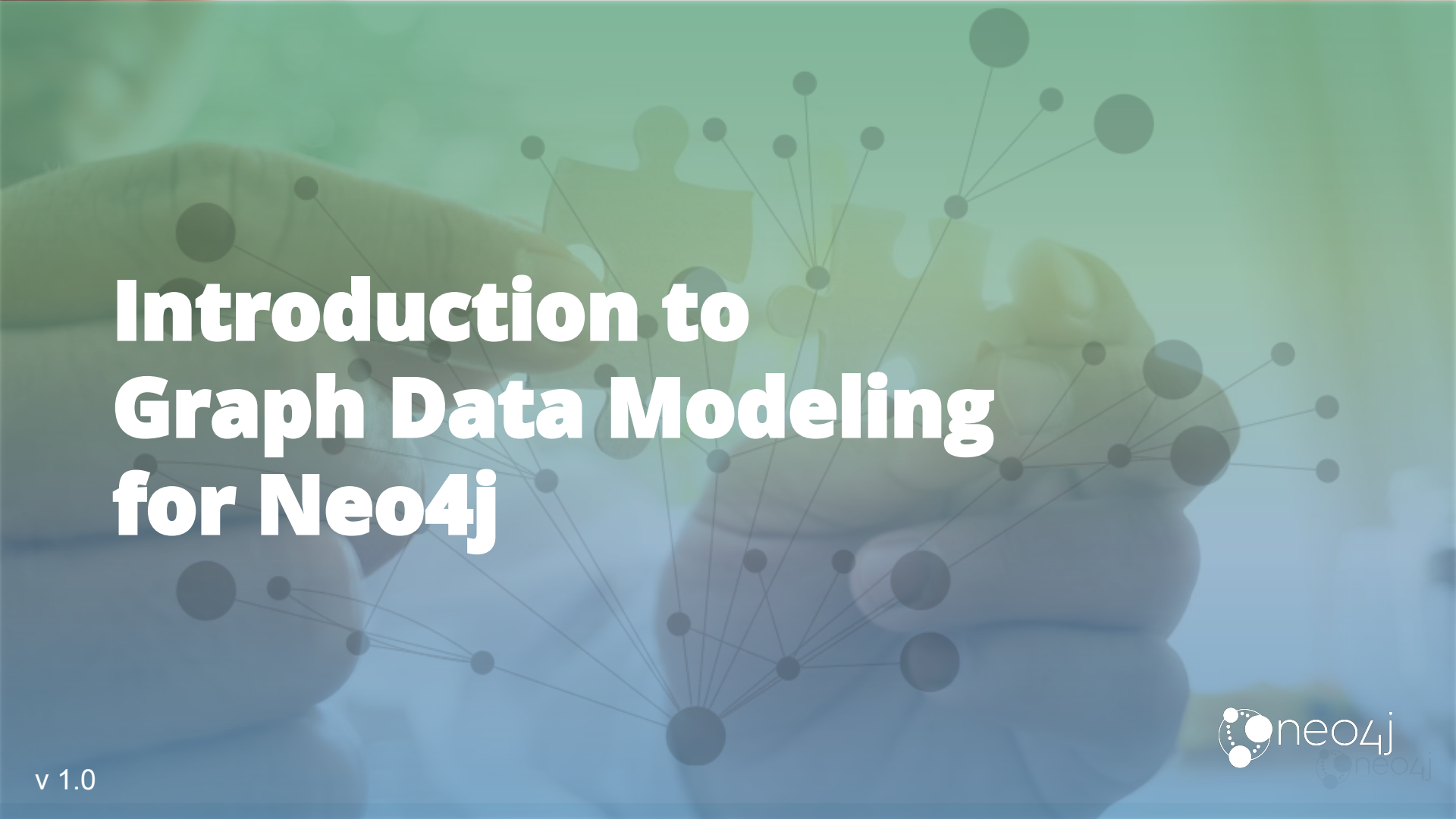

Introduction to
Graph Data Modeling
for Neo4j
v 1.0
neo4j

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.

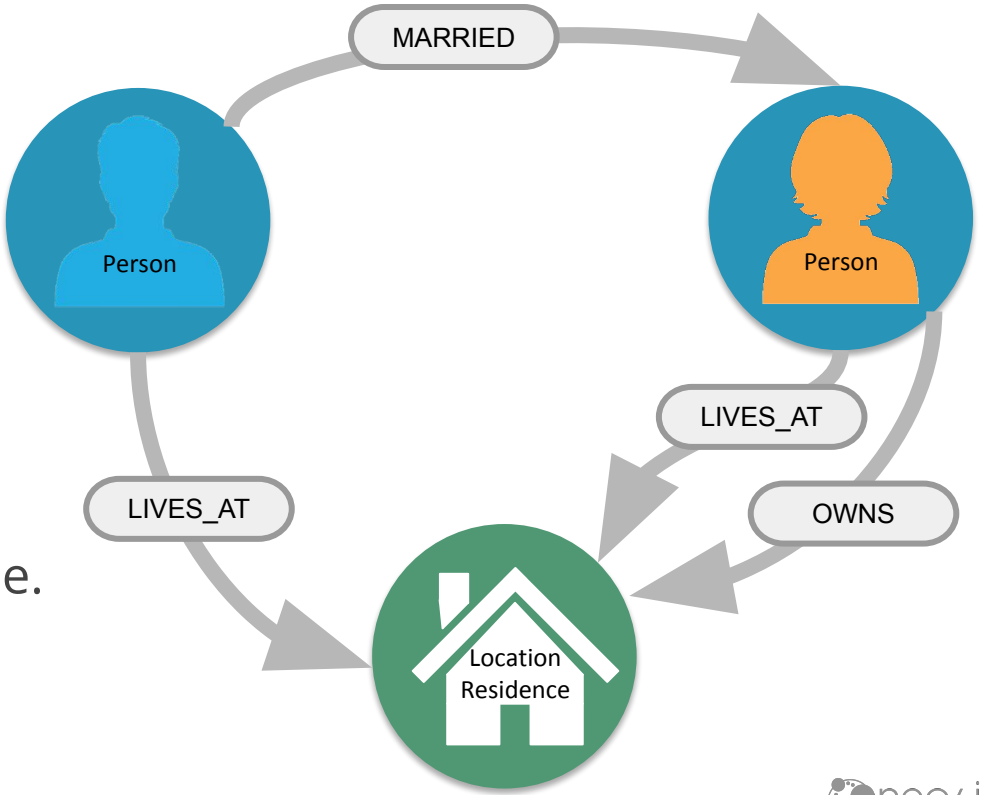
Paths are created and navigated based upon the relationships between nodes.

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.

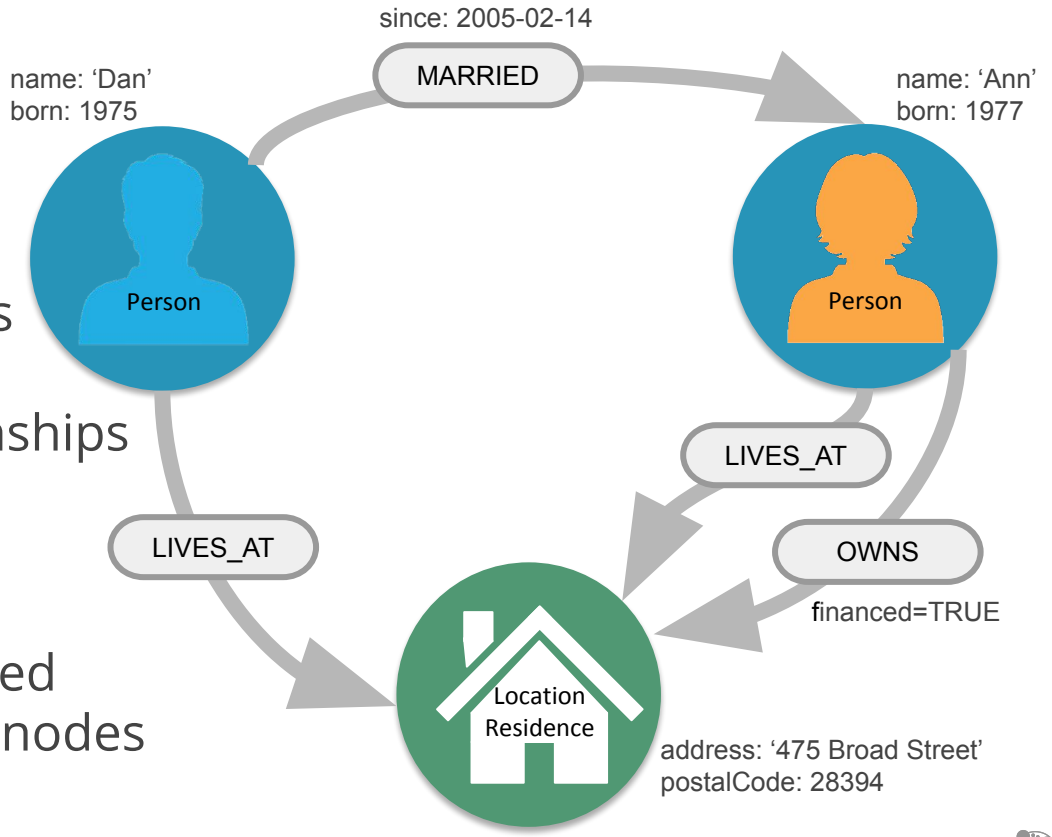
In Neo4j, connections are relationships

- 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

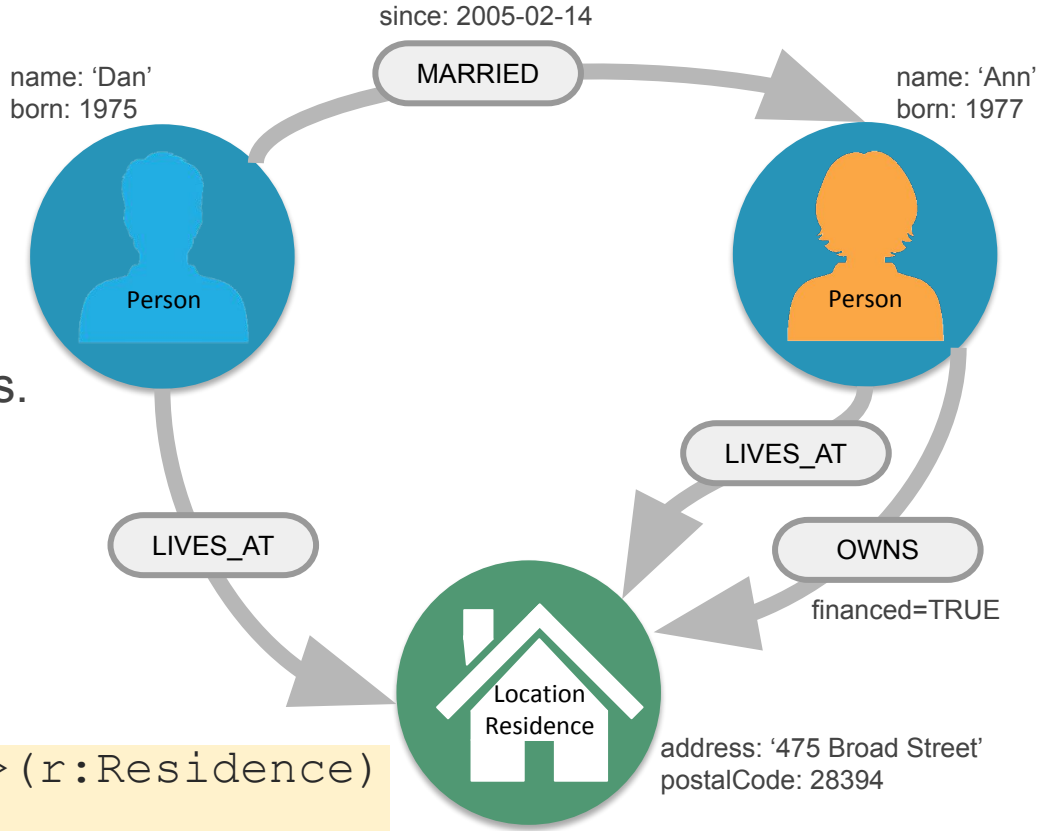
- Adjectives to describe nodes
  - Proper nouns
- Adverbs to describe relationships
  - Strength, weight, quality
- Property:
  - Key/value pair
  - Can be optional or required
  - Values can be unique for nodes



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.

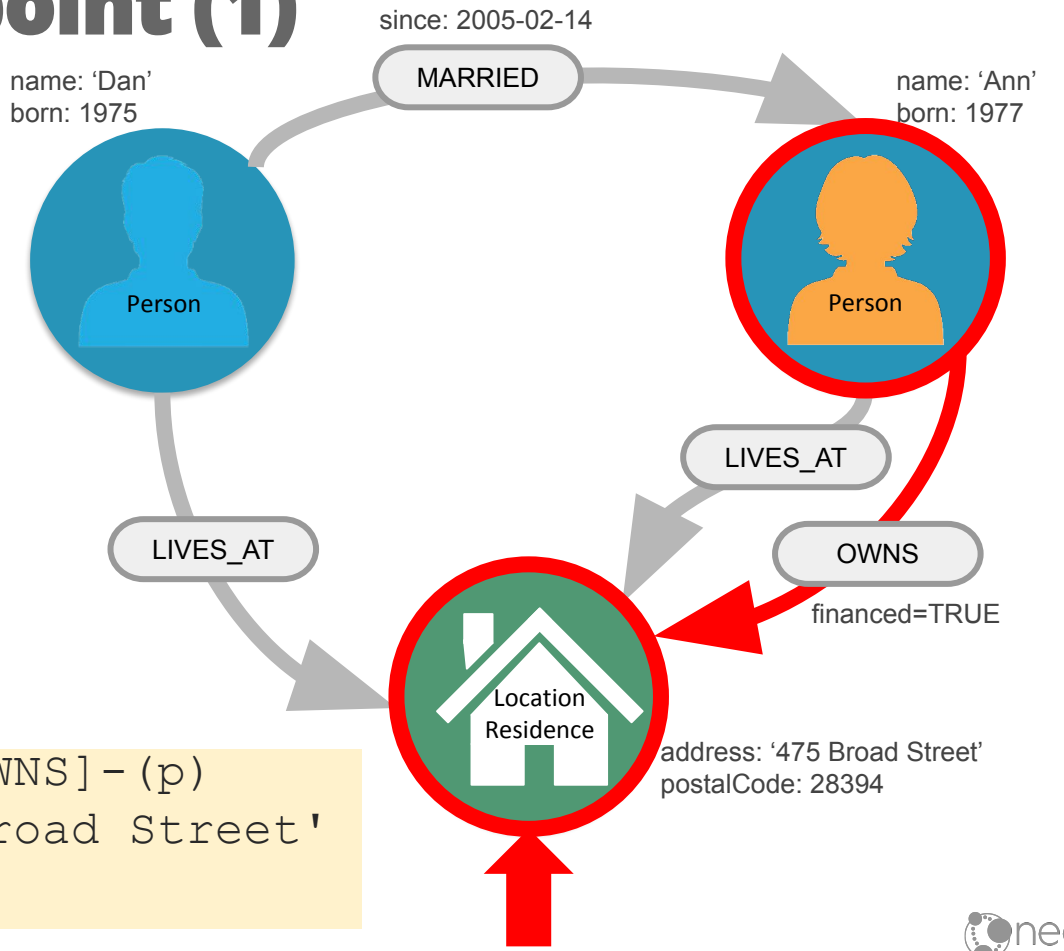


Cypher query:

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

Path is starting point (1)

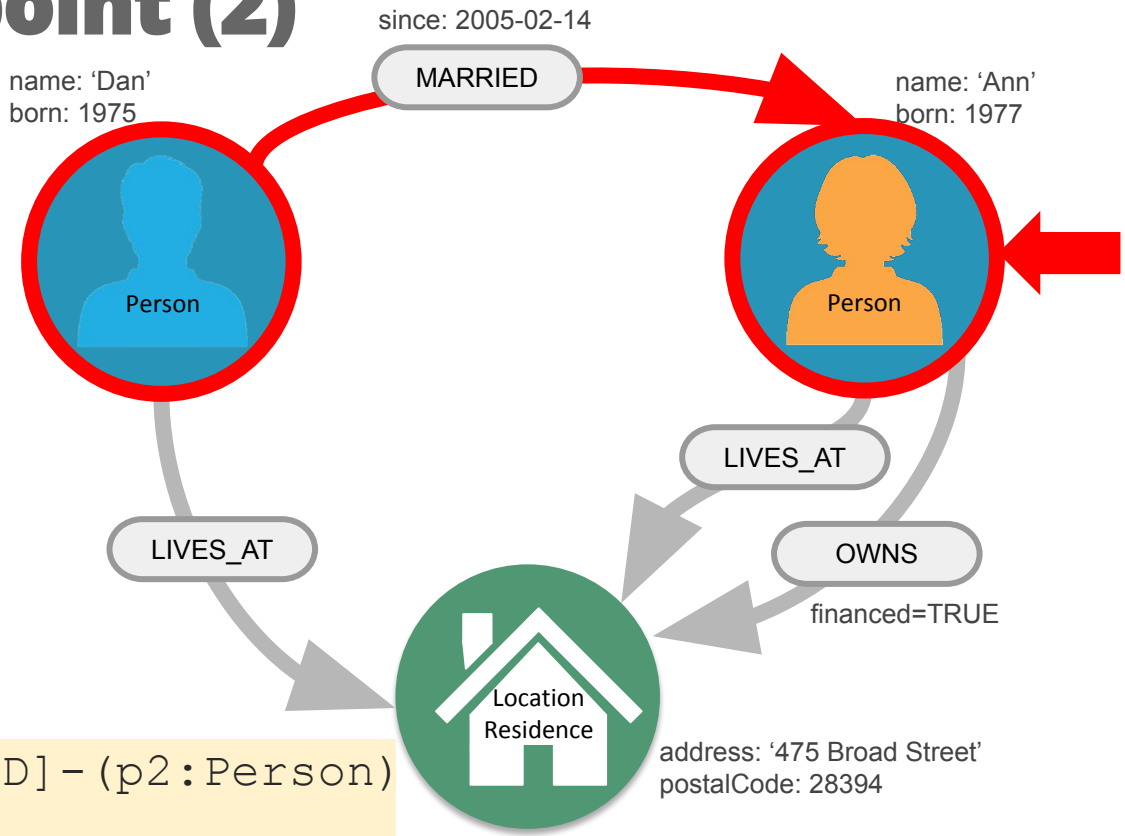
What nodes are visited to find all Person nodes that own the house at 475 Broad Street?




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

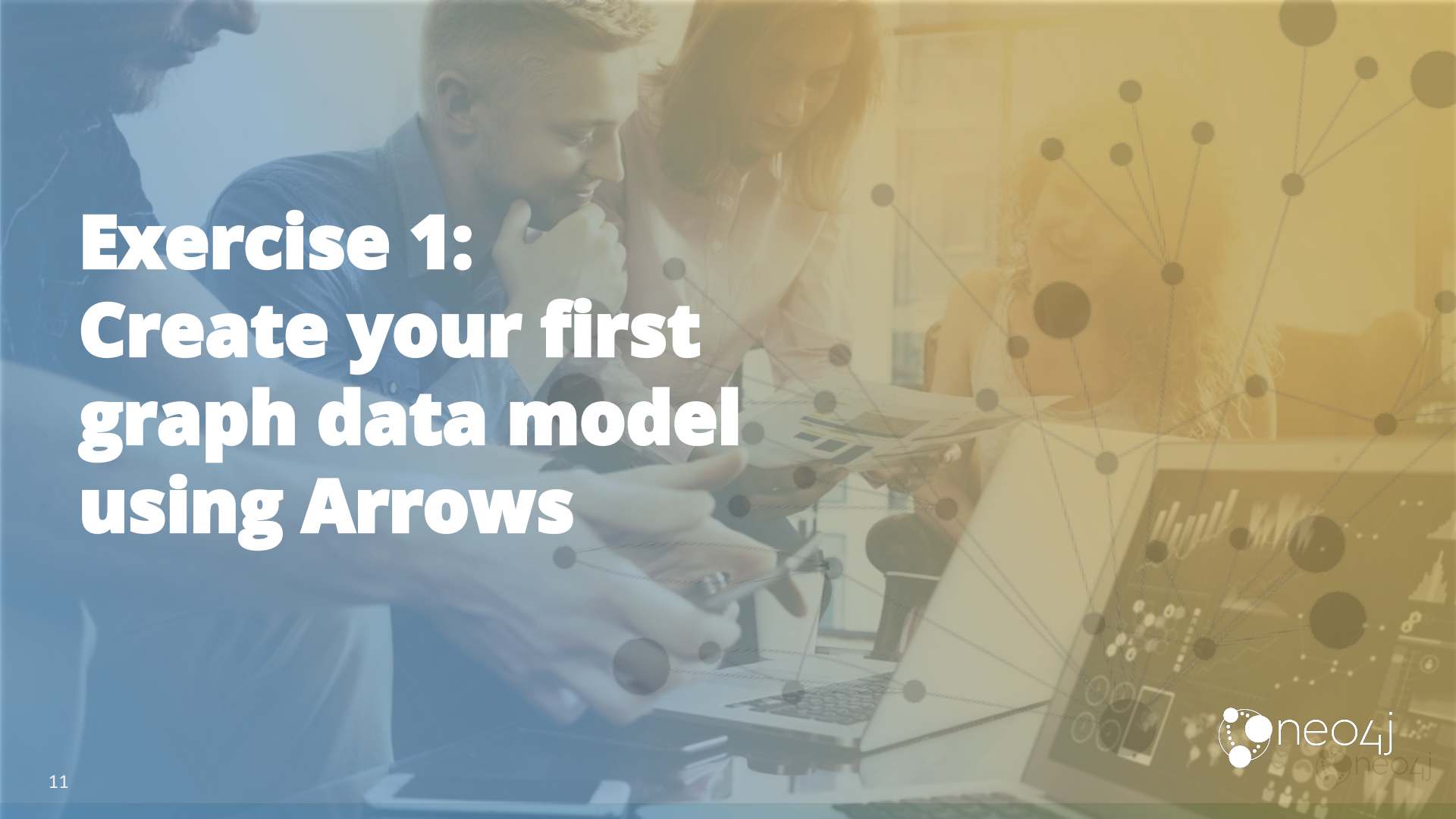
Path is starting point (2)

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

Domain: Knowledge management of the skills of people that work for the same company

Question: I want to know 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.

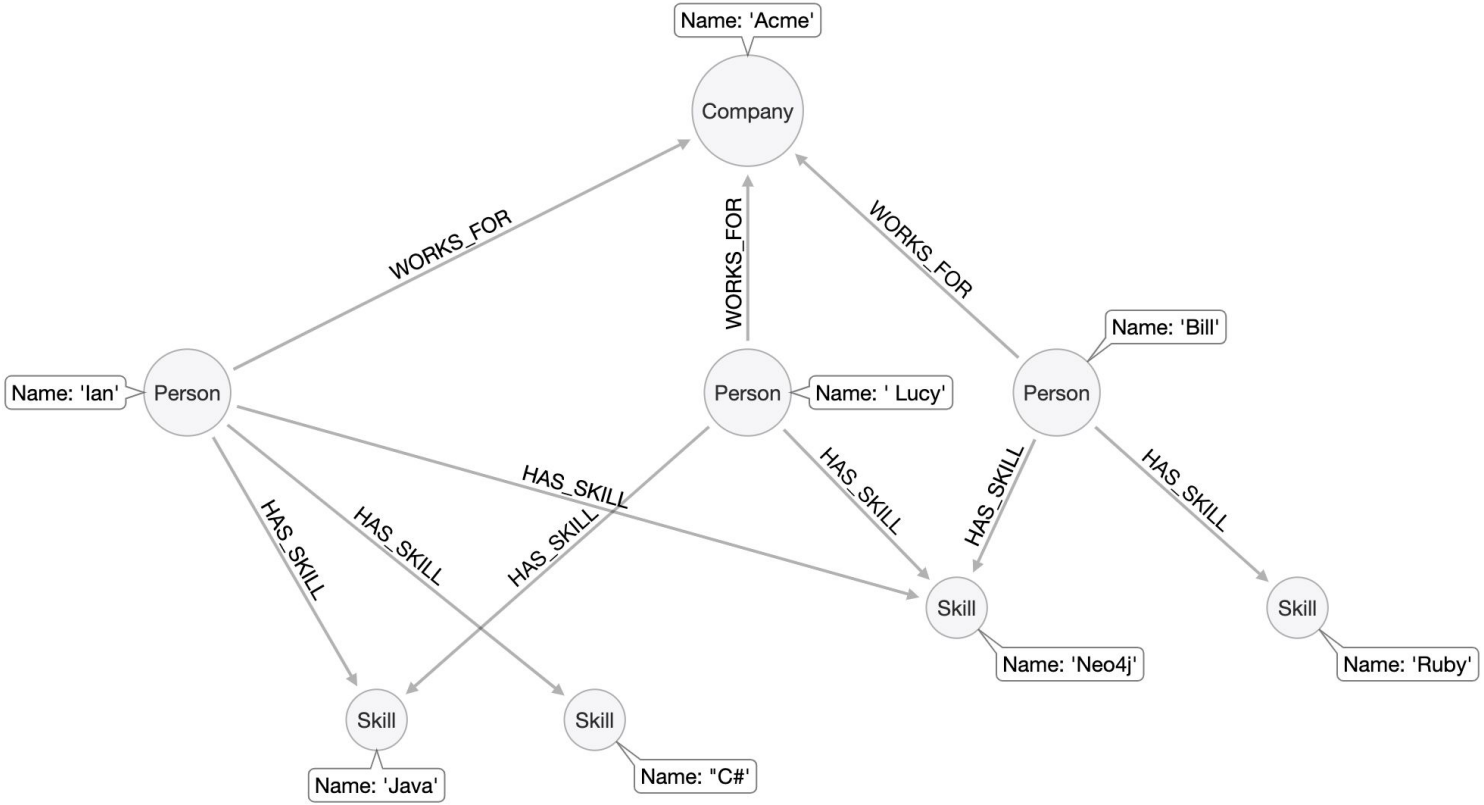
Answer: Person, Company, Skill

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	✔	✔

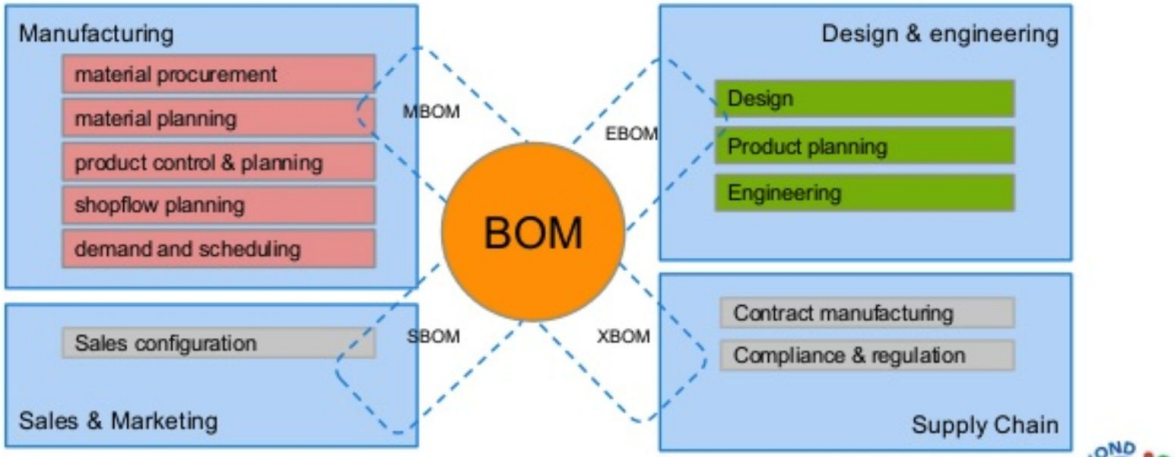
Developing the initial graph data model

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



BOM connects all engineering and non-engineering processes

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

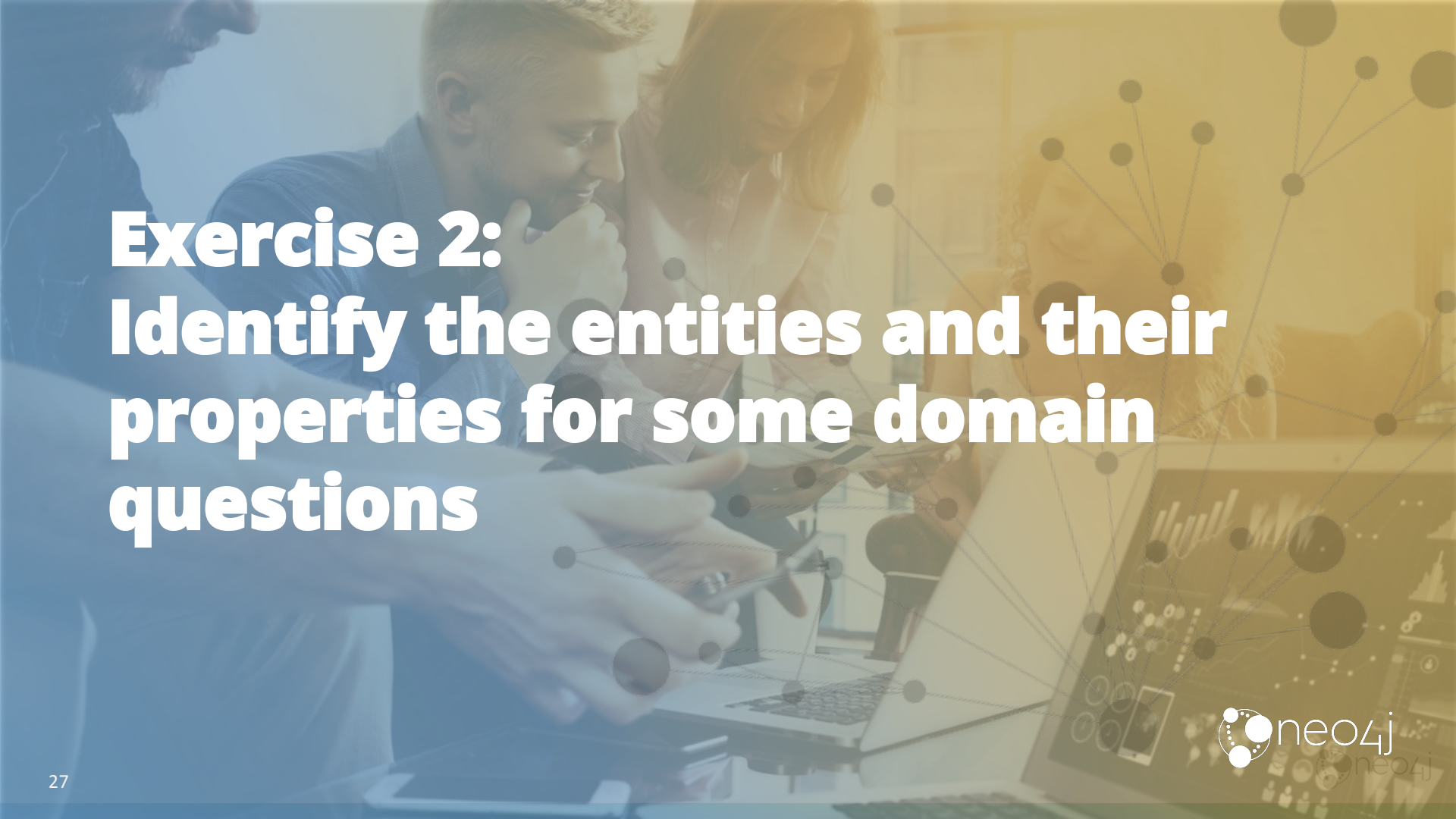
- What parts are needed to make Wood table 40"?
- Do we have enough parts to make 100 Deluxe wood table 60"?
- What products require a table brace?
- How much will the parts cost to make product Wood table 60"?

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

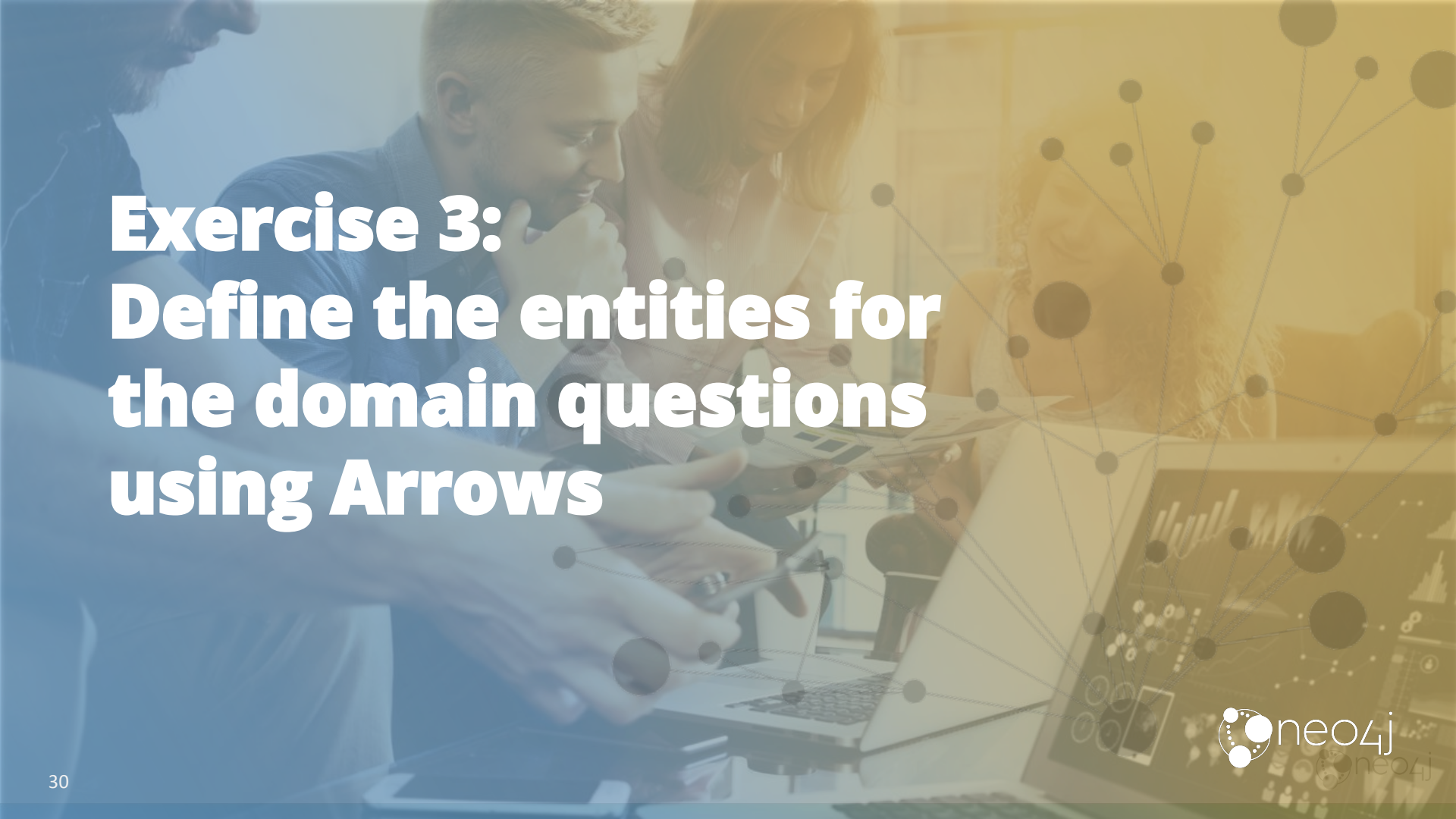
- Name
- ProductID

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

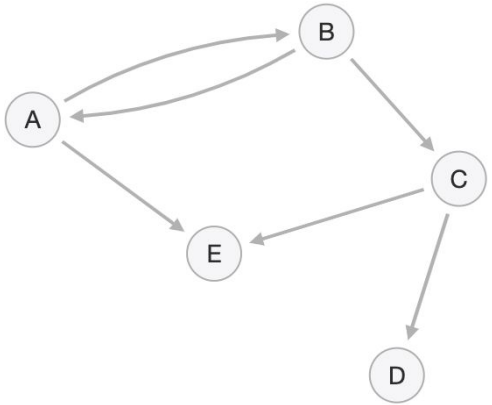
- The purpose of the model is to answer questions about the data.
- Connections are the verbs in the questions derived from use cases.
- Most questions are typically about the the connectedness of the data:
  - What ingredients are used in a recipe?

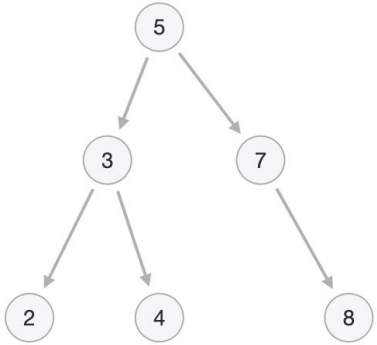
  - Who is married to this person?

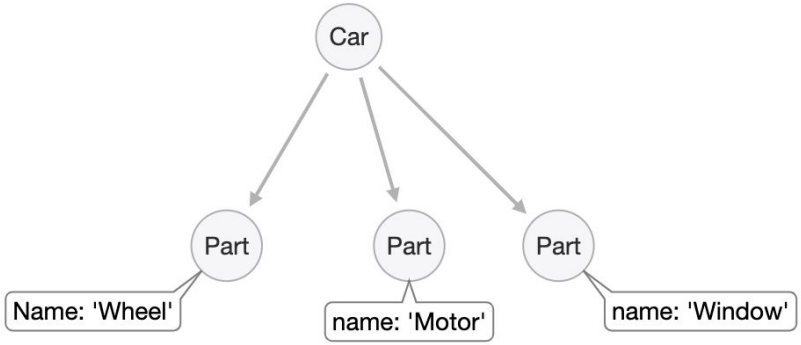
- Exactly **one** node at the end of every relationship (connection).
- A relationship must have direction when it is created.

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

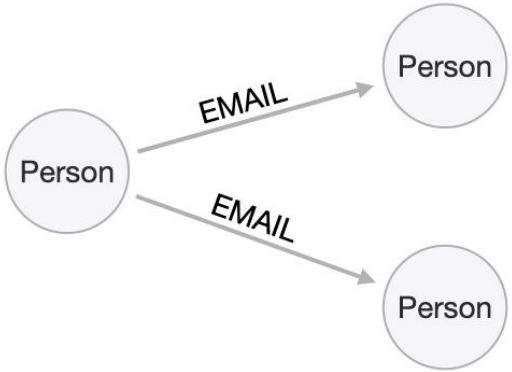


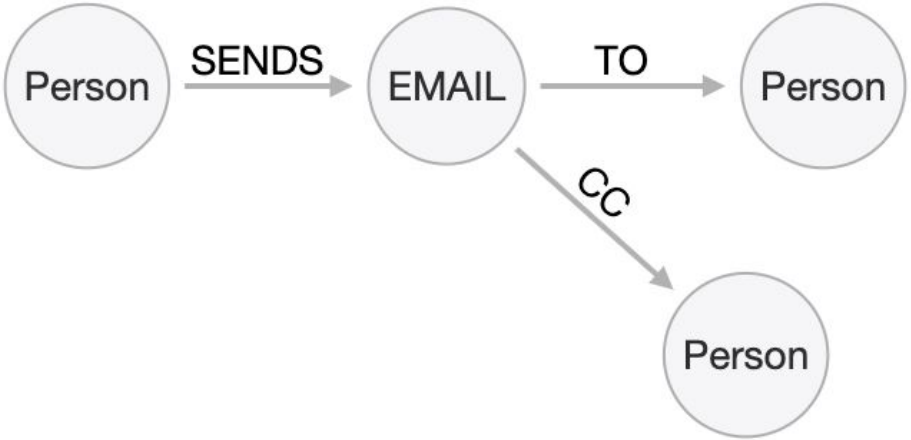




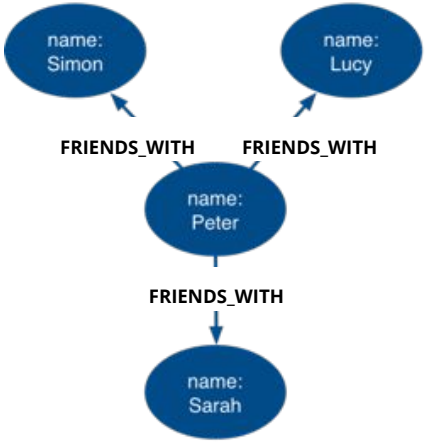
Naming relationships

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

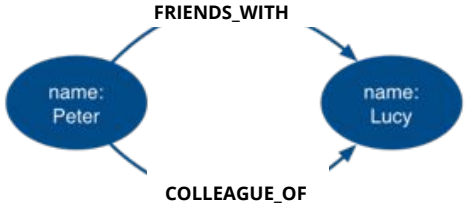




How many relationships?



Nodes can have more than one relationship



Nodes can be connected by more than one relationship

Self relationships are allowed

How important is direction?

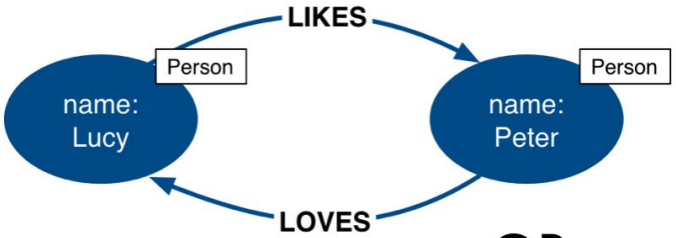
Direction is required for creating the model and in particular for implementing the model (Cypher code) in the underlying Neo4j graph.

Whether direction is used for queries depends on the question:

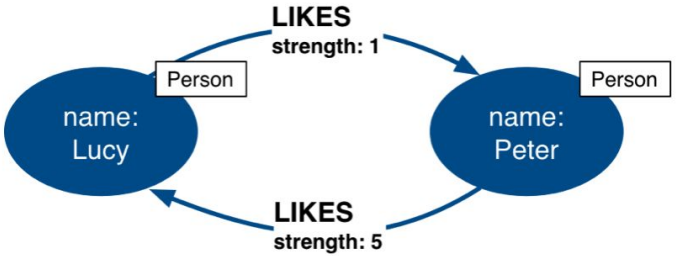
- What are the names of the episodes of the Dr. Who series? (direction not used for the query)
- What episode follows The Ark in Space? (direction used for the query)

Qualifying a relationship

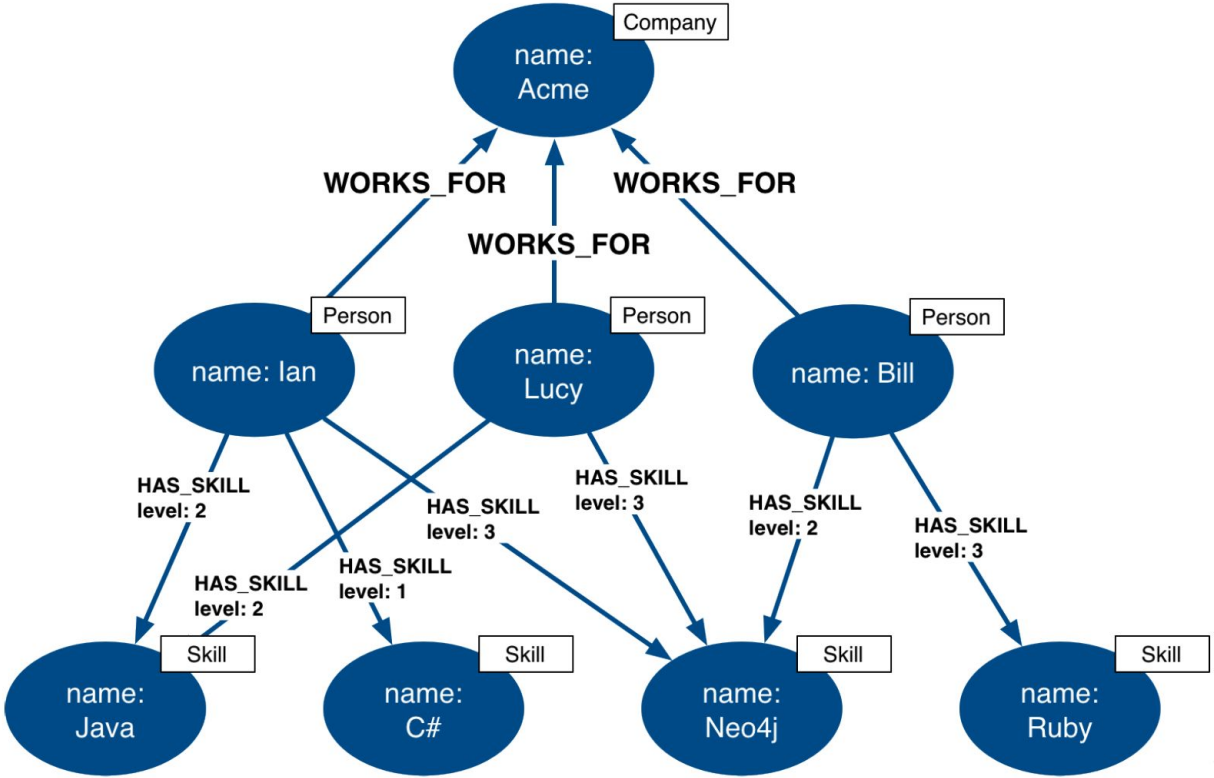
Use properties to describe the weight or quality of the relationship.



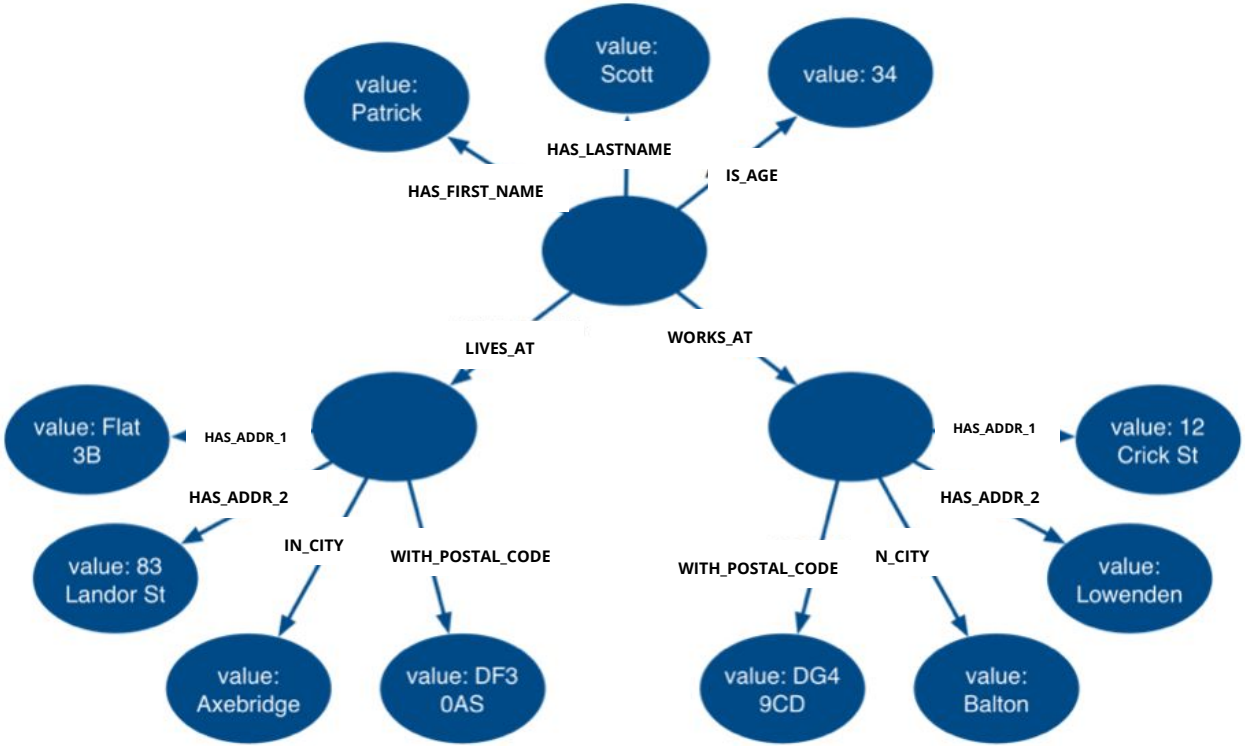
OR



Example: Find expert colleagues



How much fanout will a node have?



Exercise 4: Adding relationships to the model

Exercise 4: Instructions

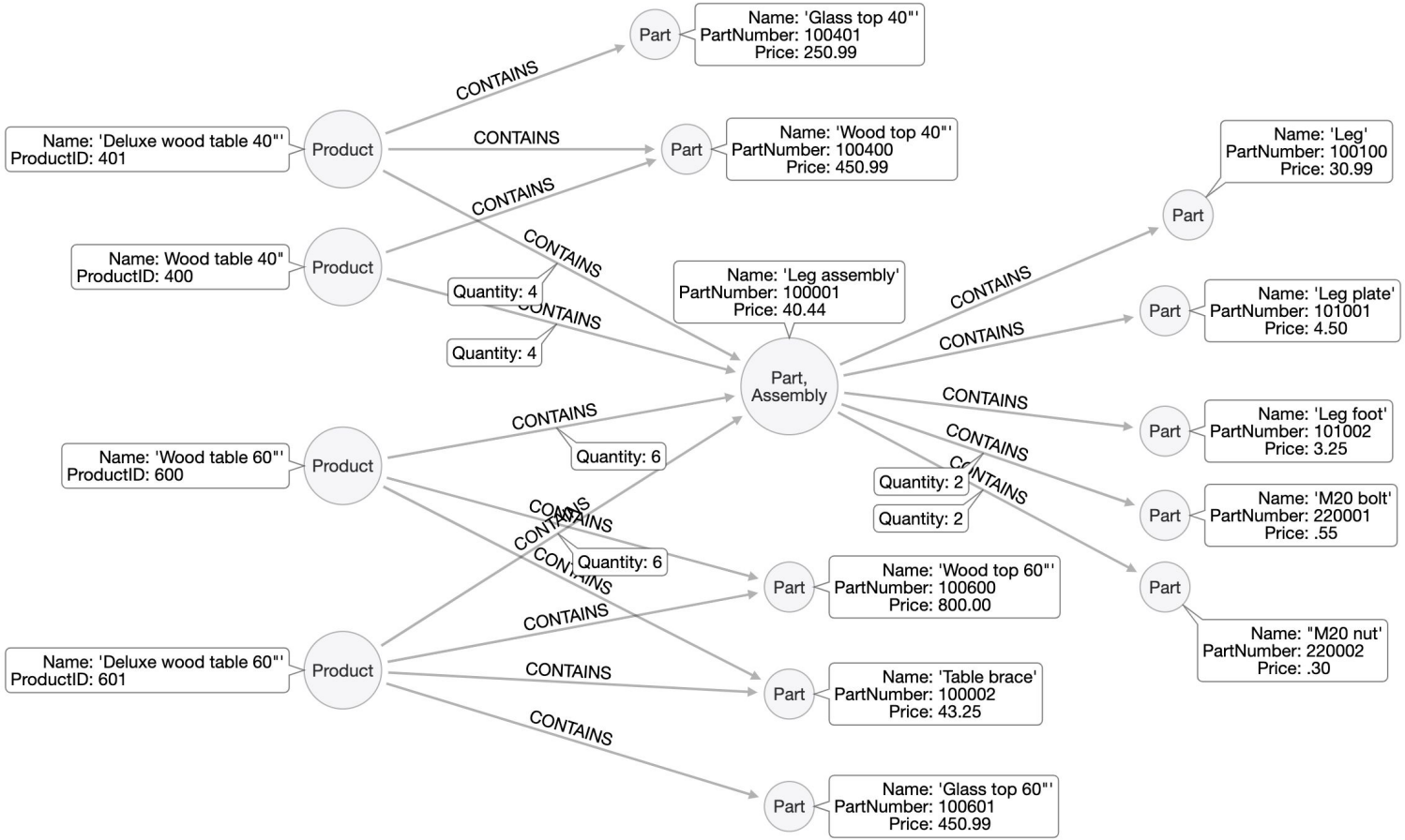
1. Continue with Arrows: http://www.apcjones.com/arrows

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

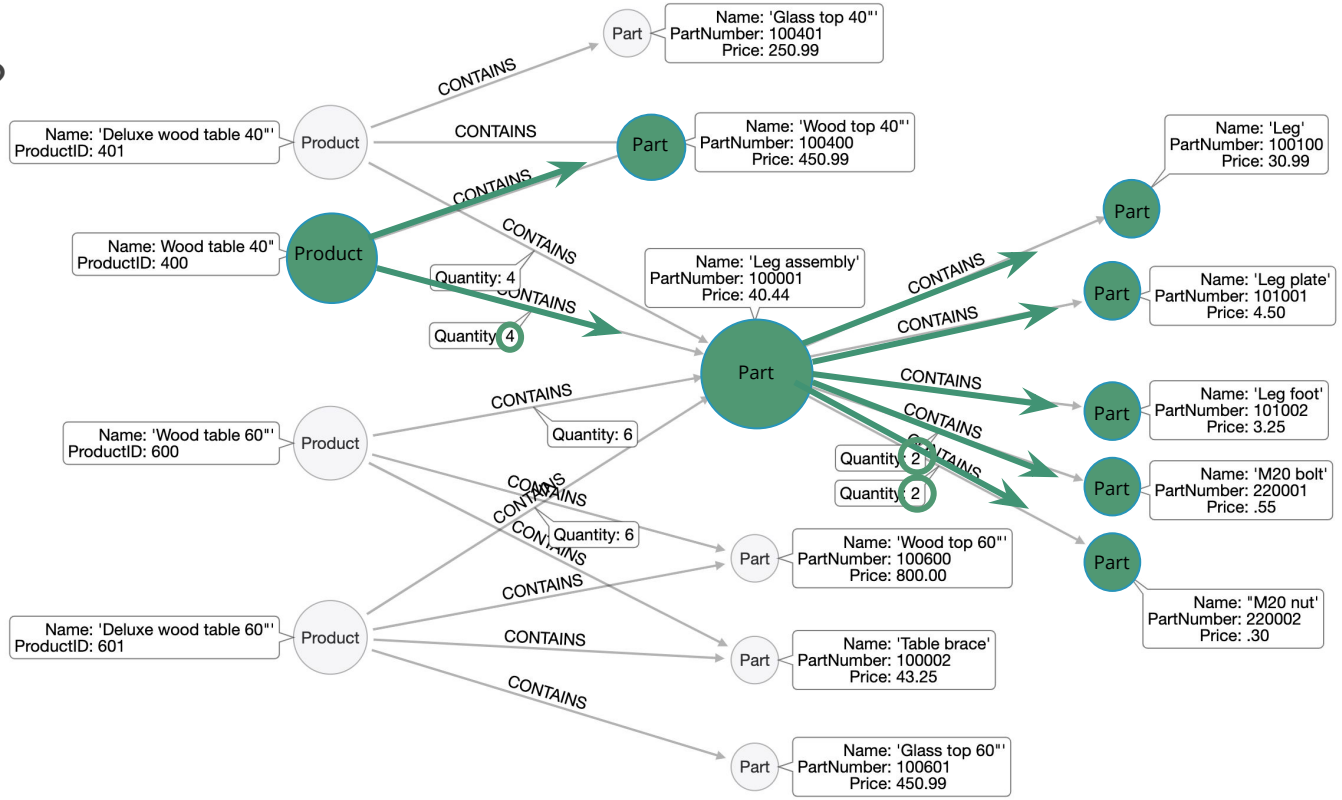


Developing the initial graph data model

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

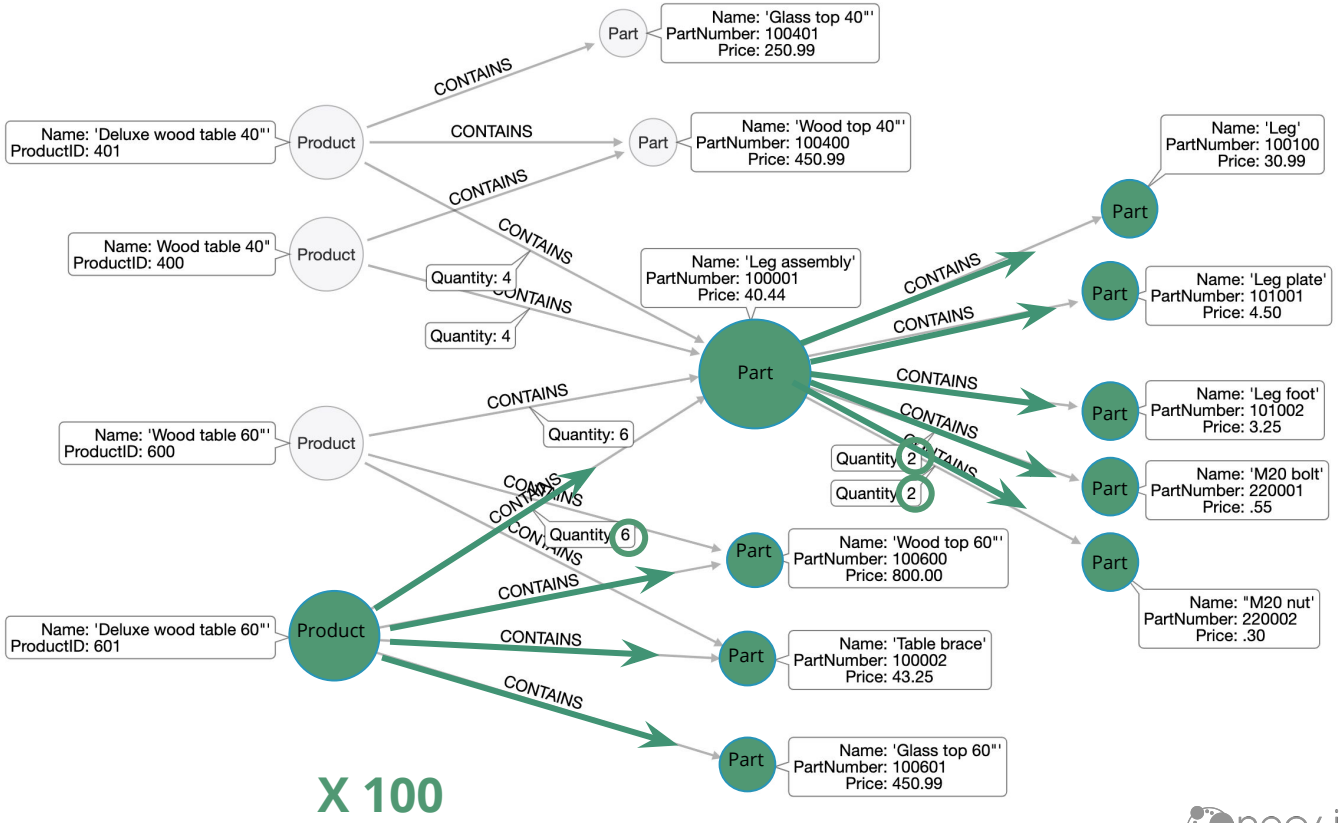
Testing the model

What parts are needed
to make Wood table 40"?



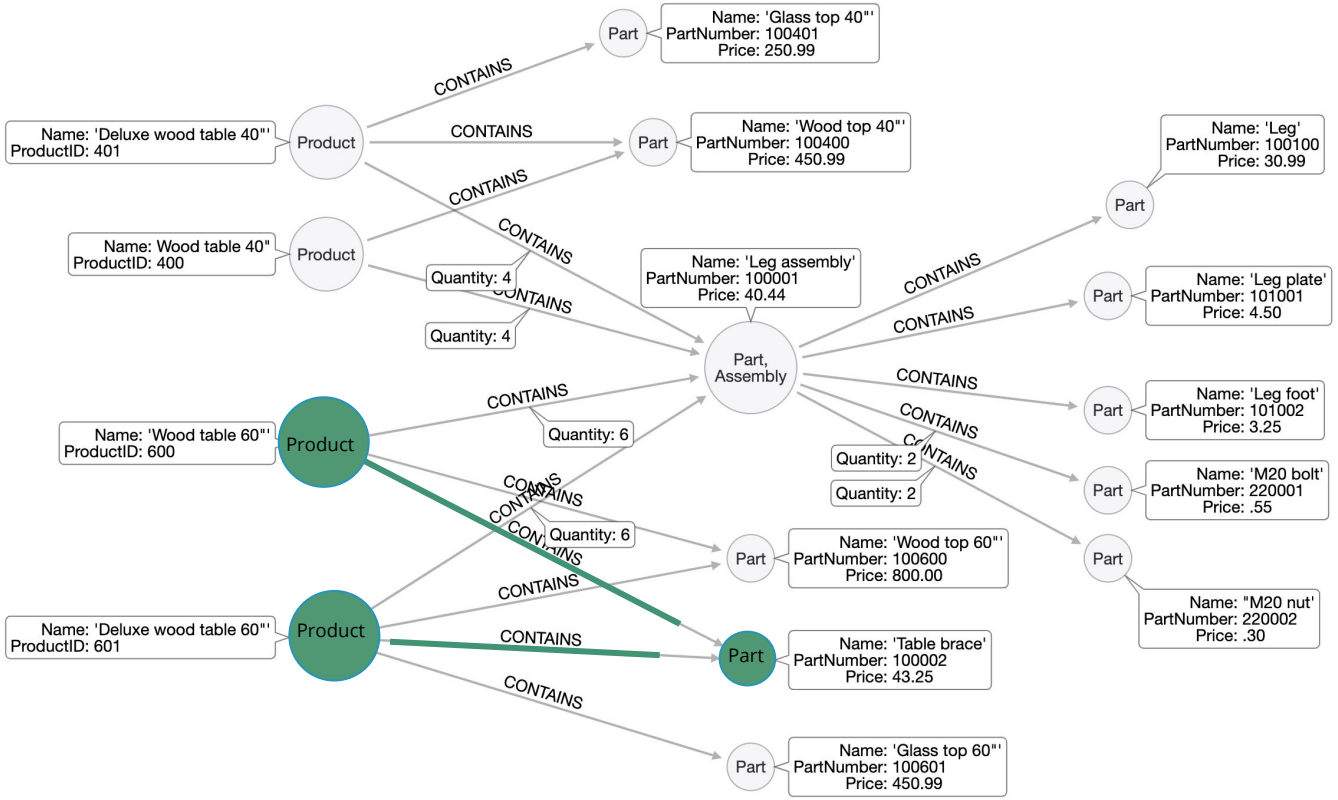
Testing the model

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



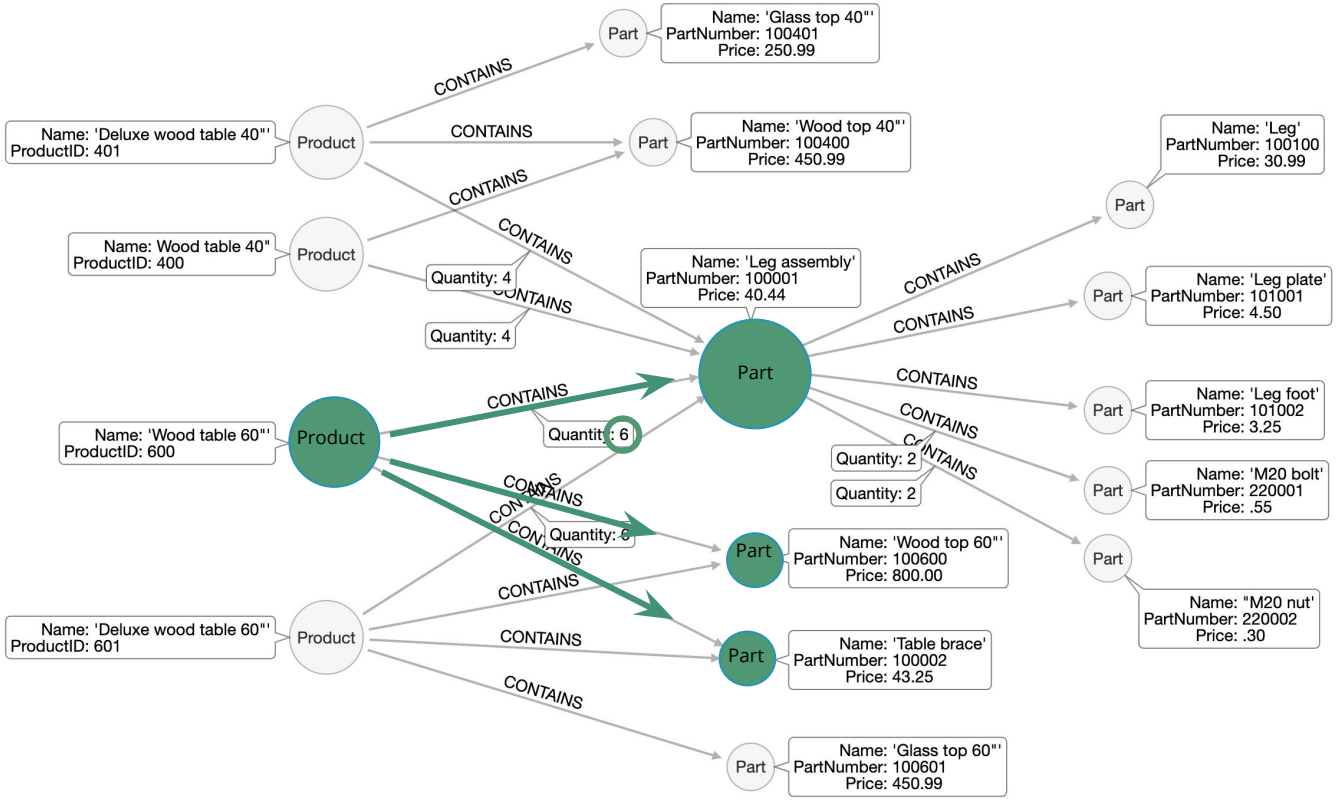
Testing the model

What products require a table brace?



Testing the model

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

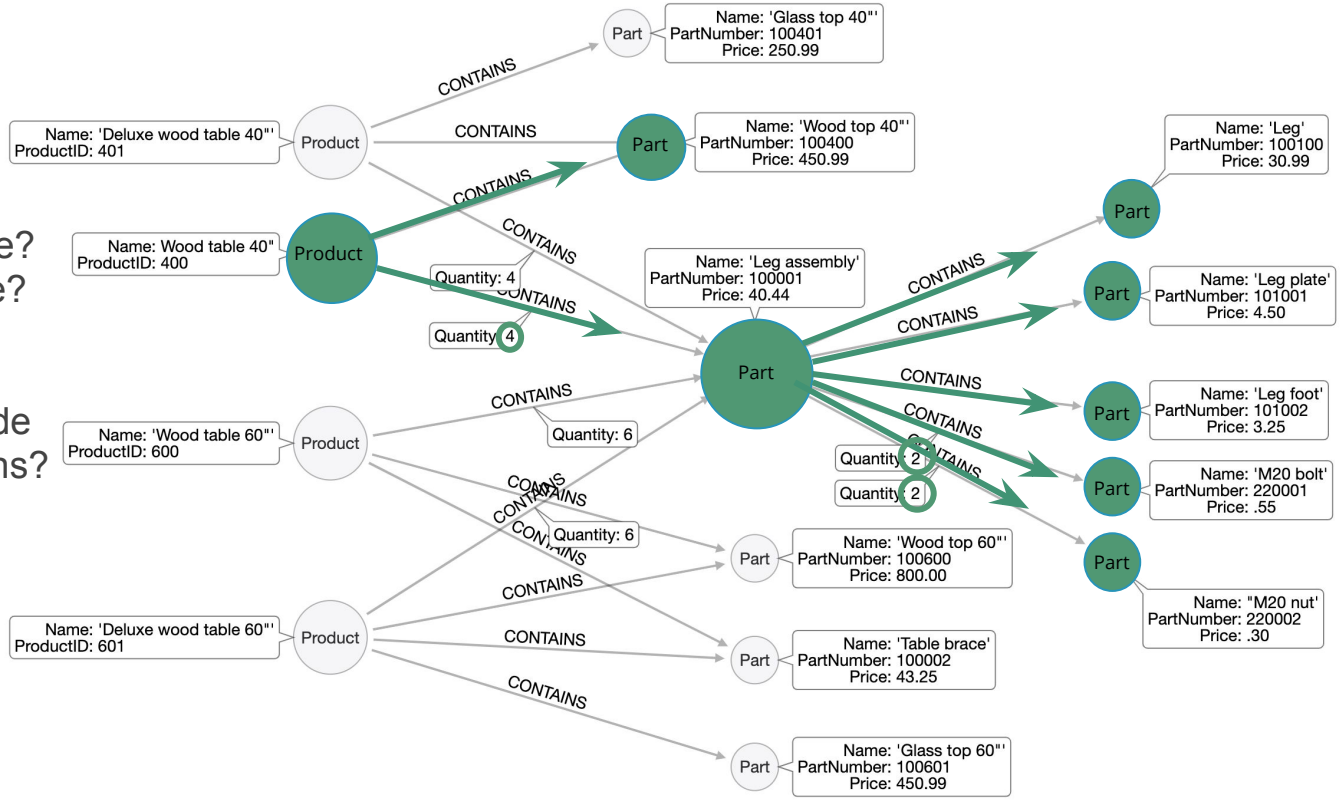


Developing the initial graph data model

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

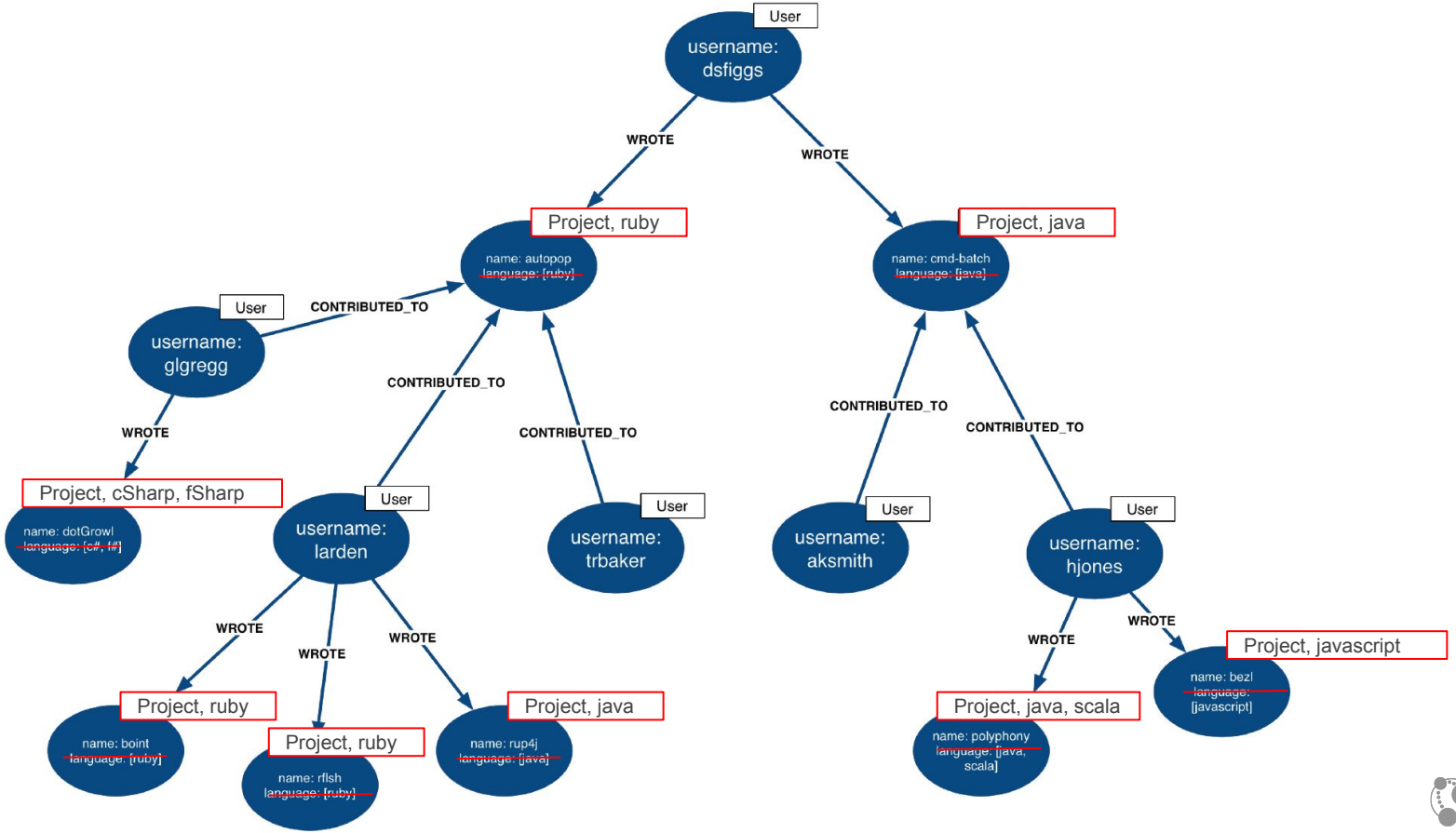
- How many products?
- How many parts?
- How often are products added?
- How often do prices change?
- Are prices based upon time?
- Is inventory part of the model?
- Does the data need to reside in different physical locations?



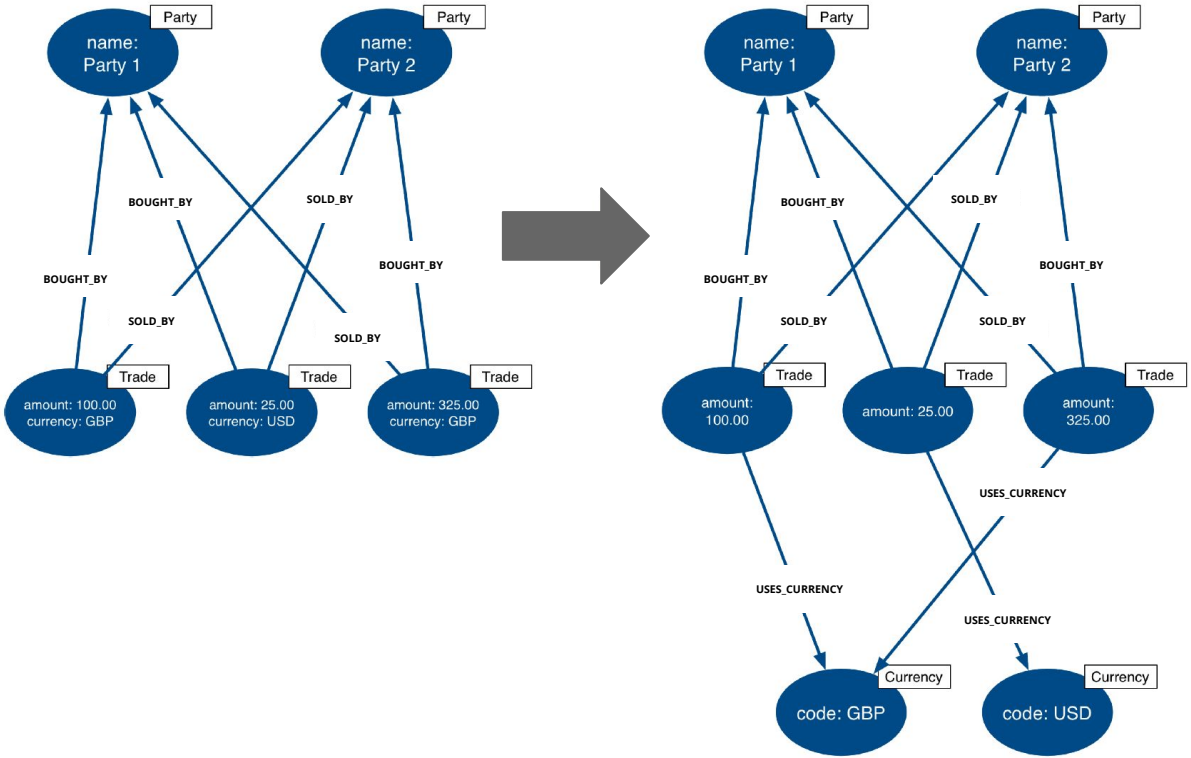
Best practices for graph data modeling

- 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

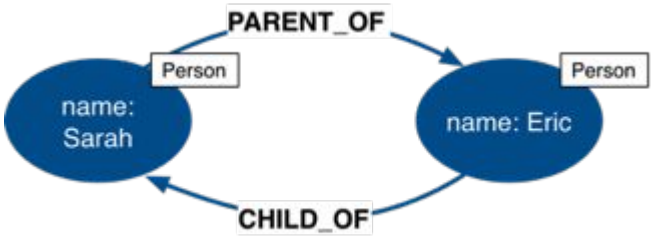
Best practice: Group nodes



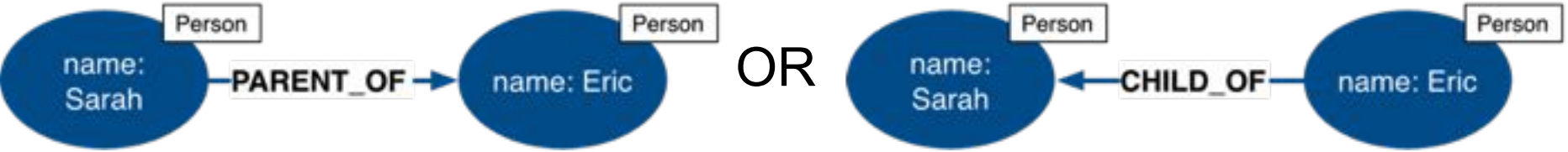
Best practice: Choose relationships over node properties



Best practice: No symmetric relationships



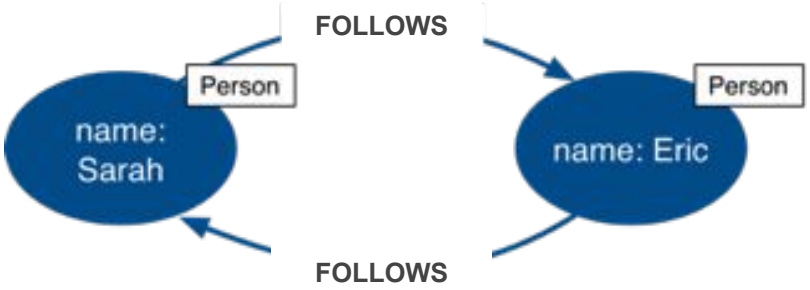
You should never do this.



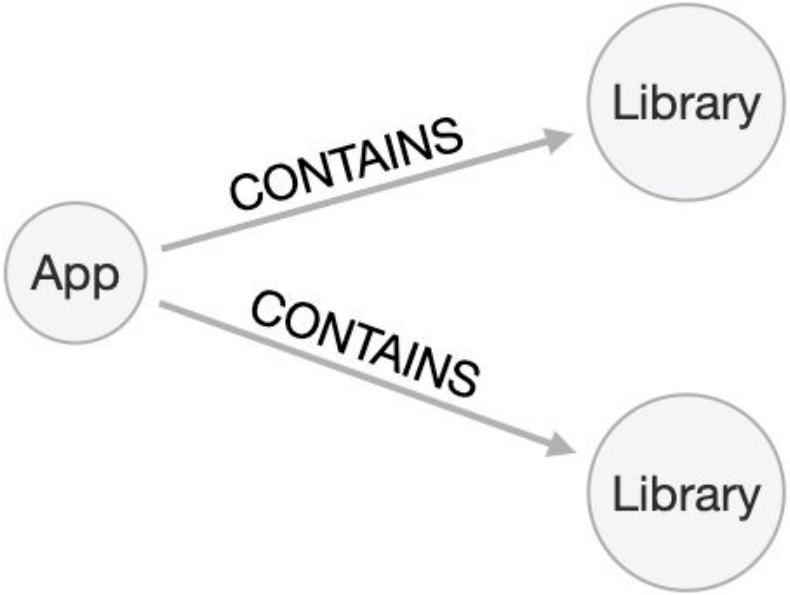
Note: Relationships require space in the graph so minimizing their numbers is always a good thing.

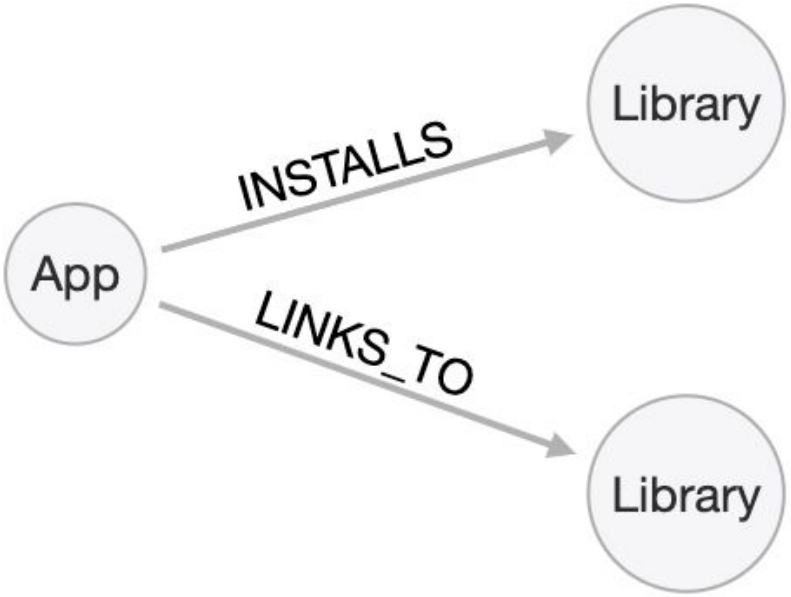
But, symmetric can be used here

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

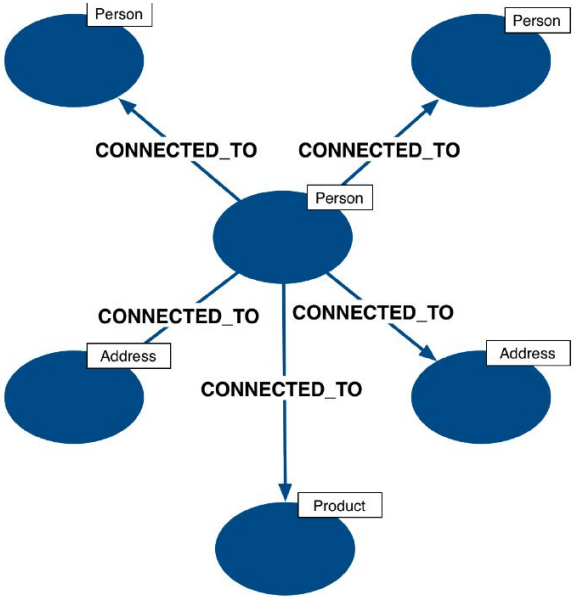


Best practice: Specific relationship types



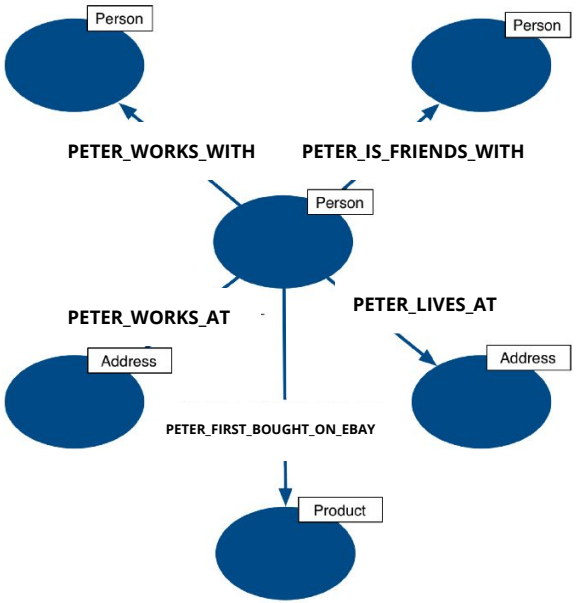


Relationship granularity

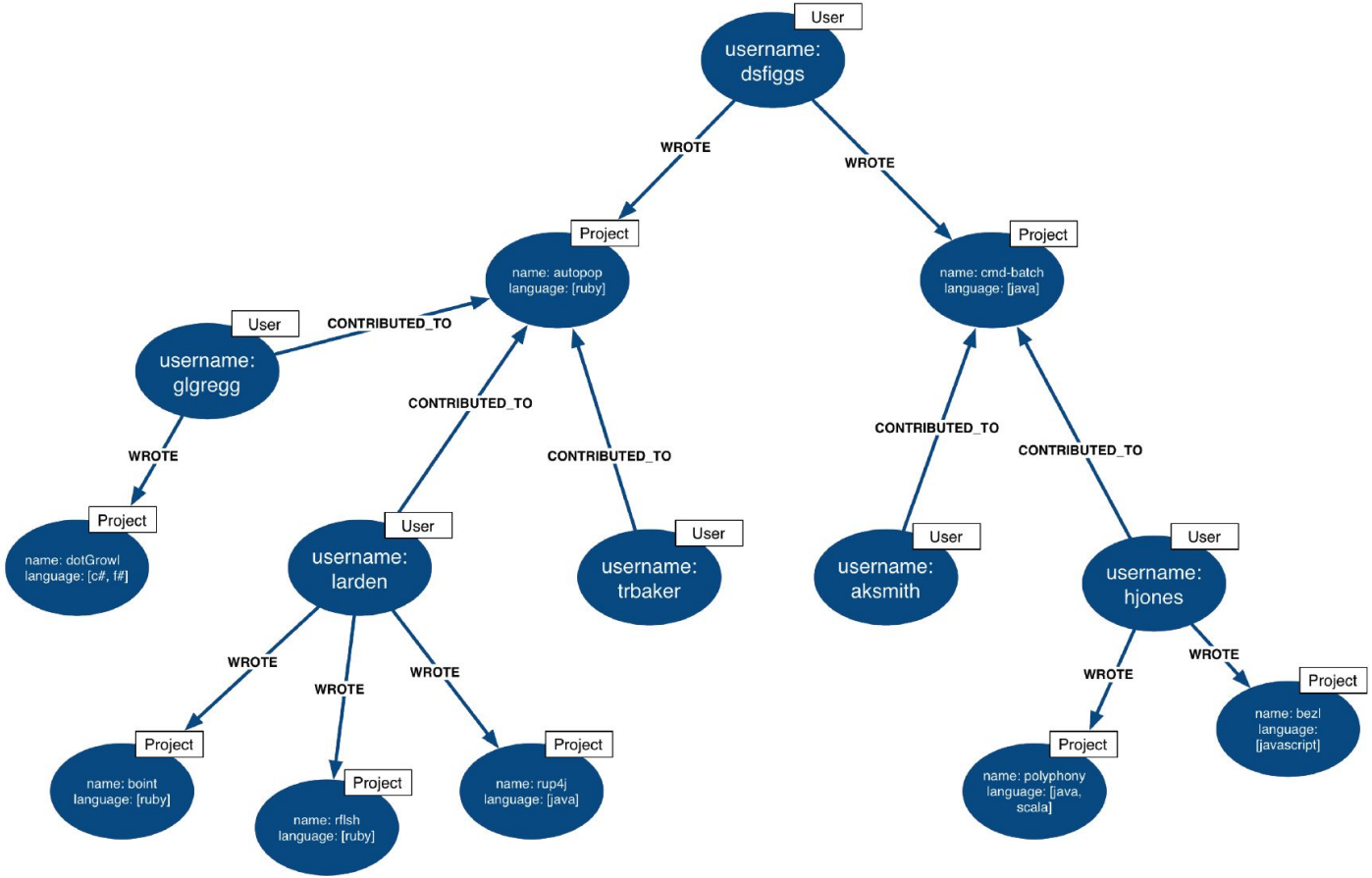


Too little granularity

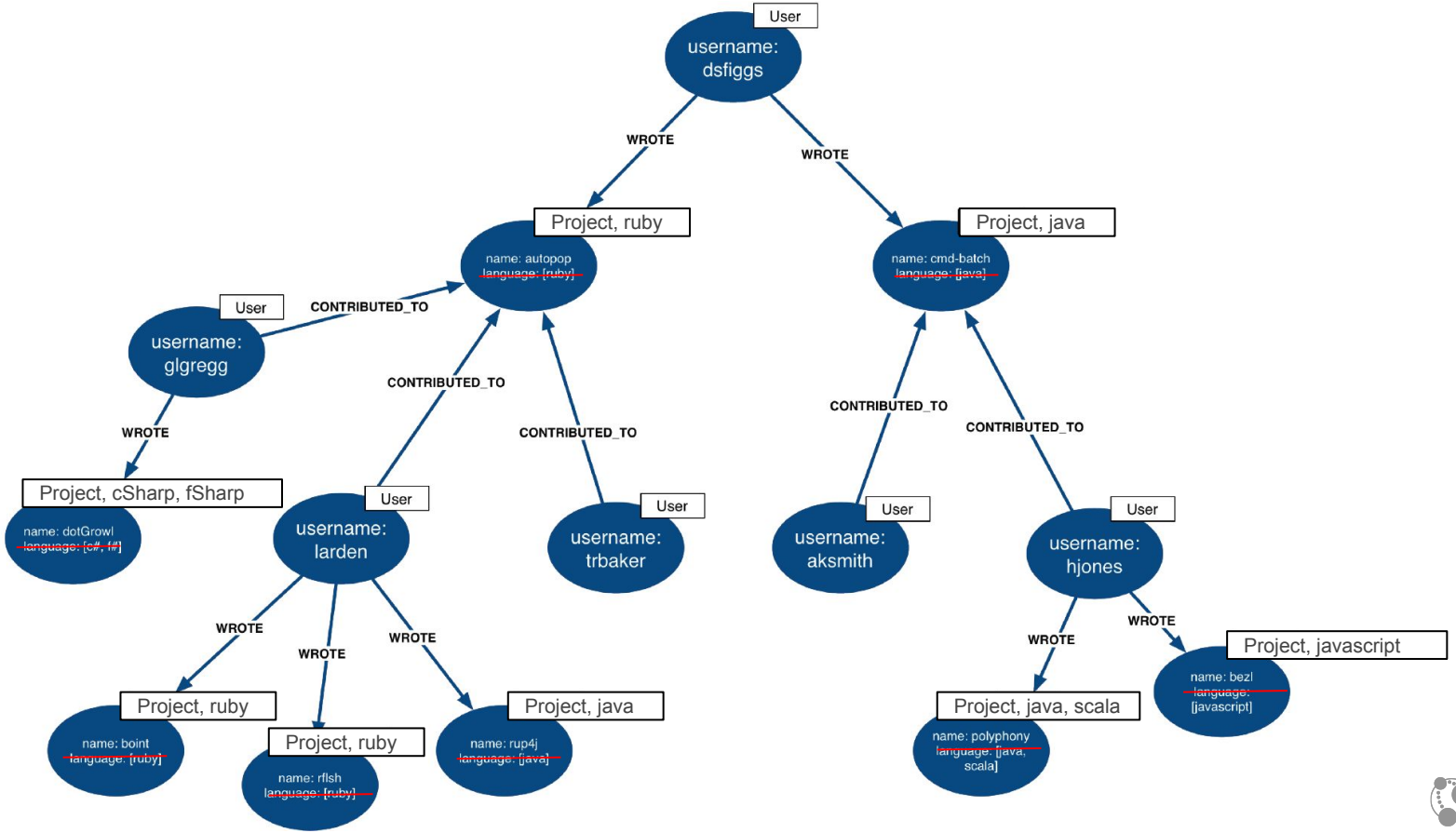
Too much granularity



Best practice: Use simple properties

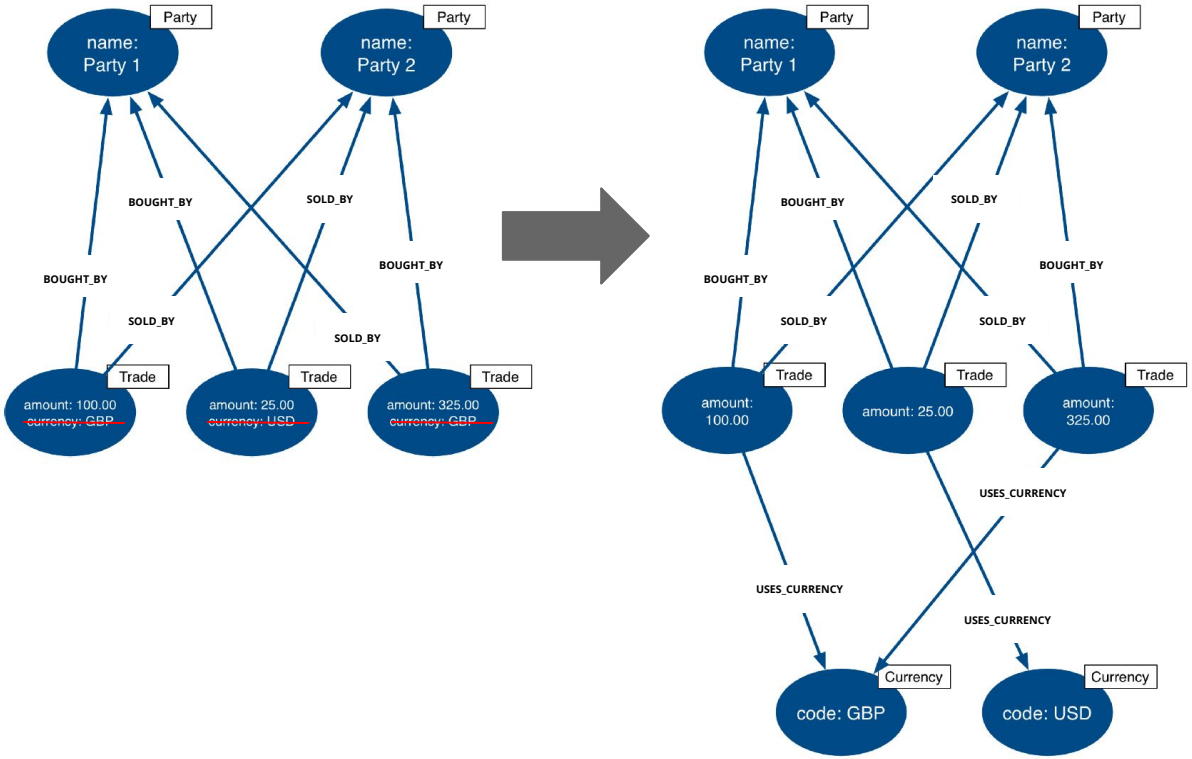


Best practice: Avoid duplication of data

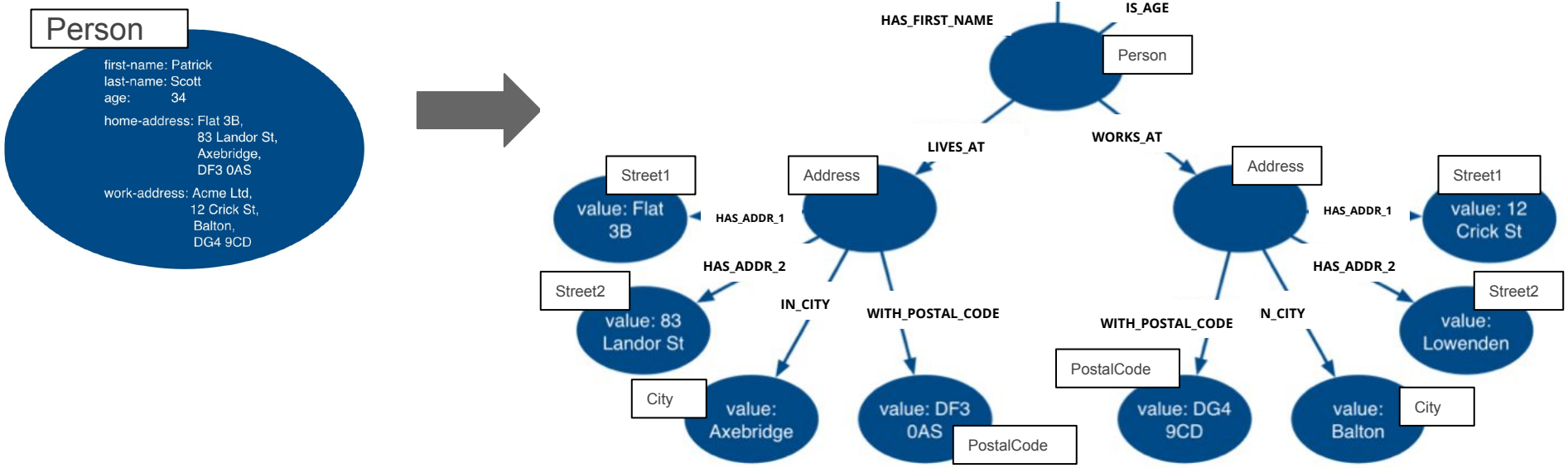


neo4j

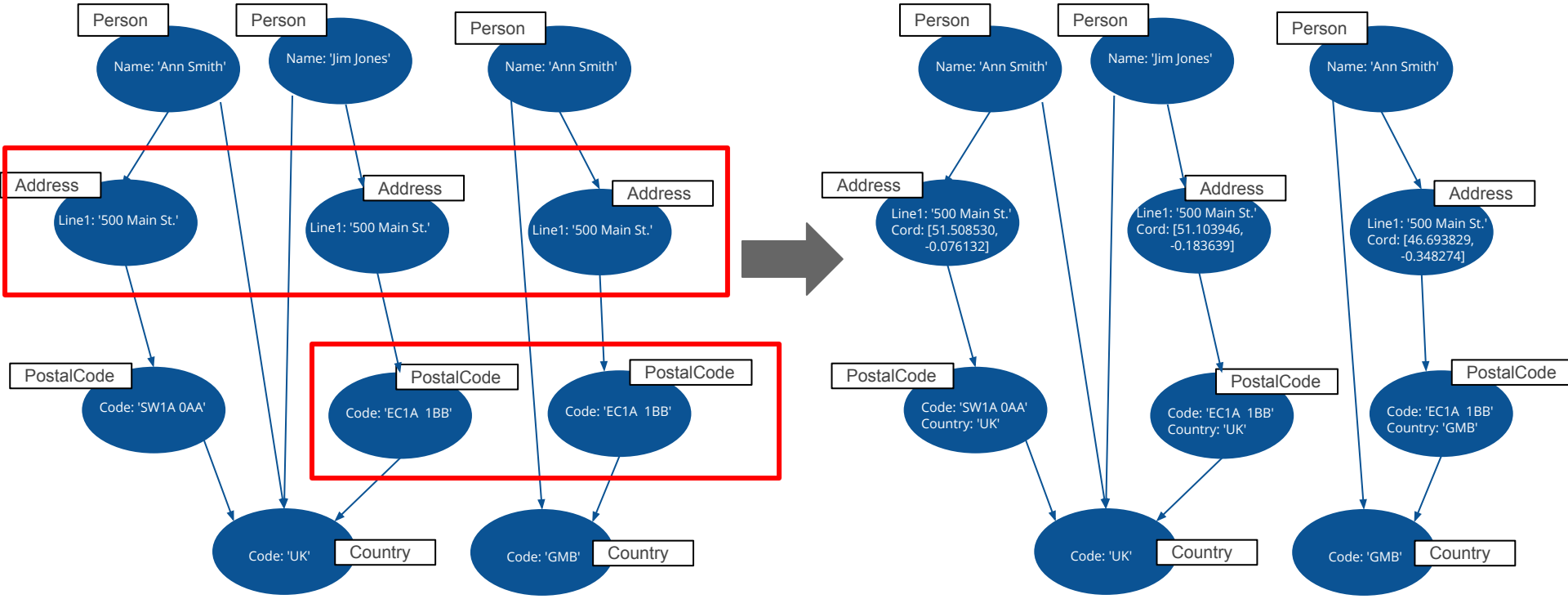
Best practice: Avoid duplication of data



Best practice: Avoid complex types for properties

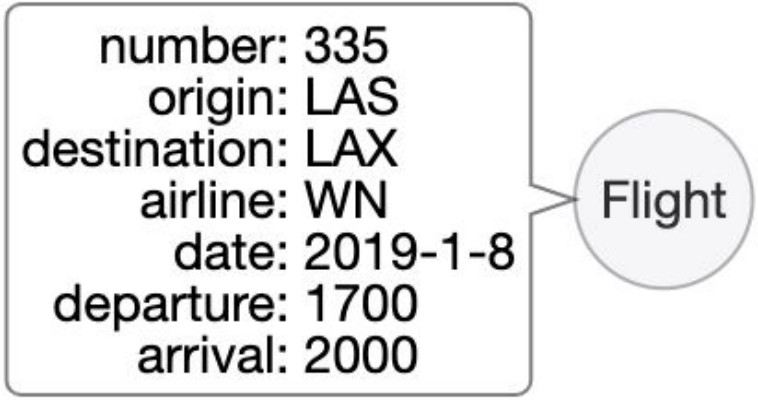


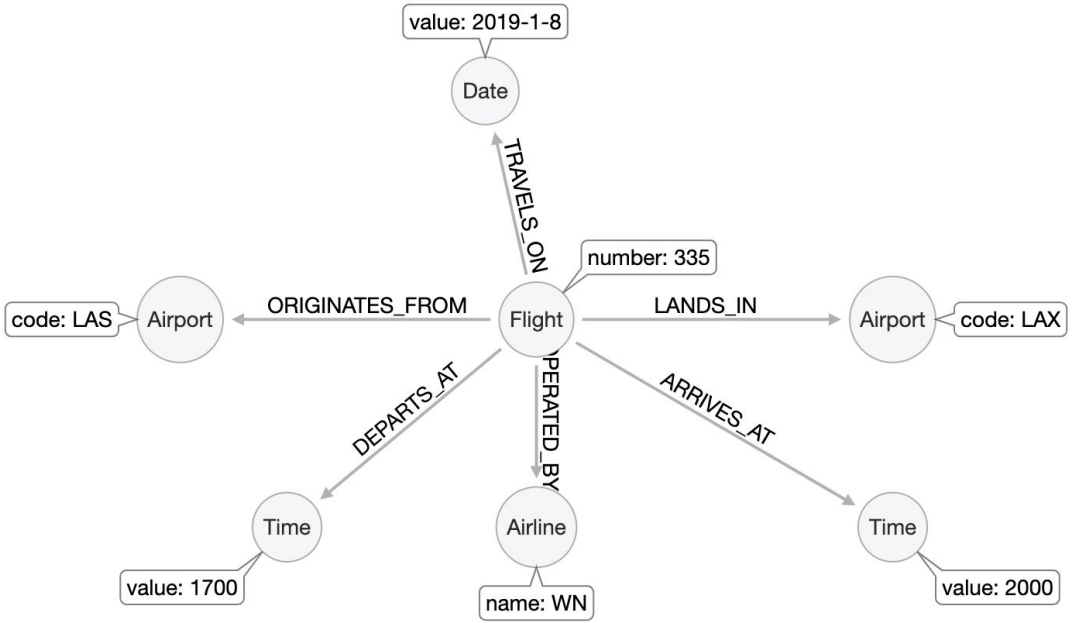
Best practice: Ensure uniqueness of some nodes



63

Best practice: Don't overuse nodes





64

Exercise 5: Using best practices

Steps:

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: Instructions

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

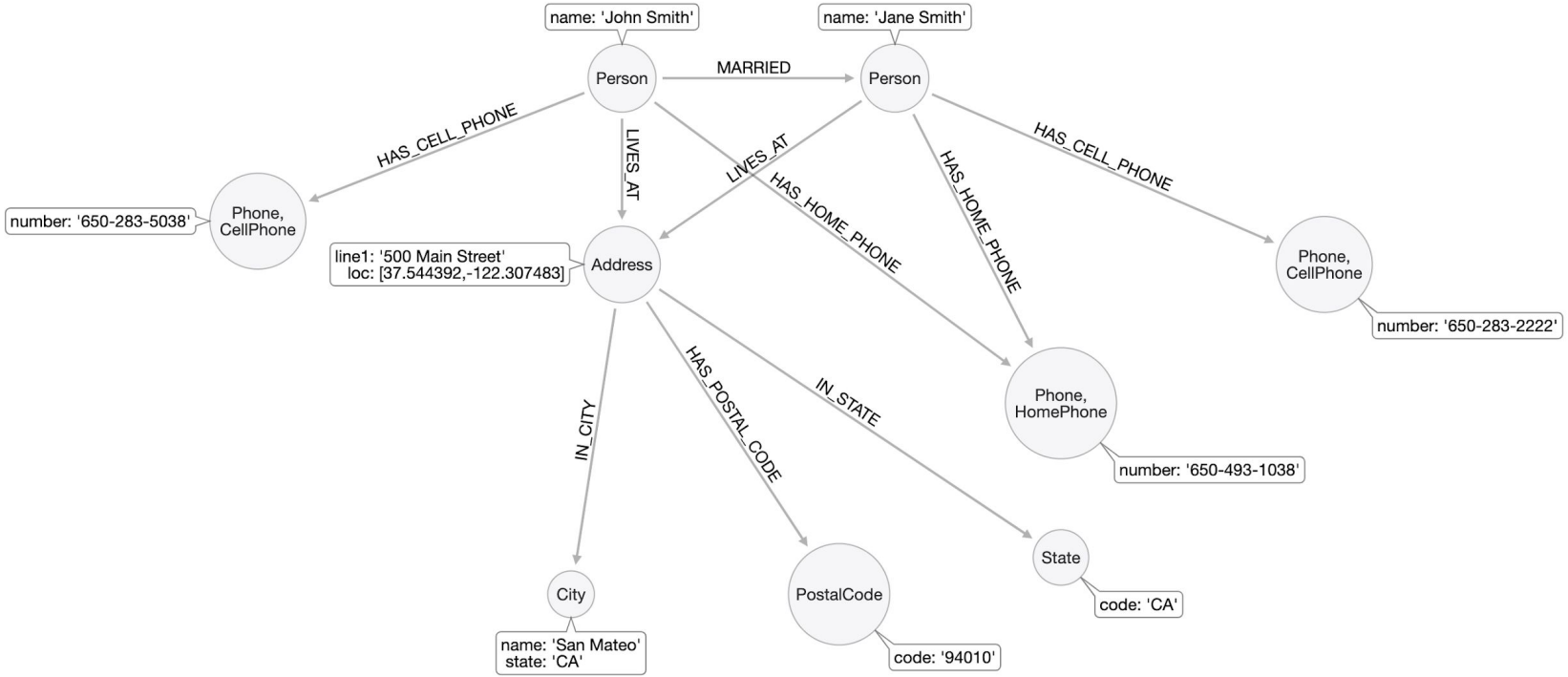
2. Open the model *JohnAndJane*.
   Hint: Use Export Markup to copy paste from **JohnAndJane.htm**.

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

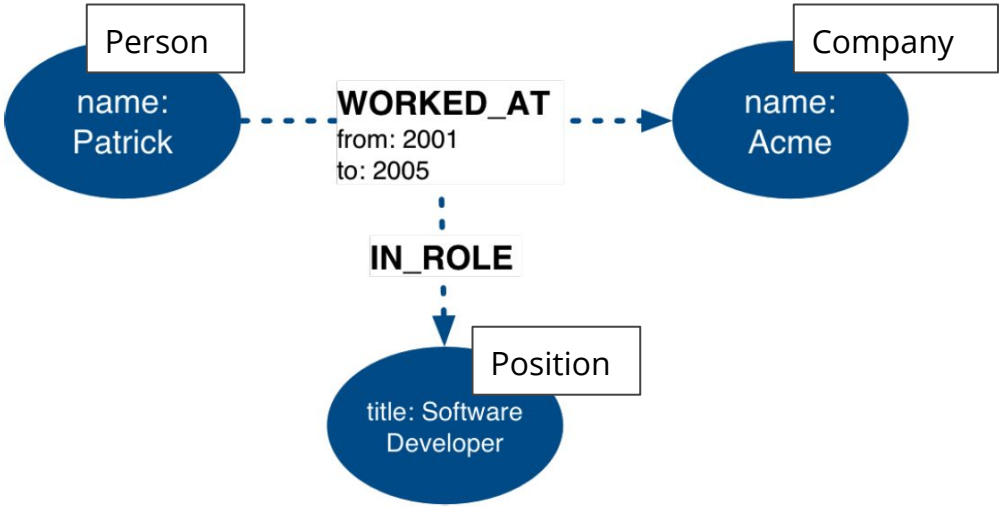


Common graph structures

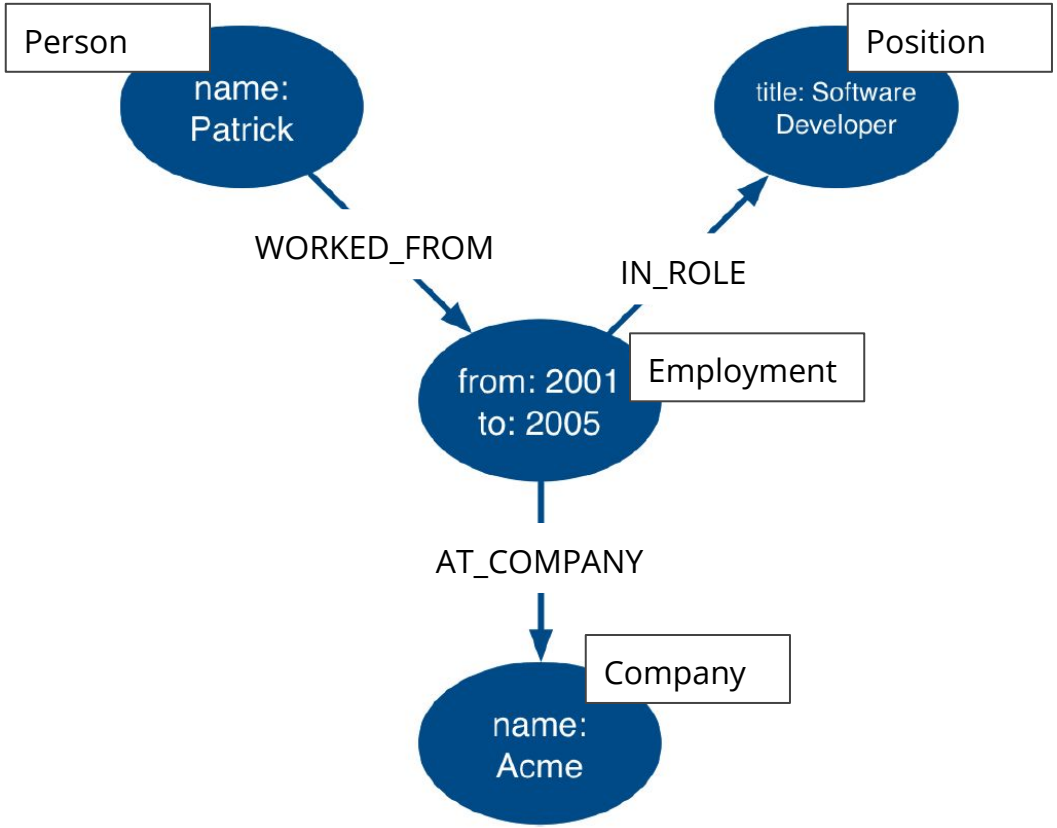
- Intermediate nodes
- Linked lists
- Specialized tree structures
- Multiple structures in a single model
- Versioning

Adding intermediate nodes

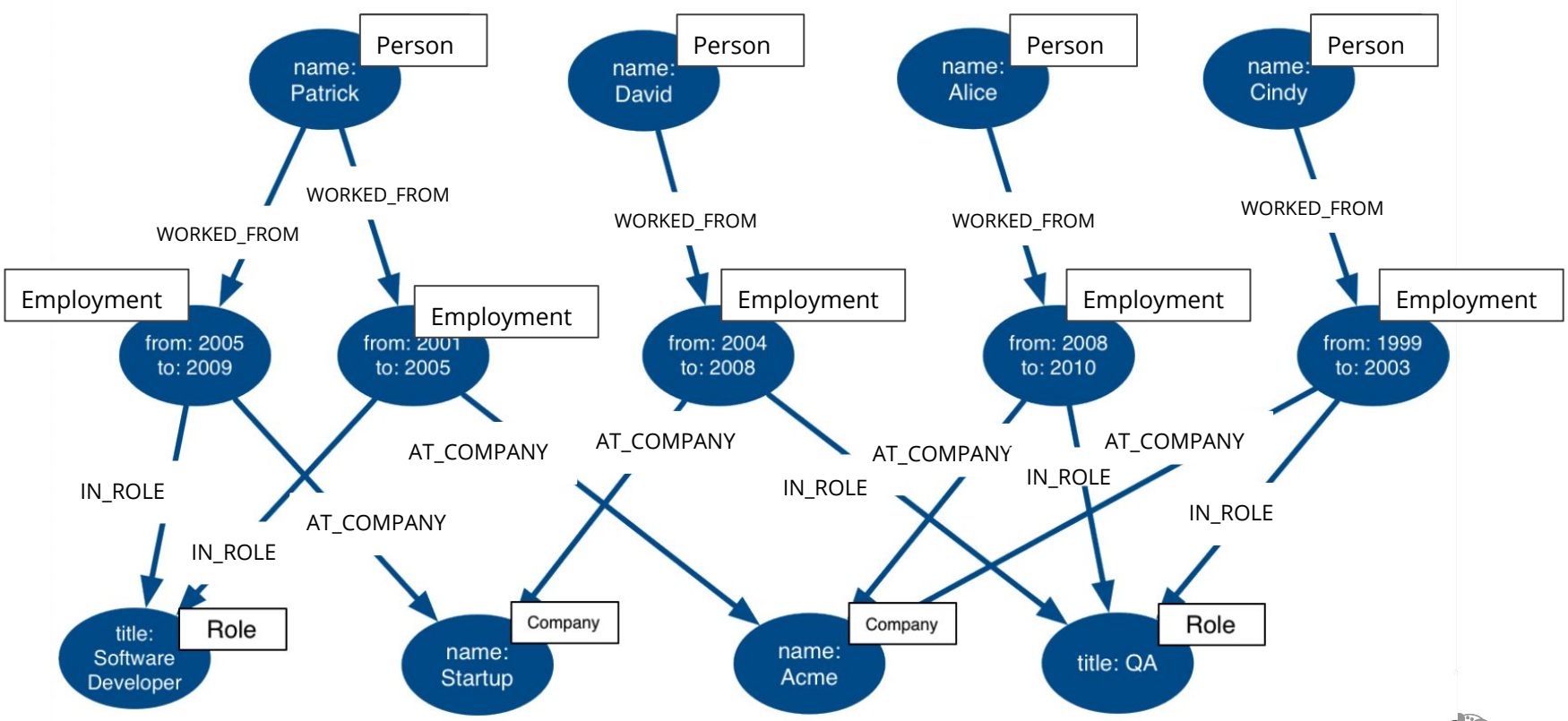
- Connect more than two nodes in a single context.
  - Hyperedges (n-ary relationships)
- When you need to relate something to a relationship.



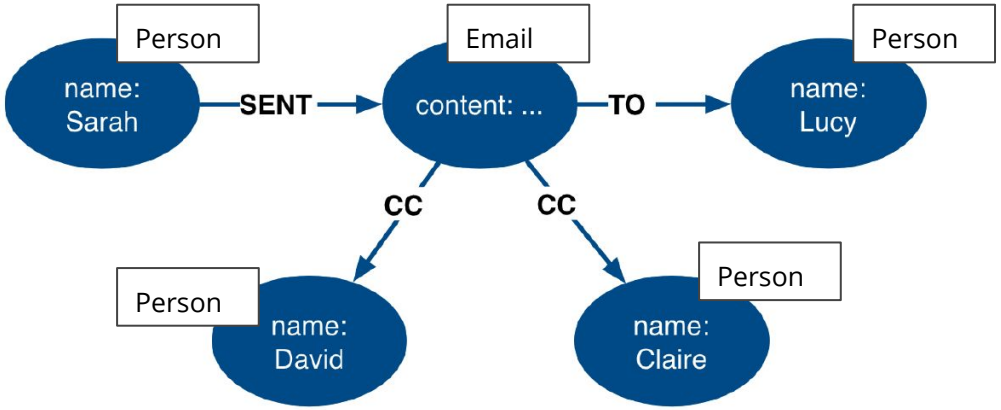
Rich context, multiple dimensions



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

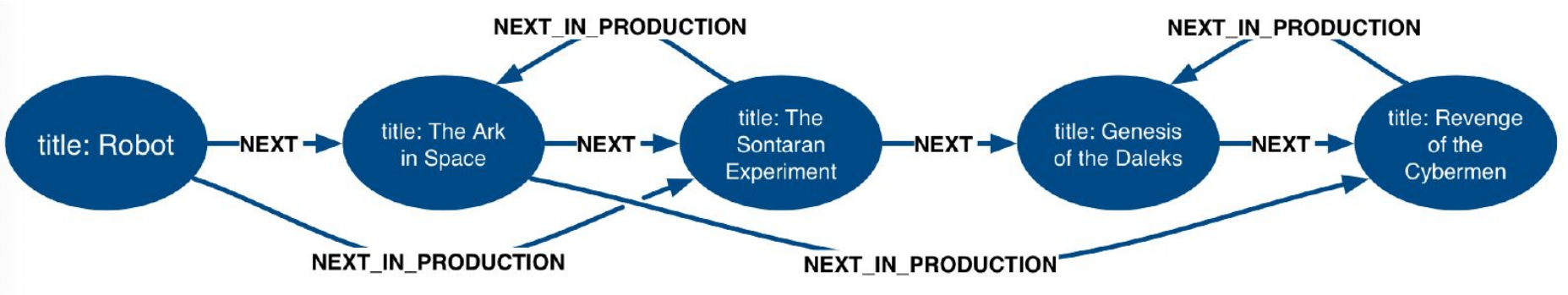
- Entities are linked in a sequence.
- Useful when you need to traverse the sequence.
  - next
  - previous
- You may need to identify:
  - first
  - last
- Examples:
  - Event stream
  - Episodes of a TV series
  - Job history

Simple linked lists

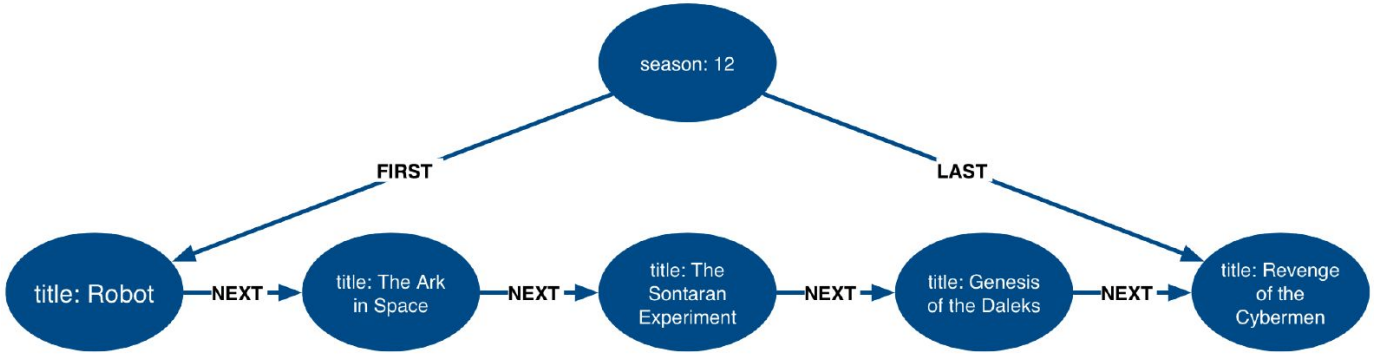
Episodes of the Dr.Who series:

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.

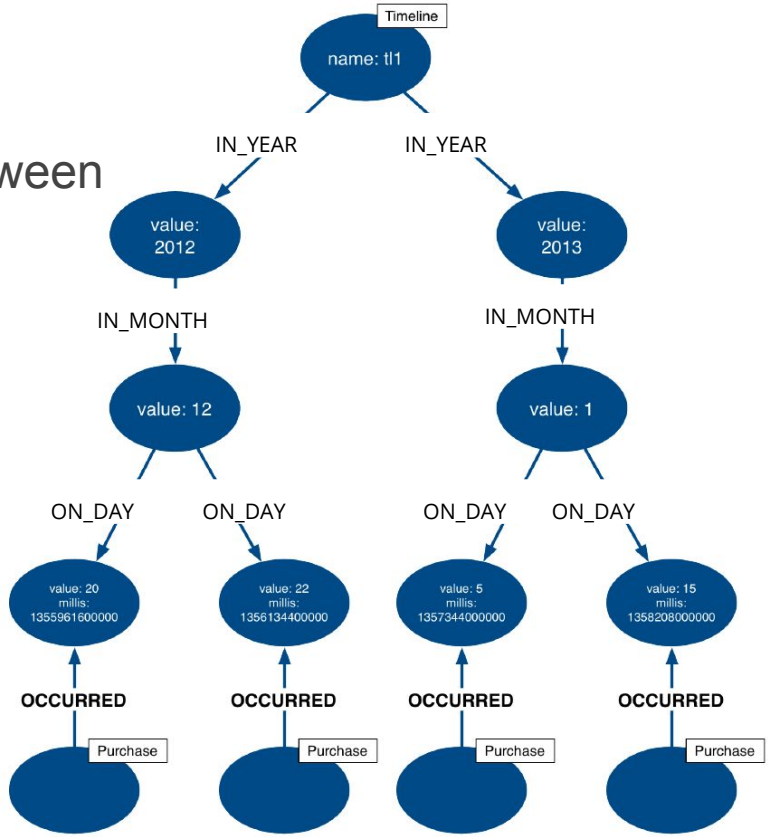
Specialized tree structures

An index is a graph:

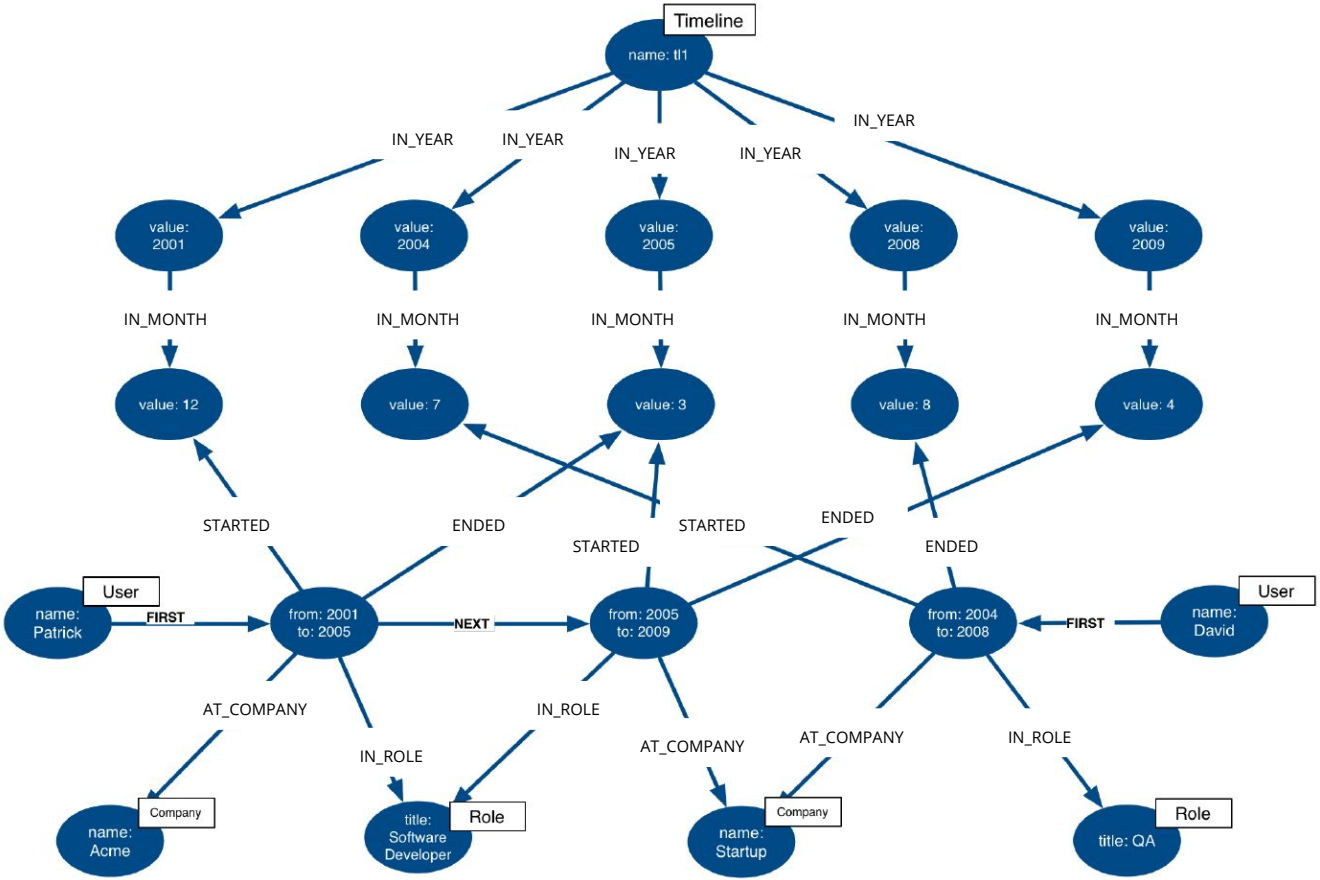
- b-tree for binary search
- r-tree
  - Spatial search
  - Multi-dimensional data
- Timeline tree
  - Discrete events
    - No natural relationship to other events
  - Search for events at differing levels of granularity between:
    - 2 days
    - 2 months
    - 2 minutes

Example: Timeline tree

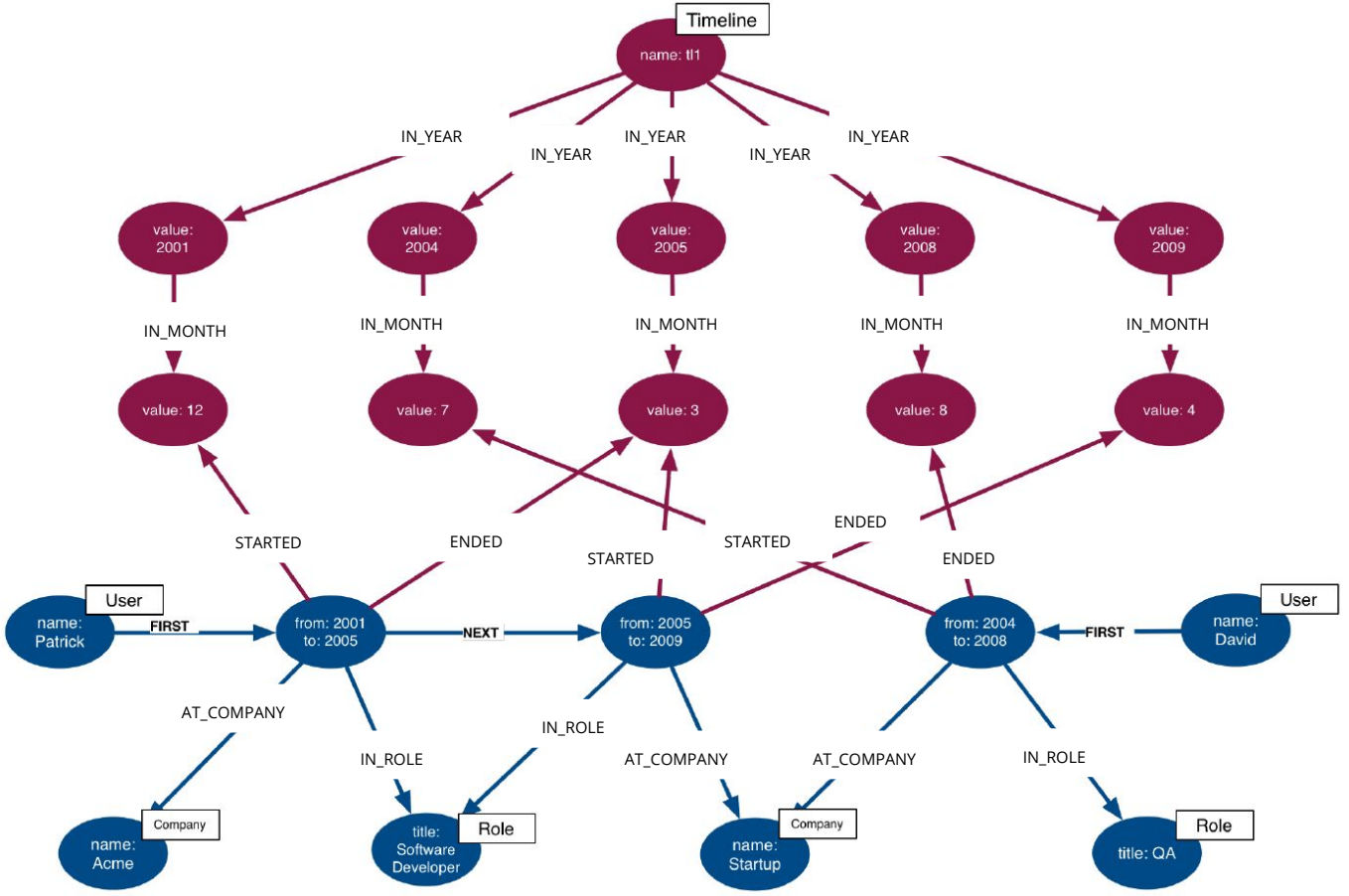
Retrieve all purchases between two dates.



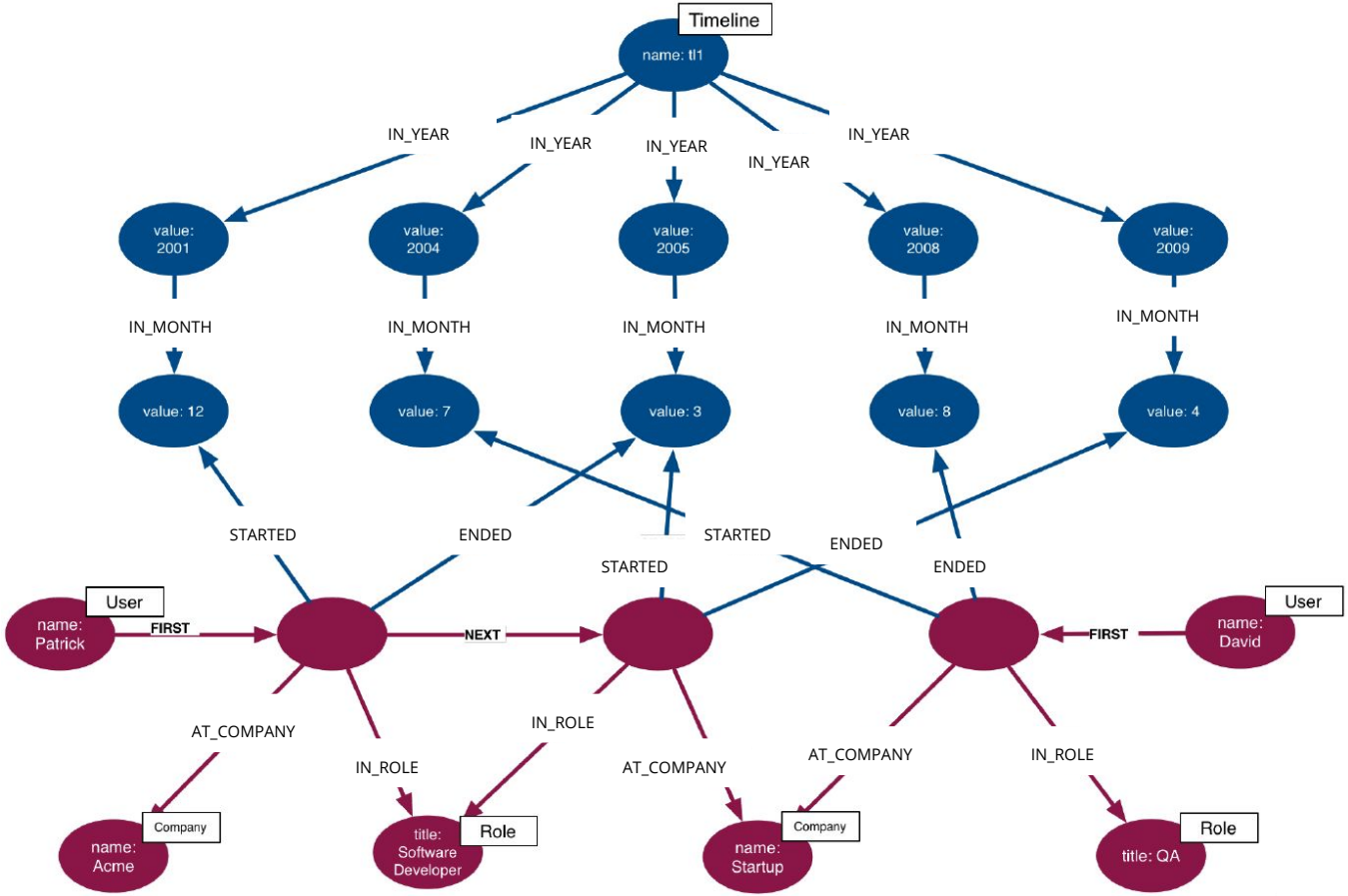
Multiple structures in a single model



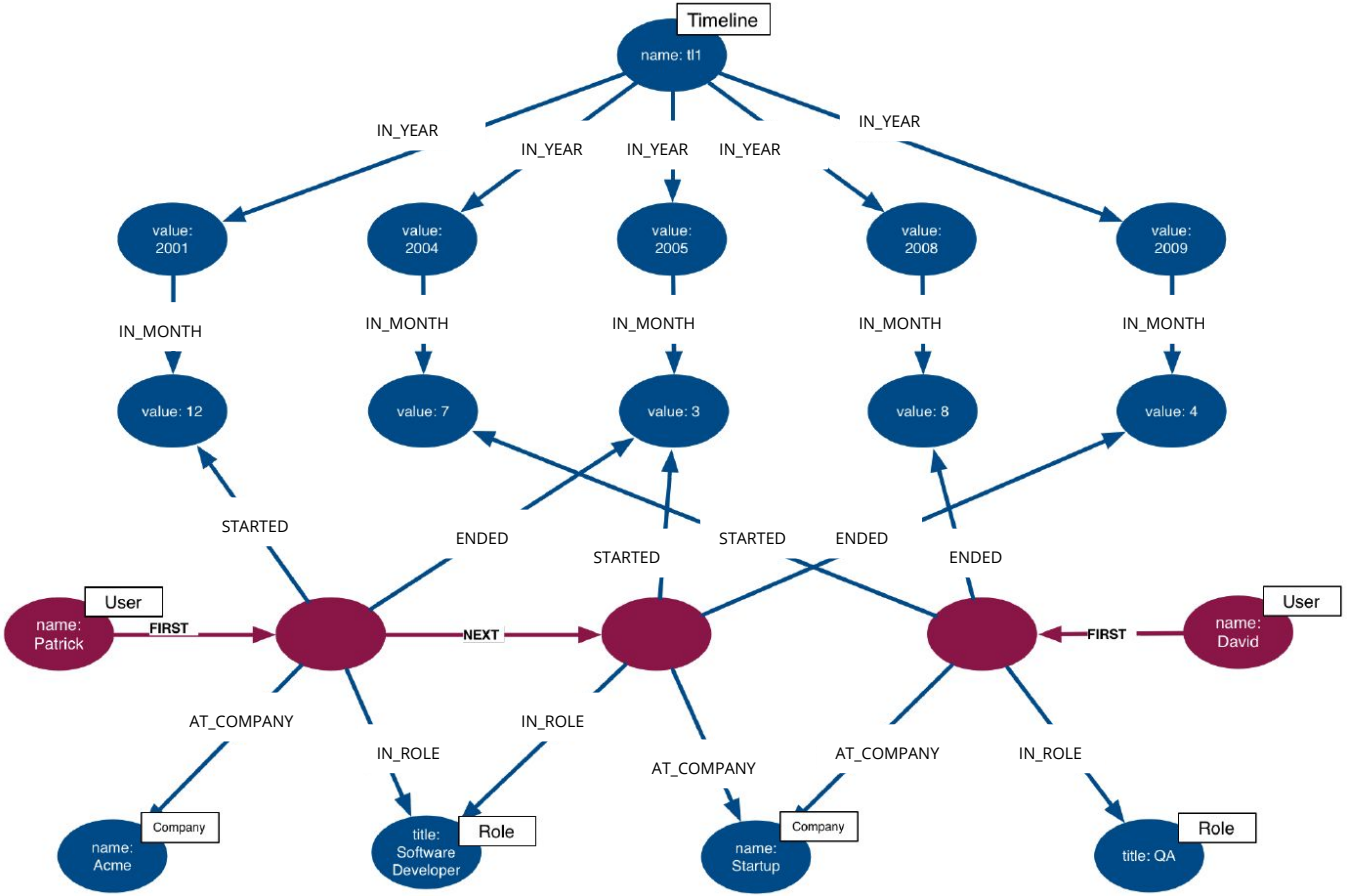
Timeline structure in the model



Intermediate nodes in the model



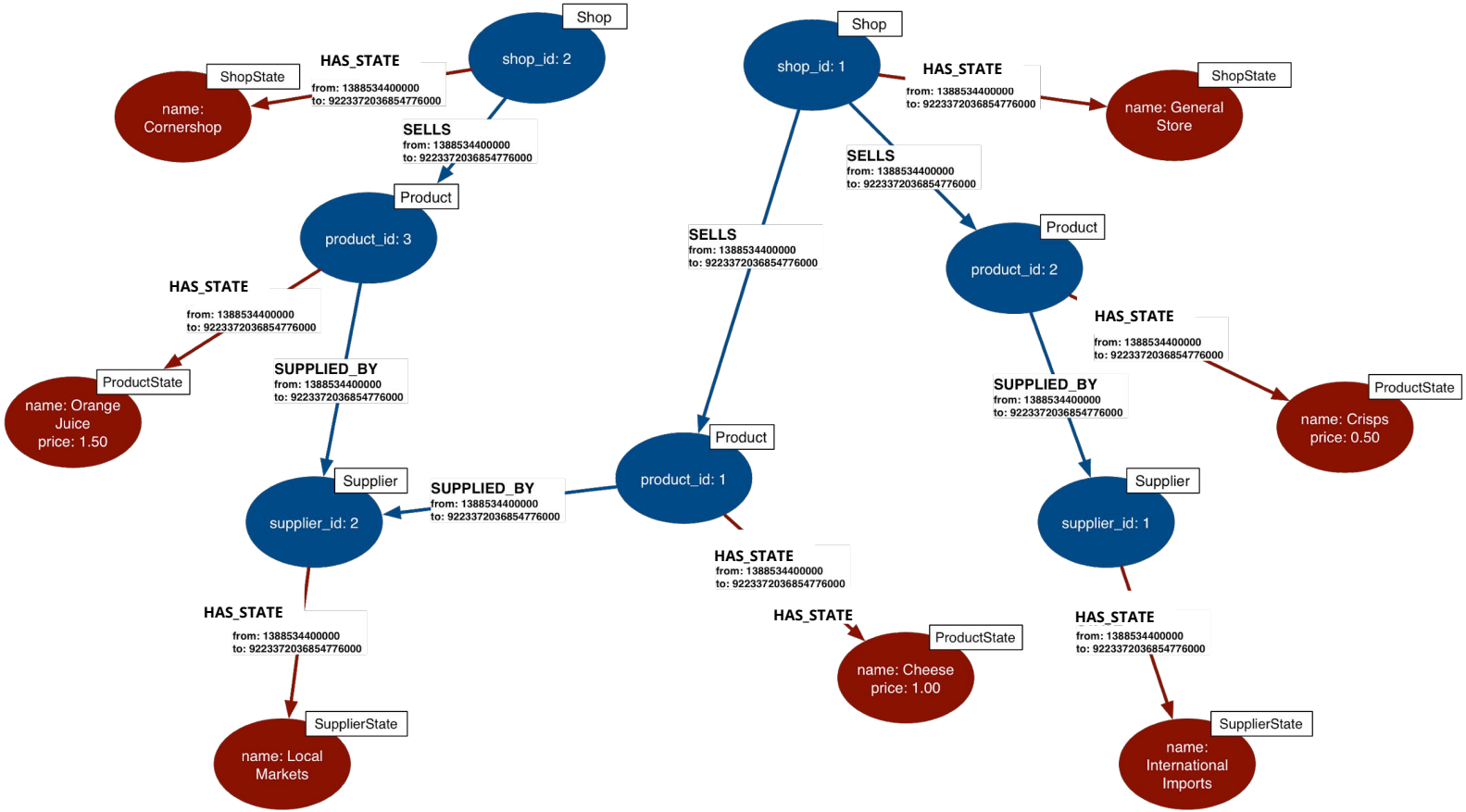
Linked lists in the model



Versioning

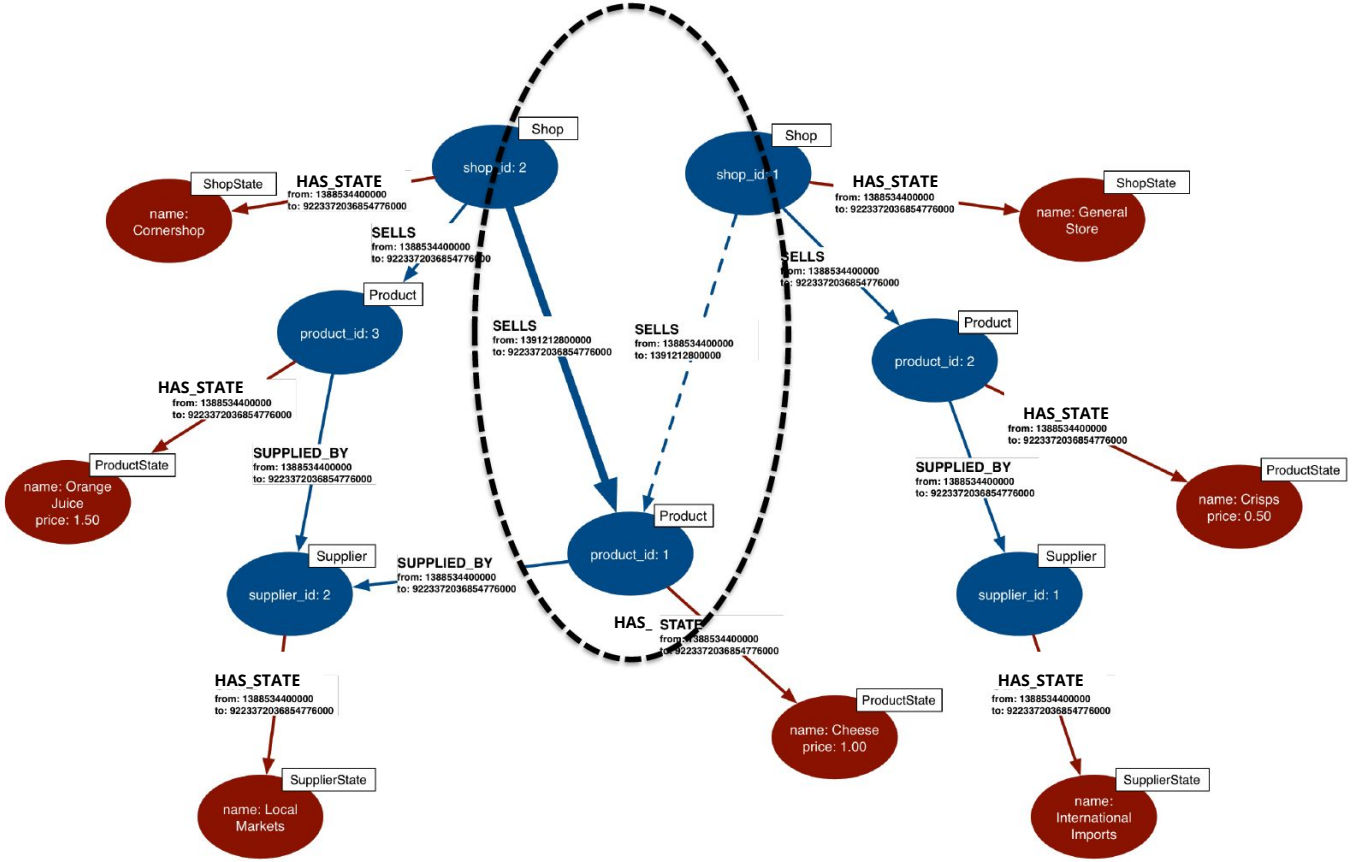
- A versioning model is time-based
  - Universal versioning schema
  - Discrete, continuous sequence
- Separate structure from state
  - Structure
    - Identity nodes as placeholders
    - Timestamped identity relationships
  - State
    - State nodes which are a snapshot of the entity state
    - Timestamped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

Versioning example (1)



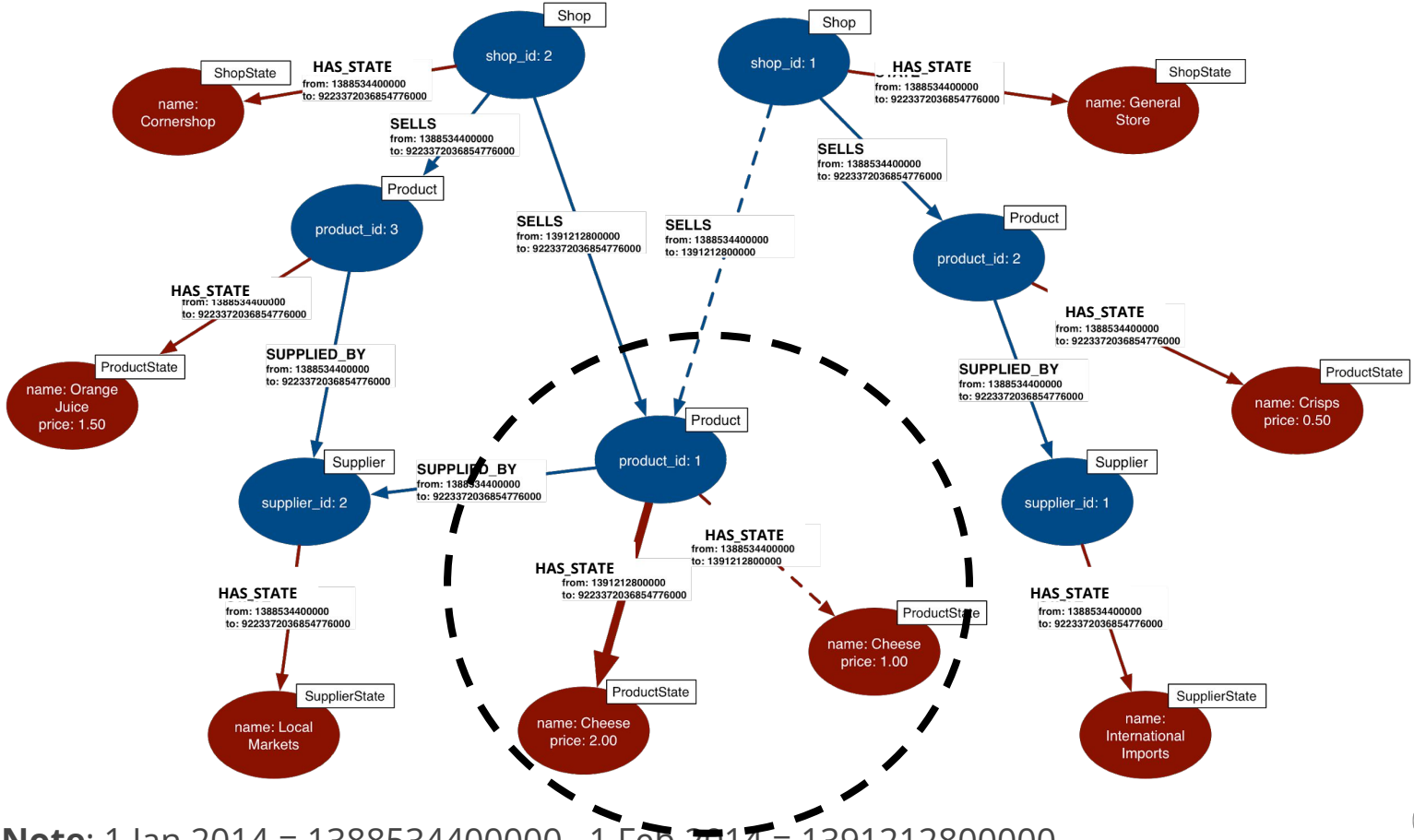
Note: 1 Jan 2014 = 1388534400000

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**.

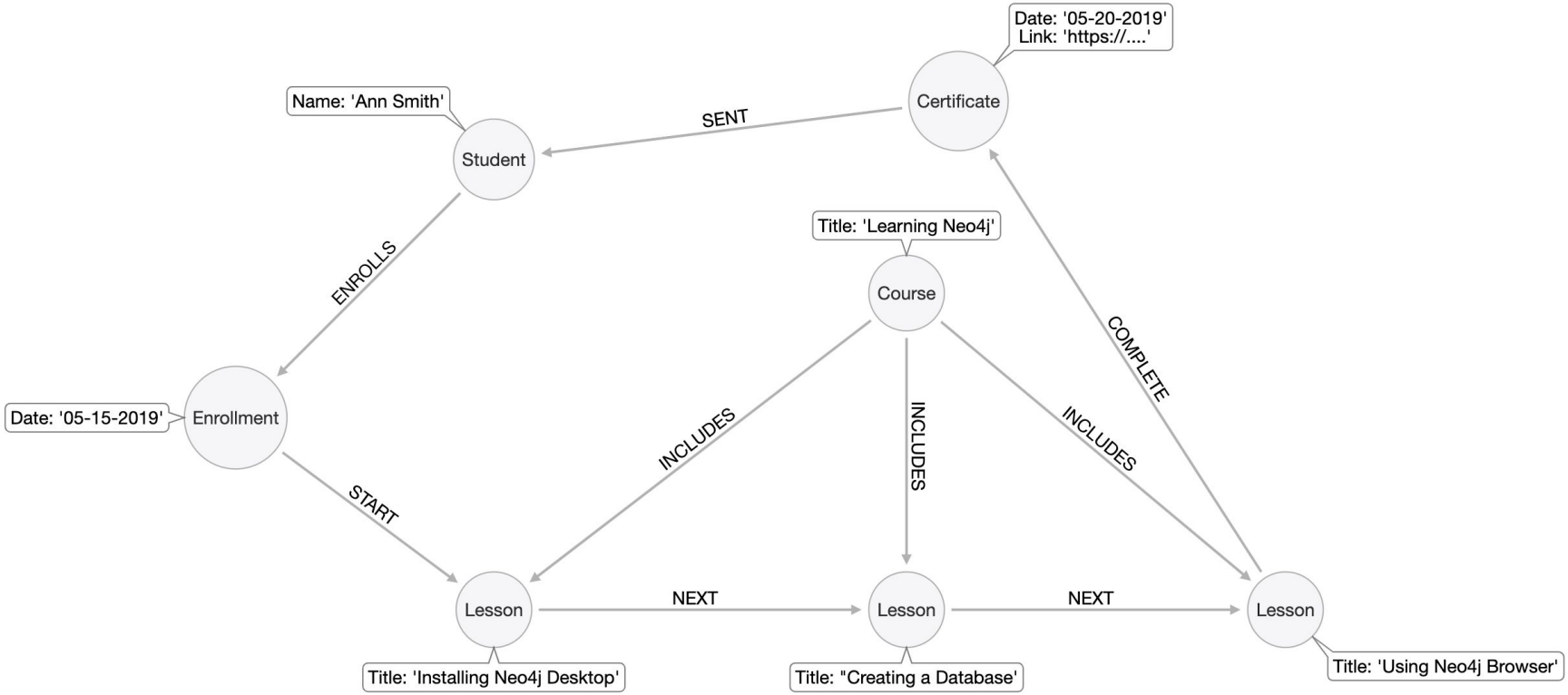
Exercise 6: Solution

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

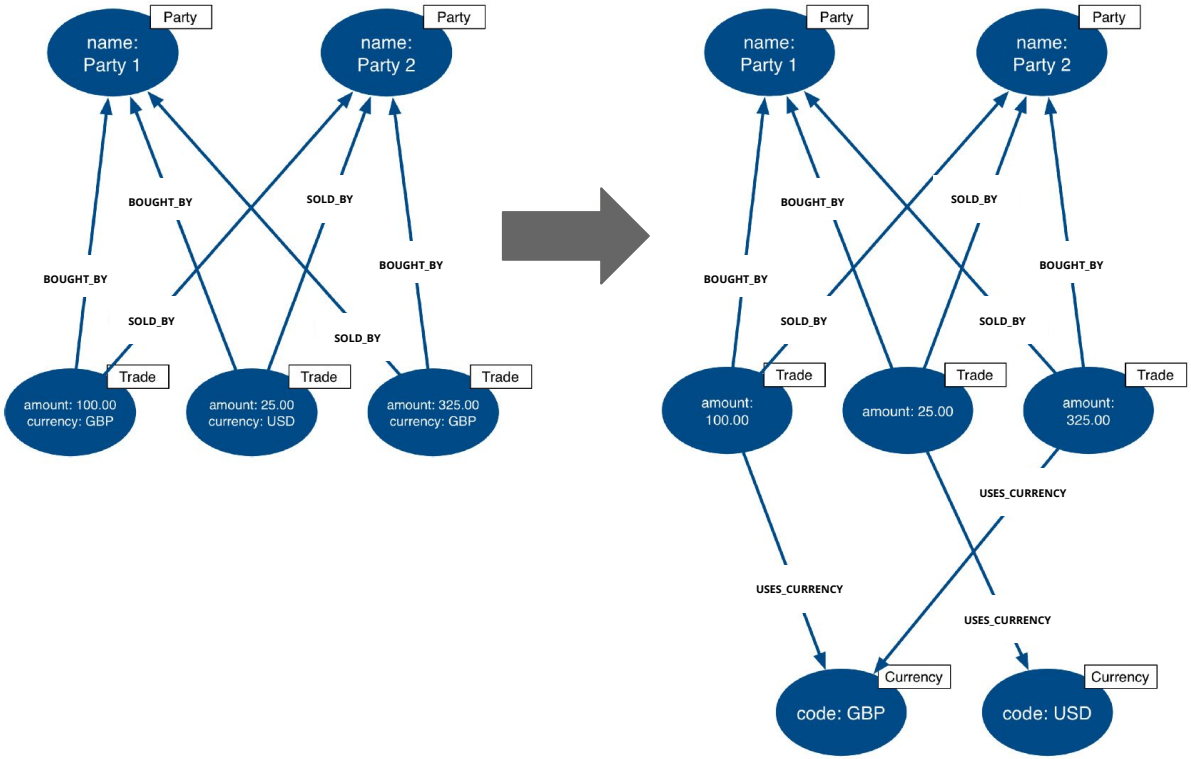


Evolving the graph data model

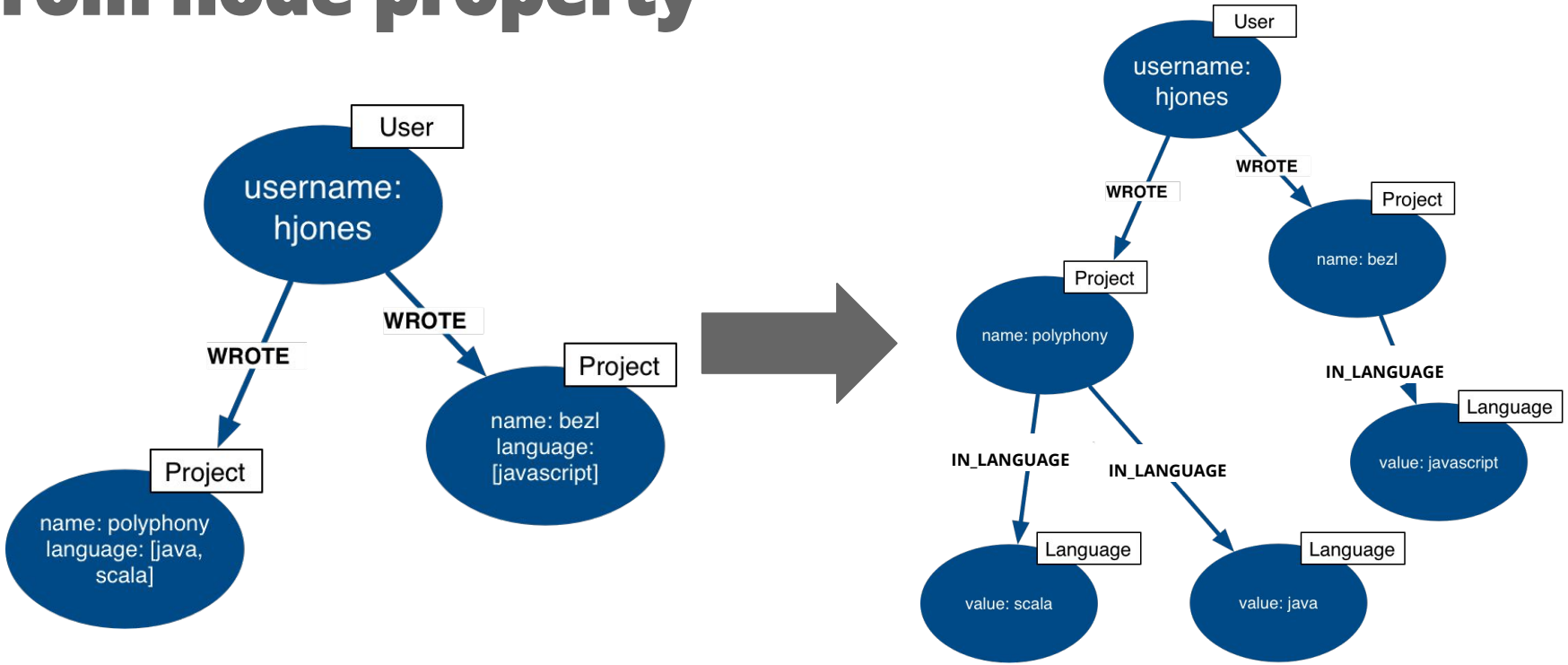
- Restructure graph without changing informational semantics
- Why refactor?
  - Improve design
  - Enhance performance
  - Accommodate new functionality
  - Enable iterative and incremental development of data model
- Examples:
  - Extract a node from a property (Address)
  - Extract node from an array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

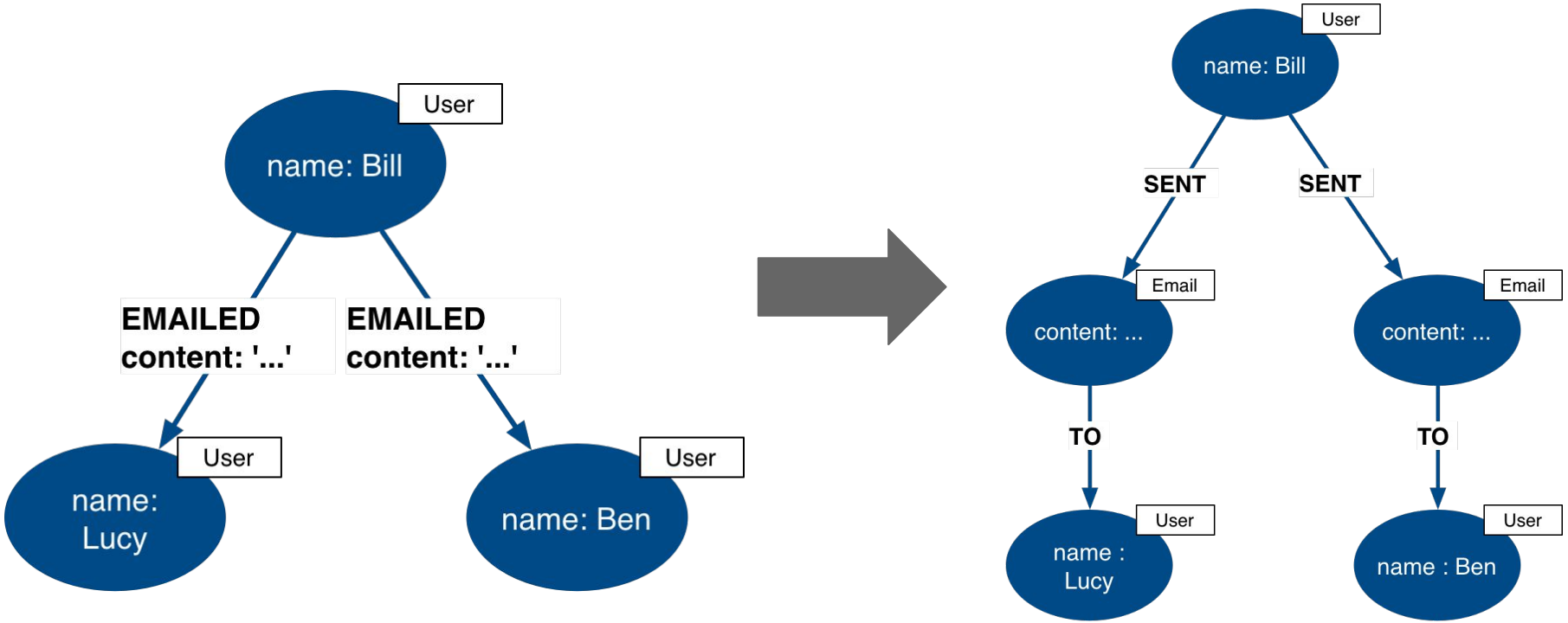


Example: Extract node and relationship from node property



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

Example: Extract node and relationship from relationship property



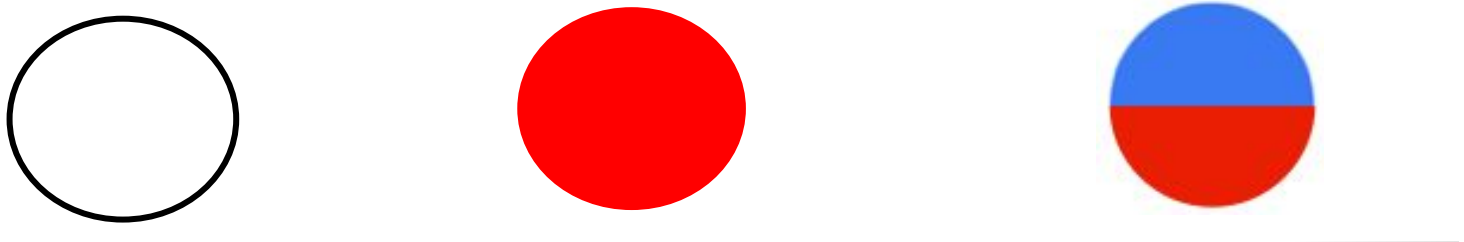
Top 4 modeling tips from the experts

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

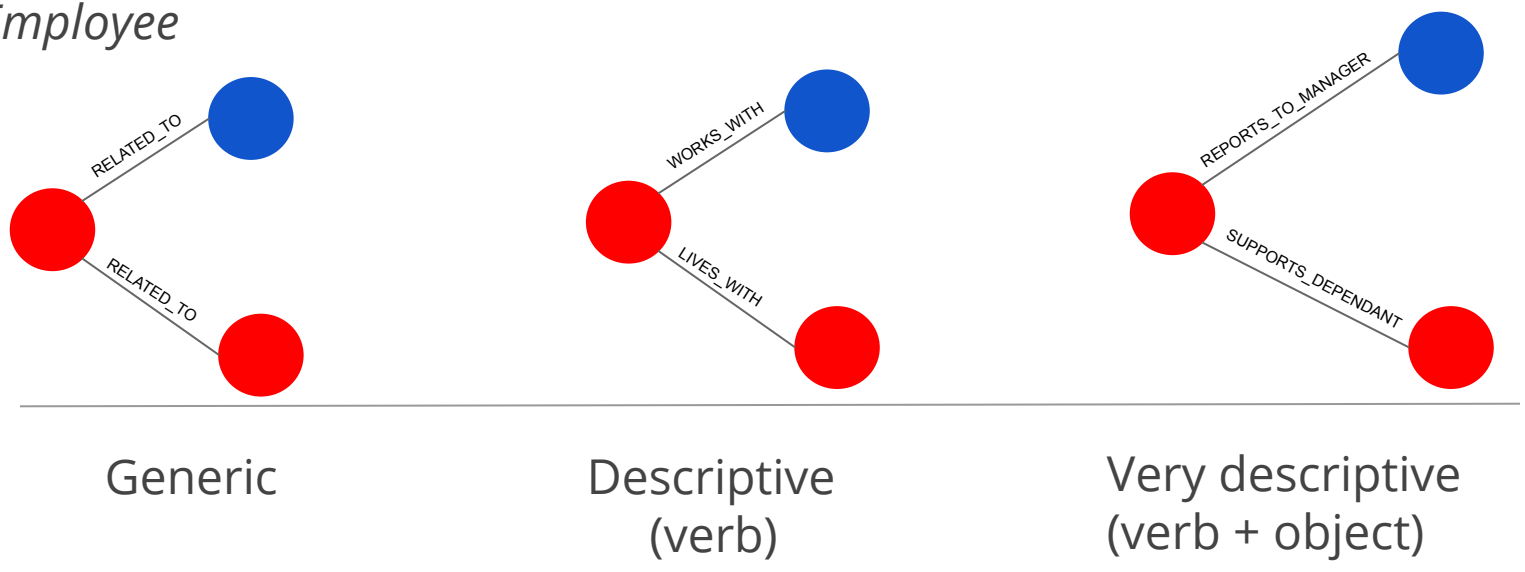
One label
Person

Multiple labels
Person
Employee

Be specific with relationship types

Person

Employee



Generic

Descriptive (verb)

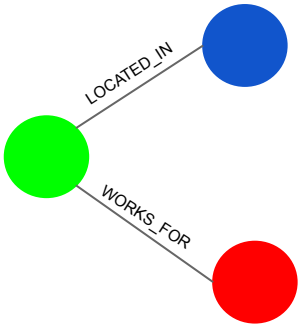
Very descriptive (verb + object)

Avoid unnecessary relationships

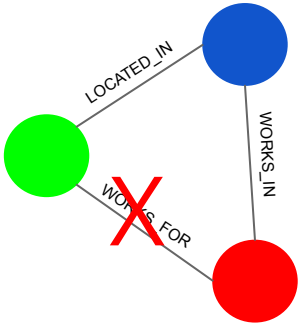
Person

Company

Location



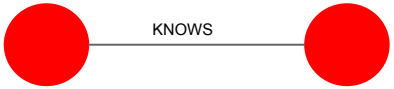
Good model



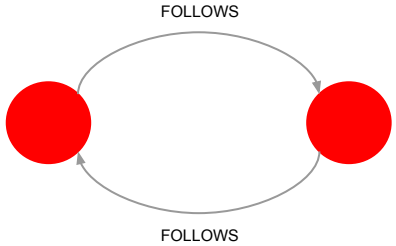
Redundant relationship

Be mindful of symmetric relationships

Person



Facebook



Twitter

KNOWS relationship is symmetrical.

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

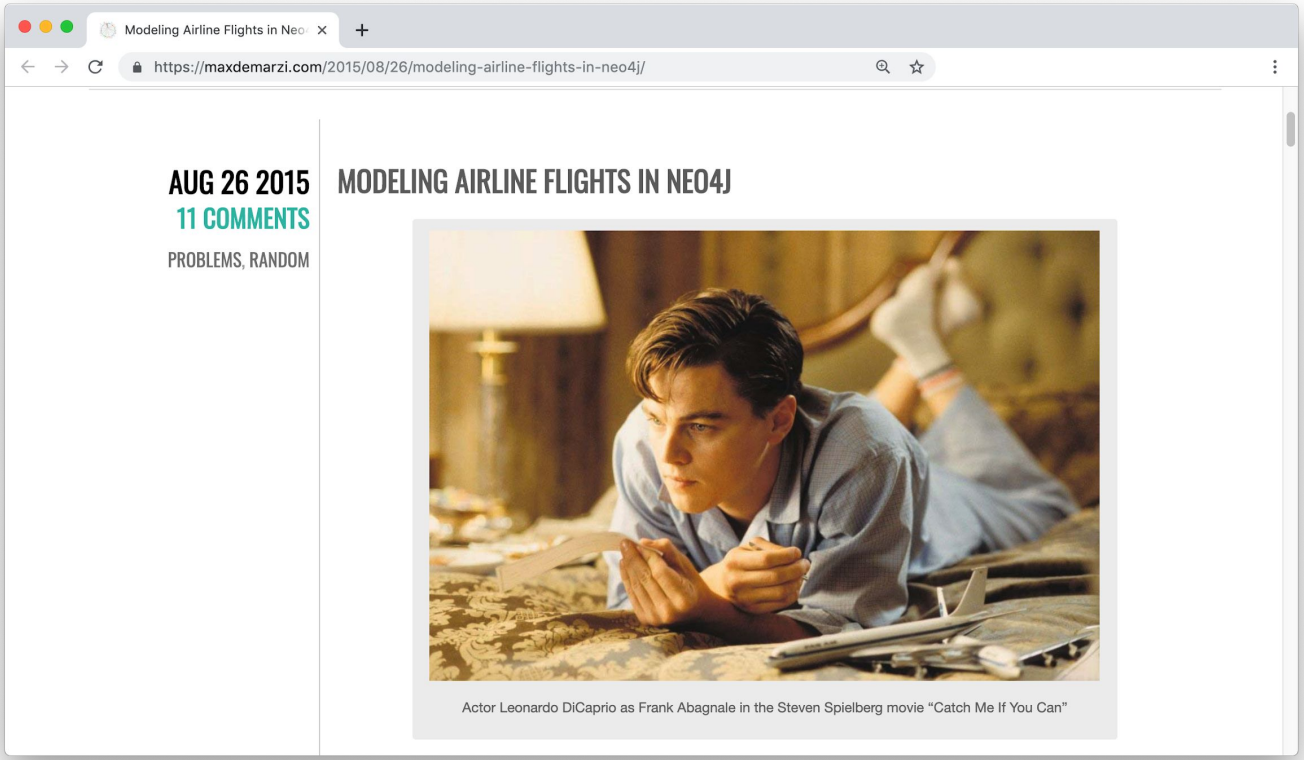
FOLLOWS relationship is asymmetrical.

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

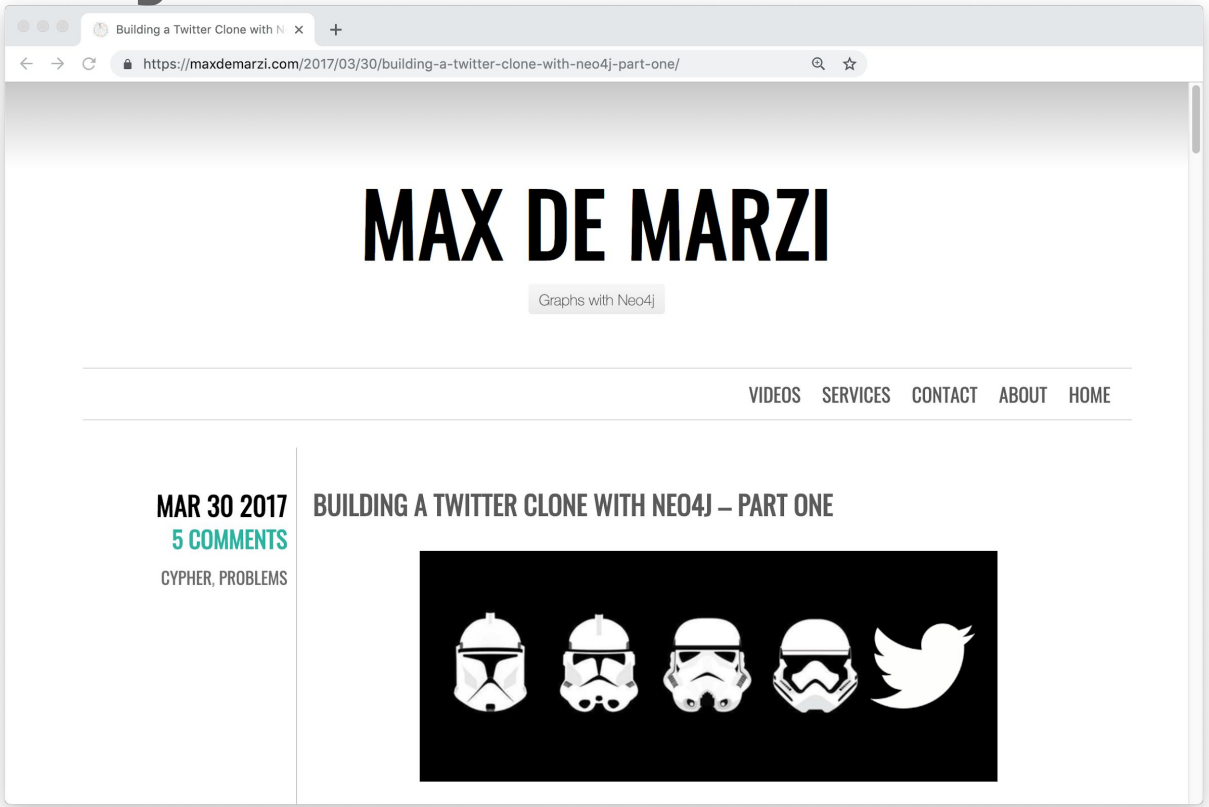
Some Neo4j use cases

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

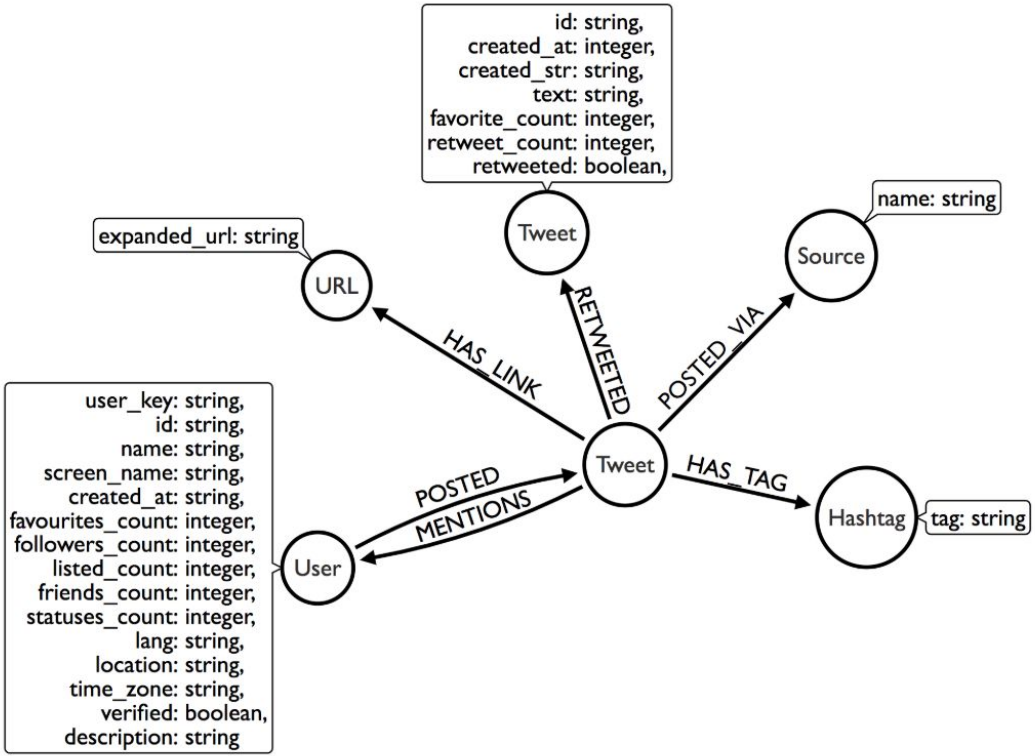
Tracking airline flights: Modeling airline flights in Neo4j



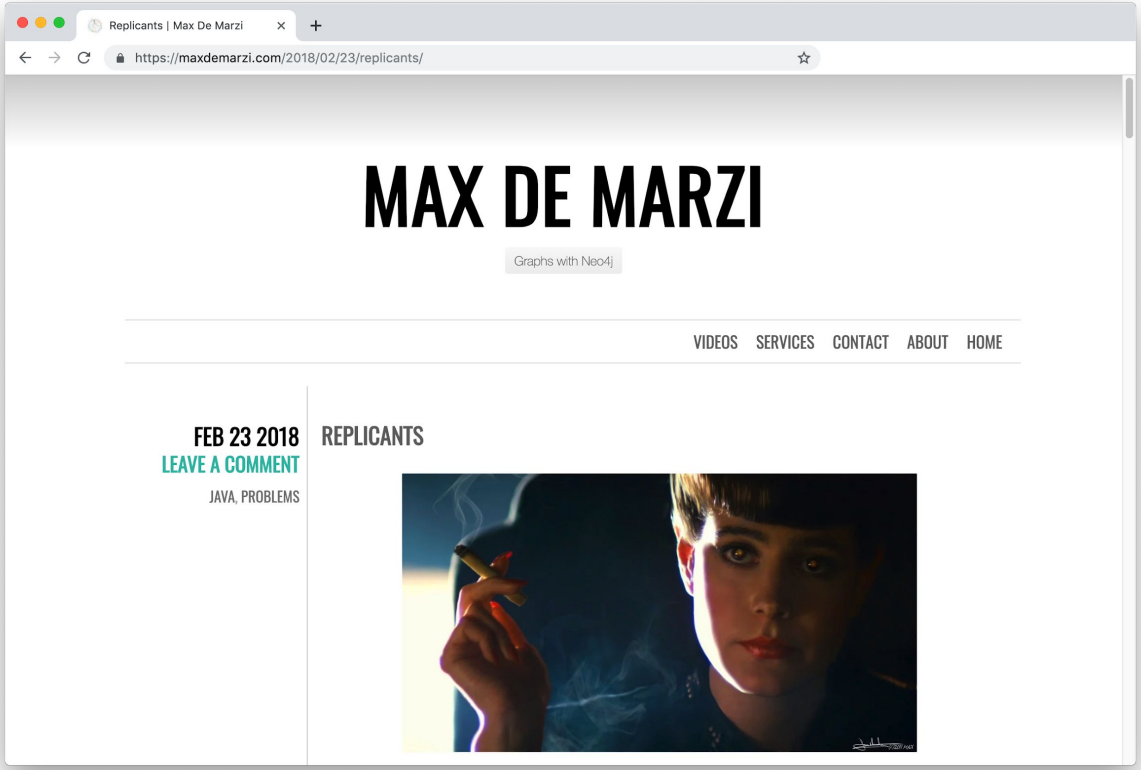
Activity feed: Building a Twitter Clone with Neo4j



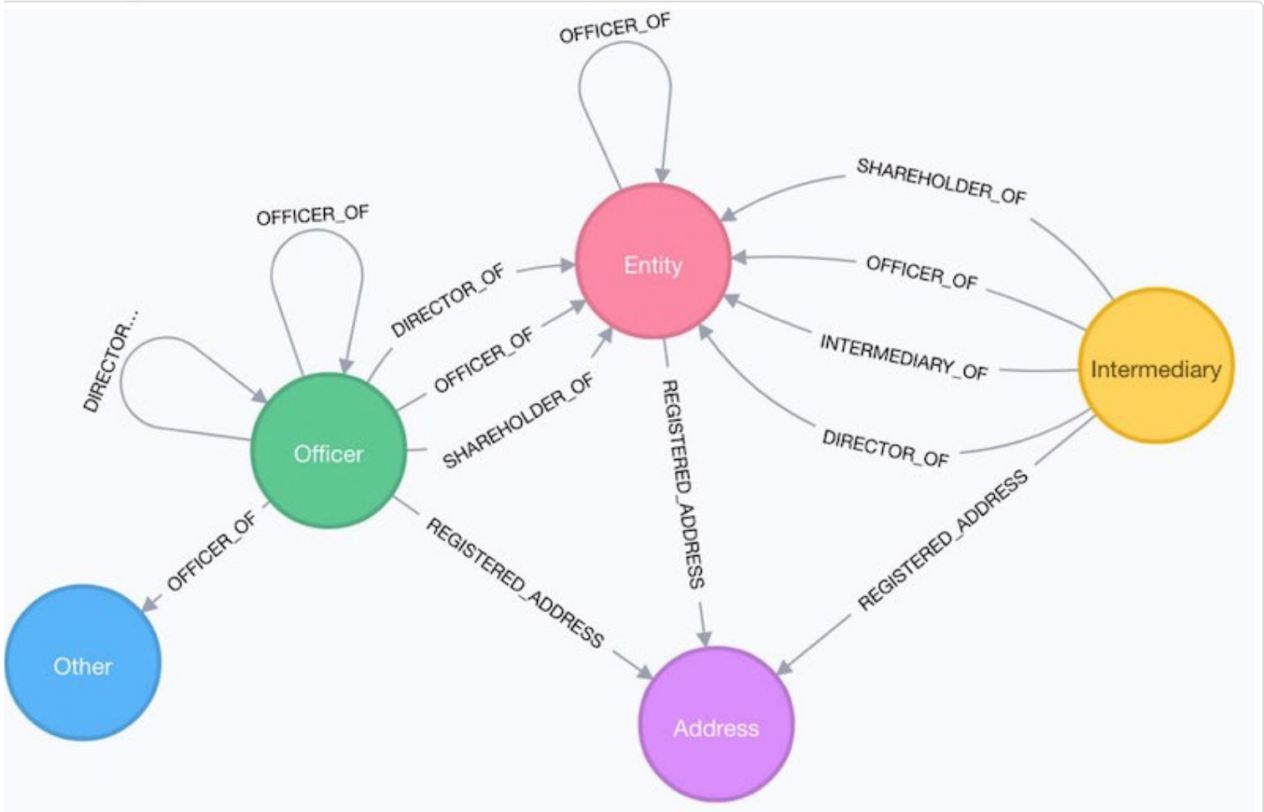
Activity feed: Russian Twitter Trolls Sandbox



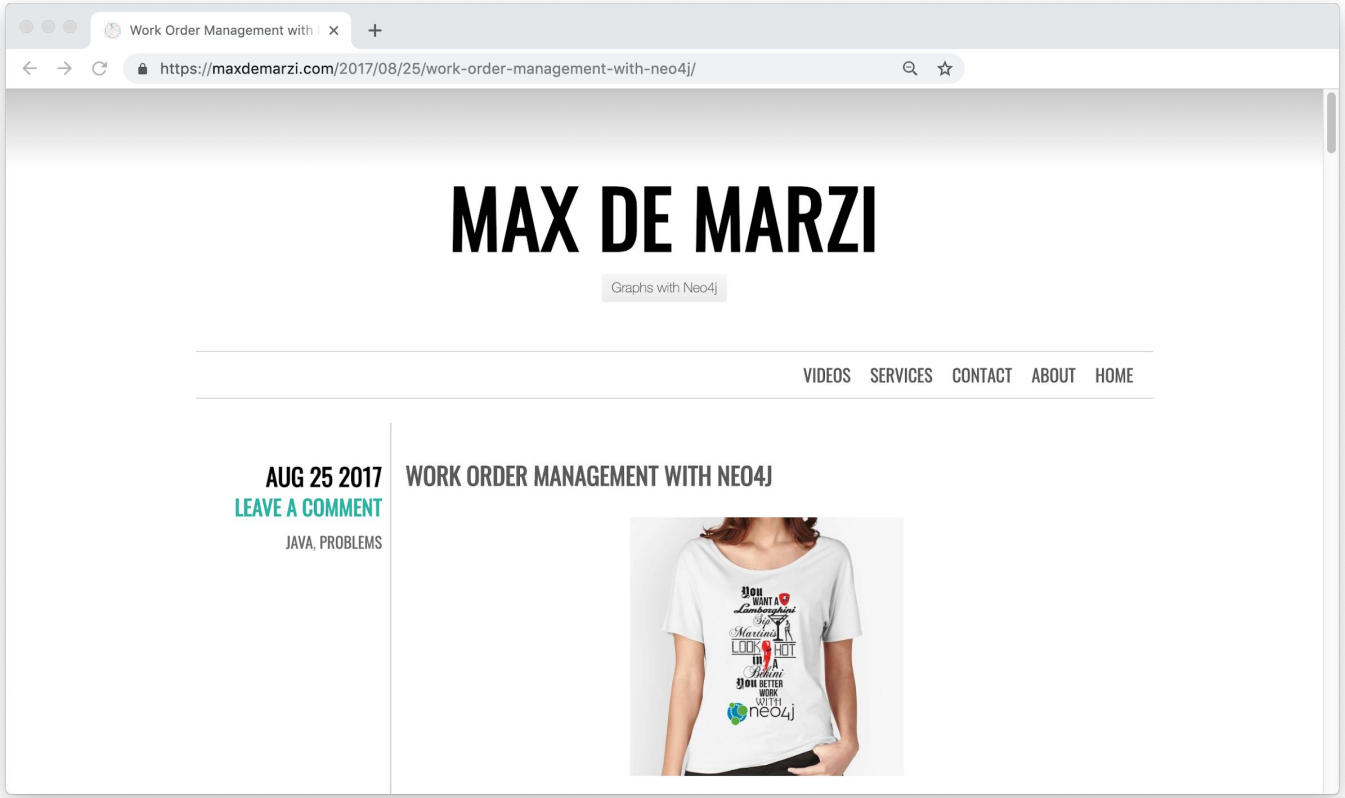
Fraud detection: Replicants



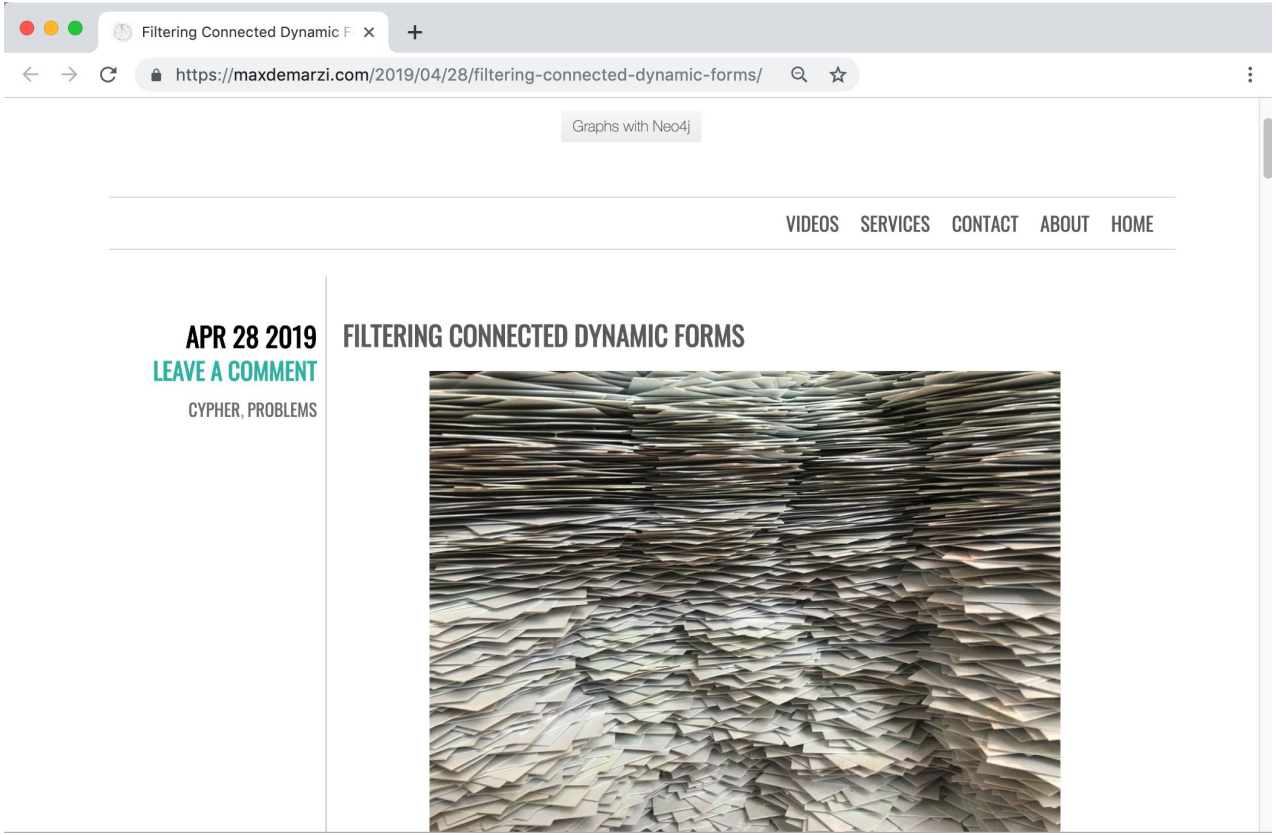
Fraud detection: Paradise Papers Sandbox



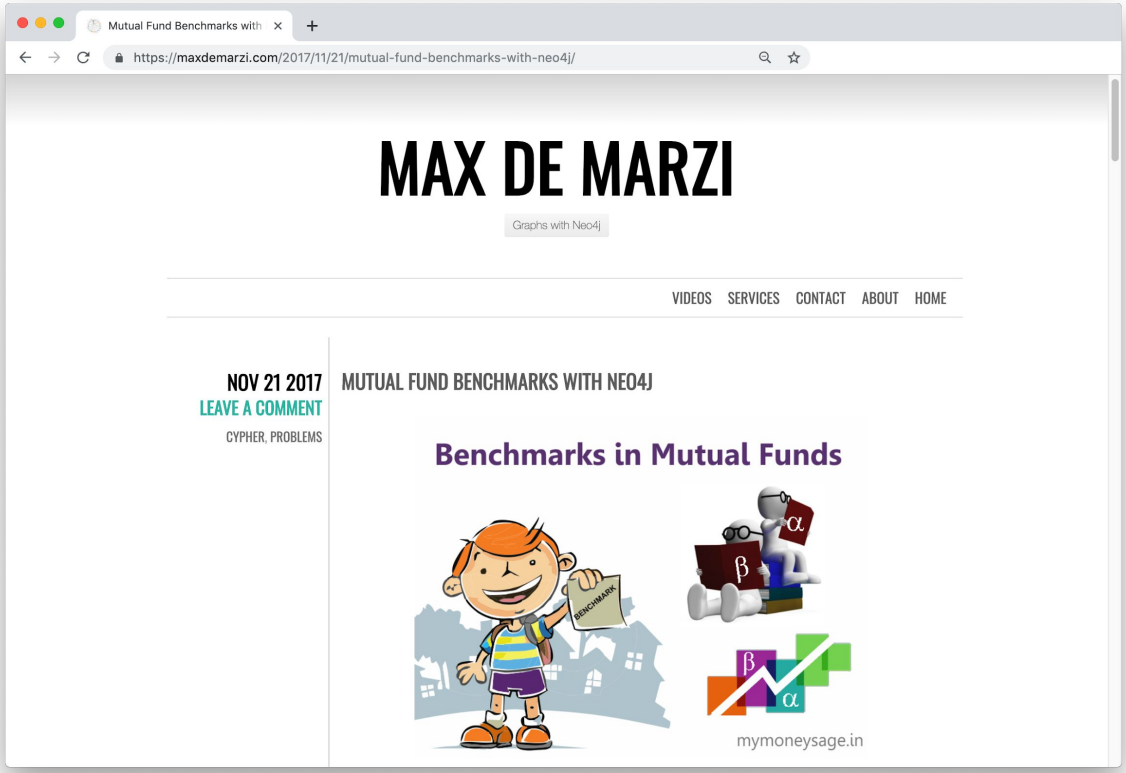
Event modeling: Work Order Management with Neo4j



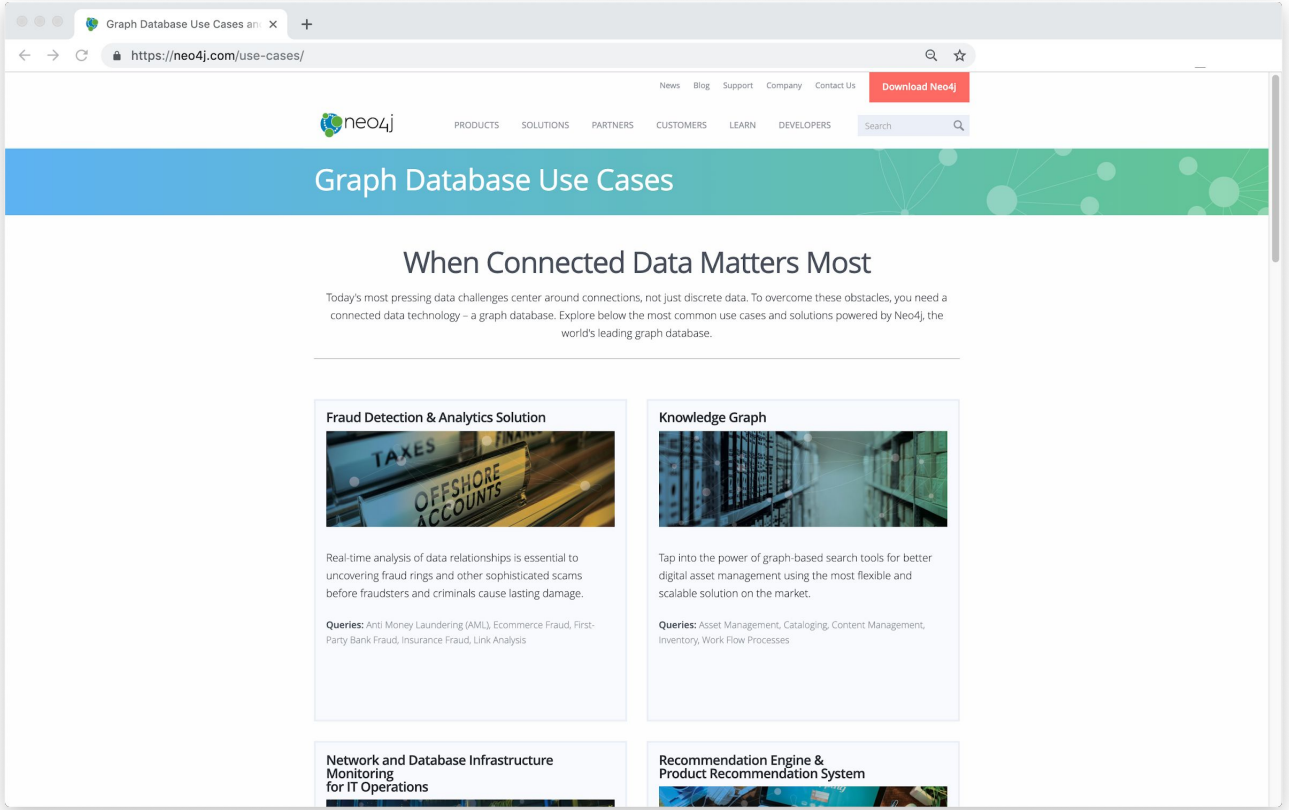
Tracking forms: Filtering connected dynamic forms



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.