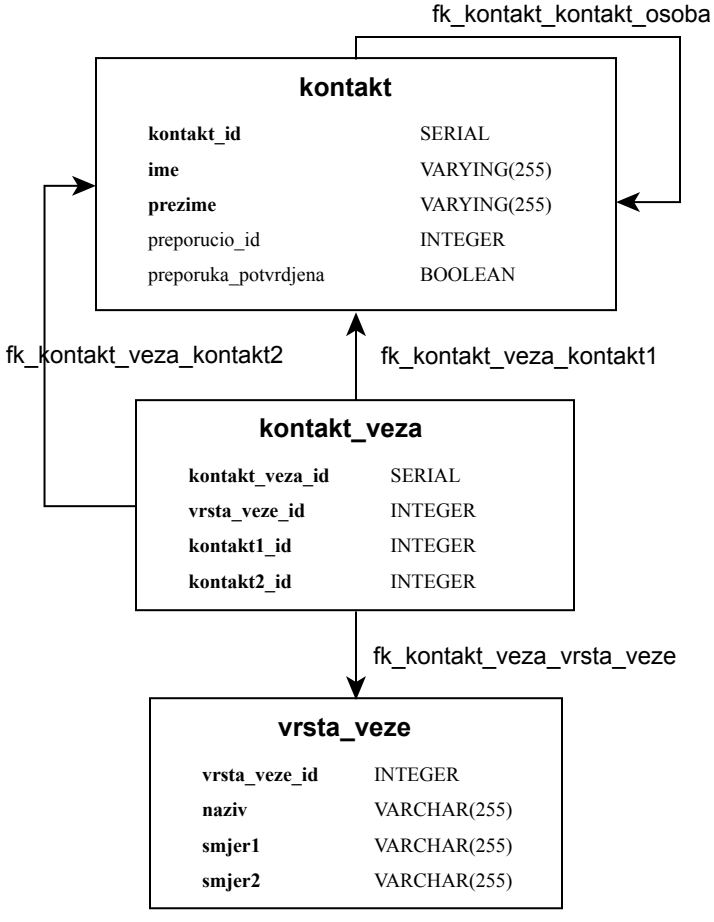
JavaCro'16

Relational database (PostgreSQL)

CRM application

- Java, SmartGWT, Spring
- data displayed in grids and forms

Starting database model



Problems with relational model

JavaCro'16

Connections between contacts spread over tables and foreign keys:

- Table *kontakt* (*preporucio_id*, *preporuka_potvrdjena*)
- Table *kontakt_veza* (*kontakt1_id*, *kontakt2_id*)

Problems with relational model

JavaCro'16

Hard to visualize:

- all connections in one place
- different levels of connections
- groups of connected contacts

Hard to analyze:

- hard to find useful or interesting patterns in data

Graph databases

JavaCro'16

Nodes (entities) and relationships

Nodes and relationships can have properties

Relationships connect individual nodes

Different kinds of relationships all stored as edges between nodes

Graph traversals involve no expensive index lookups

Easy to find patterns in a way that nodes are connected

Solution

JavaCro'16

1. Choose a graph database
2. Import data from a relational database
3. Integrate with existing CRM and visualize data
4. Implement interesting queries in a graph database

Neo4j

JavaCro'16

"Neo4j is a highly scalable native graph database that leverages data relationships as first-class entities, helping enterprises build intelligent applications to meet today's evolving data challenges." -- neo4j.com

- native graph database
- active community
- highly performant, scalability
- easy to learn and get started
- *cypher* - powerful graph query language
- easy and fast data loading

Cypher

JavaCro'16

> *Cypher is a declarative graph query language that allows for expressive and efficient querying and updating of the graph store.*

Designed for easy reading

Patterns of nodes and relationships are represented using ASCII-Art

```
(kontakt1)-->(kontakt2)

(kontakt1)-[:PREPORUCIO]->(kontakt2)-[:VEZA]->(kontakt3)
```

Cypher

JavaCro'16

Query examples:

```
// create and return a node with label Person
CREATE (p:Person {name:"Ivan"})
RETURN p;

// create a relationship with label KNOWS
MATCH (p:Person {name:"Ivan"})
CREATE (p)-[r:KNOWS]->(p2:Person {name:"Marko"})
RETURN p,r,p2;

// count nodes with label Person
MATCH (p:Person) RETURN COUNT(p);

// query nodes by pattern
MATCH (p1)-[:KNOWS]->(p2: Person {name:"Marko"})
RETURN p1;
```

CRM - export data

JavaCro'16

1. Export data from PostgreSQL

```
COPY (
    SELECT k.kontakt_id
    , k.ime
    , k.prezime
    , k.spol
    , k.preporucio_id
    , k.preporuka_potvrdjena
    , (SELECT SUM(iznos) FROM placanje WHERE kontakt_id = k.kontakt_id)
        AS placanja_iznos
    FROM kontakt k
) TO '/tmp/kontakt.csv'
    WITH (FORMAT CSV, HEADER, DELIMITER E',', QUOTE E'"', FORCE_QUOTE *);

COPY (SELECT * FROM kontakt_veza JOIN vrsta_veze USING (vrsta_veze_id))
TO '/tmp/kontakt_veza.csv'
WITH (FORMAT CSV, HEADER, DELIMITER E',', QUOTE E'"', FORCE_QUOTE *);
```

Neo4j - import data

JavaCro'16

2. Create/merge contact nodes

```
LOAD CSV WITH HEADERS FROM "file:///tmp/kontakt.csv"
  AS row FIELDTERMINATOR ","
MERGE (k:Kontakt {kontakt_id: toInt(row.kontakt_id)})
SET
  k.ime = toString(row.ime)
, k.prezime = toString(row.prezime)
, k.spol = toString(row.spol)
, k.preporucio_id = toInt(row.preporucio_id)
, k.preporuka_potvrdjena
    = CASE row.preporuka_potvrdjena
      WHEN "t" THEN true ELSE false
      END
, k.placanja_iznos = toFloat(row.placanja_iznos)
;
```

```
CREATE INDEX ON :Kontakt(kontakt_id);
CREATE INDEX ON :Kontakt(preporucio_id);
CREATE INDEX ON :Kontakt(kontakt_osoba_id);
```

Neo4j - import data

JavaCro'16

3. Add relationships from *kontakt* table

```
//:PREPORUCIO
MATCH (kontakt:Kontakt)
WHERE kontakt.preporucio_id IS NOT NULL
WITH kontakt
MATCH (preporucitelj:Kontakt)
WHERE preporucitelj.kontakt_id = kontakt.preporucio_id
MERGE (preporucitelj)
        -[:PREPORUCIO {
            preporuka_potvrdjena: kontakt.preporuka_potvrdjena
        }]->(kontakt)
;
```

Neo4j - import data

JavaCro'16

4. Add relationships from *kontakt_veze* table

```
//:VEZA
LOAD CSV WITH HEADERS FROM "file:///tmp/kontakt_veza.csv" AS row
  FIELDTERMINATOR ","
MATCH (k1:Kontakt {kontakt_id: toInt(row.kontakt1_id)})
MATCH (k2:Kontakt {kontakt_id: toInt(row.kontakt2_id)})
MERGE (k1)-[:VEZA {
        vrsta_veze_id:toInt(row.vrsta_veze_id),
        naziv:toString(row.naziv),
        smjer1:toString(row.smjer1),
        smjer2:toString(row.smjer2),
        rang:toInt(row.rang)
      }]->(k2)
;
```

Neo4j - embedded vs standalone

JavaCro'16

Embedded (currently implemented)

Pros:

- easier to get started
- java API
- lower latency

Cons:

- sharing physical resources with CRM application
- 46MB of extra dependencies (neo4j, scala)

Neo4j - embedded vs standalone

JavaCro'16

Standalone (planned for production)

Pros:

- CRM and Neo4j have independent resources
- able to fine tune resources (memory, garbage collection)
- installation on a separate machine
- web admin
- access from multiple clients

Cons:

- need to use REST API

Visualization: vis.js

JavaCro'16

> *"A dynamic, browser based visualization library. The library is designed to be easy to use, to handle large amounts of dynamic data, and to enable manipulation of and interaction with the data." -- visjs.org*

- open source
- many components: DataSet, Timeline, **Network**, Graph2d and Graph3d
- runs on Chrome, Firefox, Opera, Safari, IE9+ and most mobile browsers

Integration

JavaCro'16

Server-side

Embedded Neo4j database

Neo4jServlet

- handles client **GET** requests
- executes *Cypher* queries
- transforms data into *vis.js* compatible format

Integration

Neo4jServlet

Support for different Cypher query results:

- nodes
- relationships
- sequences of relationships
- nodes wrapped in maps

Integration

JavaCro'16

SmartGWT

HTMLPane

- displays a HTML page (vis.html) that uses vis.js to visualize data

DynamicForm

- used to define the query to visualize
- passes parameters to *HTMLPane* component
- *HTMLPane* fetches *vis.html* with parameters received from *DynamicForm*

Integration

JavaCro'16

vis.html

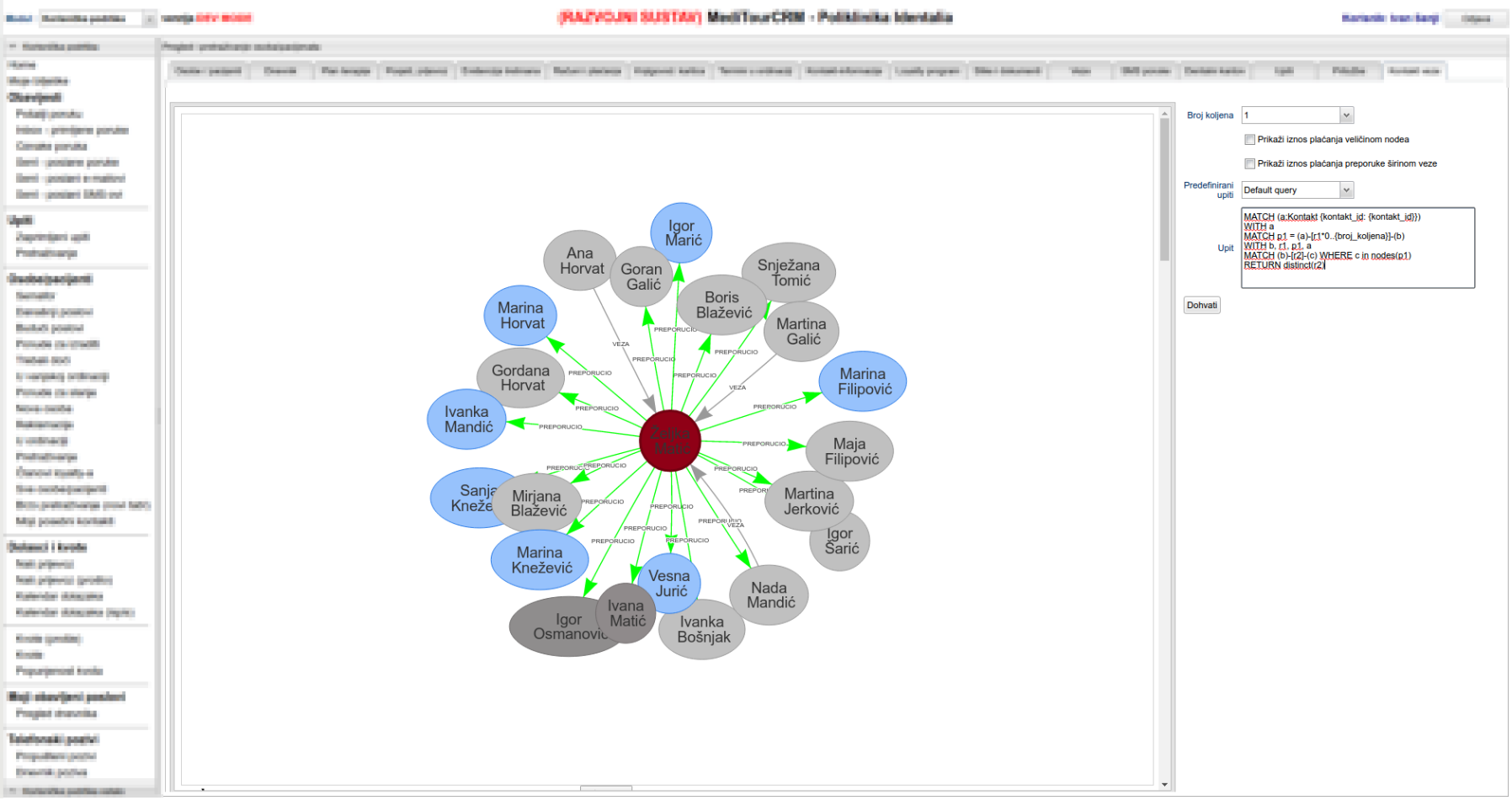
jQuery

- page initialization
- event handling
- fetch data from *Neo4jServlet*

vis.js

- visualize data received from server

CRM GUI


Igor Marić
Ana Horvat
Goran Galić
Snježana Tomić
Boris Blažević
Martina Galić
Marina Horvat
Gordana Horvat
Marina Filipović
Ivanka Mandić
Željko Matić
Maja Filipović
Martina Jerković
Igor Šarić
Mirjana Blažević
Marina Knežević
Vesna Jurić
Nada Mandić
Ivana Matić
Ivanka Bošnjak
PREPORUCIO
VEZA
Broj koljena
1
Prikaži iznos plaćanja veličinom nodea
Prikaži iznos plaćanja preporuke širinom veze
Predefinirani upiti
Default query
Upit
MATCH (a:Kontakt {kontakt_id: {kontakt_id}})
WITH a
MATCH p1 = (a)-[r1*0..{broj_koljena}]-(b)
WITH b, r1, p1, a
MATCH (b)-[r2]-(c) WHERE c in nodes(p1)
RETURN distinct(r2)
Dohvati

Example

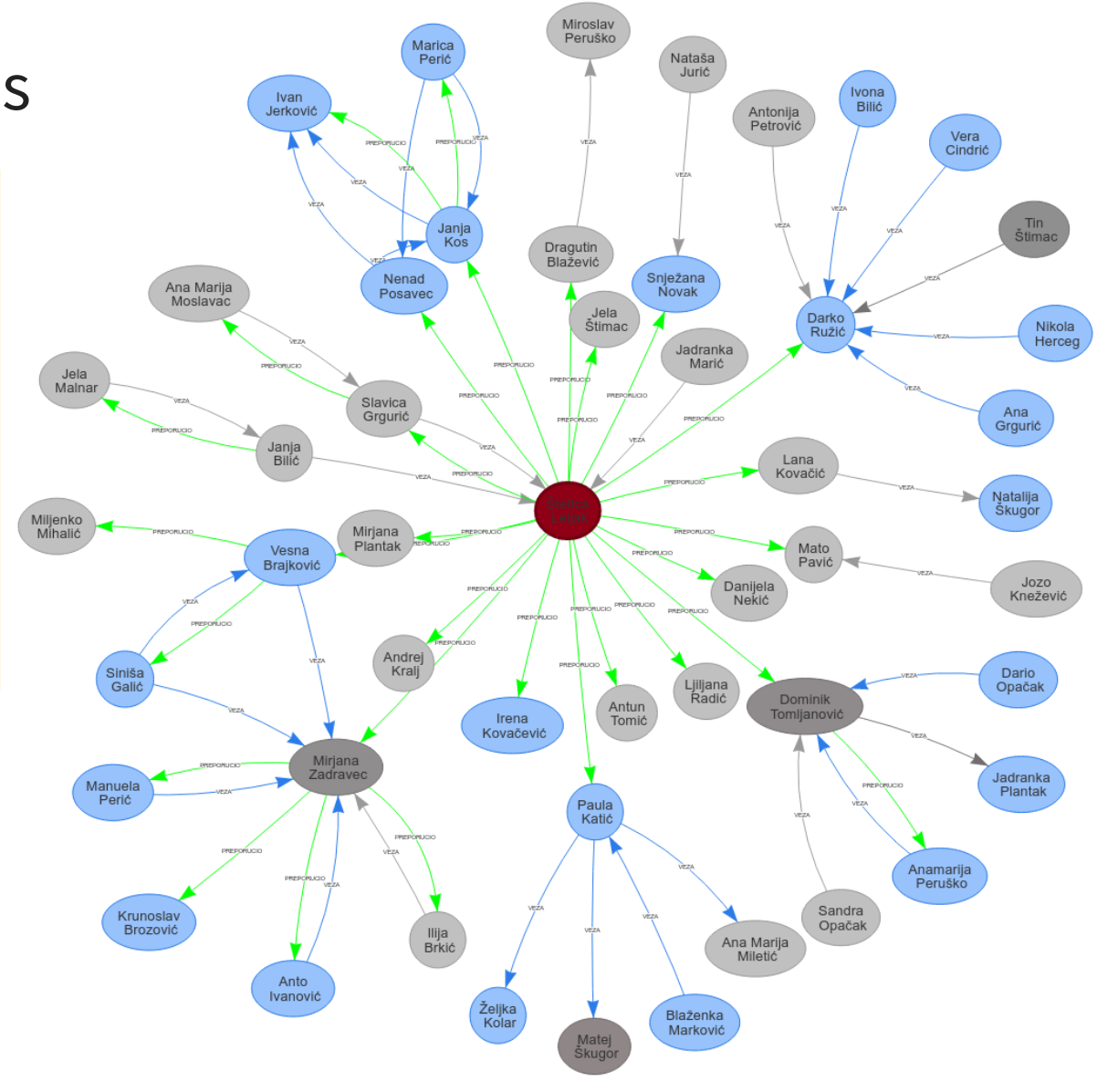
Contacts with connections

```
MATCH (a:Kontakt
  {kontakt_id: {kontakt_id}})
WITH a
MATCH p1 = (a)-[r1*0..1]-(b)
WITH b, r1, p1, a
MATCH (b)-[r2]-(c)
WHERE c in nodes(p1)
RETURN distinct(r2)
WITH params {
  kontakt_id=63976
}
```

Example

Contacts with connections

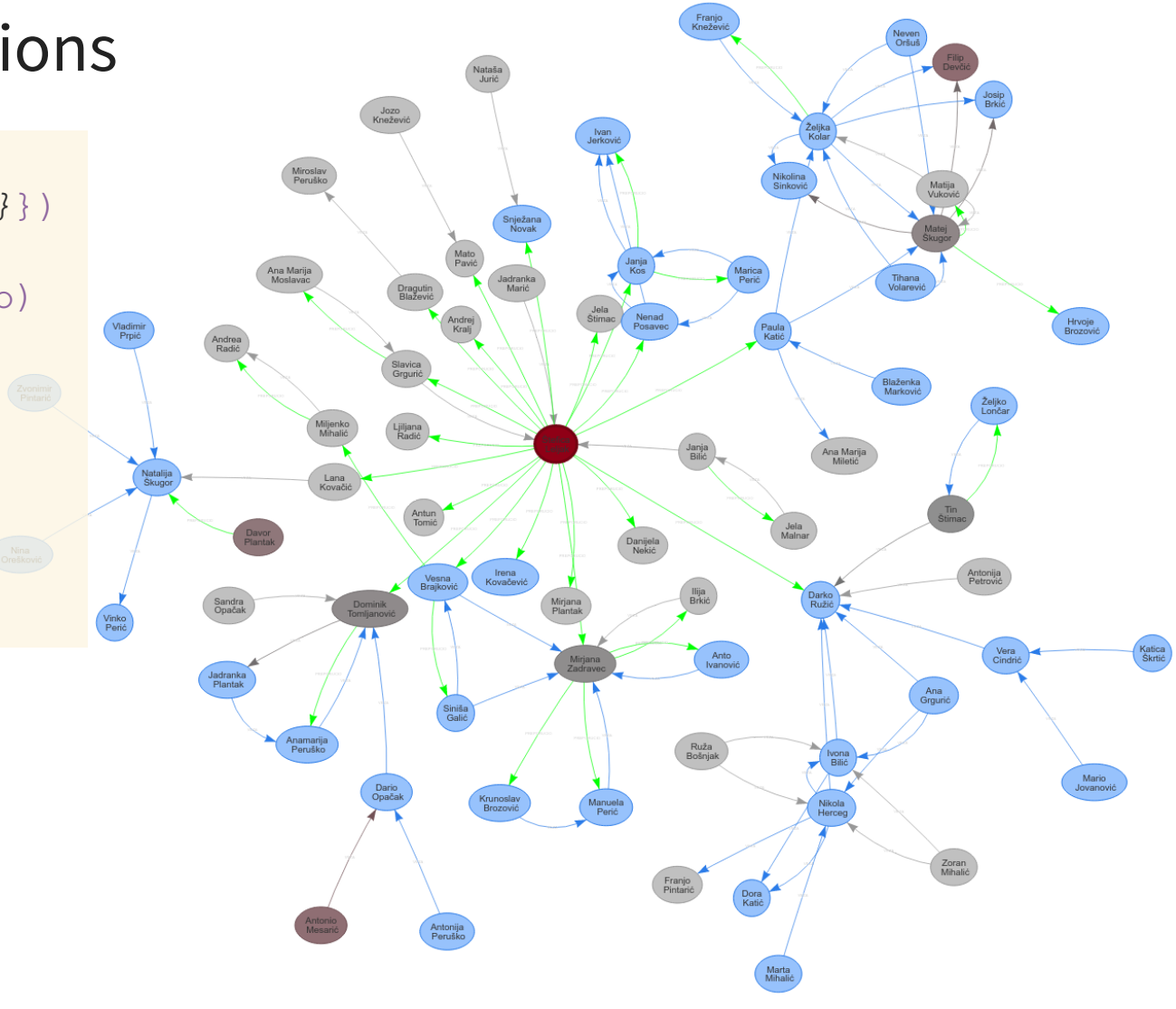
```
MATCH (a:Kontakt
  {kontakt_id: {kontakt_id}})
WITH a
MATCH p1 = (a)-[r1*0..2]-(b)
WITH b, r1, p1, a
MATCH (b)-[r2]-(c)
WHERE c in nodes(p1)
RETURN distinct(r2)
with params {
  kontakt_id=63976
}
```



Example

Contacts with connections

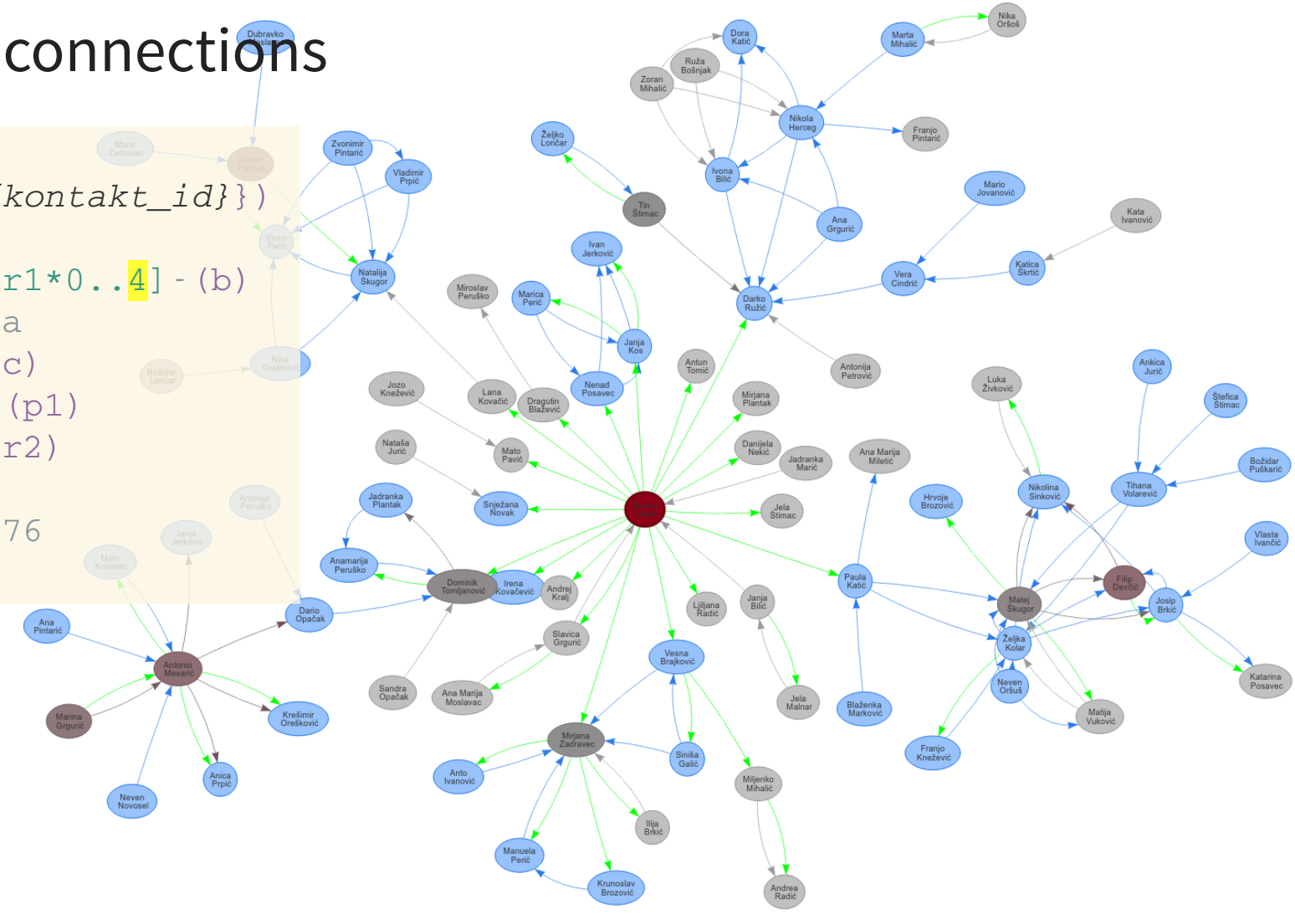
```
MATCH (a:Kontakt
  {kontakt_id: {kontakt_id}})
WITH a
MATCH p1 = (a)-[r1*0..3]-(b)
WITH b, r1, p1, a
MATCH (b)-[r2]-(c)
WHERE c in nodes(p1)
RETURN distinct(r2)
with params {
  kontakt_id=63976
}
```



Example

Contacts with connections

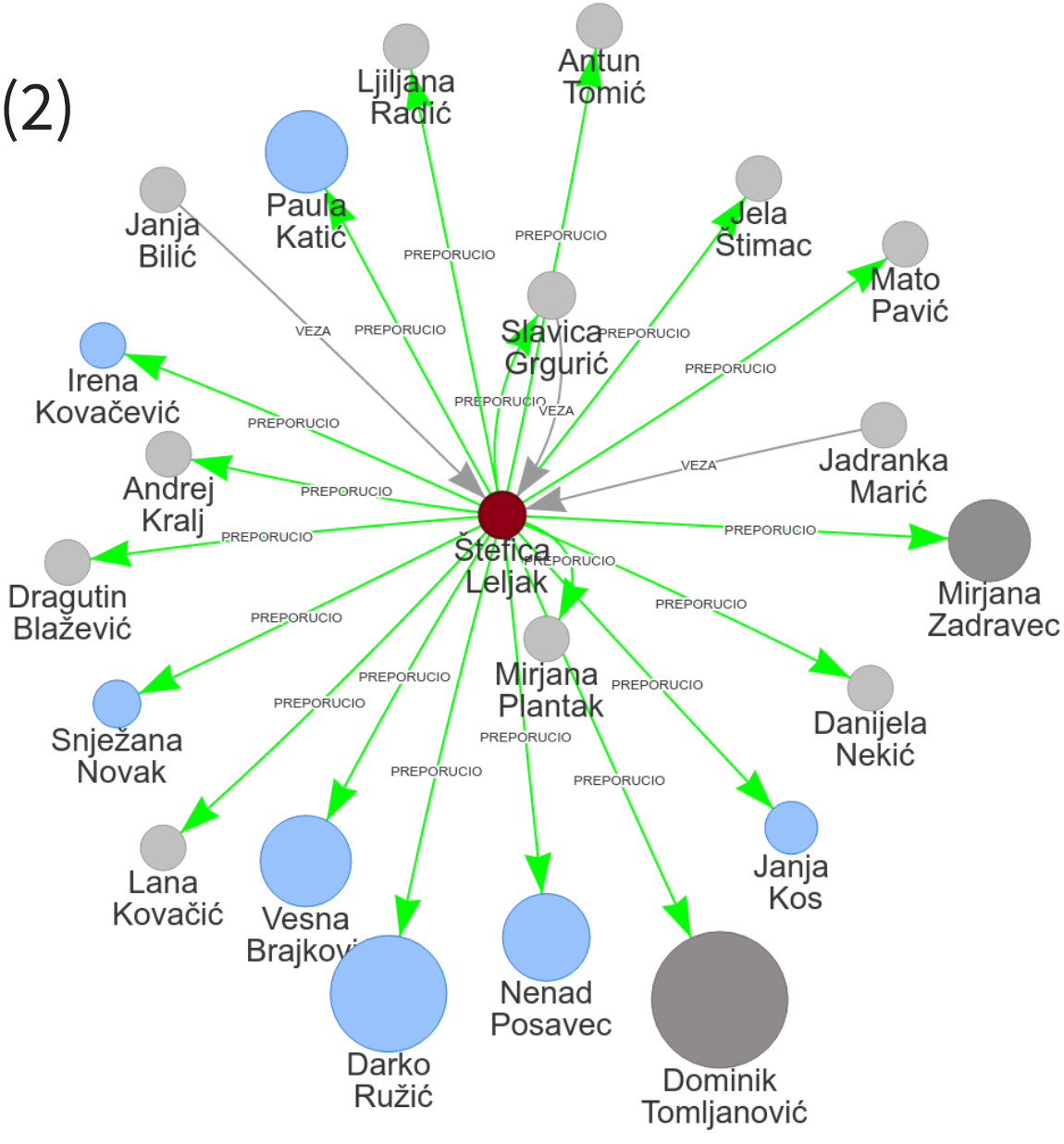
```
MATCH (a:Kontakt
  {kontakt_id: {kontakt_id}})
WITH a
MATCH p1 = (a)-[r1*0..4]-(b)
WITH b, r1, p1, a
MATCH (b)-[r2]-(c)
WHERE c in nodes(p1)
RETURN distinct(r2)
with params {
  kontakt_id=63976
}
```



Example

Contacts with connections (2)

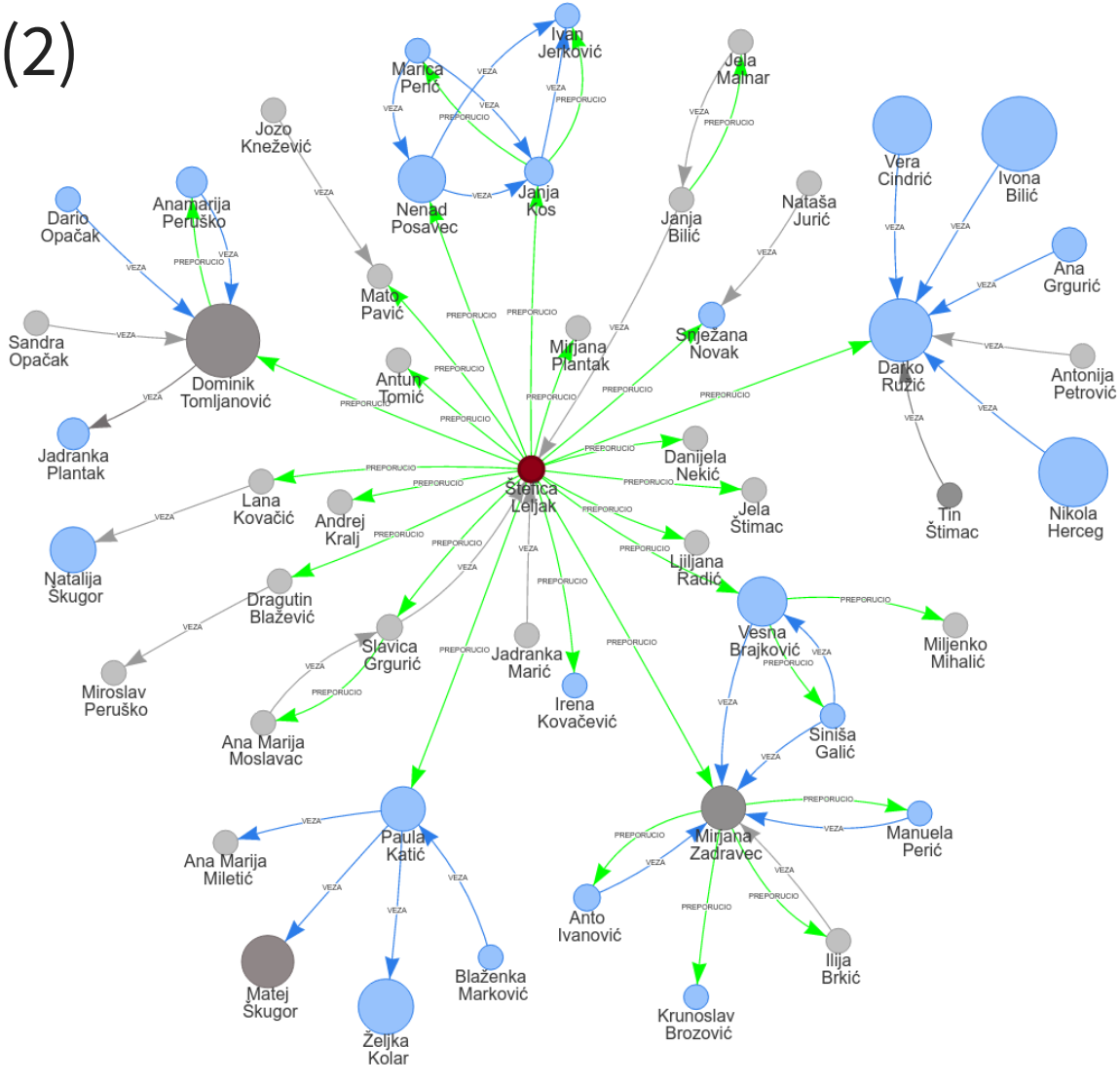
```
MATCH (a:Kontakt
  {kontakt_id: {kontakt_id}})
WITH a
MATCH p1 = (a)-[r1*0..1]-(b)
WITH b, r1, p1, a
MATCH (b)-[r2]-(c)
WHERE c in nodes(p1)
RETURN distinct(r2)
with params {
  kontakt_id=63976
}
```



Example

Contacts with connections (2)

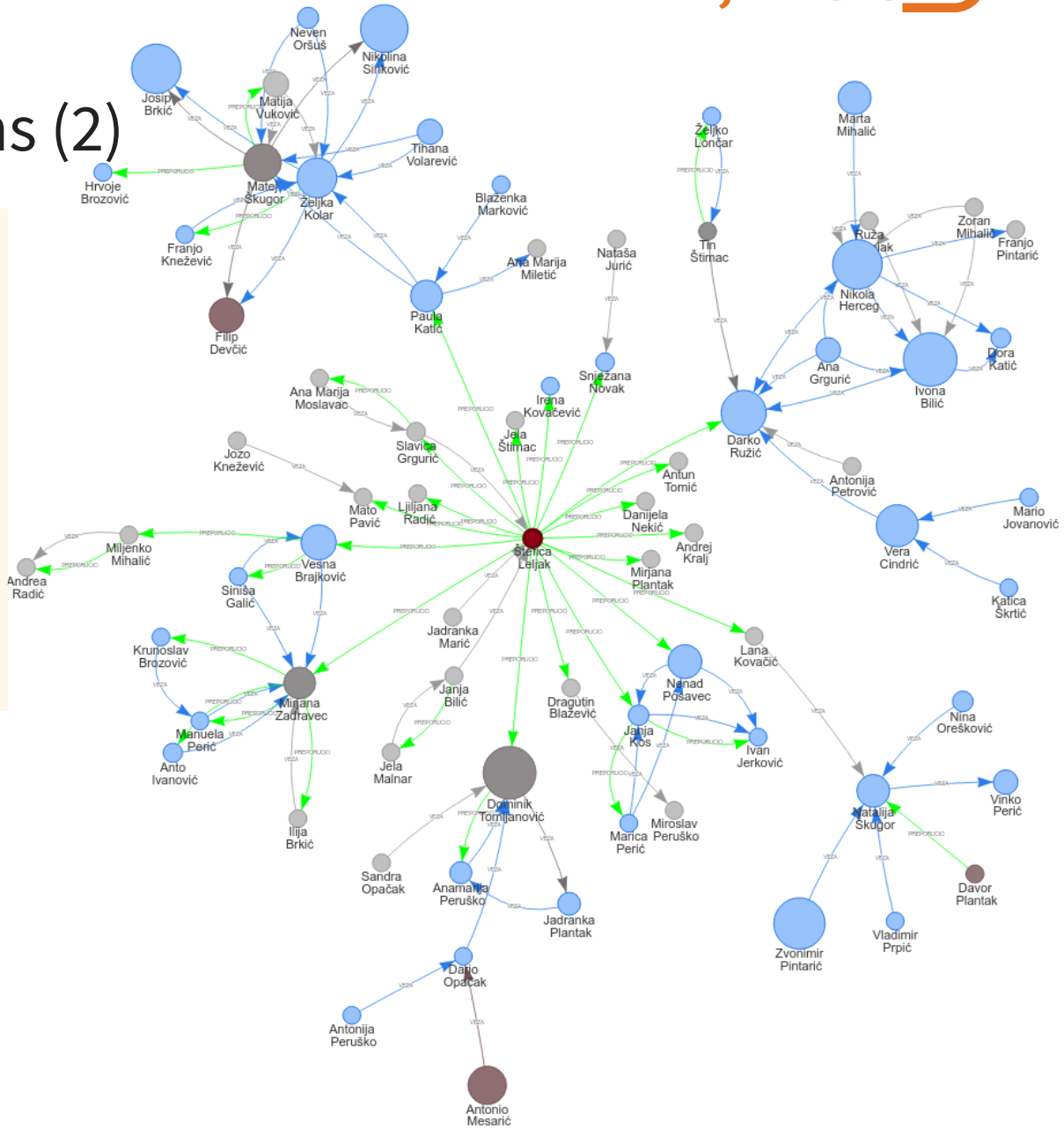
```
MATCH (a:Kontakt
  {kontakt_id: {kontakt_id}})
WITH a
MATCH p1 = (a)-[r1*0..2]-(b)
WITH b, r1, p1, a
MATCH (b)-[r2]-(c)
WHERE c in nodes(p1)
RETURN distinct(r2)
with params {
  kontakt_id=63976
}
```



Example

Contacts with connections (2)

```
MATCH (a:Kontakt
  {kontakt_id: {kontakt_id}})
WITH a
MATCH p1 = (a)-[r1*0..3]-(b)
WITH b, r1, p1, a
MATCH (b)-[r2]-(c)
WHERE c in nodes(p1)
RETURN distinct(r2)
with params {
  kontakt_id=63976
}
```

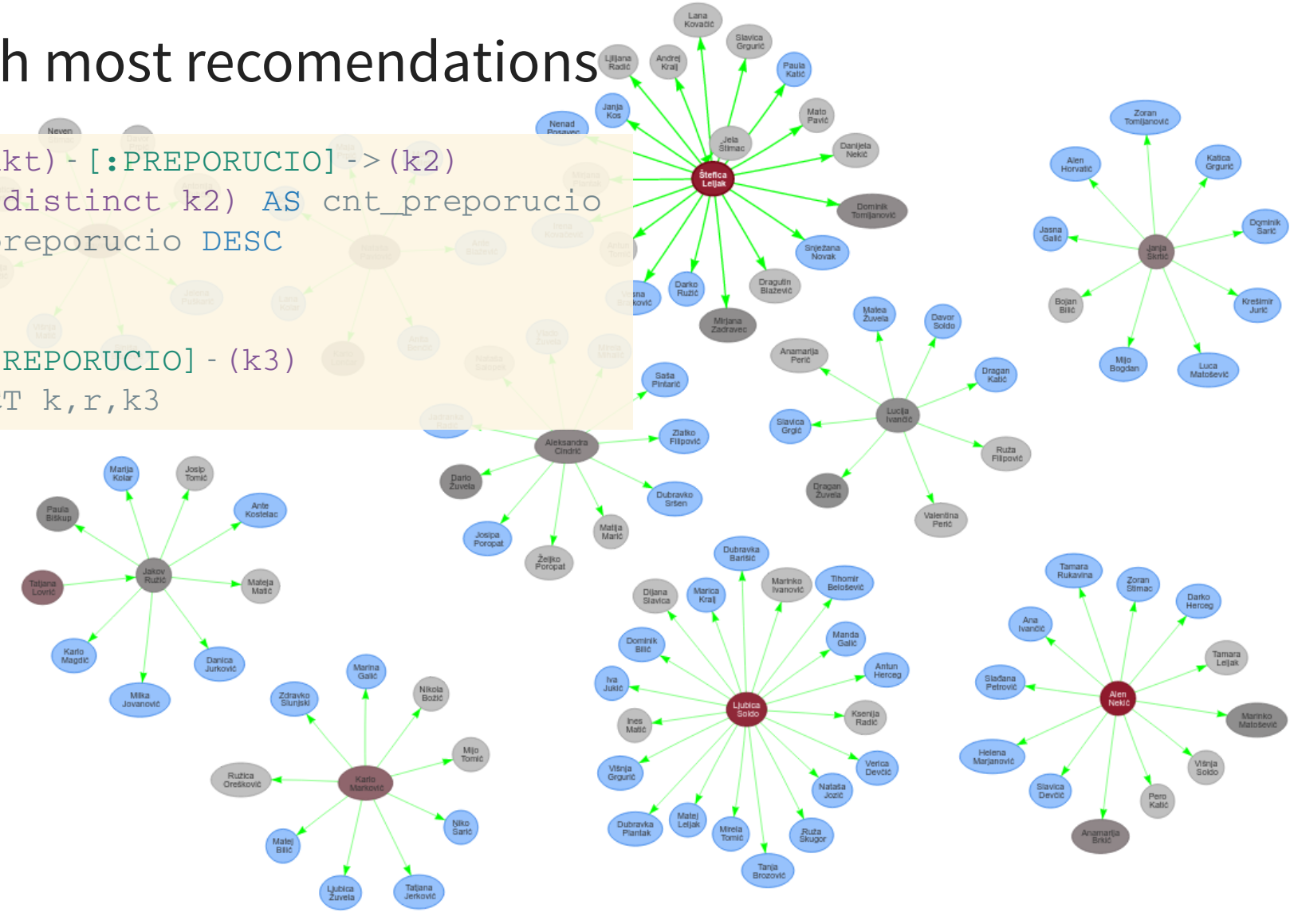


Example

JavaCro'16

Contacts with most recomendations

```
MATCH (k:Kontakt)-[:PREPORUCIO]->(k2)
WITH k, count(distinct k2) AS cnt_preporucio
ORDER BY cnt_preporucio DESC
SKIP 0
LIMIT 10
MATCH (k)-[r:PREPORUCIO]-(k3)
RETURN DISTINCT k,r,k3
```

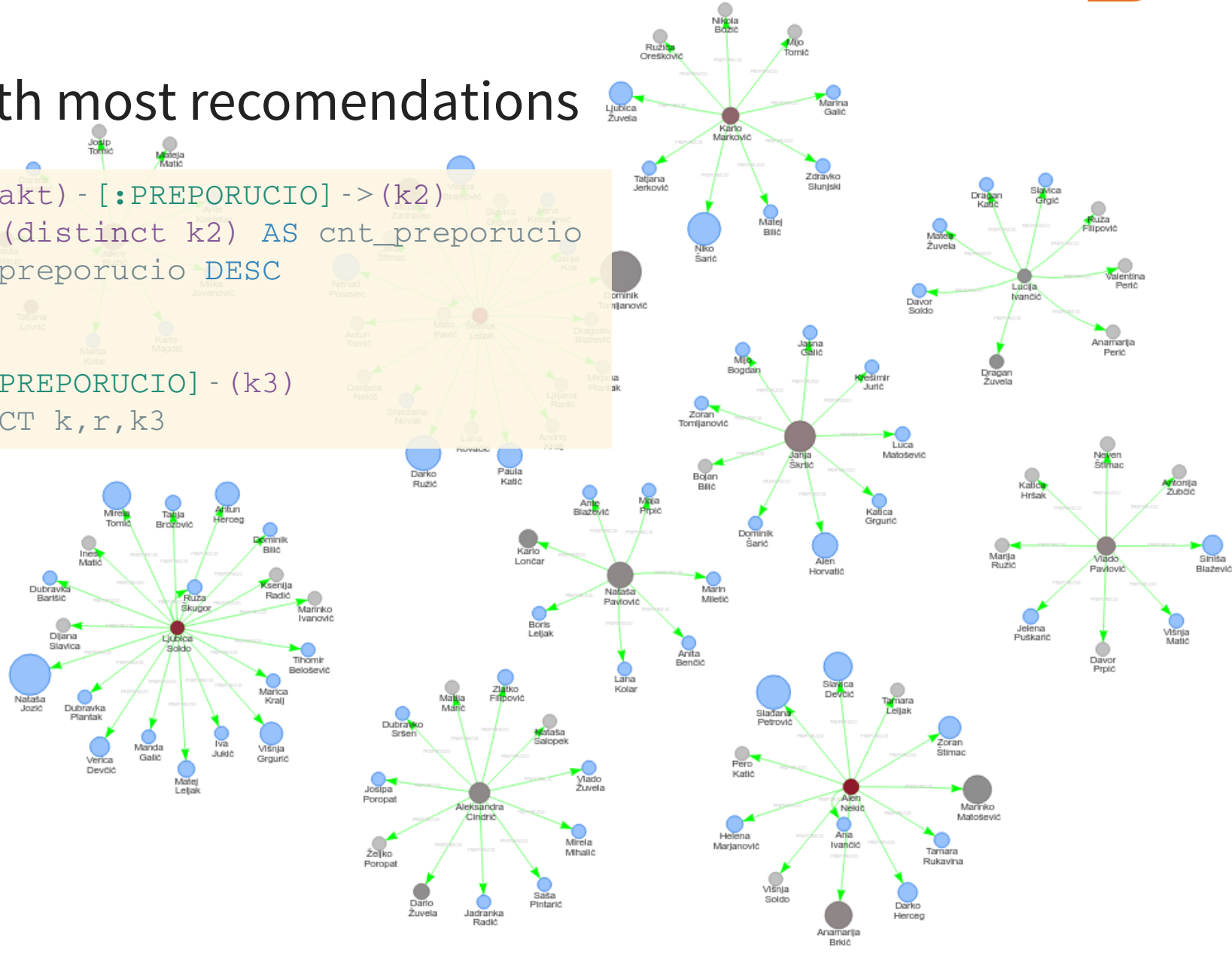


Example

JavaCro'16

Contacts with most recomendations

```
MATCH (k:Kontakt)-[:PREPORUCIO]->(k2)
WITH k, count(distinct k2) AS cnt_preporucio
ORDER BY cnt_preporucio DESC
SKIP 0
LIMIT 10
MATCH (k)-[r:PREPORUCIO]-(k3)
RETURN DISTINCT k,r,k3
```

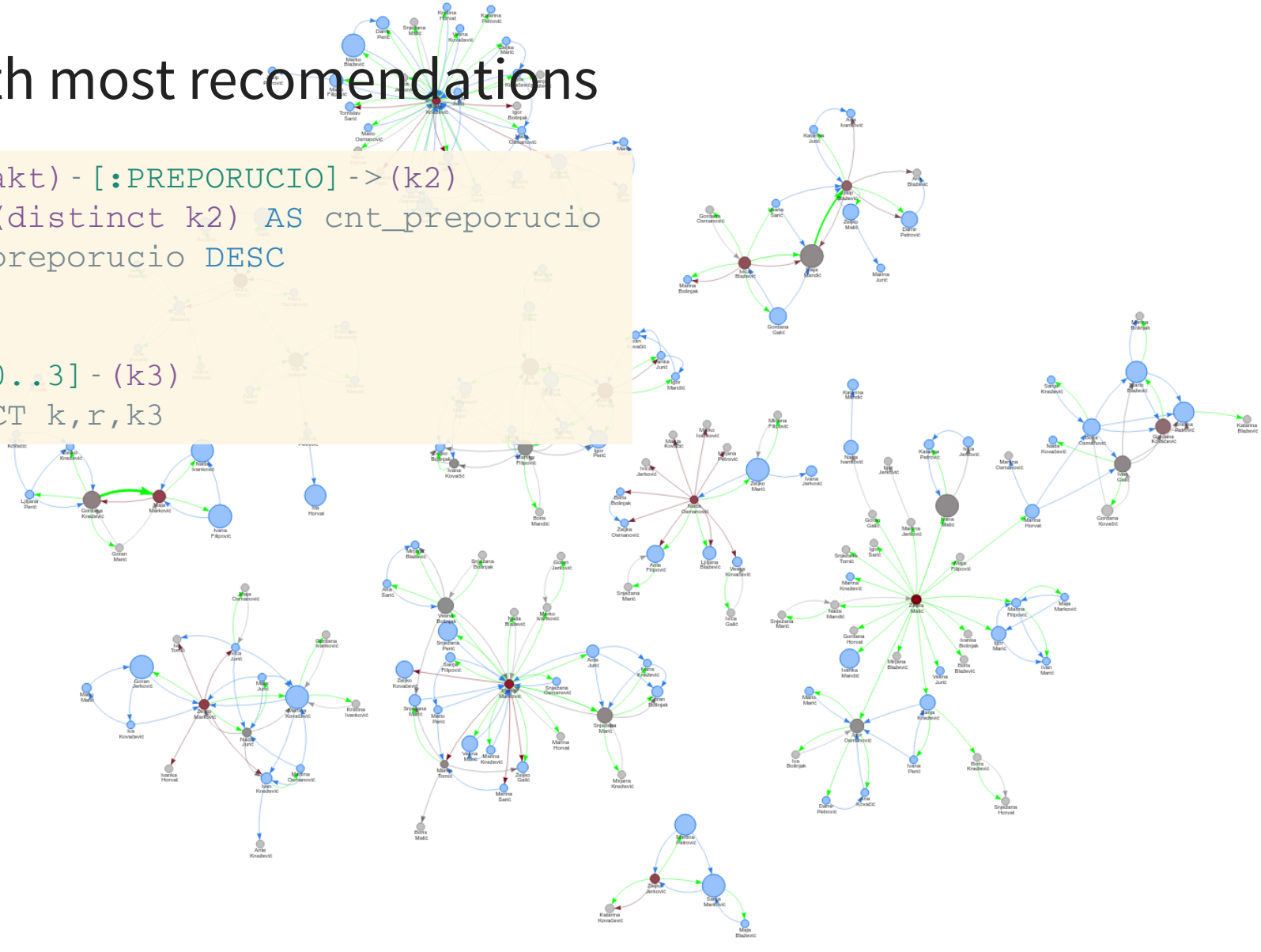


Example

JavaCro'16

Contacts with most recomendations

```
MATCH (k:Kontakt)-[:PREPORUCIO]->(k2)
WITH k, count(distinct k2) AS cnt_preporucio
ORDER BY cnt_preporucio DESC
SKIP 0
LIMIT 10
MATCH (k)-[r*0..3]-(k3)
RETURN DISTINCT k,r,k3
```

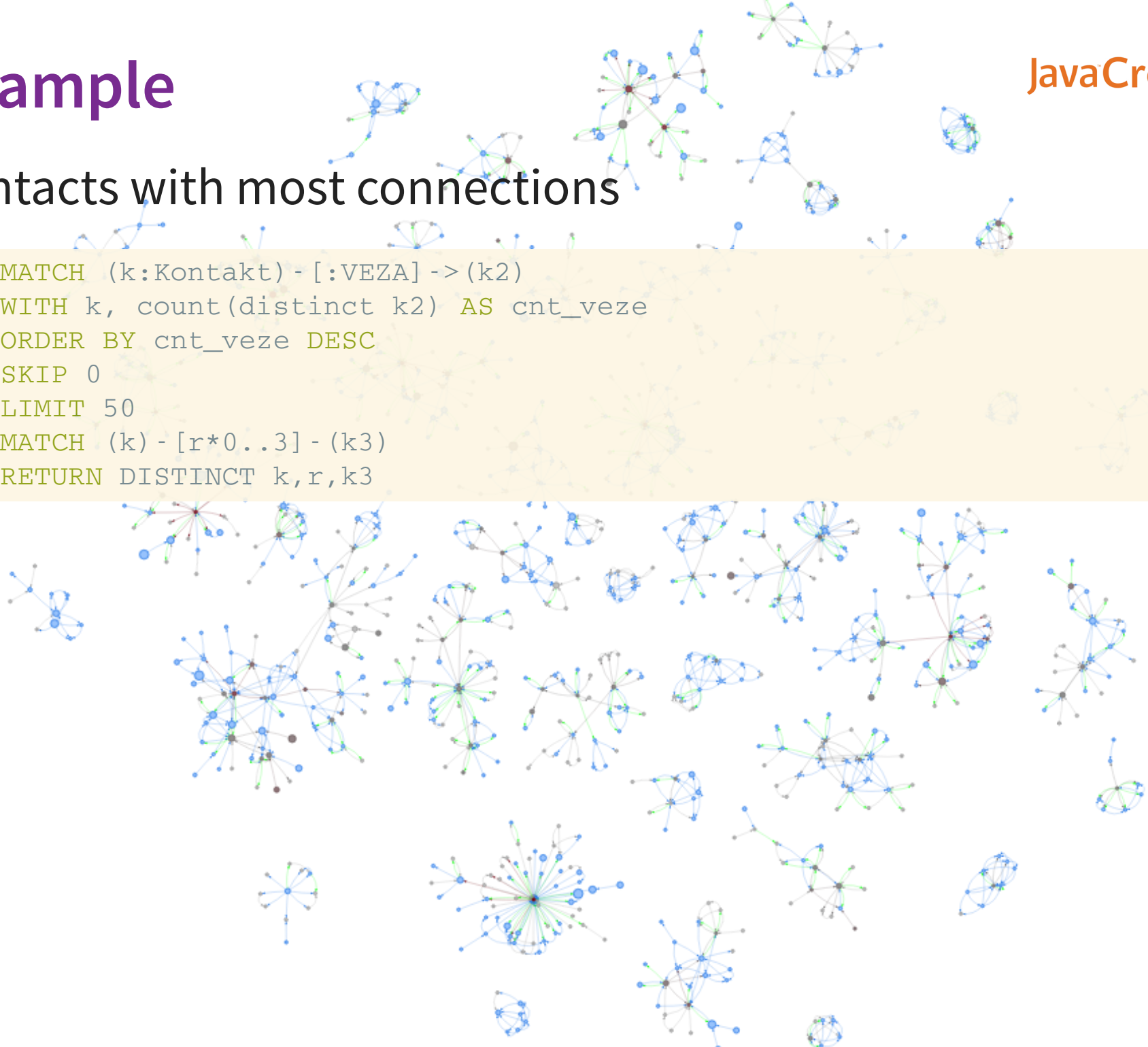


Example

JavaCro'16

Contacts with most connections

```
MATCH (k:Kontakt)-[:VEZA]->(k2)
WITH k, count(distinct k2) AS cnt_veze
ORDER BY cnt_veze DESC
SKIP 0
LIMIT 50
MATCH (k)-[r*0..3]-(k3)
RETURN DISTINCT k,r,k3
```




Questions?