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 Data Model

- John Le Carre wrote "Tinker, Tailor, Soldier, Spy".
- Ian purchased "Tinker, Tailor, Soldier, Spy" on 03-02-2011.
- Graham Greene wrote "Our Man in Havana".
- Alan purchased "Our Man in Havana" on 05-07-2011.
- Ian purchased "Our Man in Havana" on 09-09-2011.
Four Building Blocks

• Nodes
• Relationships
• Properties
• Labels
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

- John Le Carre WROTE Tinker, Tailor, Soldier, Spy
- Graham Greene WROTE Our Man in Havana
- lan PURCHASED Tinker, Tailor, Soldier, Spy on 03-02-2011
- lan PURCHASED Our Man in Havana on 09-09-2011
- Alan PURCHASED Our Man in Havana on 05-07-2011
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.

Self relationships are allowed.

Nodes can be connected by more than one relationship.
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
  - Title: Tinker, Tailor, Soldier, Spy
  - PURCHASED
    - Date: 03-02-2011

- Name: lan
  - PURCHASED
    - Date: 09-09-2011

- Name: Graham Greene
  - WROTE
  - Title: Our Man in Havana
  - PURCHASED
    - Date: 05-07-2011

- Name: Alan
  - Person
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

Query
User stories

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

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
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

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
Second match
Third match
Running the query

<table>
<thead>
<tr>
<th>name</th>
<th>score</th>
<th>skills</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Lucy&quot;</td>
<td>2</td>
<td>[&quot;Java&quot;,&quot;Neo4j&quot;]</td>
</tr>
<tr>
<td>&quot;Bill&quot;</td>
<td>1</td>
<td>[&quot;Neo4j&quot;]</td>
</tr>
</tbody>
</table>

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

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

#neo4j
Modeling tips
Nodes for things

- name: Acme
- name: ian
  email: ...
- name: Neo4j

#neo4j
Labels for grouping

name: Acme

person
class: company

name: Ian
email: ...

class: skill

name: Neo4j
Relationships for structure

name: Acme

name: Ian
email: ...

WORKS_FOR
start: 20090334

HAS_SKILL
level: expert

name: Neo4j
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

- **Company**: Acme
- **Person**: Ian
  - **HAS_SKILL**: Java, level: expert
  - **HAS_SKILL**: C#, level: advanced
  - **HAS_SKILL**: C#, level: beginner

- **Person**: Lucy
  - **HAS_SKILL**: C#, level: advanced

- **Person**: Bill
  - **HAS_SKILL**: Neo4j, level: expert
  - **HAS_SKILL**: Ruby, level: expert
Find Expert Colleagues

MATCH (user:Person)-[:HAS_SKILL]->(skill),
    (user)-[:WORKS_FOR]->(company),
    (colleague)-[:WORKS_FOR]->(company),
    (colleague)-[: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)-[: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
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
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
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

name: Peter

WORK_ADDRESS
first line: ...

HOME_ADDRESS
first line: ...

#neo4j
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

name: Peter

WORK_ADDRESS
ADDRESS
type: work
first line: ...

HOME_ADDRESS
ADDRESS
type: home
first line: ...

#neo4j
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

name: Ian

REVIEWED
  text: This is ...
  source: amazon.co.uk
  date: 20121125

film
  title: Lincoln
Add another review

name: Ian
REVIEWED
text: This is ...
source: amazon.co.uk
date: 20121125

name: Alan
REVIEWED
text: When I saw ...
source: filmreview.org
date: 20121204

title: Lincoln

#neo4j
And another
Problems

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

```java
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);
}
```
@Test
class 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() );
}

#neo4j
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();
}
@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
    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

```java
@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();
    }
}
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

```java
@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](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](https://github.com/jexp/cypher-rs)
Learning More
Graph Databases Book

www.graphdatabases.com
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

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 - Graph 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

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.

```
Cypher Query 2
```

<table>
<thead>
<tr>
<th>Airport</th>
<th>Average Departure Taxi Time</th>
<th>Average Arrival Taxi Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Los Angeles International Airport</td>
<td>28</td>
<td>10</td>
</tr>
<tr>
<td>Harrisfield-Jackson Atlanta International Airport</td>
<td>18</td>
<td>9</td>
</tr>
<tr>
<td>Dallas/Fort Worth International Airport</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>O'Hare International Airport</td>
<td>13</td>
<td>6</td>
</tr>
</tbody>
</table>

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 (passenger delays)? A flight planner would be interested to learn which of these types of delays are most prevalent at each of its airports.
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