Advanced Java Programming Course

Neo4j

Faculty of Information Technologies
Industrial University of Ho Chi Minh City

Session objectives

- NoSQL databases
- Graph databases
- Introduction to Neo4j
- Neo4j: Key concepts and characteristics
- How to install Neo4j
- Graphs and Modeling
- Neo4j and Cypher
- Advanced queries using Cypher

NoSQL databases

- Four types of NoSQL databases:

  + Key-value stores

    Amazon's SimpleDB, Oracle NoSQL Database

  + Document databases

    MongoDB, CouchDB

  + Column family databases

    Cassandra, HBase, Google BigTable

  + Graph databases

    Neo4j, InfiniteGraph

Graph databases

- Graph databases are NoSQL databases that use graph structures with nodes, edges, and properties to represent and store data.
  - Nodes represent entities such as people, businesses, accounts, or any other item to be tracked.
  - Edges represent the relationship between two nodes.
  - Properties are information or attributes about the nodes and edges.
- Optimal for searching social networks, recommendation engines, fraud detection, and knowledge graphs.

Why graph databases?

- In relational databases, data is stored in tables. When we need to query data, we usually need to join multiple tables or cross-looking tables. So the query is complex and slow.
- In a graph database, there are no JOINs or lookups. Relationships are stored natively alongside the data elements (the nodes) in a much more flexible format. Everything about the system is optimized for traversing through data quickly; with millions of connections per second, per core.

Why graph databases?

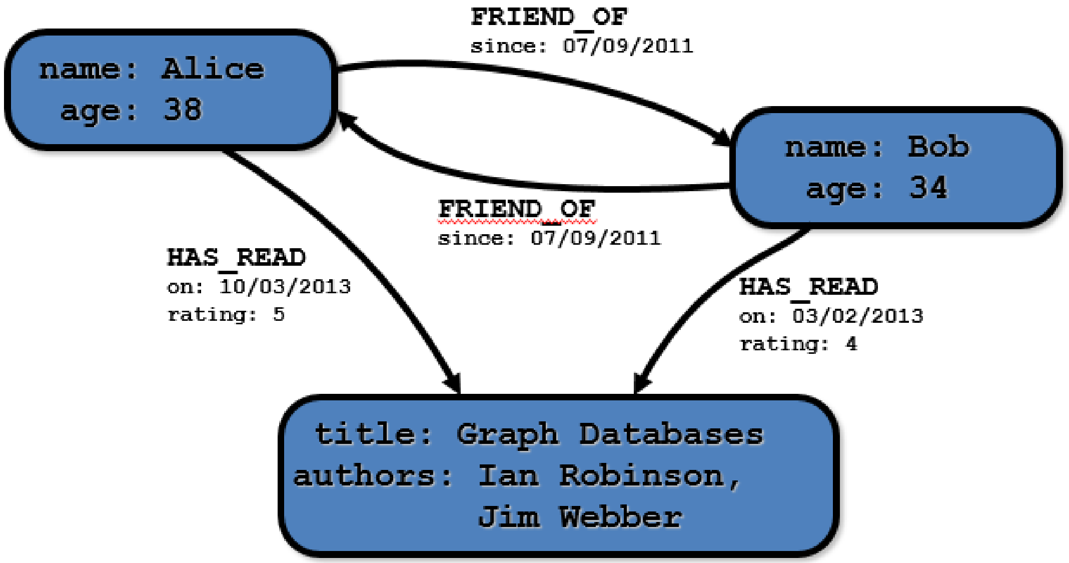
- Graph databases address big challenges many of us tackle daily. Modern data problems often involve many-to-many relationships with heterogeneous data that set up needs to:
  - Navigate deep hierarchies.
  - Find hidden connections between distant items.
  - Discover inter-relationships between items.

Introduction to Neo4j

- Neo4j is a graph database management system developed by Neo4j, Inc.
- Introduced in 2010
- Open source
- Java based
- NoSQL graph database

Neo4j

- Neo4j Graph Database is the core product, a native graph database that is built to store and retrieve connected data.
- Neo4j stores data in nodes and relationships between nodes.



How to install Neo4j

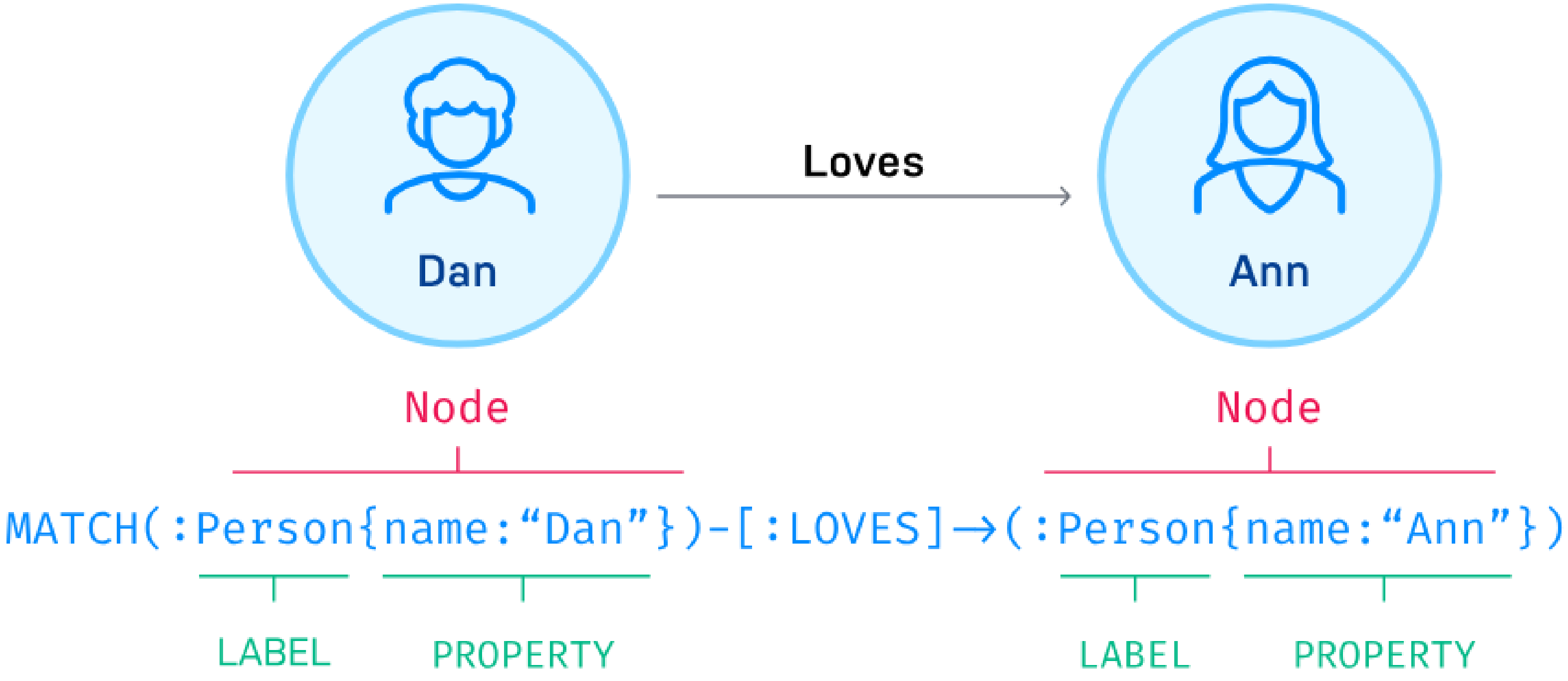
- Download Neo4j Desktop from https://neo4j.com/download/
- Install Neo4j Desktop
- Create a new project
- Create a new database
- Start the database
- Open the database
- Open the Neo4j Browser
- CRUD operations in Neo4j Browser

How to install Neo4j

- Demo

How does Neo4j database work?

- In Neo4j, information is organized as nodes, relationships, and properties.



How does Neo4j database work?

- Nodes are the entities in the graph.
  - Nodes can be tagged with labels
  - Nodes can hold any number of key-value pairs, or properties.
  - Node labels may also attach metadata (such as index or constraint information) to certain nodes.
- Relationships provide directed, named connections between two node entities
  - Relationships always have a direction, a type, a start node, and an end node, and they can have properties, just like nodes.
  - Nodes can have any number or type of relationships without sacrificing performance.
  - Although relationships are always directed, they can be navigated efficiently in any direction.

Neo4j Data browser

- http://localhost:7474/browser/
- The Data browser is useful for easy introspection, exploration, or debugging.
- Get them to enter 1, and then switch to the visualizer. Walk through changing the style to show the {title} property, and use a circle, for nodes that have that prop.

Property Graph Model

- Characteristics
  - It contains Nodes and Relationships, both of which can contain properties (key-value pairs).
  - Relationships are always between exactly 2 nodes.
  - They have a type, and they are directed.
- Four types of elements in a property graph model:
  - Node
  - Relationship
  - Property
  - Label

Nodes

- Used to represent entities in the domain
- Ex: Person, book, movie, or any other item to be tracked
- Can contain properties
  - Used to represent attributes of the entity
  - Key-value pairs
- Every node has a unique identifier
- Every node can have different properties

Relationships

- Every relationship has a name and a direction.
  - Add structure to the graph
  - Provide semantic context for nodes
- Can contain properties
  - Used to represent quality or weight of relationship, or metadata
- Every relationship must have a start node and end node
  - No dangling relationships
- Node can have more than one relationship with other nodes
- Self relationship is allowed

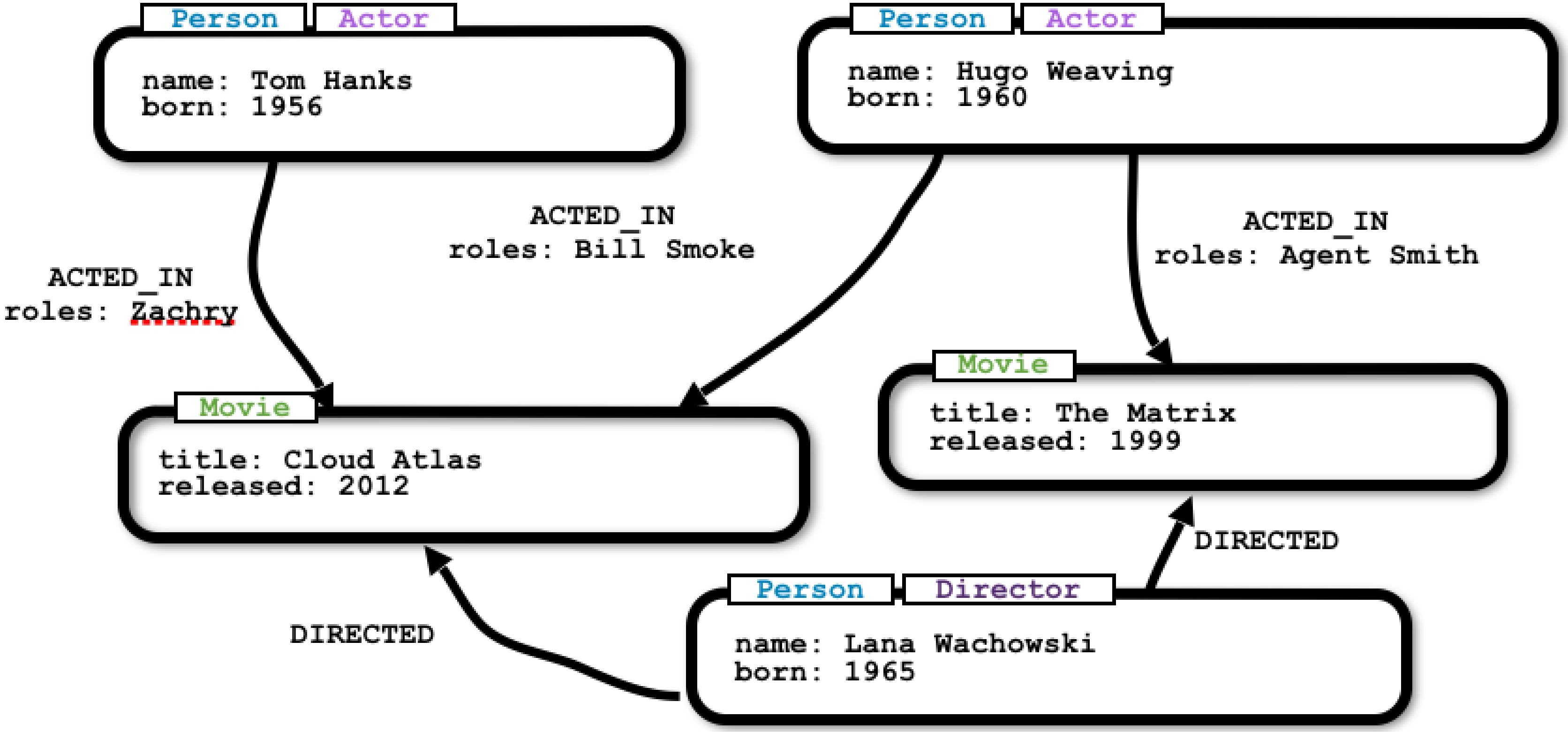
Labels

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

Four Building Blocks

- Nodes
  - Entities
- Relationships
  - Connect entities and structure domain
- Properties
  - Attributes and metadata
- Labels
  - Group nodes by role

Entities, Relationships and labels



Designing a graph model

- Identify application / end-user requirements
- Figure out what questions the ask of the domain
- Identify the entities in each question
- Identify the relationships between the entities in each question
- Convert the entities and relationships to path
  - These become the basis of the data model
- Express questions as graph patterns
  - These become the basis for queries

Designing a graph model

1. End user goals
   - As an employee
   - I want to know who in the company has similar skills to me. So that we can exchange knowledge
2. Questions To Ask of the Domain
   - Which people, who work for the same company as me, have similar skills to me?
3. Identify Entities

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

  Person

  Company

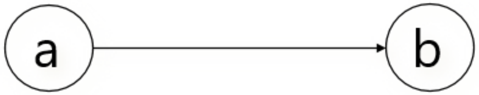
  Skill

Cypher Query Language

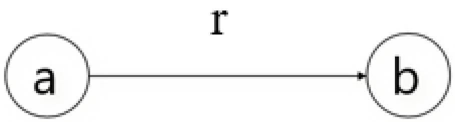
- Cypher is Neo4j's graph query language
- Declarative Pattern-Matching language
- SQL-like syntax
- Designed for graphs

Path

- Relationship giữa 2 nodes

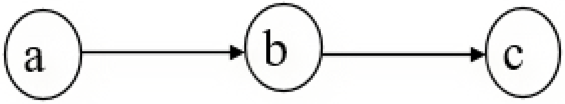


(a) --> (b)

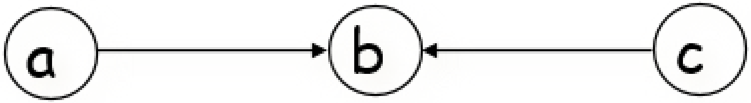


(a) -[:r]-> (b)

Paths



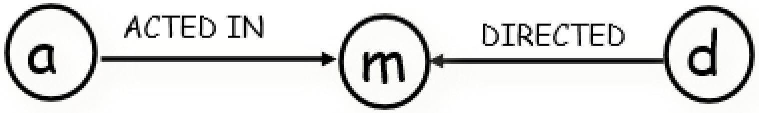
(a)-->(b)-->(c)



(a)-->(b)<--(c)

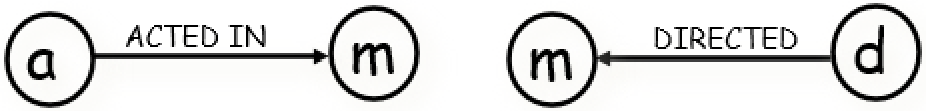
Paths

```
START a=node(*)
MATCH (a)-[:ACTED_IN]->(m)<-[:DIRECTED]-(d)
RETURN a.name, m.title, d.name;
```



Paths

```
START a=node(*)
MATCH (a)-[:ACTED_IN]->(m), (m)<-[:DIRECTED]-(d)
RETURN a.name, m.title, d.name;
```



Aggregation

```
START a=node(*)
MATCH (a)-[:ACTED_IN]->(m)<-[:DIRECTED]-(d)
RETURN a.name, m.title, d.name;
```

a.name	m.title	d.name
“Keanu Reeves”	“The Matrix”	“Andy Wachowski”
“Keanu Reeves”	“The Matrix Reloaded”	“Andy Wachowski”
“Noah Wyle”	“A Few Good Men”	“Rob Reiner”
“Tom Hanks”	“Cloud Atlas”	“Andy Wachowski”
…	…	…

Aggregation

```
START a=node(*)
MATCH (a)-[:ACTED_IN]->(m)<-[:DIRECTED]-(d)
RETURN a.name, d.name, count(*); // count(m)
```

a.name	d.name	count(*)
“Aaron Sorkin"	“Rob Reiner”	2
“Keanu Reeves”	“Andy Wachowski”	3
“Hugo Weaving”	“Tom Tykwer”	1
...	...	...

Sort & Limit

```
START a=node(*)MATCH (a)-[:ACTED_IN]->(m)<-
[:DIRECTED]-(d)RETURN a.name, d.name, count(*) AS count
ORDER BY(count) DESC
LIMIT 5;
```

Aggregation

https://neo4j.com/docs/cypher-manual/current/functions/aggregating/

All-nodes Query

START n=node(*) RETURN n;

- START - clause for looking up starting points
- node(*) - all nodes in the graph
- RETURN n - clause to specify data to return

Find a specific node (all-node query)

```
START n=node(*)
WHERE has(n.name) AND n.name = "Tom Hanks"
RETURN n;
```

- WHERE - filter the results
- has(n.name) - the name property must exist
- n.name = "Tom Hanks" - and have that value

Constraints on properties

- Movies in which Tom Hanks acted, that were released before 1980?

```
START tom=node:node_auto_index(name="Tom Hanks")
MATCH (tom)-[:ACTED_IN]->(movie)
WHERE movie.released < 1992
RETURN DISTINCT movie.title;
```

Updating Graphs with Cypher

- Creating nodes

```
CREATE ({title:"Mystic River", released:1993});
```

- Adding properties

```
START movie=node:node_auto_index(title="Mystic River")
SET movie.tagline = "We bury our sins here, Dave. We wash them clean."
RETURN movie;
```

- Deleting nodes

```
MATCH (n: Movie) WHERE n.movieID = "Matrix" DELETE n
```

Creating Relationships

- Creating nodes

```
CREATE ({title:"Mystic River", released:1993});
```

- Adding properties

```
START movie=node:node_auto_index(title="Mystic River")
SET movie.tagline = "We bury our sins here, Dave. We wash
them clean."
RETURN movie;
```

- Creating Relationships
- Deleting nodes

```
MATCH (n: Movie) WHERE n.movieID = "Matrix" DELETE n
```

Matching multiple relationships

- Create KNOWS relationships between anyone, Actors or Directors, who worked together.

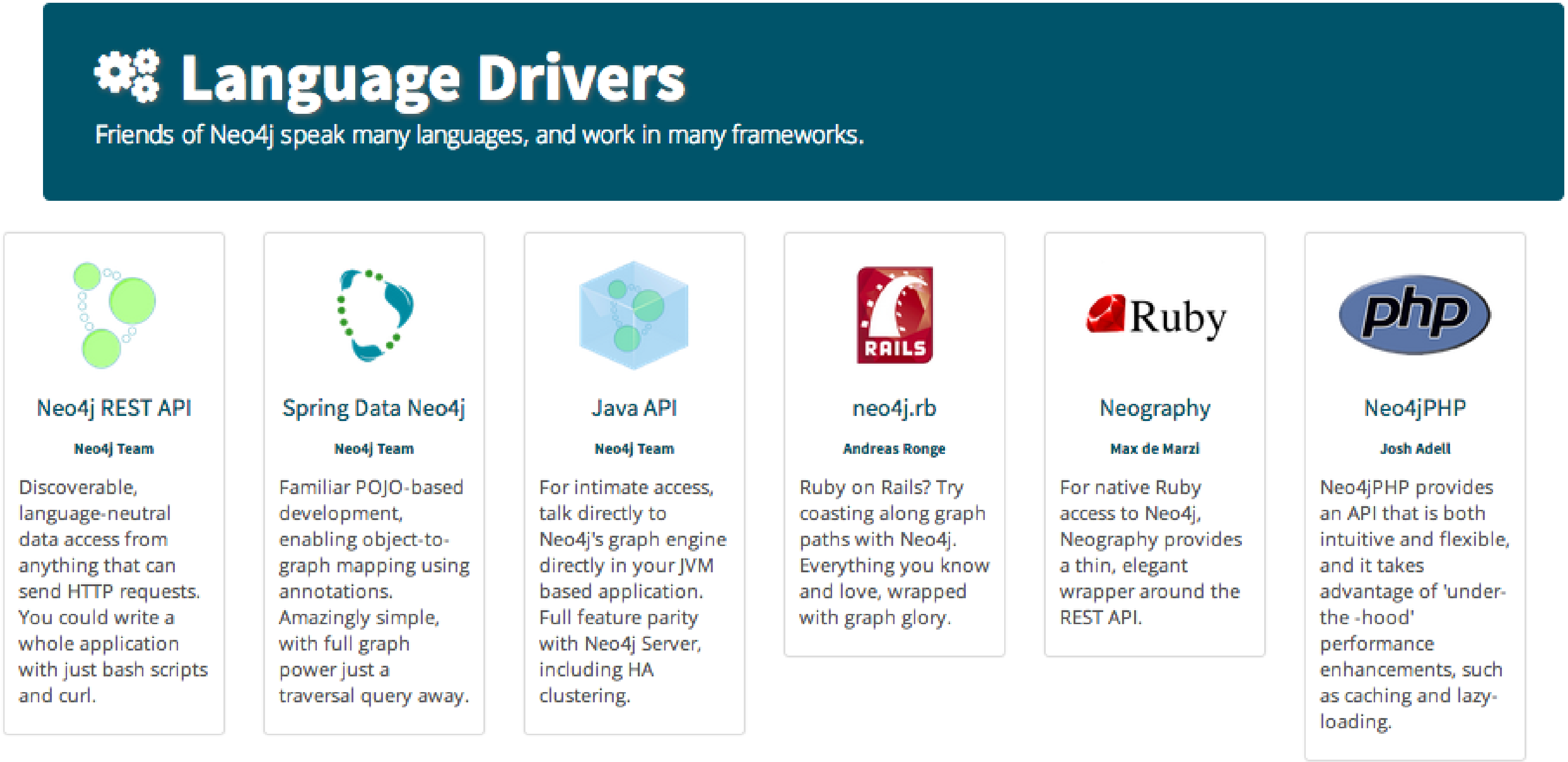
```
START a=node(*)
MATCH (a)-[:ACTED_IN|DIRECTED]->()<-[:ACTED_IN|DIRECTED]-(b)
CREATE UNIQUE (a)-[:KNOWS]->(b);
```

(More about Neo4j)

Neo4j APIs

Neo4j offers three APIs:

1. Cypher for most work
2. REST for management
3. Plugin API for special cases



Language Drivers

Friends of Neo4j speak many languages, and work in many frameworks.

Neo4j REST API

Neo4j Team

Discoverable, language-neutral data access from anything that can send HTTP requests. You could write a whole application with just bash scripts and curl.

Spring Data Neo4j

Neo4j Team

Familiar POJO-based development, enabling object-to-graph mapping using annotations. Amazingly simple, with full graph power just a traversal query away.

Java API

Neo4j Team

For intimate access, talk directly to Neo4j's graph engine directly in your JVM based application. Full feature parity with Neo4j Server, including HA clustering.

neo4j.rb

Andreas Ronge

Ruby on Rails? Try coasting along graph paths with Neo4j. Everything you know and love, wrapped with graph glory.

Neography

Max de Marzi

For native Ruby access to Neo4j, Neography provides a thin, elegant wrapper around the REST API.

Neo4jPHP

Josh Adell

Neo4jPHP provides an API that is both intuitive and flexible, and it takes advantage of 'under-the -hood' performance enhancements, such as caching and lazy-loading.

https://neo4j.com/developer/language-guides/