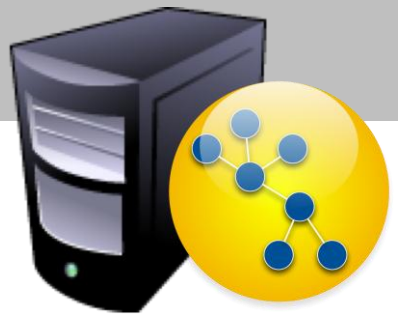
Graph Modeling Do's and Don'ts

@markhneedham
mark.needham@neotechnology.com

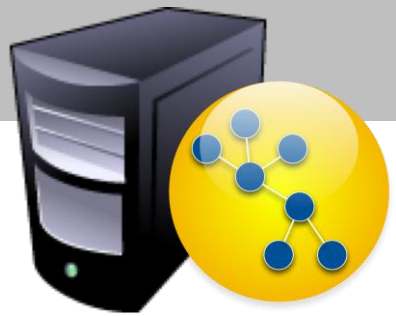
Credit for the slides goes to Ian Robinson
@iansrobinson on twitter

Outline

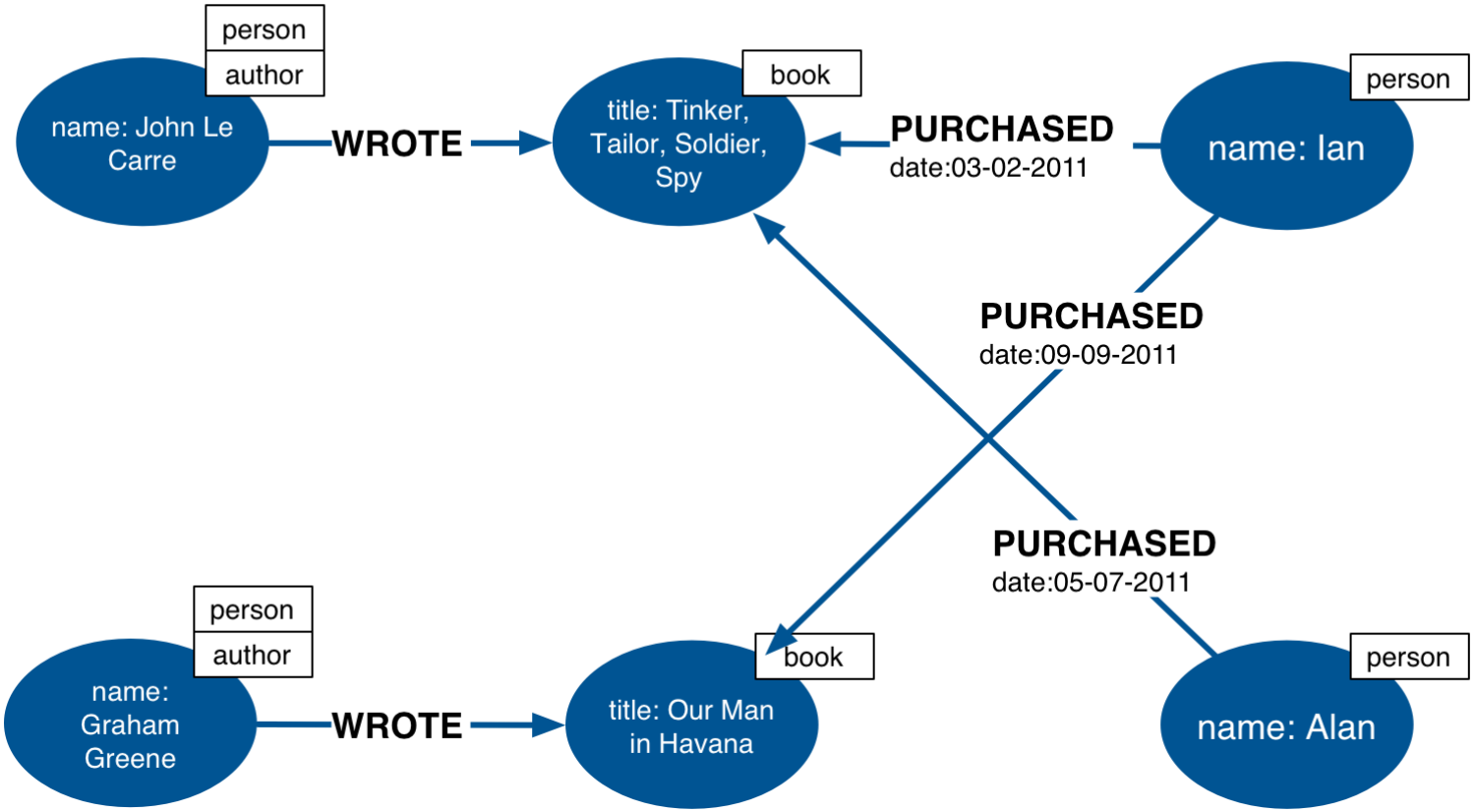


- Property Graph Refresher
- A modeling workflow
- Modeling tips
- Testing your data model

Property Graph Refresher



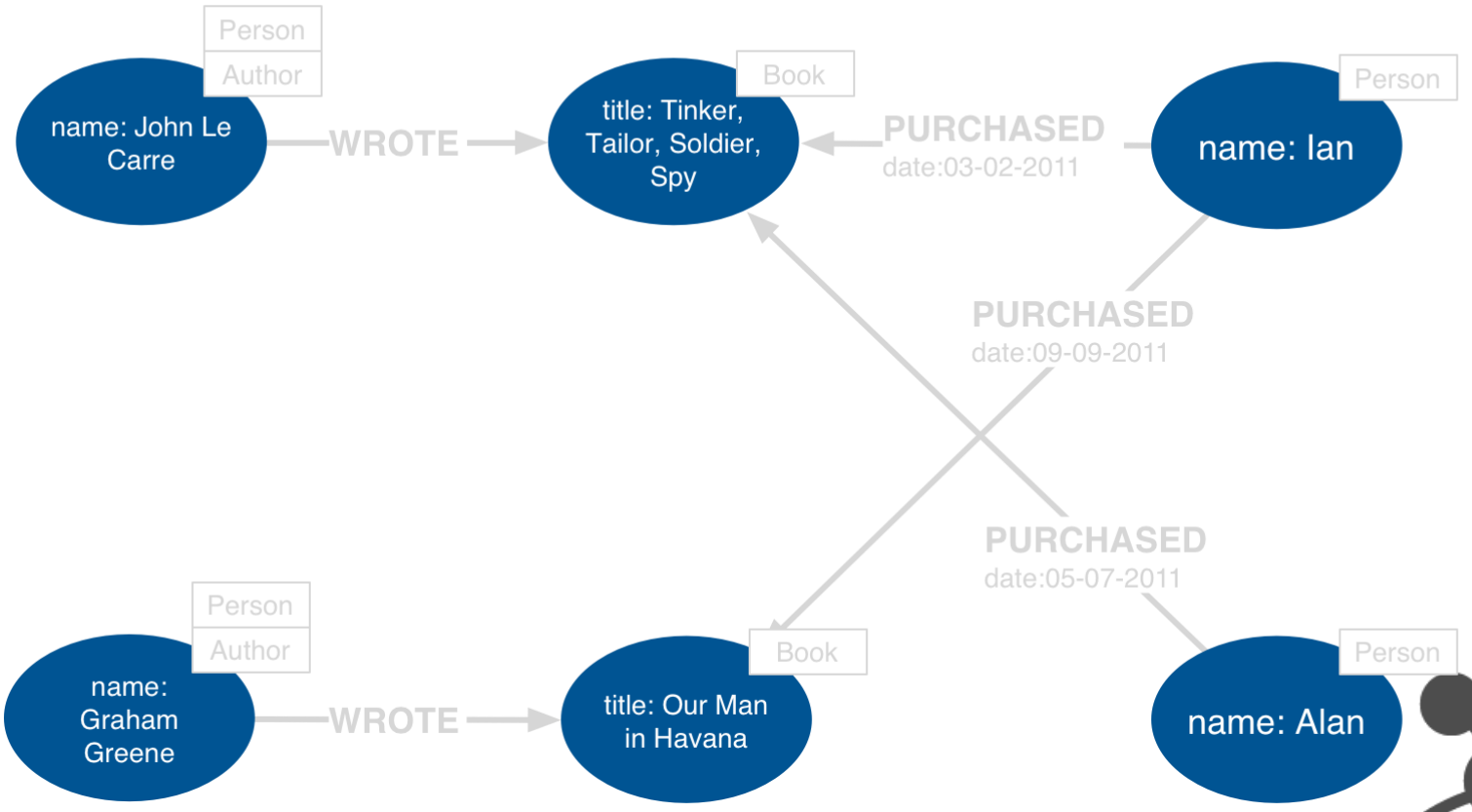
Property Graph Data Model



Four Building Blocks

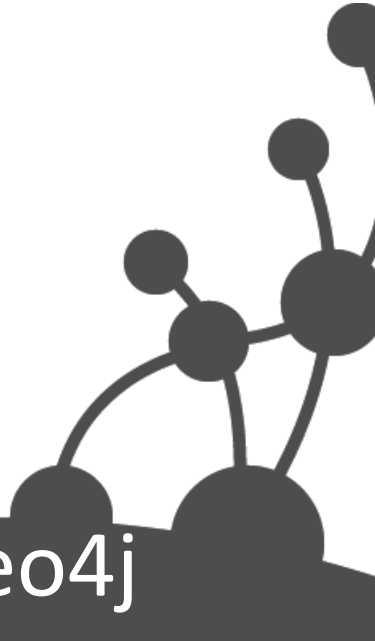
- Nodes
- Relationships
- Properties
- Labels

Nodes


Person
Author
name: John Le Carre
WROTE
Book
title: Tinker, Tailor, Soldier, Spy
PURCHASED
date:03-02-2011
Person
name: Ian
PURCHASED
date:09-09-2011
PURCHASED
date:05-07-2011
Person
Author
name: Graham Greene
WROTE
Book
title: Our Man in Havana
Person
name: Alan

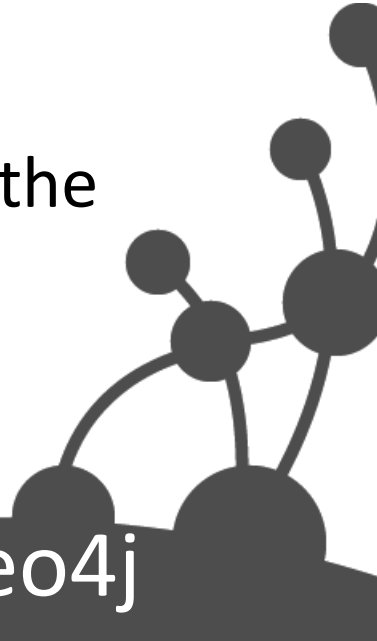
Nodes

- Used to represent *entities* and *complex value types* in your domain
- Can contain properties
  - Used to represent entity *attributes* and/or *metadata* (e.g. timestamps, version)
  - Key-value pairs
    - Java primitives
    - Arrays
    - null is not a valid value
  - Every node can have different properties

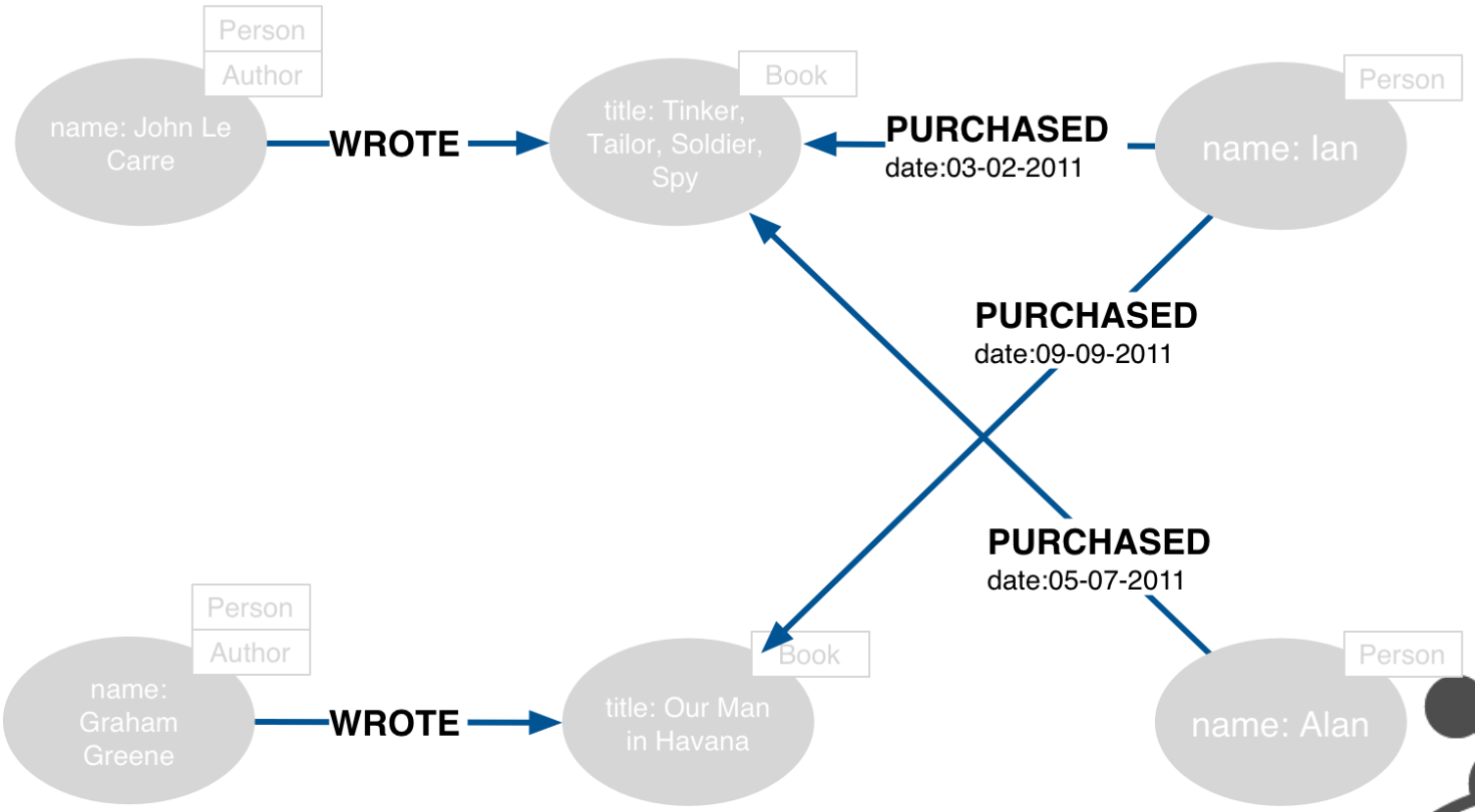


Entities and Value Types

- Entities
  - Have unique conceptual identity
  - Change attribute values, but identity remains the same
- Value types
  - No conceptual identity
  - Can substitute for each other if they have the same value
    - Simple: single value (e.g. colour, category)
    - Complex: multiple attributes (e.g. address)

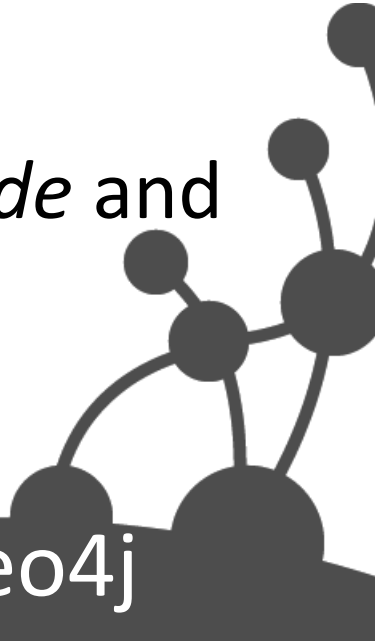


Relationships

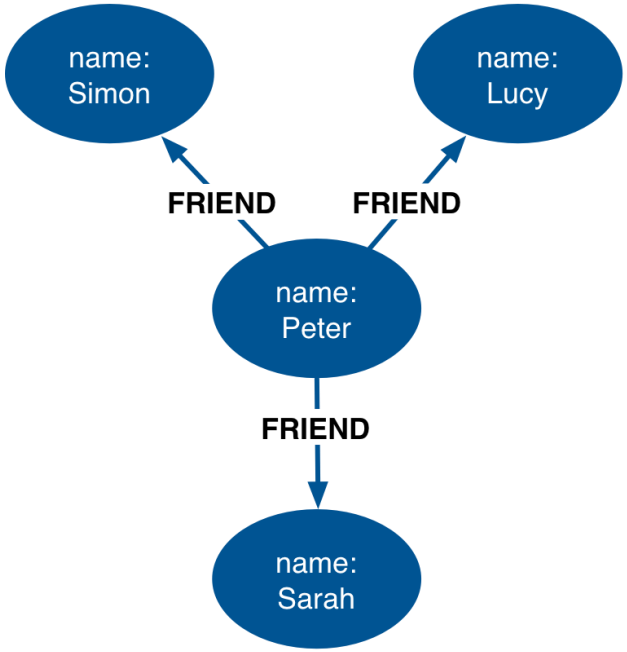


Relationships

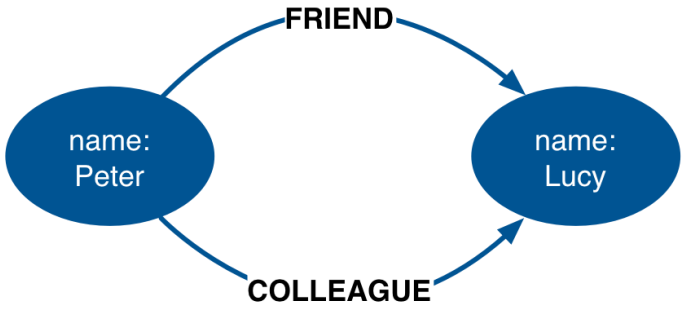
- Every relationship has a *name* and a *direction*
  - Add structure to the graph
  - Provide semantic context for nodes
- Can contain properties
  - Used to represent *quality* or *weight* of relationship, or *metadata*
- Every relationship must have a *start node* and *end node*
  - No dangling relationships



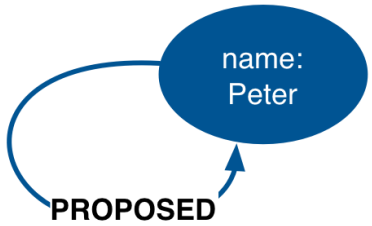
Relationships (continued)



Nodes can have more than one relationship



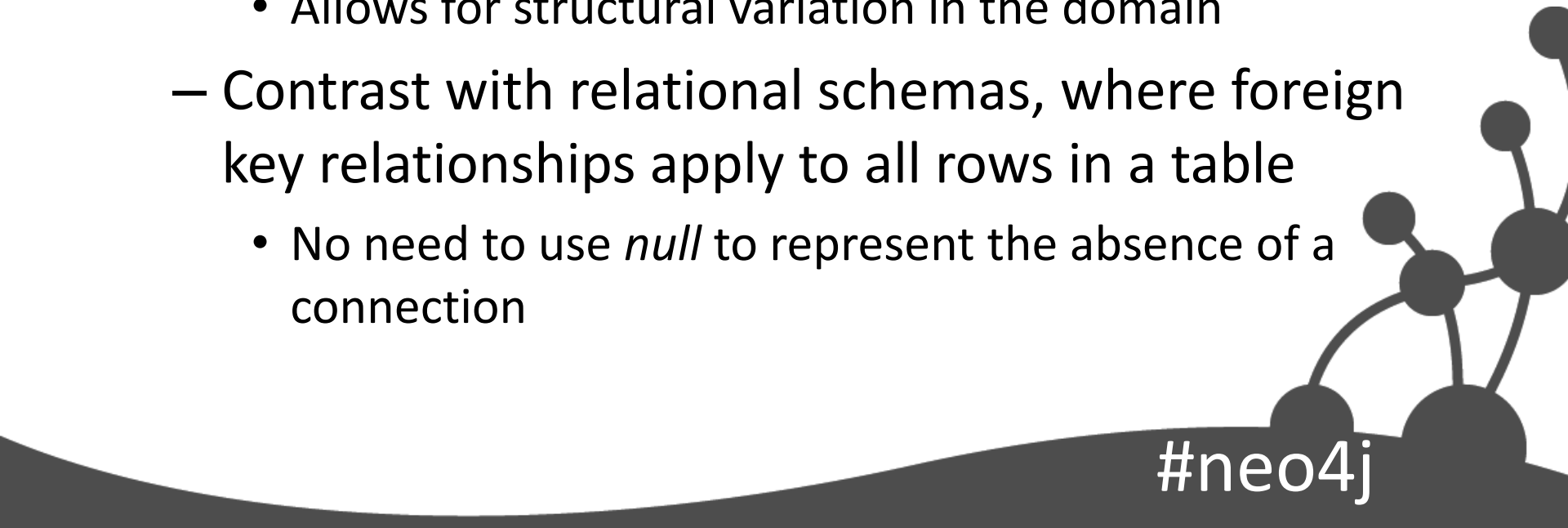
Nodes can be connected by more than one relationship



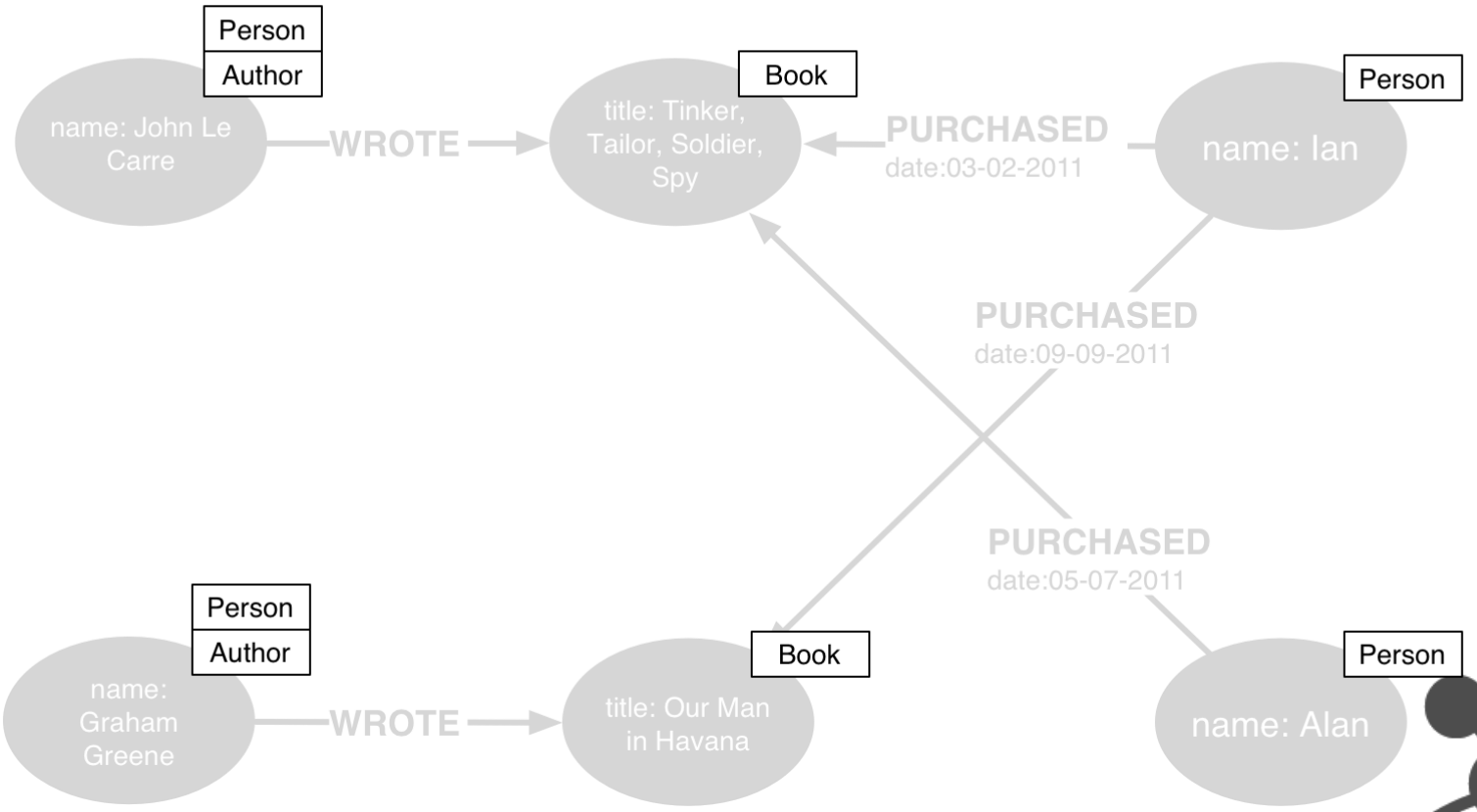
Self relationships are allowed

Variable Structure

- Relationships are defined with regard to node *instances*, not *classes* of nodes
  - Two nodes representing the same kind of “thing” can be connected in very different ways
    - Allows for structural variation in the domain
  - Contrast with relational schemas, where foreign key relationships apply to all rows in a table
    - No need to use *null* to represent the absence of a connection

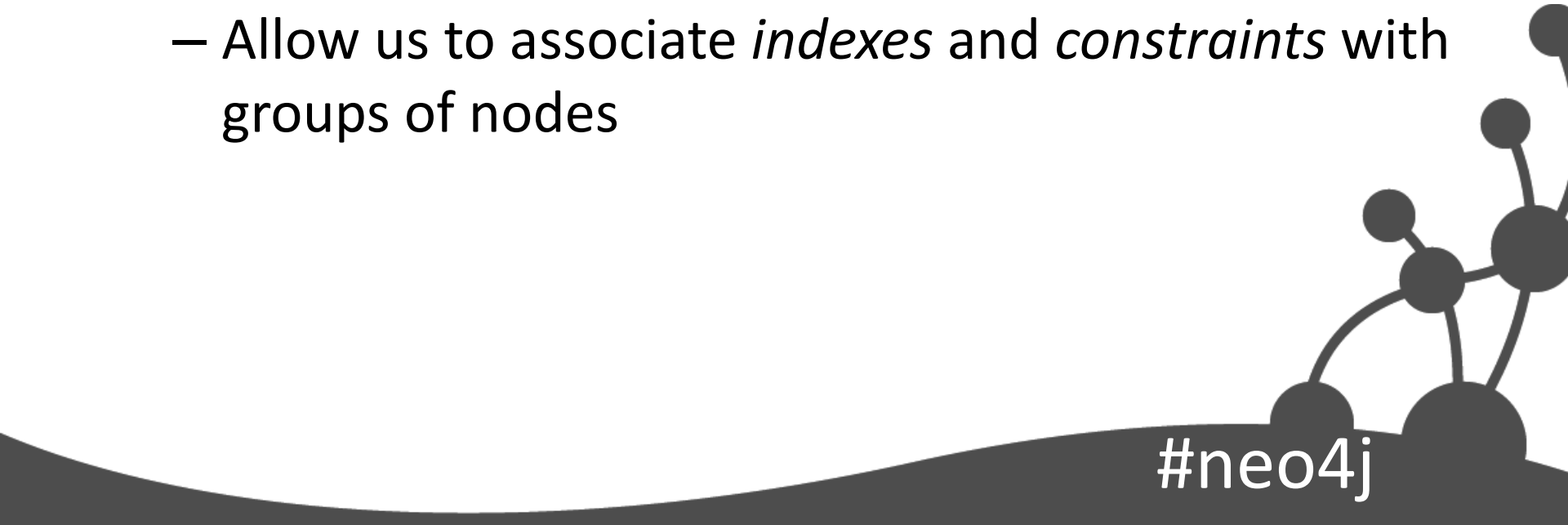


Labels


Person
Author
name: John Le Carre
WROTE
Book
title: Tinker, Tailor, Soldier, Spy
PURCHASED
date:03-02-2011
Person
name: Ian
PURCHASED
date:09-09-2011
PURCHASED
date:05-07-2011
Person
Author
name: Graham Greene
WROTE
Book
title: Our Man in Havana
Person
name: Alan

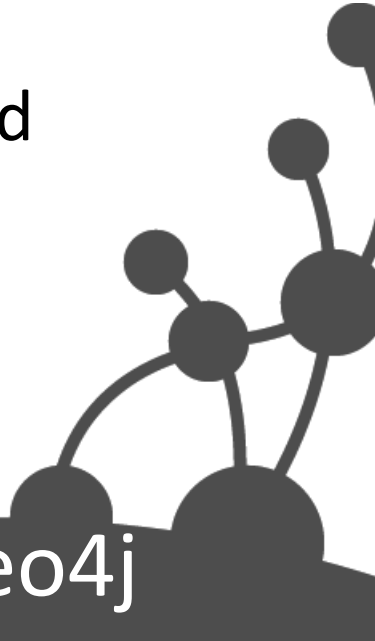
Labels

- Every node can have *zero or more* labels
- Used to represent *roles* (e.g. user, product, company)
  - Group nodes
  - Allow us to associate *indexes* and *constraints* with groups of nodes

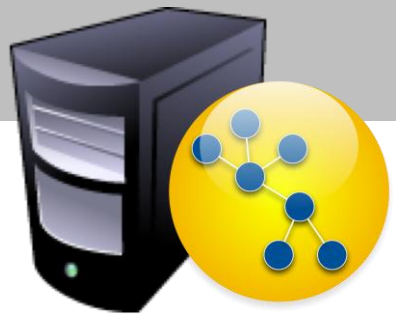


Four Building Blocks

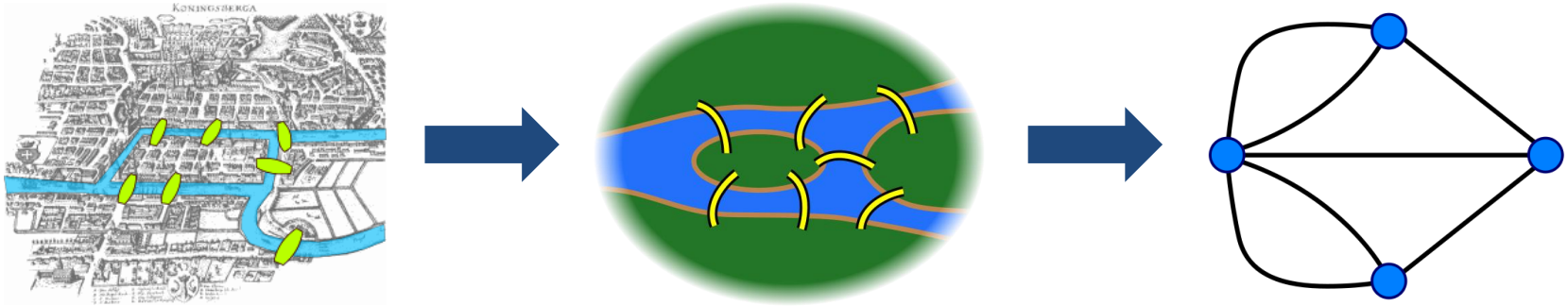
- Nodes
  - Entities
- Relationships
  - Connect entities and structure domain
- Properties
  - Entity attributes, relationship qualities, and metadata
- Labels
  - Group nodes by role



A modeling workflow

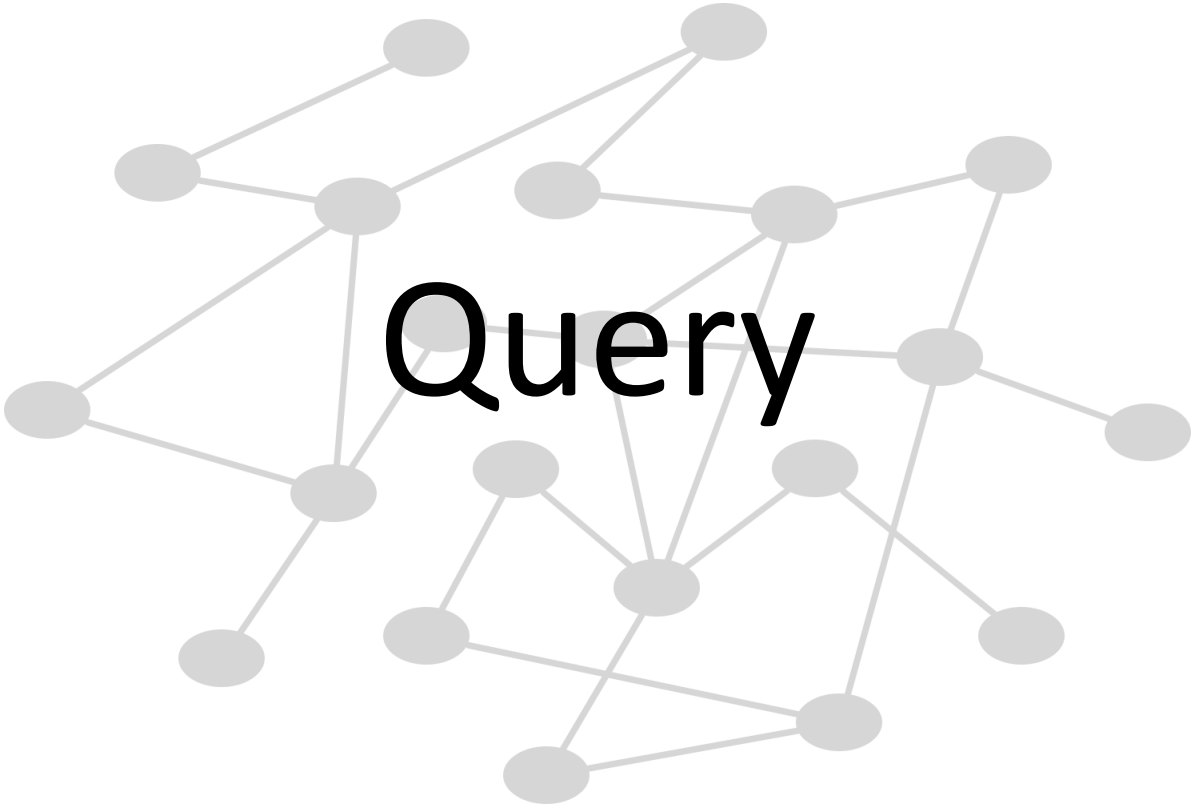


Models



Images: en.wikipedia.org

Design for Queryability

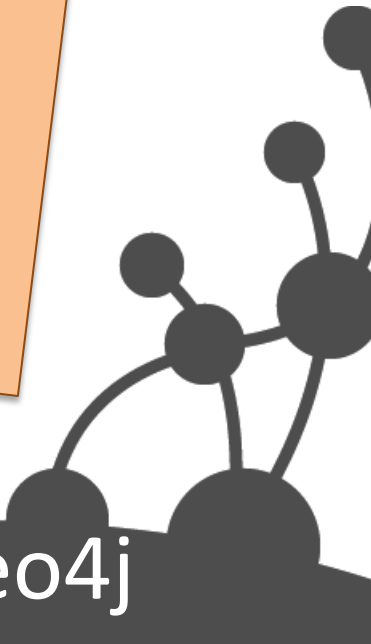


User stories

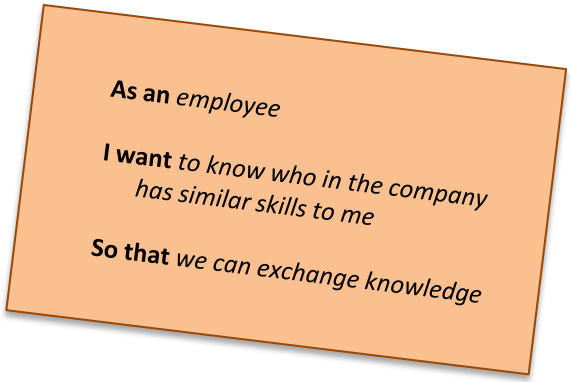
As an *employee*

I want *to know who in the company has similar skills to me*

So that *we can exchange knowledge*



Derive questions



Which people, who work for the same company as me, have similar skills to me?

Identify entities

Which people, who work for the same company as me, have similar skills to me?

person

company

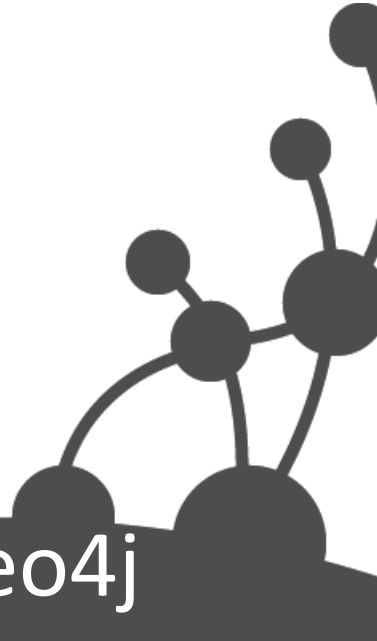
skill

Identify relationships between entities

Which people, who work for the same company as me, have similar skills to me?

person WORKS_FOR company

person HAS_SKILL skill

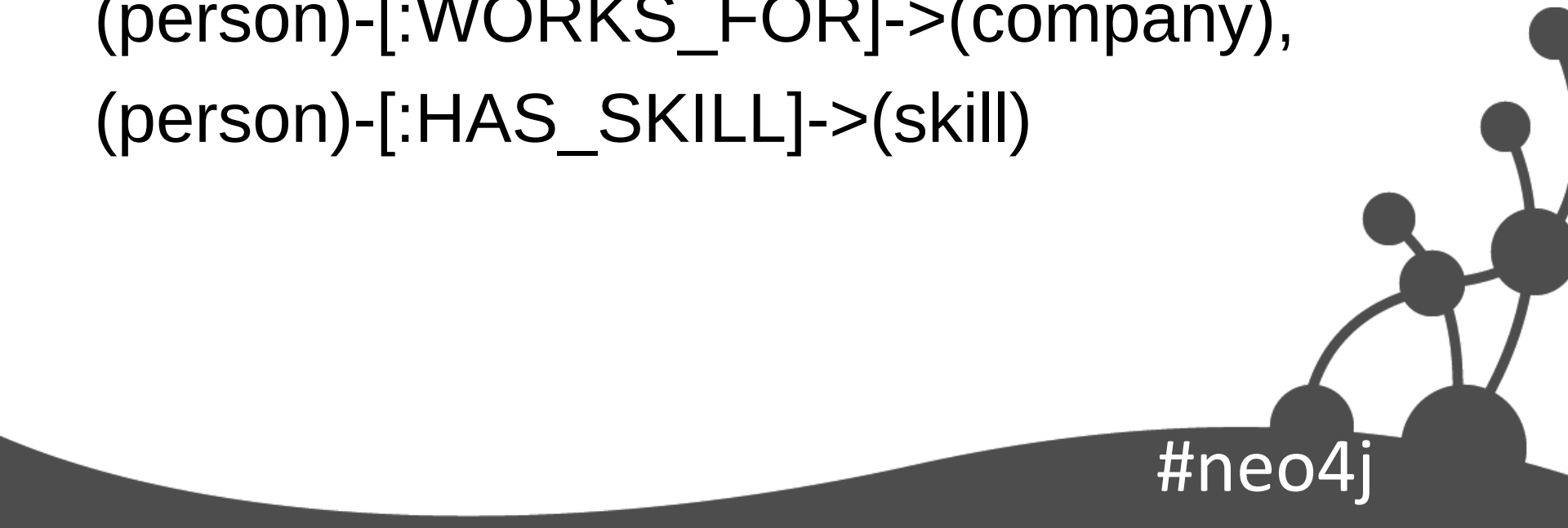


Convert to Cypher paths

person WORKS_FOR company

person HAS_SKILL skill

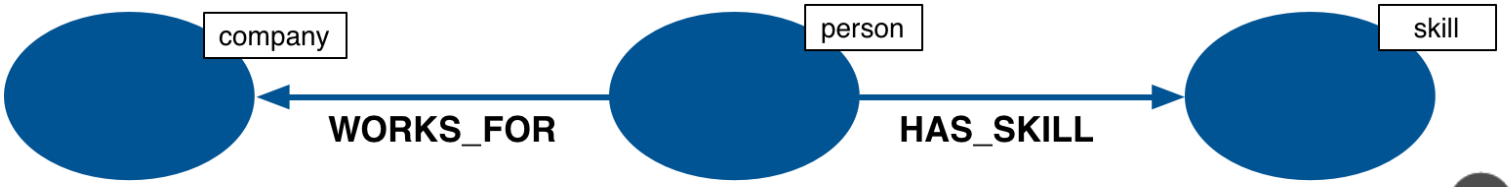
(person)-[:WORKS_FOR]->(company),
(person)-[:HAS_SKILL]->(skill)



Cypher paths

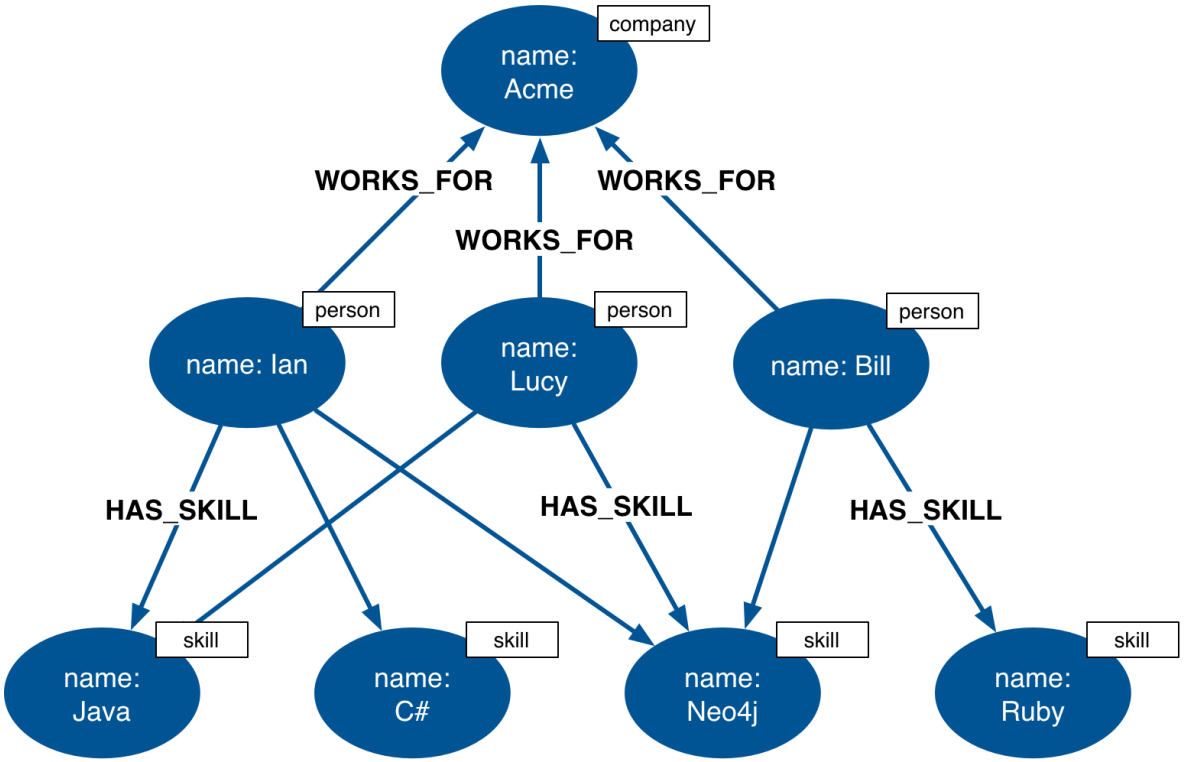
(person)-[:WORKS_FOR]->(company),
(person)-[:HAS_SKILL]->(skill)

(company)<-[:WORKS_FOR]-(person)-[:HAS_SKILL]->(skill)



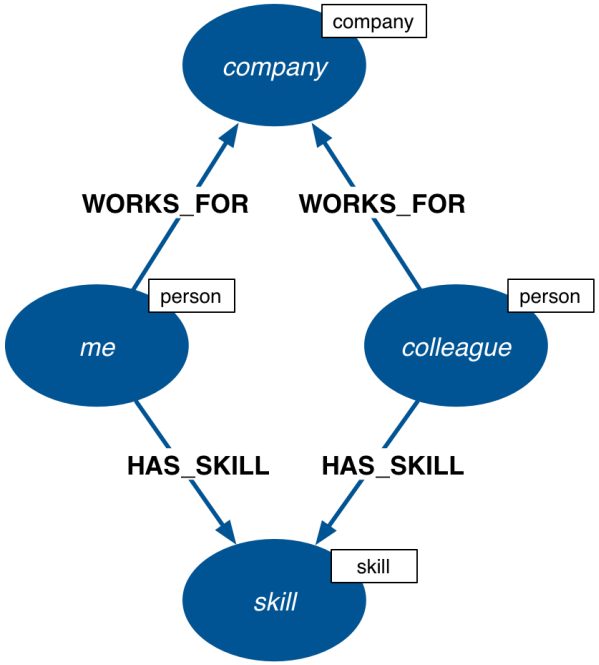
Data model

(company)<-[:WORKS_FOR]-(person)-[:HAS_SKILL]->(skill)



Formulating question as graph pattern

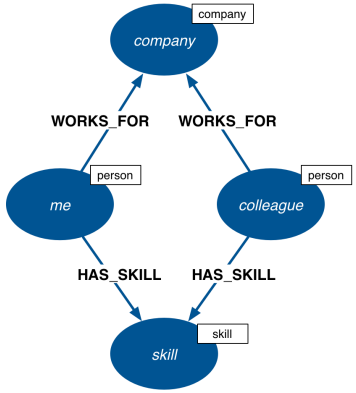
Which people, who work for the same company as me, have similar skills to me?



Cypher query

Which people, who work for the same company as me, have similar skills to me?

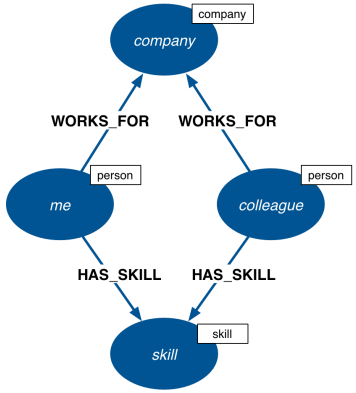
```
MATCH (company)<-[:WORKS_FOR]-(me:person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE  me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```



Graph pattern

Which people, who work for the same company as me, have similar skills to me?

```
MATCH (company)<-[:WORKS_FOR]-(me:person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE  me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```

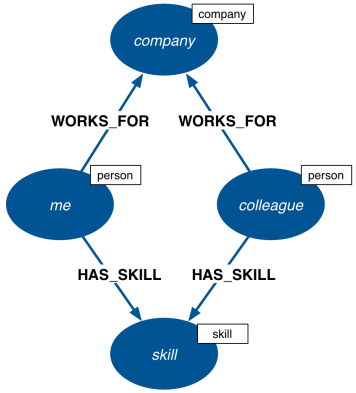


Anchor pattern in graph

Which people, who work for the same company as me, have similar skills to me?

```
MATCH (company)<-[:WORKS_FOR]-(me:person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE  me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DES
```

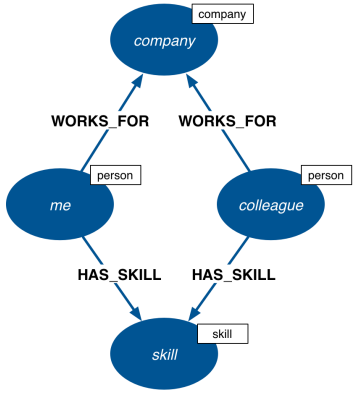
If an index for Person.name exists, Cypher will use it



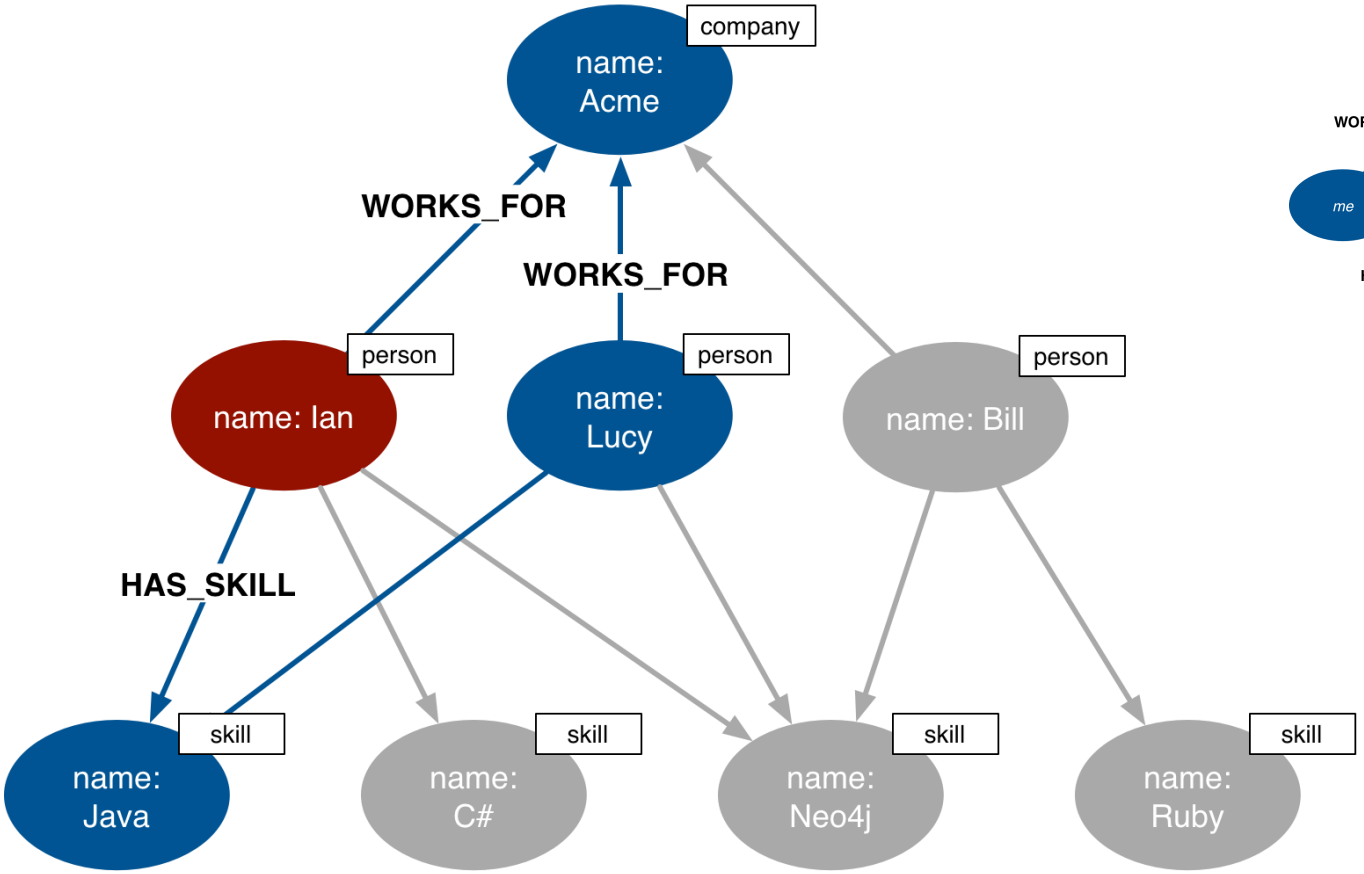
Create projection of results

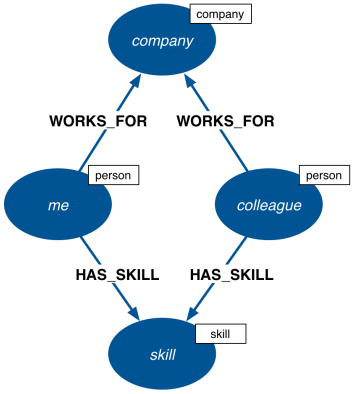
Which people, who work for the same company as me, have similar skills to me?

```
MATCH (company)<-[:WORKS_FOR]-(me:person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE  me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```

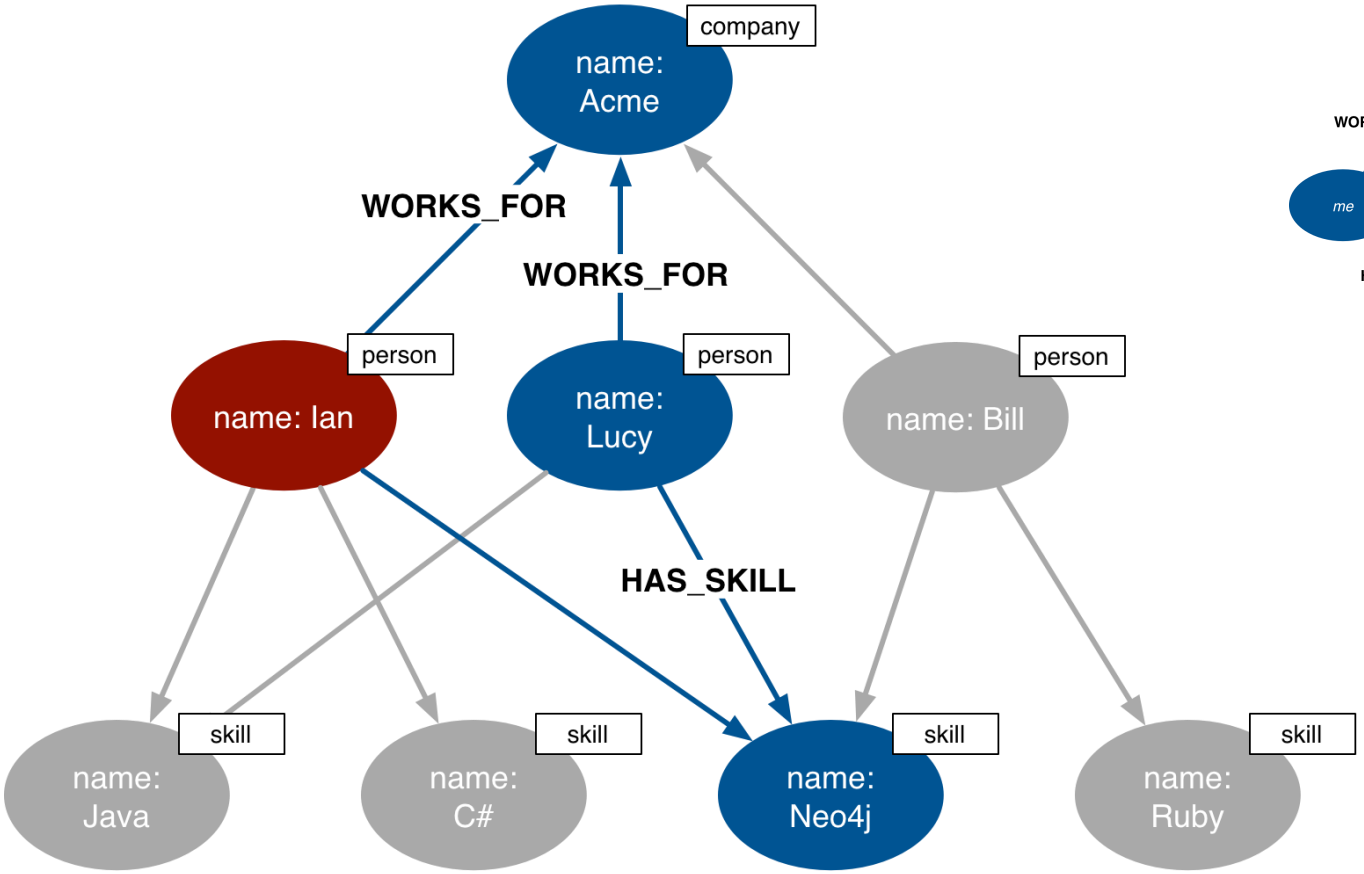


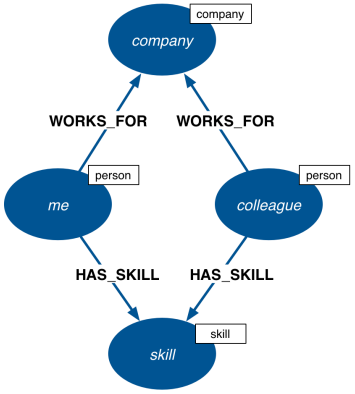
First match


company
name:
Acme
WORKS_FOR
WORKS_FOR
person
name: Ian
person
name:
Lucy
person
name: Bill
HAS_SKILL
skill
name:
Java
skill
name:
C#
skill
name:
Neo4j
skill
name:
Ruby

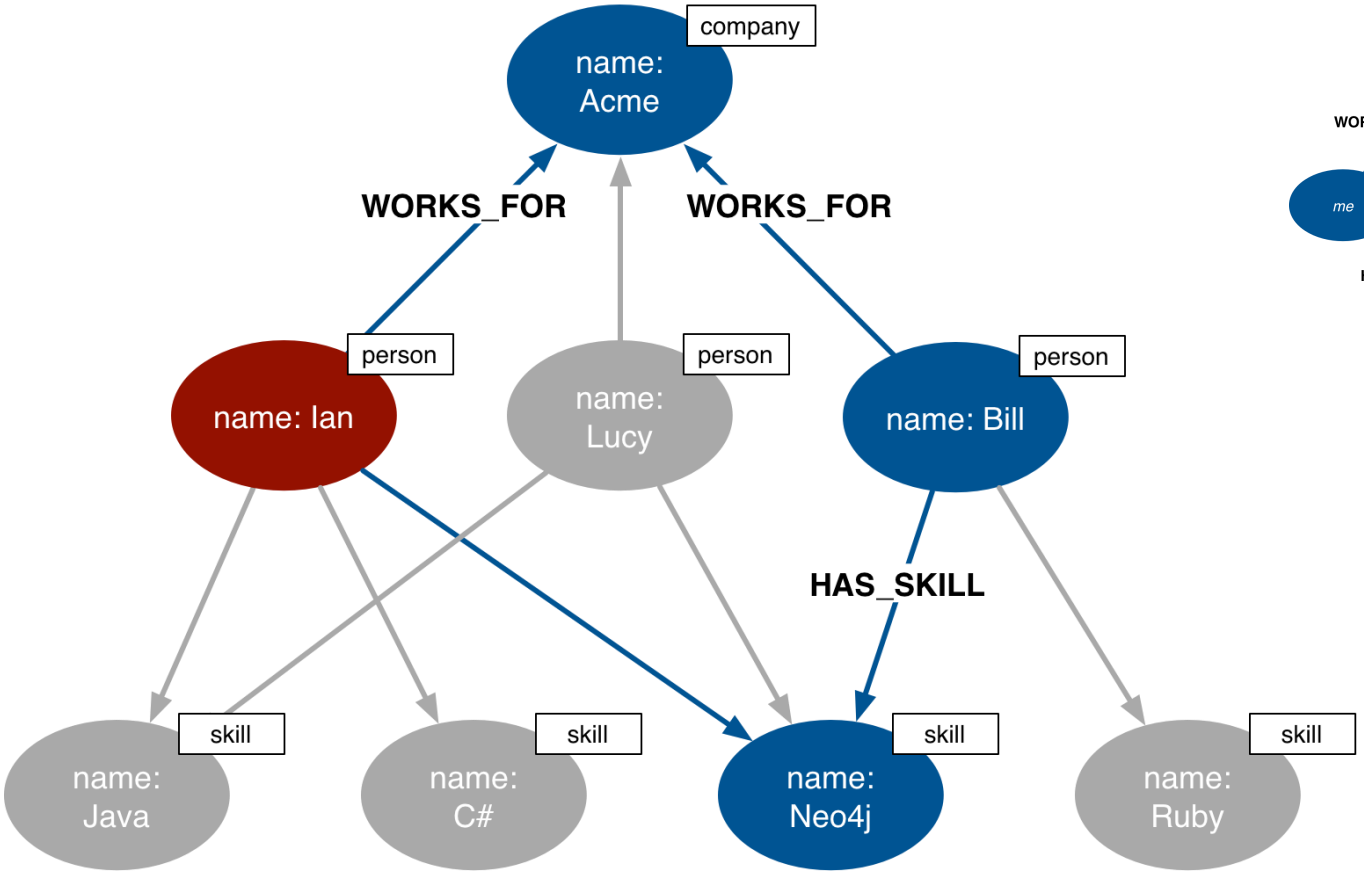

company
company
WORKS_FOR
WORKS_FOR
person
me
person
colleague
HAS_SKILL
HAS_SKILL
skill
skill

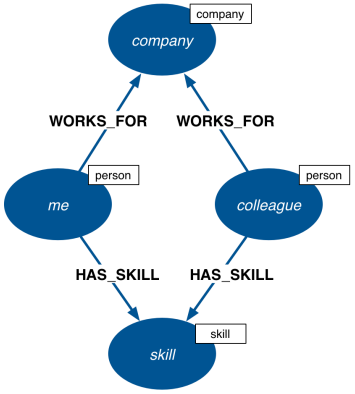
Second match





Third match

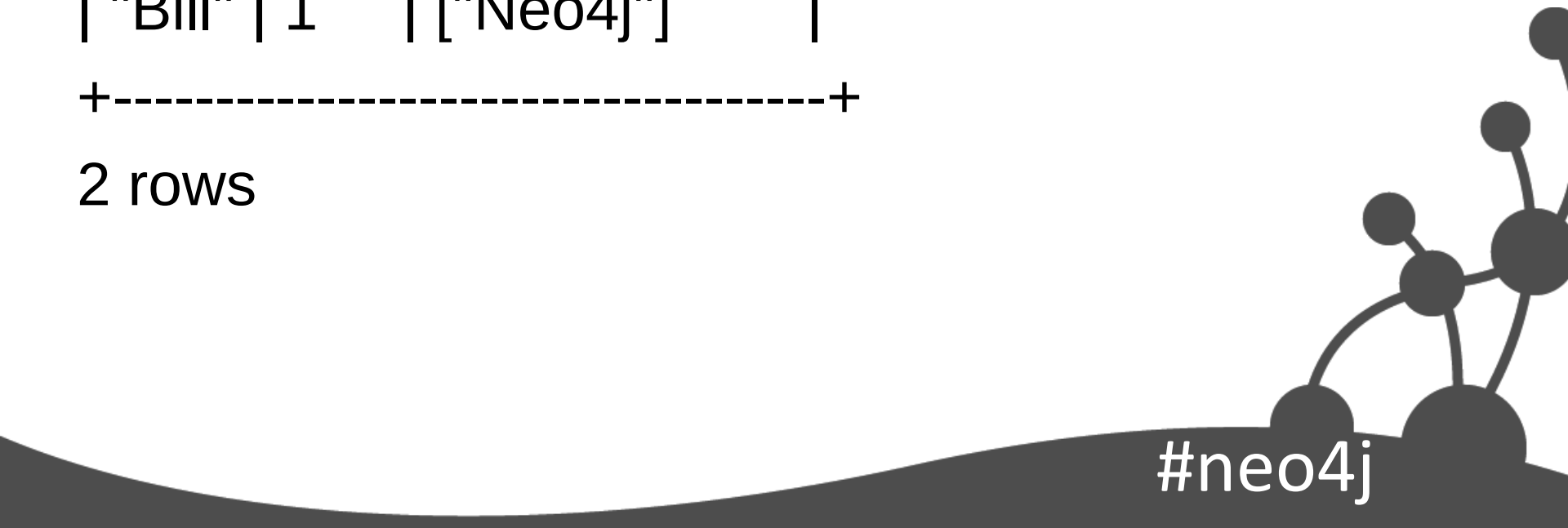




#neo4j

Running the query

```
+-----------------------------------+
| name    | score | skills          |
+-----------------------------------+
| "Lucy" | 2      | ["Java","Neo4j"] |
| "Bill" | 1      | ["Neo4j"]        |
+-----------------------------------+
2 rows
```



From user story to model

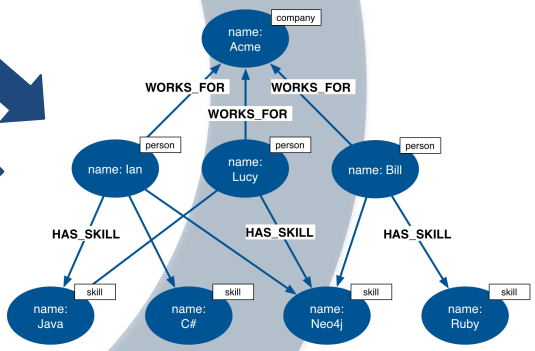
As an *employee*

I want *to know who in the company has similar skills to me*

So that *we can exchange knowledge*

```
MATCH (company)<-[:WORKS_FOR]-(me:person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE  me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```

Which people, who work for the same company as me, have similar skills to me?



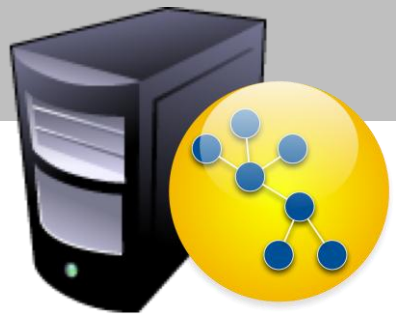
person WORKS_FOR company

person HAS_SKILL skill

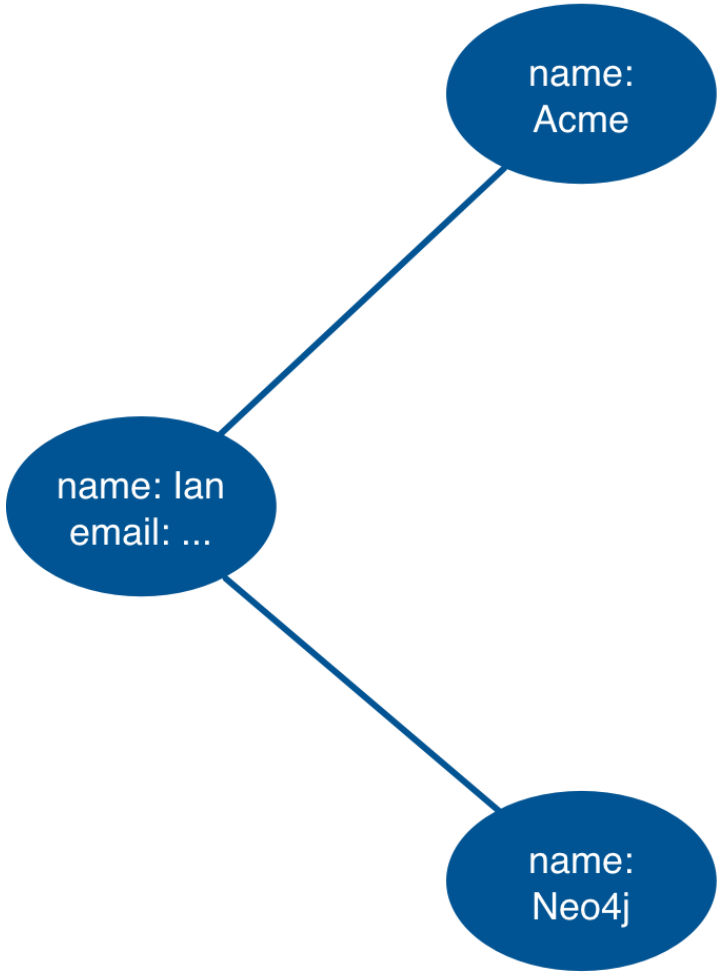
(company)<-[:WORKS_FOR]-(person)-[:HAS_SKILL]->(skill)

#neo4j

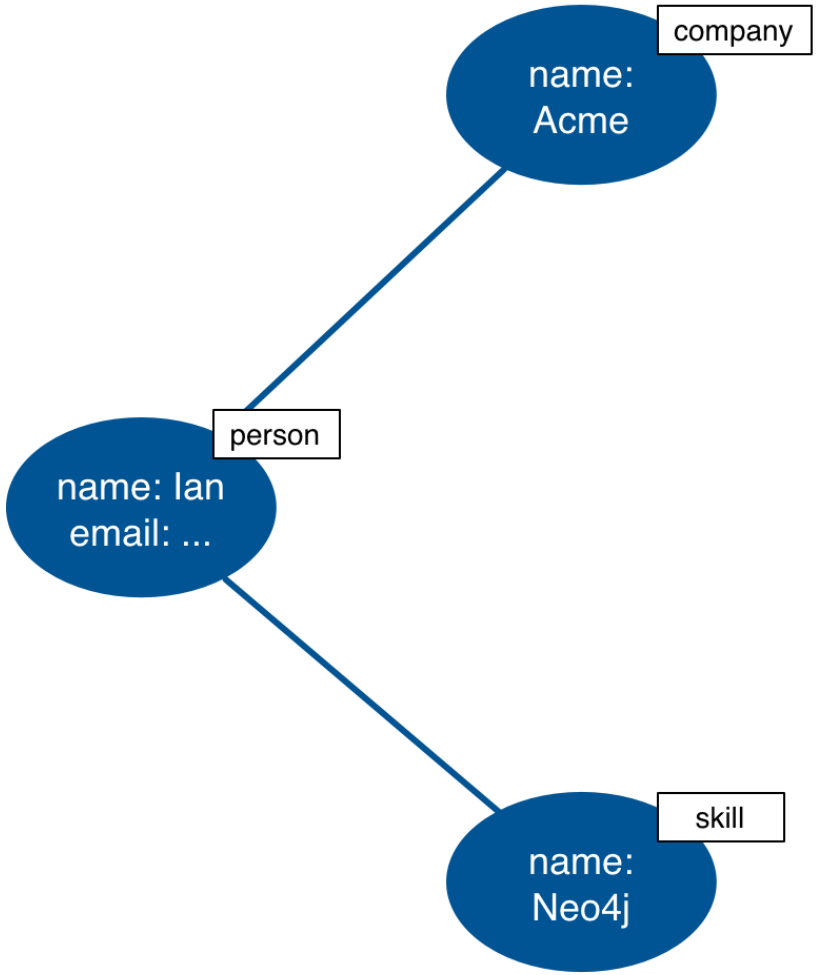
Modeling tips



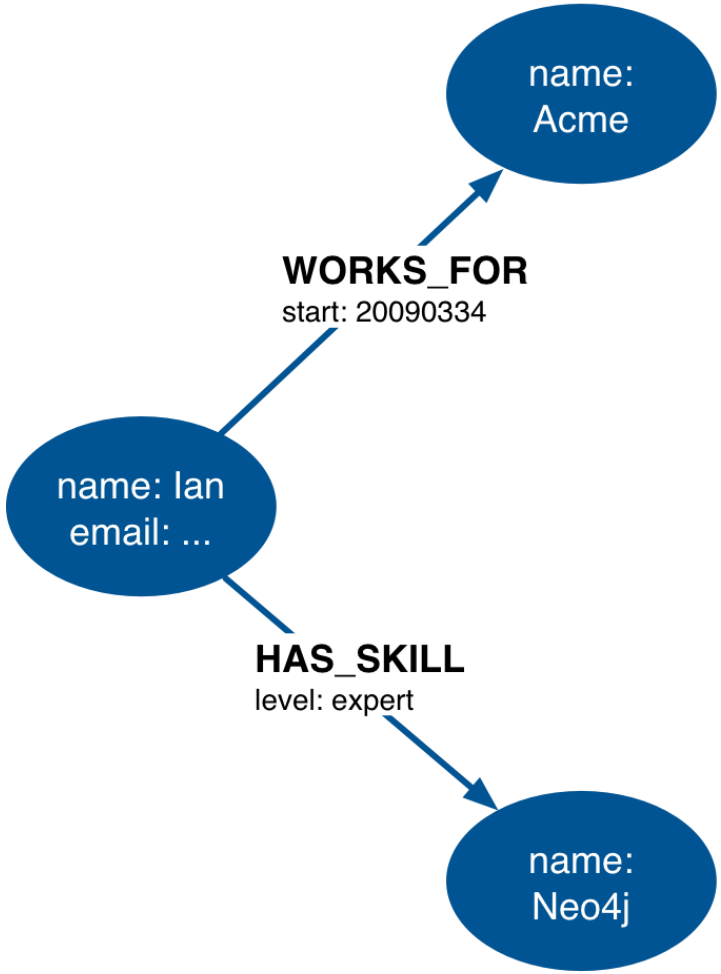
Nodes for things



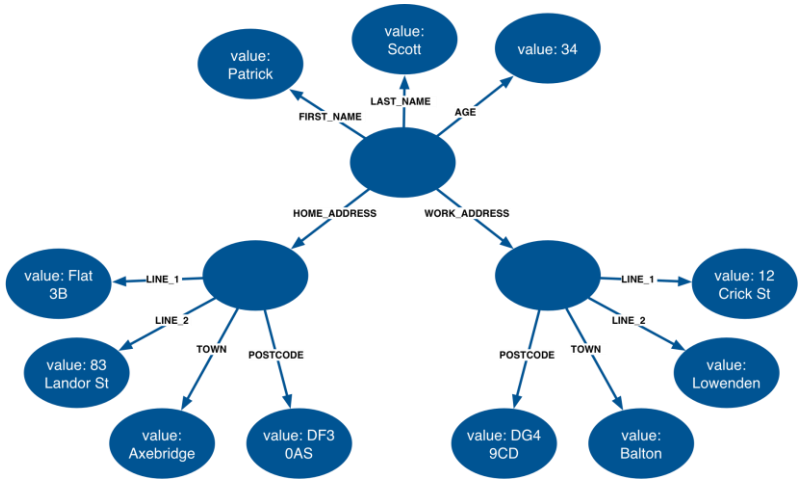
Labels for grouping



Relationships for structure



Properties vs Relationships



first-name: Patrick
last-name: Scott
age: 34

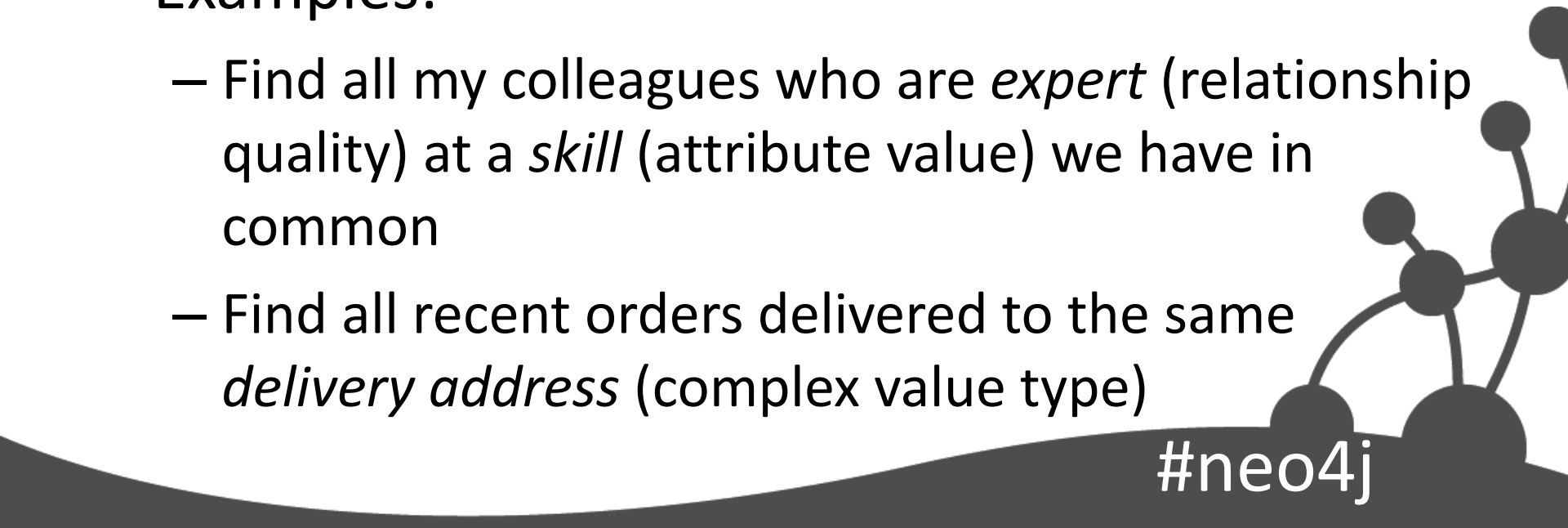
home-address: Flat 3B,
83 Landor St,
Axebridge,
DF3 0AS

work-address: Acme Ltd,
12 Crick St,
Balton,
DG4 9CD

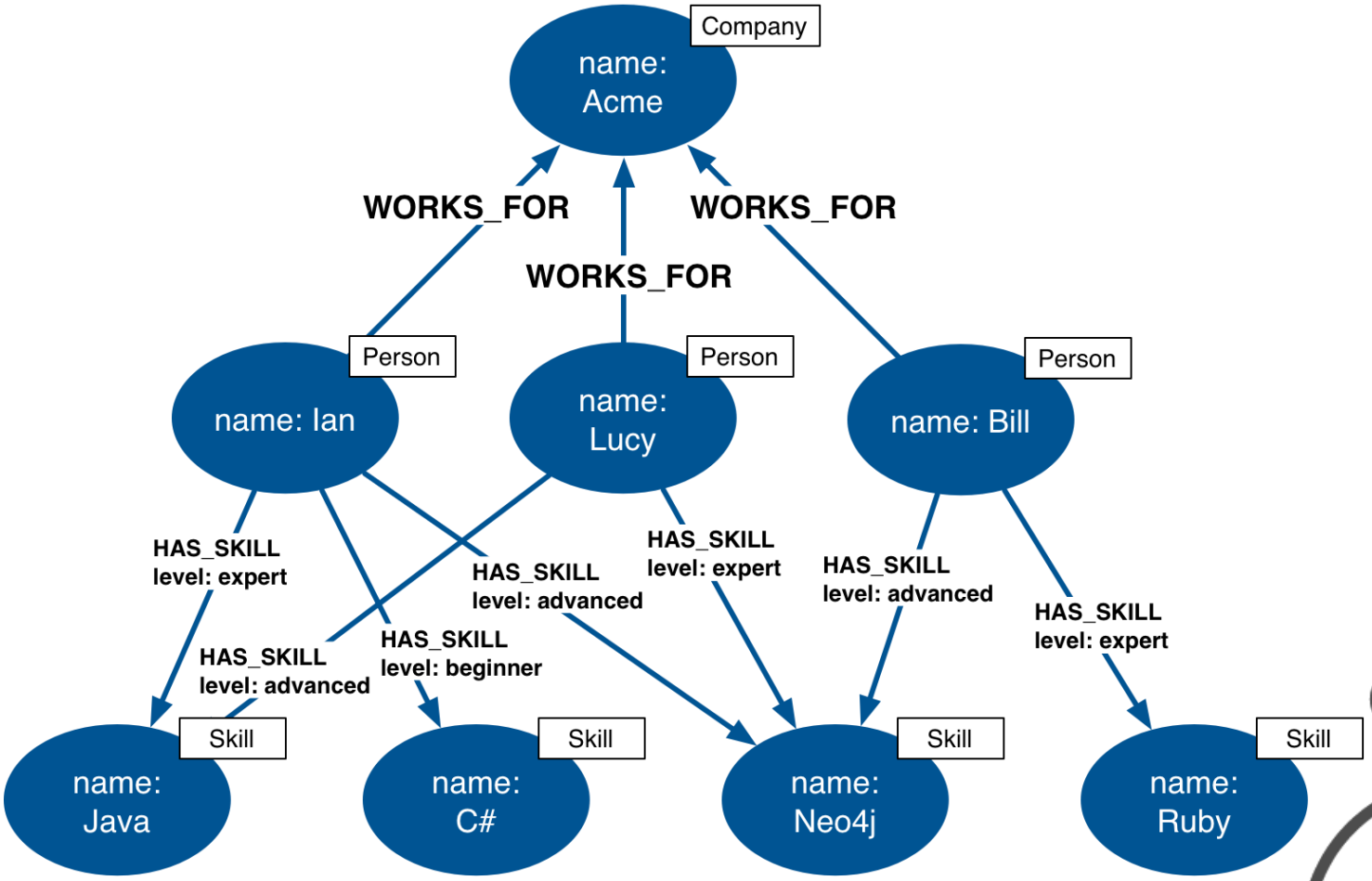
#neo4j

Use relationships when…

- You need to specify the weight, strength, or some other quality of the *relationship*
- AND/OR the attribute value comprises a complex value type (e.g. address)
- Examples:
  - Find all my colleagues who are *expert* (relationship quality) at a *skill* (attribute value) we have in common
  - Find all recent orders delivered to the same *delivery address* (complex value type)



Find Expert Colleagues



#neo4j

Find Expert Colleagues

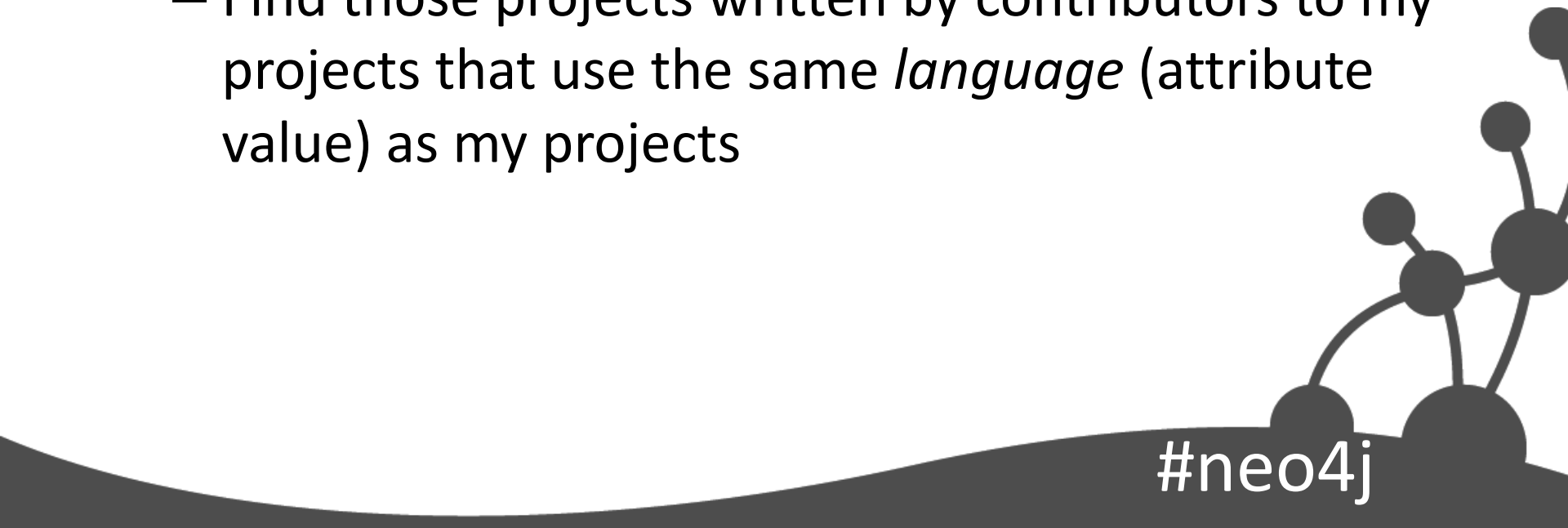
```
MATCH (user:Person)-[:HAS_SKILL]->(skill),
      (user)-[:WORKS_FOR]->(company),
      (colleague)-[:WORKS_FOR]->(company),
      (colleague)-[r:HAS_SKILL]->(skill)
WHERE user.name = {name} AND r.level = {skillLevel}
RETURN colleague.name AS name, skill.name AS skill
```

Relate and Filter

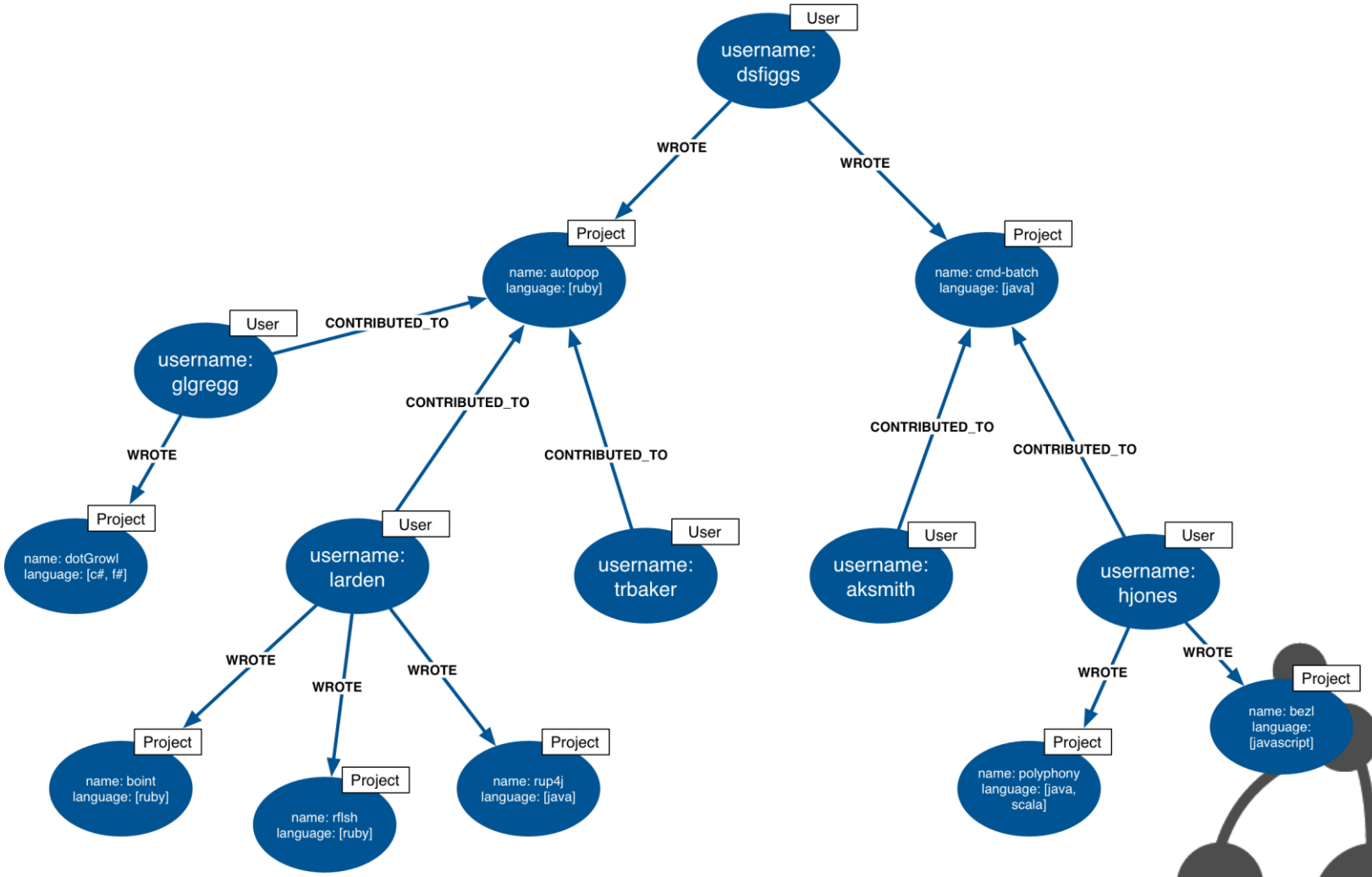
```
MATCH (user:Person)-[:HAS_SKILL]->(skill),
      (user)-[:WORKS_FOR]->(company),
      (colleague)-[:WORKS_FOR]->(company),
      (colleague)-[r:HAS_SKILL]->(skill)
WHERE user.name = {name} AND r.level = {skillLevel}
RETURN colleague.name AS name, skill.name AS skill
```

Use properties when…

- There's no need to qualify the relationship
- AND the attribute value comprises a simple value type (e.g. colour)
- Examples:
  - Find those projects written by contributors to my projects that use the same *language* (attribute value) as my projects



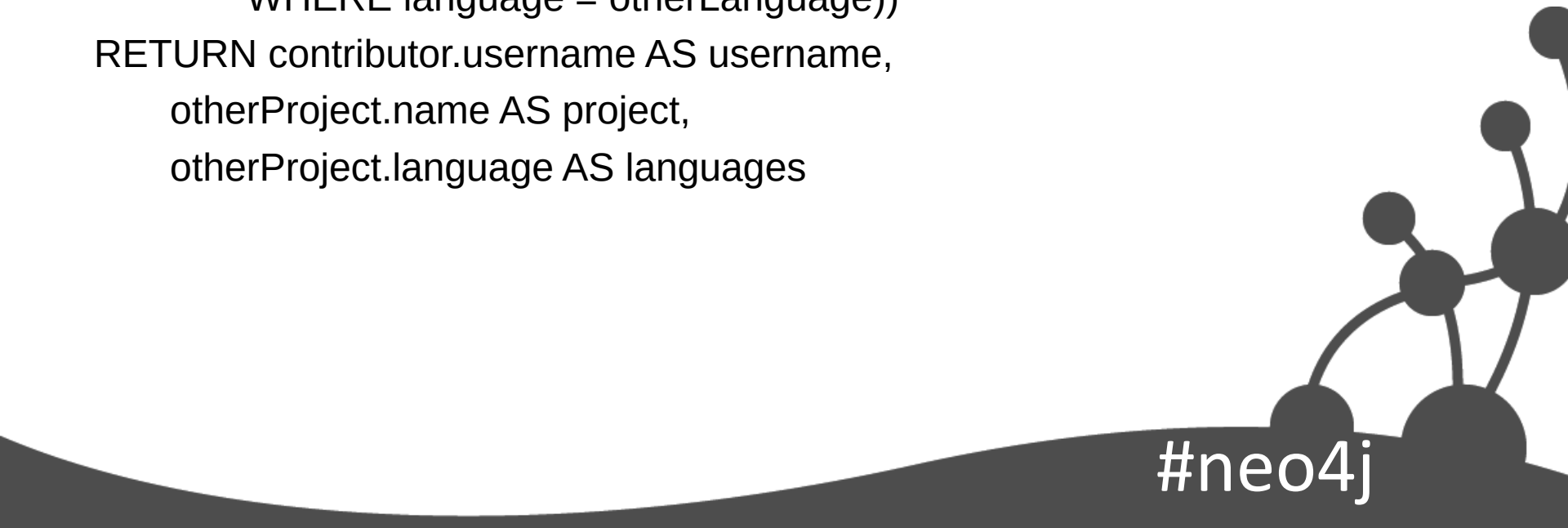
Find Projects With Same Languages



#neo4j

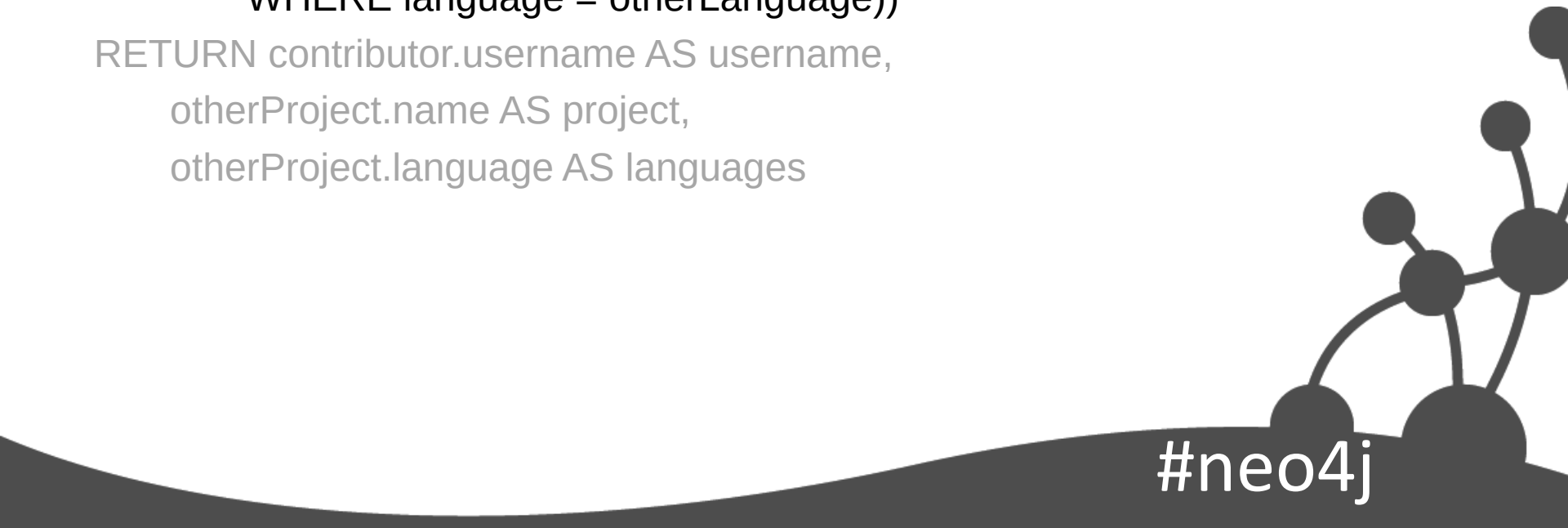
Find Projects With Same Languages

```
MATCH (user:User)-[:WROTE]->(project:Project),
      (contributor)-[:CONTRIBUTED_TO]->(project),
      (contributor)-[:WROTE]->(otherProject:Project)
WHERE user.username = {username}
      AND ANY (otherLanguage IN otherProject.language
          WHERE ANY (language IN project.language
              WHERE language = otherLanguage))
RETURN contributor.username AS username,
       otherProject.name AS project,
       otherProject.language AS languages
```



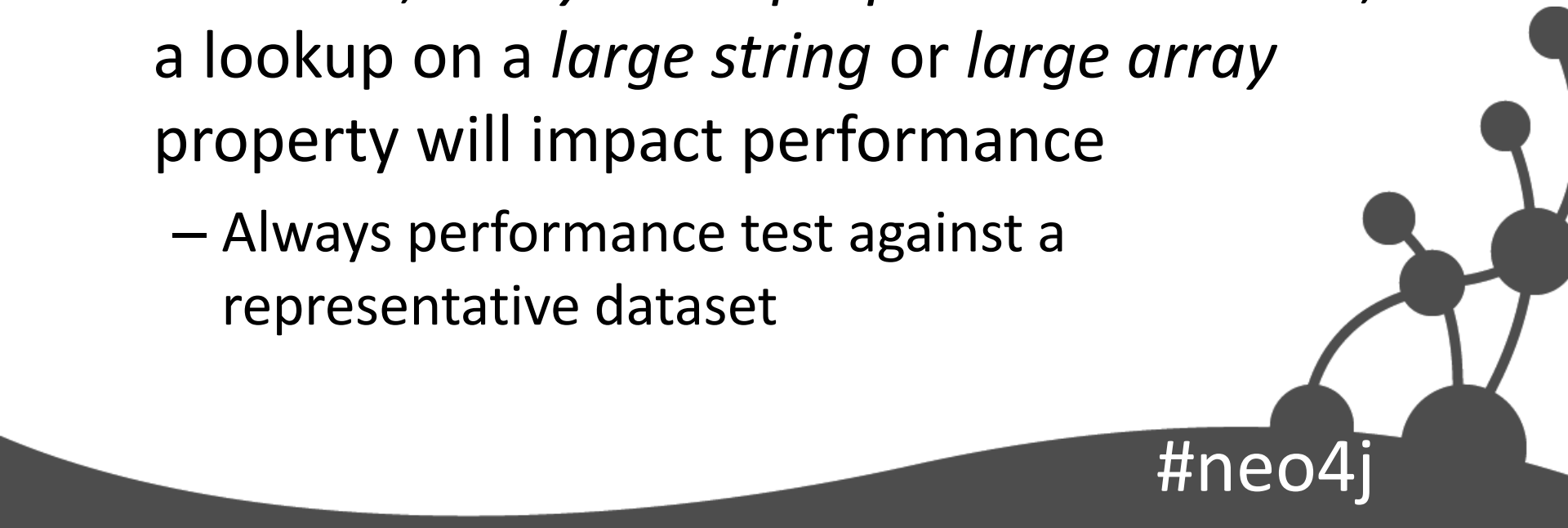
Relate and Filter

```
MATCH (user:User)-[:WROTE]->(project:Project),
      (contributor)-[:CONTRIBUTED_TO]->(project),
      (contributor)-[:WROTE]->(otherProject:Project)
WHERE user.username = {username}
      AND ANY (otherLanguage IN otherProject.language
          WHERE ANY (language IN project.language
              WHERE language = otherLanguage))
RETURN contributor.username AS username,
       otherProject.name AS project,
       otherProject.language AS languages
```

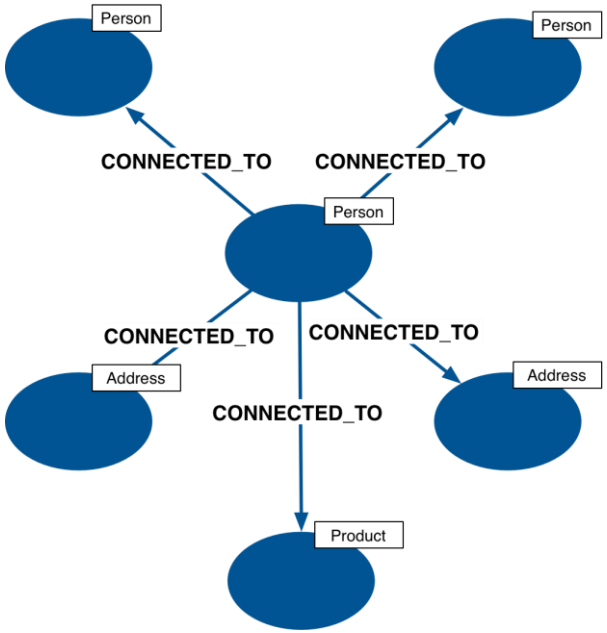


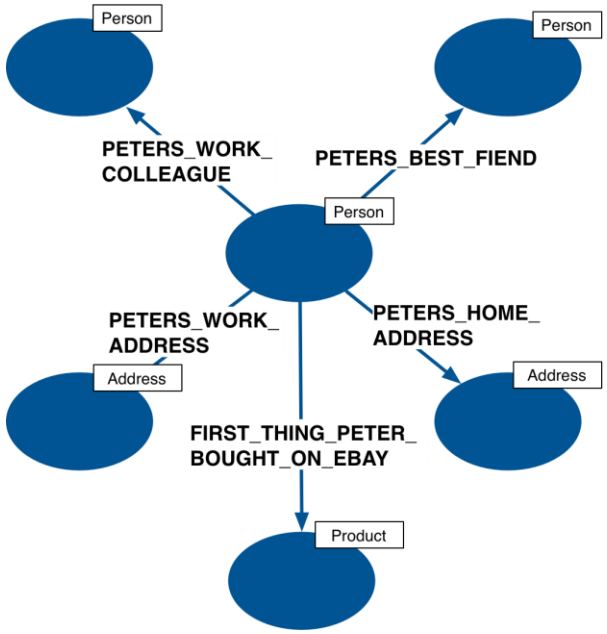
If Performance is Critical…

- Small property lookup on a node will be quicker than traversing a relationship
  - But traversing a relationship is still faster than a SQL join…
- However, *many small properties* on a node, or a lookup on a *large string* or *large array* property will impact performance
  - Always performance test against a representative dataset



Relationship Granularity

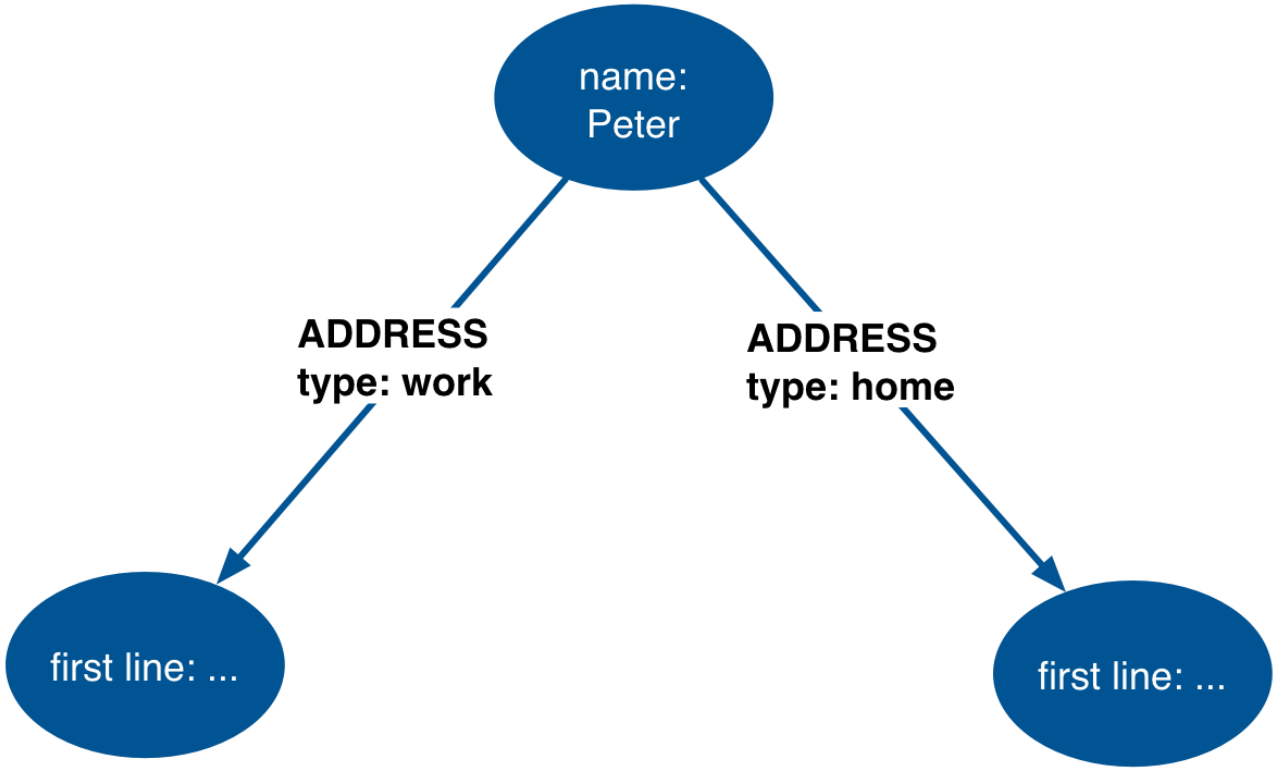




#neo4j

General Relationships

- Qualified by property



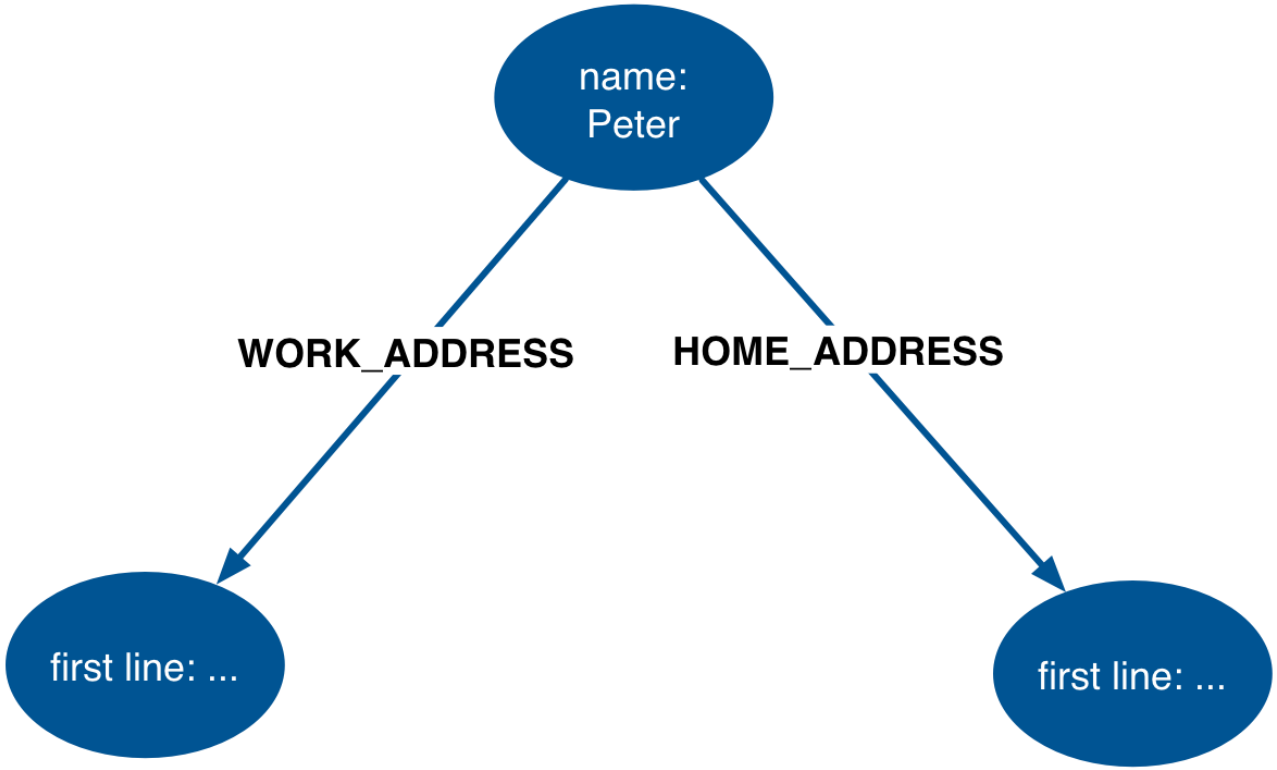
Easy to Query Across All Types

```
MATCH  (person)-[a:ADDRESS]->(address)
WHERE  person.name = {name}
RETURN a.type AS type,
       address.firstline AS firstline
```

Property Access to Discover Sub-Types

```
MATCH  (person)-[a:ADDRESS]->(address)
WHERE  person.name = {name}
       AND a.type = {type}
RETURN address.firstline AS firstline
```

Specific Relationships

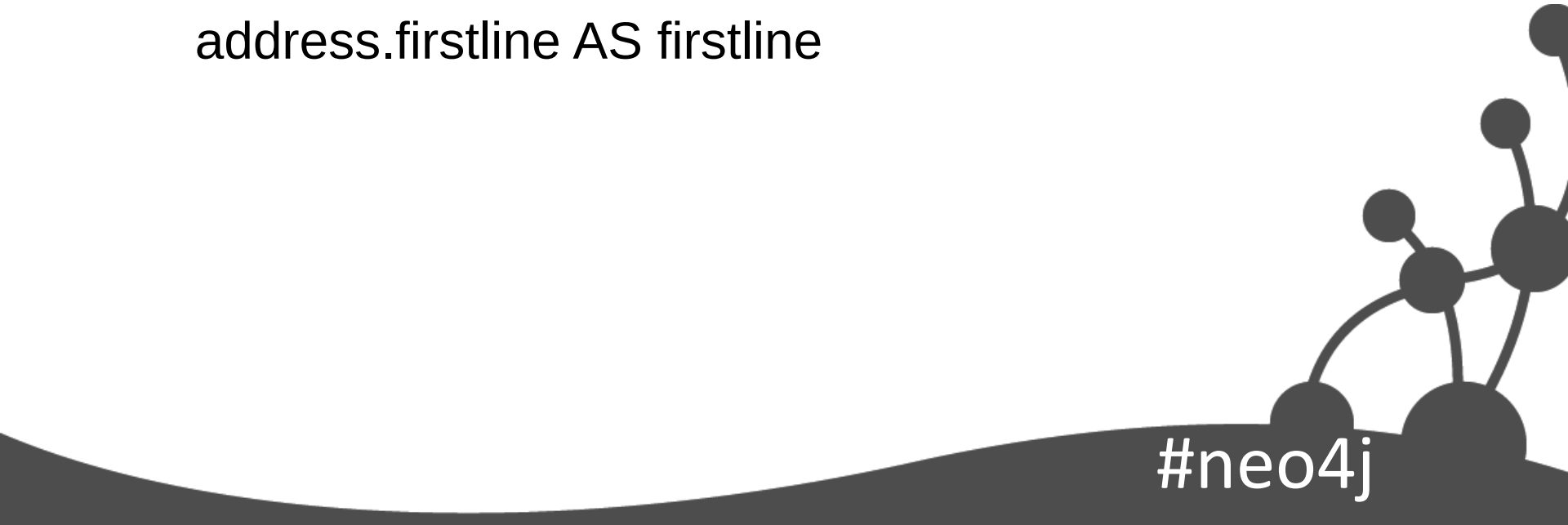


Easy to Query Specific Types

```
MATCH  (person)-[:HOME_ADDRESS]->(address)
WHERE  person.name = {name}
RETURN address.firstline AS firstline
```

Cumbersome to Discover All Types

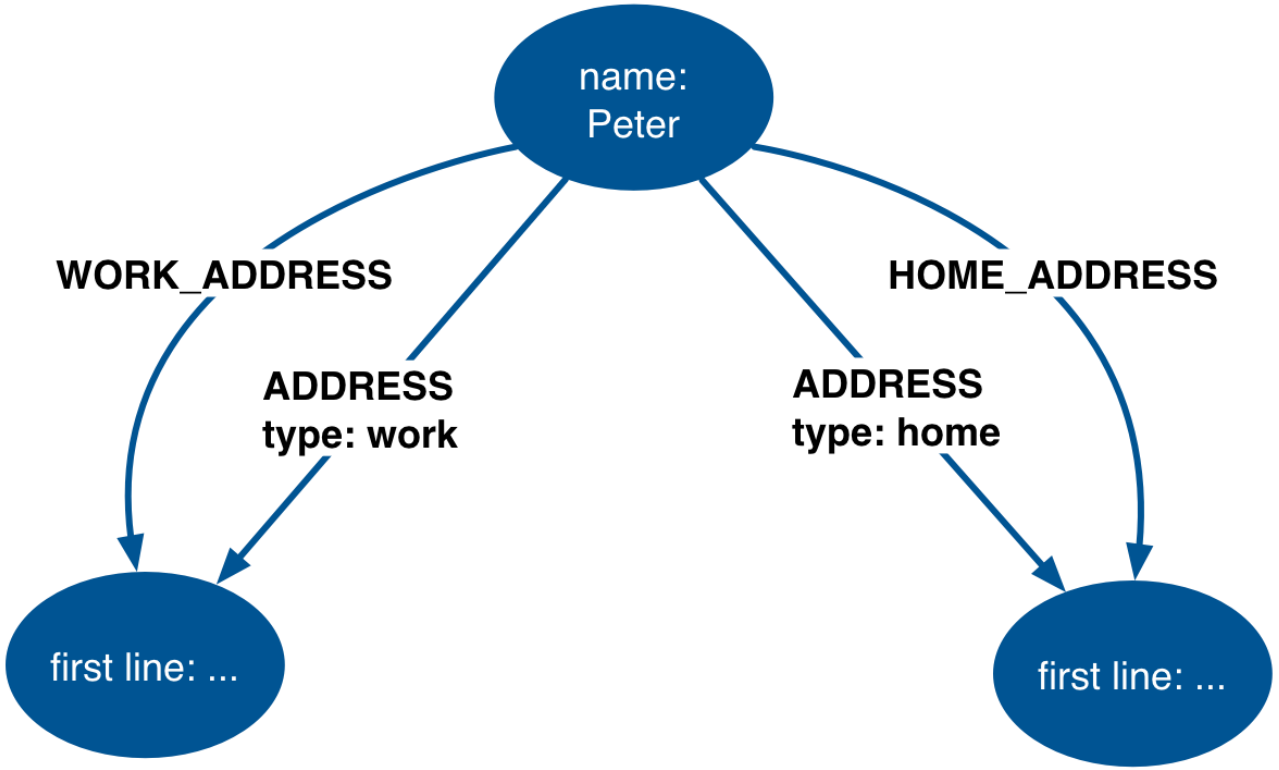
```
MATCH  (person)-
       [a:HOME_ADDRESS|WORK_ADDRESS]
       ->(address)
WHERE  person.name = {name}
RETURN type(a) AS type,
       address.firstline AS firstline
```



Cumbersome to Discover All Types

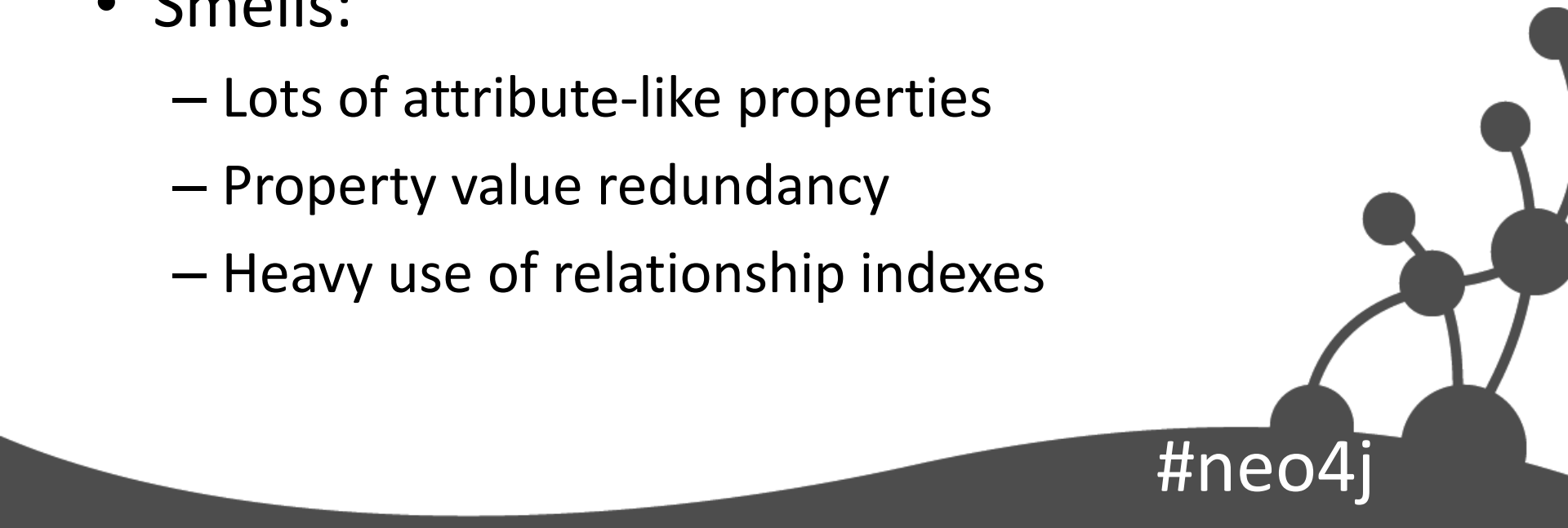
```
MATCH  (person)-
       [a:HOME_ADDRESS|WORK_ADDRESS]
       ->(address)
WHERE  person.name = {name}
RETURN type(a) AS type,
       address.firstline AS firstline
```

Best of Both Worlds

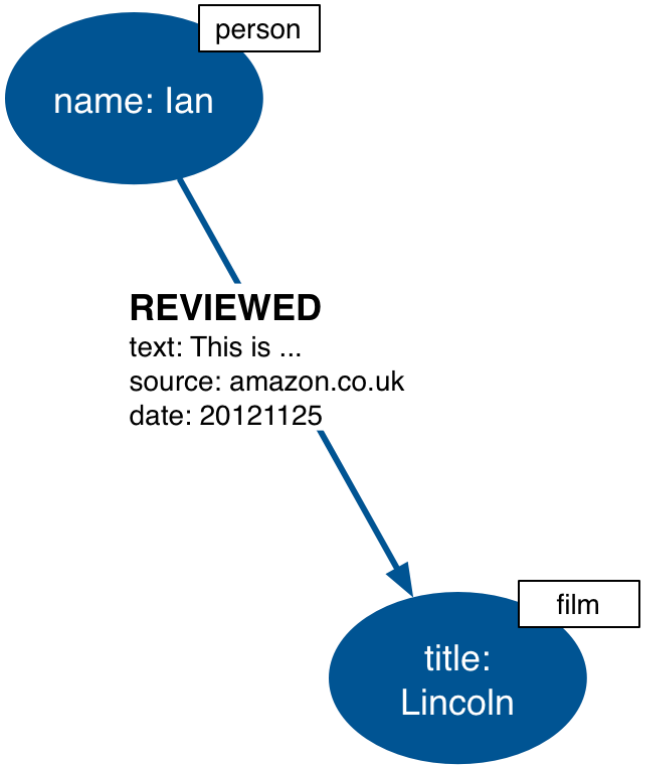


Don’t model entities as relationships

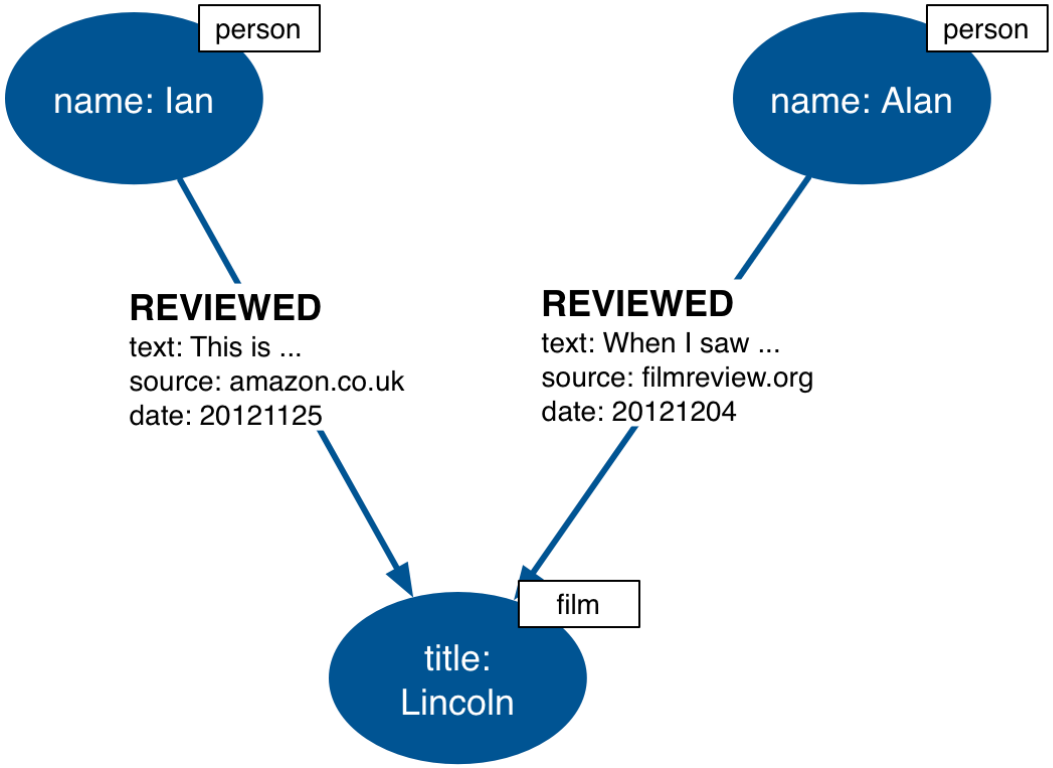
- Limits data model evolution
  - Unable to associate more entities
- Entities sometimes hidden in a verb
- Smells:
  - Lots of attribute-like properties
  - Property value redundancy
  - Heavy use of relationship indexes



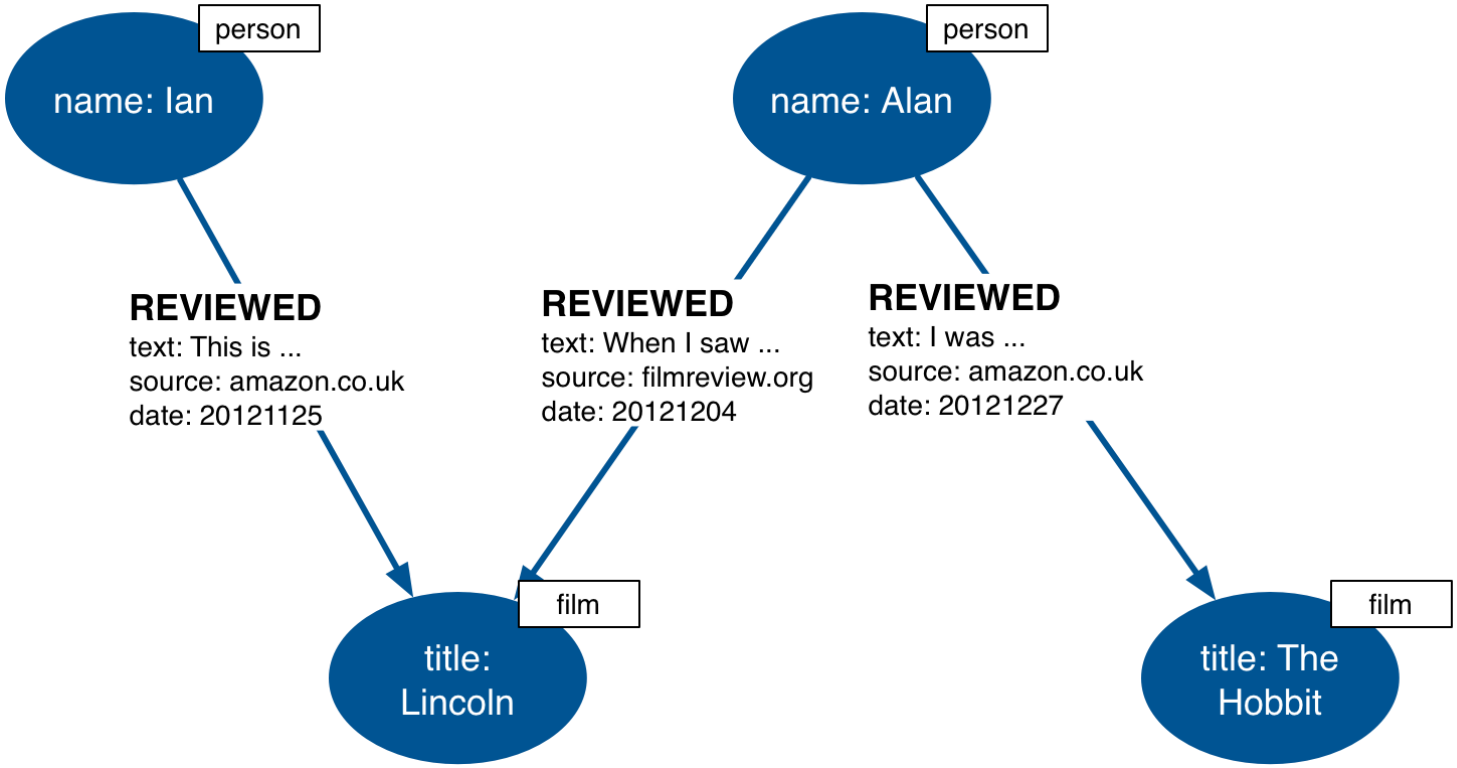
Example: Reviews



Add another review

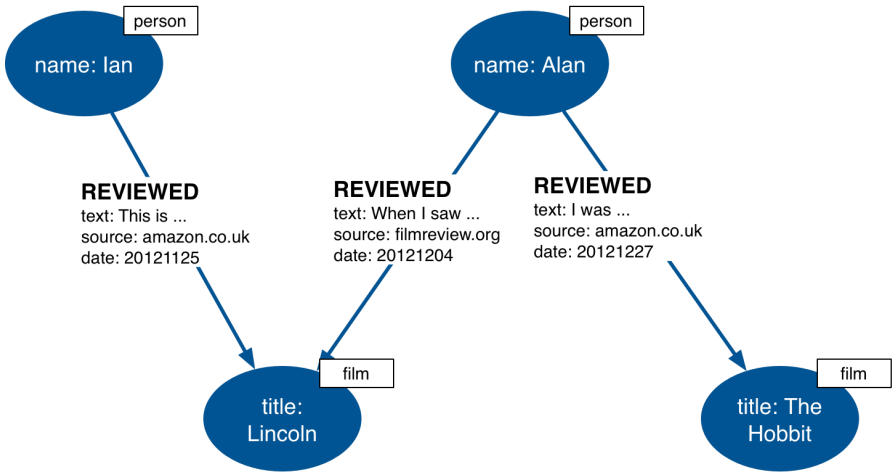


And another

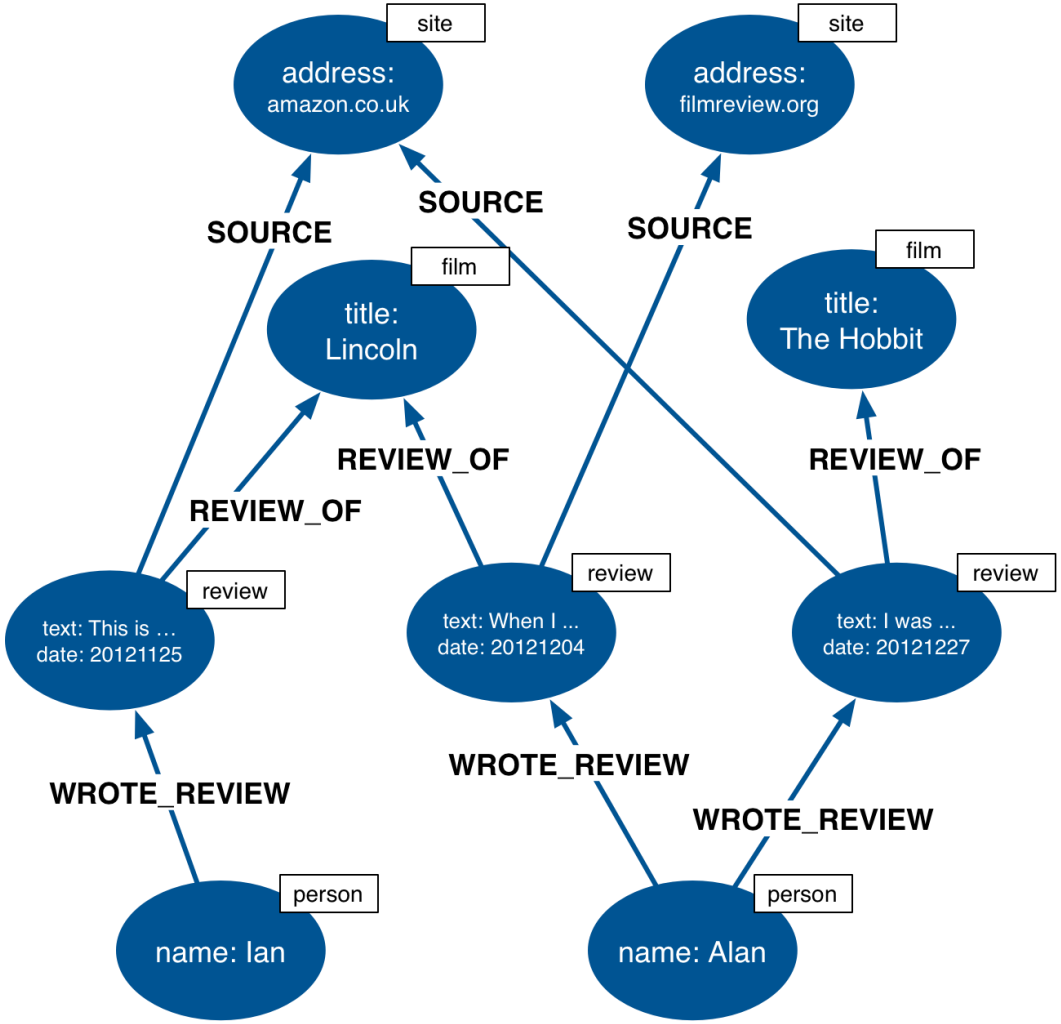

person
name: Ian
person
name: Alan
REVIEWED
text: This is ...
source: amazon.co.uk
date: 20121125
REVIEWED
text: When I saw ...
source: filmreview.org
date: 20121204
REVIEWED
text: I was ...
source: amazon.co.uk
date: 20121227
film
title: Lincoln
film
title: The Hobbit

Problems

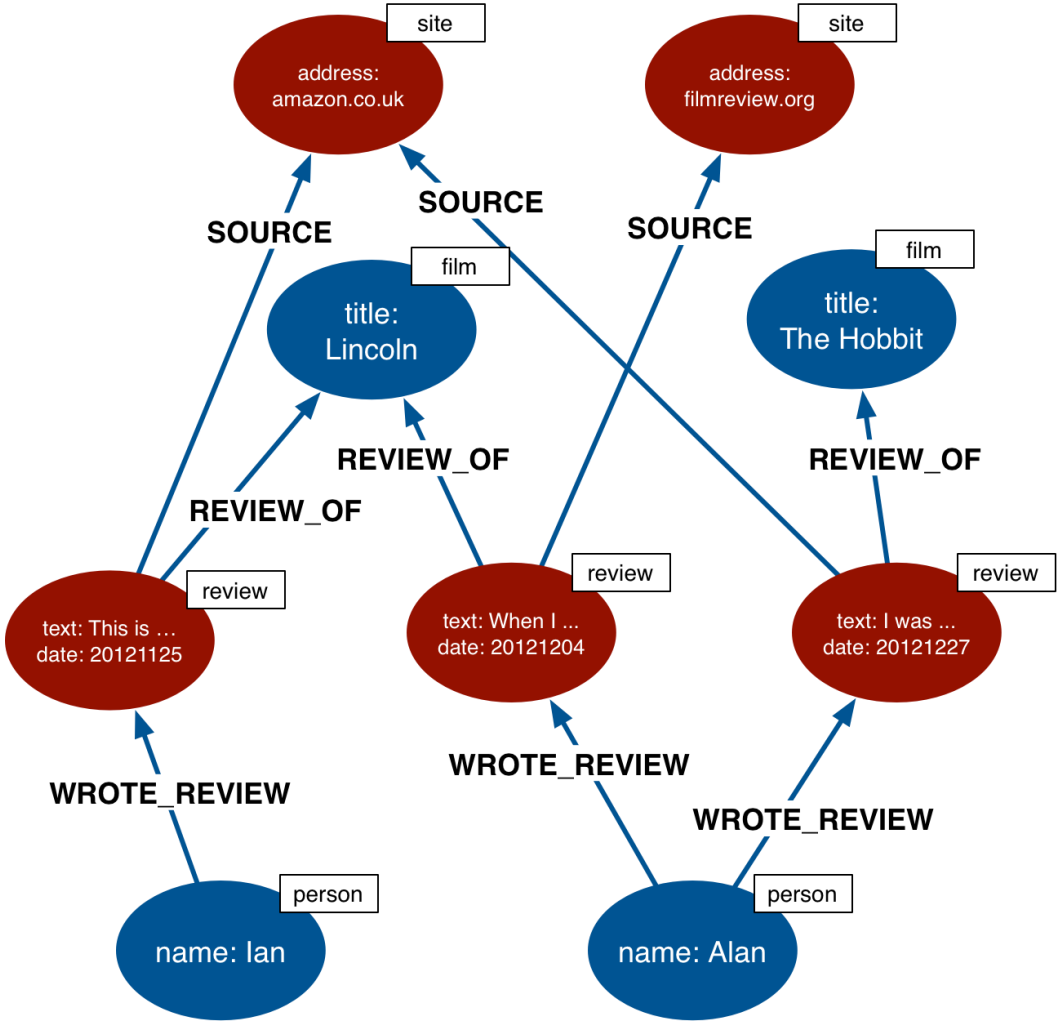
- Redundant data (2 x amazon.co.uk)
- Difficult to find reviews for source
- Users can't comment on reviews



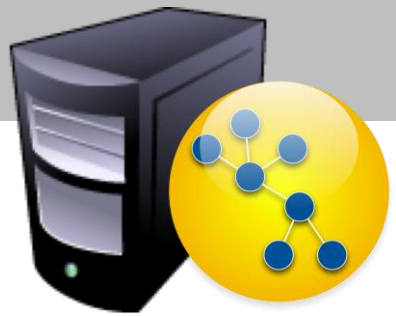
Revised model



Model actions in terms of products

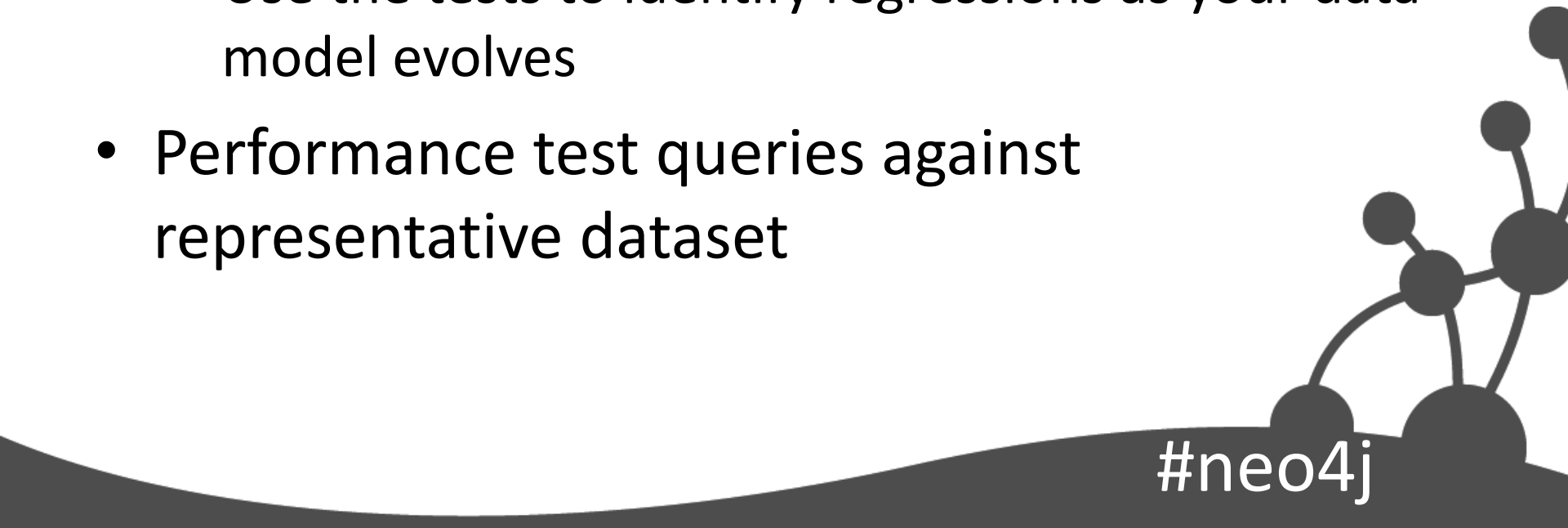


Testing

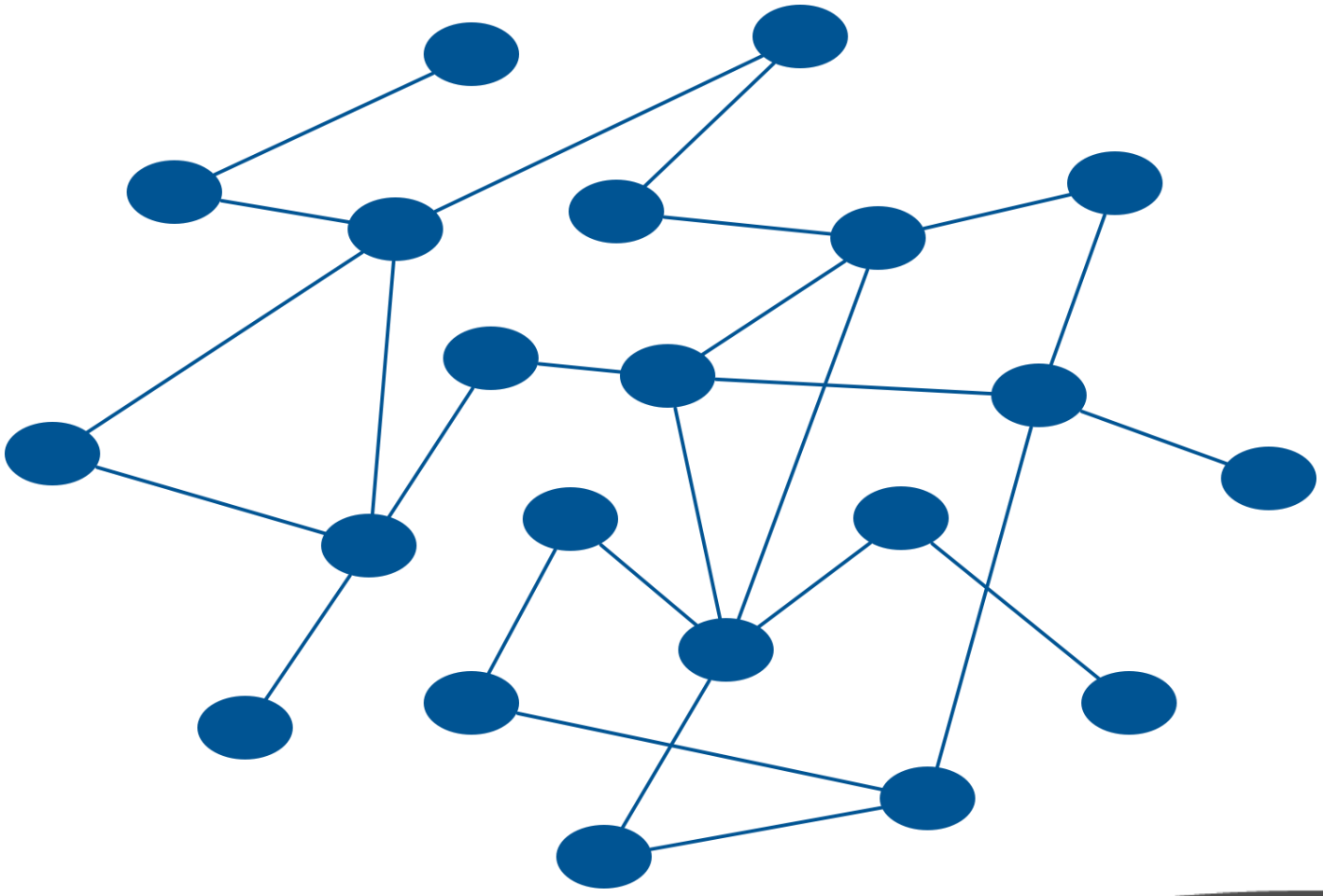


Test-driven data modeling

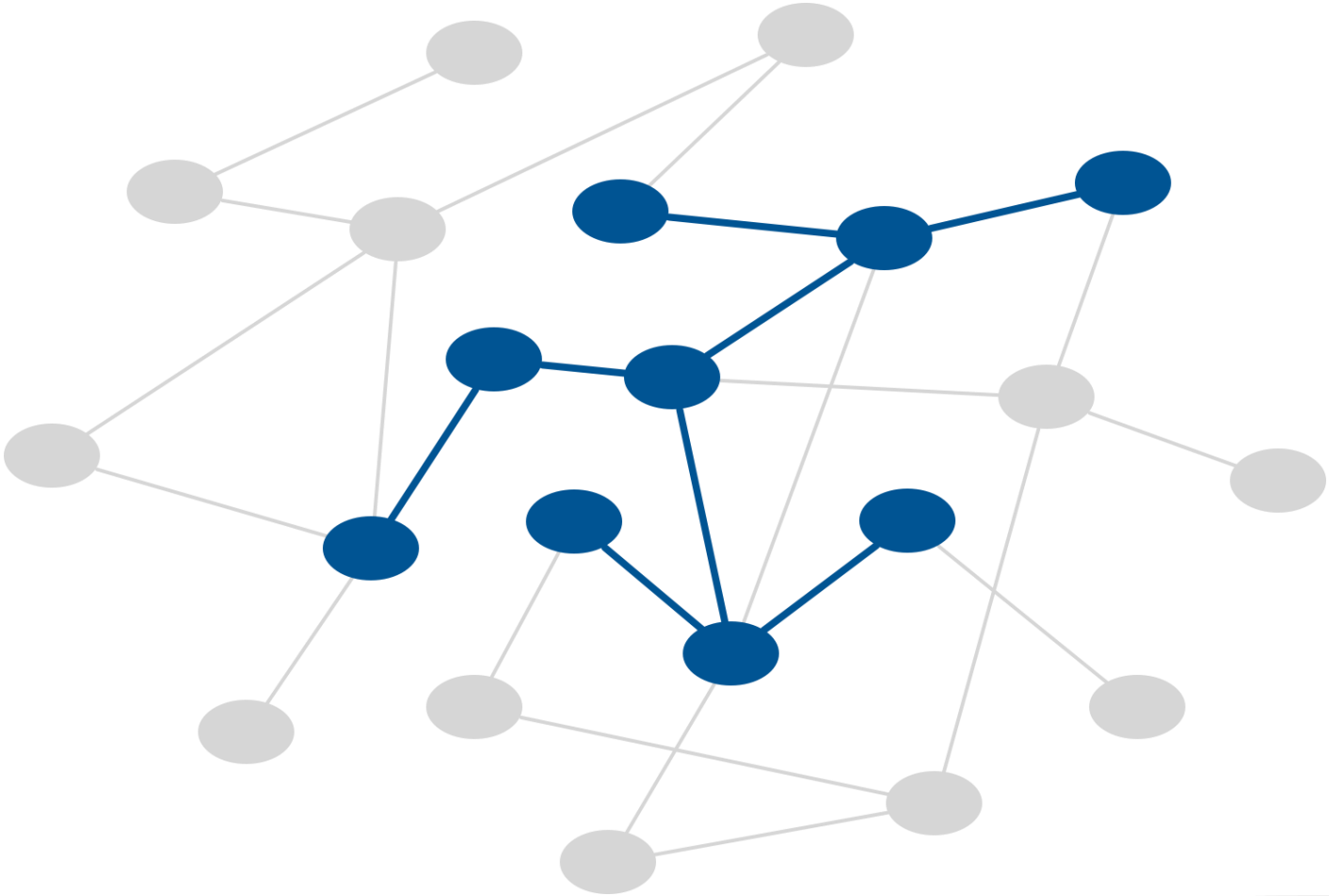
- Unit test with small, well-known datasets
  - Inject small graphs to test individual queries
  - Datasets express understanding of domain
  - Use the tests to identify regressions as your data model evolves
- Performance test queries against representative dataset



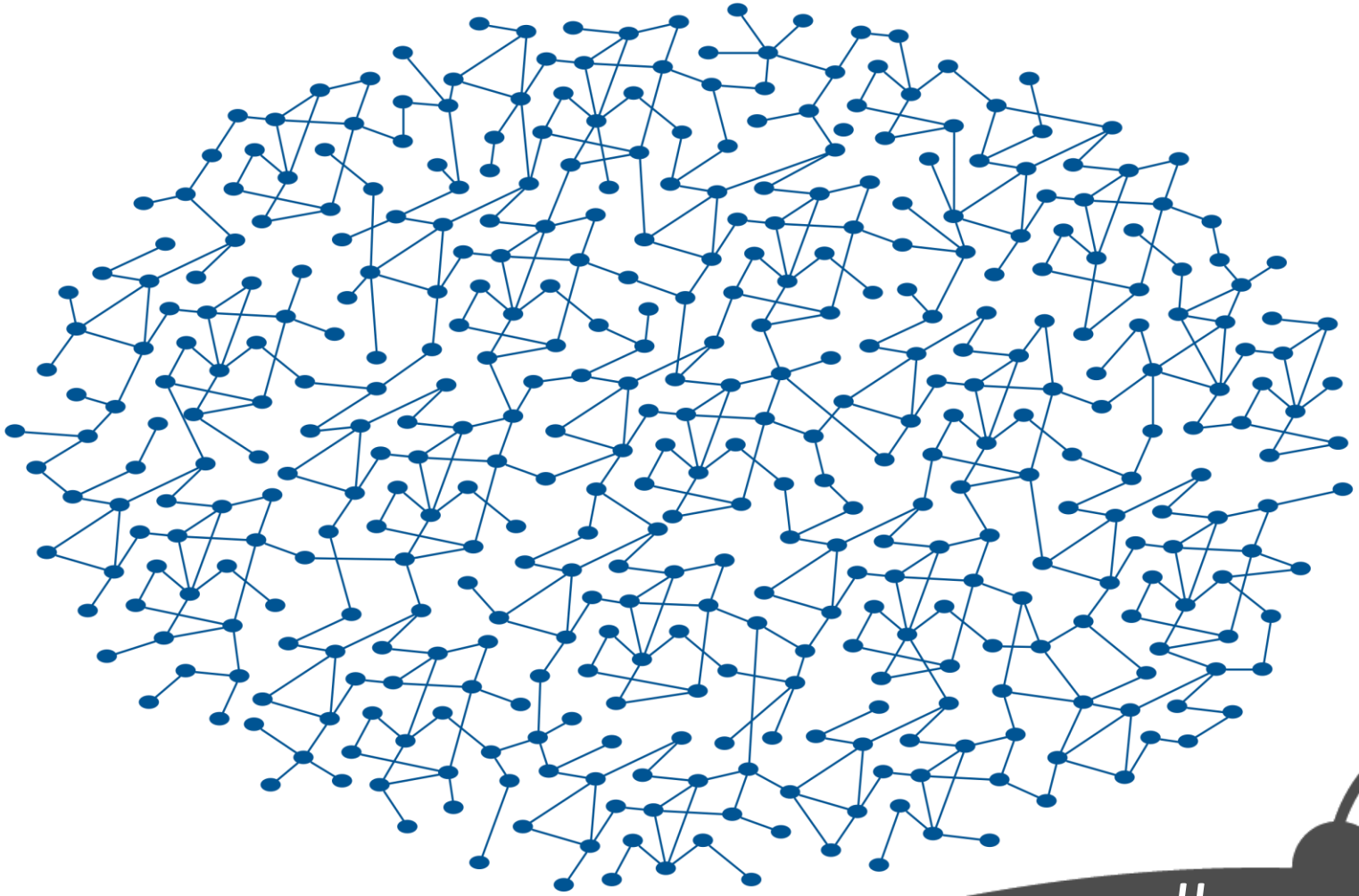
Query times proportional to size of subgraph searched



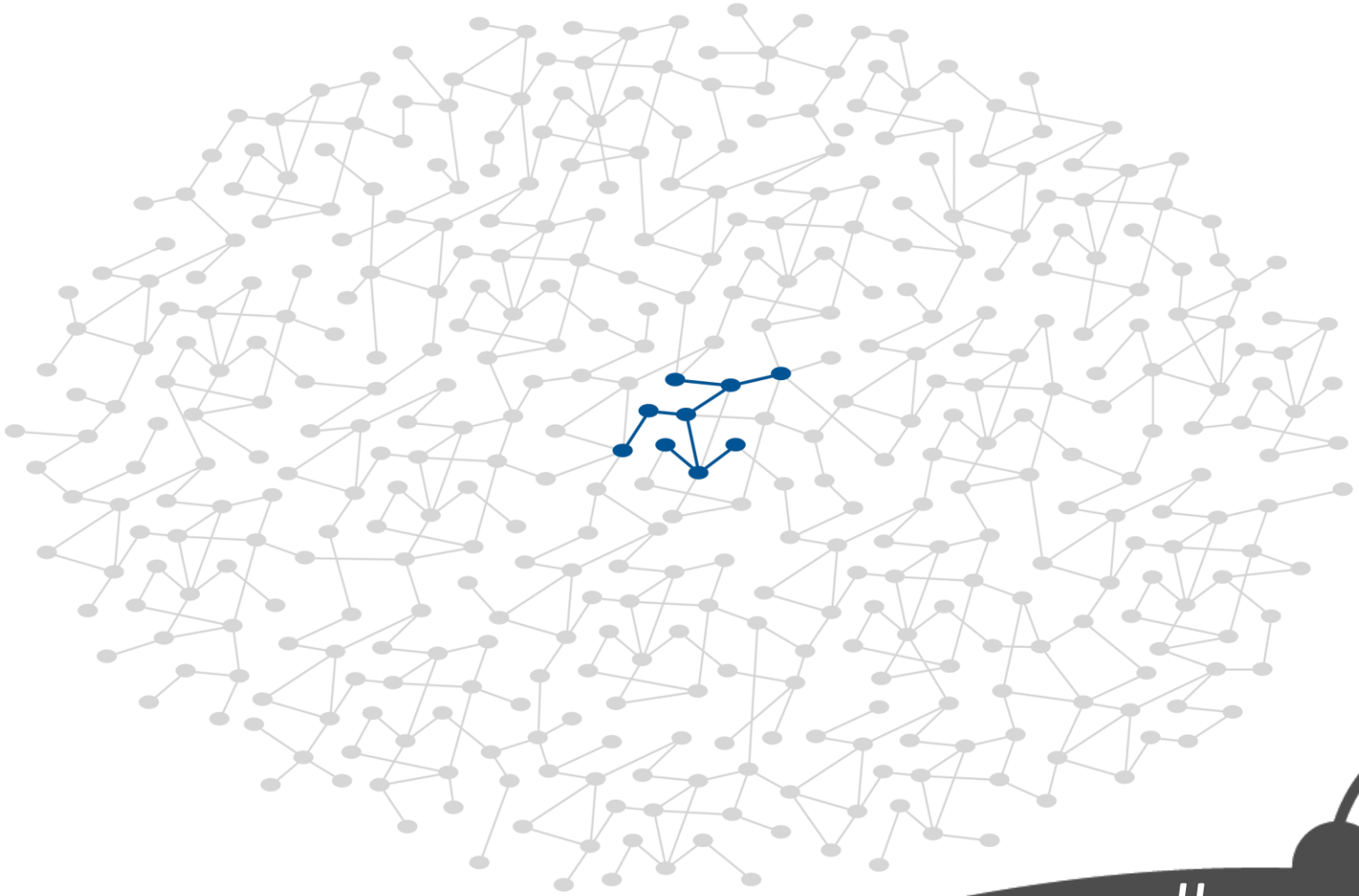
Query times proportional to size of subgraph searched



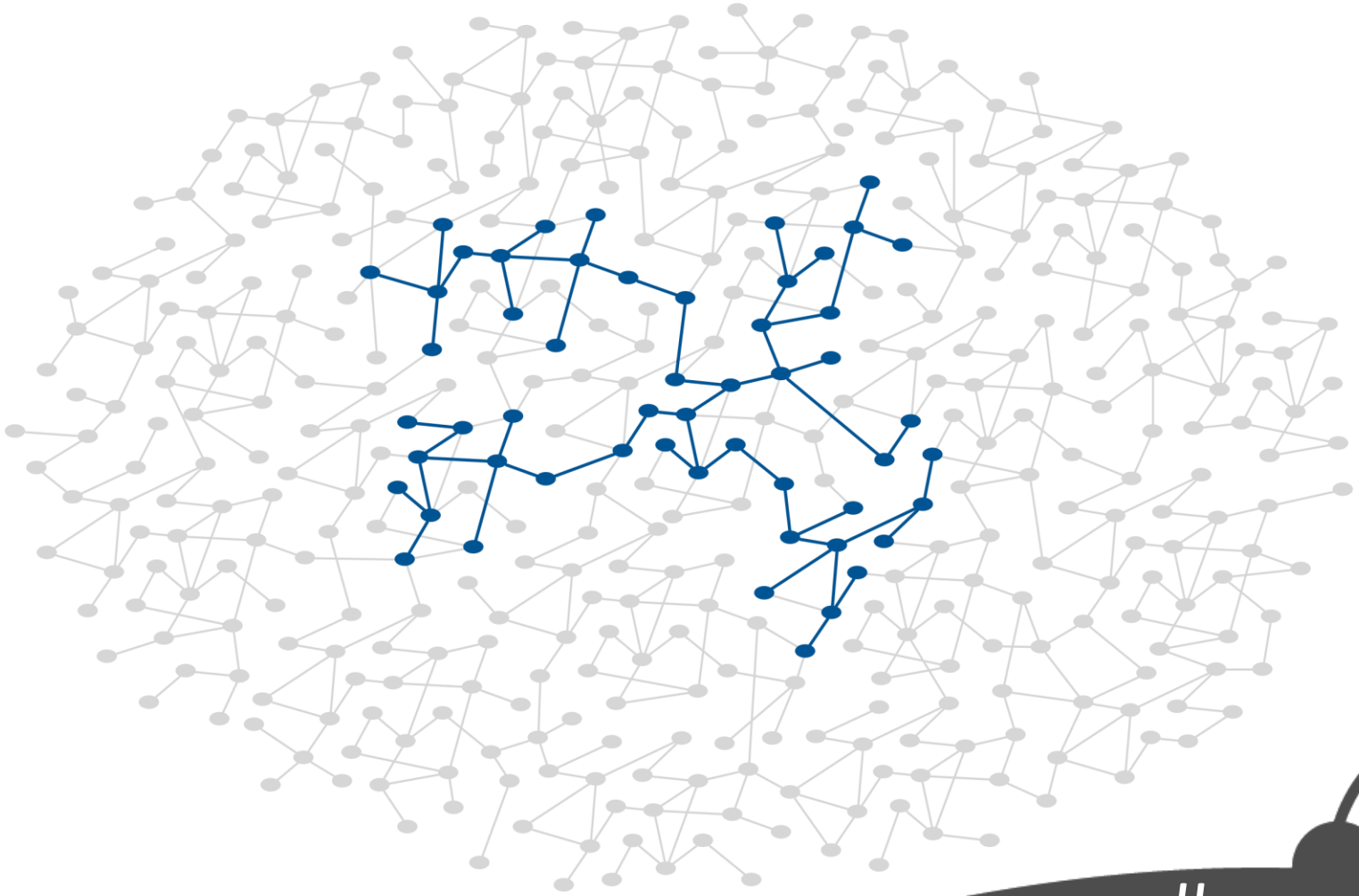
Query times proportional to size of subgraph searched



Query times remain constant …

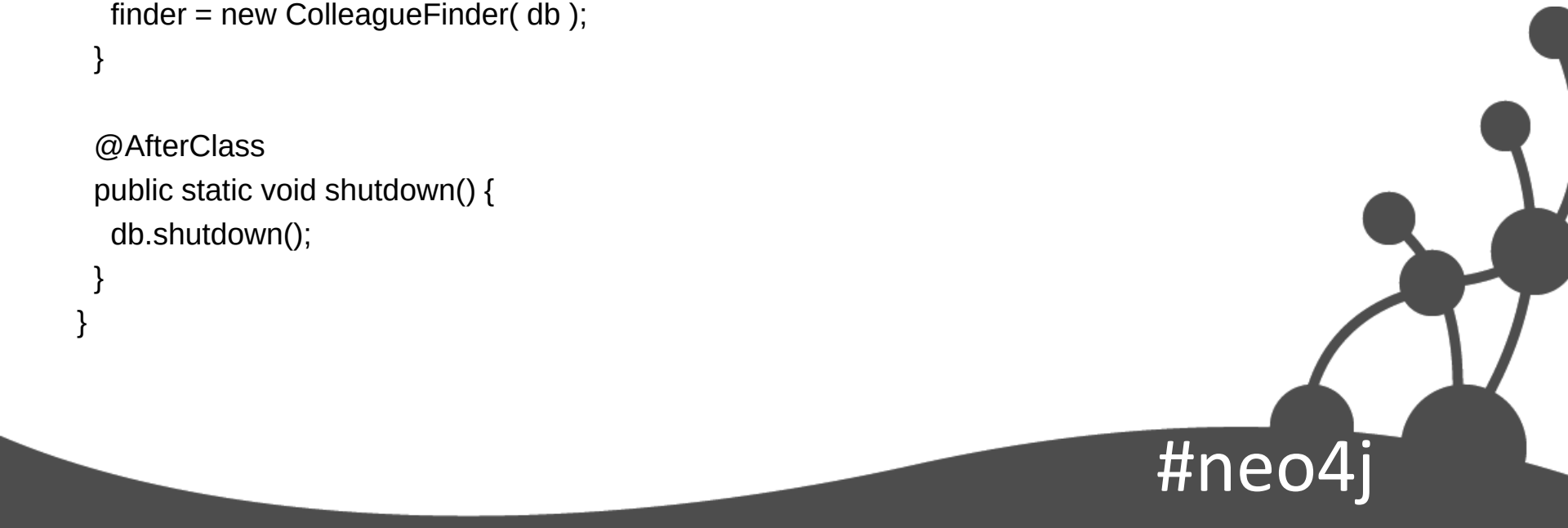


… unless subgraph searched grows



Unit test fixture

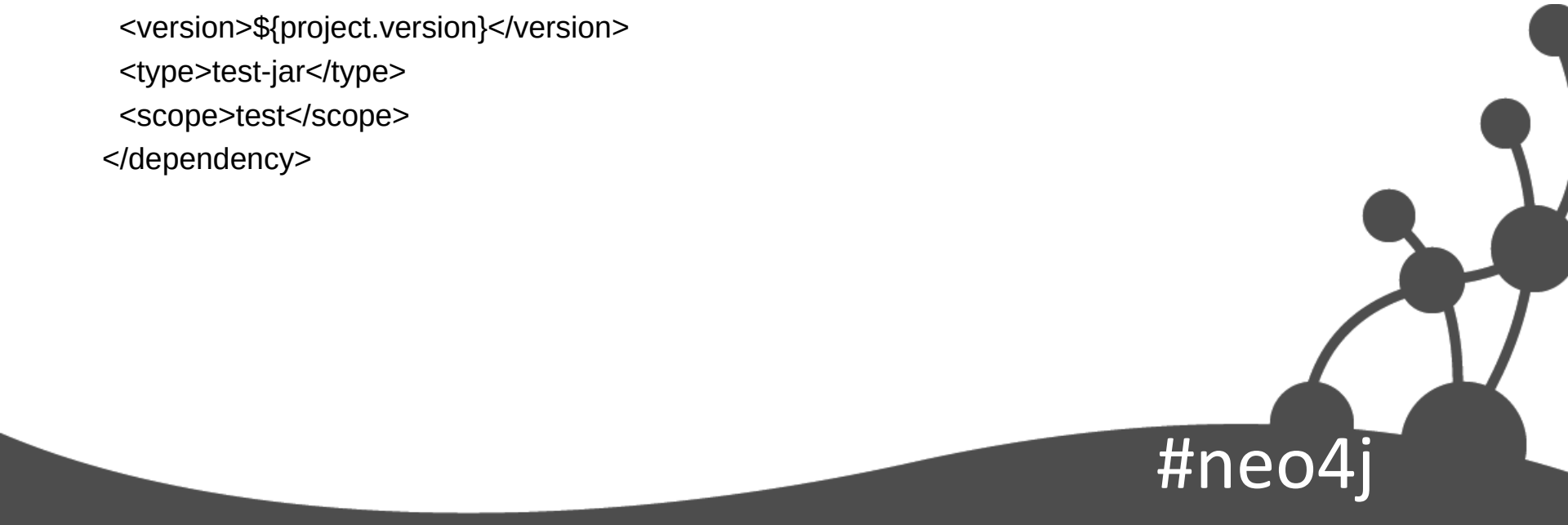
```
public class ColleagueFinderTest {

  private static GraphDatabaseService db;
  private static ColleagueFinder finder;

  @BeforeClass
  public static void init() {
    db = new TestGraphDatabaseFactory().newImpermanentDatabase();
    ExampleGraph.populate( db );
    finder = new ColleagueFinder( db );
  }

  @AfterClass
  public static void shutdown() {
    db.shutdown();
  }
}
```



ImpermanentGraphDatabase

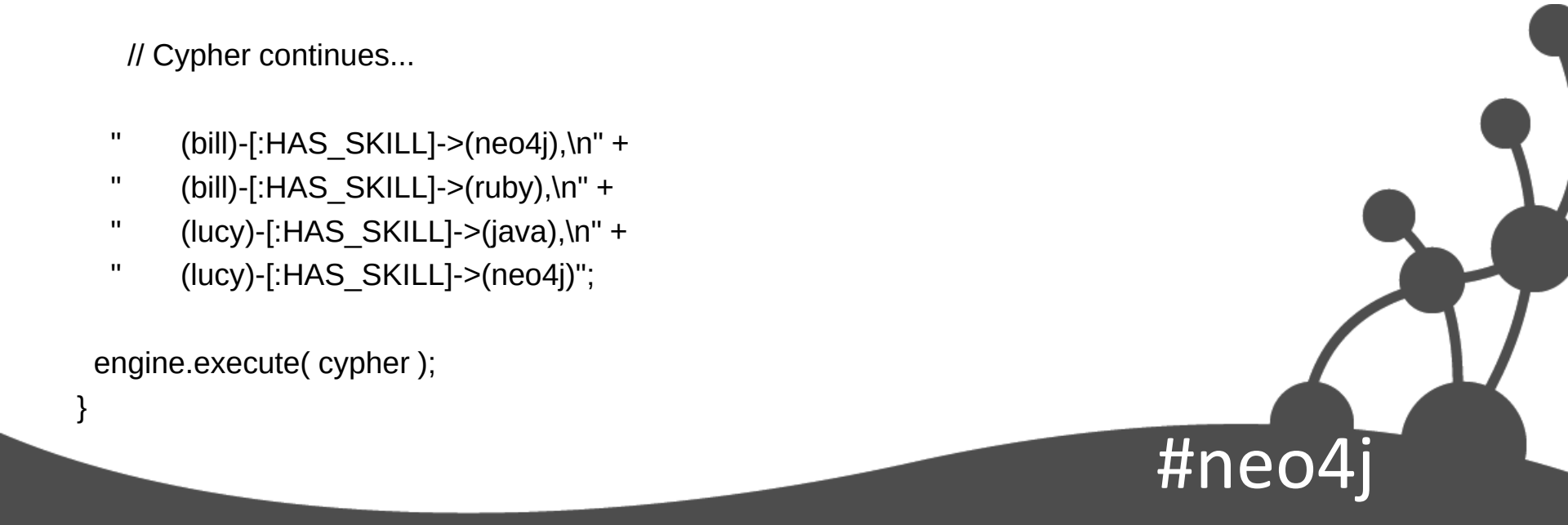
- In-memory
- For testing only

```
<dependency>
  <groupId>org.neo4j</groupId>
  <artifactId>neo4j-kernel</artifactId>
  <version>${project.version}</version>
  <type>test-jar</type>
  <scope>test</scope>
</dependency>
```



Create sample data

```
public static void populate( GraphDatabaseService db ) {

  ExecutionEngine engine = new ExecutionEngine( db );

  String cypher =
    "CREATE ian:person VALUES {name:'Ian'},\n" +
    "       bill:person VALUES {name:'Bill'},\n" +
    "       lucy:person VALUES {name:'Lucy'},\n" +
    "       acme:company VALUES {name:'Acme'},\n" +

      // Cypher continues...

    "       (bill)-[:HAS_SKILL]->(neo4j),\n" +
    "       (bill)-[:HAS_SKILL]->(ruby),\n" +
    "       (lucy)-[:HAS_SKILL]->(java),\n" +
    "       (lucy)-[:HAS_SKILL]->(neo4j)";

  engine.execute( cypher );
}
```



Unit test

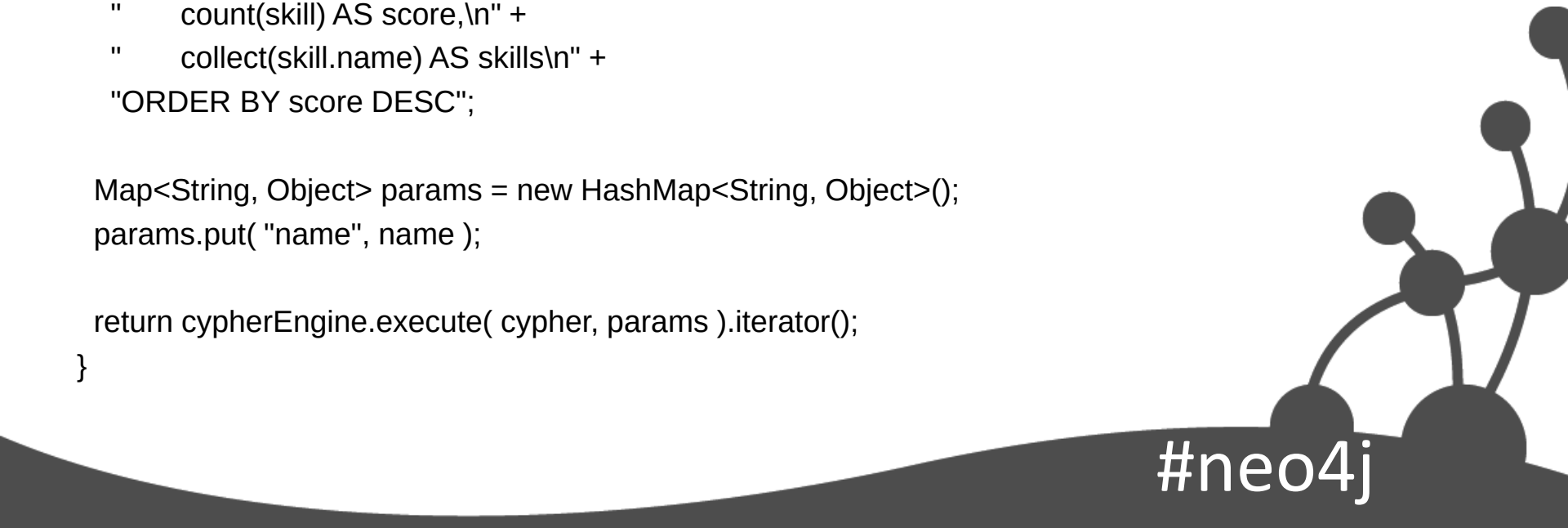
```
@Test
public void shouldFindColleaguesWithSimilarSkills() throws Exception {

  // when
  Iterator<Map<String, Object>> results = finder.findFor( "Ian" );

  // then
  assertEquals( "Lucy", results.next().get( "name" ) );
  assertEquals( "Bill", results.next().get( "name" ) );

  assertFalse( results.hasNext() );
}
```

Object under test

```
public class ColleagueFinder {

  private final ExecutionEngine cypherEngine;

  public ColleagueFinder( GraphDatabaseService db ) {
      this.cypherEngine = new ExecutionEngine( db );
  }

  public Iterator<Map<String, Object>> findFor( String name ) {
      ...
  }
}
```

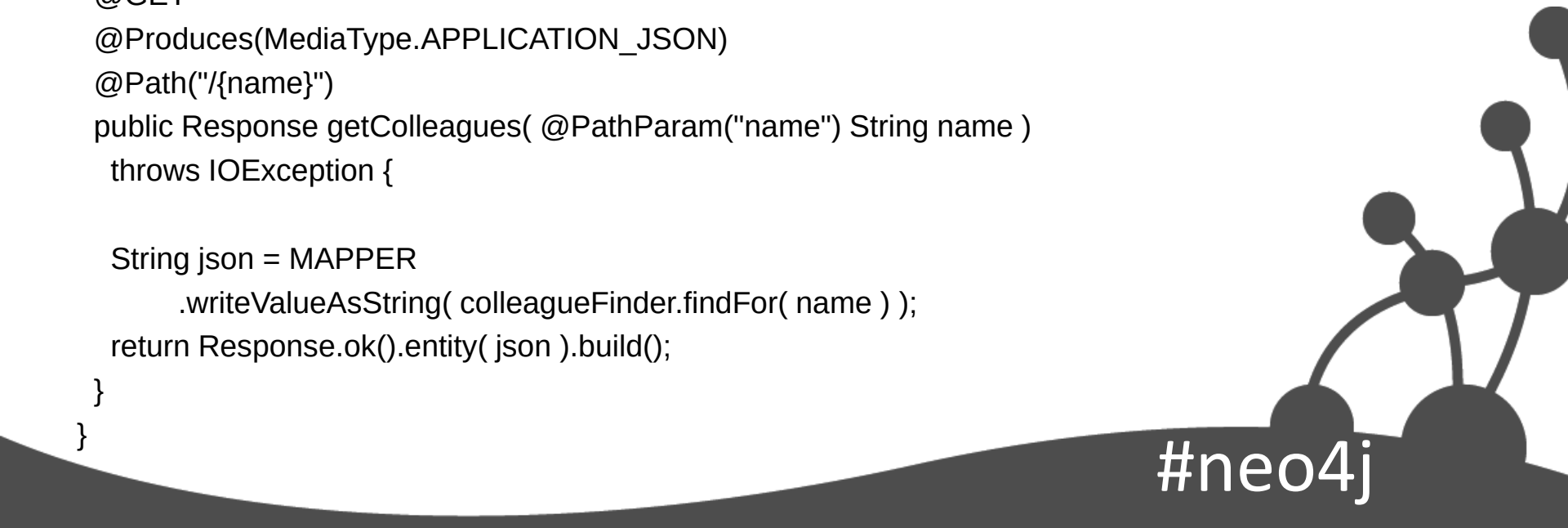
findFor() method

```
public Iterator<Map<String, Object>> findFor( String name ) {

  String cypher =
    "MATCH (me:person)-[:WORKS_FOR]->(company),\n" +
    "      (me)-[:HAS_SKILL]->(skill),\n" +
    "      (colleague)-[:WORKS_FOR]->(company),\n" +
    "      (colleague)-[:HAS_SKILL]->(skill)\n" +
    "WHERE  me.name = {name}\n" +
    "RETURN colleague.name AS name,\n" +
    "       count(skill) AS score,\n" +
    "       collect(skill.name) AS skills\n" +
    "ORDER BY score DESC";

  Map<String, Object> params = new HashMap<String, Object>();
  params.put( "name", name );

  return cypherEngine.execute( cypher, params ).iterator();
}
```



Unmanaged extension

```
@Path("/similar-skills")
public class ColleagueFinderExtension {
  private static final ObjectMapper MAPPER = new ObjectMapper();
  private final ColleagueFinder colleagueFinder;

  public ColleagueFinderExtension( @Context GraphDatabaseService db ) {
    this.colleagueFinder = new ColleagueFinder( db );
  }

  @GET
  @Produces(MediaType.APPLICATION_JSON)
  @Path("/{name}")
  public Response getColleagues( @PathParam("name") String name )
    throws IOException {

    String json = MAPPER
          .writeValueAsString( colleagueFinder.findFor( name ) );
    return Response.ok().entity( json ).build();
  }
}
```

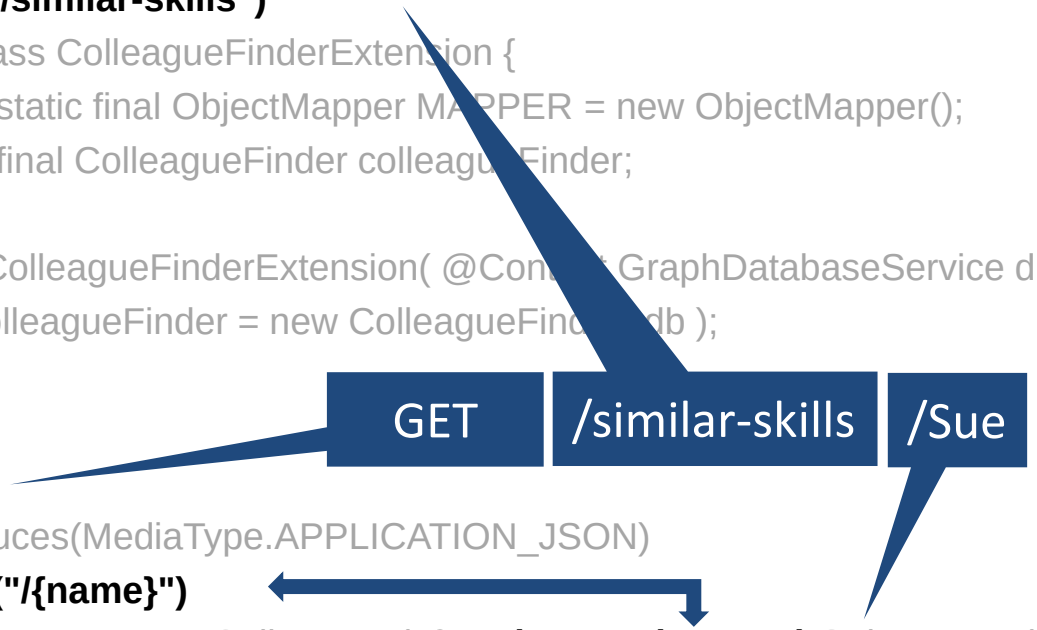


JAX-RS annotations

```
@Path("/similar-skills")
public class ColleagueFinderExtension {
  private static final ObjectMapper MAPPER = new ObjectMapper();
  private final ColleagueFinder colleagueFinder;

  public ColleagueFinderExtension( @Context GraphDatabaseService db ) {
    this.colleagueFinder = new ColleagueFinder( db );
  }

  @GET
  @Produces(MediaType.APPLICATION_JSON)
  @Path("/{name}")
  public Response getColleagues( @PathParam("name") String name )
    throws IOException {

    String json = MAPPER
          .writeValueAsString( colleagueFinder.findFor( name ) );
    return Response.ok().entity( json ).build();
  }
}
```

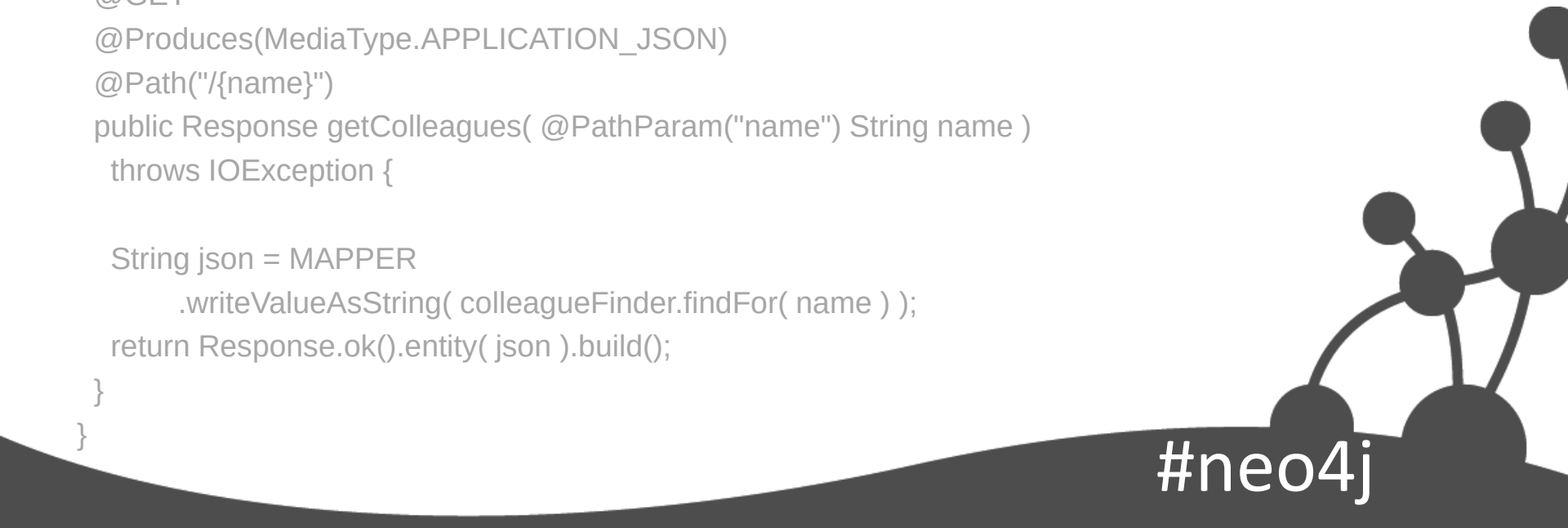
Map HTTP request to object+method

```
@Path("/similar-skills")
public class ColleagueFinderExtension {
  private static final ObjectMapper MAPPER = new ObjectMapper();
  private final ColleagueFinder colleagueFinder;

  public ColleagueFinderExtension( @Context GraphDatabaseService db ) {
    this.colleagueFinder = new ColleagueFinder( db );
  }

  @GET
  @Produces(MediaType.APPLICATION_JSON)
  @Path("/{name}")
  public Response getColleagues( @PathParam("name") String name )
    throws IOException {

    String json = MAPPER
          .writeValueAsString( colleagueFinder.findFor( name ) );
    return Response.ok().entity( json ).build();
  }
}
```



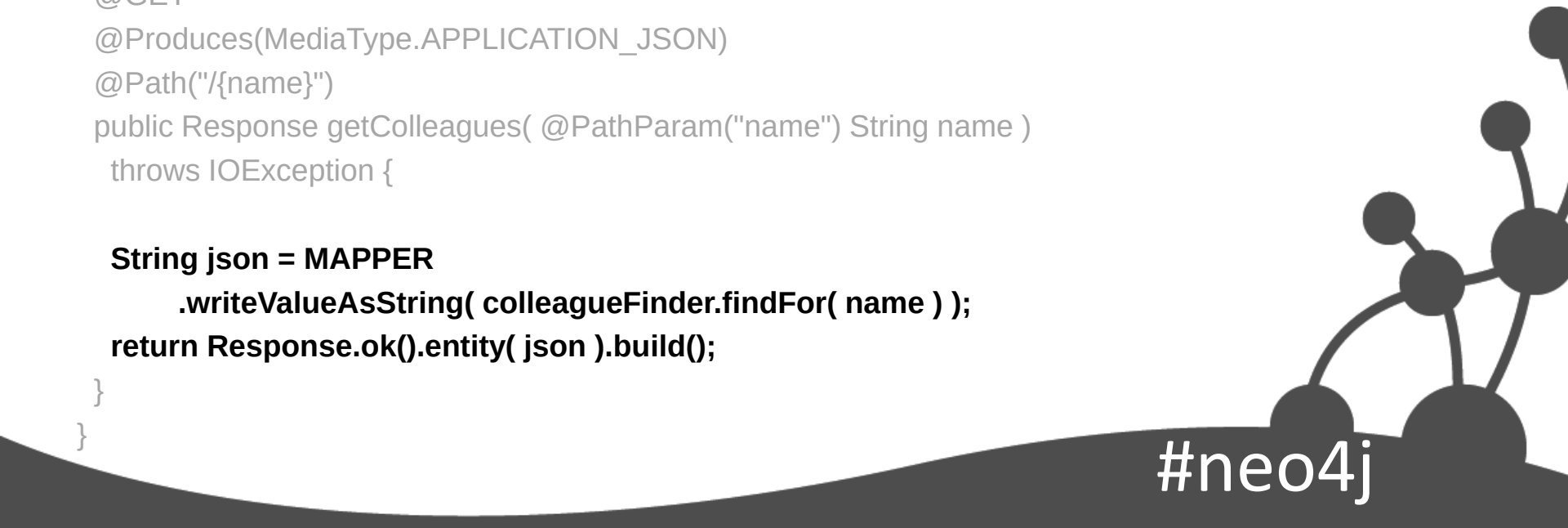
Database injected by server

```
@Path("/similar-skills")
public class ColleagueFinderExtension {
  private static final ObjectMapper MAPPER = new ObjectMapper();
  private final ColleagueFinder colleagueFinder;

  public ColleagueFinderExtension( @Context GraphDatabaseService db ) {
    this.colleagueFinder = new ColleagueFinder( db );
  }

  @GET
  @Produces(MediaType.APPLICATION_JSON)
  @Path("/{name}")
  public Response getColleagues( @PathParam("name") String name )
    throws IOException {

    String json = MAPPER
          .writeValueAsString( colleagueFinder.findFor( name ) );
    return Response.ok().entity( json ).build();
  }
}
```



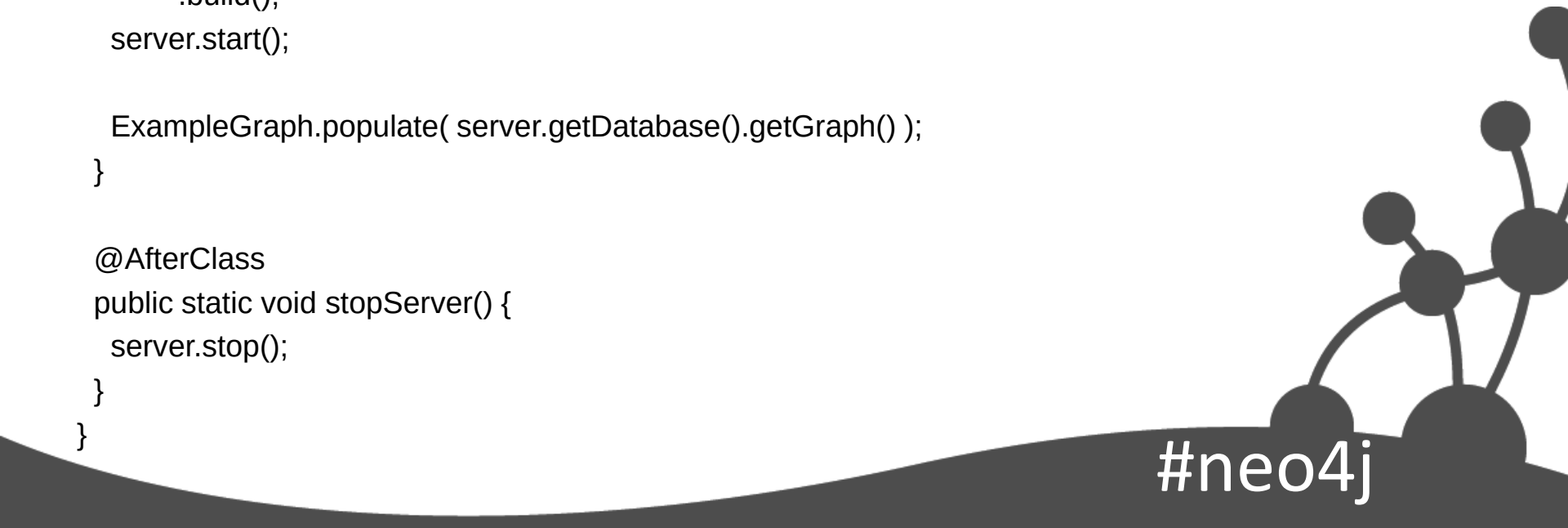
Generate and format response

```
@Path("/similar-skills")
public class ColleagueFinderExtension {
  private static final ObjectMapper MAPPER = new ObjectMapper();
  private final ColleagueFinder colleagueFinder;

  public ColleagueFinderExtension( @Context GraphDatabaseService db ) {
    this.colleagueFinder = new ColleagueFinder( db );
  }

  @GET
  @Produces(MediaType.APPLICATION_JSON)
  @Path("/{name}")
  public Response getColleagues( @PathParam("name") String name )
    throws IOException {

    String json = MAPPER
          .writeValueAsString( colleagueFinder.findFor( name ) );
    return Response.ok().entity( json ).build();
  }
}
```



Extension test fixture

```
public class ColleagueFinderExtensionTest {
  private static CommunityNeoServer server;

  @BeforeClass
  public static void startServer() throws IOException
  {
    server = CommunityServerBuilder.server()
            .withThirdPartyJaxRsPackage(
                    "org.neo4j.good_practices", "/colleagues" )
            .build();
    server.start();

    ExampleGraph.populate( server.getDatabase().getGraph() );
  }

  @AfterClass
  public static void stopServer() {
    server.stop();
  }
}
```



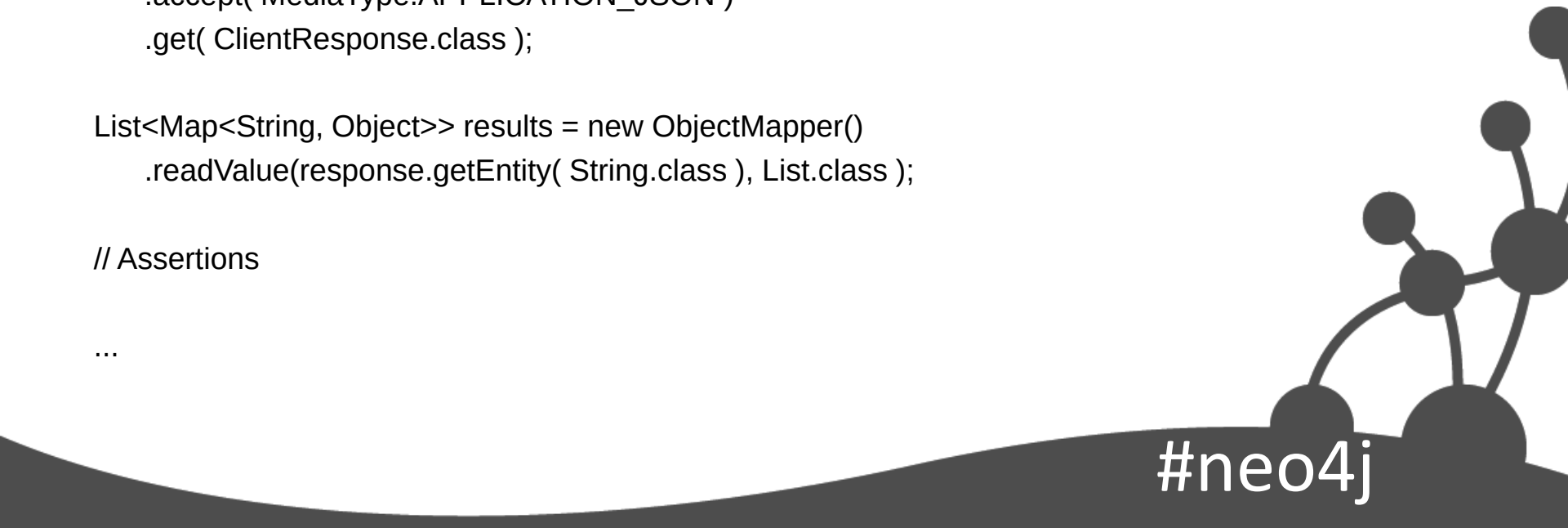
CommunityServerBuilder

- Programmatic configuration

```
<dependency>
  <groupId>org.neo4j.app</groupId>
  <artifactId>neo4j-server</artifactId>
  <version>${project.version}</version>
  <type>test-jar</type>
</dependency>
```

Testing extensions

```
@Test
public void shouldReturnColleaguesWithSimilarSkills() throws Exception {

  Client client = Client.create( new DefaultClientConfig() );

  WebResource resource = client
        .resource( "http://localhost:7474/colleagues/similar-skills/Ian" );

  ClientResponse response = resource
        .accept( MediaType.APPLICATION_JSON )
        .get( ClientResponse.class );

  List<Map<String, Object>> results = new ObjectMapper()
        .readValue(response.getEntity( String.class ), List.class );

  // Assertions

  ...
```



Testing extensions (continued)

```
  ...

  assertEquals( 200, response.getStatus() );
  assertEquals( MediaType.APPLICATION_JSON,
                response.getHeaders().get( "Content-Type" ).get( 0 ) );

  assertEquals( "Lucy", results.get( 0 ).get( "name" ) );
  assertThat( (Iterable<String>) results.get( 0 ).get( "skills" ),
              hasItems( "Java", "Neo4j" ) );
}
```

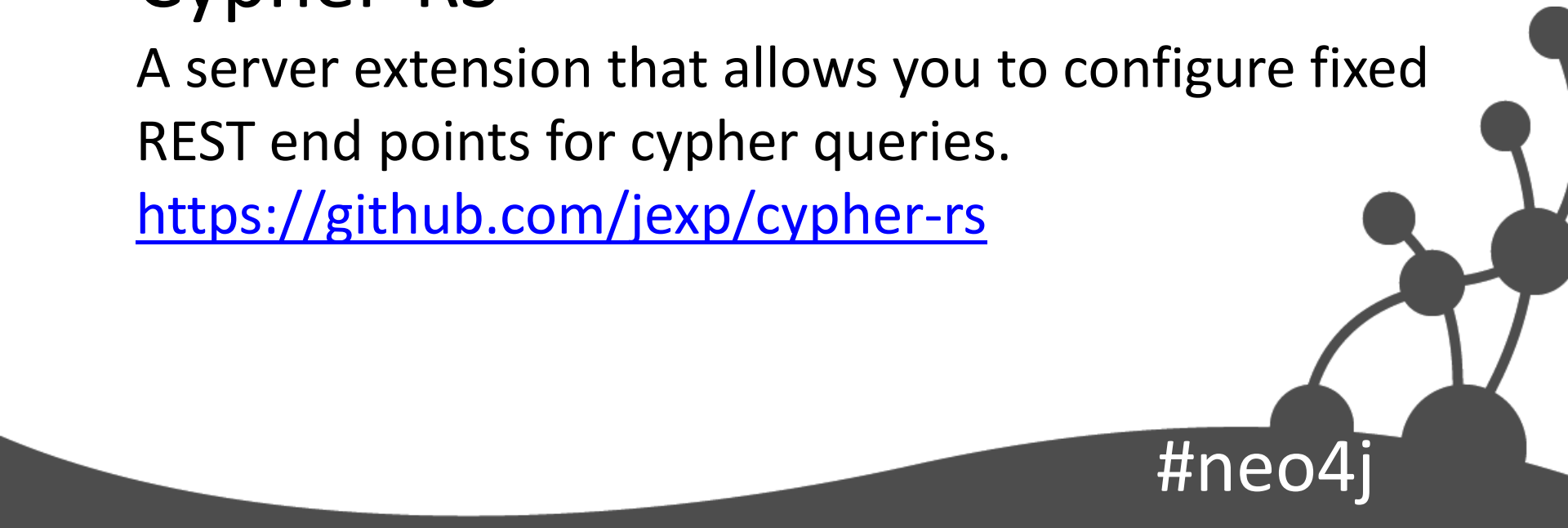
Examples to follow

- Neo4j Good Practices

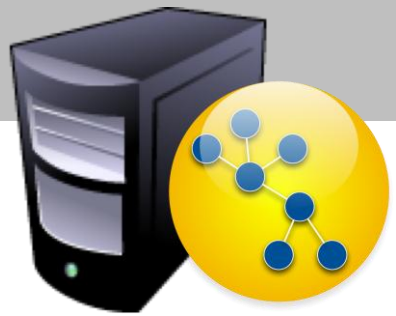
  Accompanying code for some of the examples in this talk.https://github.com/iansrobinson/neo4j-good-practices

- Cypher-RS

  A server extension that allows you to configure fixed REST end points for cypher queries.
  https://github.com/jexp/cypher-rs



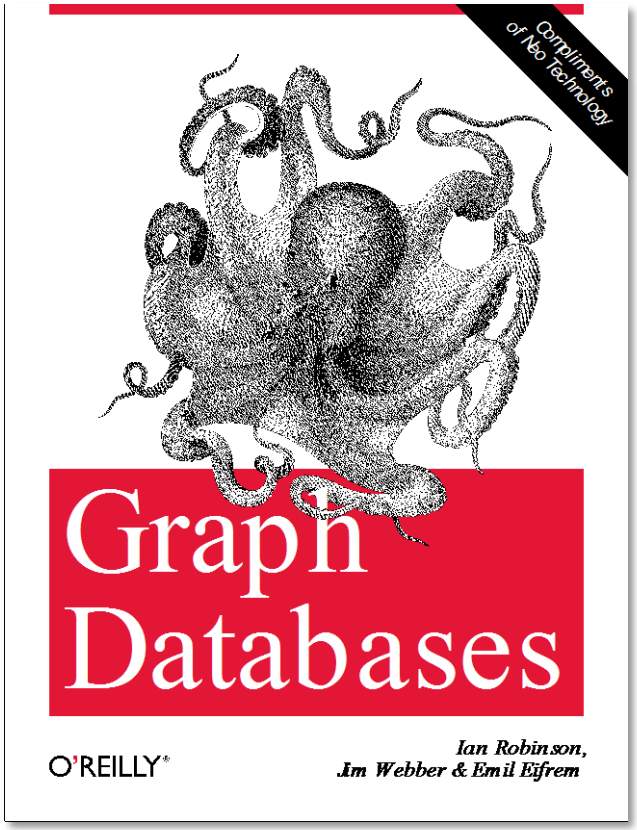
Learning More



#neo4j

Graph Databases Book

www.graphdatabases.com



#neo4j

Neo4j Manual Modeling Examples

Google “neo4j modeling manual”

Chapter 7. Data Modeling Examples

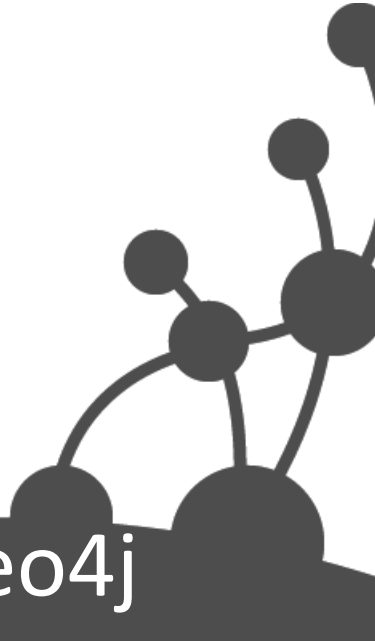
The Neo4j Manual > Tutorials > Data Modeling Examples

Table of Contents

7.1. User roles in graphs
7.2. ACL structures in graphs
7.3. Linked Lists
7.4. Hyperedges
7.5. Basic friend finding based on social neighborhood
7.6. Co-favorited places
7.7. Find people based on similar favorites
7.8. Find people based on mutual friends and groups
7.9. Find friends based on similar tagging
7.10. Multirelational (social) graphs
7.11. Implementing newsfeeds in a graph
7.12. Boosting recommendation results
7.13. Calculating the clustering coefficient of a network
7.14. Pretty graphs
7.15. A multilevel indexing structure (path tree)
7.16. Complex similarity computations
7.17. The Graphity activity stream model

The following chapters contain simplified examples of how different domains can be modeled using Neo4j. The aim is not to give full examples, but to suggest possible ways to think using nodes, relationships, graph patterns and data locality in traversals.

The examples use Cypher queries a lot, read Part III, “Cypher Query Language” for more information.



Cypher Modeling Challenge

- What is the average taxi time at each airport for both departures and arrivals?
- What is the leading cause of departure delays at each airport?
- How many outbound flights were cancelled at each airport?

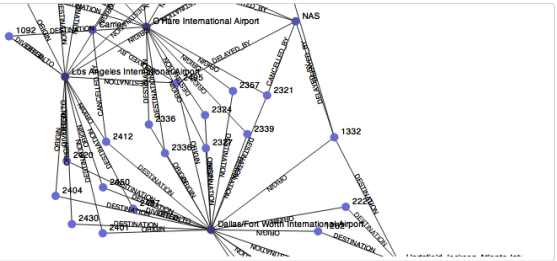
Or more specific questions such as:

- Which flights from Los Angeles (LAX) to Chicago (ORD) were delayed for more than 10 minutes due to late arrivals?
- How does seasonality affect departure taxi times at Chicago's O'Hare International Airport (ORD)?
- What is the standard deviation of arrival taxi times at Dallas/Fort Worth (DFW)?

These are the types of questions airline carriers are asking when attempting to construct efficient flight plans for their customers. After initializing the data (Query 1), we can start answering these questions and drawing insights.

Query 1 — this query has been used to initialize the console

Test run OK



What is the average taxi time at each airport for both departures and arrivals?

A flight planner will want to take into consideration how long it takes on average for a plane to travel from its gate to the runway, or vice versa, at a given airport. The consequences for leaving customers sitting on a tarmac for too long can range from a few angry letters to a PR nightmare.

Query 2

```
MATCH (a)<-[q:ORIGIN]-(), ()-[r:DESTINATION|DIVERTED_TO]->(a)
WITH a, ROUND(AVG(q.taxi_time)) AS avg1, ROUND(AVG(r.taxi_time)) AS avg2
RETURN a.name AS Airport, avg1 AS `Average Departure Taxi Time`, avg2 AS `Average Arrival Taxi Time`
```

Test run OK

Airport	Average Departure Taxi Time	Average Arrival Taxi Time
Los Angeles International Airport	26	10
Hartsfield-Jackson Atlanta International Airport	18	9
Dallas/Fort Worth International Airport	13	8
O'Hare International Airport	13	6

What is the leading cause of departure delays at each airport?

Are the delays at a given airport mostly out of one's control (weather delays) or are the delays mostly preventable (carrier delays)? A flight planner would be interested to learn which of these types of delays are most prevalent at each of its airports.

GraphGist Challenge Submissions

The GraphGist Challenge was run during September 2013 and had the following submissions:

- Holiday Resorts by Raju Rama Krishna
- Sports League by @yaravind
- Learning Graph by jotomo
- IKEA furniture Graph by @rvanbruggen
- Enterprise Content Management Graph by @PieterJanVA
- US Flights & Airports by @_nicolemargaret
- Chess Games and Positions by @wefreema
- Why JIRA should use Neo4J by @PieterJanVA
- Mystery Science Theater 3000 Actors and Characters by @virtualswede
- Breaking Bad characters are interested in some products, let's see which are by @fforbeck
- Ditching Grandma - Graphy Accounting by @ShaunDaley1
- MotoGp Graph Gist by @ricshouse
- European Royalty by @frant_hartm
- Product Catalog by @yaravind
- A Simple Meta-Data Model by @perival

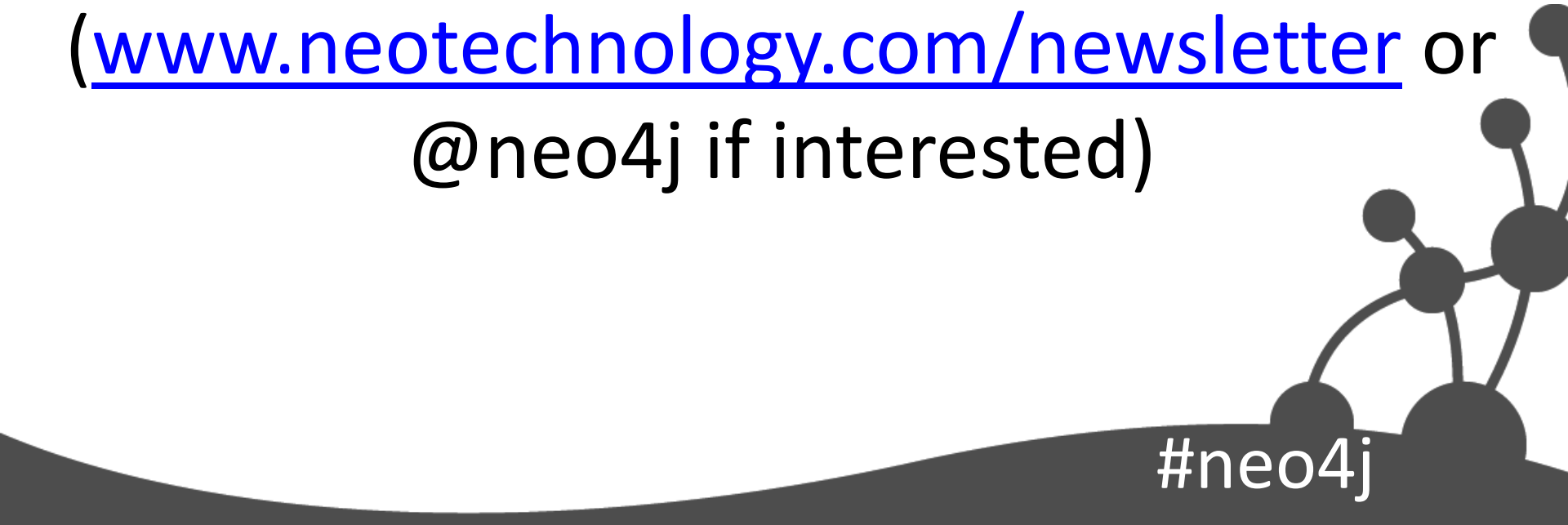
https://github.com/neo4j-contrib/graphgist/wiki

#neo4j

Modeling Webinar

Coming soon…

(www.neotechnology.com/newsletter or @neo4j if interested)



Modeling Workshop

Coming soon…

(rik@neotechnology.com if interested)

And that’s it

@markhneedham

#neo4j