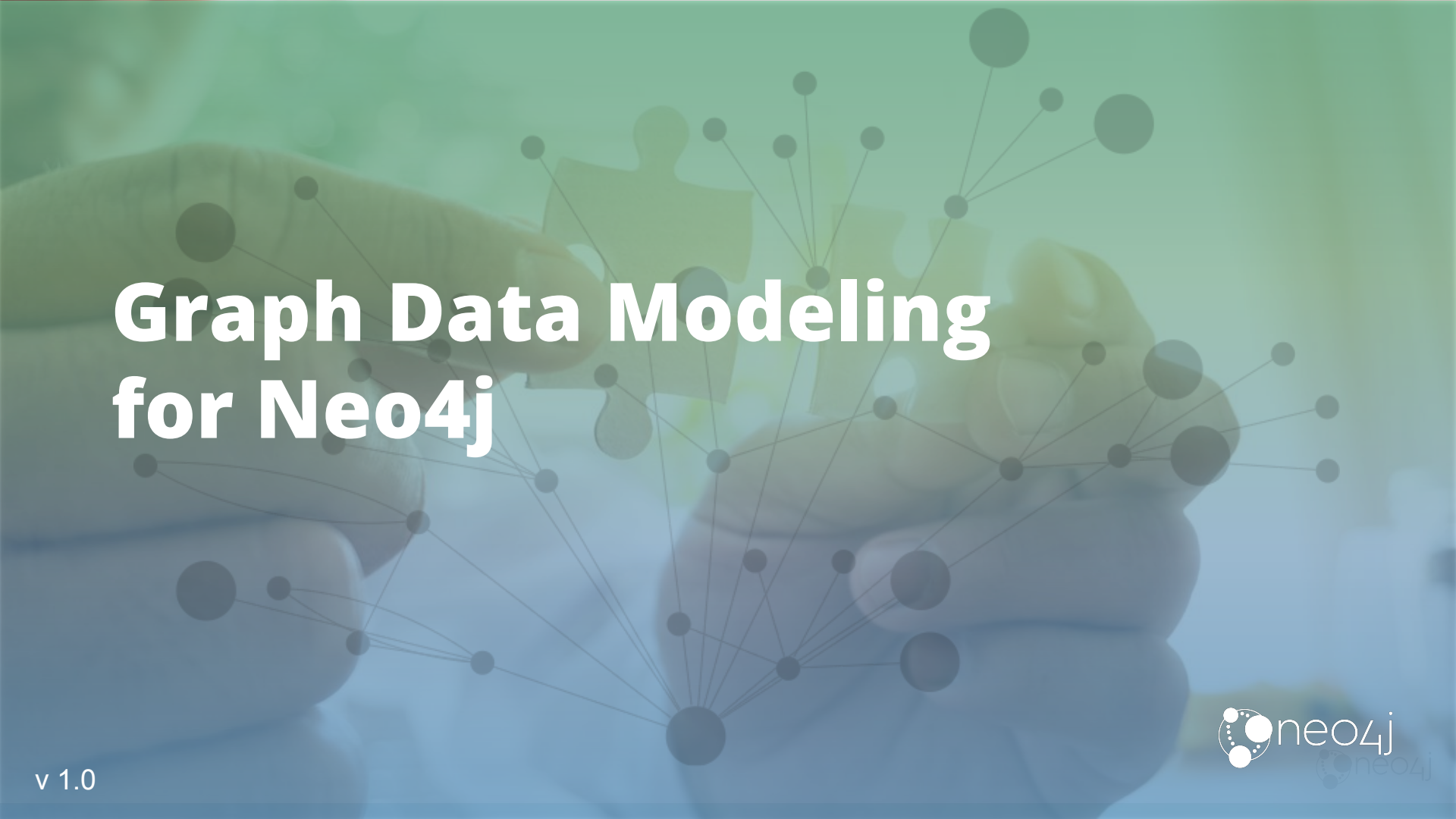
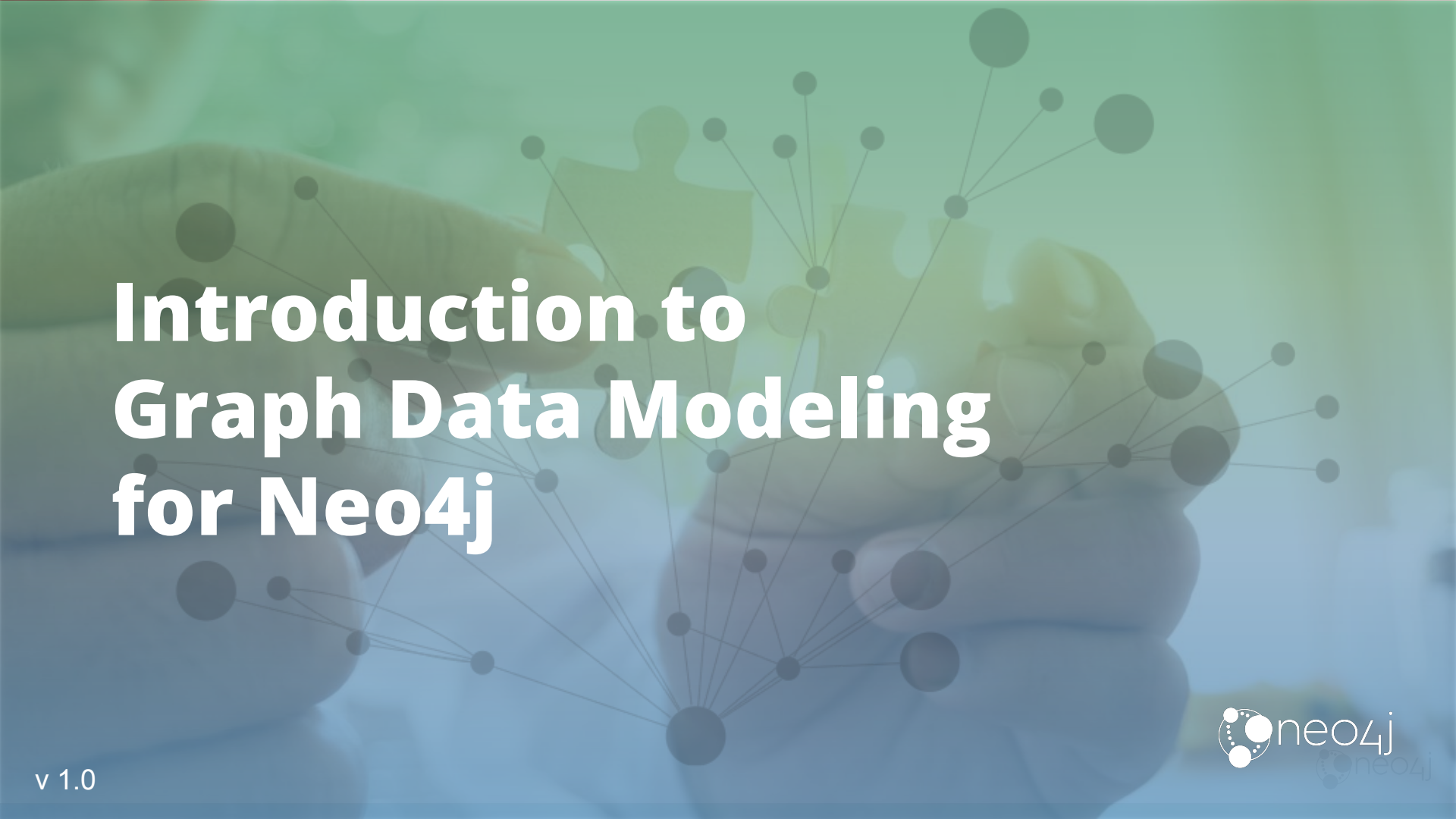


Graph Data Modeling for Neo4j

v 1.0

neo4j



Introduction to Graph Data Modeling for Neo4j

v 1.0

neo4j

Overview

At the end of this module, you should be able to:

- Describe how Neo4j supports a graph data model.
- Use Arrows to define a graph data model.
- Describe the workflow for creating a graph data model.
- Define use cases and questions for an application domain.
- Define entities for the application domain.
- Define connections between the entities for the application domain.
- Test a graph data model.
- Evolve a graph data model.

Neo4j is a property graph data model

The components of a Neo4j property graph include:

- Nodes (Entities)
- Relationships (Connections between Entities)
- Properties
- Labels

You create graphs to answer important questions for your domain.

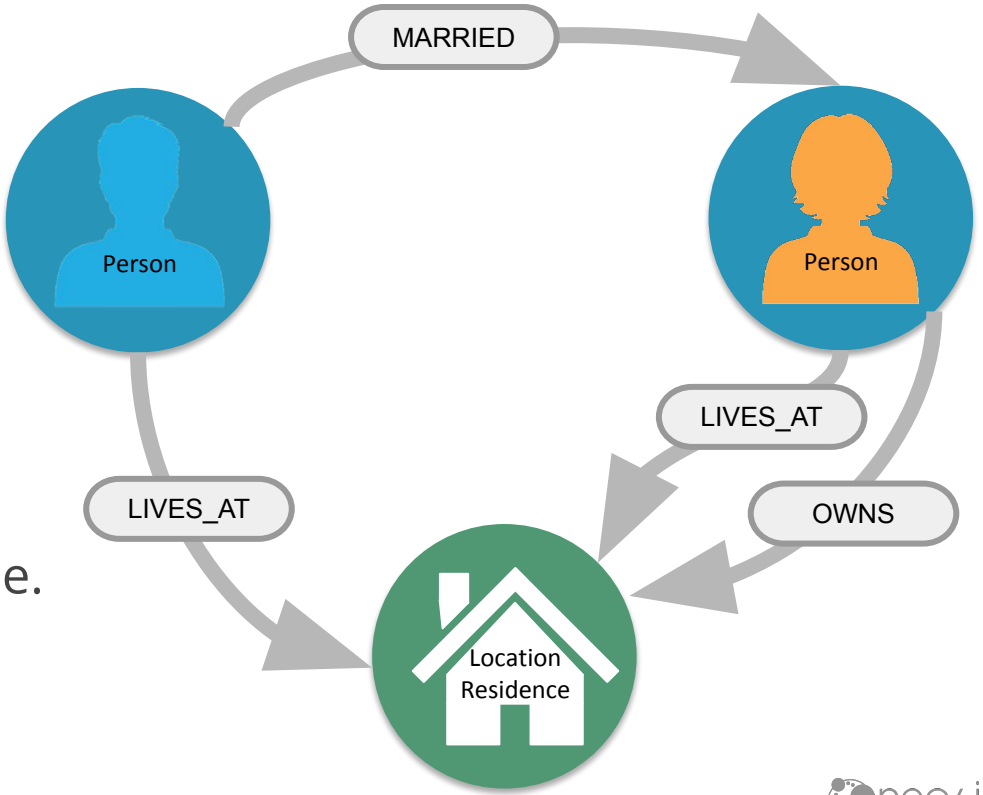
Paths are created and navigated based upon the relationships between nodes.

In Neo4j, entities are nodes with labels

- Represent the nouns in your domain questions .
- Represent the objects or entities in the graph.
- Can be ***labeled*** to represent the type of node or role of node:
  - Person
  - Location
  - Residence
  - Business
- Nodes are grouped by their **labels** for optimizing queries.

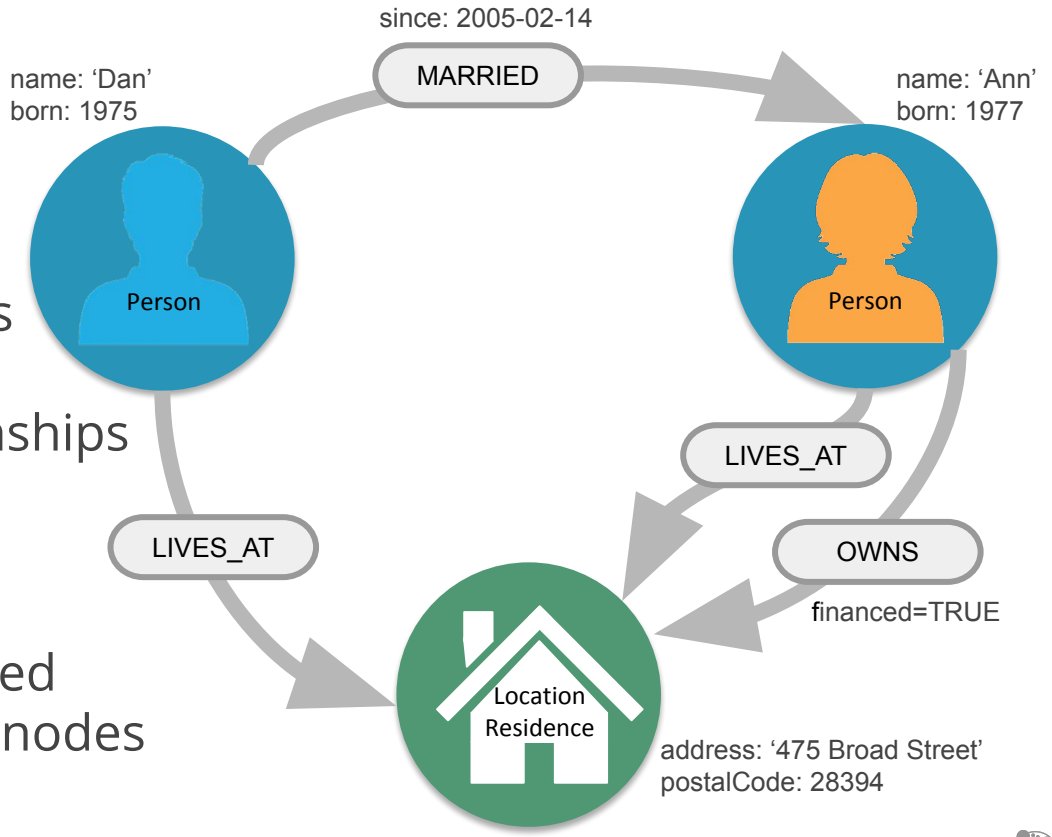
In Neo4j, connections are relationships

- Represent verbs in your domain questions.
- Represent the **connection** between exactly two nodes in the graph.
- Connect nodes of same type or of different types.
- Has a type:
  - MARRIED
  - LIVES_AT
  - OWNS
- Directed relationship in the database.
- Used to define paths navigated at runtime.



In Neo4j, properties provide the data

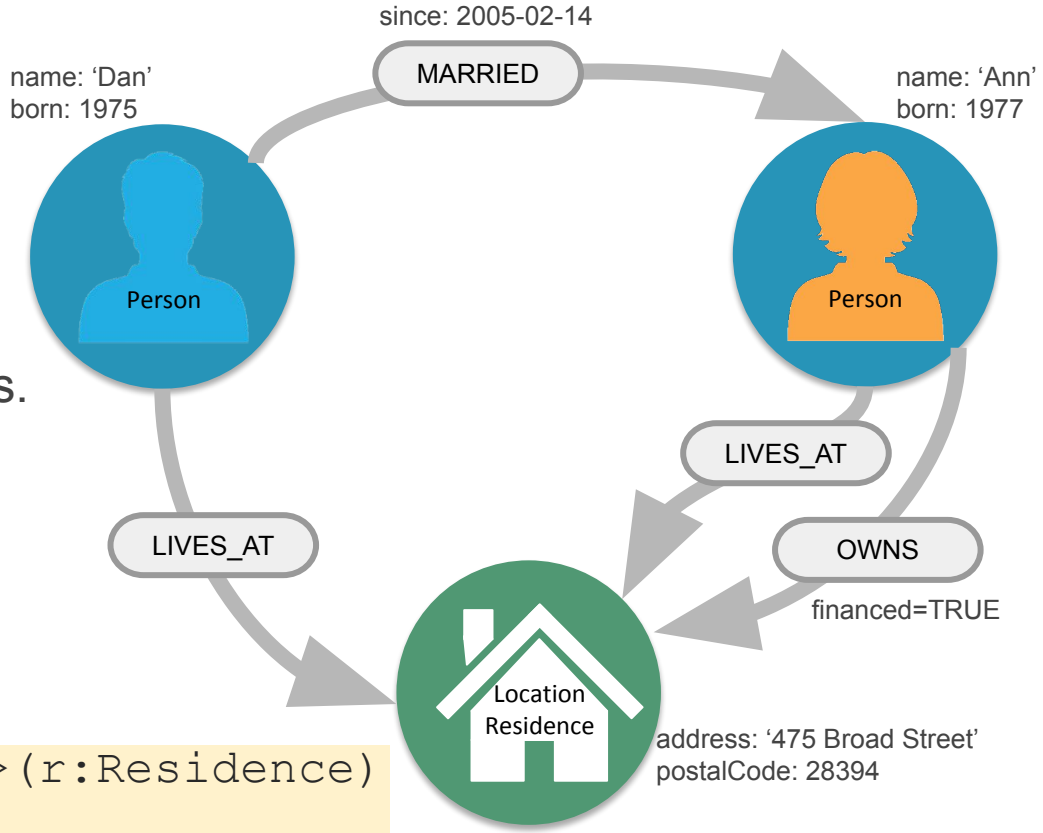
- Adjectives to describe nodes
  - Proper nouns
- Adverbs to describe relationships
  - Strength, weight, quality
- Property:
  - Key/value pair
  - Can be optional or required
  - Values can be unique for nodes



In Neo4j, paths are traversed a runtime

Paths are used to:

- Discover what is absolutely necessary to answer questions.
- Eliminate parts of graph not needed to answer a question.

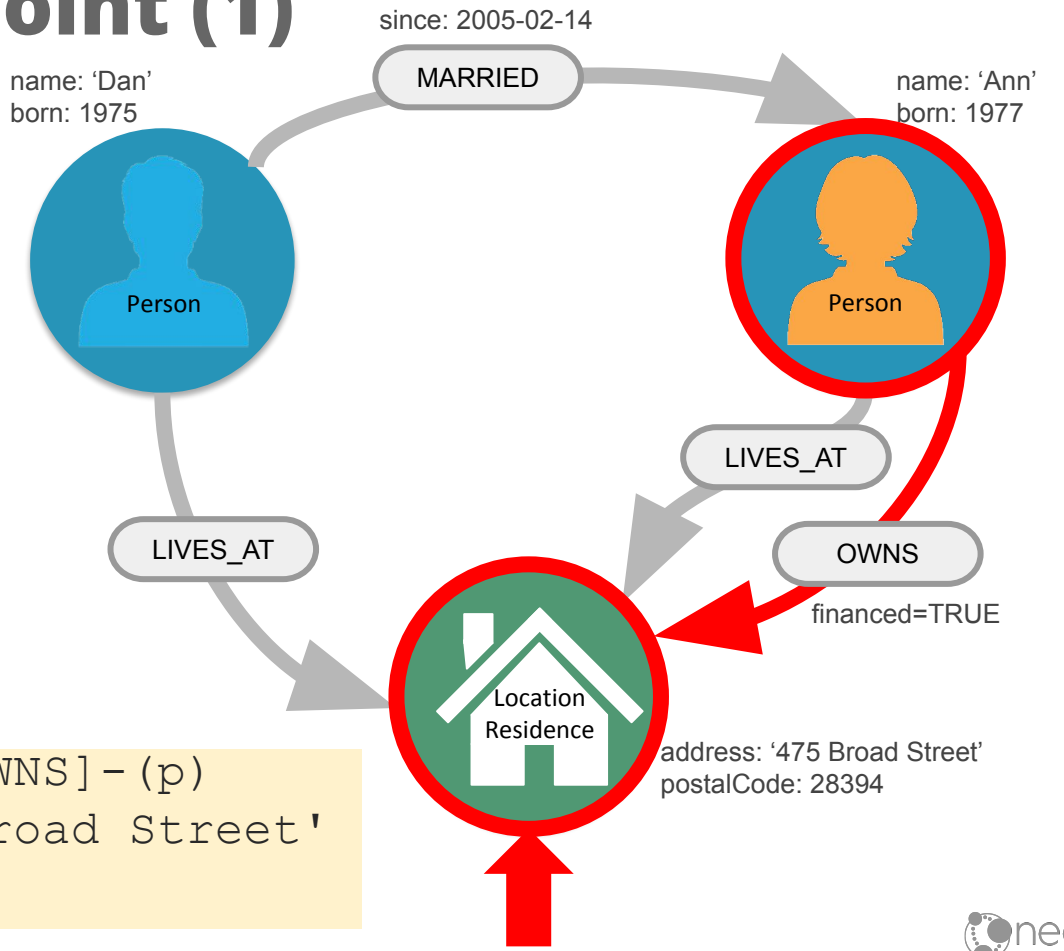


Cypher query:

```
MATCH (p:Person)-[:OWNS]->(r:Residence)
RETURN p,r
```

Path is starting point (1)

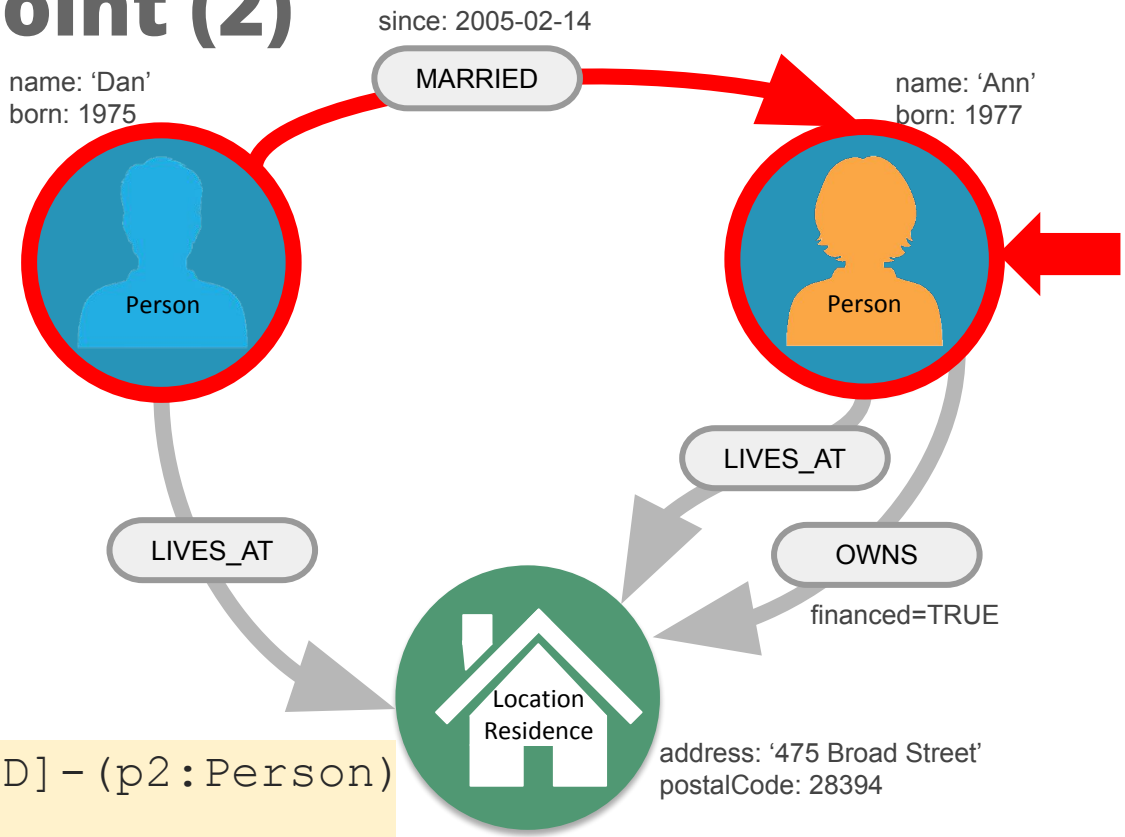
What nodes are visited to find all Person nodes that own the house at 475 Broad Street?




```
MATCH (r:Residence)<-[:OWNS]-(p)
WHERE r.address = '475 Broad Street'
RETURN p
```

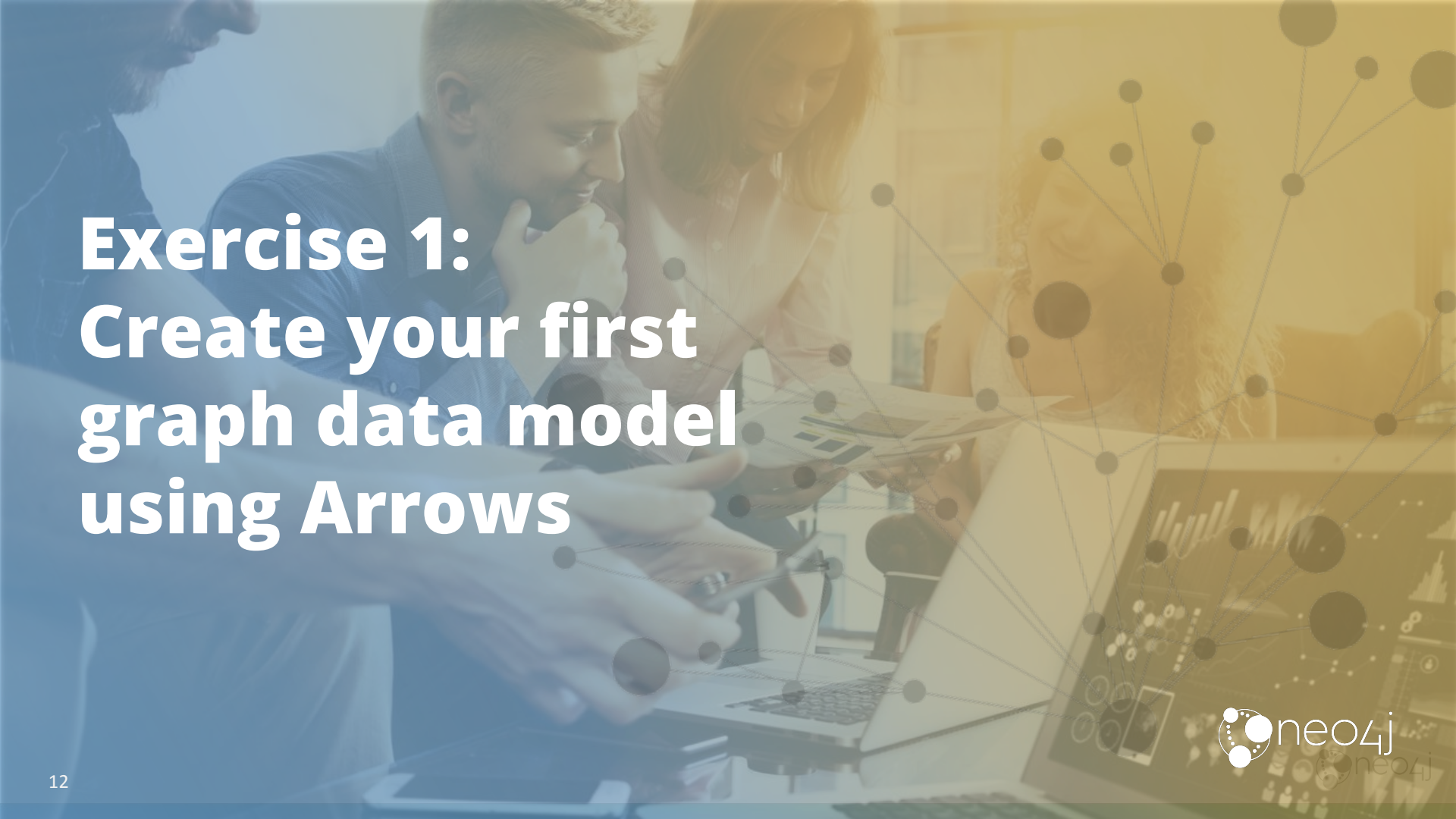
Path is starting point (2)

What nodes are visited to find who is married to Ann?




```
MATCH (p1:Person)-[:MARRIED]-(p2:Person)
WHERE p1.name = 'Ann'
RETURN p2
```

Demonstration: Using Arrows to create a graph data model



Exercise 1: Create your first graph data model using Arrows

Exercise 1: Instructions

Important: Download and uncompress the course files: https://r.neo4j.com/neo4j-intro-modeling

Domain: Knowledge management of the skills of people that work for the same company
Question: What people in the company have the same skills as me?

Steps:

1. Identify the entities and relationships based upon the above question.

2. Go to Arrows: http://www.apcjones.com/arrows.

3. Open the **KM-sample-data.txt** and view the sample data you will work with to develop the model using Arrows.

4. Create the graph data model using the sample data and the entities and relationships you have identified.

5. Save the model as **KM.htm** and **KM.svg**.

Exercise 1: Solution

Entities:

I want to know what people who work for the company have the same skills as me.

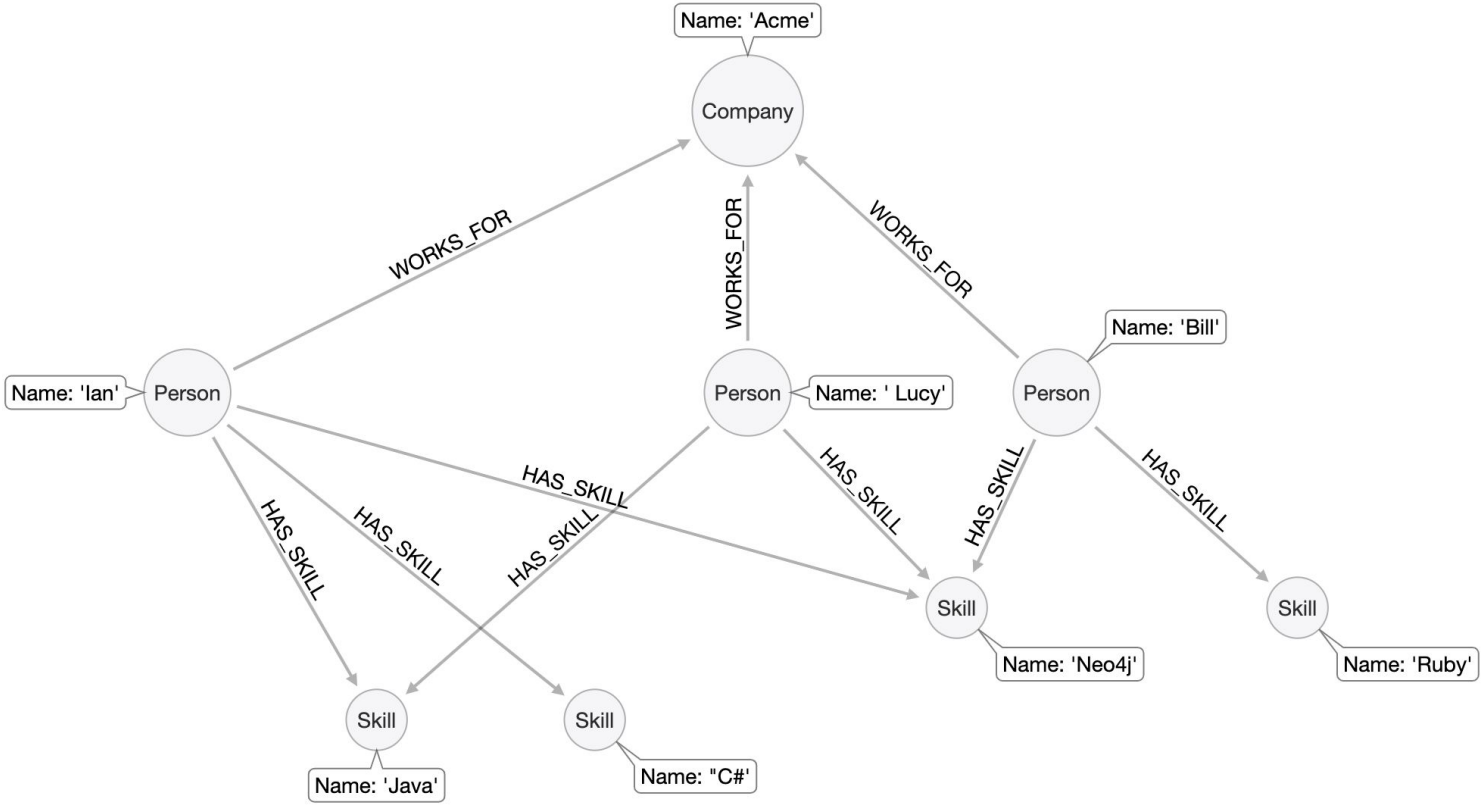
Answer: Person, Company, Skill

Relationships:

I want to know what people who work for the company have the same skills as me.

Answer: WORKS_FOR, HAS_SKILL

Exercise 1: Solution



What is graph data modeling?

Collaborative effort where the application domain is analyzed by **stakeholders** and **developers** to come up with the optimal model for use with Neo4j.

Who are the stakeholders?

- Business analysts
- Architects
- Managers
- Project leaders

Graph data modeling workflow

Step	Description	Stakeholders	Developers
1.	Define the initial graph data model	✓	✓
2.	Create and profile Cypher queries to support the model		✓
3.	Create data in the database to support the model		✓
4.	Identify additional questions for the application	✓	✓
5.	Modify the graph data model to support new questions		✓
6.	Refactor the database to support the revised graph data model		✓
7.	Create and profile Cypher queries to support the revised model		✓
	Repeat Steps 4-7	✓	✓

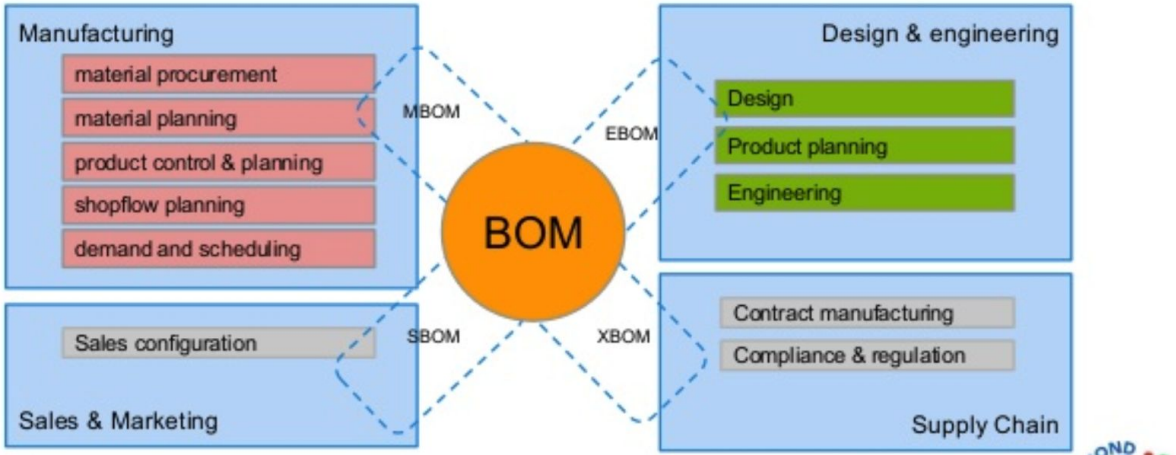
Developing the initial graph data model

1. Define high-level domain requirements
2. Create sample data for modeling purposes
3. Define the questions for the domain
4. Identify entities
5. Identify connections between entities
6. Test the model against the questions
7. Test scalability

Domain requirements

- Description of the application
- Identify stakeholders, developers
- Identify users of the application (people, systems)
- Enumerate the use cases that are agreed upon by all stakeholders where users are part of the use case

Example: Bill of materials application



Some use cases: Bill of Materials

- System produces list of parts to make a product
- System produces list of products that can be made with available parts
- System produces list of parts that are made with other parts.
- User picks parts to make a product
- System creates a price for a product based upon the part prices
- System creates list of parts that need to be ordered

Notes:

- A product or part can be made of multiple parts of the same type
- Some parts are made from other parts (sub-assembly)

Developing the initial graph data model

1. Define domain requirements
2. Create sample data for modeling purposes
3. Define the questions for the domain
4. Identify entities
5. Identify connections between entities
6. Test the model against the questions
7. Test scalability

Sample data for modeling

Products	Parts	Assemblies	Notes
Wood table 40"	Wood top 40"	Leg assembly	Has 4 legs
Deluxe wood table 40"	Wood top 60"		Has 4 legs
Wood table 60"	Glass top 40"		Has 6 legs, table brace
Deluxe wood table 60"	Glass top 60"		Has 6 legs, table brace
	Leg		
	Leg foot		
	M20 bolt		
	M20 nut		
	Leg plate		Uses 2 bolts/nuts per leg
	Table brace		

Developing the initial graph data model

1. Define domain requirements
2. Create sample data for modeling purposes
3. Define the questions for the domain
4. Identify entities
5. Identify connections between entities
6. Test the model against the questions
7. Test scalability

Some questions: Bill of Materials

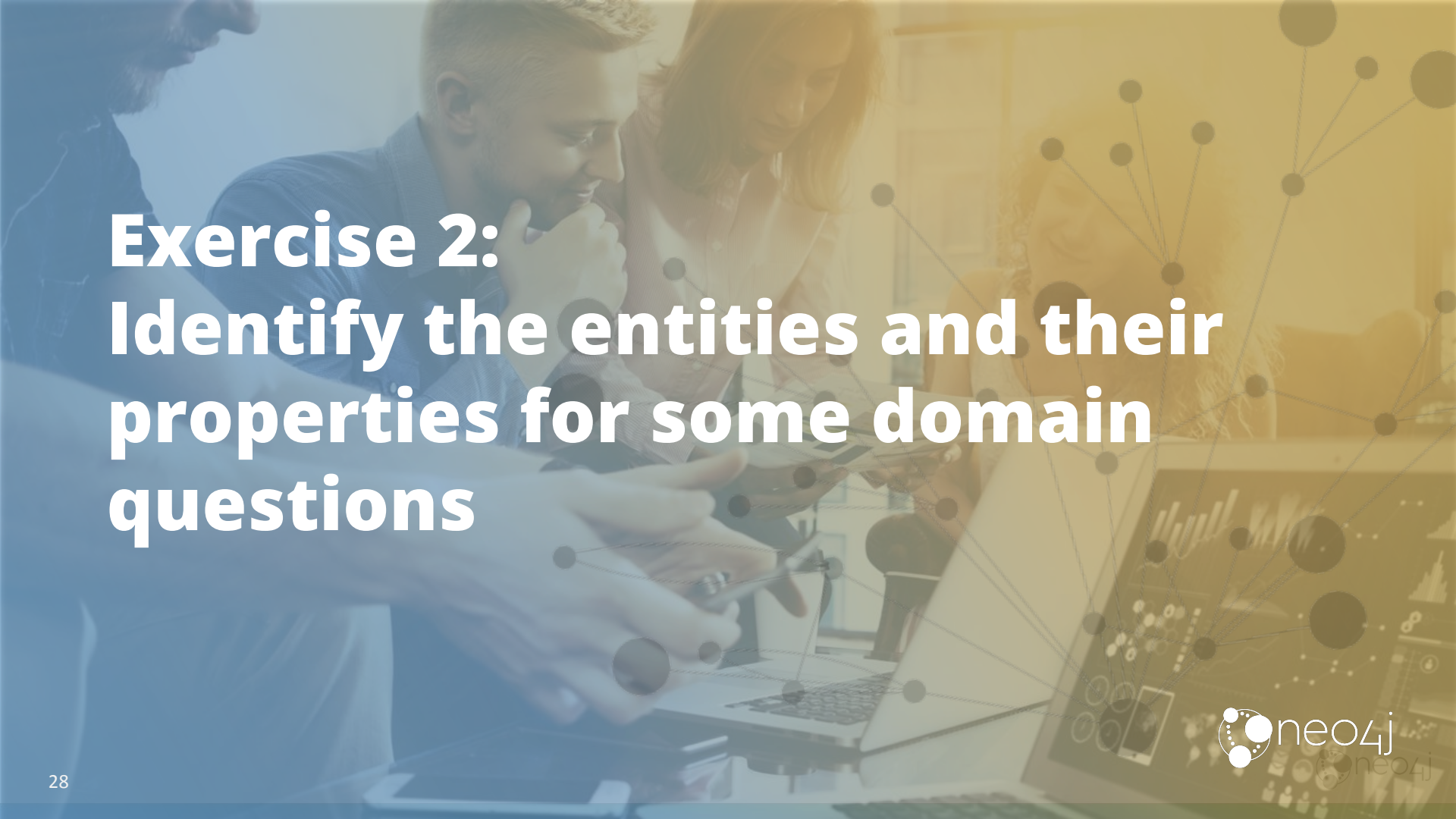
- What parts are needed to make Wood table 40"?
- Do we have enough parts to make 100 Deluxe wood table 60"?
- What products require a table brace?
- How much will the parts cost to make product Wood table 60"?

Developing the initial graph data model

1. Define domain requirements
2. Create sample data for modeling purposes
3. Define the questions for the domain
4. Identify entities
5. Identify connections between entities
6. Test the model against the questions
7. Test scalability

Identify the entities from the questions

- Nouns
  - Entities are the generic nouns (example: City)
  - Name the entities which will be nodes in the graph:
    - Unambiguous meaning for the name
    - Agreement by stakeholders
  - Can entities be grouped or categorized?
    - Pets, dogs, cats
- Properties for the entities
  - Property value is a proper noun for the name of the entity (example: Boston)
  - Uniquely identify an entity
  - Used to describe the entity to answer questions (anchor for the query)



Exercise 2: Identify the entities and their properties for some domain questions

Exercise 2: Instructions

Identify the entities and their properties for these domain questions:

- What parts are needed to make Wood table 40"?
- Do we have enough parts to make 100 Deluxe wood table 60"?
- What products require a table brace?
- How much will the parts cost to make product Wood table 60"?

Exercise 2: Solution

Questions:

- What parts are needed to make Wood table 40"?
- Do we have enough parts to make 100 Deluxe wood table 60"?
- What products require a table brace?
- How much will the parts cost to make product Wood table 60"?

Answer: 3 entities and their properties:

Product

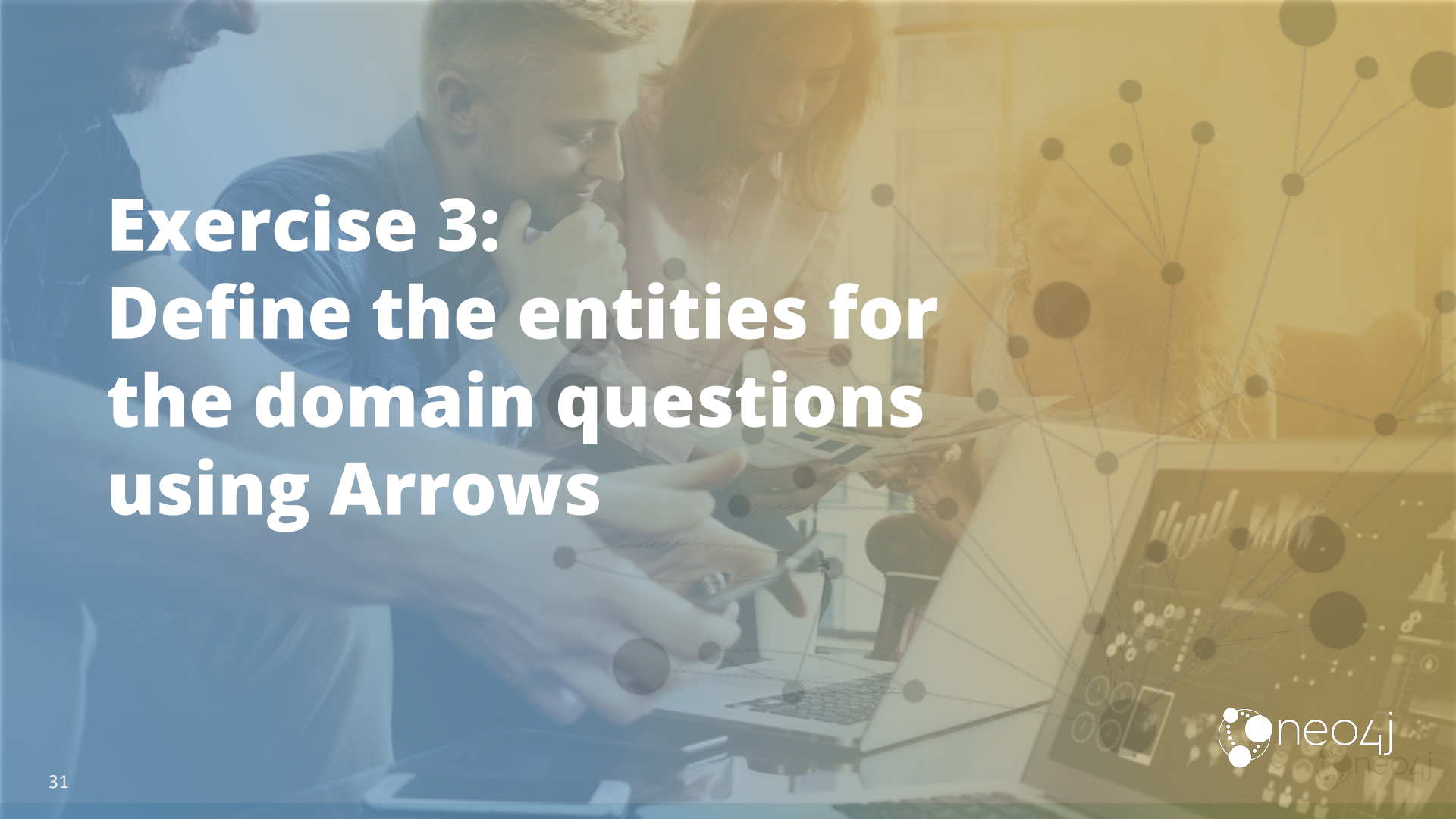
- Name
- ProductID

Part

- Name
- PartNumber
- Price

Part, Assembly

- Name
- PartNumber
- Price



Exercise 3: Define the entities for the domain questions using Arrows

Exercise 3: Instructions

1. Go to http://www.apcjones.com/arrows.
2. Clear the graph data model.

   Hint: Click **Export Markup** and delete all <li> entries, then click **Save**.

3. Create the nodes and properties for the Bill of Materials entities you identified earlier using the data in **BOM-1.txt**
4. Save the model as **BOM-1.htm** and **BOM-1.svg**

Exercise 3: Solution

Developing the initial graph data model

1. Define domain requirements
2. Create sample data for modeling purposes
3. Define the questions for the domain
4. Identify entities
5. Identify connections between entities
6. Test the model against the questions
7. Test scalability

Identify connections between entities

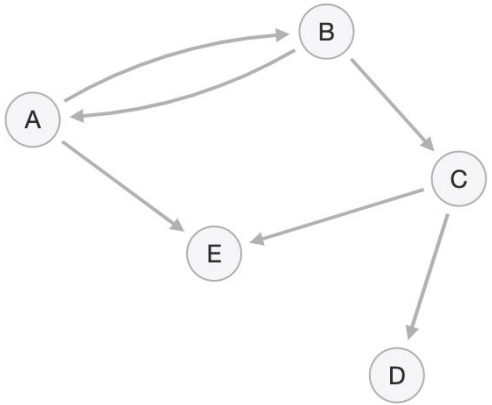
- The purpose of the model is to answer questions about the data.
- Connections are the verbs in the questions derived from use cases.
- Most questions are typically about the the connectedness of the data:
  - What ingredients are used in a recipe?

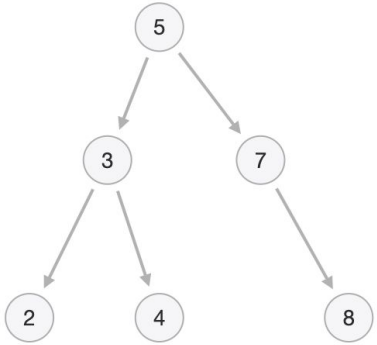
  - Who is married to this person?

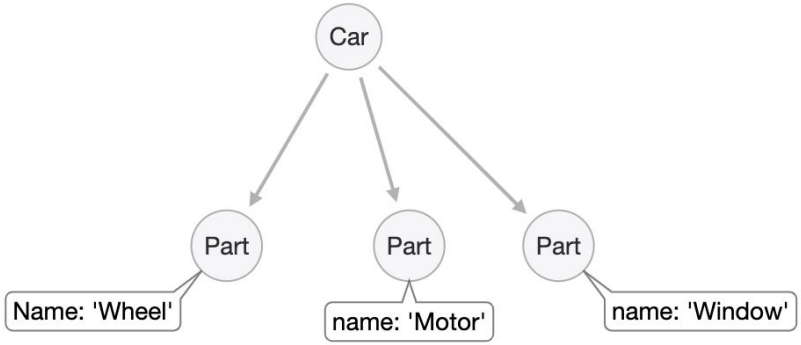
- Exactly **one** node at the end of every relationship (connection).
- A relationship must have direction when it is created.

Graph structure

- Relationships define the **structure** of the graph which is the **model**:
  - Between nodes of same type
  - Between nodes of different types
  - Determine navigational paths used at runtime (optimization)
  - Can also connect two different models in a graph
    - HR model connects to Payroll model

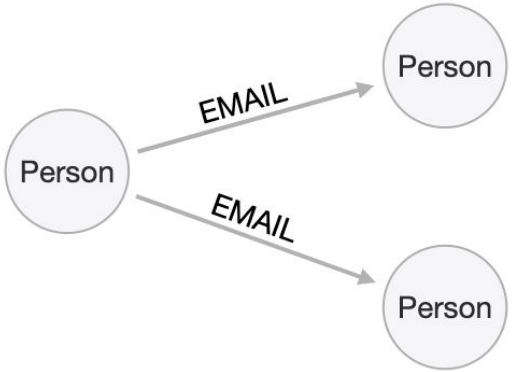


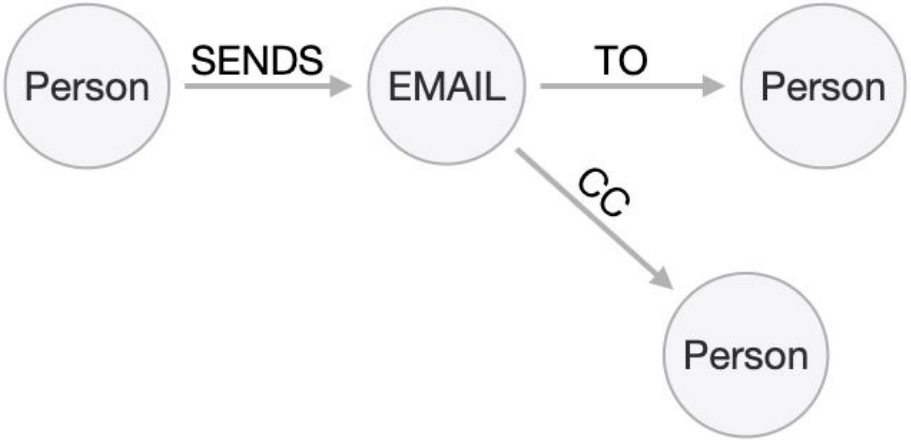




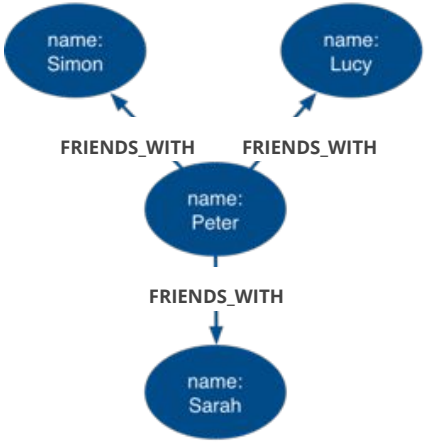
Naming relationships

- Stakeholders must agree upon name that denotes the verbs.
- Avoid names that could be construed as nouns (for example **email**)

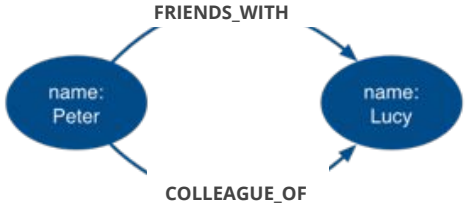




How many relationships?



Nodes can have more than one relationship



Nodes can be connected by more than one relationship

Self relationships are allowed

How important is direction?

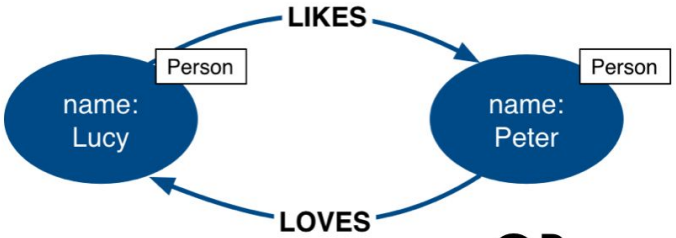
Direction is required for creating the model and in particular for implementing the model (Cypher code) in the underlying Neo4j graph.

Whether direction is used for queries depends on the question:

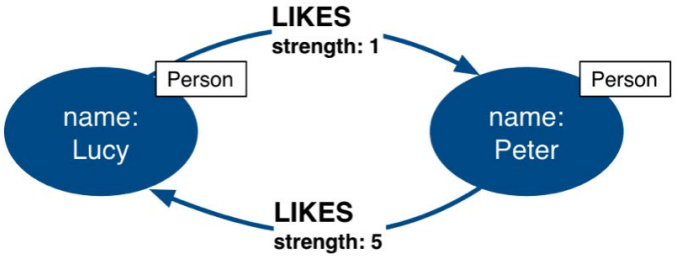
- What are the names of the episodes of the Dr. Who series? (direction not used for the query)
- What episode follows The Ark in Space? (direction used for the query)

Qualifying a relationship

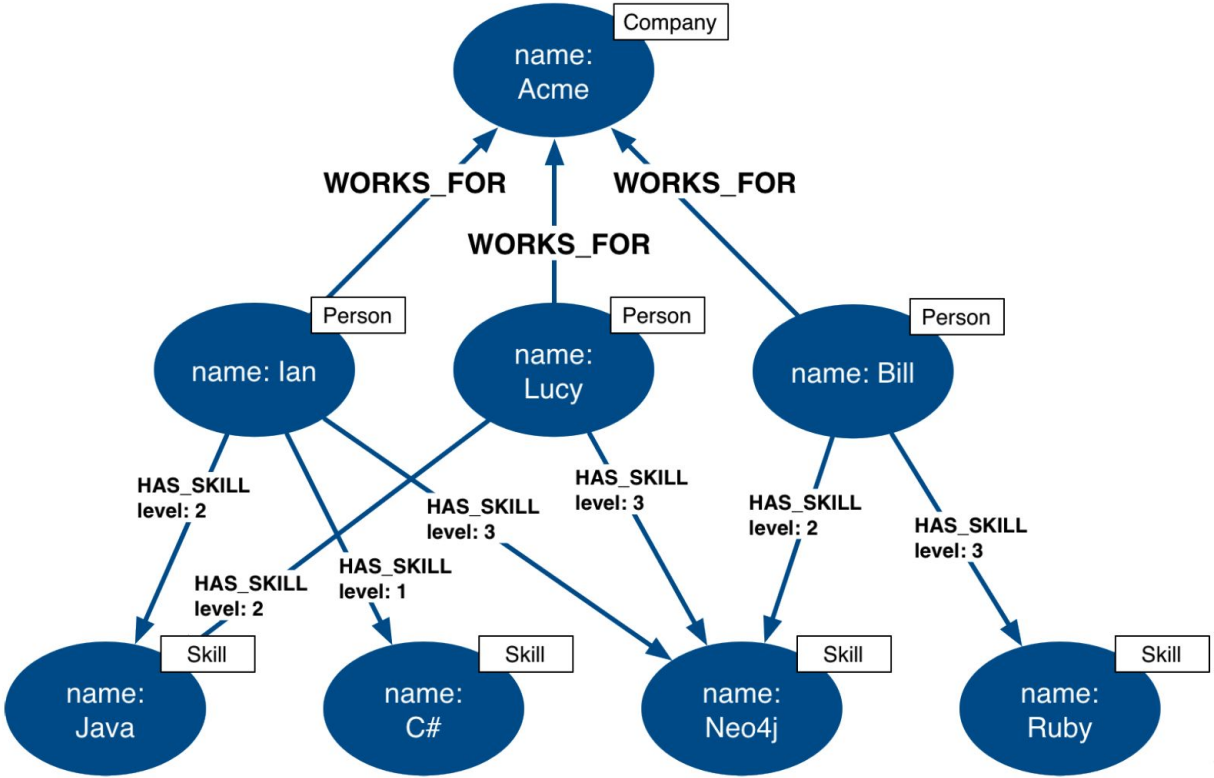
Use properties to describe the weight or quality of the relationship.



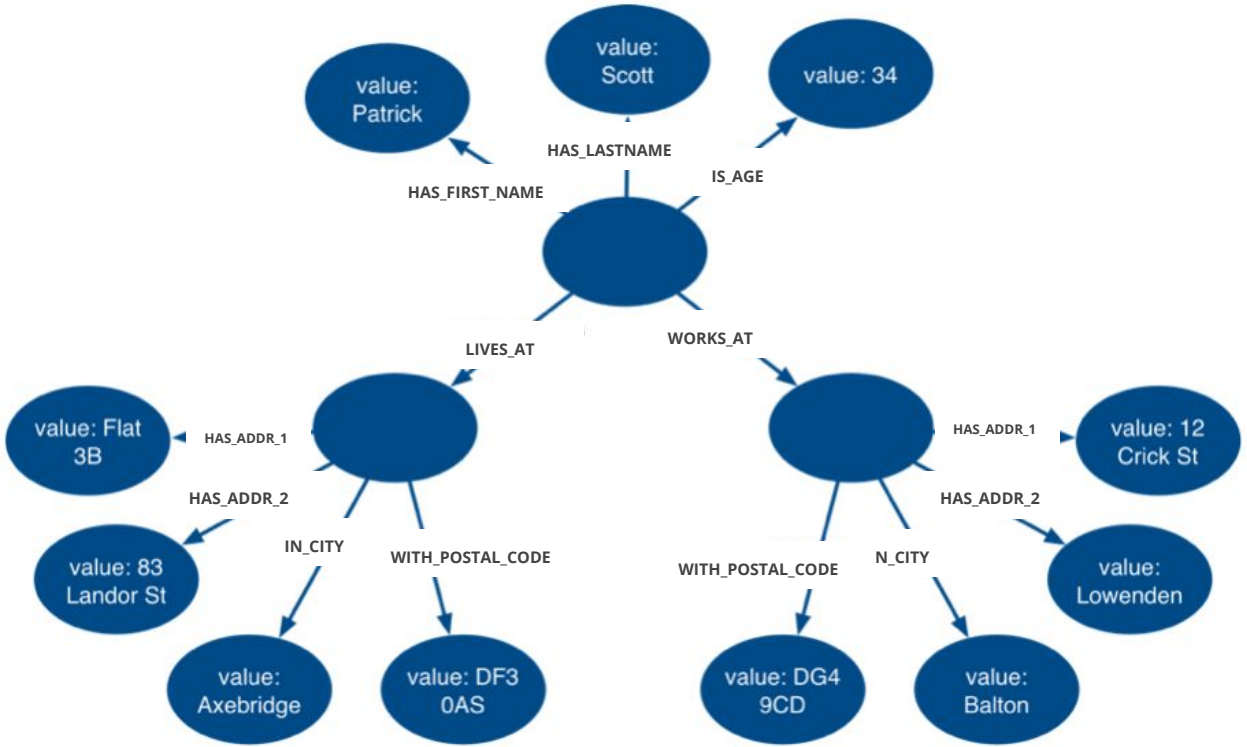
OR



Example: Find expert colleagues



How much fanout will a node have?



Exercise 4: Adding relationships to the model

Exercise 4: Instructions

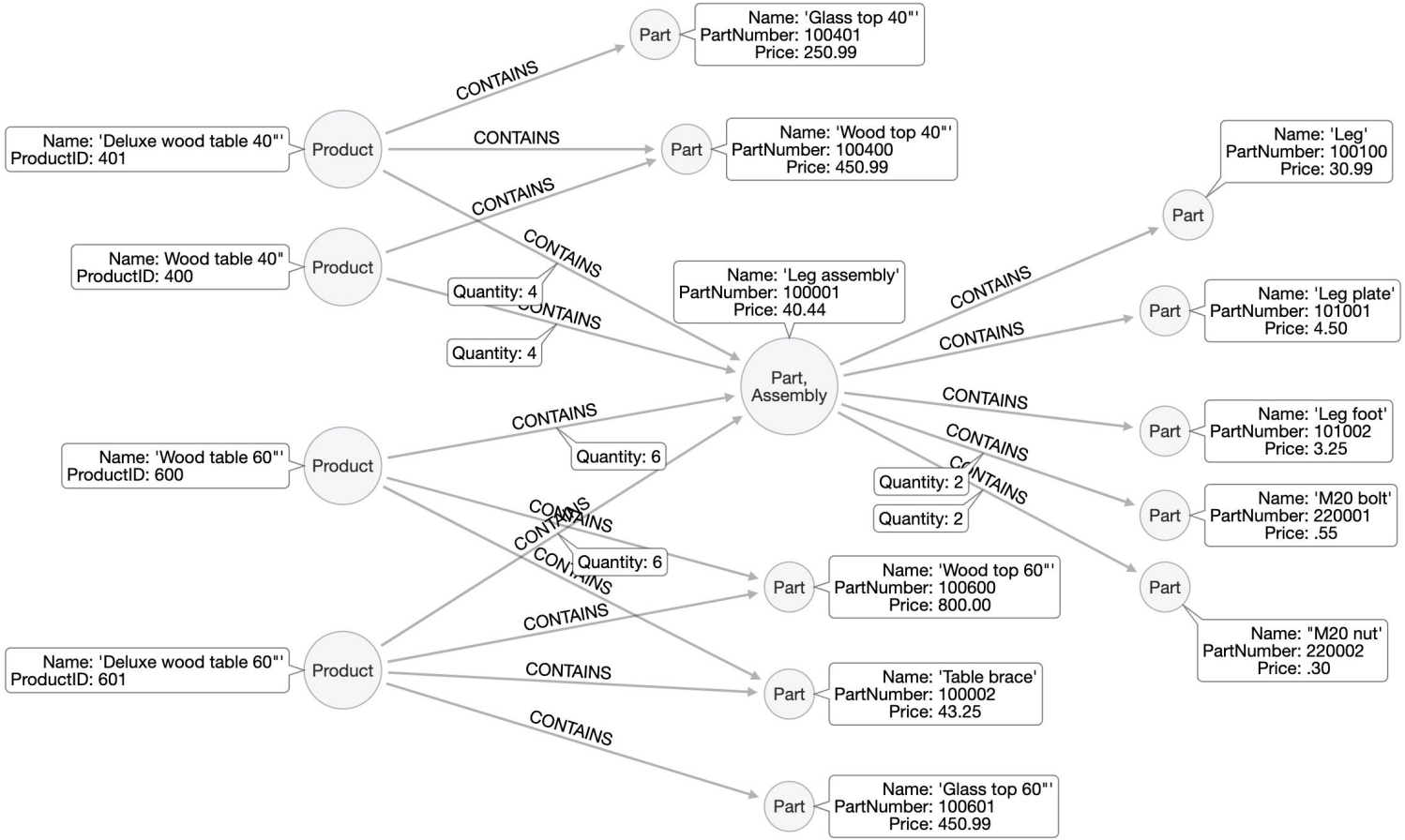
1. Continue with Arrows: http://www.apcjones.com/arrows

2. Use the **BOM-1.htm** to re-display the BOM model.
   Hint: Use **Export Markup** to copy paste from **BOM-1.htm**.

3. Add relationships to the model using the previous questions as well as the sample data presented earlier:
   - What parts are needed to make Wood table 40"?
   - Do we have enough parts to make 100 Deluxe wood table 60"?
   - What products require a table brace?
   - How much will the parts cost to make product Wood table 60"?

4. Save the model as **BOM-2.htm** and **BOM-2.svg**.

Exercise 4: Solution

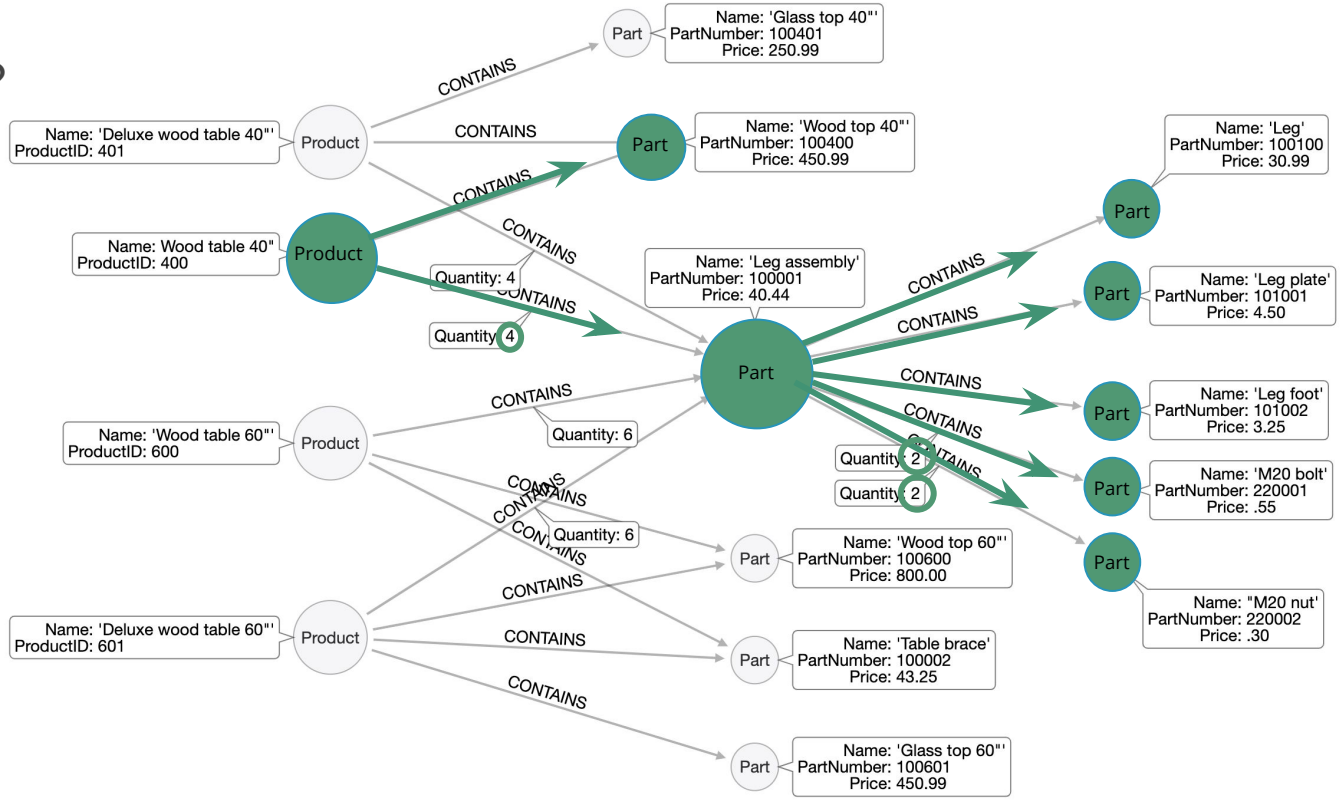


Developing the initial graph data model

1. Define domain requirements
2. Create sample data for modeling purposes
3. Define the questions for the domain
4. Identify entities
5. Identify connections between entities
6. Test the model against the questions
7. Test scalability

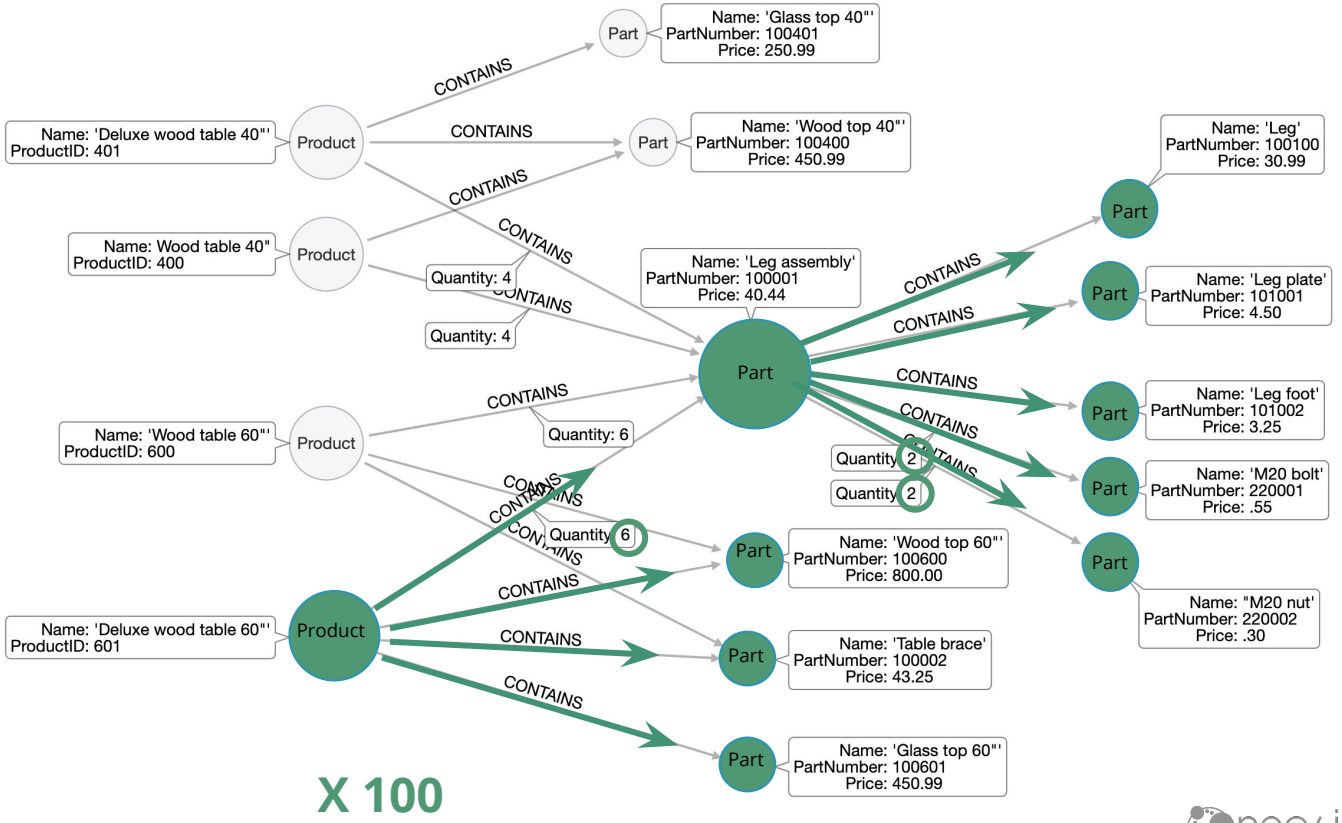
Testing the model

What parts are needed
to make Wood table 40"?



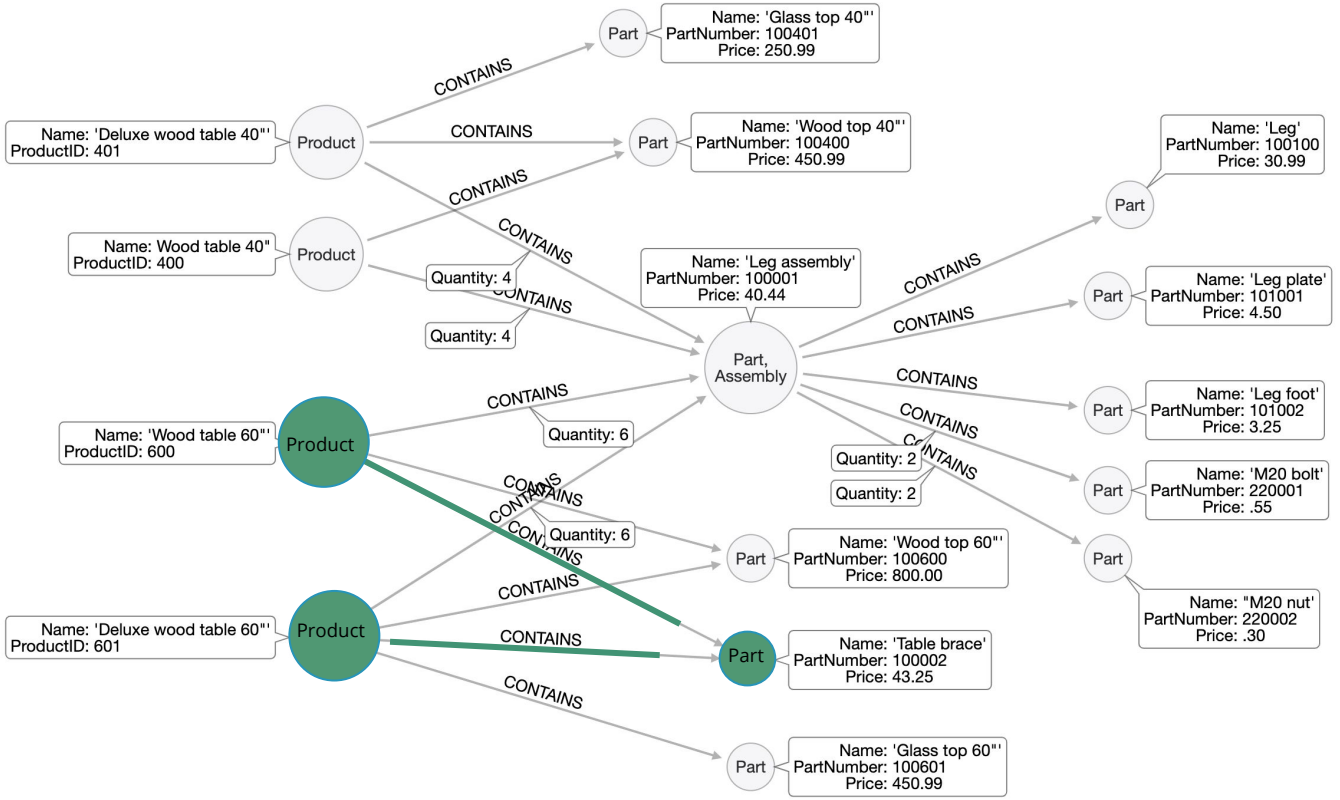
Testing the model

Do we have enough
parts to make
100
Deluxe wood table 60"?



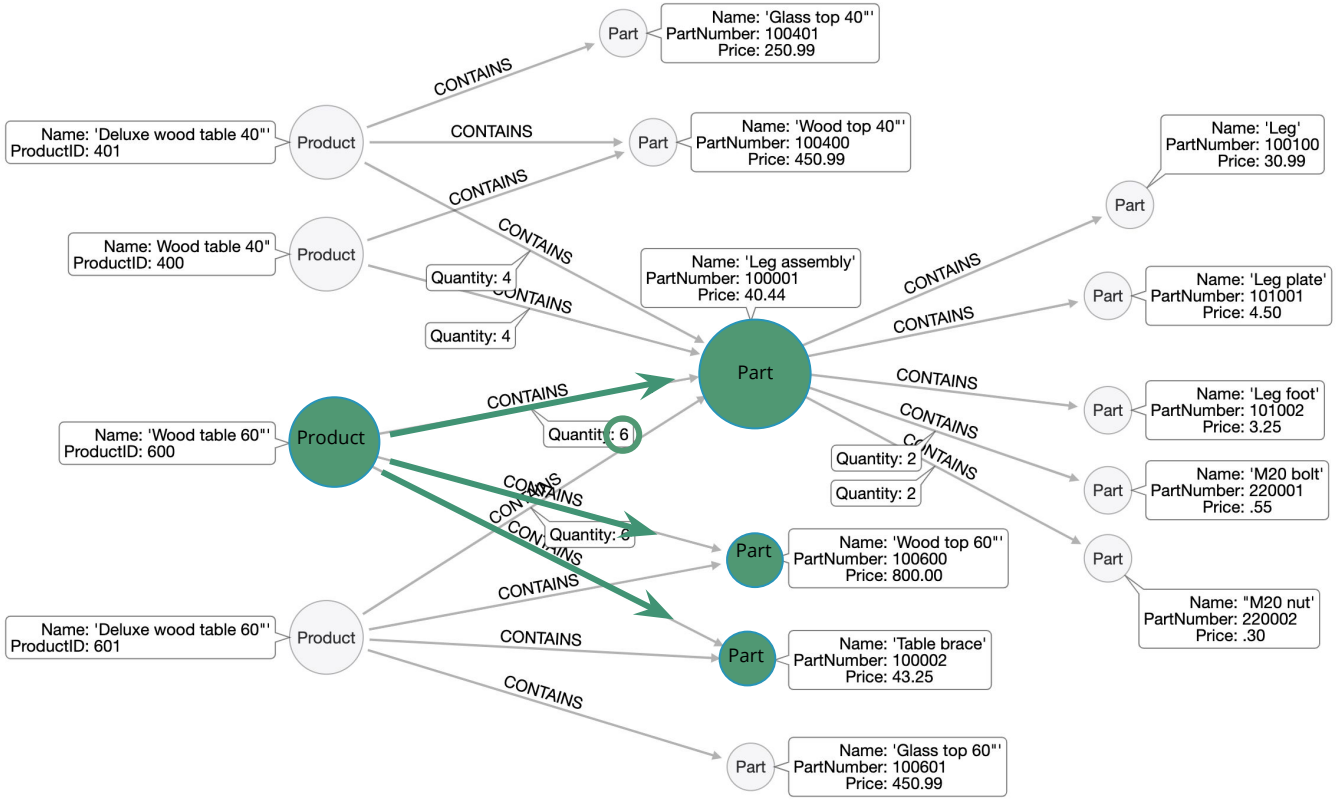
Testing the model

What products require a table brace?



Testing the model

How much will the parts cost to make Wood table 60"?

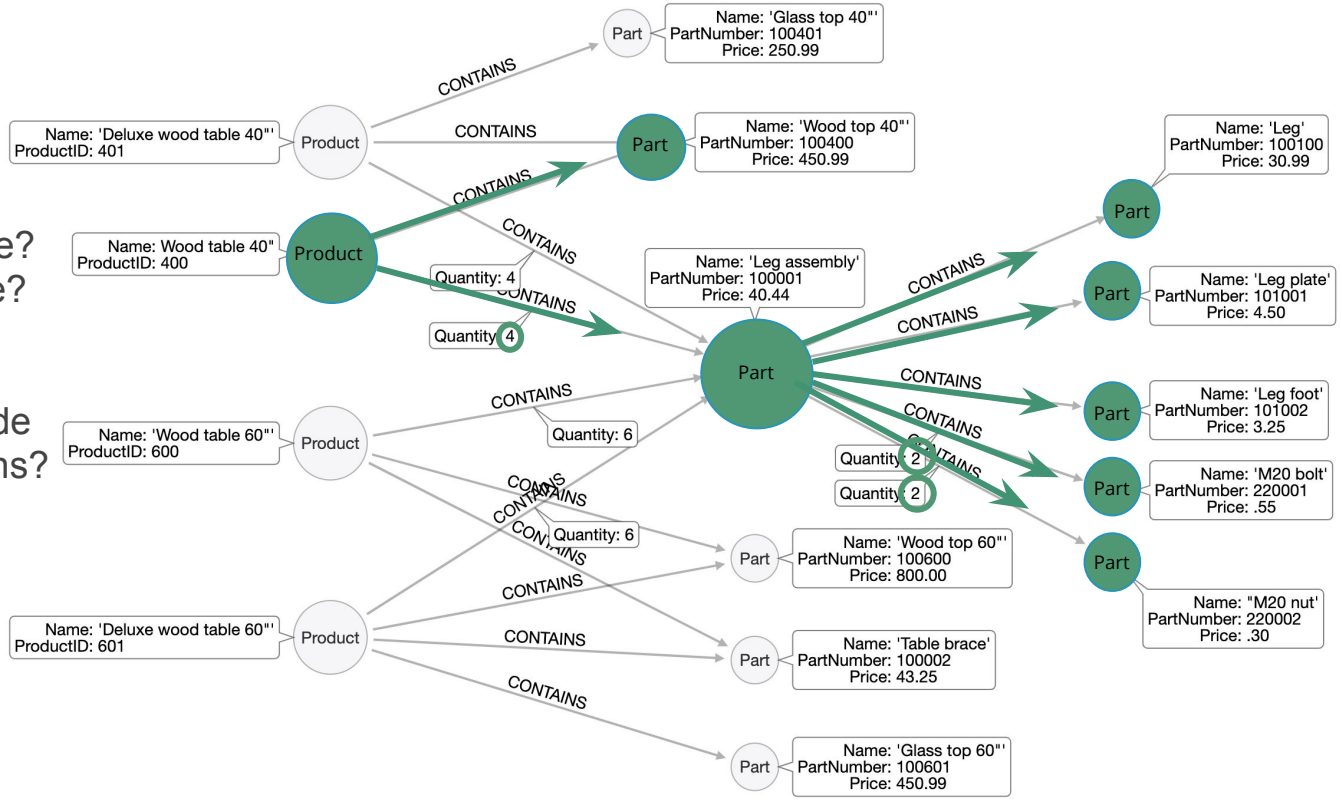


Developing the initial graph data model

1. Define domain requirements
2. Create sample data for modeling purposes
3. Define the questions for the domain
4. Identify entities
5. Identify connections between entities
6. Test the model against the questions
7. Test scalability

Testing scalability

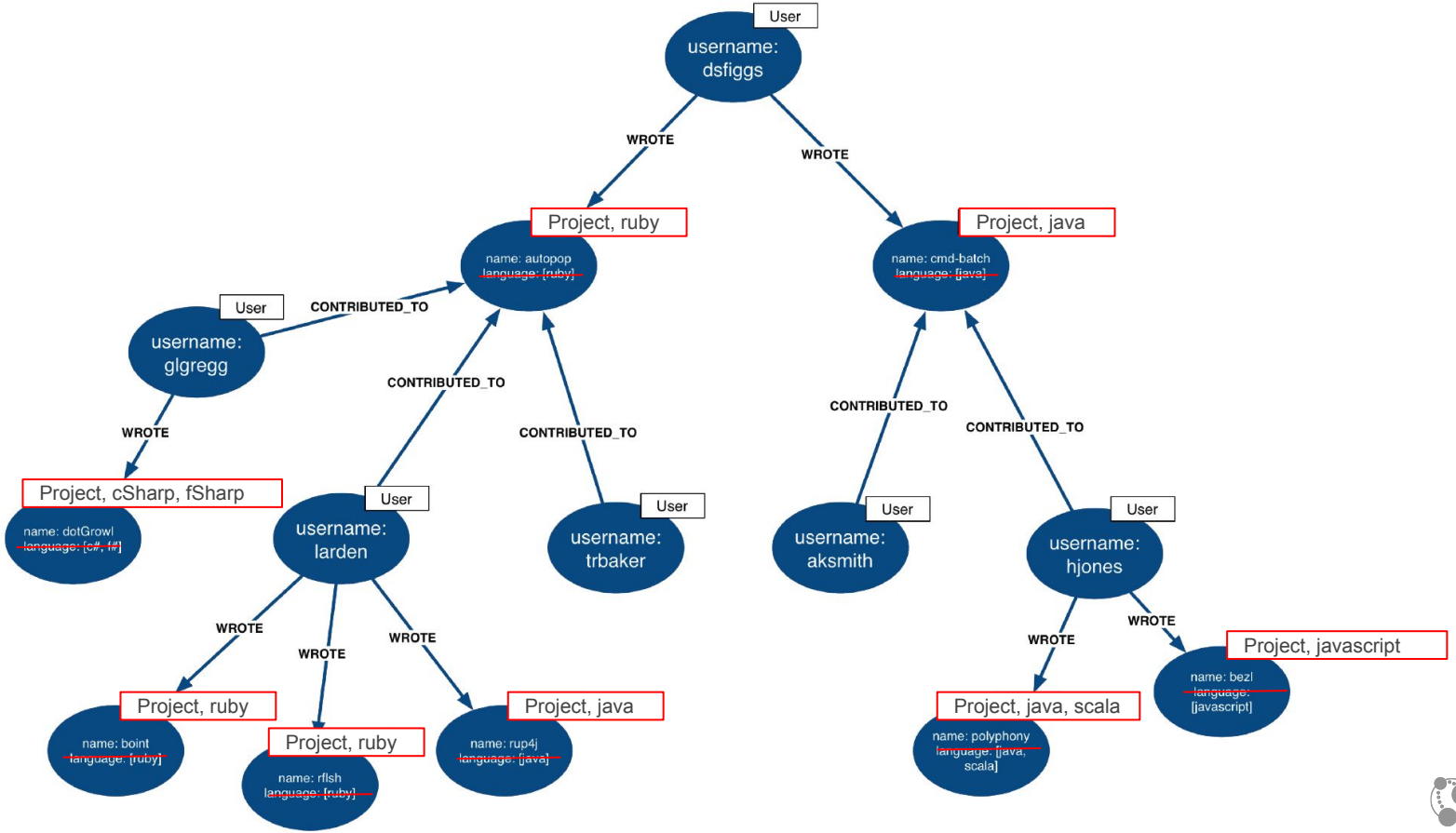
- How many products?
- How many parts?
- How often are products added?
- How often do prices change?
- Are prices based upon time?
- Is inventory part of the model?
- Does the data need to reside in different physical locations?



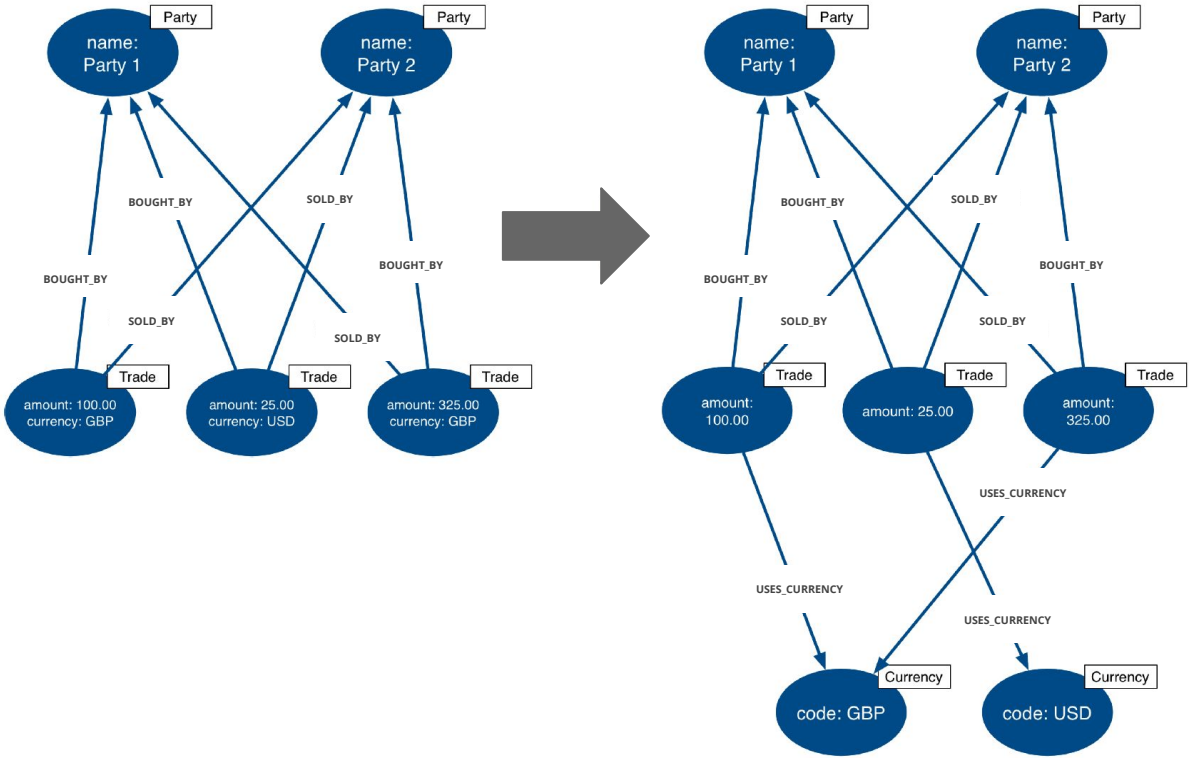
Best practices for graph data modeling

- Group nodes with labels
- Choose relationships over node properties
- No symmetric relationships
- Prefer multiple relationship types
- Avoid duplication of data
- Avoid complex property values for nodes and relationships

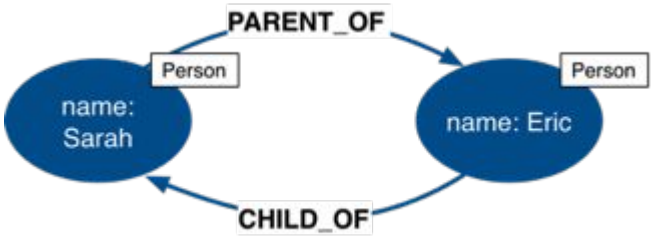
Best practice: Group nodes



Best practice: Choose relationships over node properties



Best practice: No symmetric relationships



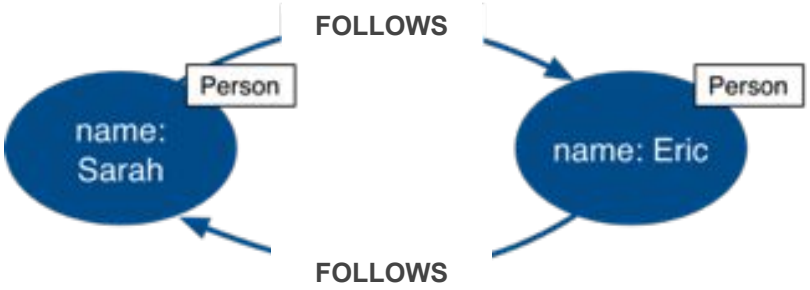
You should never do this.

OR

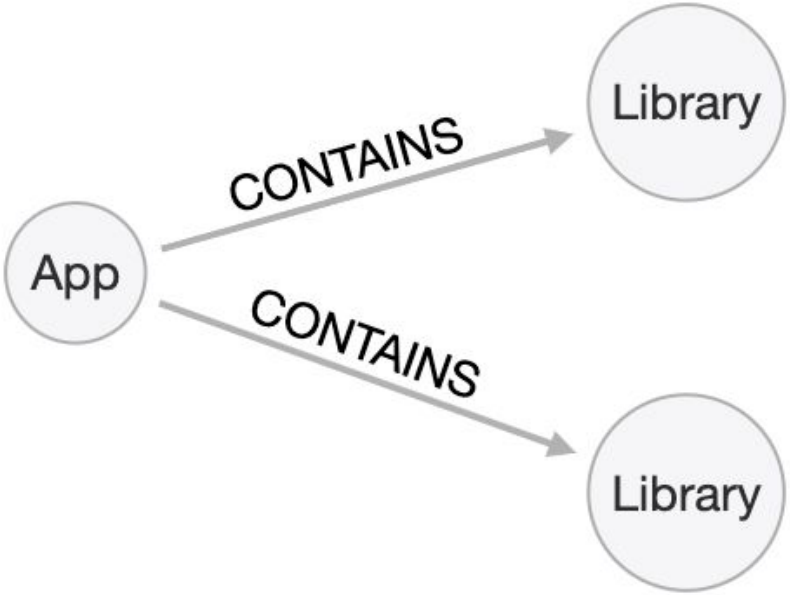
Note: Relationships require space in the graph so minimizing their numbers is always a good thing.

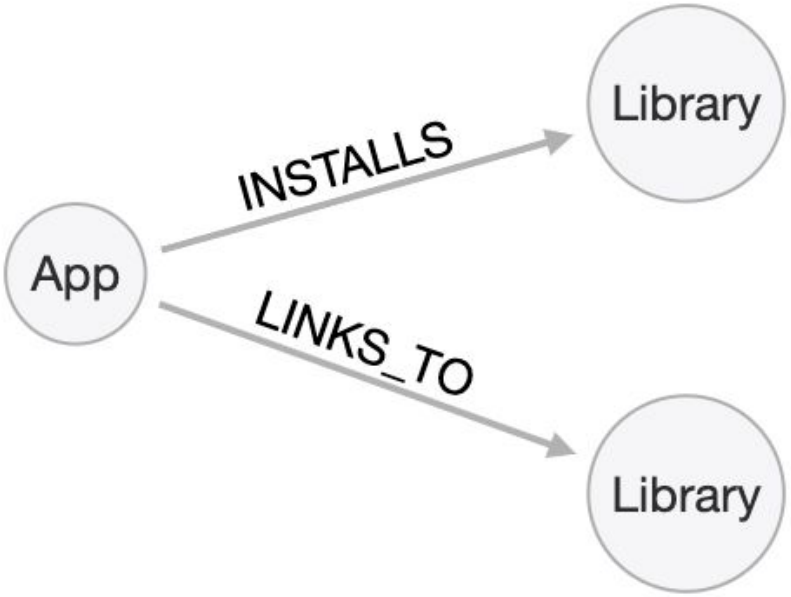
But, symmetric can be used here

For example, the direction of the FOLLOWS relationship on Twitter is significant. It matters who has followed who.

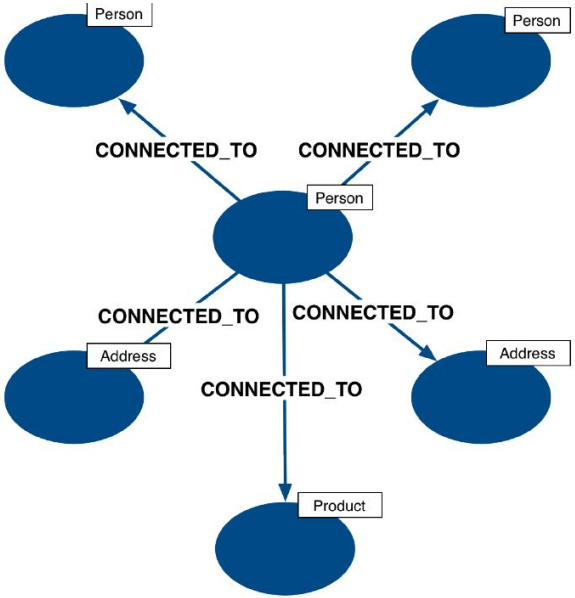


Best practice: Specific relationship types



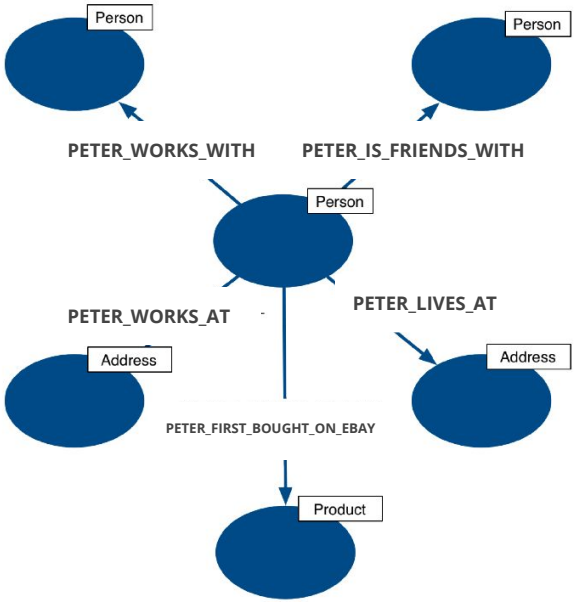


Relationship granularity

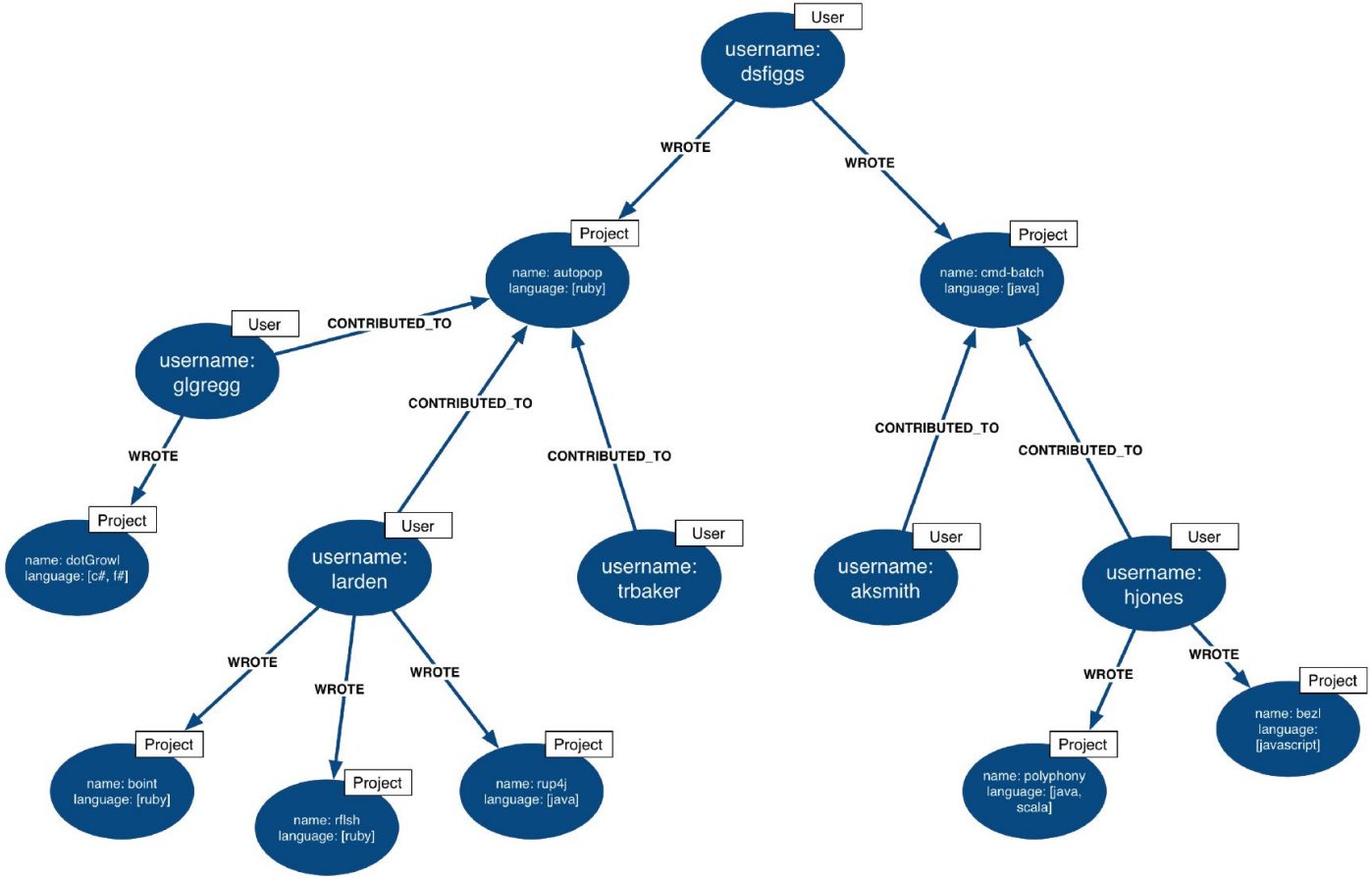


Too little

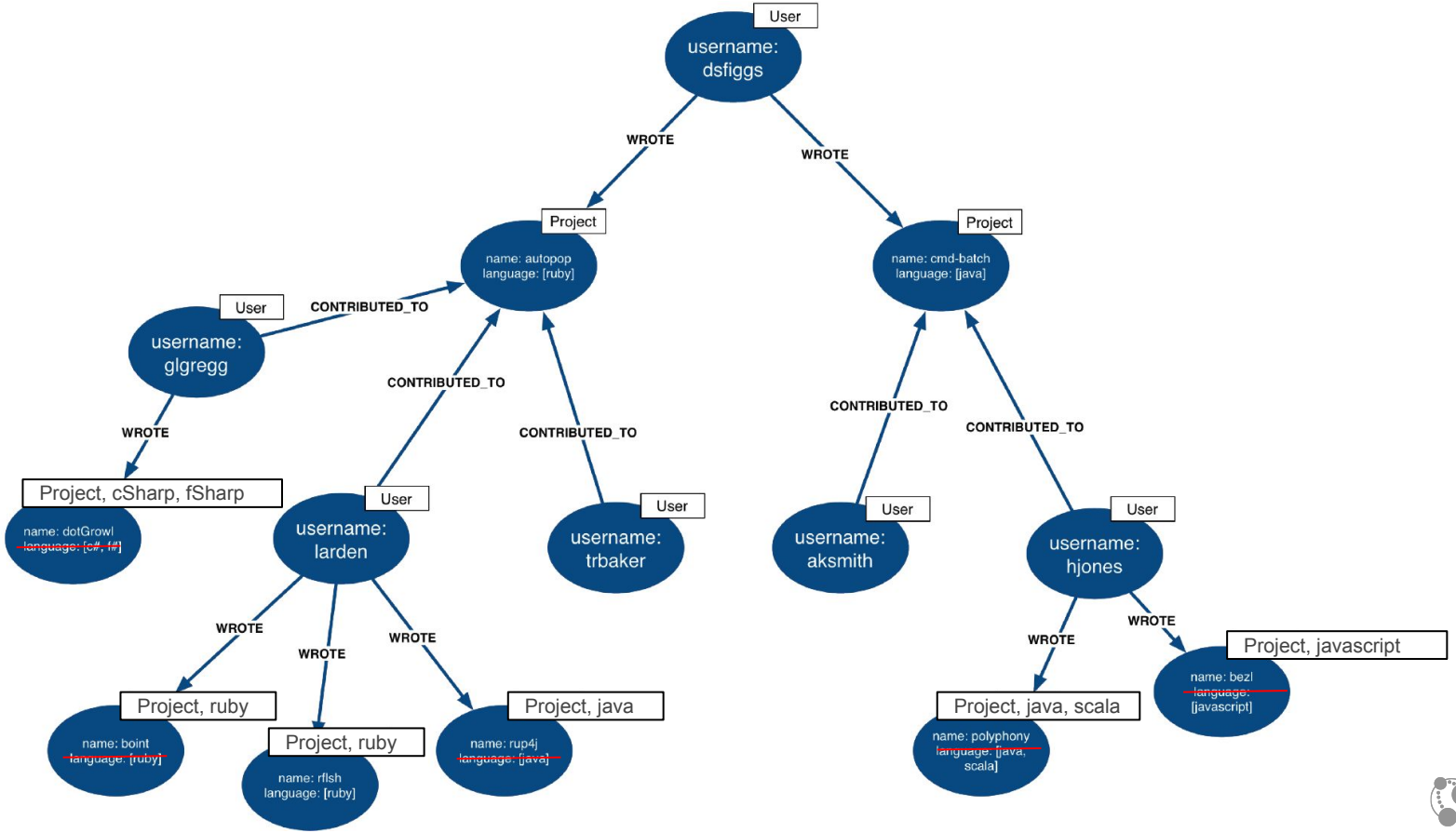
larity



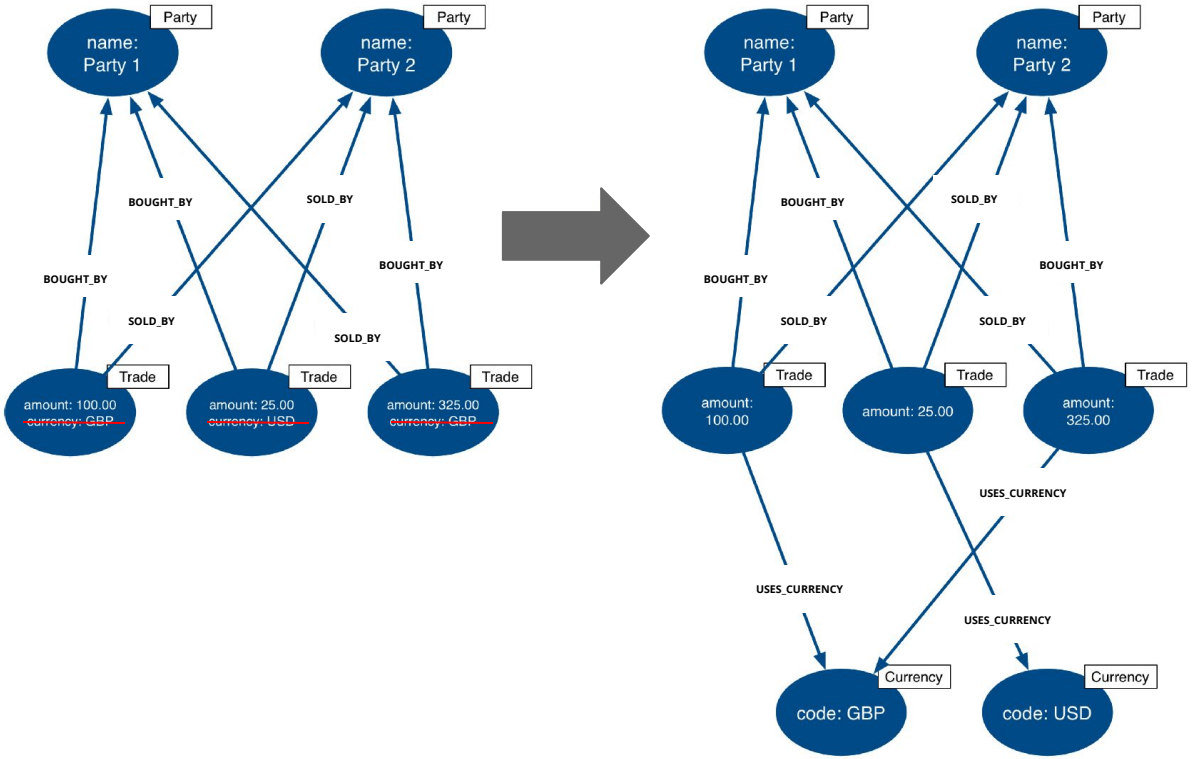
Best practice: Use simple properties



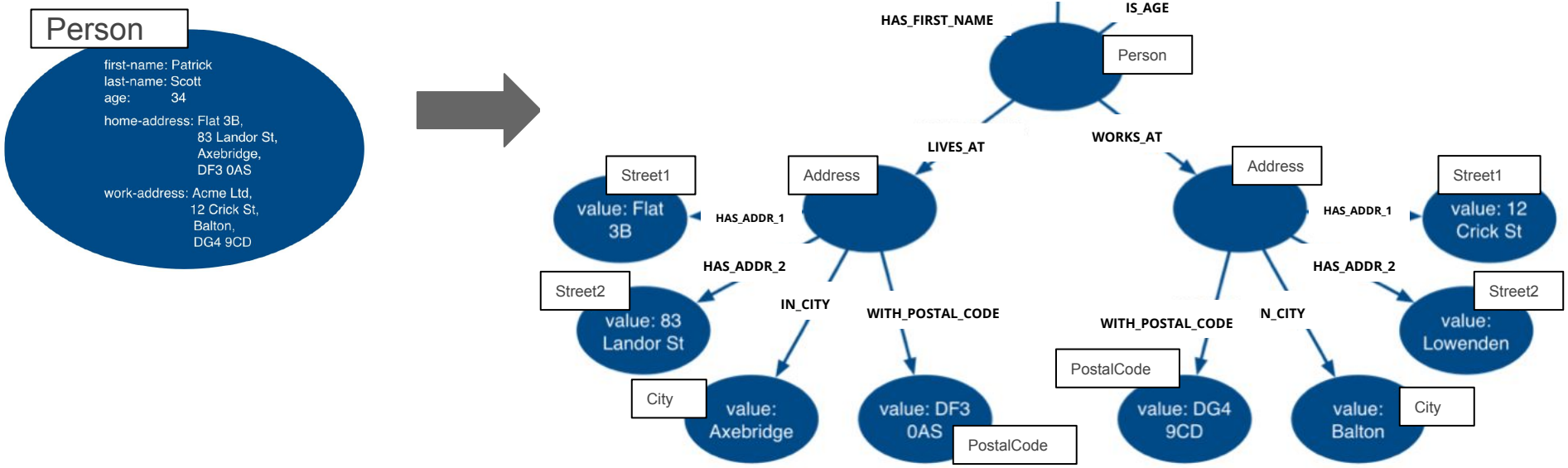
Best practice: Avoid duplication of data



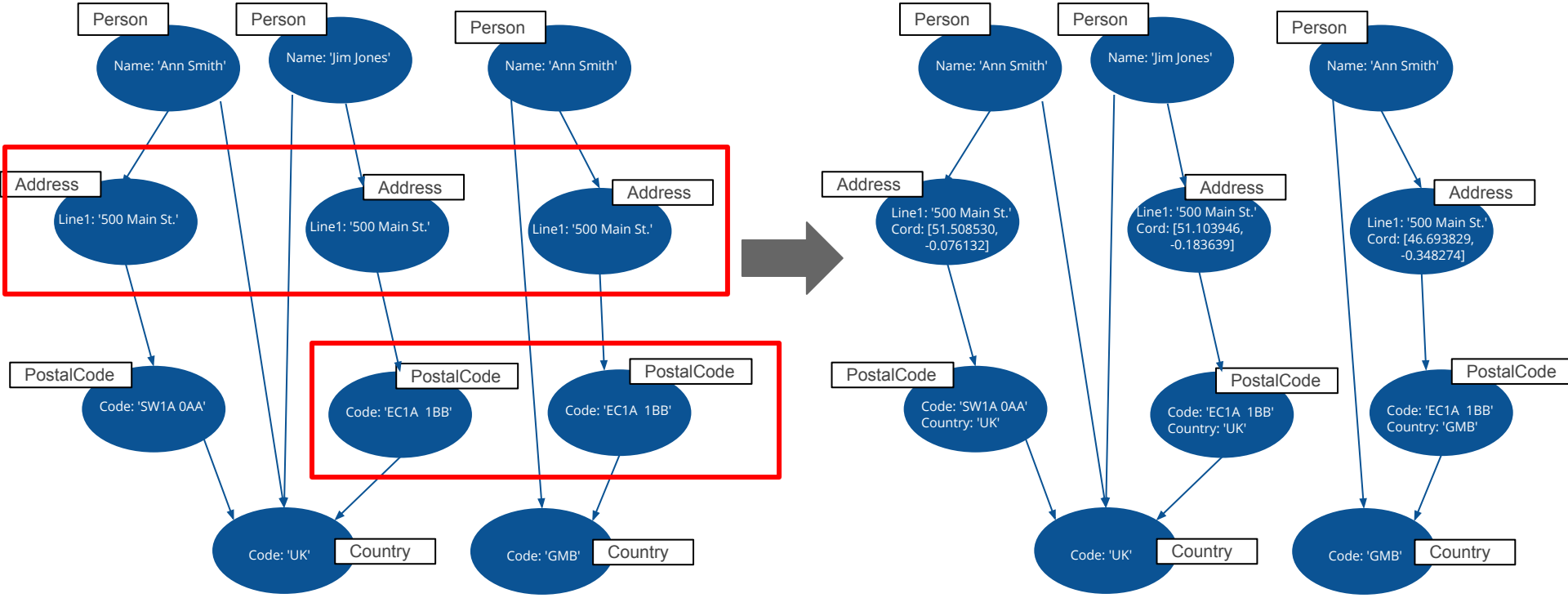
Best practice: Avoid duplication of data



Best practice: Avoid complex types for properties

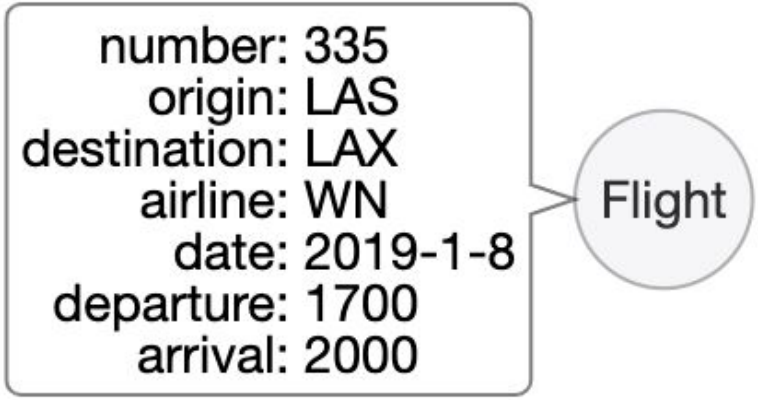


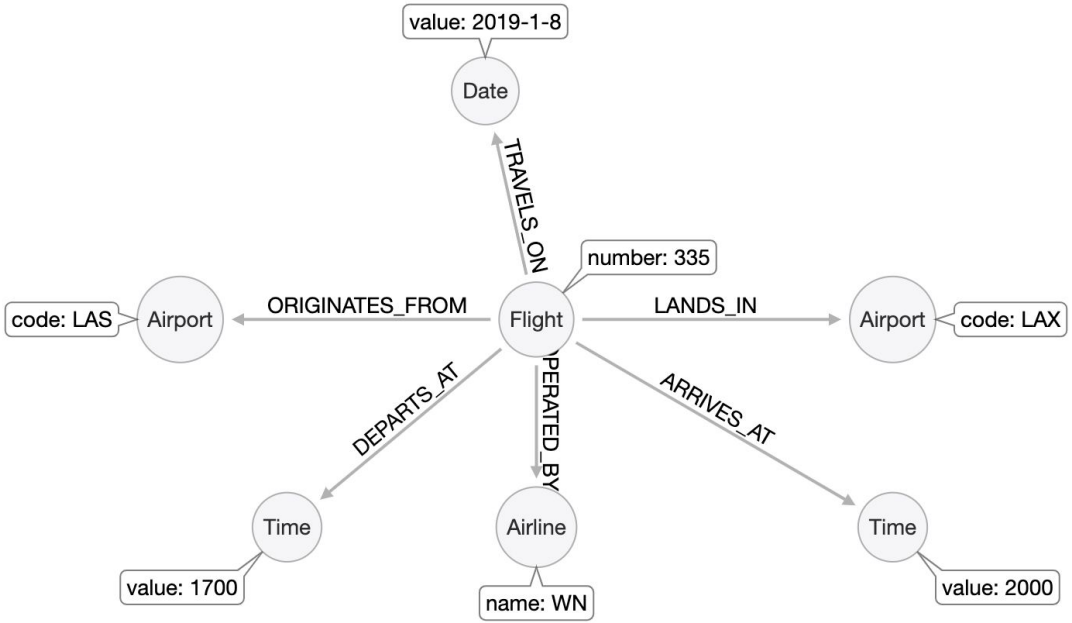
Best practice: Ensure uniqueness of some nodes



64

Best practice: Don't overuse nodes





65

Exercise 5:
Using best practices

Exercise 5: Instructions

1. Continue with Arrows: http://www.apcjones.com/arrows.

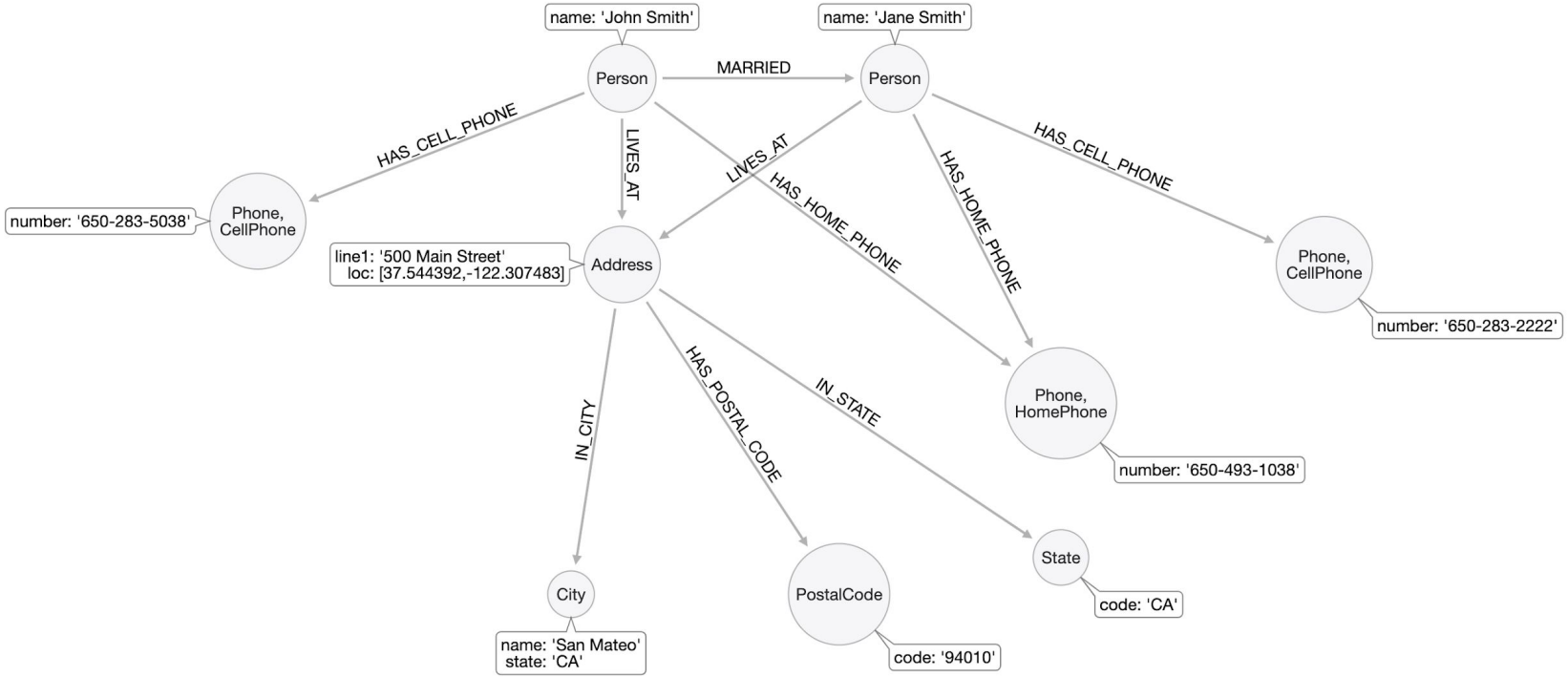
2. Open the model *JohnAndJane*.
   Hint: Use Export Markup to copy paste from **JohnAndJane.htm**.

3. Modify the JohnAndJane model to use some best practices.

4. Save the model as **JohnAndJane-2.htm** and **JohnAndJane-2.svg**.

Exercise 5: Current JohnAndJane model

Exercise 5: Solution

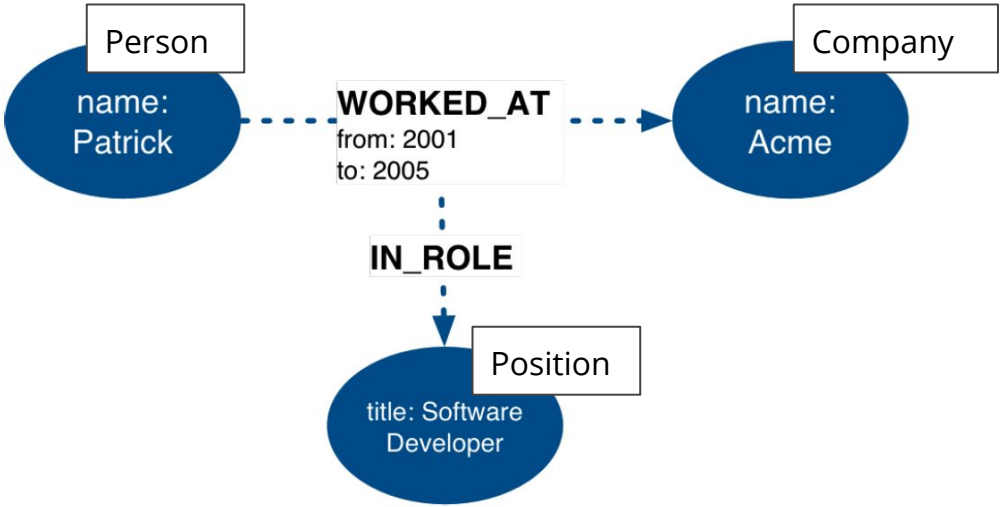


Common graph structures

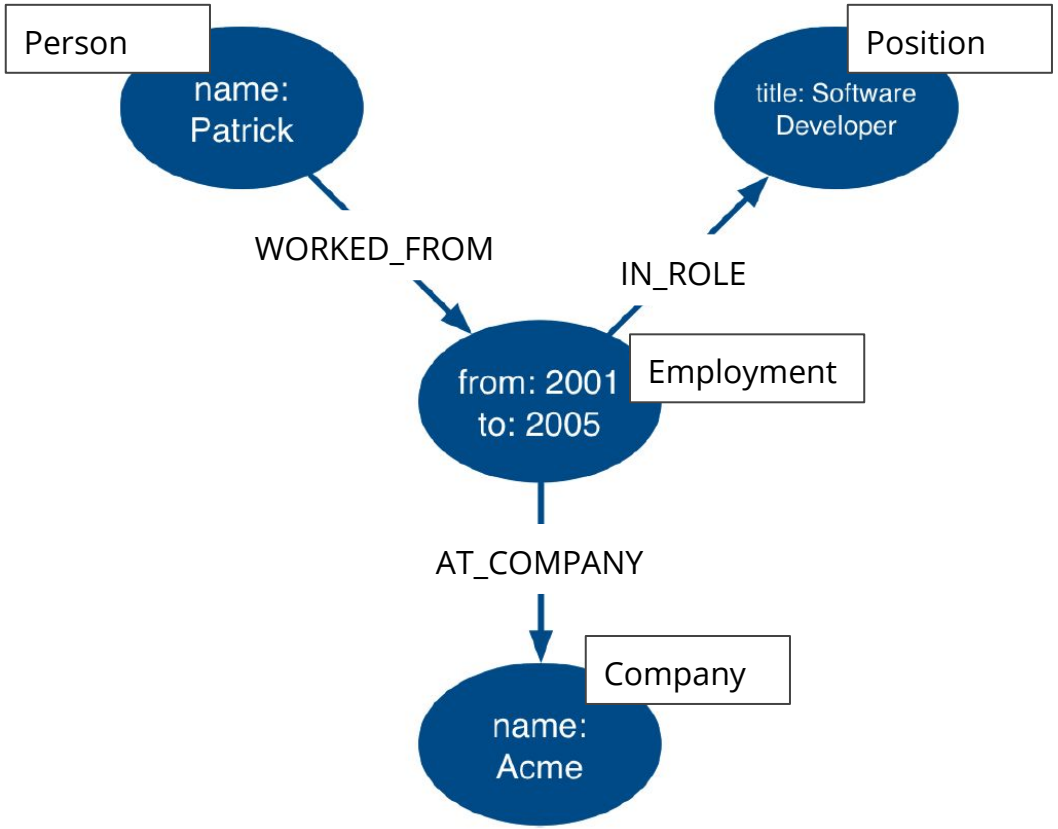
- Intermediate nodes
- Linked lists
- Specialized tree structures
- Multiple structures in a single model
- Versioning

Adding intermediate nodes

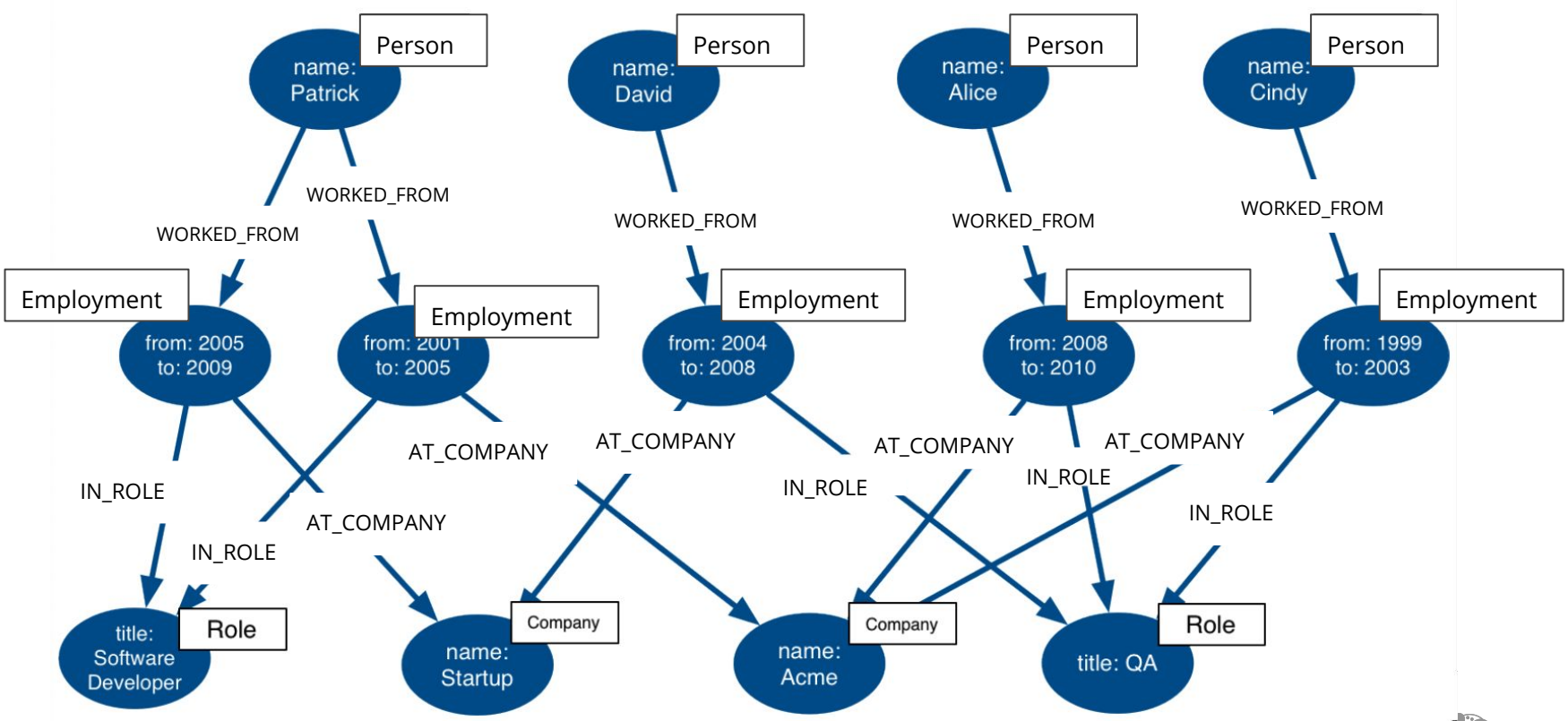
- Connect more than two nodes in a single context.
  - Hyperedges (n-ary relationships)
- When you need to relate something to a relationship.



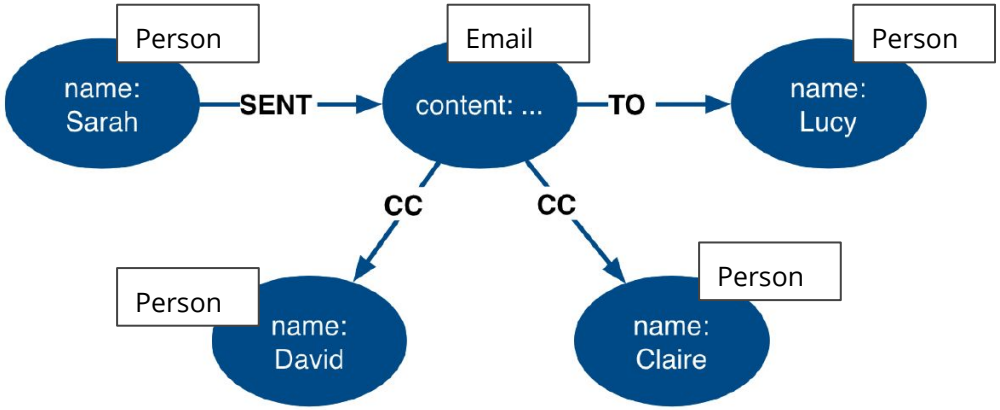
Rich context, multiple dimensions



Dimensions shared between contexts



Intermediate node shared by multiple parties



Intermediate nodes provides flexibility as it allows more than two nodes to be connected in a single context.

But… it can be overkill and could have an impact on performance.

Linked lists

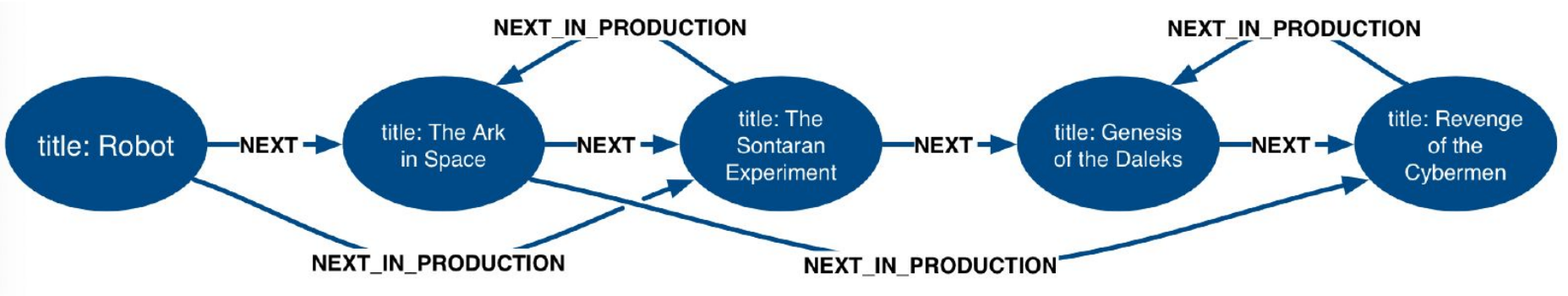
- Entities are linked in a sequence.
- Useful when you need to traverse the sequence.
  - next
  - previous
- You may need to identify:
  - first
  - last
- Examples:
  - Event stream
  - Episodes of a TV series
  - Job history

Simple linked lists

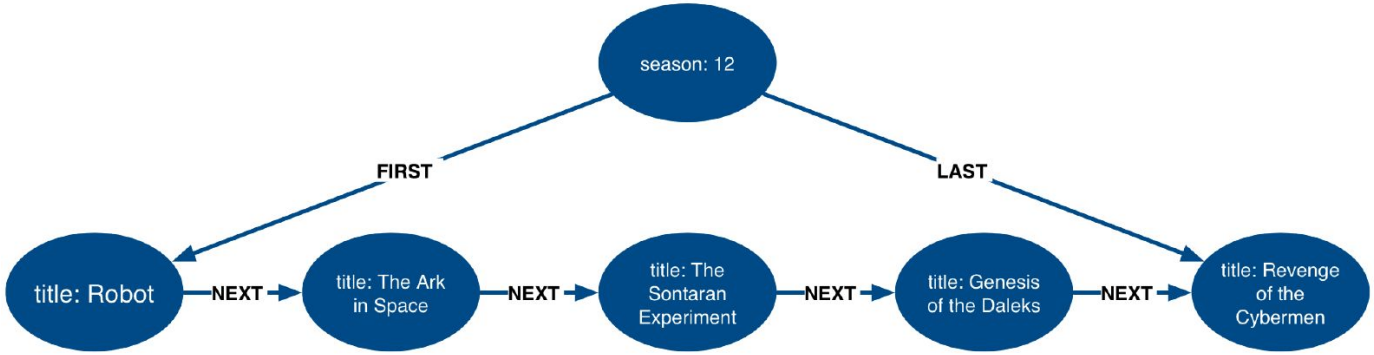
Episodes of the Dr.Who series:

Doubly linked list:

Interleaved linked list



Pointers to head and tail



Use cases:

- Add episodes as they are broadcast.
  - Maintain pointer to first and last episodes.
- Find all episodes broadcast so far.
- Find latest episode broadcast so far.

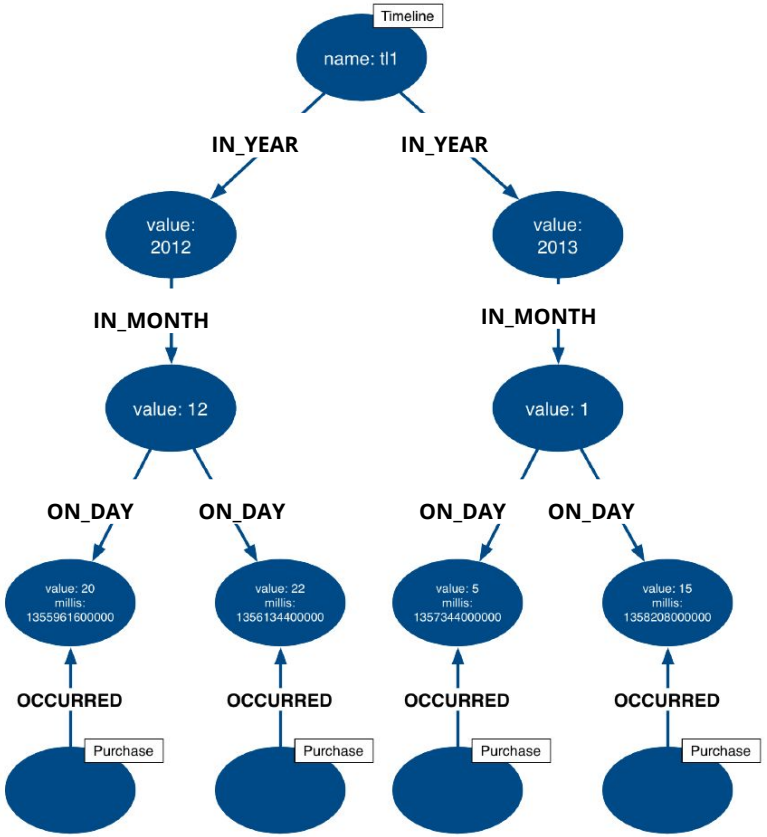
Specialized tree structures

An index is a graph:

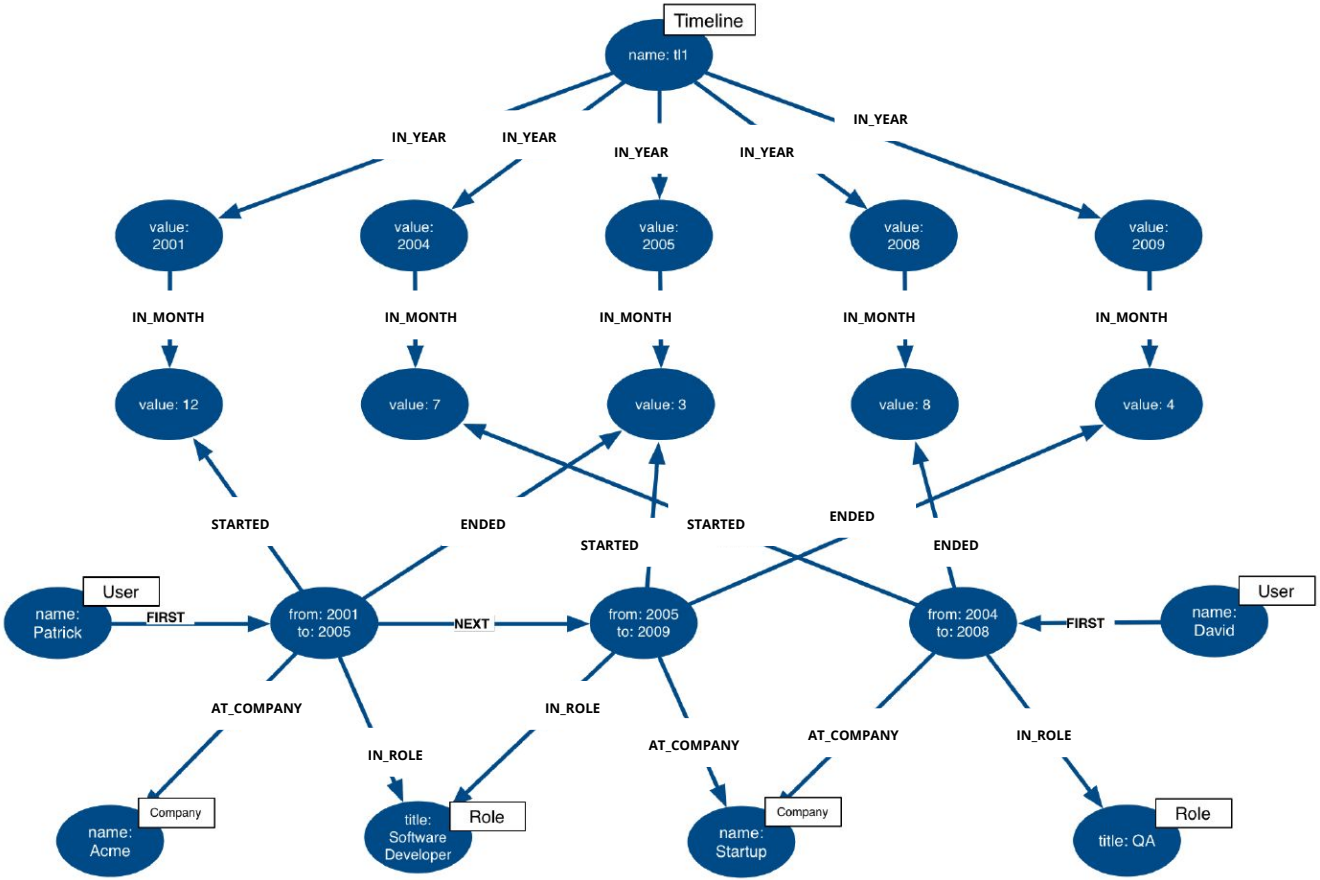
- b-tree for binary search
- r-tree
  - Spatial search
  - Multi-dimensional data
- Timeline tree
  - Discrete events
    - No natural relationship to other events
  - Search for events at differing levels of granularity between:
    - 2 days
    - 2 months
    - 2 minutes

Example: Timeline tree

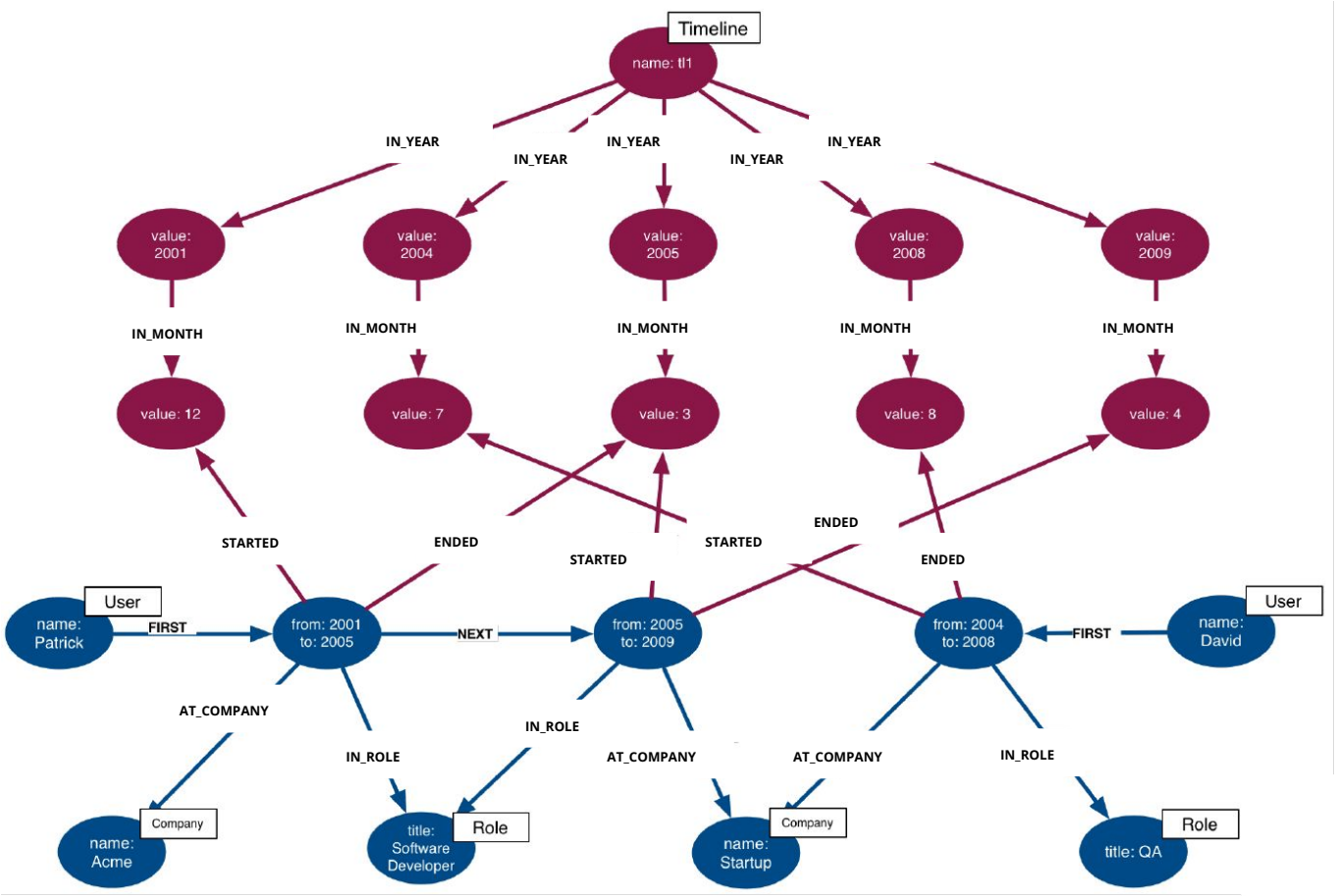
Retrieve all purchases two dates.



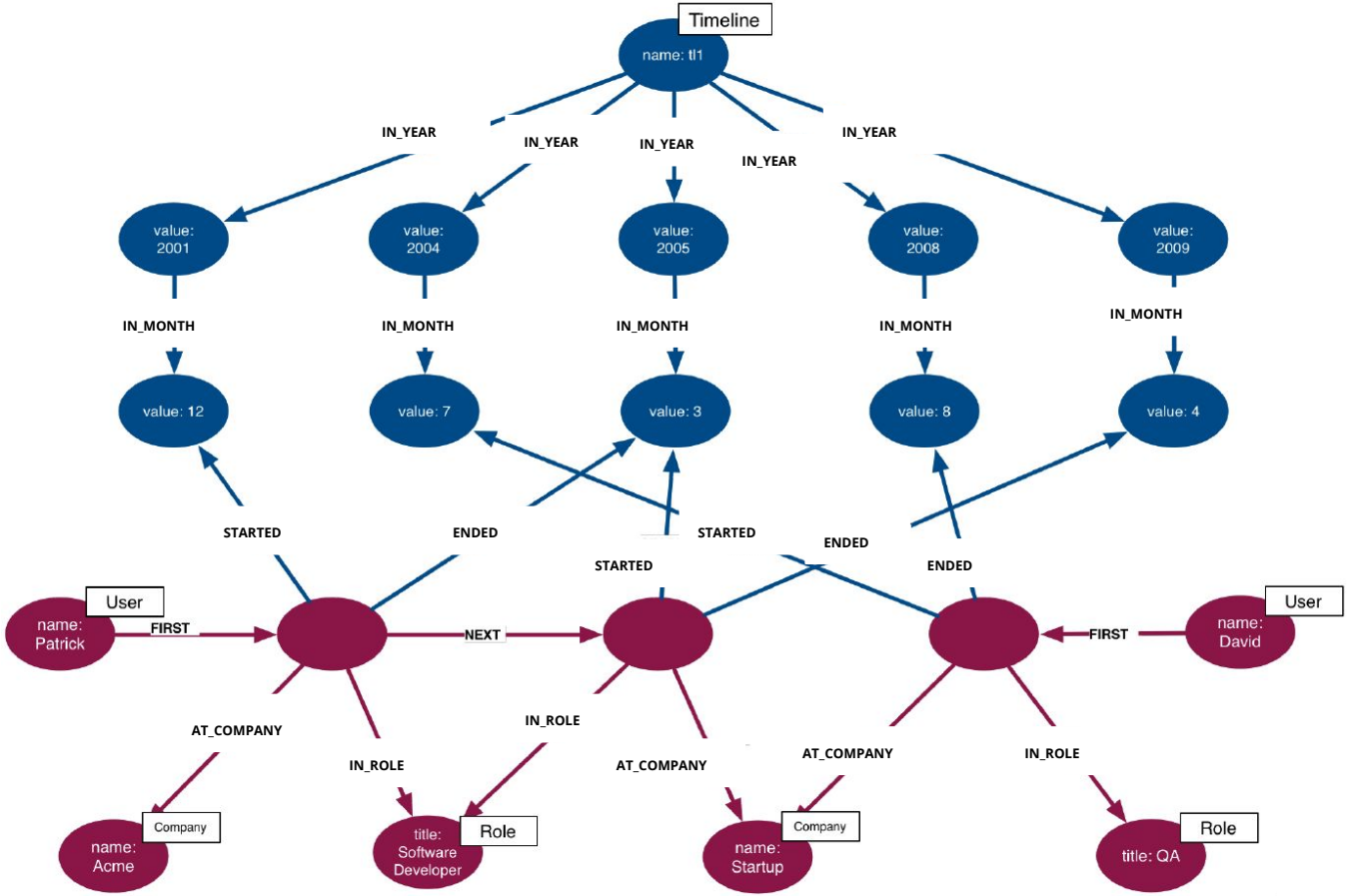
Multiple structures in a single model



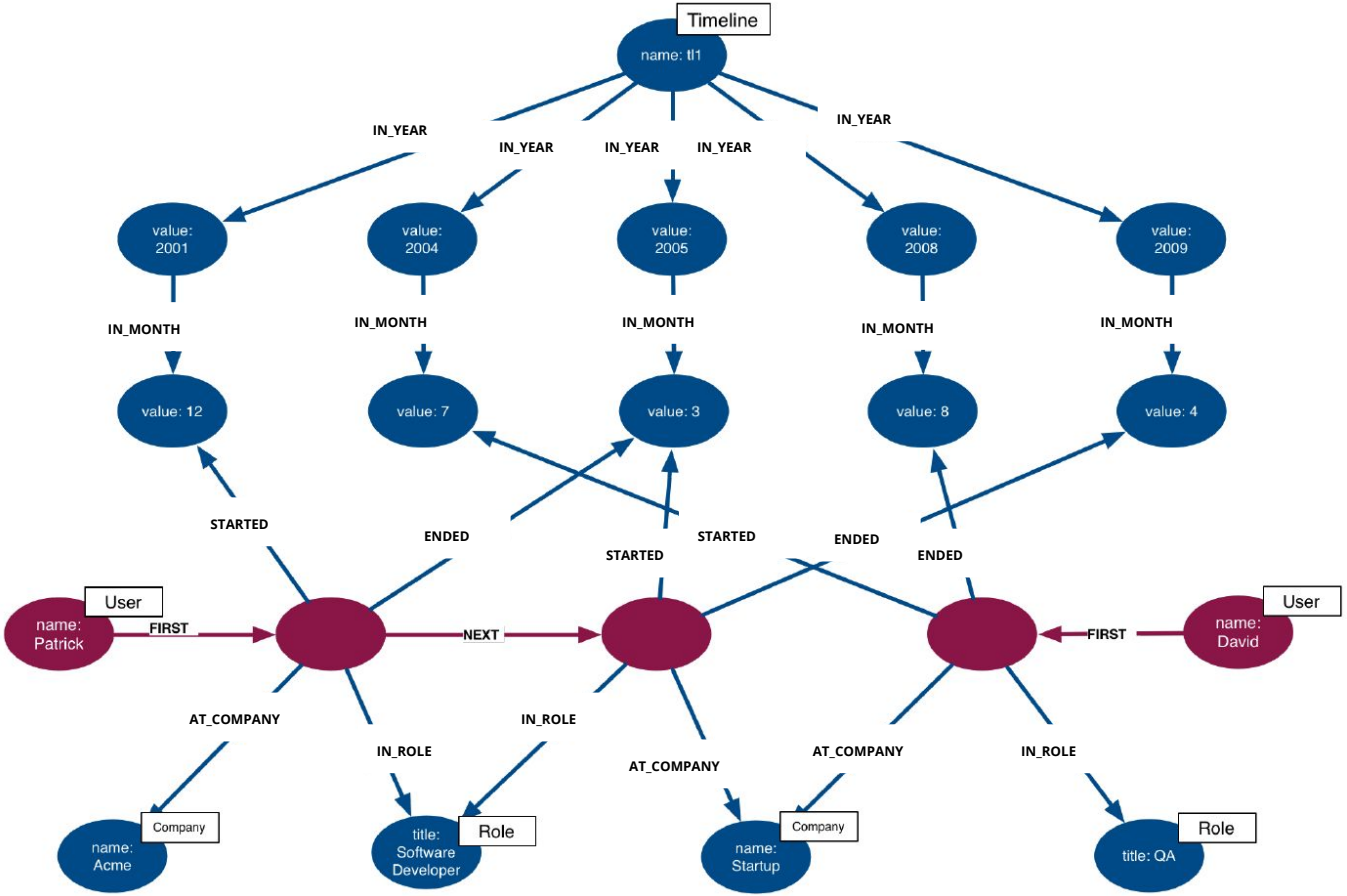
Timeline structure in the model



Intermediate nodes in the model



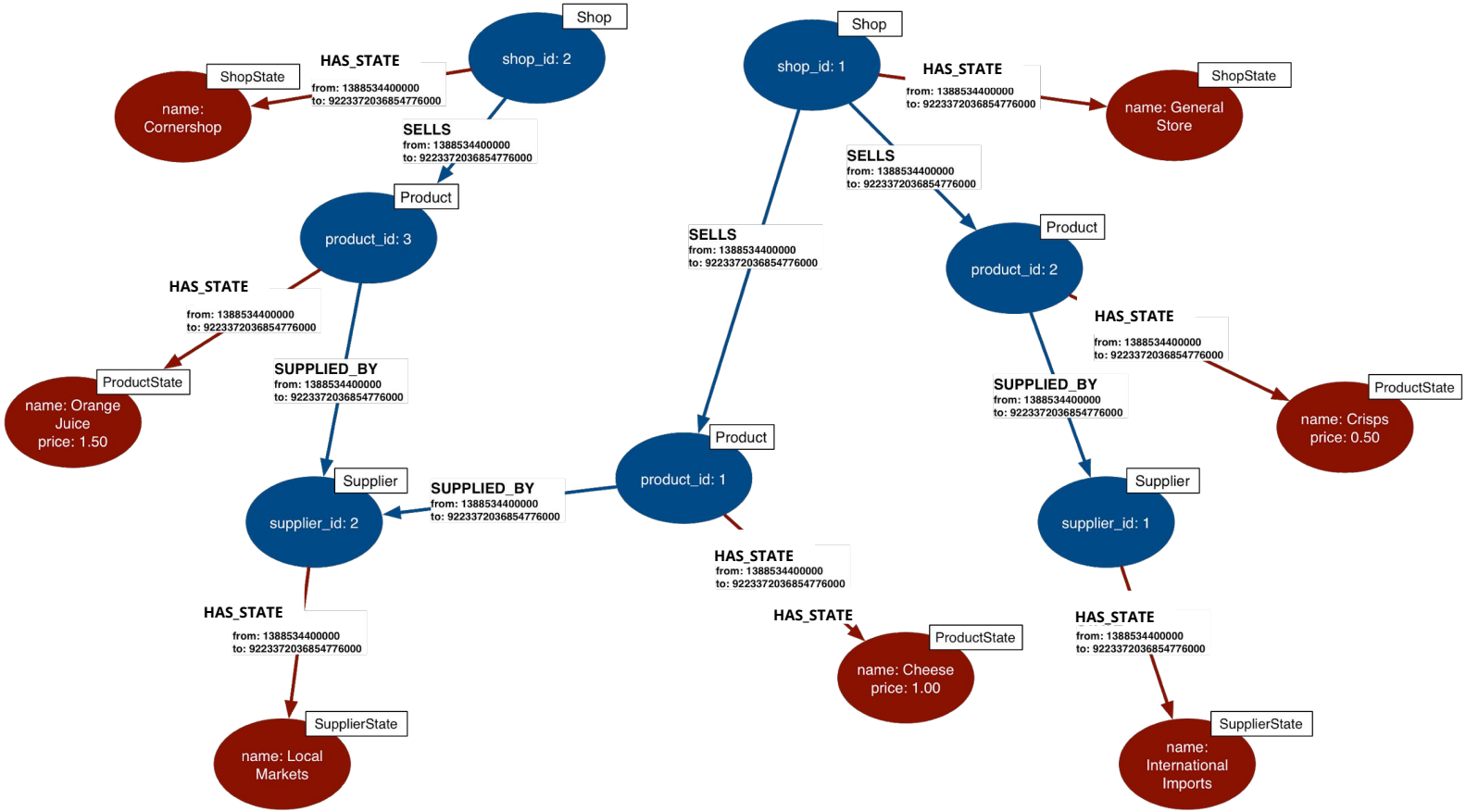
Linked lists in the model



Versioning

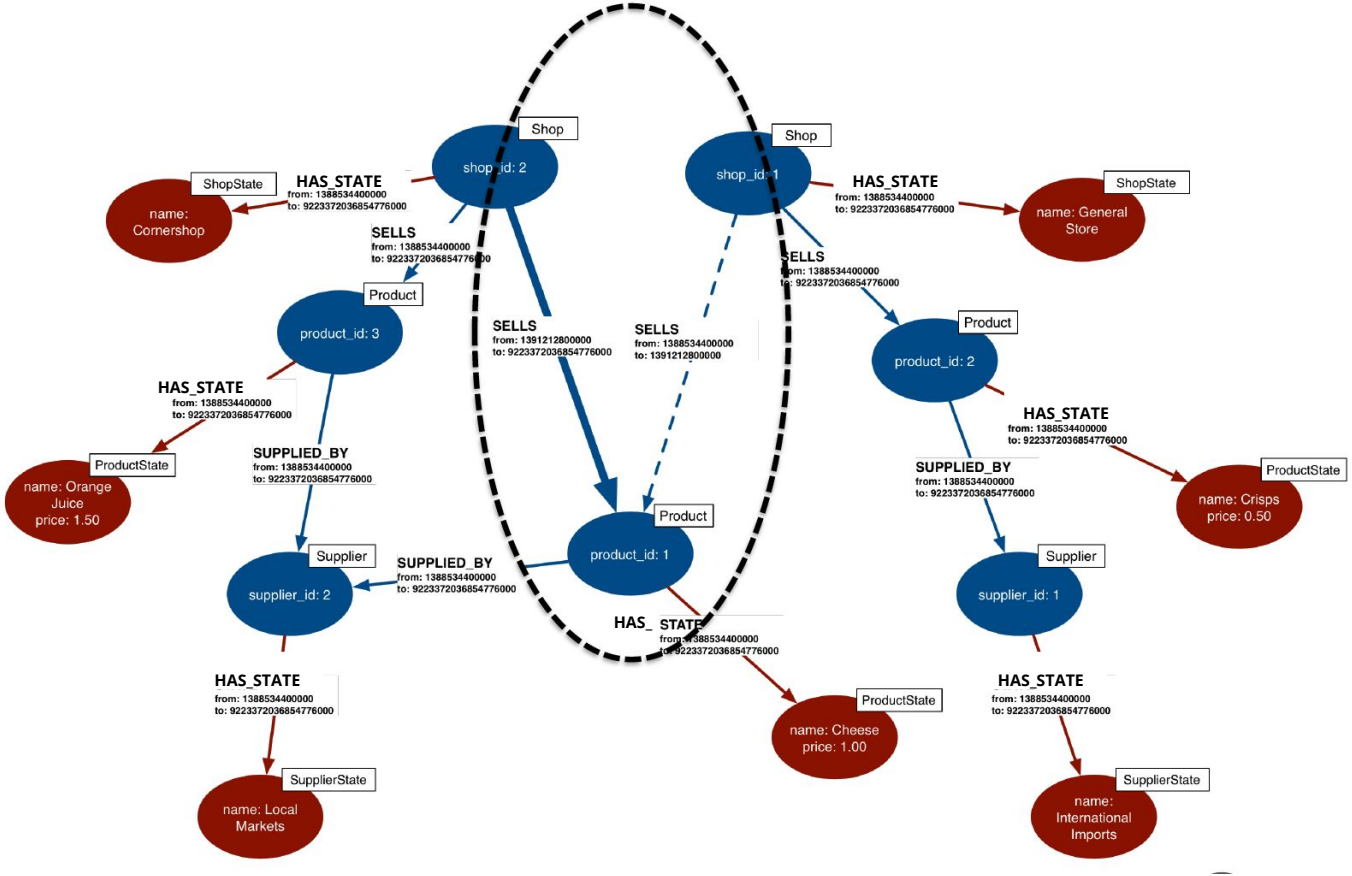
- A versioning model is time-based
  - Universal versioning schema
  - Discrete, continuous sequence
- Separate structure from state
  - Structure
    - Identity nodes as placeholders
    - Timestamped identity relationships
  - State
    - State nodes which are a snapshot of the entity state
    - Timestamped state relationships

Versioning example (1)



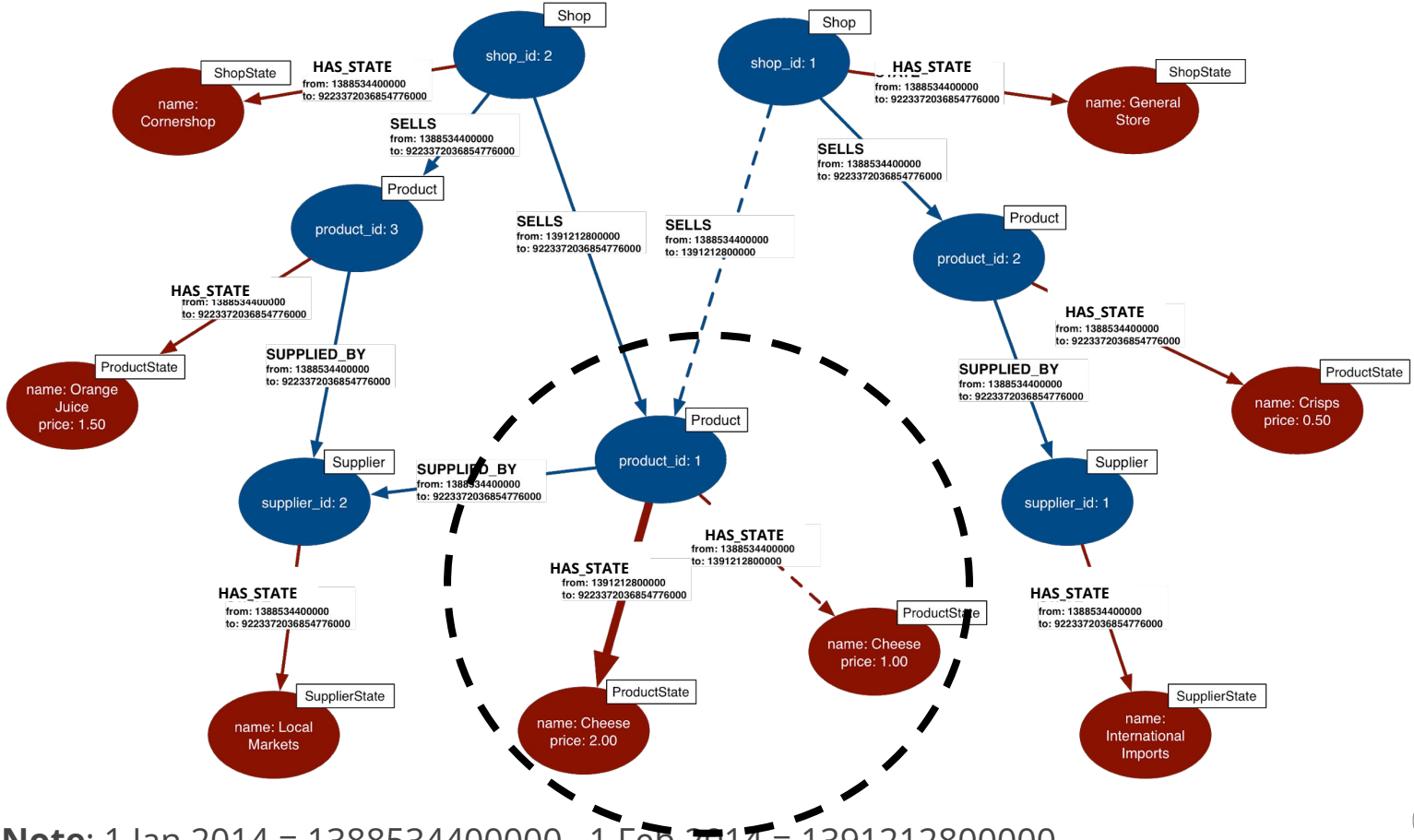
Note: 1 Jan 2014 = 1388534400000

Versioning example (2)



Note: 1 Jan 2014 = 1388534400000, 1 Feb 2014 = 1391212800000

Versioning example (3)



Note: 1 Jan 2014 = 1388534400000, 1 Feb 2014 = 1391212800000

Exercise 6:
Model an online course

Exercise 6: Instructions

Suppose you want to model an online course. A student enrolls in the course. The student starts the course which has 3 lessons. When the student completes a lesson, they go the the next lesson in the course. When the student completes the last lesson successfully, the student is sent a certificate.

Steps:

1. Identify the entities and relationships of this model.

2. Use Arrows: http://www.apcjones.com/arrows to create the model. You can use a single student and a single course for your model.

3. Save the model as **Course.htm** and **Course.svg**.

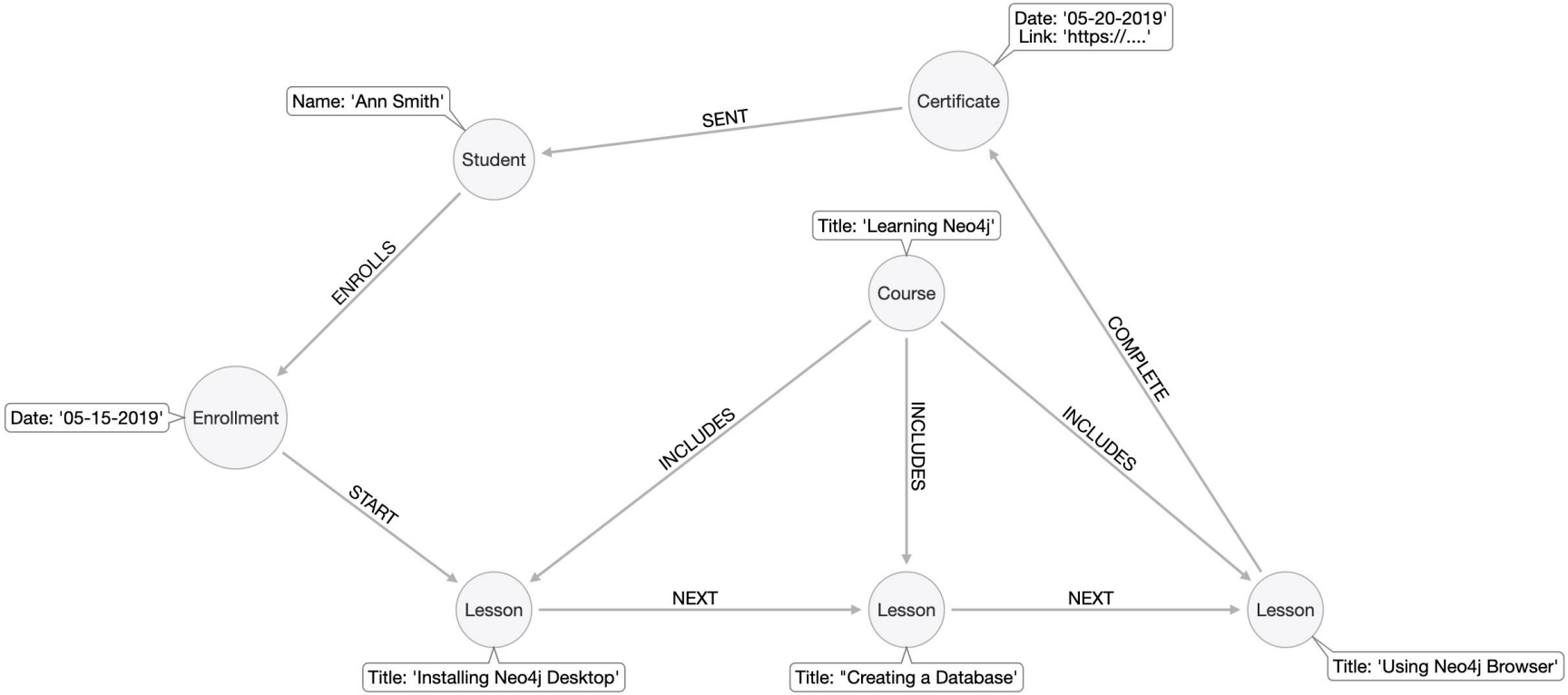
Exercise 6: Solution

Suppose you want to model an online course. A student enrolls in the course. The student starts the course which has 3 lessons. When the student completes a lesson, they go the the next lesson in the course. When the student completes the last lesson successfully, the student is sent a certificate.

Entities:
Student, Course, Lesson, Enrollment, Certificate

Relationships:
ENROLLS, START, NEXT, PREVIOUS, INCLUDES, COMPLETE, SENT

Exercise 6: Solution

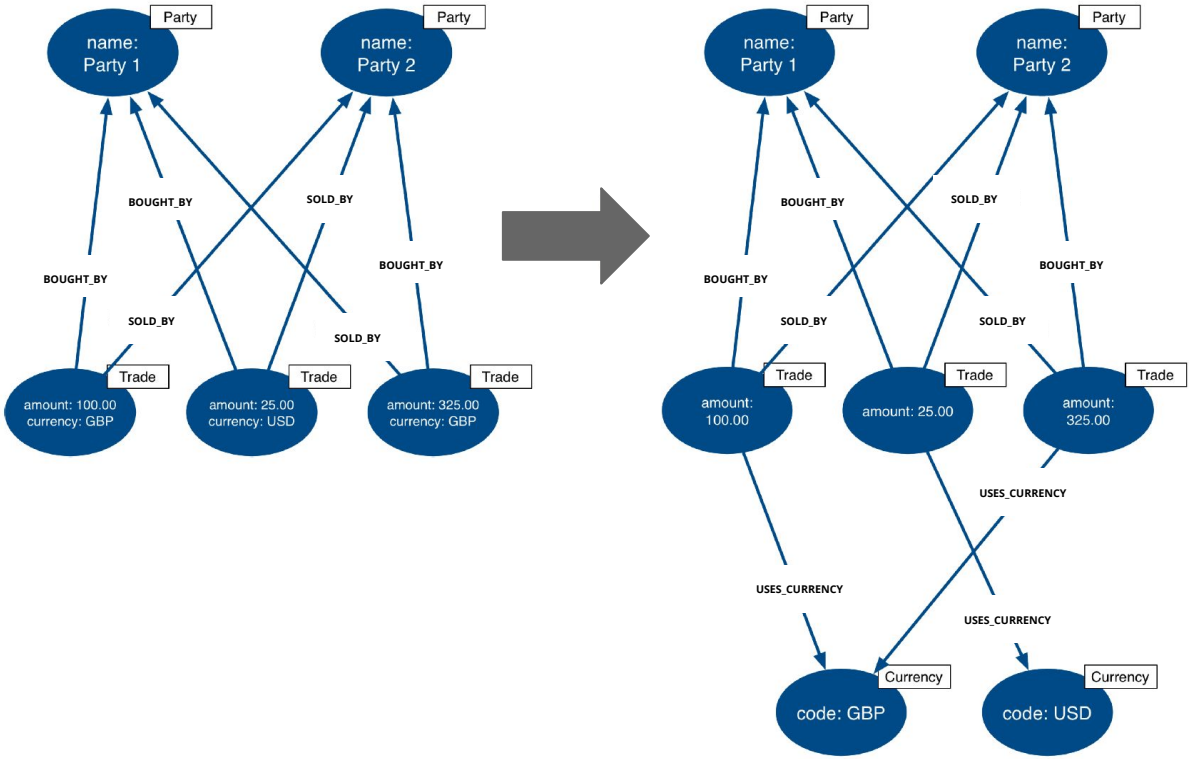


Evolving the graph data model

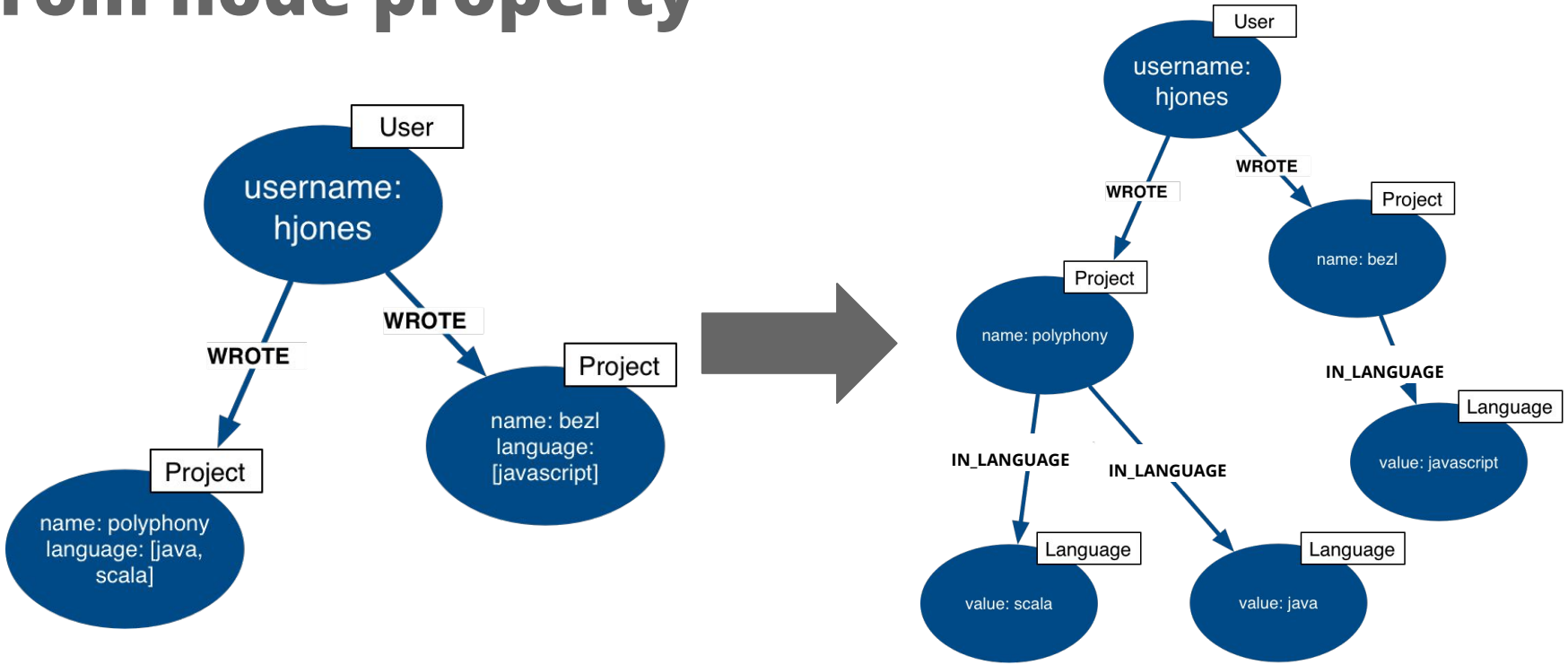
- Restructure graph without changing informational semantics
- Why refactor?
  - Improve design
  - Enhance performance
  - Accommodate new functionality
  - Enable iterative and incremental development of data model
- Examples:
  - Extract a node from a property (Address)
  - Extract node from an array property (Skills)
  - Extract node from a relationship (SENT_EMAIL)
  - Extract relationship from property (City)

Note: Refactoring must be done with developers to measure impact of changes.

Example: Extract node from a property

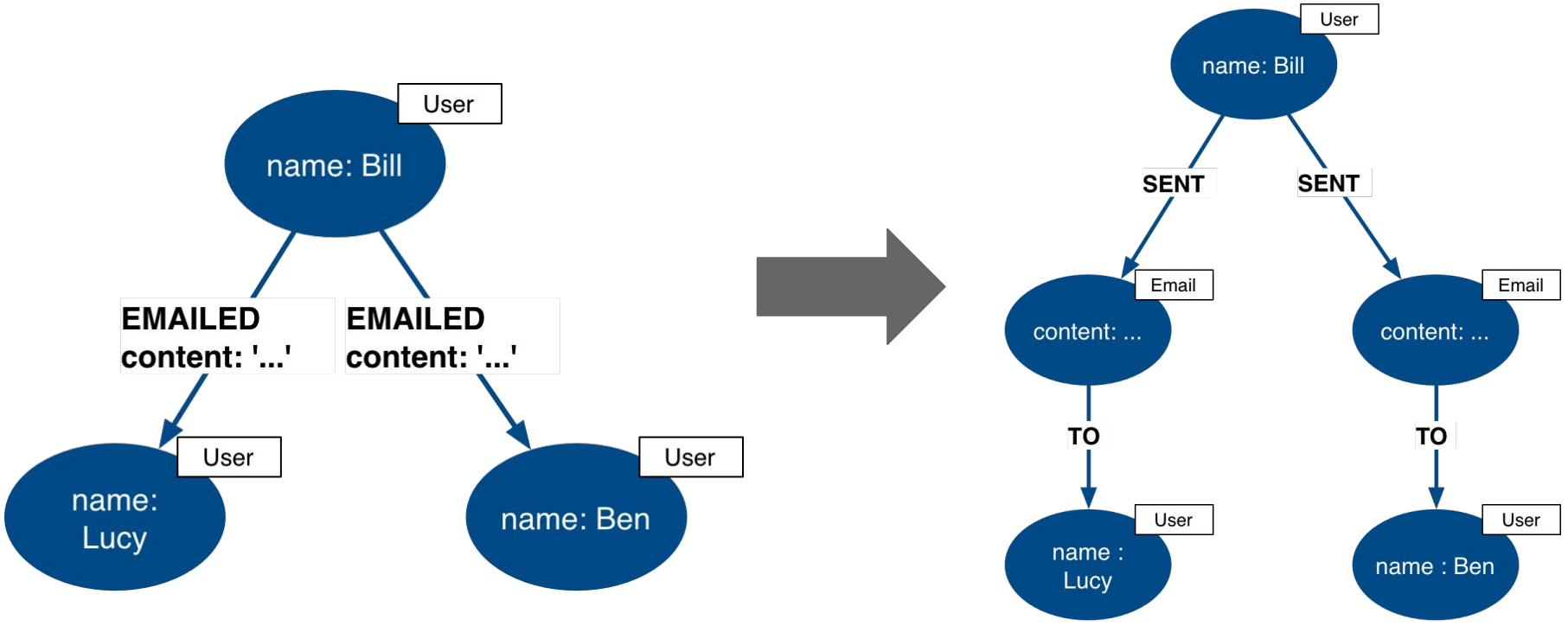


Example: Extract node and relationship from node property



Note: Refactoring must be done with developers to measure impact of changes.

Example: Extract node and relationship from relationship property



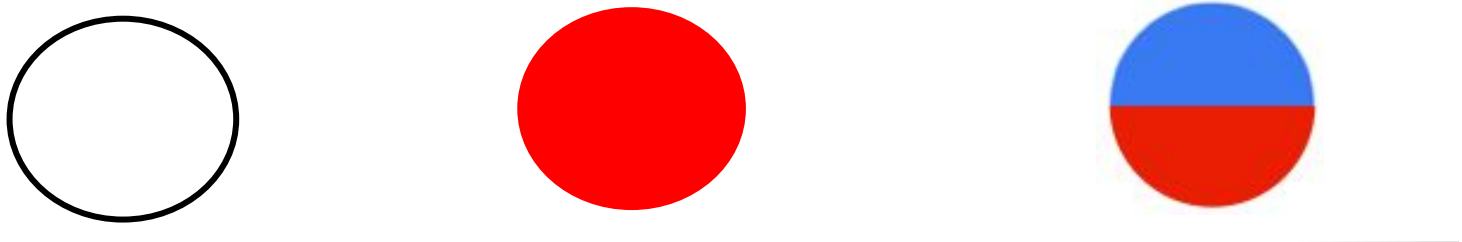
Top 4 modeling tips from the experts

1. One label per node
2. Be specific with relationship types
3. Avoid unnecessary relationships
4. Be mindful of symmetric relationships

One label per node

Person

Employee



No label

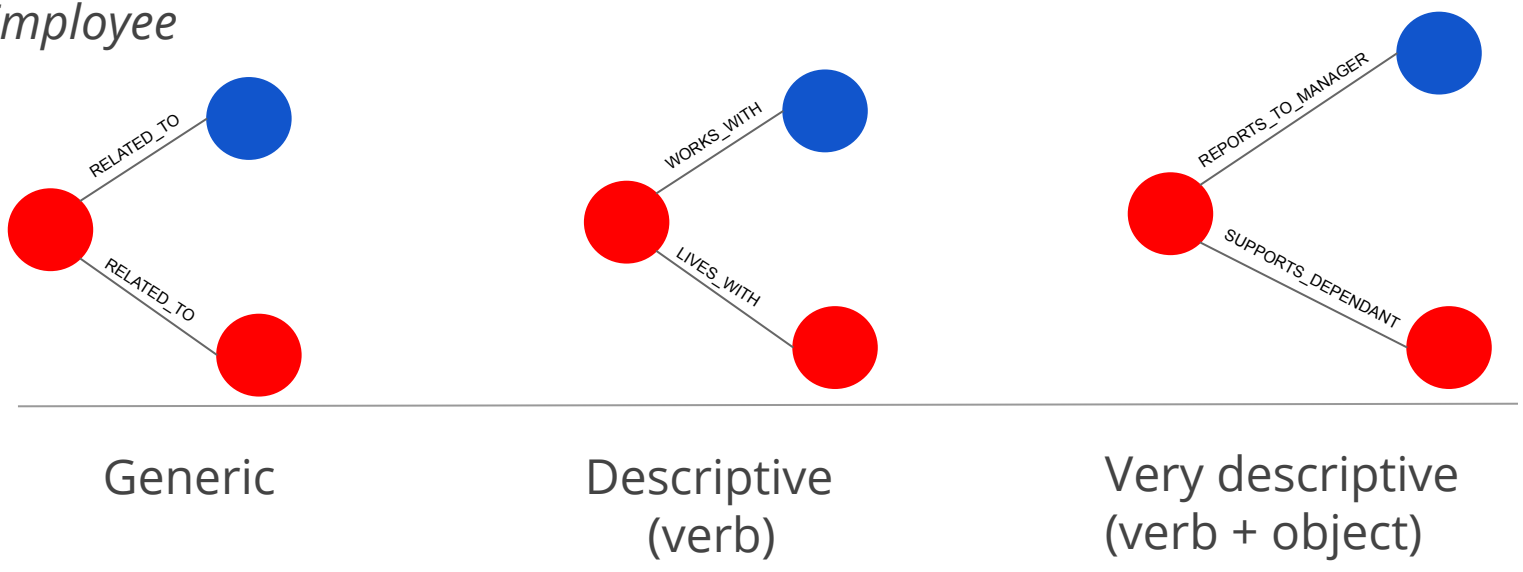
One label
Person

Multiple labels
Person
Employee

Be specific with relationship types

Person

Employee



Generic

Descriptive (verb)

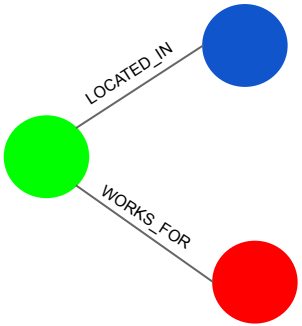
Very descriptive (verb + object)

Avoid unnecessary relationships

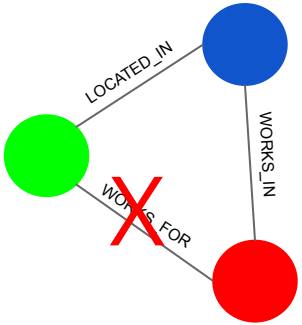
Person

Company

Location



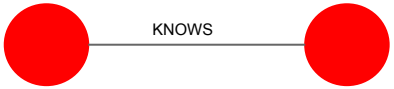
Good model



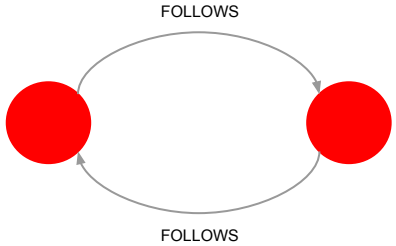
Redundant relationship

Be mindful of symmetric relationships

Person



Facebook



Twitter

KNOWS relationship is symmetrical.

Even though direction is always stored, relationships can be traversed as undirected.

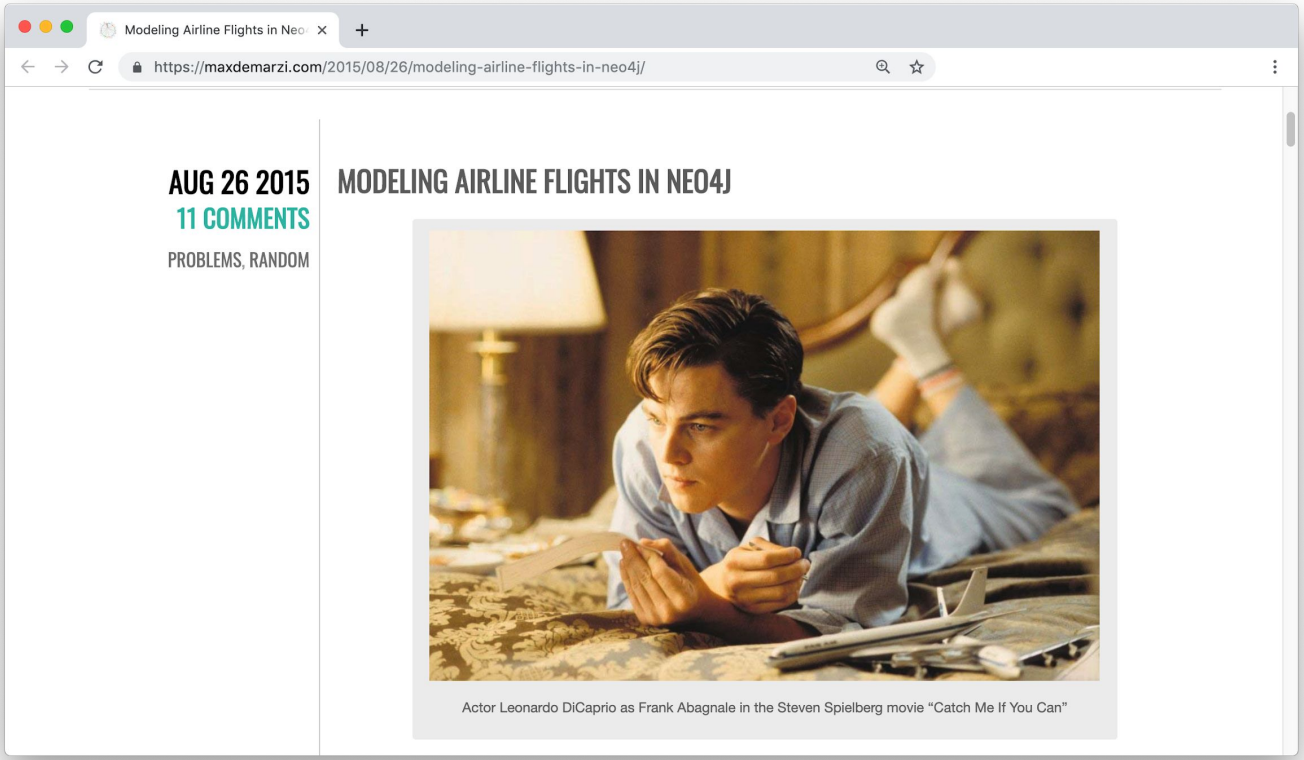
FOLLOWS relationship is asymmetrical.

Direction matters. Two users may follow each other, or one may follow the other but not visa versa.

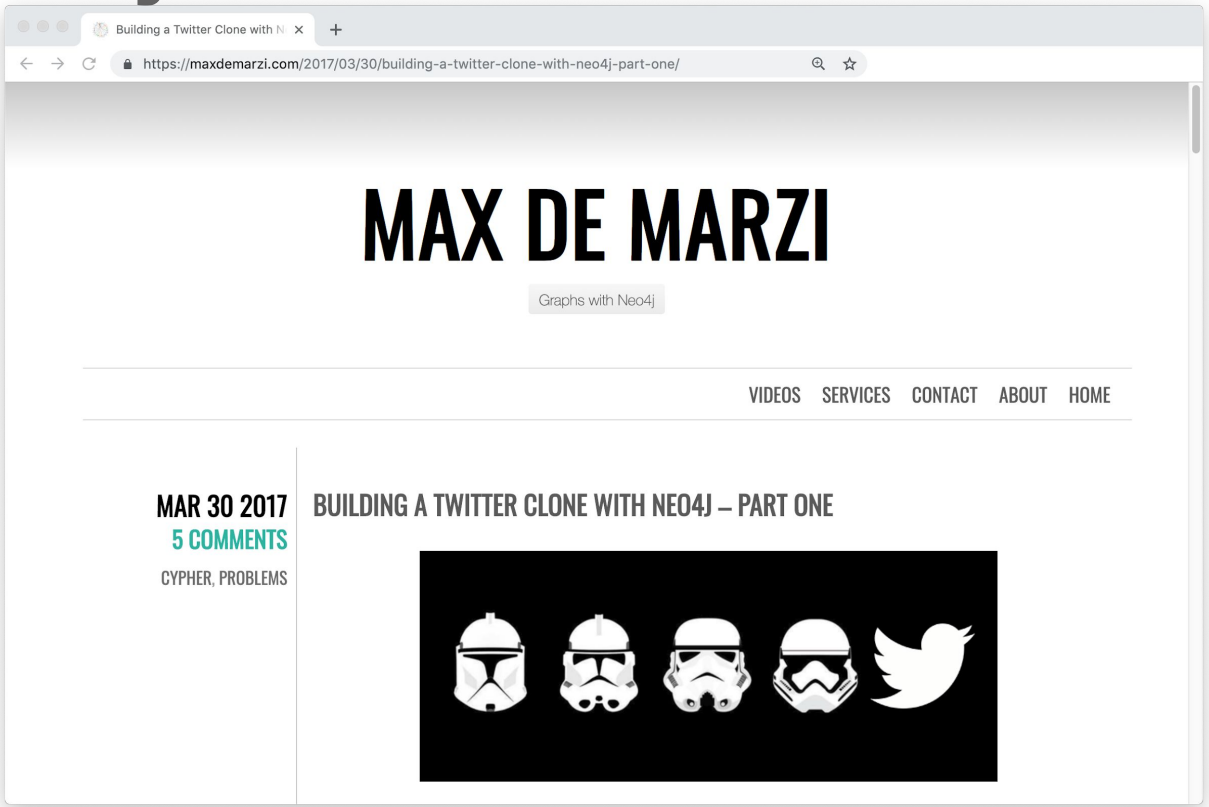
Some Neo4j use cases

- Tracking airline flights
- Activity feed
- Fraud detection
- Event modeling
- Time series

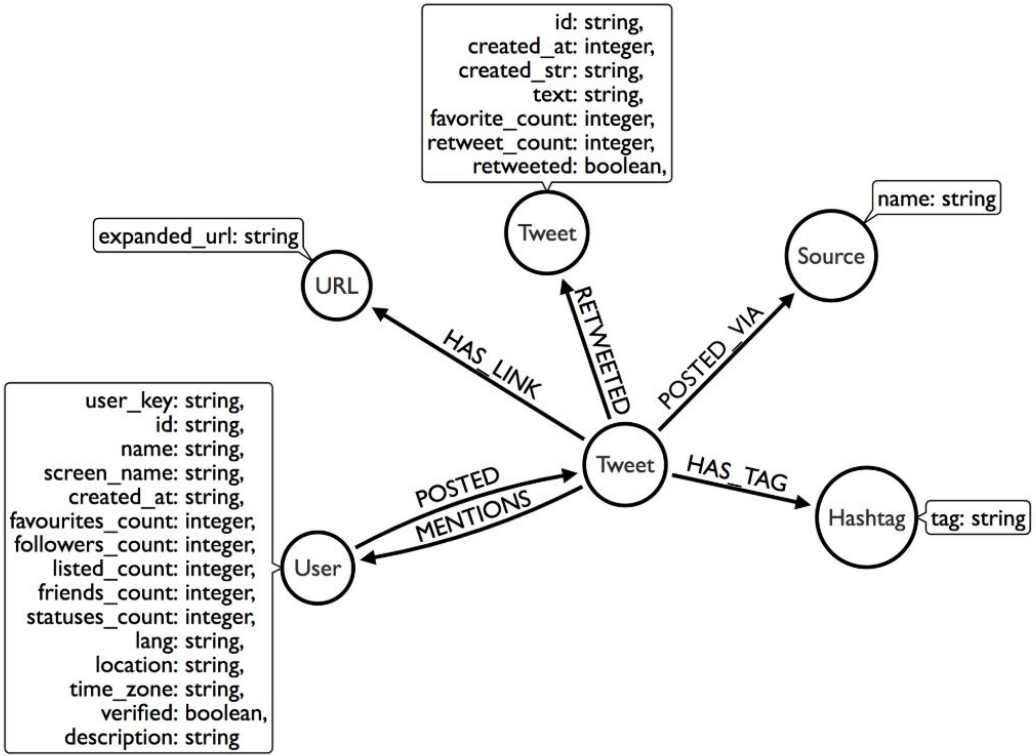
Tracking airline flights: Modeling airline flights in Neo4j



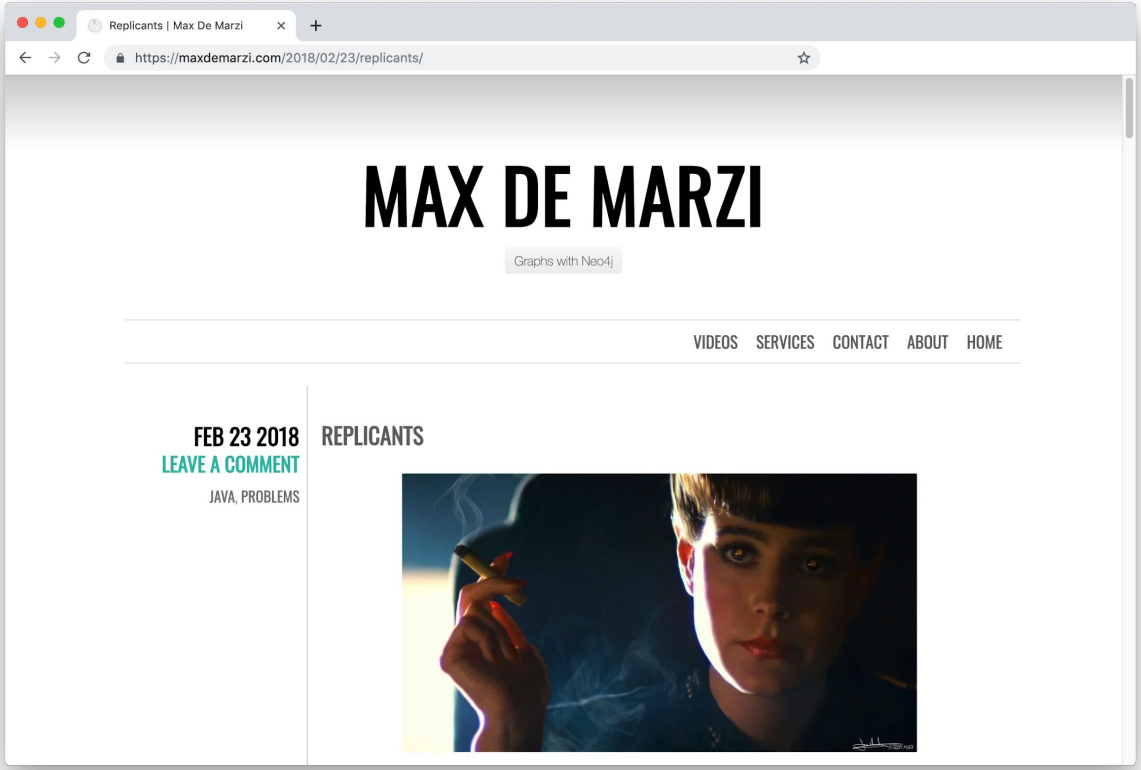
Activity feed: Building a Twitter Clone with Neo4j



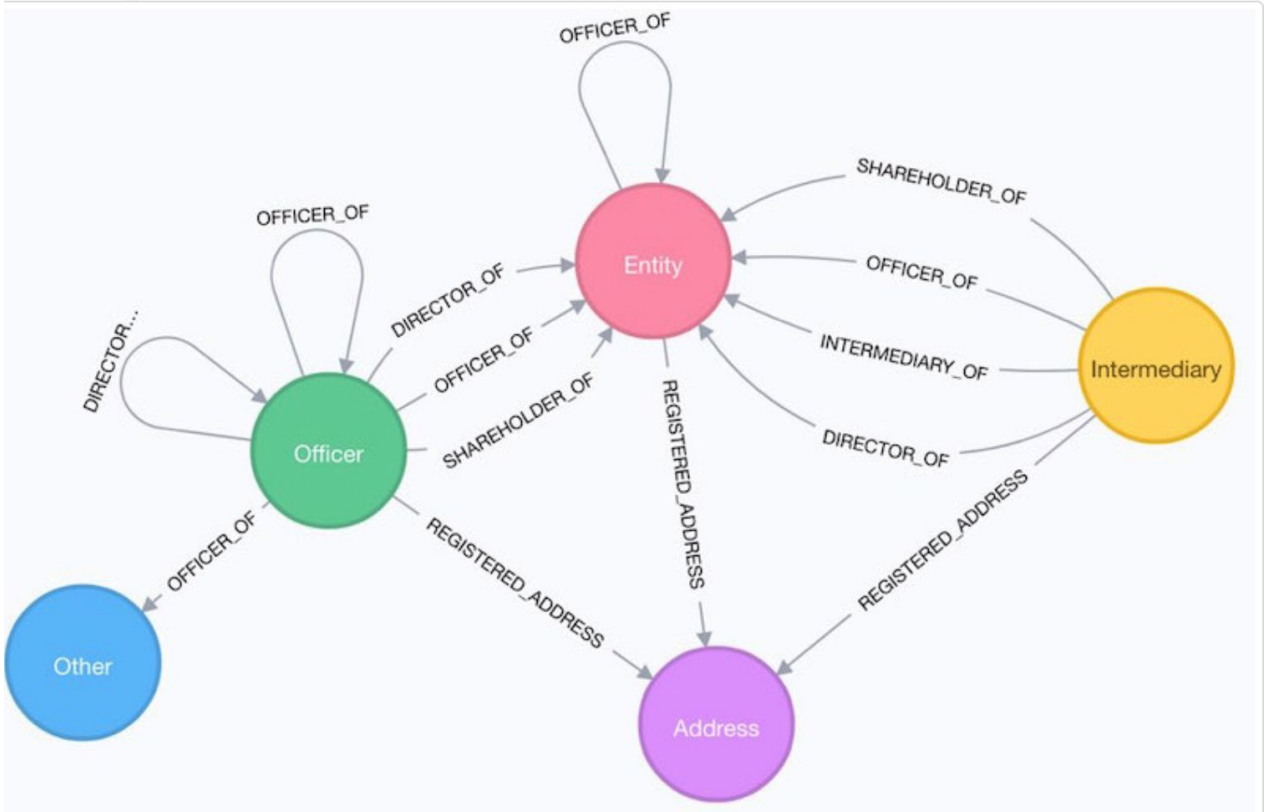
Activity feed: Russian Twitter Trolls Sandbox



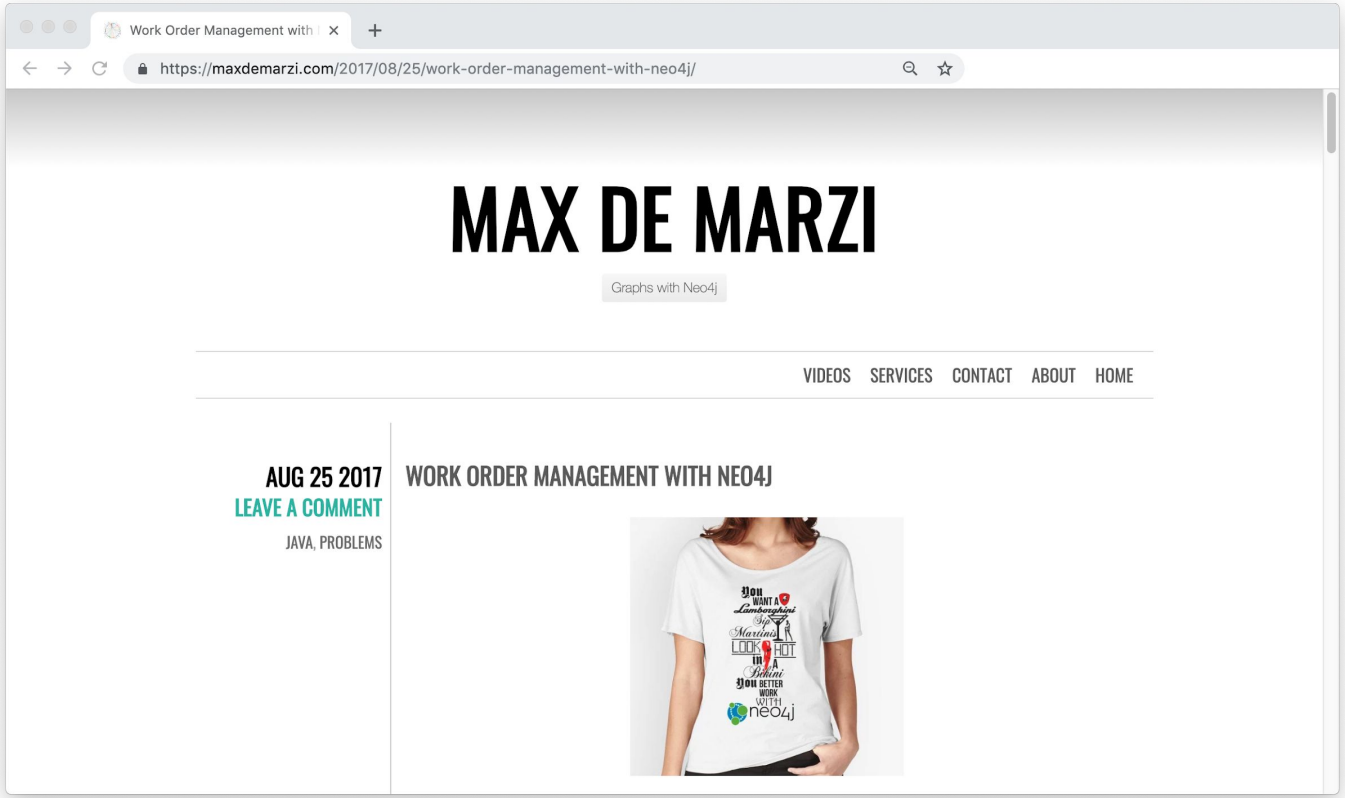
Fraud detection: Replicants



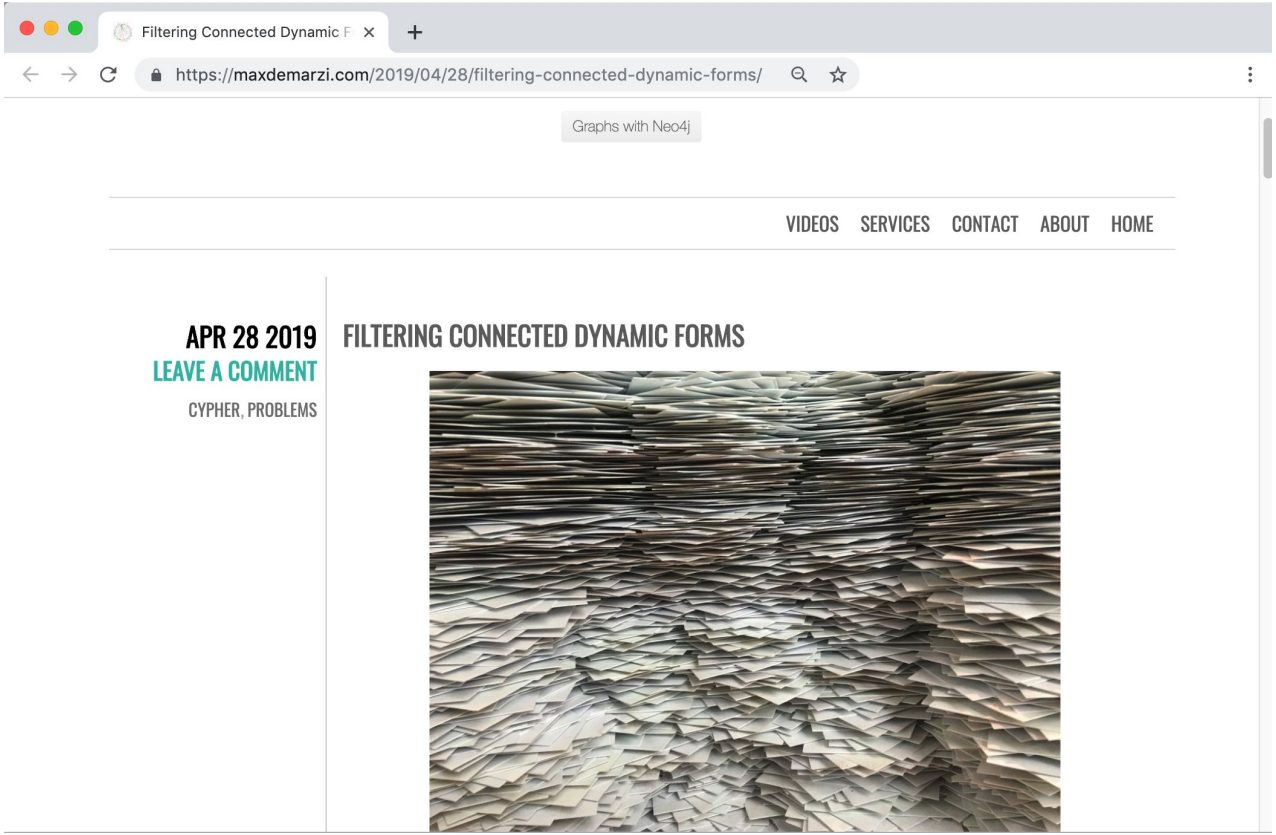
Fraud detection: Paradise Papers Sandbox



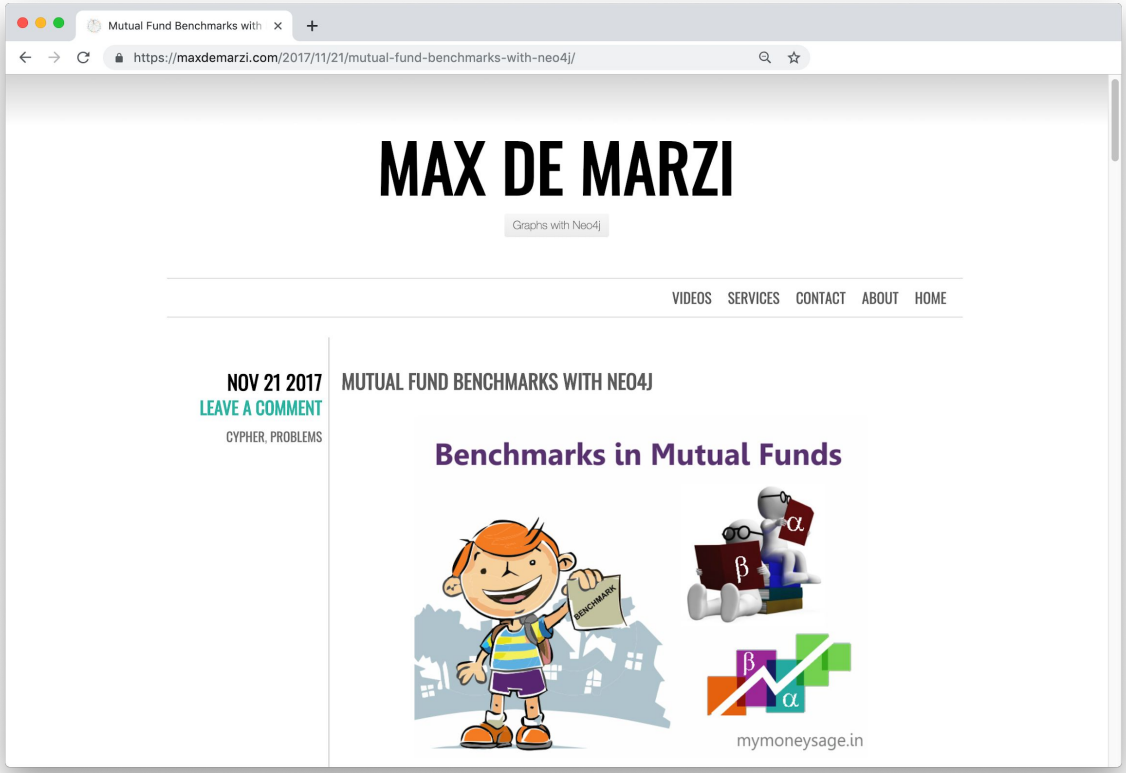
Event modeling: Work Order Management with Neo4j



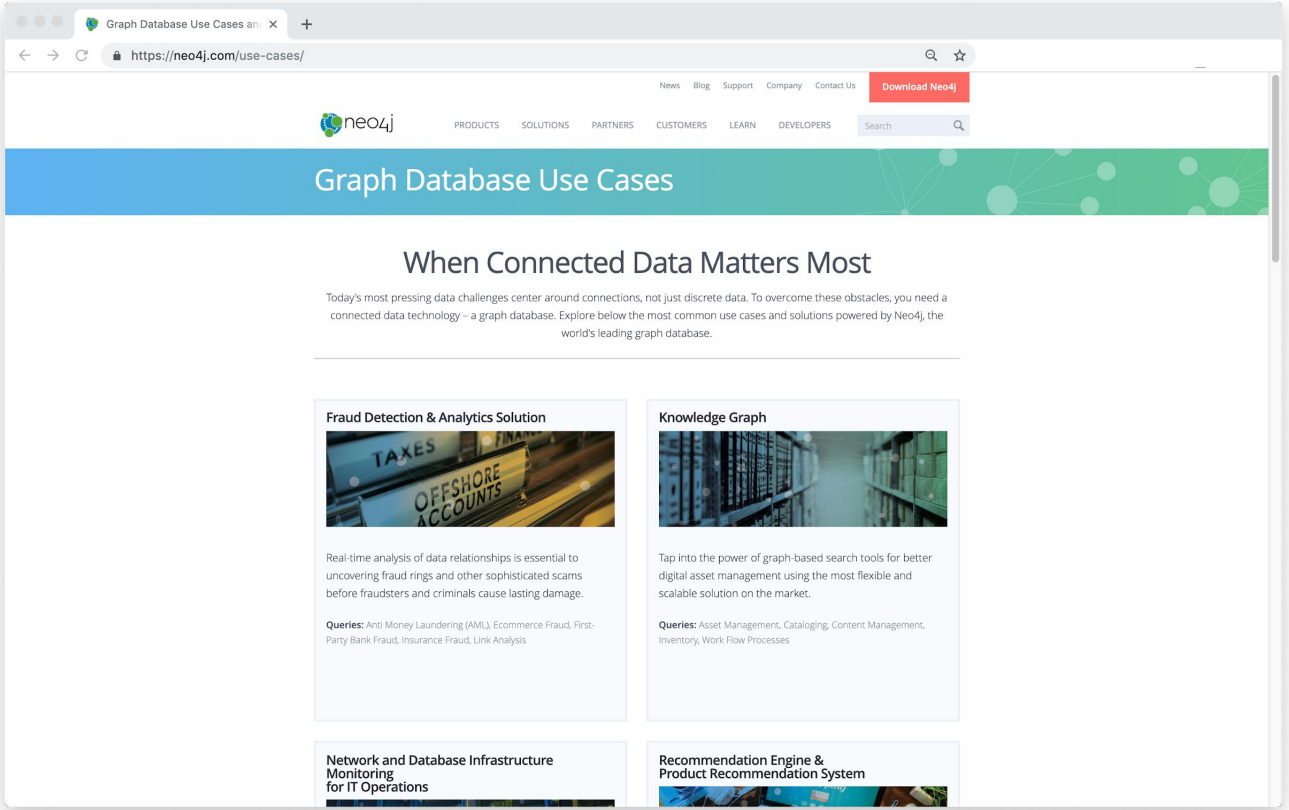
Tracking forms: Filtering connected dynamic forms



Time series: Modeling Mutual Fund Benchmarks with Neo4j



Neo4j use cases



Next steps

- Work with developers who implement/refactor (evolve) the graph data model
  - *Implementing a Graph Data Model* course.
- Neo4j Community site:
  - https://community.neo4j.com/c/neo4j-graph-platform/modeling
- Neo4j GraphGists
  - https://neo4j.com/graphgists/
- Neo4j Sales Engineering
  - Proof of Concept
- Neo4j Professional Services
  - Develop and implement model

Check your understanding

Question 1

What component of a graph data model is used to model the nouns of the questions for the domain?

Select the correct answer.

- ❑ Relationship
- ❑ Property
- ❑ Entity
- ❑ Data source

Answer 1

What component of a graph data model is used to model the nouns of the questions for the domain?

Select the correct answer.

- ❏ Relationship
- ❏ Property
- ✔ Entity
- ❏ Data source

Question 2

What is the maximum number of relationships types you can define in a graph data model?

Select the correct answer.

- ❑ 64K
- ❑ 16M
- ❑ 64M
- ❑ There is no maximum

Answer 2

What is the maximum number of relationships types you can define in a graph data model?

Select the correct answer.

- ❏ 64K
- ❏ 16M
- ❏ 64M
- ✔ There is no maximum

Question 3

Suppose you have a graph data model with a node named *Person* that has a property, *PrimaryLanguage*. In the graph, there will be millions of *Person* nodes. How can you define the graph data model so that the value for *PrimaryLanguage* will not be part of each *Person* node (Goal: Eliminate duplication of data.)?

Select the correct answers.

❏ Create a *Language* node and have the *Person* node connect to the *Language* node using the PRIMARY_LANGUAGE relationship.

❏ Create a *Language* node and have the *Person* node connect to the *Language* node using the PRIMARY_LANGUAGE_<value> relationship.

❏ Add a label to the *Person* node which is the value of *PrimaryLanguage*.

❏ Add a label to the *Person* node which is the value of *PrimaryLanguage_<value>*.

Answer 3

Suppose you have a graph data model with a node named *Person* that has a property, *PrimaryLanguage*. In the graph, there will be millions of *Person* nodes. How can you define the graph data model so that the value for *PrimaryLanguage* will not be part of each *Person* node (Goal: Eliminate duplication of data.)?

Select the correct answers.

✔️ Create a *Language* node and have the *Person* node connect to the *Language* node using the PRIMARY_LANGUAGE relationship.

✔️ Create a *Language* node and have the *Person* node connect to the *Language* node using the PRIMARY_LANGUAGE_<value> relationship.

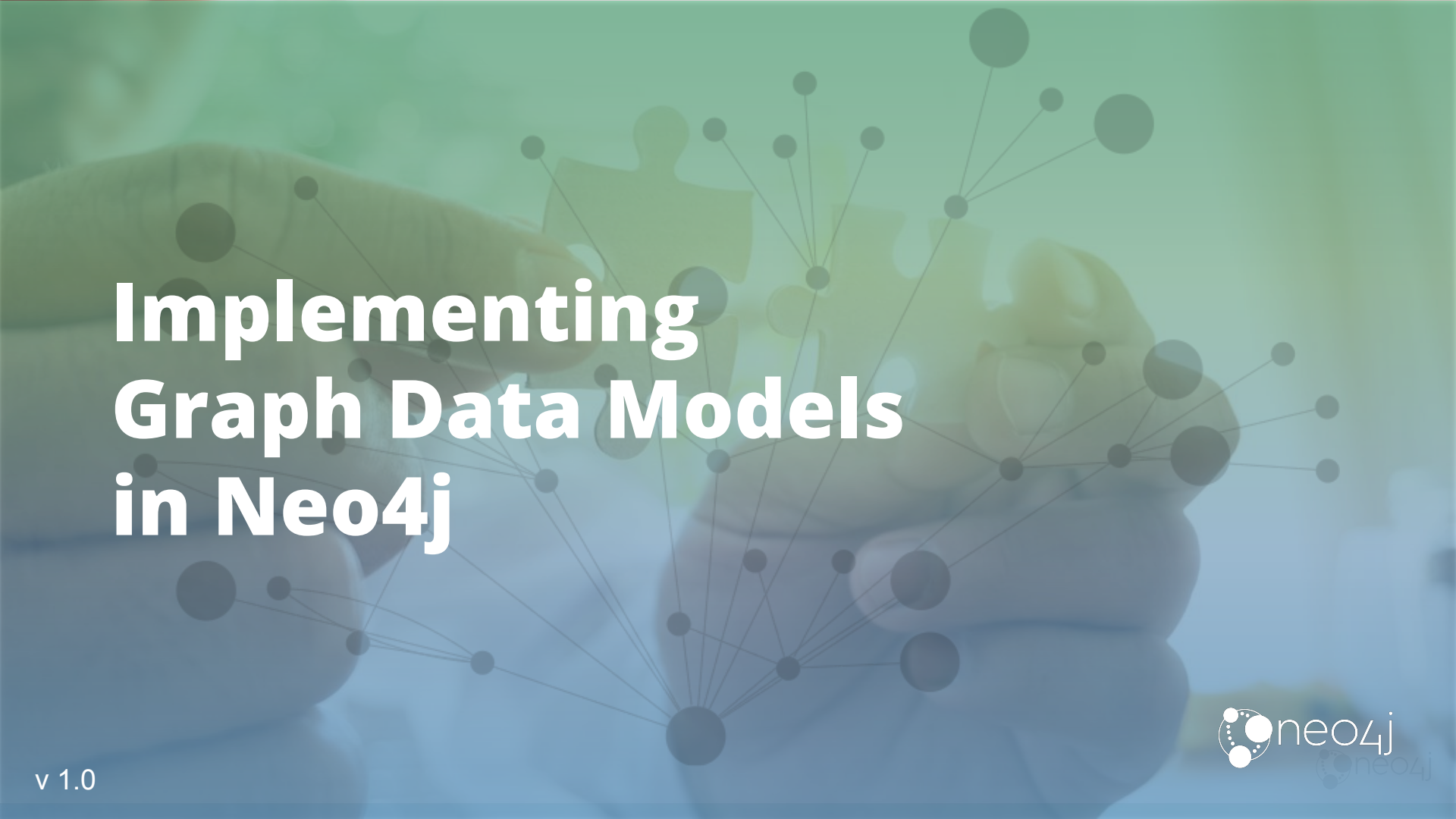
✔️ Add a label to the *Person* node which is the value of *PrimaryLanguage*.

✔️ Add a label to the *Person* node which is the value of *PrimaryLanguage_<value>*.

Summary

You should now be able to:

- Describe how Neo4j supports a graph data model.
- Use Arrows to define a graph data model.
- Describe the workflow for creating a graph data model.
- Define use cases and questions for an application domain.
- Define entities for the application domain.
- Define connections between the entities for the application domain.
- Test a graph data model.
- Evolve a graph data model.



Implementing Graph Data Models in Neo4j

v 1.0

neo4j

Overview

At the end of this module, you should be able to write Cypher code to:

- Create a graph using sample data for a simple graph data model.
- Load data for a graph from CSV files.
- Profile queries against the graph.
- Refactor the graph to add intermediate nodes.
- Refactor the graph to create relationships from nodes.
- Refactor the graph to specialize relationships.
- Refactor a large graph using batching.
- Maintain a graph.

Implementing our first model

Knowledge management

- People, companies, skills
- Cross organizational

Find my professional social network

- Exchange knowledge
- Interest groups
- Help
- Staff projects

Knowledge management domain question

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

Identify entities and relationships

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

Entities:

Person
Company
Skill

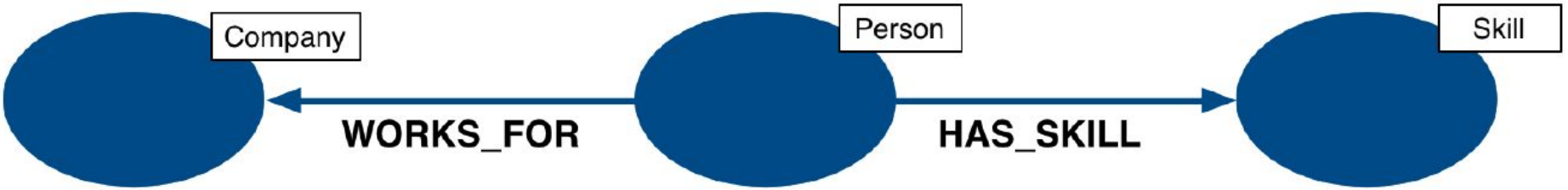
Relationships:

Person **WORKS_FOR** Company
Person **HAS_SKILL** Skill

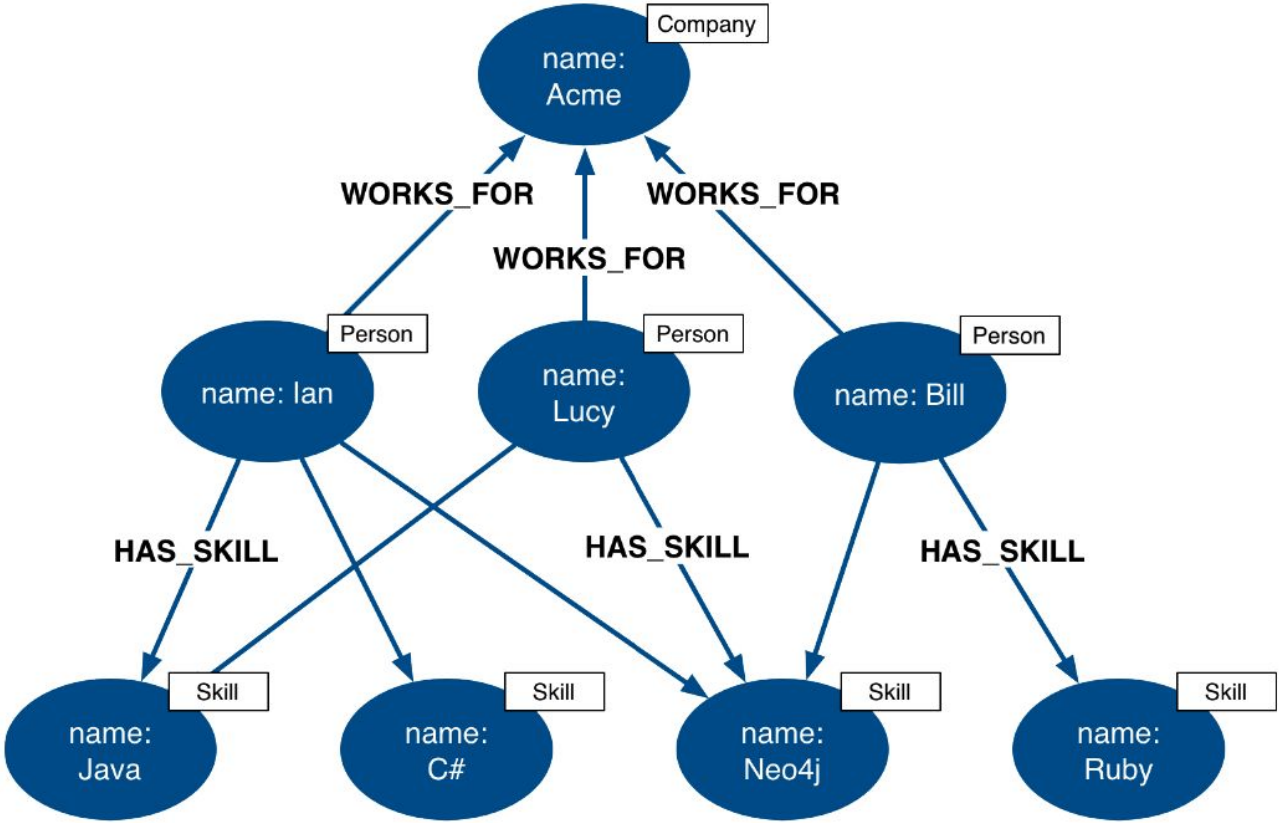
Determine paths used for model

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

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



Data model with sample data

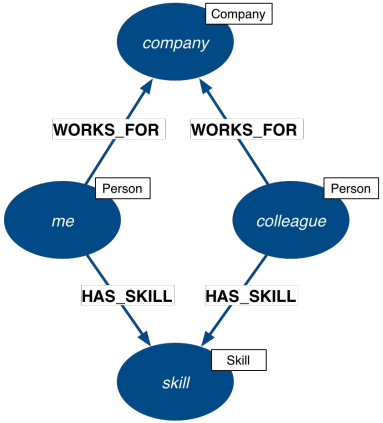


Cypher query for the question

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

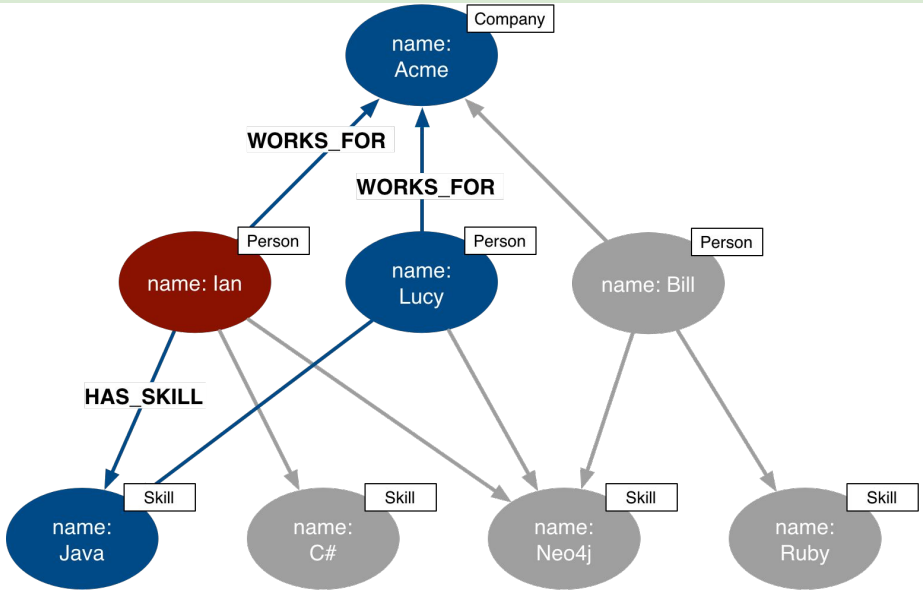
Anchor used to start the navigation

```
MATCH (company)<-[:WORKS_FOR]-(:Person{name:'Ian'})
                -[:HAS_SKILL]->(skill),
(company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)

RETURN colleague.name AS name,
count(skill) AS score,
collect(skill.name) AS skills
ORDER BY score DESC
```



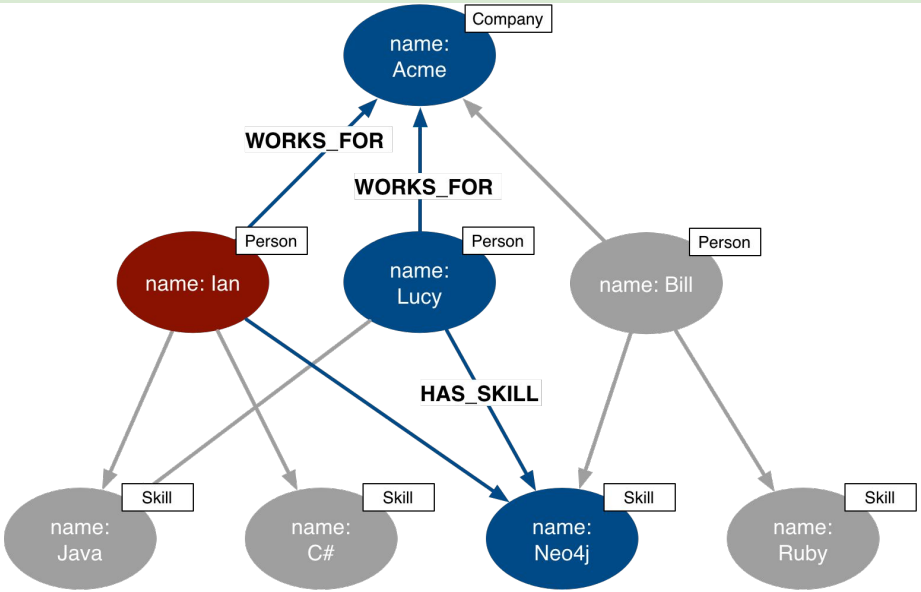
First match

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



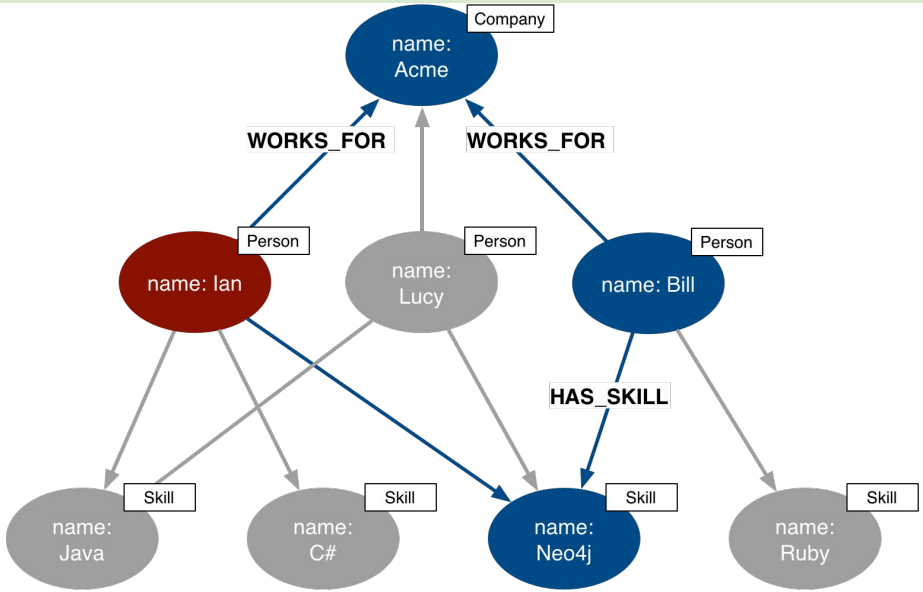
Second match

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



Third match

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



Result

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

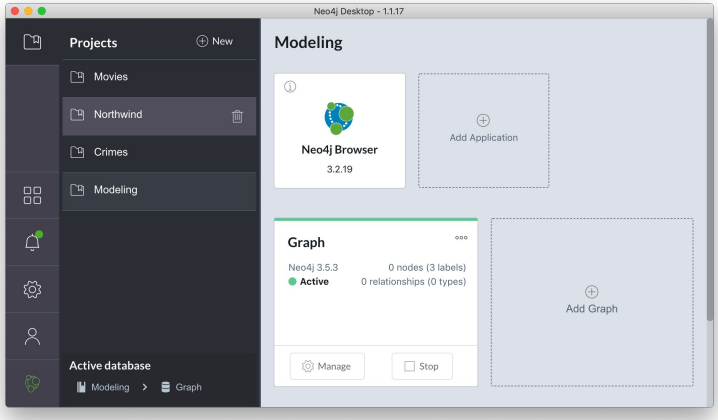
```
+-----------------------------------------------+
| name       | score | skills                   |
+-----------------------------------------------+
| "Lucy"     | 2     | ["Java","Neo4j"]         |
| "Bill"     | 1     | ["Neo4j", "Ruby"]        |
+-----------------------------------------------+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

MERGE is your friend

Specifying a property value when a node is merged prevents duplicate nodes.

```
MERGE (p:Person{name:'Ian'})
MERGE (s1:Skill{name:'Java'})
MERGE (s2:Skill{name:'Python'})
MERGE (s3:Skill{name:'Neo4j'})
MERGE (c)<-[:WORKS_FOR]-(p)
MERGE (p)-[r1:HAS_SKILL]->(s1)
MERGE (p)-[r2:HAS_SKILL]->(s2)
MERGE (p)-[r3:HAS_SKILL]->(s3)
SET r1.level = 2
SET r2.level = 2
SET r3.level = 3
RETURN   p, s1, s2, s3
```

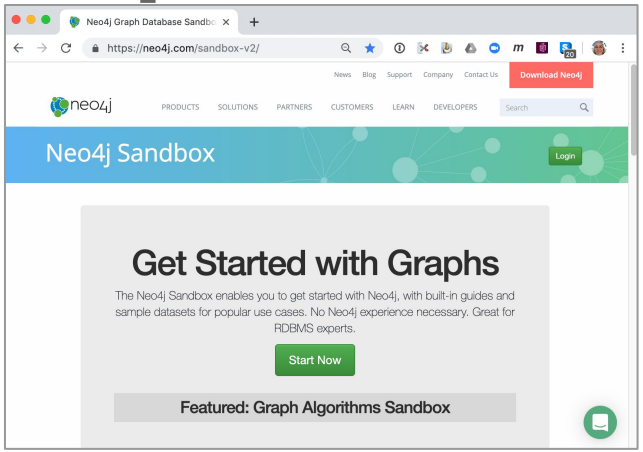
Development environment setup: Neo4j Desktop



Note: Neo4j Desktop should already be installed.

1. Start Neo4j Desktop.
2. Create a Project named **Modeling**.
3. Create a Graph in the Modeling project.
4. Add the **APOC** plugin to your project.
5. Set this property at the bottom of the **neo4j.conf** file (Manage > Settings):
   apoc.export.file.enabled=true
6. Start the database.
7. Open a Neo4j Browser window for the database.

Development environment setup: Neo4j Sandbox



1. Log in to the Neo4j Sandbox site: https://neo4j.com/sandbox-v2

2. Create an account (requires an email confirmation).

3. Create the **Blank** Sandbox.
   Note: Sandbox expires in 3 days unless you extend it to 10 days.

4. Open Neo4j Browser for the sandbox instance.

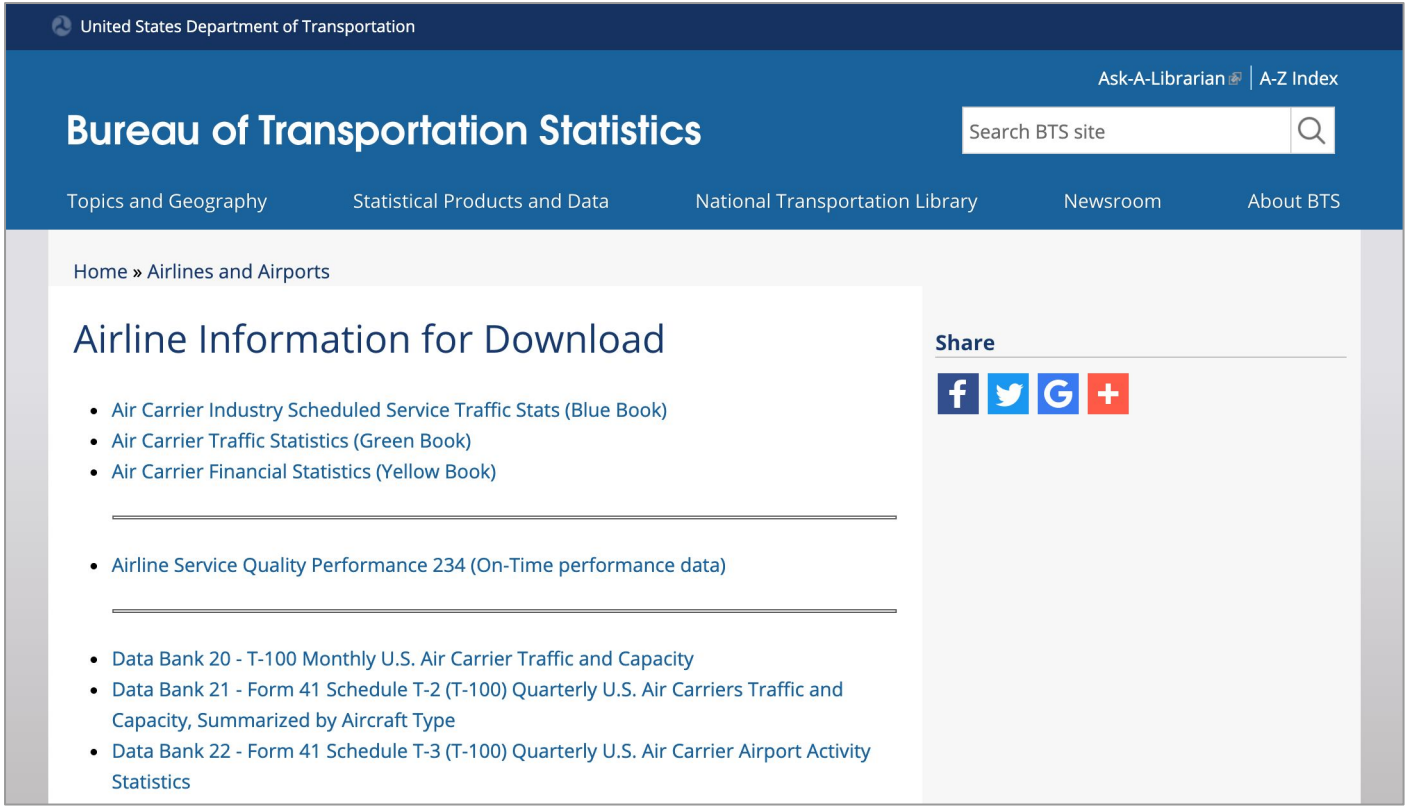
Exercise 1: Implementing your first model

In Neo4j Browser:

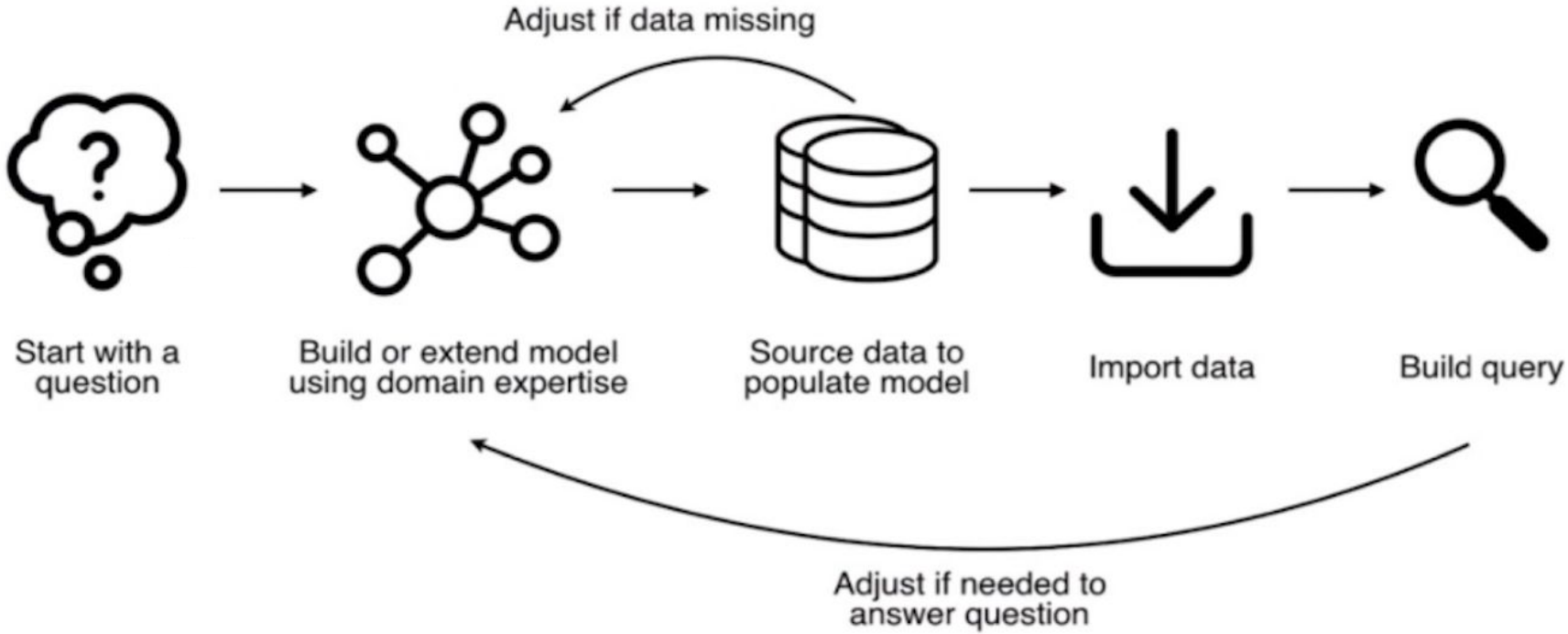
:play neo4j-modeling-exercises

Then follow instructions for Exercise 1.

Application domain for this training



The graph data modeling and implementation process



Data we will use

Origin		
OriginAirportID	Origin Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused.	Analysis
OriginAirportSeqID	Origin Airport, Airport Sequence ID. An identification number assigned by US DOT to identify a unique airport at a given point of time. Airport attributes, such as airport name or coordinates, may change over time.	
OriginCityMarketID	Origin Airport, City Market ID. City Market ID is an identification number assigned by US DOT to identify a city market. Use this field to consolidate airports serving the same city market.	Analysis
Origin	Origin Airport	Analysis
OriginCityName	Origin Airport, City Name	
OriginState	Origin Airport, State Code	Analysis
OriginStateFips	Origin Airport, State Fips	Analysis
OriginStateName	Origin Airport, State Name	
OriginWac	Origin Airport, World Area Code	Analysis

Destination		
DestAirportID	Destination Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused.	Analysis
DestAirportSeqID	Destination Airport, Airport Sequence ID. An identification number assigned by US DOT to identify a unique airport at a given point of time. Airport attributes, such as airport name or coordinates, may change over time.	
DestCityMarketID	Destination Airport, City Market ID. City Market ID is an identification number assigned by US DOT to identify a city market. Use this field to consolidate airports serving the same city market.	Analysis
Dest	Destination Airport	Analysis
DestCityName	Destination Airport, City Name	
DestState	Destination Airport, State Code	Analysis
DestStateFips	Destination Airport, State Fips	Analysis
DestStateName	Destination Airport, State Name	
DestWac	Destination Airport, World Area Code	Analysis

Time Period		
Year	Year	
Quarter	Quarter (1-4)	Analysis
Month	Month	Analysis
DayofMonth	Day of Month	
DayOfWeek	Day of Week	Analysis
FlightDate	Flight Date (yyyymmdd)	

Domain question for our model

As an air travel enthusiast

I want to know how airports are connected

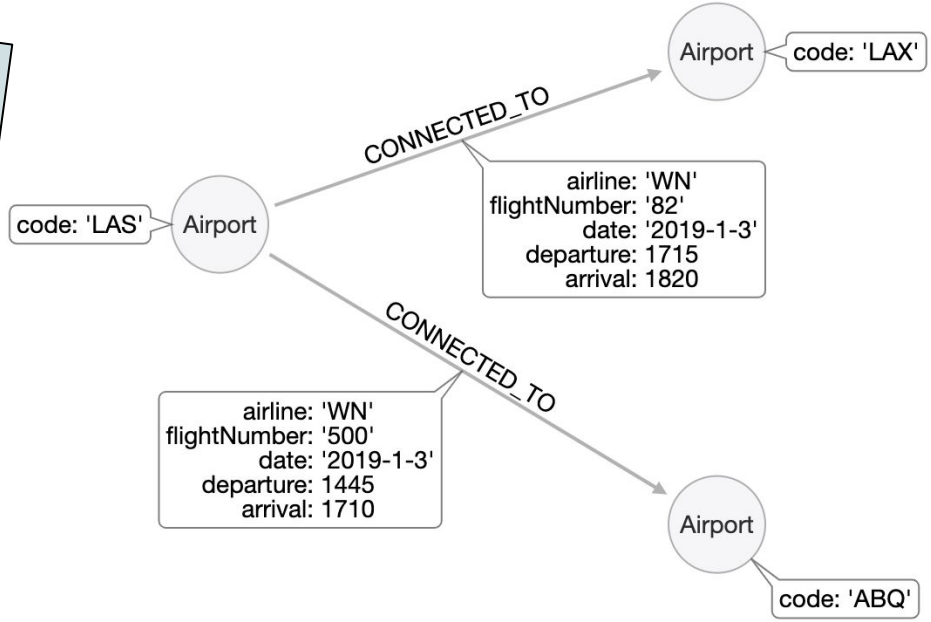
So that I can find the busiest ones

Initial data model with sample data

As an air travel enthusiast

I want to know how airports are connected

So that I can find the busiest ones



Working with the sample data

A best practice when creating a graph with Cypher is to use the MERGE statement. This ensures that you will not create duplicate nodes or relationships in the graph.

In the next exercise, Exercise 2, you will:

1. Create three *Airport* nodes and two *CONNECTED_TO* relationships per the initial data model by using the MERGE statement.

2. Query the graph to show the newly-created nodes and relationships.

Creating the nodes and relationships using the MERGE statement and hard-coding values is fine for small sample of data, but once you start working with larger amounts of data, you will want to load the data (typically from CSV files).

Exercise 2: Getting started with the airport graph data model

In Neo4j Browser:

:play neo4j-modeling-exercises

Then follow instructions for Exercise 2.

LOAD CSV (review)

Cypher statement used to load a moderate (< 10K rows) amount of data into a graph.

```
LOAD CSV      // load csv data

WITH HEADERS // optionally use first header row as keys in "row" map

FROM "url"    // file:// URL relative to $NEO4J_HOME/import or http://

AS row        // return each row of the CSV as list of strings or map

// ... rest of the Cypher statement ...
```

Note: You should use PERIODIC COMMIT or the APOC library for loading larger amounts of data which you will do later in this course.

Importing CSV data

1. Copy CSV files to the **import** folder associated with the database you are working with.

   Hint: File locations vary by platform and how you installed Desktop.

2. Execute this Cypher statement to examine the first 5 rows of the data.

```
LOAD CSV WITH HEADERS FROM
'file:///flights_2019_1k.csv' AS row
RETURN row LIMIT 5
```

Exercise 3: Loading airport data

In Neo4j Browser:

:play neo4j-modeling-exercises

Then follow instructions for Exercise 3.

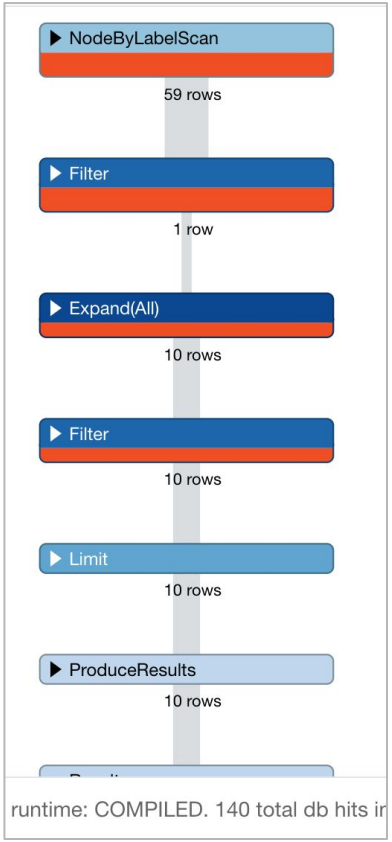
Profiling queries

Profiling is an important part of the modeling workflow:

1. Load data into the graph using the agreed-upon graph data model.
2. Create queries that answer the domain questions.
3. Execute the queries with test data (not a small sample).
4. PROFILE the query execution to understand what happens under the covers for the query.
5. Possibly modify the model and refactor (change) the graph.
6. PROFILE the same type of query against the refactored graph.
   Note: The new query may not be the same as the original due to the change in the graph data model.

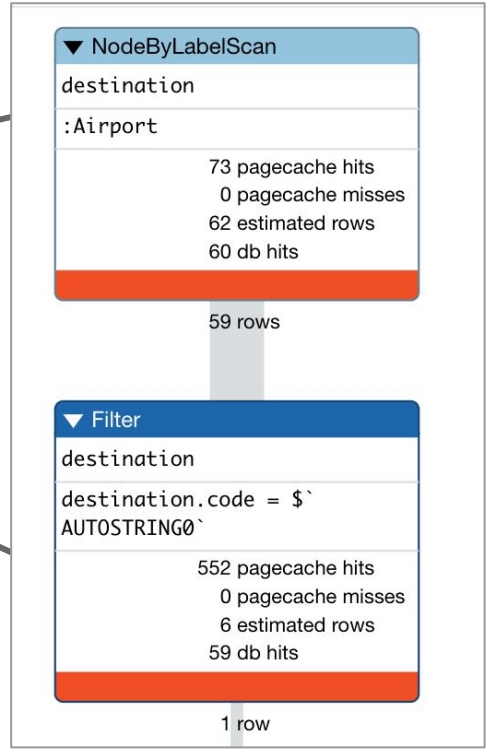
Example: Profiling a query

```
PROFILE
MATCH (origin:Airport)-
[c:CONNECTED_TO]->(destination:Airport)
WHERE destination.code = 'LAS'
RETURN origin, destination, c LIMIT 10
```



Analyzing the query profile (1)

```
PROFILE
MATCH (origin:Airport)-
[c:CONNECTED_TO]->(destination:Airport)
WHERE destination.code = 'LAS'
RETURN origin, destination, c LIMIT 10
```



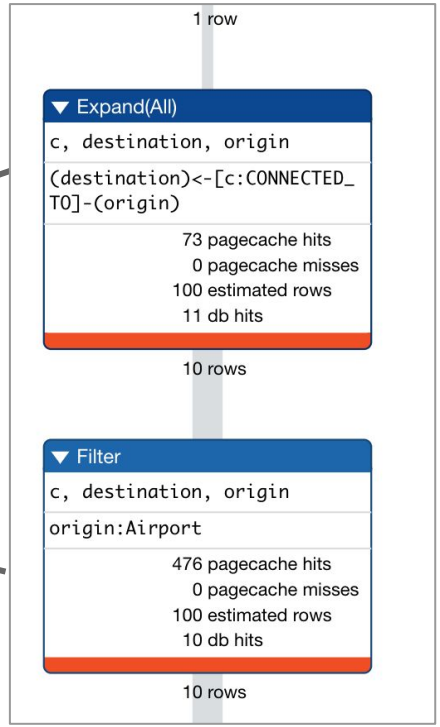
Rows come in

Do some work

Rows go out

Analyzing the query profile (2)

```
PROFILE
MATCH (origin:Airport)-
[c:CONNECTED_TO]->(destination:Airport)
WHERE destination.code = 'LAS'
RETURN origin, destination, c LIMIT 10
```



Rows come in

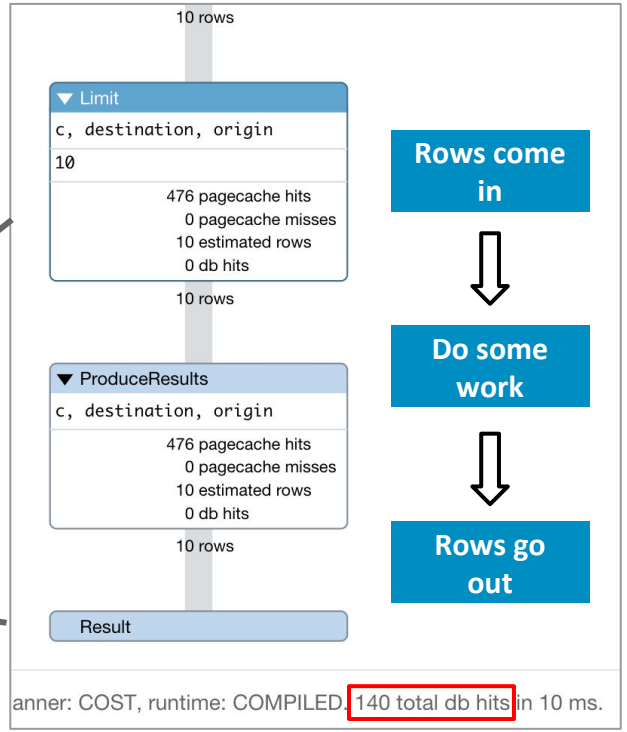
Do some work

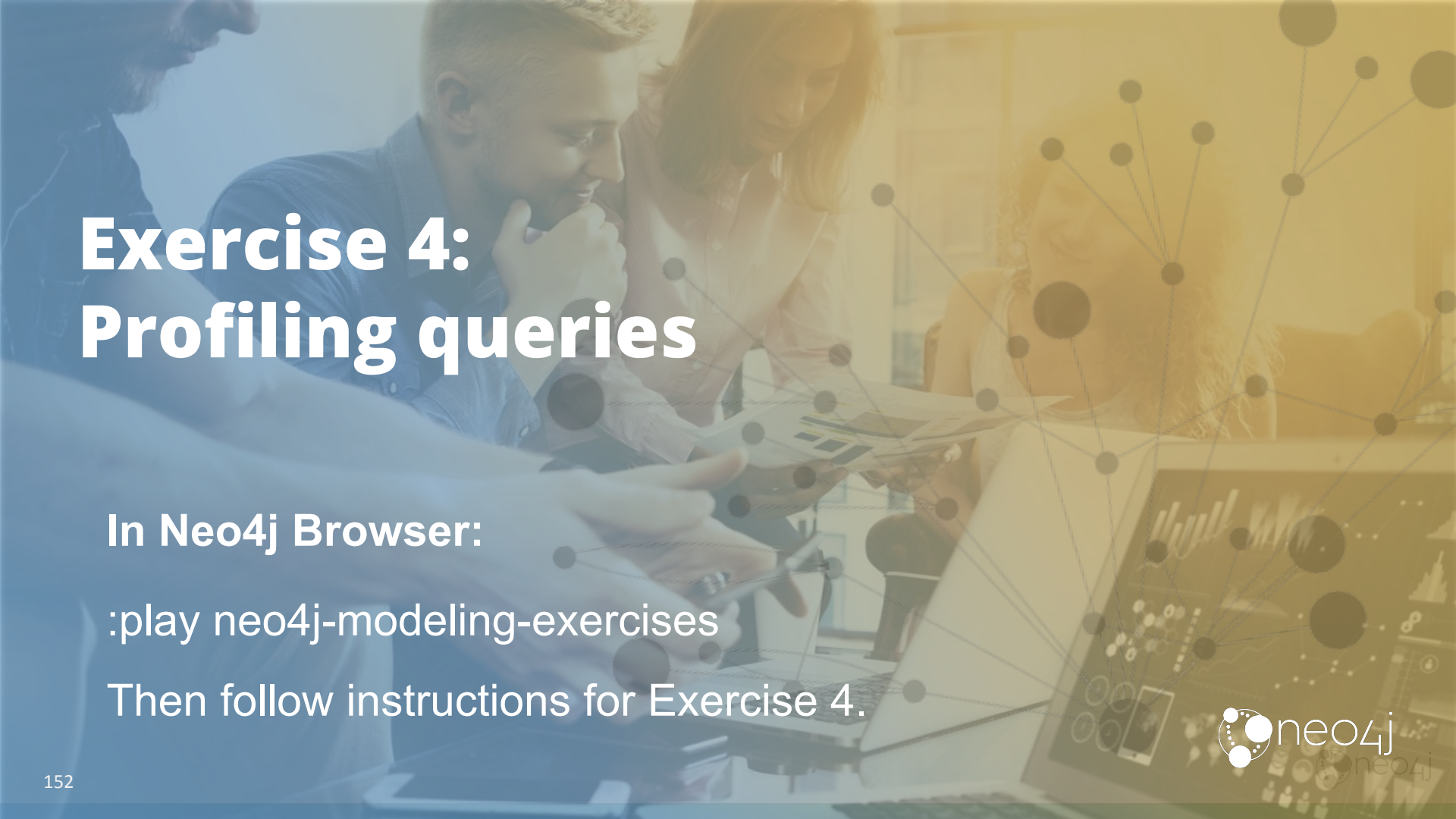
Rows go out

neo4j

Analyzing the query profile (3)

```
PROFILE
MATCH (origin:Airport)-
[c:CONNECTED_TO]->(destination:Airport)
WHERE destination.code = 'LAS'
RETURN origin, destination, c LIMIT 10
```





Exercise 4: Profiling queries

In Neo4j Browser:

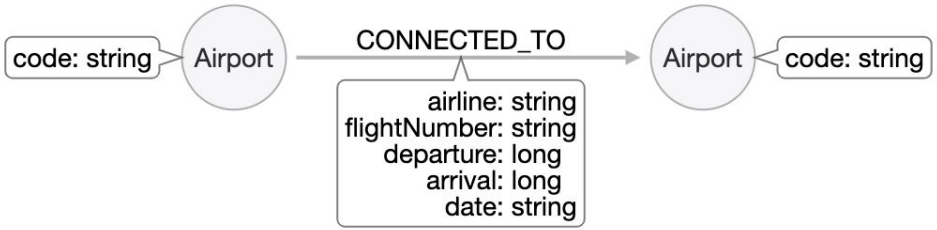
:play neo4j-modeling-exercises

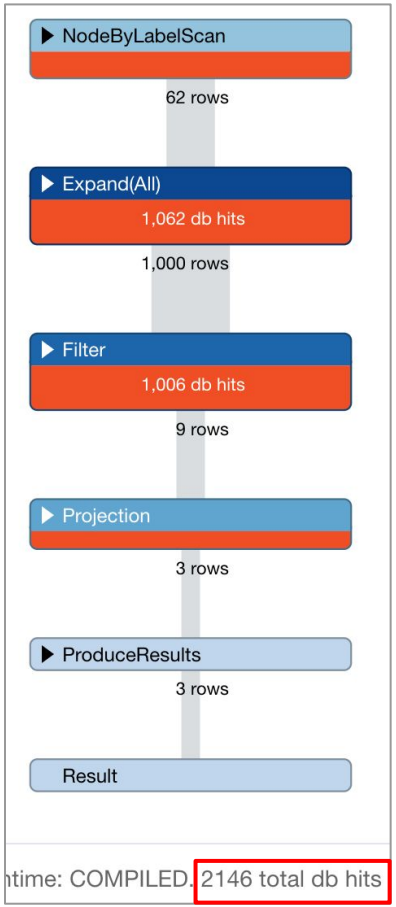
Then follow instructions for Exercise 4.

Do we need to change the model?

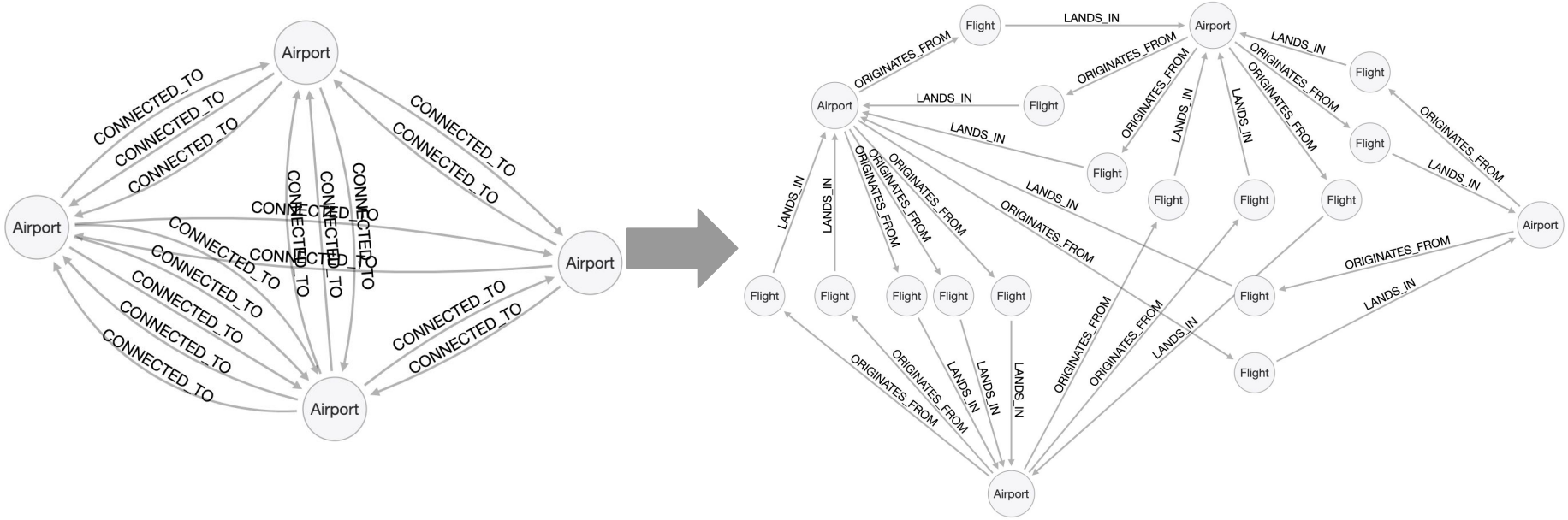
Question: What are the airports and flight information for flight number 1016 for airline WN?

```
PROFILE

MATCH
(origin:Airport)-[connection:CONNECTED_TO]-
>(destination:Airport)

WHERE connection.airline = 'WN' AND
connection.flightNumber = '1016'

RETURN origin.code, destination.code,
connection.date, connection.departure,
connection.arrival
```

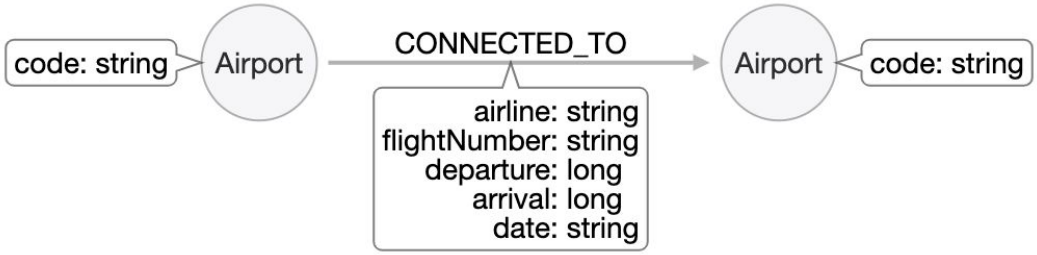




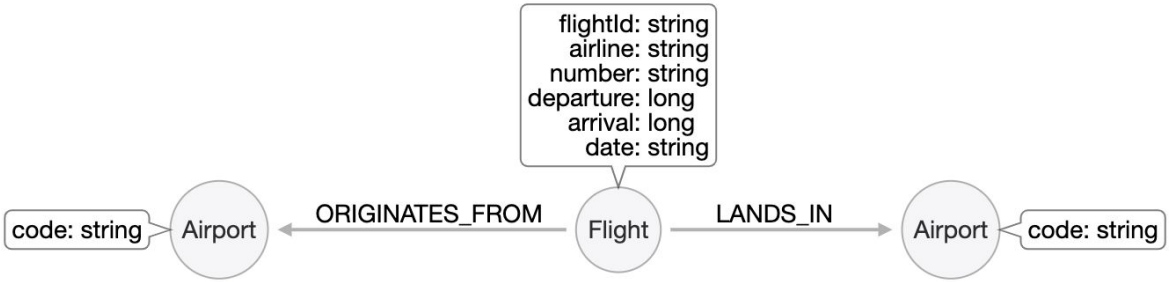
Adding a Flight node from relationship



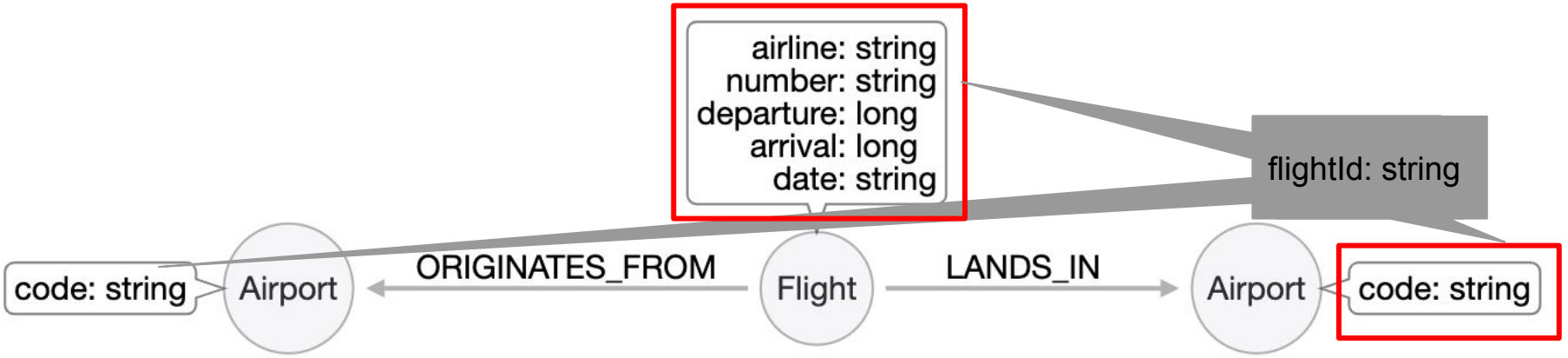
Steps: Refactoring the model



1. Create index on Airport.code
2. Create unique constraint on Flight nodes.
3. Create index on Flight.number.
4. Create Flight nodes from CONNECTED_TO relationships.
5. PROFILE query.
6. Delete CONNECTED_TO relationships.



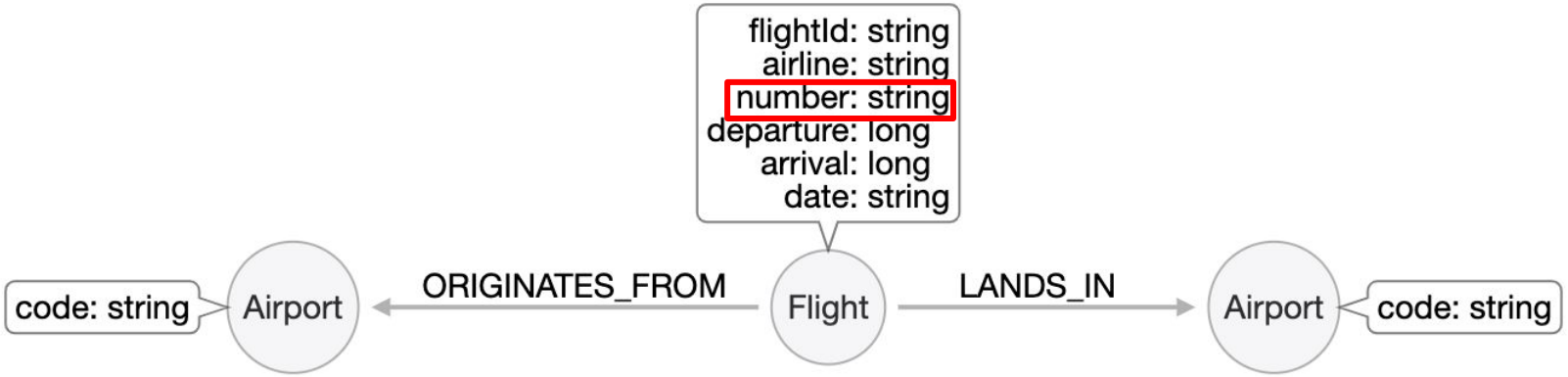
Create unique constraint on Flight nodes



The flight number for a particular airline may not be unique, especially if we take into account the origin and destination Airport.

```
CREATE CONSTRAINT ON (f:Flight)
ASSERT f.flightId IS UNIQUE
```

Create an index on Flight nodes

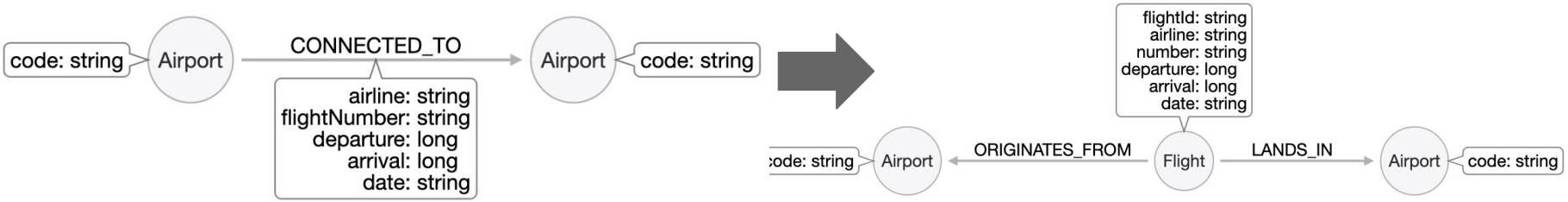


We want our queries for a flight to be fast when the flight number is specified.

```
CREATE INDEX ON :Flight(number)
```

Create Flight nodes from existing CONNECTED_TO relationships

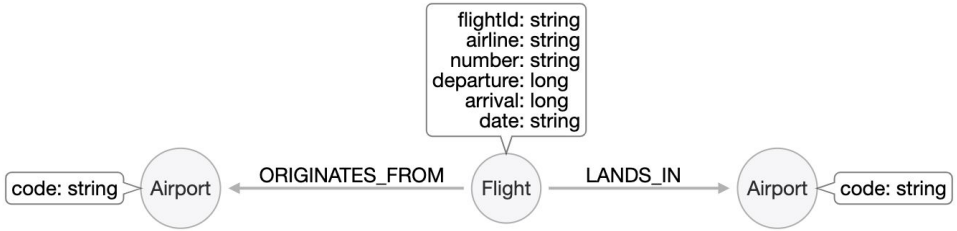
```
MATCH (origin:Airport)-[connection:CONNECTED_TO]->(destination:Airport)
MERGE (newFlight:Flight {flightId: connection.airline + connection.flightNumber +
       '_' + connection.date +  '_' + origin.code + '_' + destination.code })
ON CREATE SET newFlight.date = connection.date,
              newFlight.airline = connection.airline,
              newFlight.number = connection.flightNumber,
              newFlight.departure = connection.departure,
              newFlight.arrival = connection.arrival
MERGE (origin)<-[:ORIGINATES_FROM]-(newFlight)
MERGE (newFlight)-[:LANDS_IN]->(destination)
```

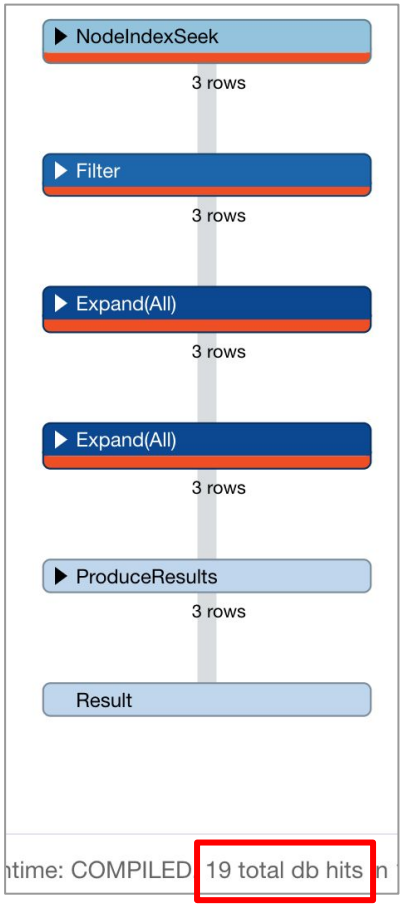


PROFILE the query with the new model implementation

Question: What are the airports and flight information for flight number 1016 for airline WN?

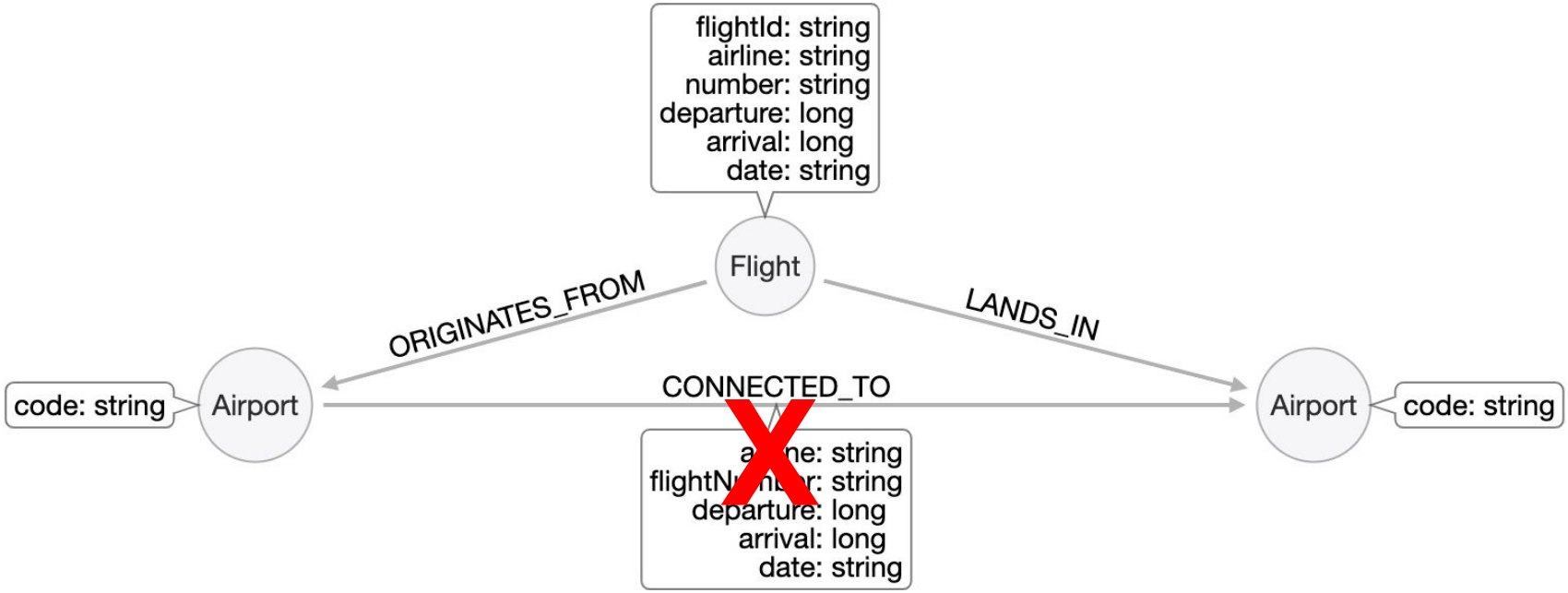
```
PROFILE
MATCH
(origin)<-[:ORIGINATES_FROM]-(flight:Flight)-
      [:LANDS_IN]->(destination)
WHERE flight.airline = 'WN' AND
      flight.number = '1016' RETURN origin,
destination, flight
```

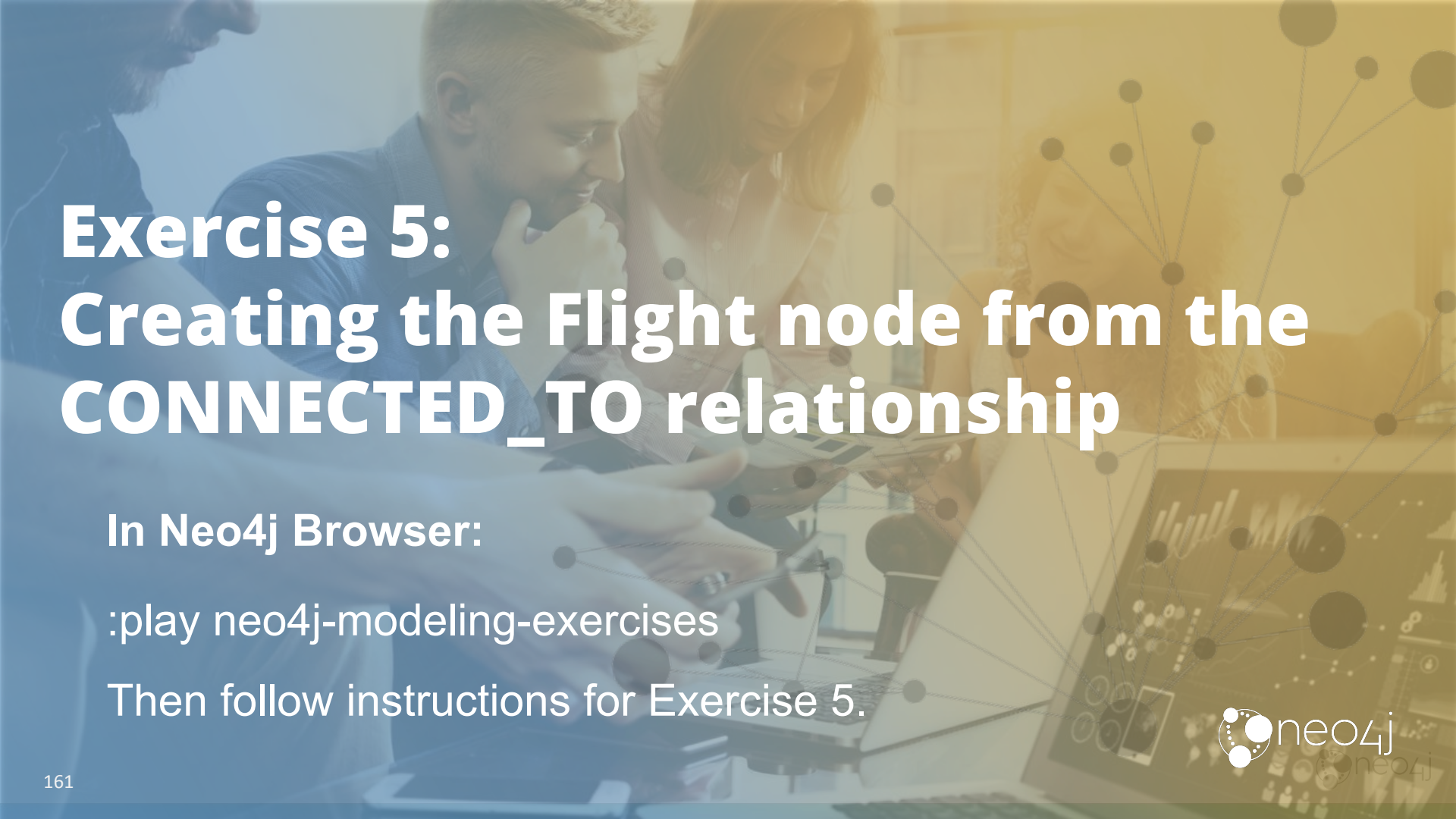




Delete the CONNECTED_TO relationships

```
MATCH ()-[connection:CONNECTED_TO]->()
DELETE connection
```





Exercise 5: Creating the Flight node from the CONNECTED_TO relationship

In Neo4j Browser:

:play neo4j-modeling-exercises

Then follow instructions for Exercise 5.

Let's implement another domain question

As a frequent traveller

I want to find flights from <origin> to <destination> on <date>

So that I can book my business flight

Implementing the model

1. Create the query that satisfies the domain question:

   Find all the flights going from Los Angeles (LAS) to Chicago Midway International (MDW) on the 3rd January, 2019.

2. PROFILE the query.
3. Refactor the model to improve the query performance.
4. PROFILE the queries thus far:
   a. Find all the flights going from Los Angeles (LAS) to Chicago Midway International (MDW) on the 3rd January.
   b. What are the airports and flight information for flight number 1016 for airline WN?
   c. What flights land in LAS?

Create the query

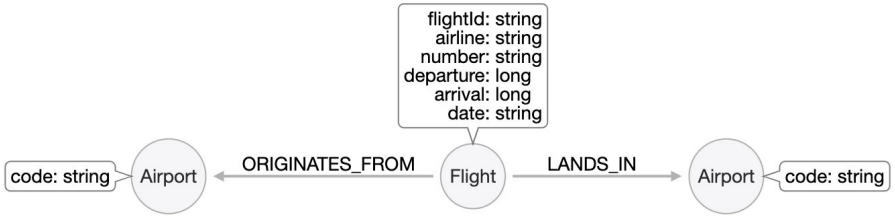
Question: What are the flights from Los Angeles (LAS) to Chicago International (MDW) on January 3, 2019?

```
MATCH (origin:Airport {code: 'LAS'})
    <-[:ORIGINATES_FROM]-(flight:Flight)-[:LANDS_IN]->
    (destination:Airport {code: 'MDW'})
WHERE flight.date = '2019-1-3'
RETURN origin, destination, flight
```

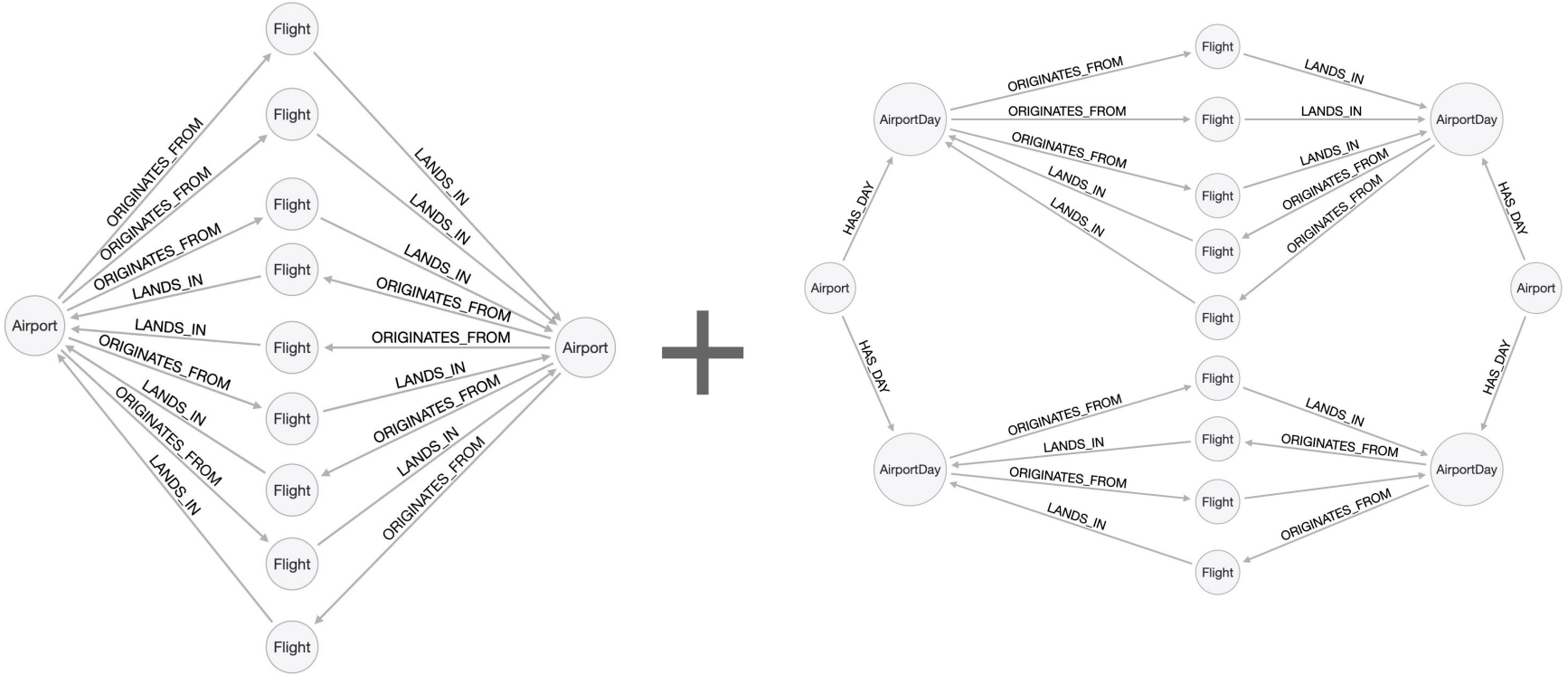
Do we need to change the model?

Question: What are the flights from Los Angeles (LAS) to Chicago International (MDW) on January 3, 2019?

```
PROFILE MATCH (origin:Airport {code:
'LAS'})
    <-[:ORIGINATES_FROM]-(flight:Flight)-
    [:LANDS_IN]->
    (destination:Airport {code: 'MDW'})
WHERE flight.date = '2019-1-3'
RETURN origin, destination, flight
```

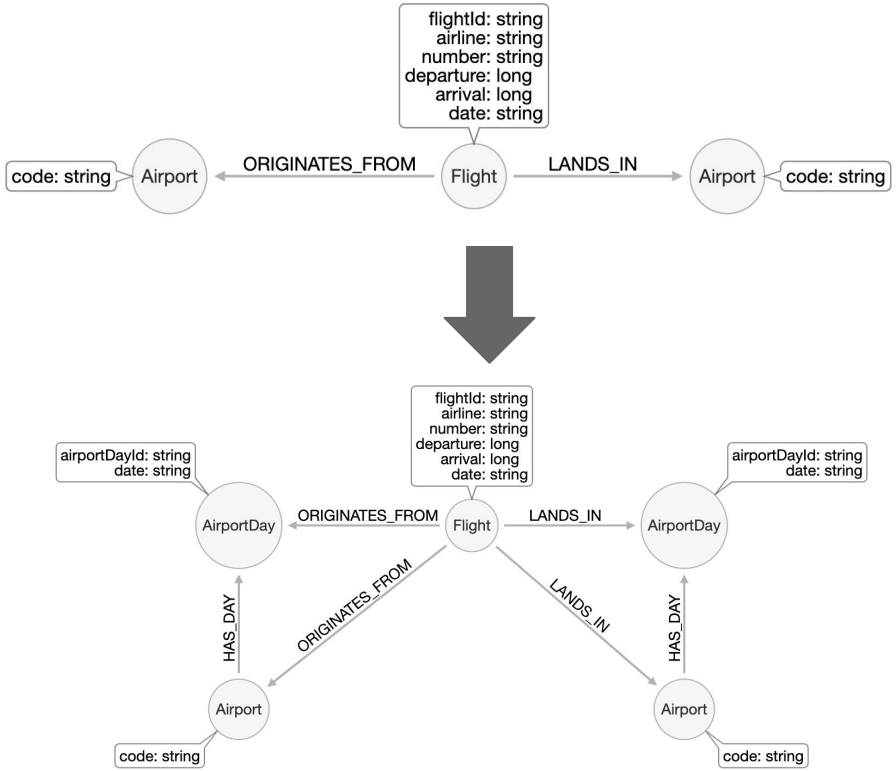


Creating AirportDay nodes



Note: You only pull out a node if you are going to query through it, otherwise a property will suffice.

Steps: Refactoring the model to create AirportDay nodes from Flight nodes



1. Create unique constraint on AirportDay nodes using date and Airport code.

```
CREATE CONSTRAINT ON
(a:AirportDay)
ASSERT a.airportDayId IS
UNIQUE
```

2. Create the AirportDay nodes and its relationships to flights as well as its relationship to an Airport.

3. PROFILE query.

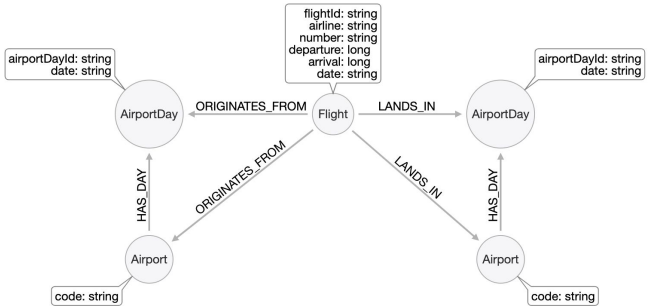
Example: Adding the AirportDay nodes

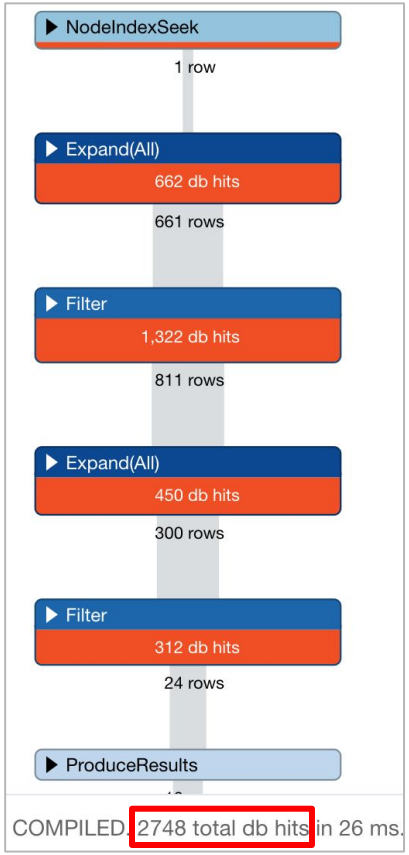
```
MATCH (origin:Airport)<-[:ORIGINATES_FROM]-(flight:Flight)-
       [:LANDS_IN]->(destination:Airport)
MERGE (originAirportDay:AirportDay
      {airportDayId: origin.code + '_' + flight.date})
SET originAirportDay.date = flight.date
MERGE (destinationAirportDay:AirportDay
       {airportDayId: destination.code + '_' + flight.date})
SET destinationAirportDay.date = flight.date
MERGE (origin)-[:HAS_DAY]->(originAirportDay)
MERGE (flight)-[:ORIGINATES_FROM]->(originAirportDay)
MERGE (flight)-[:LANDS_IN]->(destinationAirportDay)
MERGE (destination)-[:HAS_DAY]->(destinationAirportDay)
```

Profile our original query

Question: What are the flights from Los Angeles (LAS) to Chicago International (MDW) on January 3, 2019?

```
PROFILE MATCH (origin:Airport {code:
'LAS'})
    <-[:ORIGINATES_FROM]-(flight:Flight)-
    [:LANDS_IN]->
    (destination:Airport {code: 'MDW'})
WHERE flight.date = '2019-1-3'
RETURN origin, destination, flight
```

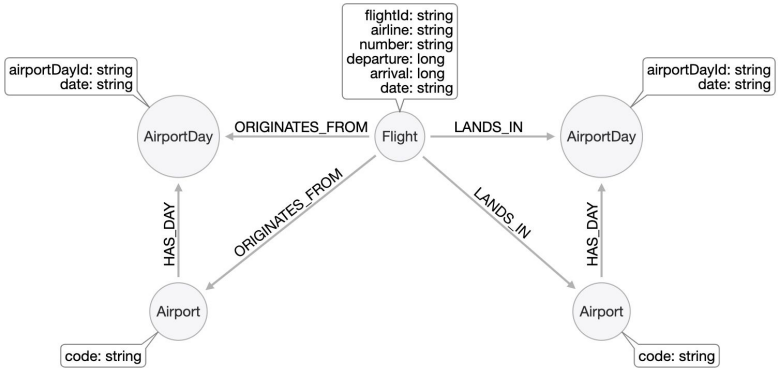


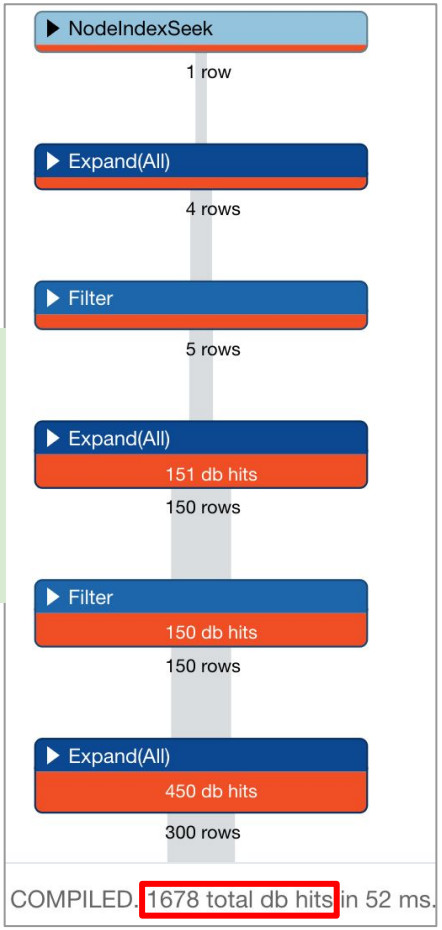


Profile our revised query

Question: What are the flights from Los Angeles (LAS) to Chicago International (MDW) on January 3, 2019?

```
PROFILE MATCH (origin:Airport {code: 'LAS'})-[:HAS_DAY]->(:AirportDay
      {date: '2019-1-3'})<-[:ORIGINATES_FROM]-(flight:Flight),

      (flight)-[:LANDS_IN]->(:AirportDay
      {date: '2019-1-3'})<-[:HAS_DAY]-(destination:Airport {code: 'MDW'})
RETURN origin, destination, flight
```

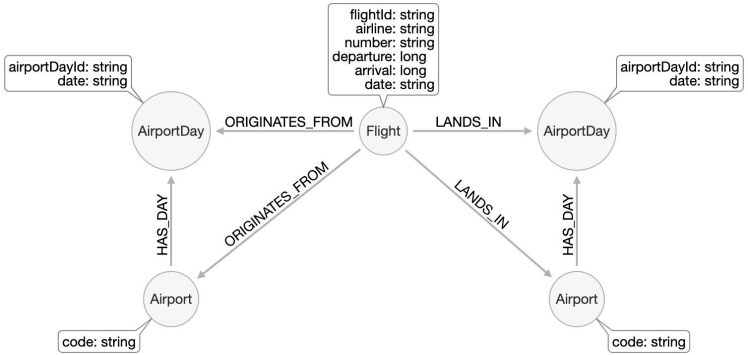


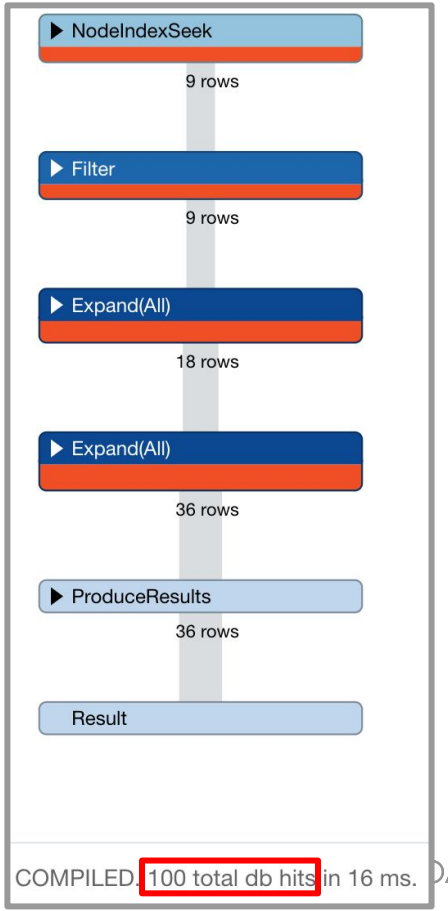


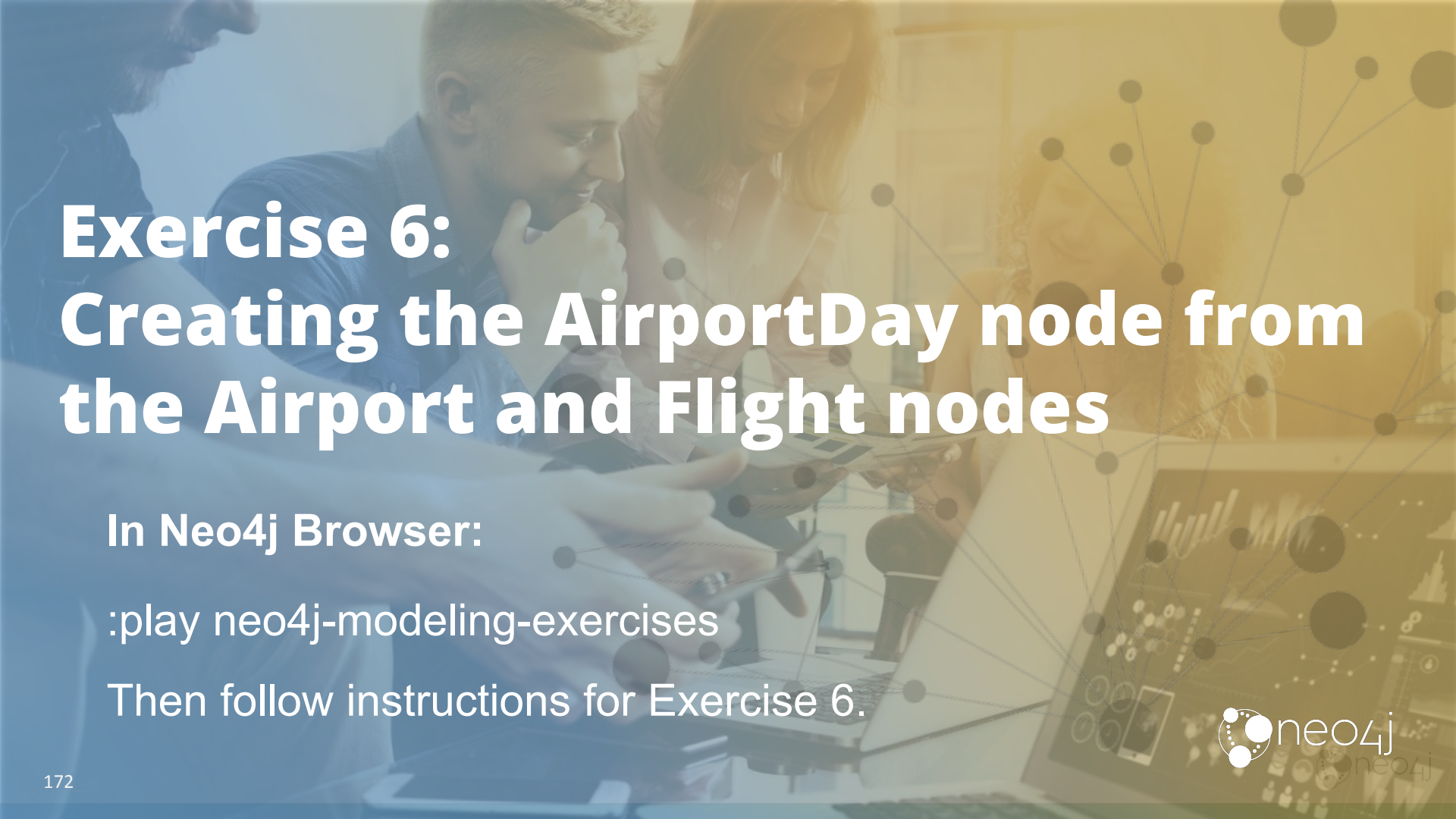
Profile the query for one of our first questions

Question: What are the airports and flight information for flight number 1016 for airline WN?

```
PROFILE MATCH (origin: Airline)<-[:ORIGINATES_FROM]-(flight:Flight)-
   [:LANDS_IN]->(destination: Airline)
WHERE flight.airline = 'WN' AND flight.number = '1016'
RETURN origin, destination, flight
```







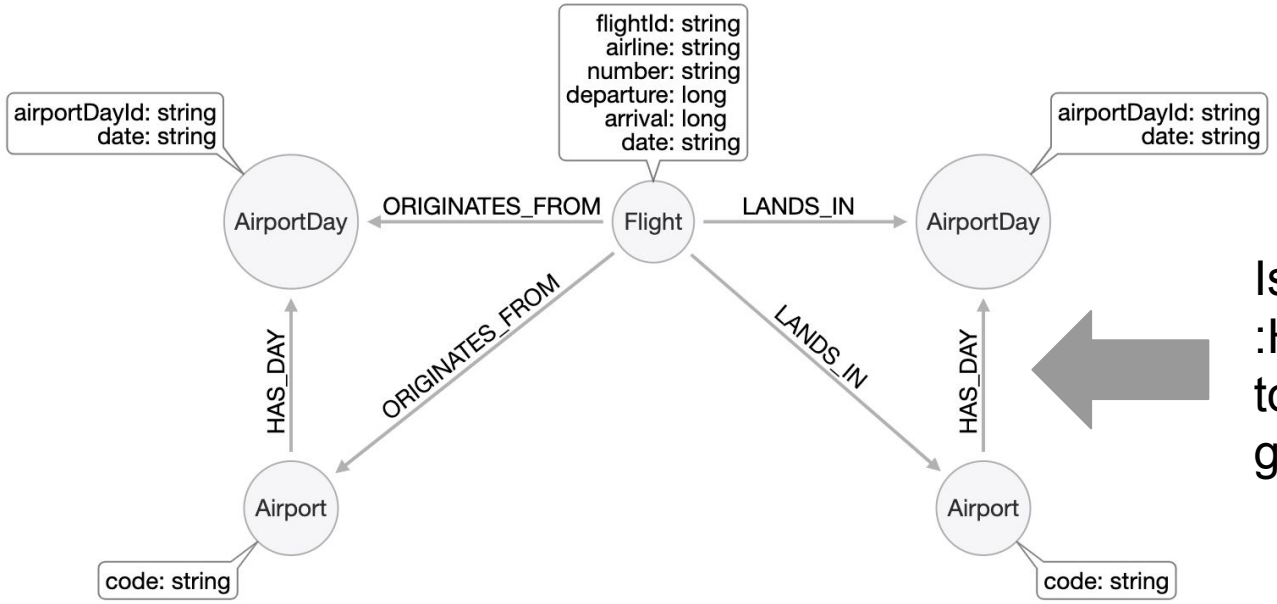
Exercise 6: Creating the AirportDay node from the Airport and Flight nodes

In Neo4j Browser:

:play neo4j-modeling-exercises

Then follow instructions for Exercise 6.

General or specific relationships?

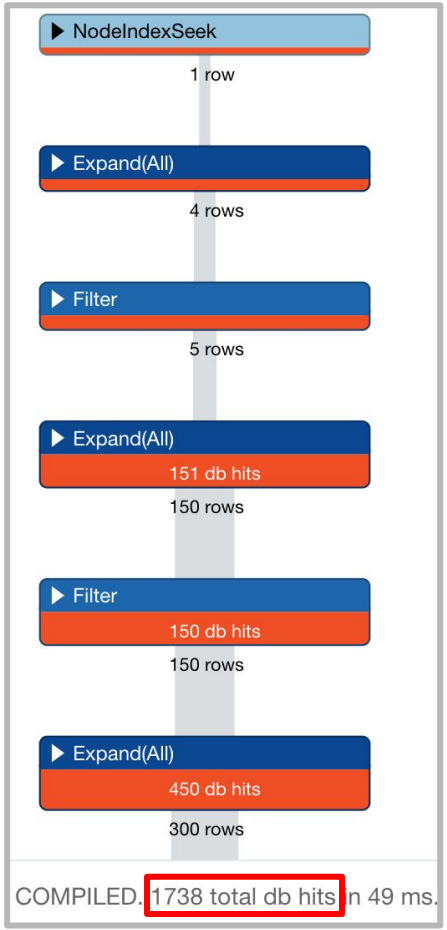


Is :HAS_DAY too general?

Query using general relationship

Question: What are the flights that arrive at MDW on 2019-1-3?

```
PROFILE
MATCH (origin:Airport {code: 'LAS'})-
[:HAS_DAY]->(originDay:AirportDay),
(originDay)<-[:ORIGINATES_FROM]-(flight:Flight),
(flight)-[:LANDS_IN]->(destinationDay),
(destinationDay:AirportDay)<-[:HAS_DAY]-
(destination:Airport {code: 'MDW'})
WHERE originDay.date = '2019-1-3' AND
destinationDay.date = '2019-1-3'
RETURN flight.date, flight.number, flight.airline,
flight.departure, flight.arrival
ORDER BY flight.date, flight.departure
```



Specific relationships

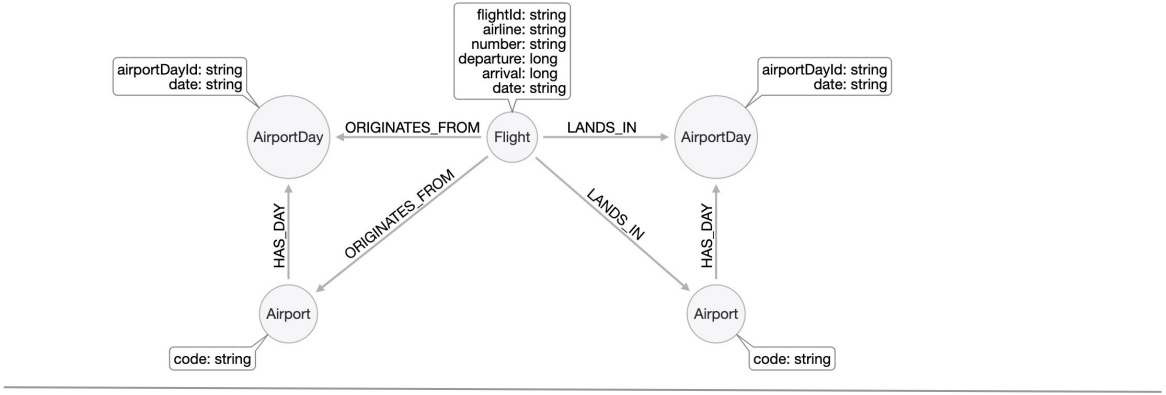
Neo4j is optimized for searching by relationship types. As we add more data, the number of HAS_DAY relationships that we have to traverse increases.

If we have 10 years worth of data we have to traverse 3,650 relationships from the Airport to find the AirportDay that we're interested in.

Refactor: Create specific relationships from properties and a general relationship

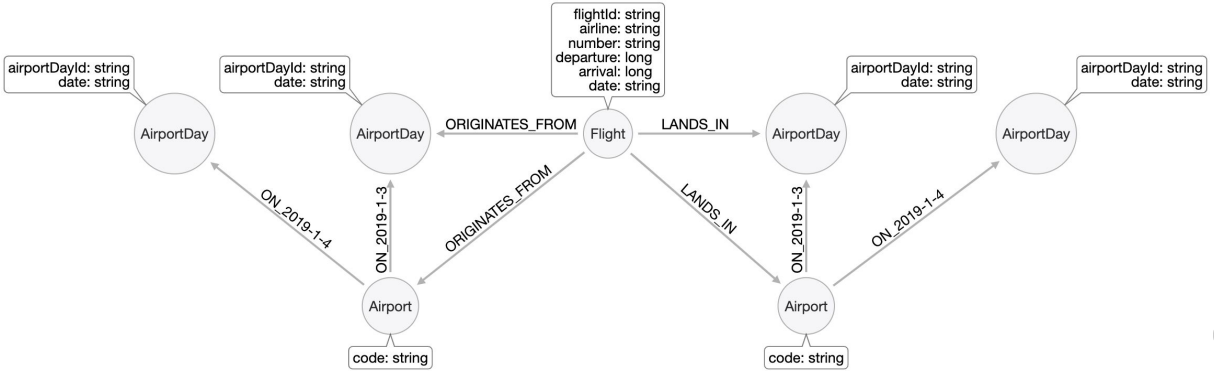
General relationship:

:HAS_DAY



Specific relationships:

:ON_2019-1-3
:ON_2019-1-4

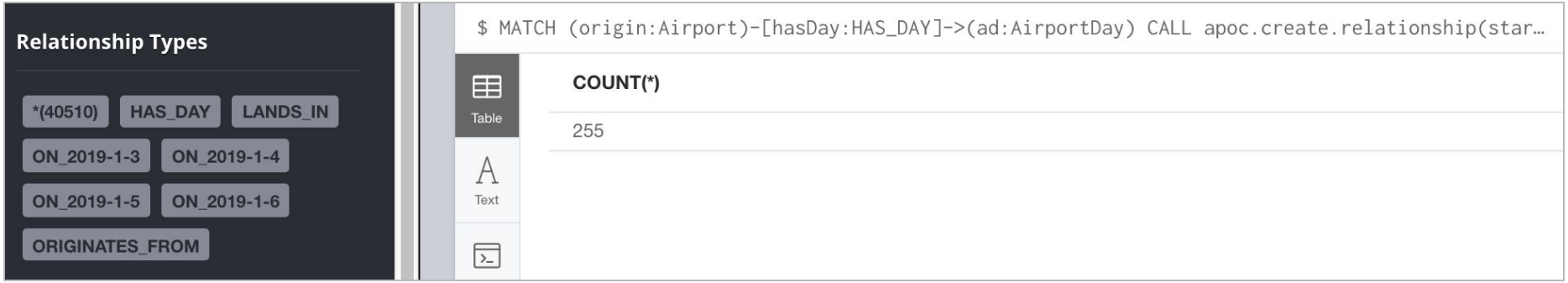


Using APOC for creating relationships

```
apoc.create.relationship(startNode(<relationship-variable>),
                         '<new-relationship-value>',
                         {<relationship-property list},
                         endNode(<relationship-variable>)
                         )
                         YIELD rel
```

Refactoring relationships

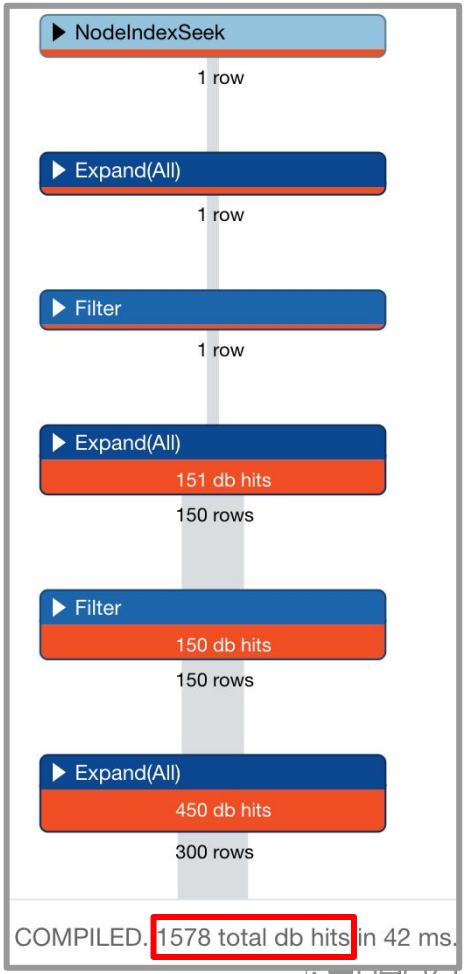
```
MATCH (origin:Airport)-[hasDay:HAS_DAY]->(ad:AirportDay)
CALL apoc.create.relationship(startNode(hasDay),
                              'ON_' + ad.date,
                              {},
                              endNode(hasDay) ) YIELD rel
RETURN COUNT(*)
```

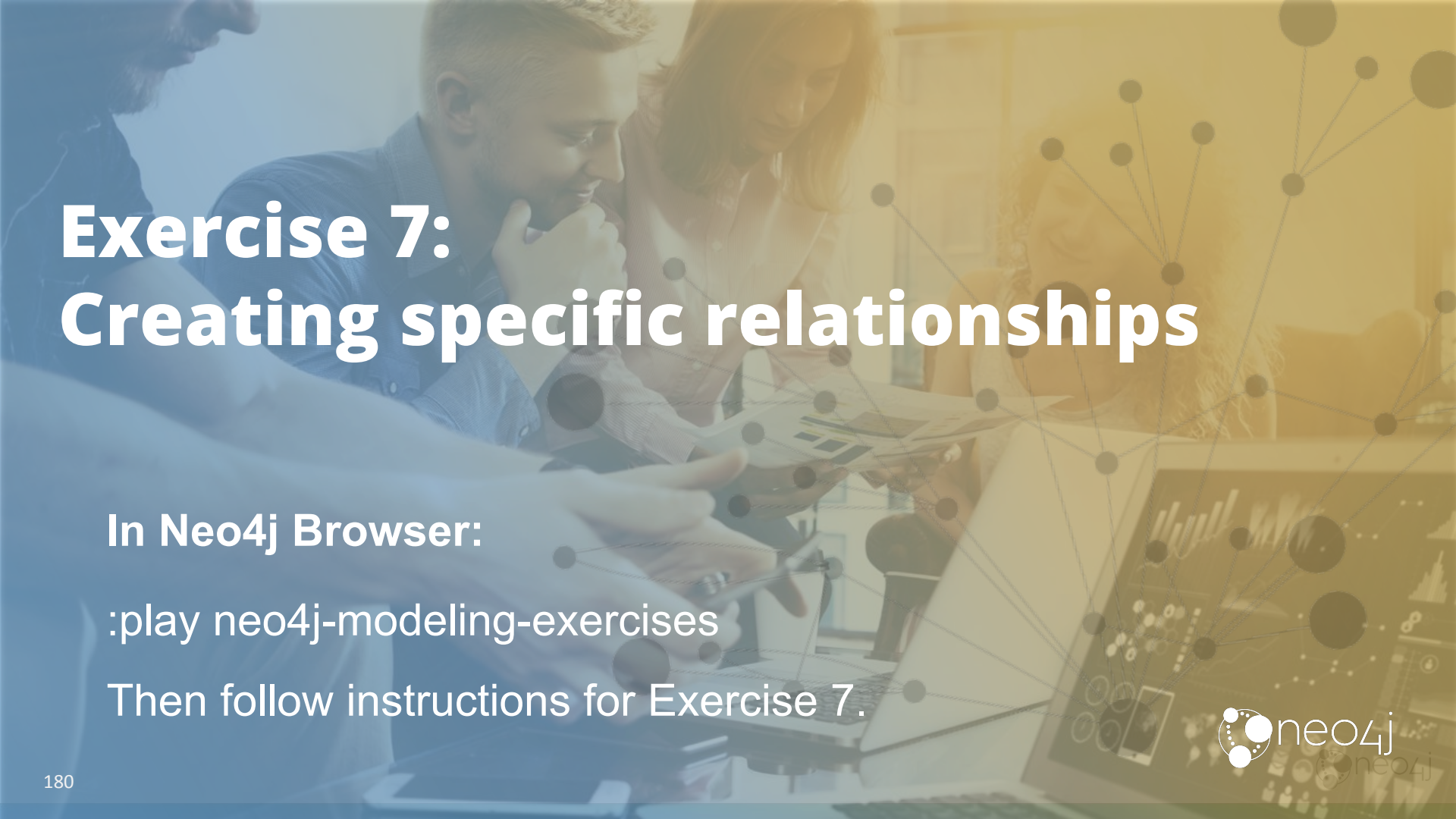


Query using specific relationship

Question: What are the flights that arrive at MDW on 2019-1-3?

```
PROFILE
MATCH (origin:Airport {code: 'LAS'})-
[:`ON_2019-1-3`]->(originDay:AirportDay),
(originDay)<-[:ORIGINATES_FROM]-(flight:Flight),
(flight)-[:LANDS_IN]->(destinationDay),
(destinationDay:AirportDay)<-[:`ON_2019-1-3`]-
(destination:Airport {code: 'MDW'})
RETURN flight.date, flight.number, flight.airline,
flight.departure, flight.arrival
ORDER BY flight.date, flight.departure
```





Exercise 7: Creating specific relationships

In Neo4j Browser:

:play neo4j-modeling-exercises

Then follow instructions for Exercise 7.

Refactoring large graphs

- Why do we need to batch?
- The batch refactoring workflow
- Automate batch refactorings with the APOC library

Why do we need to batch?

Cypher keeps all **transaction state** in memory while running a query, which is fine most of the time.

When refactoring the graph, however, this state can get very large and may result in an OutOfMemory exception.

Adapt your heap size to match, or operate in batches. For example increase these values for the server in the **neo4j.conf** file:

dbms.memory.heap.initial_size=2G (default is 512m)
dbms.memory.heap.max_size=2G (default is 1G)

The batch refactoring workflow

1. Tag all the nodes we need to process with a temporary label (for example Process).

```
MATCH (f:Flight)
SET f:Process
```

2. Iterate over a subset of nodes flagged with the temporary label (using LIMIT):
   a. Execute the refactoring code.
   b. Remove the temporary label from the nodes.
   c. Return a count of how many rows were processed in the iteration.

3. Once the count reaches 0, then we've finished.

Iteration during the batch refactoring

```
MATCH (flight:Process)
WITH flight LIMIT 500

MATCH
(origin:Airport)<-[:ORIGINATES_FROM]-(flight)-[:LANDS_IN]->(destination:Airport)

MERGE (originAirportDay:AirportDay {airportDayId: origin.code + "_" +
flight.date})
ON CREATE SET originAirportDay.date = flight.date

MERGE (destinationAirportDay:AirportDay {airportDayId: destination.code + "_" +
flight.date})
ON CREATE SET destinationAirportDay.date = flight.date

MERGE (origin)-[:HAS_DAY]->(originAirportDay)
MERGE (originAirportDay)<-[:ORIGINATES_FROM]-(flight)
MERGE (flight)-[:LANDS_IN]-(destinationAirportDay)
MERGE (destination)-[:HAS_DAY]->(destinationAirportDay)

REMOVE flight:Process
RETURN COUNT(*)
```

Using APOC for the batching

```
$ call apoc.help('apoc.periodic')
```

Table | Text | Code

type	name	text	signature	roles
"procedure"	"apoc.periodic.cancel"	"apoc.periodic.cancel(name) - cancel job with the given name"	"apoc.periodic.cancel(name :: STRING?) :: (name :: STRING?, delay :: INTEGER?, rate :: INTEGER?, done :: BOOLEAN?, cancelled :: BOOLEAN?)"	null
"procedure"	"apoc.periodic.commit"	"apoc.periodic.commit(statement,params) - runs the given statement in separate transactions until it returns 0"	"apoc.periodic.commit(statement :: STRING?, params = {} :: MAP?) :: (updates :: INTEGER?, executions :: INTEGER?, runtime :: INTEGER?, batches :: INTEGER?, failedBatches :: INTEGER?, batchErrors :: MAP?, failedCommits :: INTEGER?, commitErrors :: MAP?, wasTerminated :: BOOLEAN?)"	null
"procedure"	"apoc.periodic.countdown"	"apoc.periodic.countdown('name',statement,repeat-rate-in-seconds) submit a repeatedly-called background statement until it returns 0"	"apoc.periodic.countdown(name :: STRING?, statement :: STRING?, rate :: INTEGER?) :: (name :: STRING?, delay :: INTEGER?, rate :: INTEGER?, done ::	null

Started streaming 9 records after 121 ms and completed after 125 ms.

Using apoc.periodic.commit()

```
call apoc.periodic.commit('
MATCH (flight:Process)
WITH flight LIMIT {limit}

MATCH (origin:Airport)<-[:ORIGINATES_FROM]-(flight)-[:LANDS_IN]->(destination:Airport)

MERGE (originAirportDay:AirportDay {airportDayId: origin.code + "_" + flight.date})
ON CREATE SET originAirportDay.date = flight.date

MERGE (destinationAirportDay:AirportDay {airportDayId: destination.code + "_" + flight.date})
ON CREATE SET destinationAirportDay.date = flight.date

MERGE (origin)-[:HAS_DAY]->(originAirportDay)
MERGE (originAirportDay)<-[:ORIGINATES_FROM]-(flight)
MERGE (flight)-[:LANDS_IN]-(destinationAirportDay)
MERGE (destination)-[:HAS_DAY]->(destinationAirportDay)

REMOVE flight:Process
RETURN COUNT(*)

',{limit:500}
)
```

Result of apoc.periodic.commit()

```
$ call apoc.periodic.commit(' MATCH (flight:Process) WITH flight LIMIT {limit}  MATCH (origin:Airport)<-[:…
```

Table | Text | Code

updates	executions	runtime	batches	failedBatches	batchErrors	failedCommits	commitErrors	wasTerminated
110002	221	5	222	0	{ }	0	{ }	false

Started streaming 1 records after 5998 ms and completed after 5998 ms.

Exercise 8: Refactoring large graphs

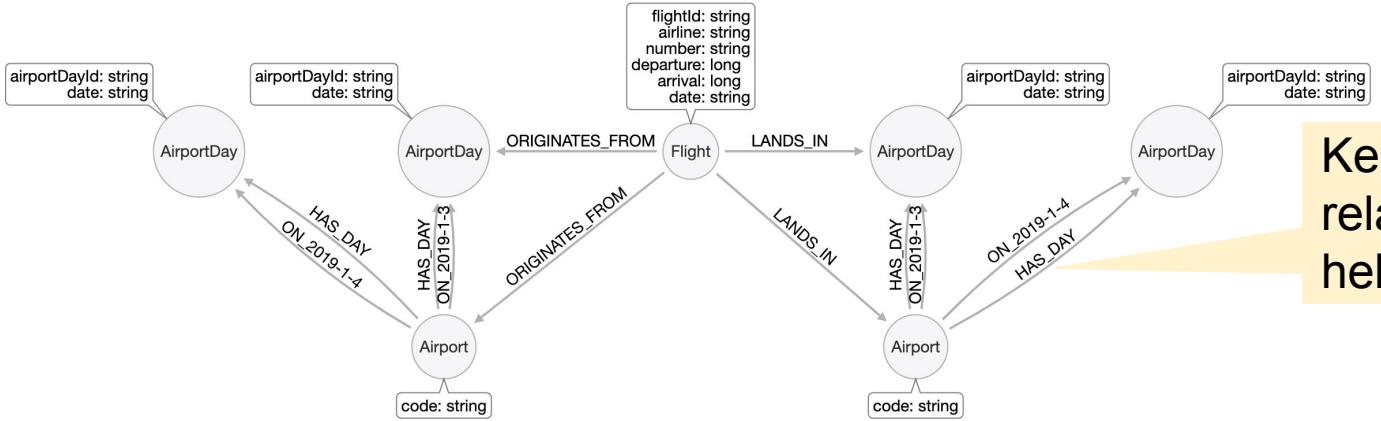
In Neo4j Browser:

:play neo4j-modeling-exercises

Then follow instructions for Exercise 8.

Working with your model

- Which query is faster?
- Which query is easier to code?
- What happens when data is added to the graph?
- What happens when data is deleted from the graph?



Keeping both relationships may help.

These queries do slightly better with specific relationships

Using general relationship (1761 db hits):

```
MATCH (origin:Airport {code: 'LAS'})-[:HAS_DAY]->(originDay:AirportDay),
(originDay)<-[:ORIGIN]-(flight:Flight)
WHERE originDay.date = '2019-1-3'
RETURN flight.date, flight.number, flight.airline, flight.departure, flight.arrival
ORDER BY flight.date, flight.departure
```

Using specific relationships (1700 db hits):

```
MATCH (origin:Airport {code: 'LAS'})-[:`ON_2019-1-3` ]->(originDay:AirportDay),
(originDay)<-[:ORIGINATES_FROM]-(flight:Flight)
RETURN flight.date, flight.number, flight.airline, flight.departure, flight.arrival
ORDER BY flight.date, flight.departure
```

These queries perform ~ the same

Using general relationship (47846 db hits):

```
MATCH (origin:Airport {code: 'LAS'})-[:HAS_DAY]->(originDay:AirportDay),
(originDay)<-[:ORIGINATES_FROM]-(flight:Flight)
WHERE originDay.date STARTS WITH '2019-1'
RETURN flight.date, flight.number, flight.airline, flight.departure, flight.arrival
ORDER BY flight.date, flight.departure
```

Using specific relationships (47815 db hits):

```
MATCH (origin:Airport {code: 'LAS'})-[:`ON_2019-1-1` | :`ON_2019-1-2` | :`ON_2019-1-3` |
                :`ON_2019-1-4` |  :`ON_2019-1-5` | :`ON_2019-1-6` | :`ON_2019-1-7` | :`ON_2019-1-8` |
                :`ON_2019-1-9` | :`ON_2019-1-10` | :`ON_2019-1-11` | :`ON_2019-1-12` | :`ON_2019-1-13` |
                :`ON_2019-1-14` | :`ON_2019-1-15` |  :`ON_2019-1-16` | :`ON_2019-1-17` | :`ON_2019-1-18`
|
                :`ON_2019-1-19` | :`ON_2019-1-20` | :`ON_2019-1-21` | :`ON_2019-1-22` | :`ON_2019-1-23`
|
                :`ON_2019-1-24` | :`ON_2019-1-25` |  :`ON_2019-1-26` | :`ON_2019-1-27` | :`ON_2019-1-28`
|
                :`ON_2019-1-29` | :`ON_2019-1-30` | :`ON_2019-1-31`   ]->(originDay:AirportDay),
(originDay)<-[:ORIGINATES_FROM]-(flight:Flight)
RETURN flight.date, flight.number, flight.airline, flight.departure, flight.arrival
ORDER BY flight.date, flight.departure
```

Model affect how you add data

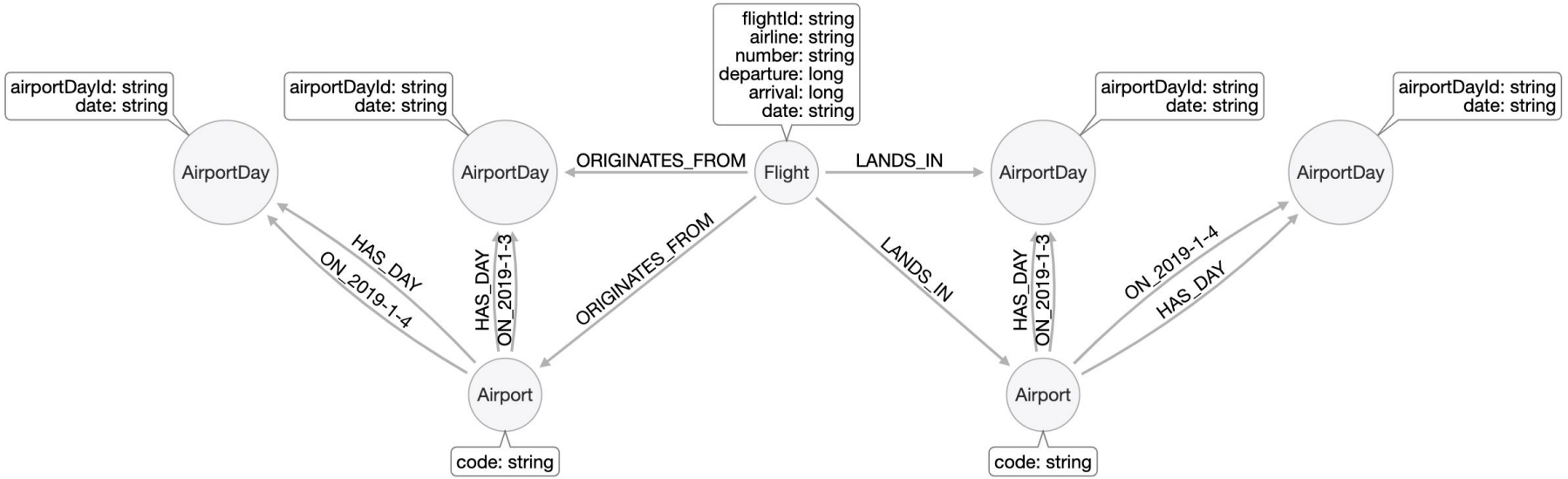
If we want to add a new day we have to add two relationships instead of one.

```
MATCH (airport {code:  'LAX'})
MERGE (airportDay:AirportDay {airportDayId: 'LAX_2019-2-1'})
ON CREATE SET airportDay.date = '2019-2-1'
CREATE (airport)-[:HAS_DAY]->(airportDay)
CREATE (airport)-[:`2019-2-1`]->(airportDay)
```

Archiving data for a day

```
//write the flight data to a CSV file
CALL apoc.export.csv.query("MATCH (f:Flight {date: '2019-1-1'})
return  f.airline as Airline, f.arrival as Arrival, f.date as
Date, f.departure as Departure, f.flightId as `Flight ID`,
f.number as FlightNumber",'/tmp/flights.csv', {})

// remove the AirportDay nodes and their relationships
MATCH (airportDay:AirportDay {date: '2019-1-1'})
DETACH DELETE airportDay;

// remove the Flight nodes and their relationships
MATCH (flight:Flight {date:'2019-1-1'})
DETACH DELETE flight
```

Evaluating the model



Do we really need the Airport node?

Exercise 9: Maintaining models

In Neo4j Browser:

:play neo4j-modeling-exercises

Then follow instructions for Exercise 9.

Check your understanding

Question 1

Suppose you have loaded some sample data into a graph for your model. Next you analyze your queries to see how they perform. What Cypher clause do you use to analyze your queries?

Select the correct answer.

- ❏ APOC
- ❏ ANALYZE
- ❏ EXPLAIN
- ❏ PROFILE

Answer 1

Suppose you have loaded some sample data into a graph for your model. Next you analyze your queries to see how they perform. What Cypher clause do you use to analyze your queries?

Select the correct answer.

❏ APOC

❏ ANALYZE

❏ EXPLAIN

✔ PROFILE

Question 2

Suppose you have decided to refactor your graph to create specific relationship types between A and B nodes, in addition to the general relationship type, IN. The B nodes have a property, state, that you want to use to create the IN_<state> relationship between the A and B Nodes. How do you do this in Cypher?

Select the correct answer.

❏ MERGE (a:A)-[:IN_`b.state`]->(b:B)

❏ MERGE (a:A)-[:IN_+`b.state`]->(b:B)

❏ CREATE SPECIFIC RELATIONSHIP (a:A)-[:IN_`b.state`]->(b:B)

❏ MATCH (a:A)-[r:LOCATED_IN]->(b:B) CALL apoc.create.relationship(startNode(r,'LOCATED_IN_' + b.state, {}, endNode(r)

Answer 2

Suppose you have decided to refactor your graph to create specific relationship types between A and B nodes, in addition to the general relationship type, IN. The B nodes have a property, state, that you want to use to create the IN_<state> relationship between the A and B Nodes. How do you do this in Cypher?

Select the correct answer.

❏ MERGE (a:A)-[:IN_`b.state`]->(b:B)

❏ MERGE (a:A)-[:IN_+`b.state`]->(b:B)

❏ CREATE SPECIFIC RELATIONSHIP (a:A)-[:IN_`b.state`]->(b:B)

✔ MATCH (a:A)-[r:LOCATED_IN]->(b:B) CALL apoc.create.relationship(startNode(r,'LOCATED_IN_' + b.state, {}, endNode(r)

Question 3

What APOC procedure do you use the perform batch processing of Cypher code on large sets of data?

Select the correct answer.

- ❏ apoc.periodic.commit()
- ❏ apoc.batch.process()
- ❏ apoc.process.batch()
- ❏ apoc.atomic.update()

Answer 3

What APOC procedure do you use the perform batch processing of Cypher code on large sets of data?

Select the correct answer.

✔️ apoc.periodic.commit()

❑ apoc.batch.process()

❑ apoc.process.batch()

❑ apoc.atomic.update()

Summary

You should now be able to:

- Create a graph using sample data for a simple graph data model.
- Load data for a graph from CSV files.
- Profile queries against the graph.
- Refactor the graph to add intermediate nodes.
- Refactor the graph to create relationships from nodes.
- Refactor the graph to specialize relationships.
- Refactor a large graph using batching.
- Maintain a graph.

Course feedback

We value your feedback!

Please fill out the feedback form and receive a

Certificate of Completion

bit.ly/neo-survey