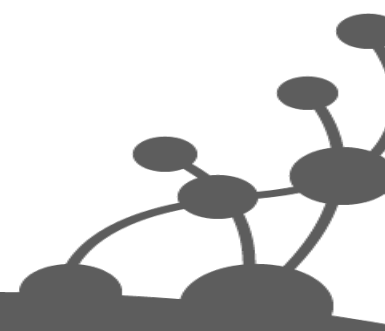
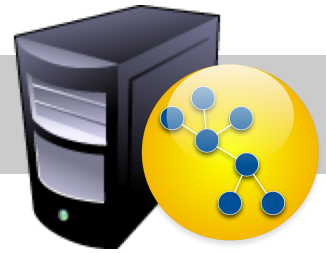


Test-Driven Data Modeling With Graphs

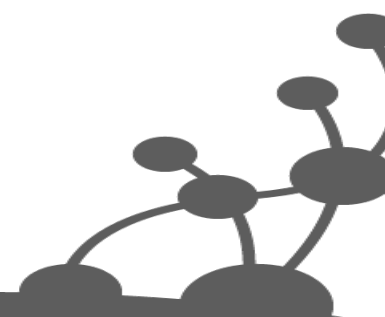
Twitter: @ianSrobinson
#neo4j



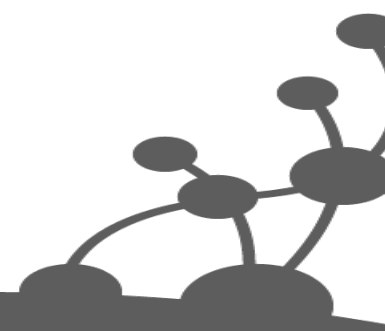
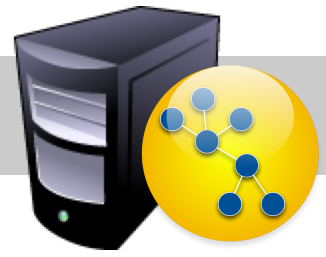
Outline



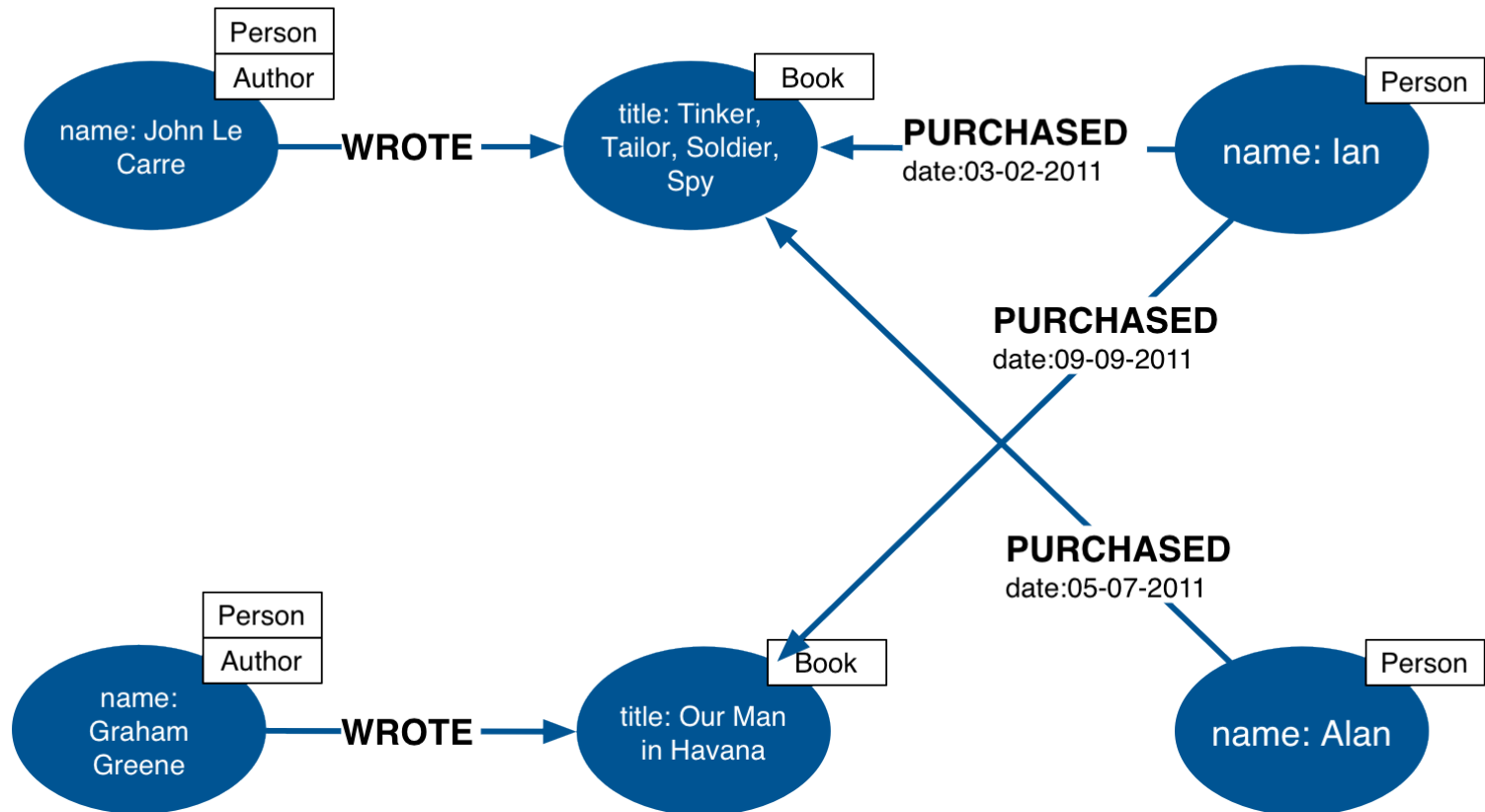
- Data modeling with graphs
- Neo4j application architecture options
- Testing your data model



Graph data modeling

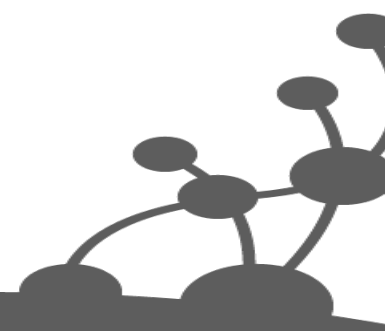
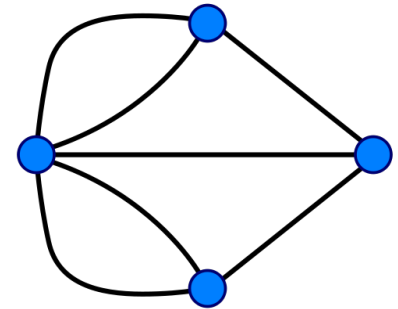
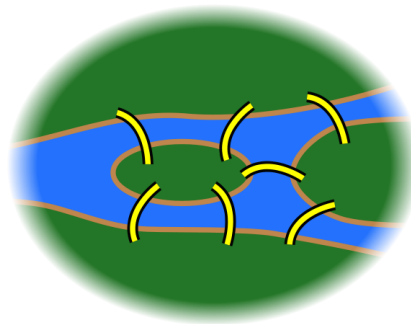
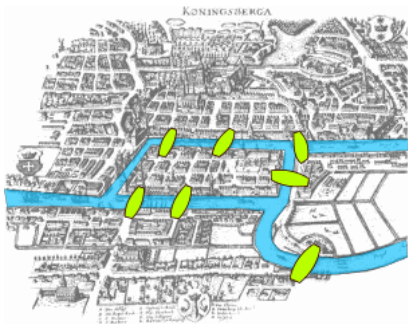


Labeled Property Graph

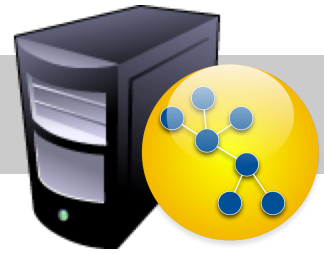


Models

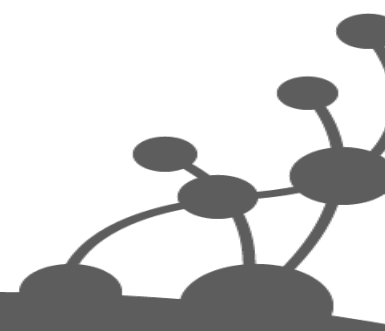
Purposeful abstraction of a domain designed to satisfy particular application/end-user goals



Example Application



- Knowledge management
 - People, companies, skills
 - Cross organizational
- Find my professional social network
 - Exchange knowledge
 - Interest groups
 - Help
 - Staff projects

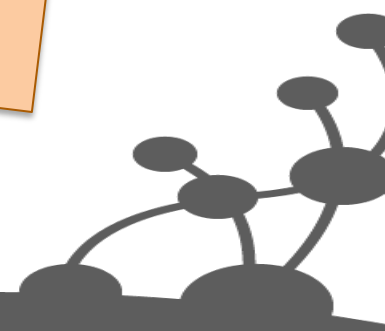


Application/End-User Goals

As an employee

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

So that we can exchange knowledge



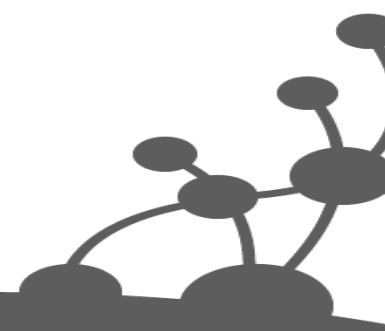
Questions To Ask of the Domain

As an employee

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

So that we can exchange knowledge

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



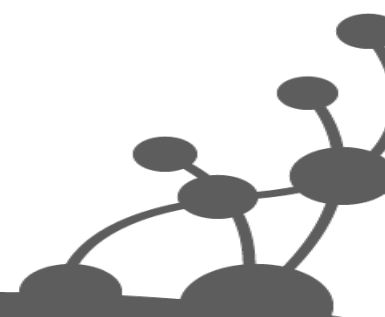
Identify Entities

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

Person

Company

Skill

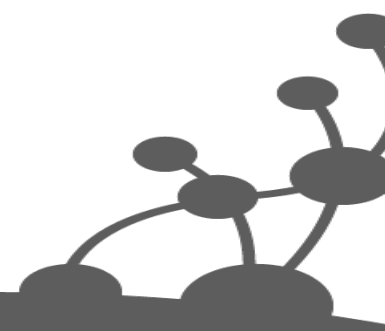


Identify Relationships Between Entities

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

Relationship

Person WORKS_FOR Company

Person HAS_SKILL Skill

Label



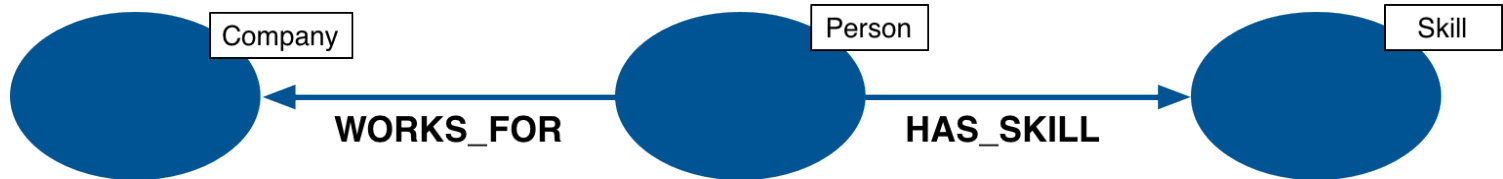
```
(:Person)-[:WORKS_FOR]->(Company),  
(:Person)-[:HAS_SKILL]->(Skill)
```

Consolidate Paths

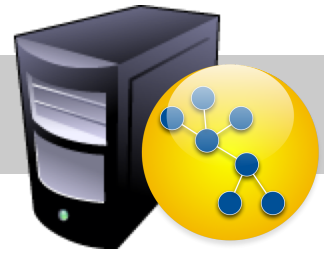
`(:Person)-[:WORKS_FOR]->(:Company),`
`(:Person)-[:HAS_SKILL]->(:Skill)`



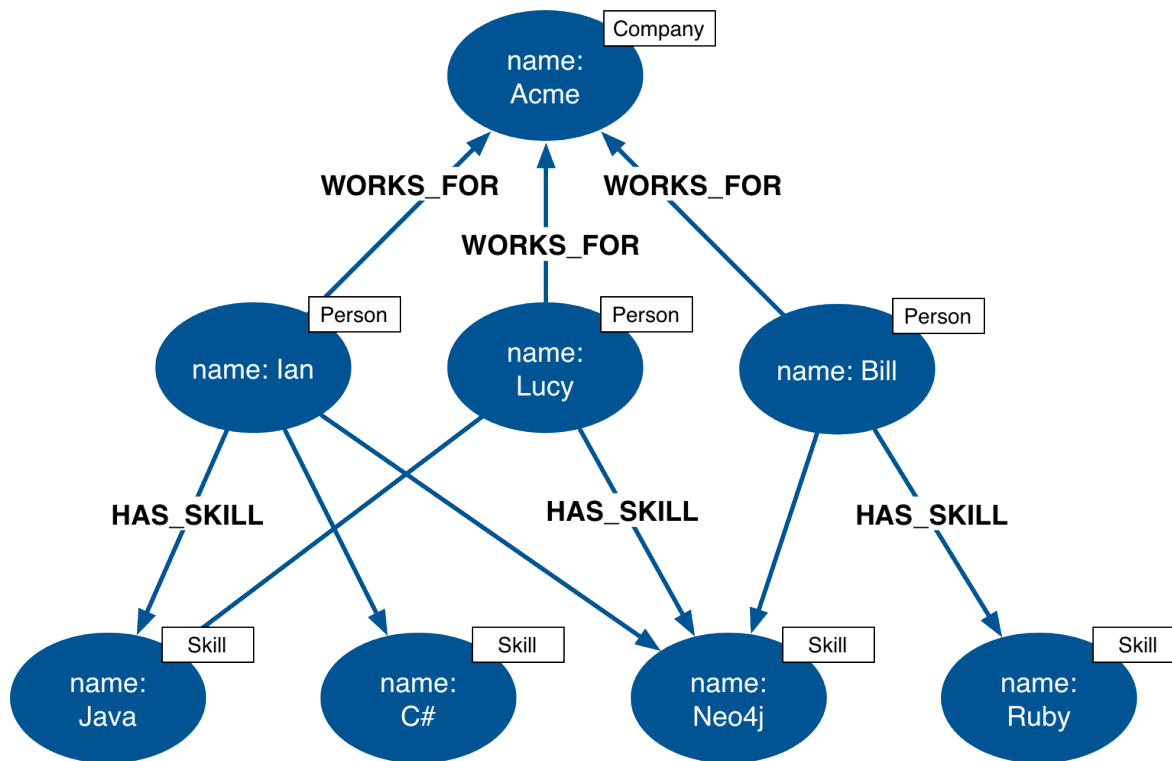
`(:Company)<-[:WORKS_FOR]-(:Person)-[:HAS_SKILL]->(:Skill)`



Candidate Data Model

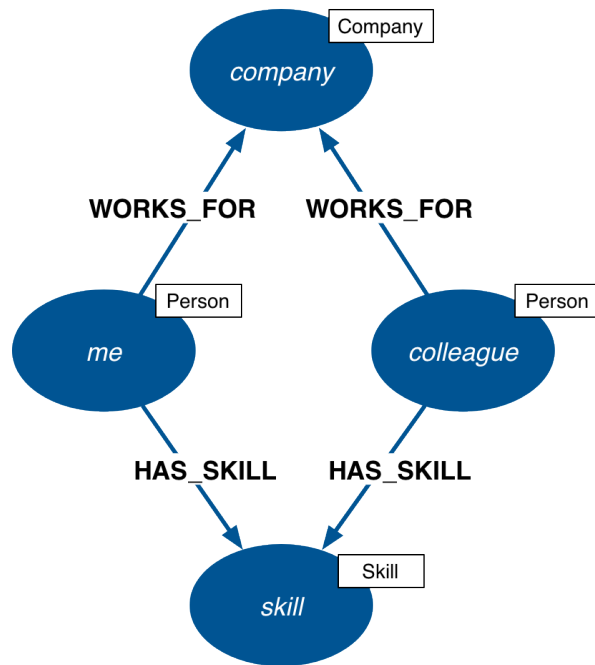


$(:Company) \leftarrow [:WORKS_FOR] - (:Person) - [:HAS_SKILL] \rightarrow (:Skill)$

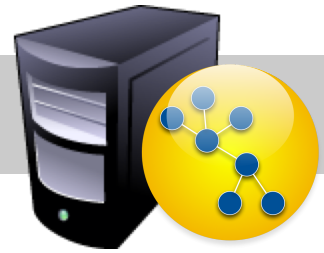


Express Question as Graph Pattern

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

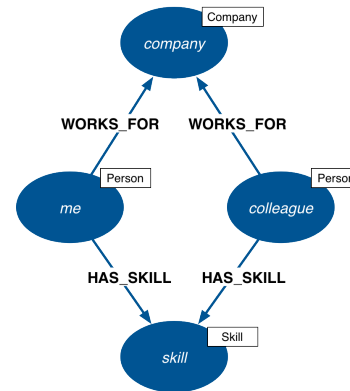


Cypher Query



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

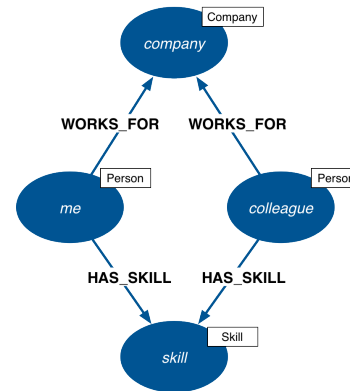
```
MATCH (company)<-[:WORKS_FOR]-(me:Person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```



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
```

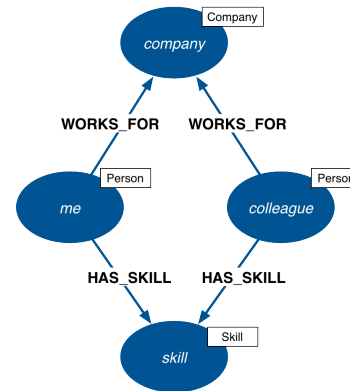


Anchor Pattern in Graph

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 DESC
```

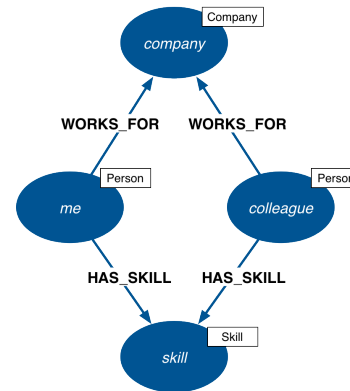
Search nodes labeled
'Person', matching on
'name' property



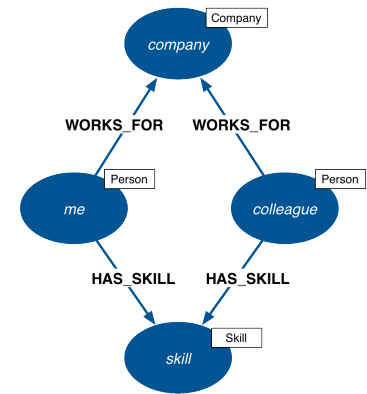
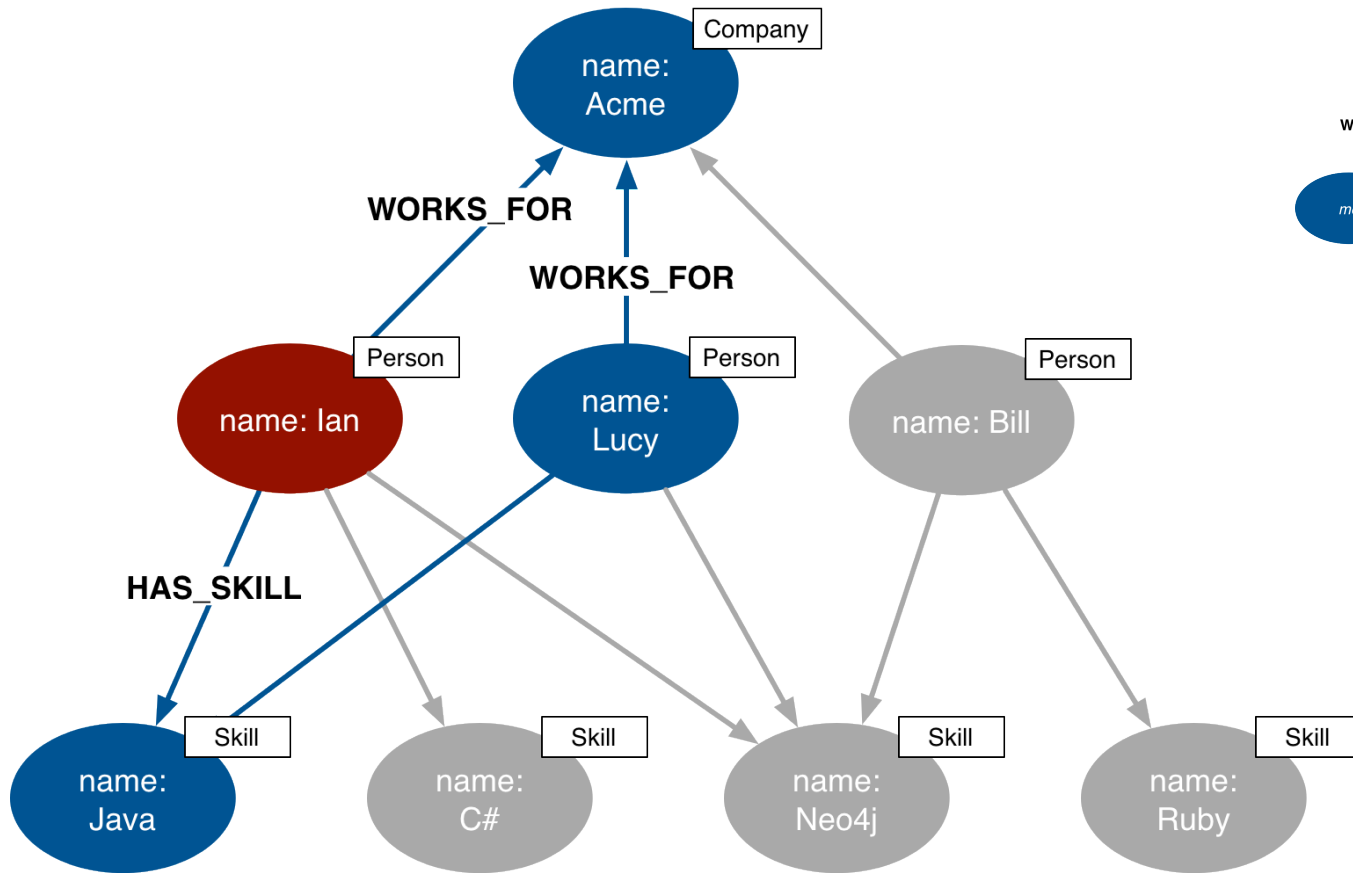
Create Projection of Results

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

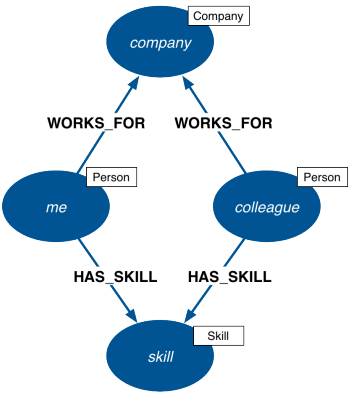
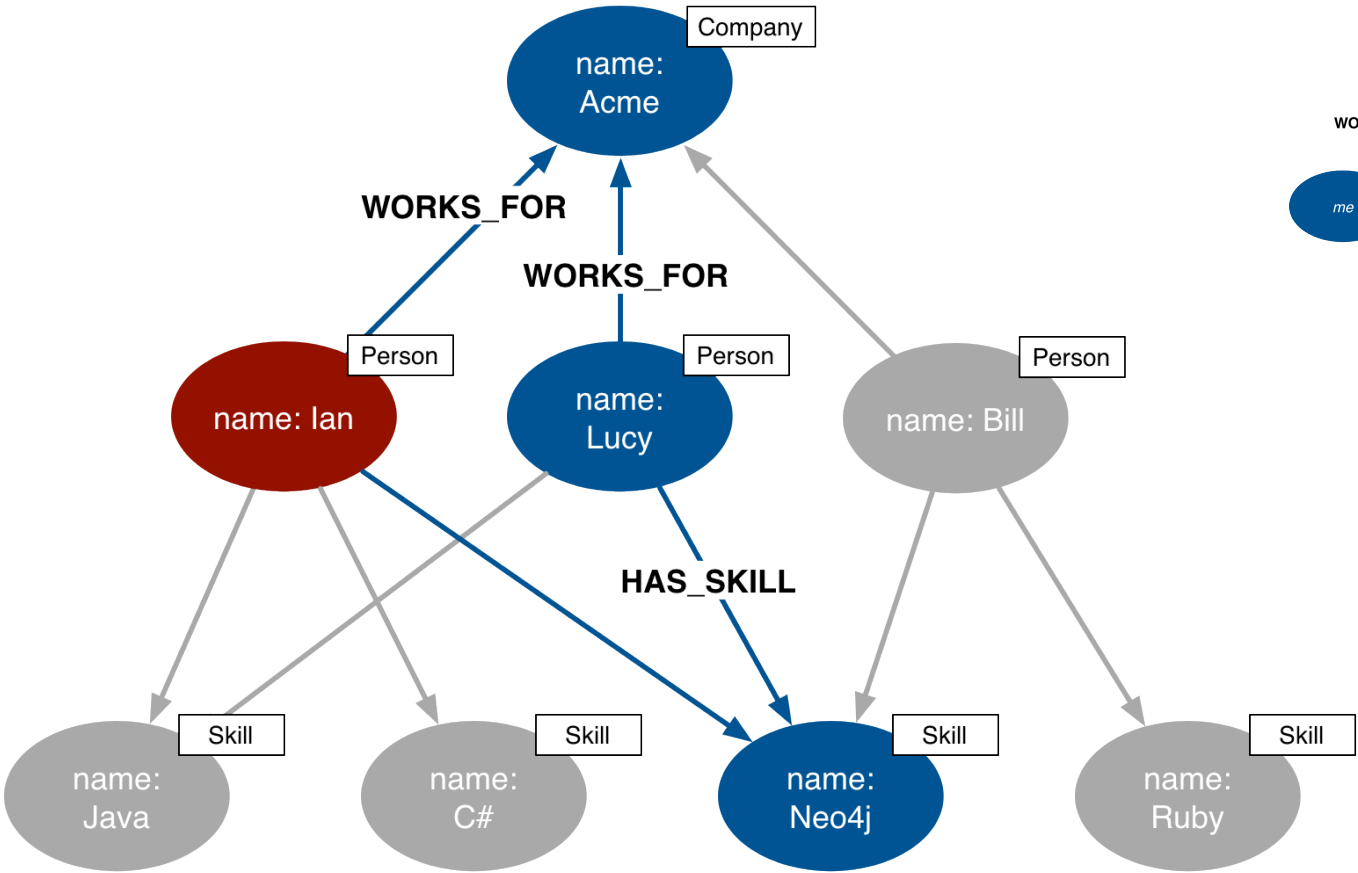
```
MATCH (company)<-[:WORKS_FOR]-(me:Person)-[:HAS_SKILL]->(skill),  
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)  
WHERE me.name = {name}  
RETURN colleague.name AS name,  
       count(skill) AS score,  
       collect(skill.name) AS skills  
ORDER BY score DESC
```



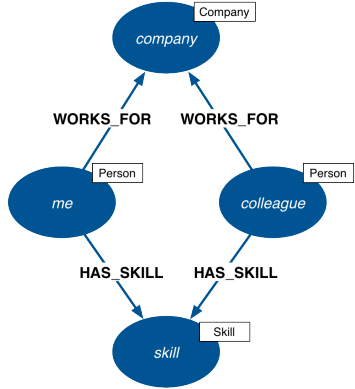
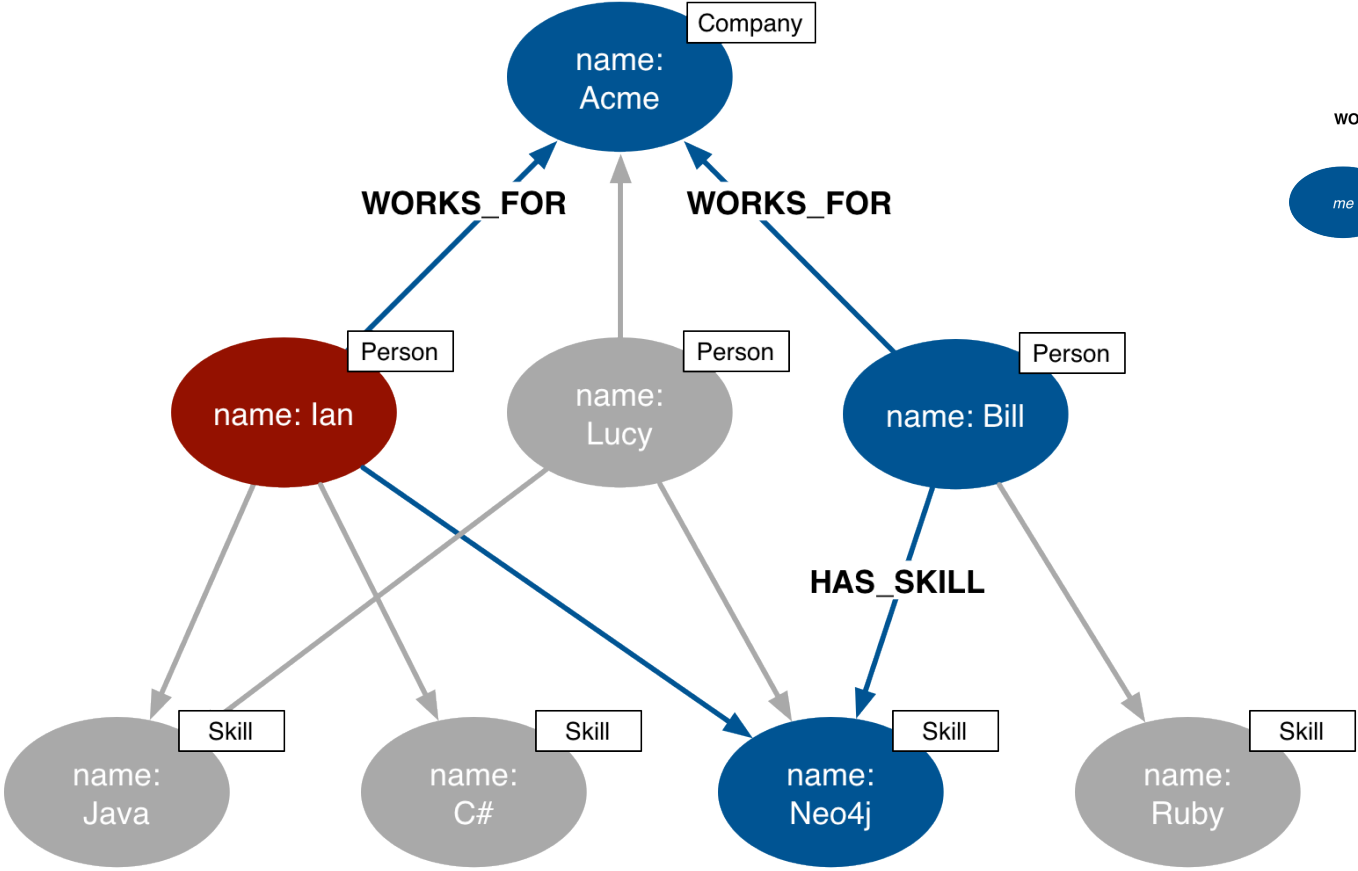
First Match



Second Match



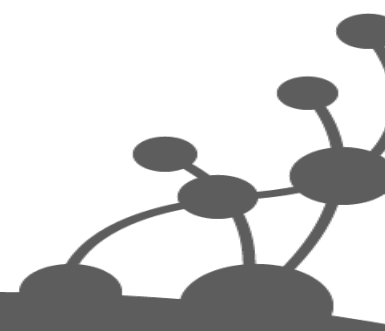
Third Match



Running the Query

name	score	skills
"Lucy"	2	["Java", "Neo4j"]
"Bill"	1	["Neo4j"]

2 rows



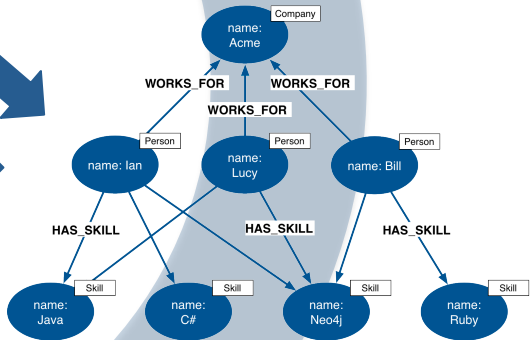
From User Story to Model and Query

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
```

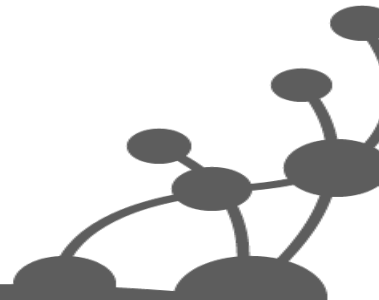
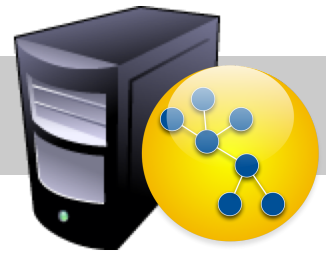
Which people, who work for the same company as me, have similar skills to me?

Person WORKS_FOR Company
Person HAS_SKILL Skill

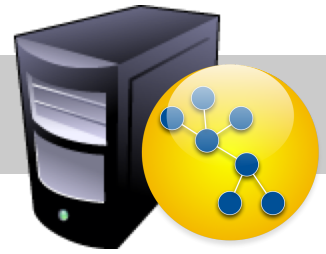


```
(:Company)-[:WORKS_FOR]-(:Person)-[:HAS_SKILL]->(:Skill)
```

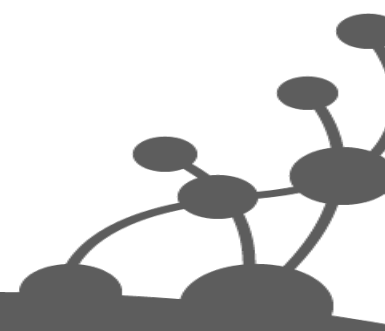
Testing



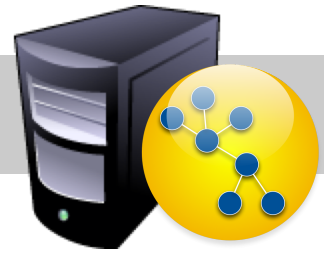
Why Test?



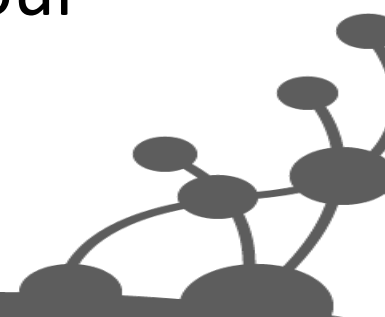
- Ensure model is fit for queries
 - Rapid feedback
- Ensure correctness of queries
- Document your understanding of your domain
 - Including corner cases and exceptions
- Provide a regression test suite
 - Allows you to change and evolve model and queries



Method

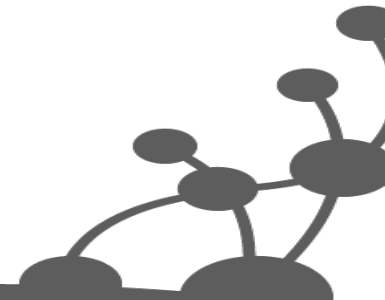


- Develop queries, or classes that encapsulate queries, using *unit tests*
- Use *small, well-understood* datasets in each test
 - Create data in test setup
 - Test dataset expresses your understanding of (part of) the domain
- Inject *in-memory graph database* (or Cypher engine) into object under test
- The exact strategy you use depends on your application architecture...



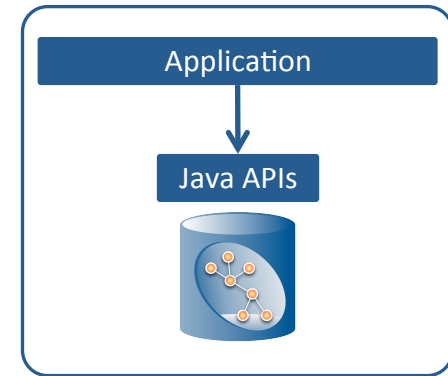
Application Architectures

- Embedded
- Server
- Server with Extensions



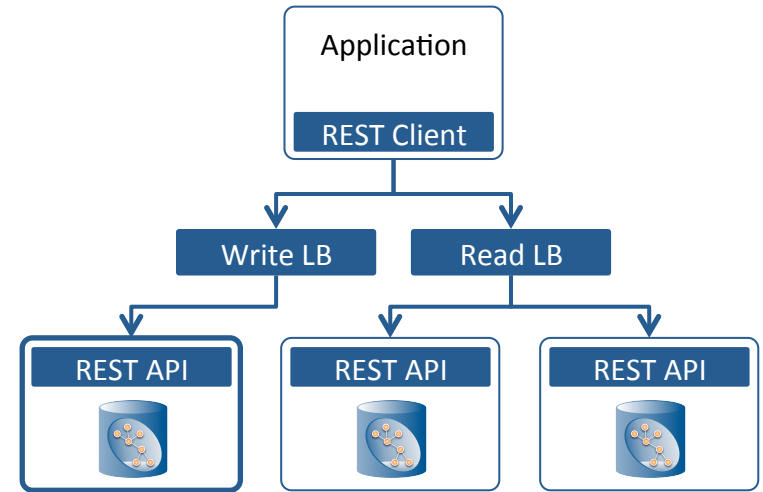
Application Architectures

- **Embedded**
 - Host in Java process
 - Access to Java APIs
- Server
- Server with Extensions



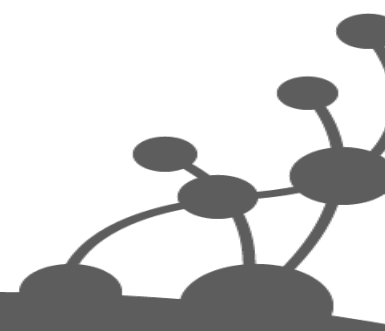
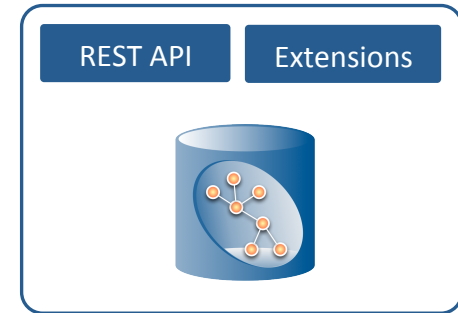
Application Architectures

- Embedded
- Server
 - HTTP/JSON interface
 - Server wraps embedded instance
- Server with Extensions

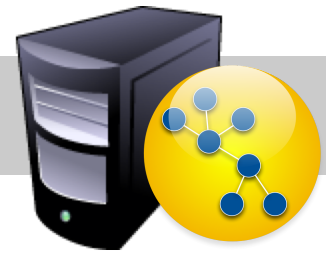


Application Architectures

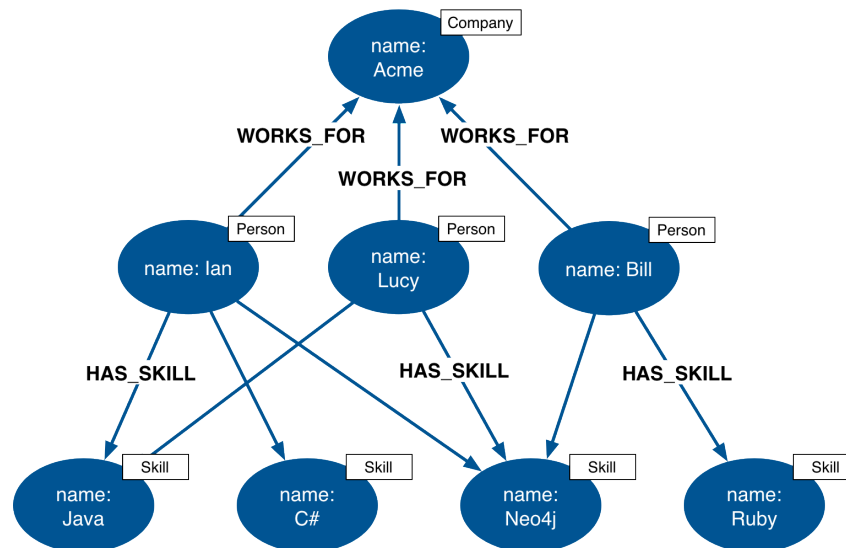
- Embedded
- Server
- **Server with Extensions**
 - Execute complex logic on server
 - Control HTTP request/response format



Embedded Example

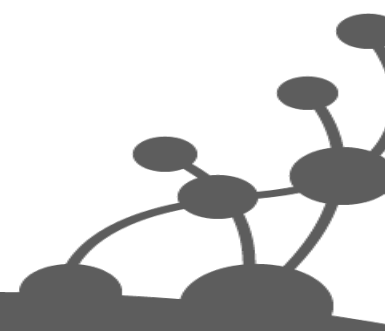


- Company social network
- Find colleagues with similar skills
- Encapsulate query in a ColleagueFinder



Unit Test Fixture

```
public class ColleagueFinderTest {  
  
    private GraphDatabaseService db;  
    private ColleagueFinder finder;  
  
    @Before  
    public void init() {  
        db = new TestGraphDatabaseFactory().newImpermanentDatabase();  
        ExampleGraph.populate( db );  
        finder = new ColleagueFinder( new ExecutionEngine( db ) );  
    }  
  
    @After  
    public void shutdown() {  
        db.shutdown();  
    }  
}
```



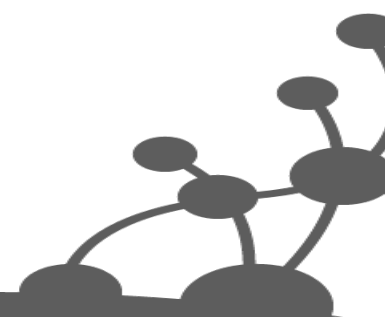
Create Database

```
public class ColleagueFinderTest {  
  
    private GraphDatabaseService db;  
    private ColleagueFinder finder;  
  
    @Before  
    public void init() {  
        db = new TestGraphDatabaseFactory().newImpermanentDatabase();  
        ExampleGraph.populate( db );  
        finder = new ColleagueFinder( new ExecutionEngine( db ) );  
    }  
  
    @After  
    public void shutdown() {  
        db.shutdown();  
    }  
}
```



Populate Graph

```
public class ColleagueFinderTest {  
  
    private GraphDatabaseService db;  
    private ColleagueFinder finder;  
  
    @Before  
    public void init() {  
        db = new TestGraphDatabaseFactory().newImpermanentDatabase();  
        ExampleGraph.populate( db );  
        finder = new ColleagueFinder( new ExecutionEngine( db ) );  
    }  
  
    @After  
    public void shutdown() {  
        db.shutdown();  
    }  
}
```

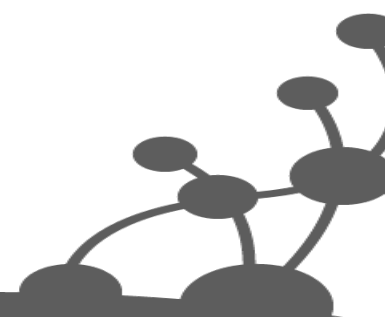


Create Object Under Test

```
public class ColleagueFinderTest {  
  
    private GraphDatabaseService db;  
    private ColleagueFinder finder;  
  
    @Before  
    public void init() {  
        db = new TestGraphDatabaseFactory().newImpermanentDatabase();  
        ExampleGraph.populate( db );  
        finder = new ColleagueFinder( new ExecutionEngine( db ) );  
    }  
  
    @After  
    public void shutdown() {  
        db.shutdown();  
    }  
}
```



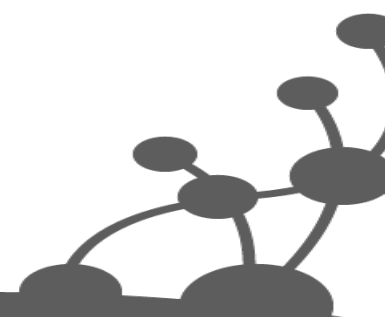
Inject
ExecutionEngine



ImpermanentGraphDatabase

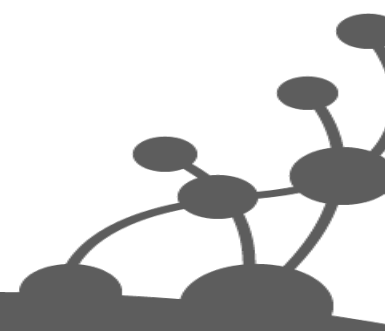
- In-memory
- For testing only, not production!

```
<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 );  
}
```



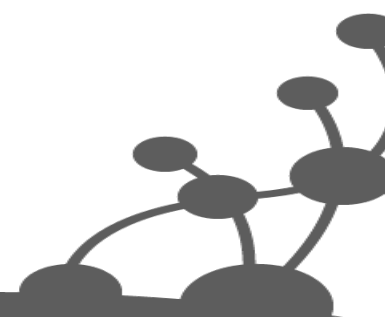
Unit Test

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

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

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

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



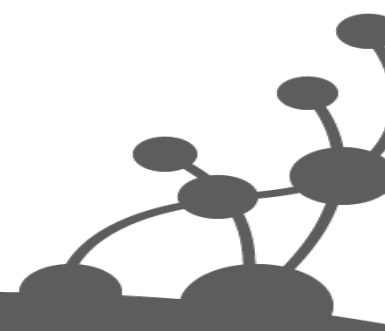
Execute Object Under Test

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

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

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

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



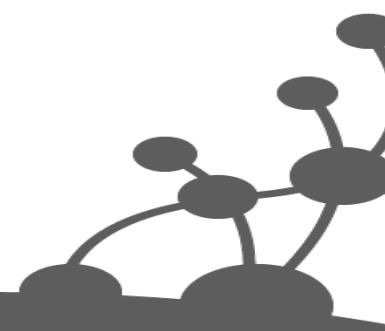
Assert Results

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

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

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

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



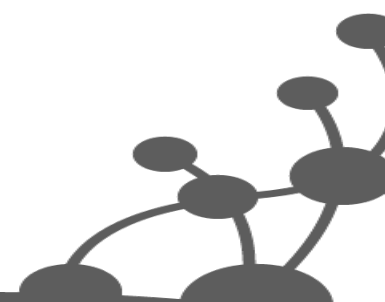
Ensure No More Results

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

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

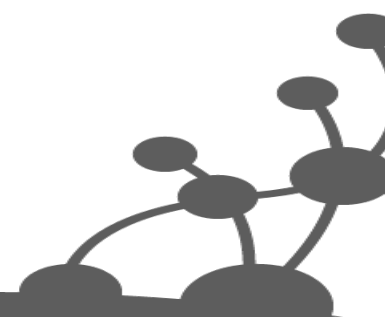
    // then
    assertEquals( "Lucy", results.next().get( "name" ) );
    assertEquals( "Bill", results.next().get( "name" ) );

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



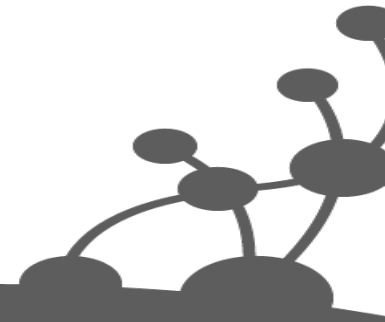
ColleagueFinder

```
public class ColleagueFinder {  
  
    private final ExecutionEngine executionEngine;  
  
    public ColleagueFinder( ExecutionEngine executionEngine ) {  
        this.executionEngine = executionEngine;  
    }  
  
    public Iterator<Map<String, Object>> findColleaguesFor( String name ) {  
        ...  
    }  
}
```



Inject ExecutionEngine

```
public class ColleagueFinder {  
  
    private final ExecutionEngine executionEngine;  
  
    public ColleagueFinder( ExecutionEngine executionEngine ) {  
        this.executionEngine = executionEngine;  
    }  
  
    public Iterator<Map<String, Object>> findColleaguesFor( String name ) {  
        ...  
    }  
}
```



findColleaguesFor() Method

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

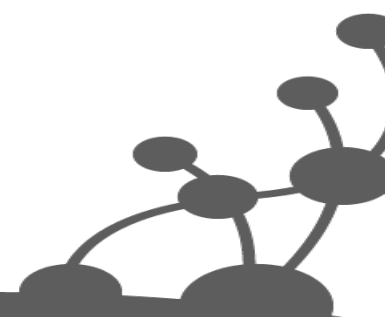
```
    String cypher =
```

```
        "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 executionEngine.execute( cypher, params ).iterator();
```

```
}
```



Cypher Query

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

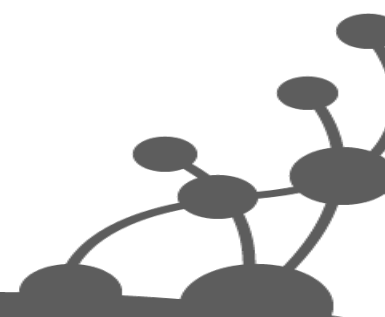
```
String cypher =
```

```
"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 executionEngine.execute( cypher, params ).iterator();
```

```
}
```



Parameterized Query

```
public Iterator<Map<String, Object>> findColleaguesFor( 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 executionEngine.execute( cypher, params ).iterator();
```

```
}
```



Execute Query

```
public Iterator<Map<String, Object>> findColleaguesFor( 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";
```

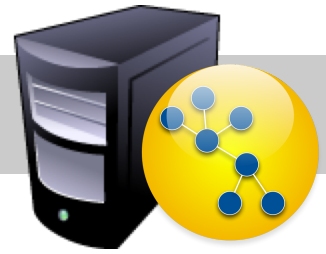
```
    Map<String, Object> params = new HashMap<String, Object>();  
    params.put( "name", name );
```

```
    return executionEngine.execute( cypher, params ).iterator();
```

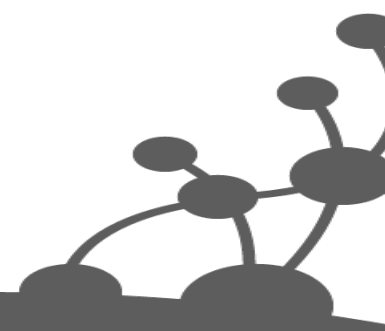
```
}
```



Server Extension Example



- Same data mode and query as before
- This time, we'll host CoLLeagueFinder in a server extension



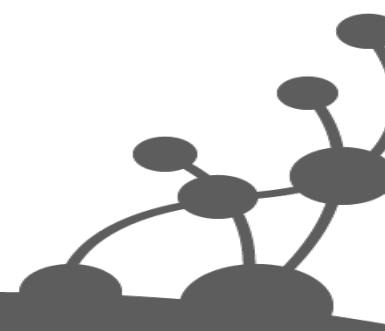
Server Extension

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

    public ColleagueFinderExtension( @Context CypherExecutor cypherExecutor ) {
        this.colleagueFinder = new ColleagueFinder( cypherExecutor.getExecutionEngine() );
    }

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

        String json = MAPPER
            .writeValueAsString( colleagueFinder.findColleaguesFor( 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 CypherExecutor cypherExecutor ) {  
        this.colleagueFinder = new ColleagueFinder( cypherExecutor.getExecutionEngine() );  
    }  
}
```

@GET

@Produces(MediaType.APPLICATION_JSON)

@Path("/{name}")

```
public Response getColleagues( @PathParam("name") String name )  
    throws IOException {  
  
    String json = MAPPER  
        .writeValueAsString( colleagueFinder.findColleaguesFor( name ) );  
    return Response.ok().entity( json ).build();  
}  
}
```



Map HTTP Request to Object + Method

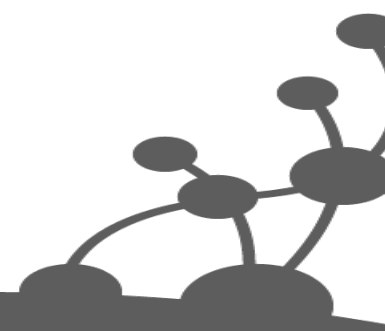
```
@Path("/similar-skills")
public class ColleagueFinderExtension {
    private static final ObjectMapper MAPPER = new ObjectMapper();
    private final ColleagueFinder colleagueFinder;

    public ColleagueFinderExtension( Context cypherExecutor cypherExecutor ) {
        this.colleagueFinder = new ColleagueFinder( cypherExecutor.getExecutionEngine() );
    }

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

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

GET /similar-skills /lan



CypherExecutor Injected by Server

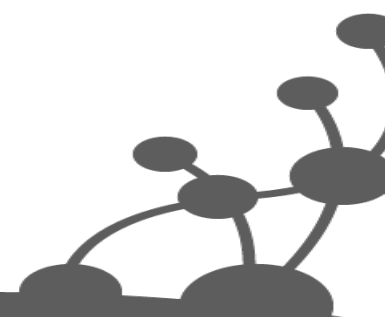
```
@Path("/similar-skills")
public class ColleagueFinderExtension {
    private static final ObjectMapper MAPPER = new ObjectMapper();
    private final ColleagueFinder colleagueFinder;

    public ColleagueFinderExtension( @Context CypherExecutor cypherExecutor ) {
        this.colleagueFinder = new ColleagueFinder( cypherExecutor.getExecutionEngine() );
    }

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

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

Ensures
ExecutionEngine
reused across
resource instances



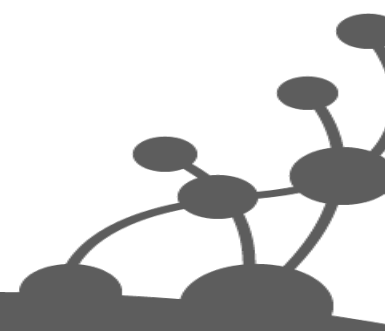
Generate and Format Response

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

    public ColleagueFinderExtension( @Context CypherExecutor cypherExecutor ) {
        this.colleagueFinder = new ColleagueFinder( cypherExecutor.getExecutionEngine() );
    }

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

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



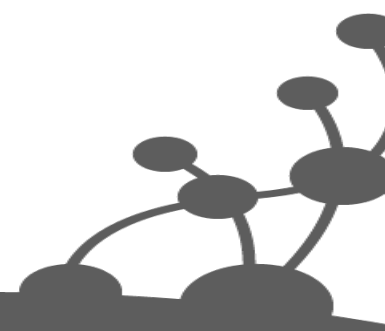
Extension Test Fixture

```
public class ColleagueFinderExtensionTest {
    private CommunityNeoServer server;

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

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

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



Build and Configure Server

```
public class ColleagueFinderExtensionTest {
    private CommunityNeoServer server;

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

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

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



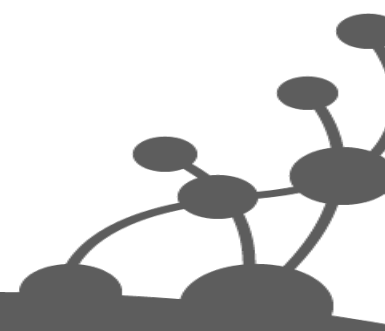
Start Server

```
public class ColleagueFinderExtensionTest {
    private CommunityNeoServer server;

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

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

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



Populate Database

```
public class ColleagueFinderExtensionTest {
    private CommunityNeoServer server;

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

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

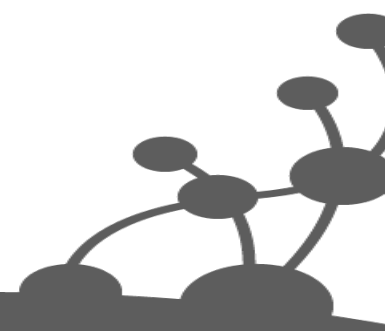
    @After
    public void stopServer() {
        server.stop();
    }
}
```



CommunityServerBuilder

- Programmatic configuration

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



Testing Extension Using HTTP Client

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

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

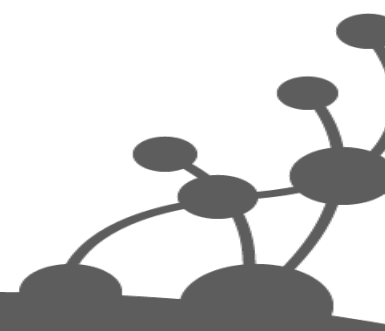
    WebResource resource = client
        .resource( "http://localhost:7474/colleagues/similar-skills/Ian" );

    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

    ...
}
```



Create HTTP Client

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

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

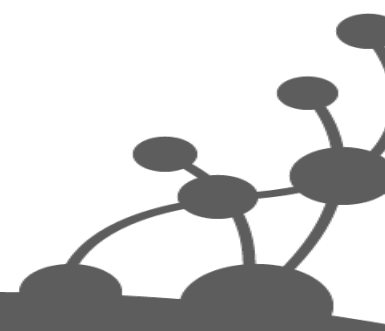
    WebResource resource = client
        .resource( "http://localhost:7474/colleagues/similar-skills/Ian" );

    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

    ...
}
```



Issue Request

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

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

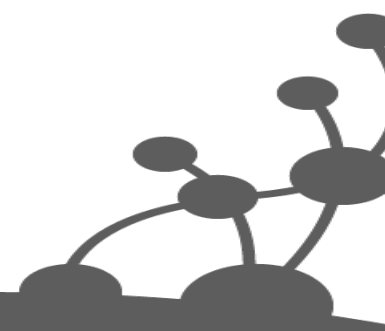
    WebResource resource = client
        .resource( "http://localhost:7474/colleagues/similar-skills/Ian" );

    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

    ...
}
```



Parse Response

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

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

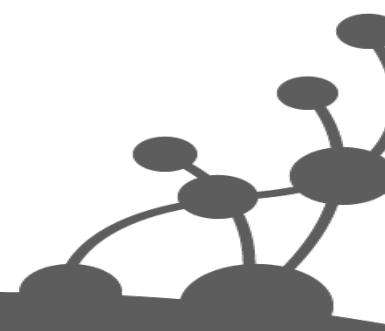
    WebResource resource = client
        .resource( "http://localhost:7474/colleagues/similar-skills/Ian" );

    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

    ...
}
```



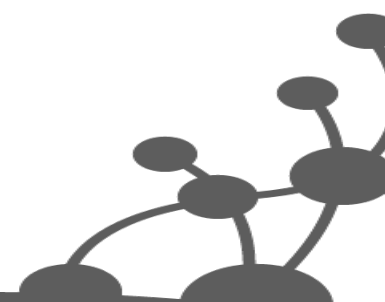
Assert Results

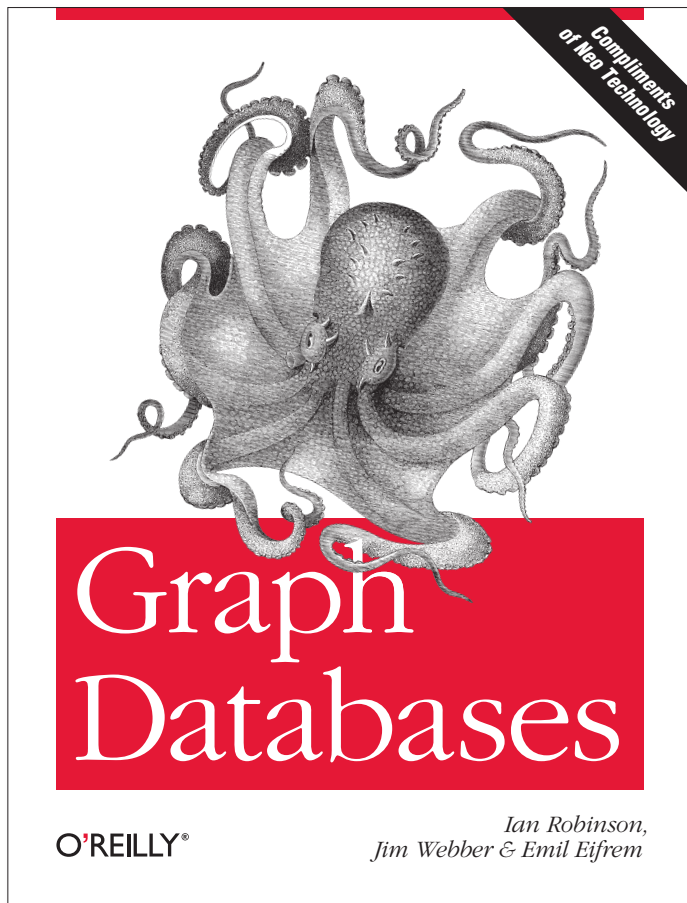
...

```
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" ) );
```

```
}
```





Thank you

Twitter: @ianSrobinson
#neo4j

github.com/iansrobinson/neo4j-good-practices

