Data Management for Data Science

Master of Science in Data Science
Facoltà di Ing. dell'Informazione, Informatica e Statistica
Sapienza Università di Roma

AA 2018/2019

An Overview of Neo4j

Domenico Lembo
Dipartimento di Ingegneria Informatica,
Automatica e Gestionale A. Ruberti

NEO4J: Overview

Neo4j:

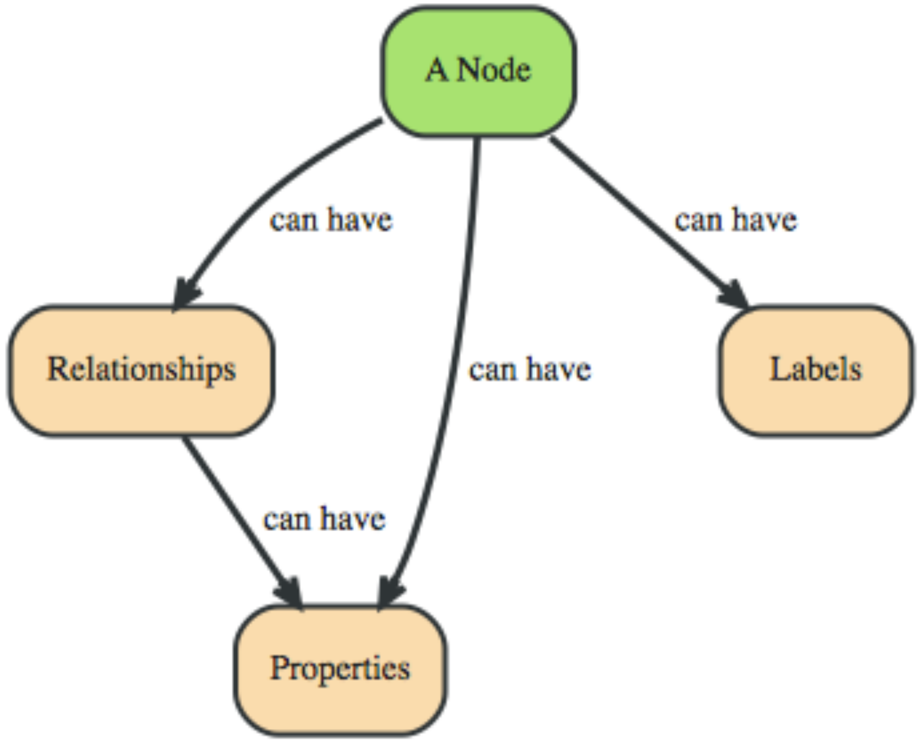
- uses a **graph model** for data representation.
- supports full **ACID transactions**.
- comes with a powerful, human readable **graph query language**.
- provides a powerful **traversal framework** for high-speed graph queries.
- can be used in **embedded mode** (the db is incorporated in the application), or **server mode**, the db is a process in itself which can be accessed through REST Interface.
- **does not allow for sharding,** then the entire graph must be stored in a single machine (at the moment, Neo4j supports cache sharding, which allows for directing queries to instances that only have certain parts of the cache preloaded).

NEO4J: Data Model

Neo4j is entirely implemented in Java. Neo4j's data model is a Property Graph, consists of labeled nodes and relationships each with properties, that is characterized by the following elements:

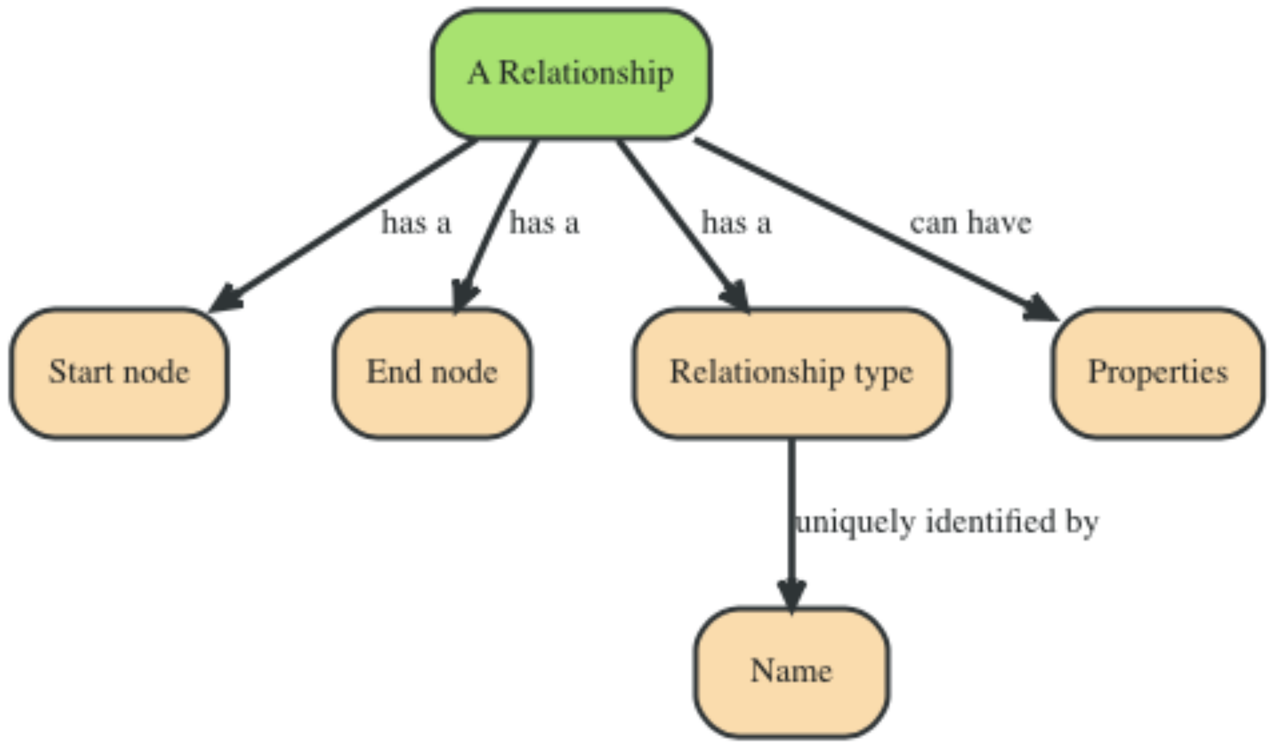
- **Nodes** are just data records, usually denoting entities (e.g., individuals).
- **Relationships** connect two nodes.
- **Properties** are simple **key-value** pairs**.** Properties can be attached to both nodes and relationships

Nodes in NEO4J



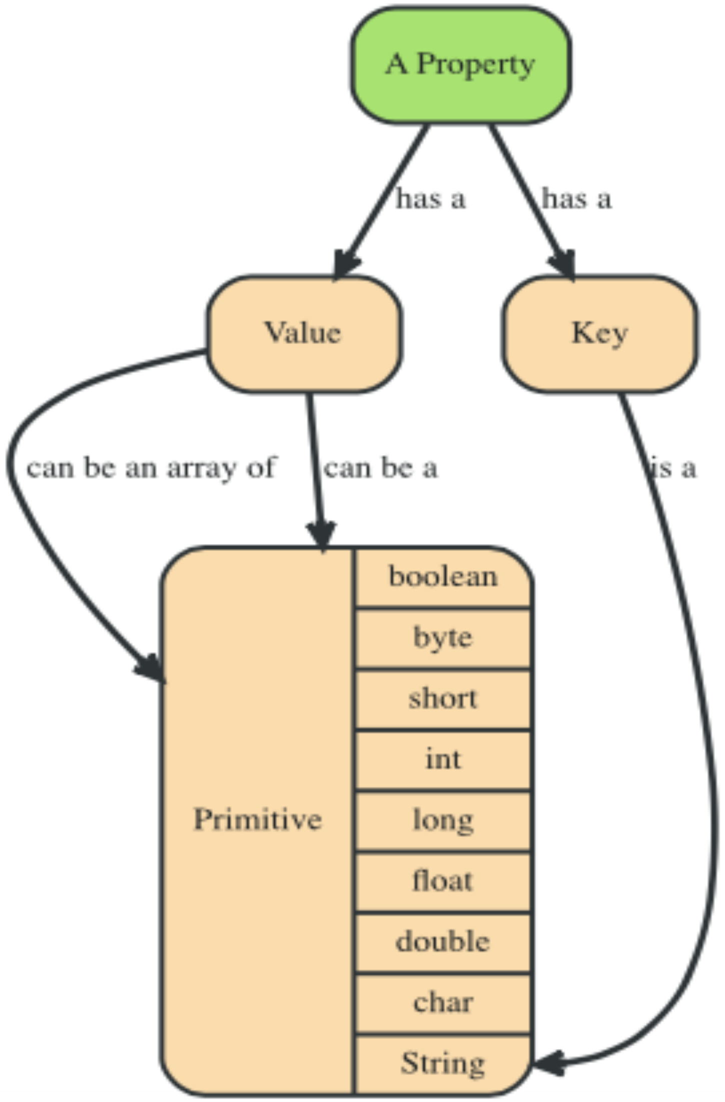
- Every node can have different properties

Relationships in NEO4J

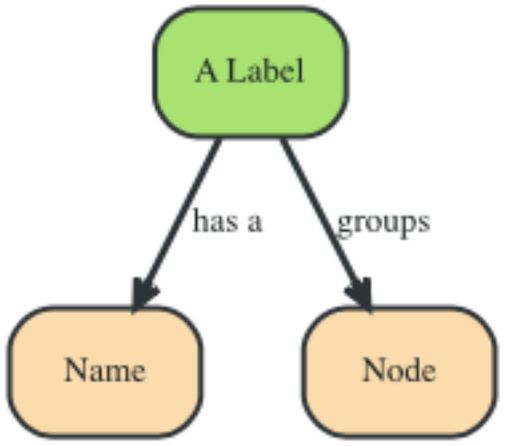


- Every relationship has a direction

Properties in NEO4J

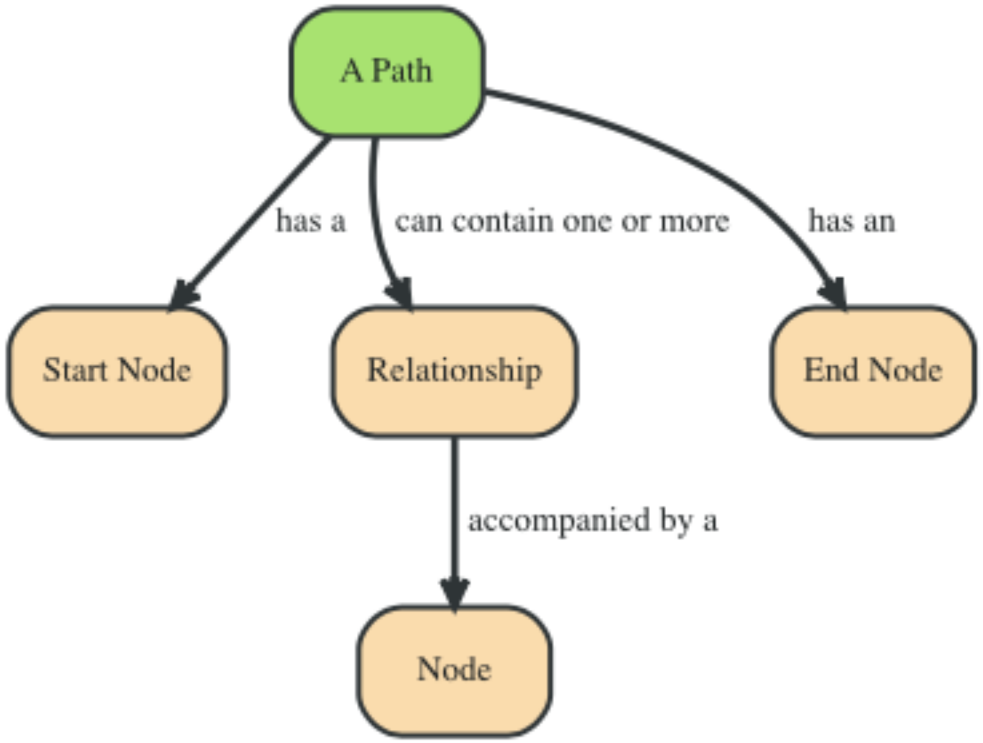


Labels in NEO4J



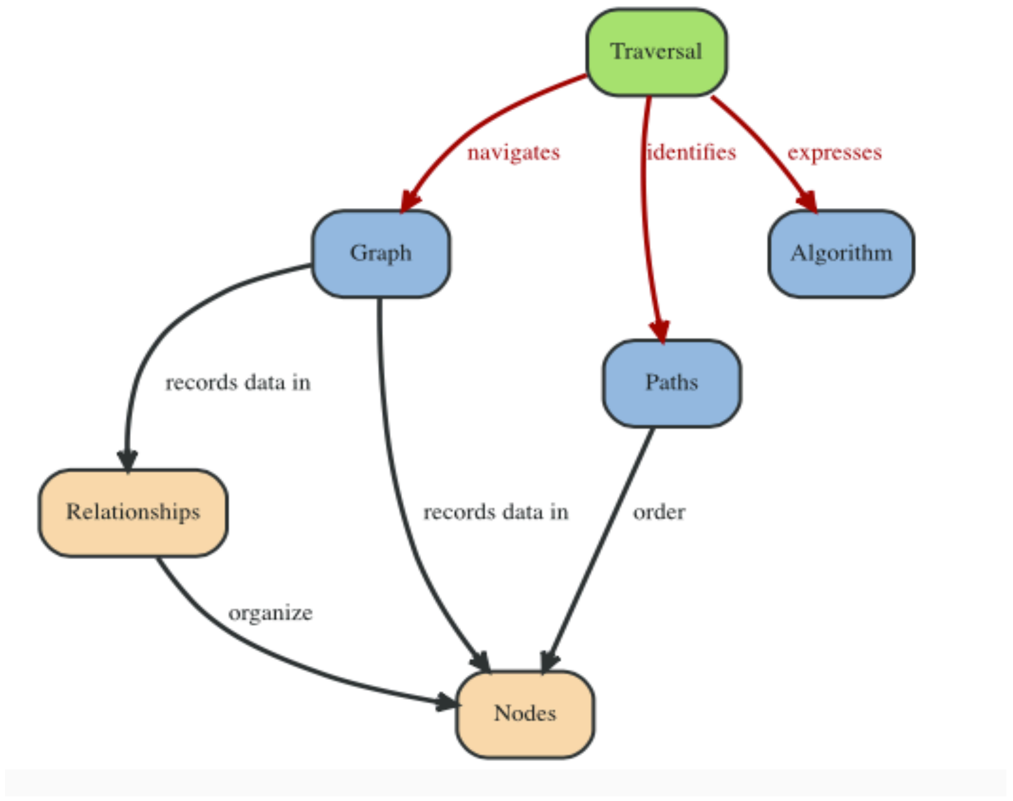
- Used to represent roles played by objects (said in other terms they indicate categories node objects belong to)
- Every node can have zero or more labels

Paths in NEO4J



- It is one or more nodes with connecting relationships

Traversal in NEO4J

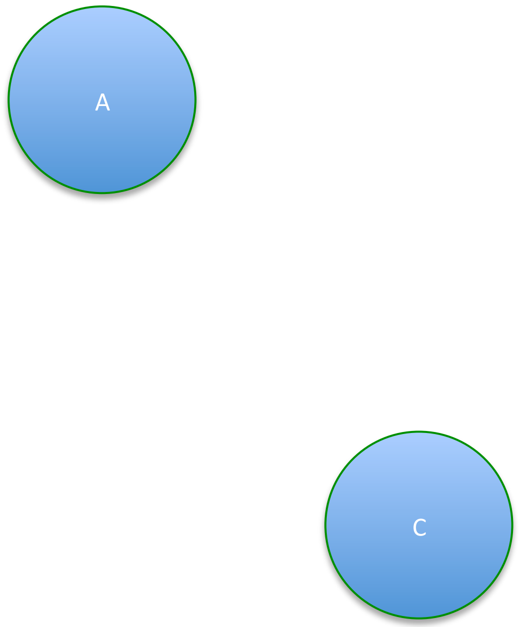


- A Traversal is how you query a Graph, navigating from starting nodes to related nodes according to an algorithm.

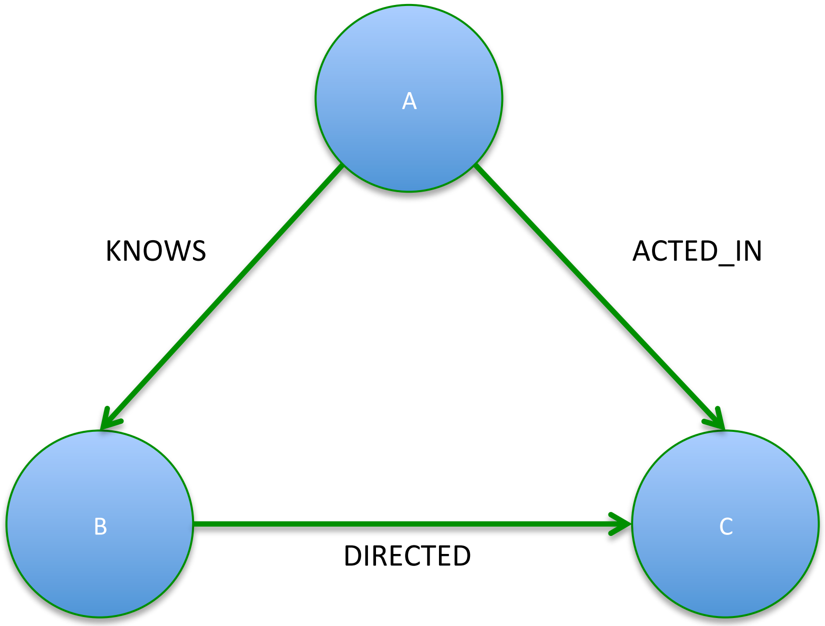
NEO4J: Example of Data Model

- Tom Hanks is an Actor.
- Ron Howard is a Director.
- “The DaVinci Code” is a movie.
- Directors and Actors are Persons.
- Tom Hanks has an acting role in “The DaVinci Code”
- “The DaVinci Code” is directed by Ron Howard
- The role of Tom Hanks in “The DaVinci Code” is Robert Langdon
- Tom Hanks knows Ron Howard since 1987.

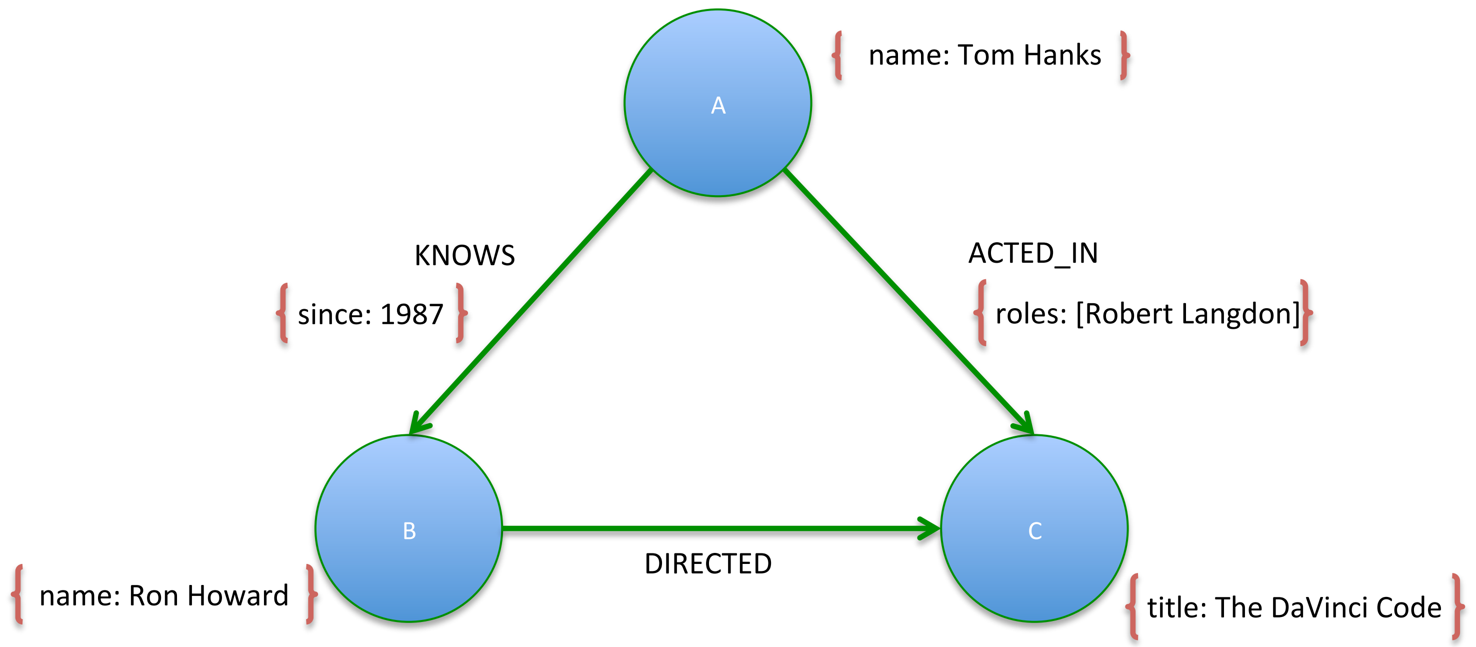
Example: Nodes



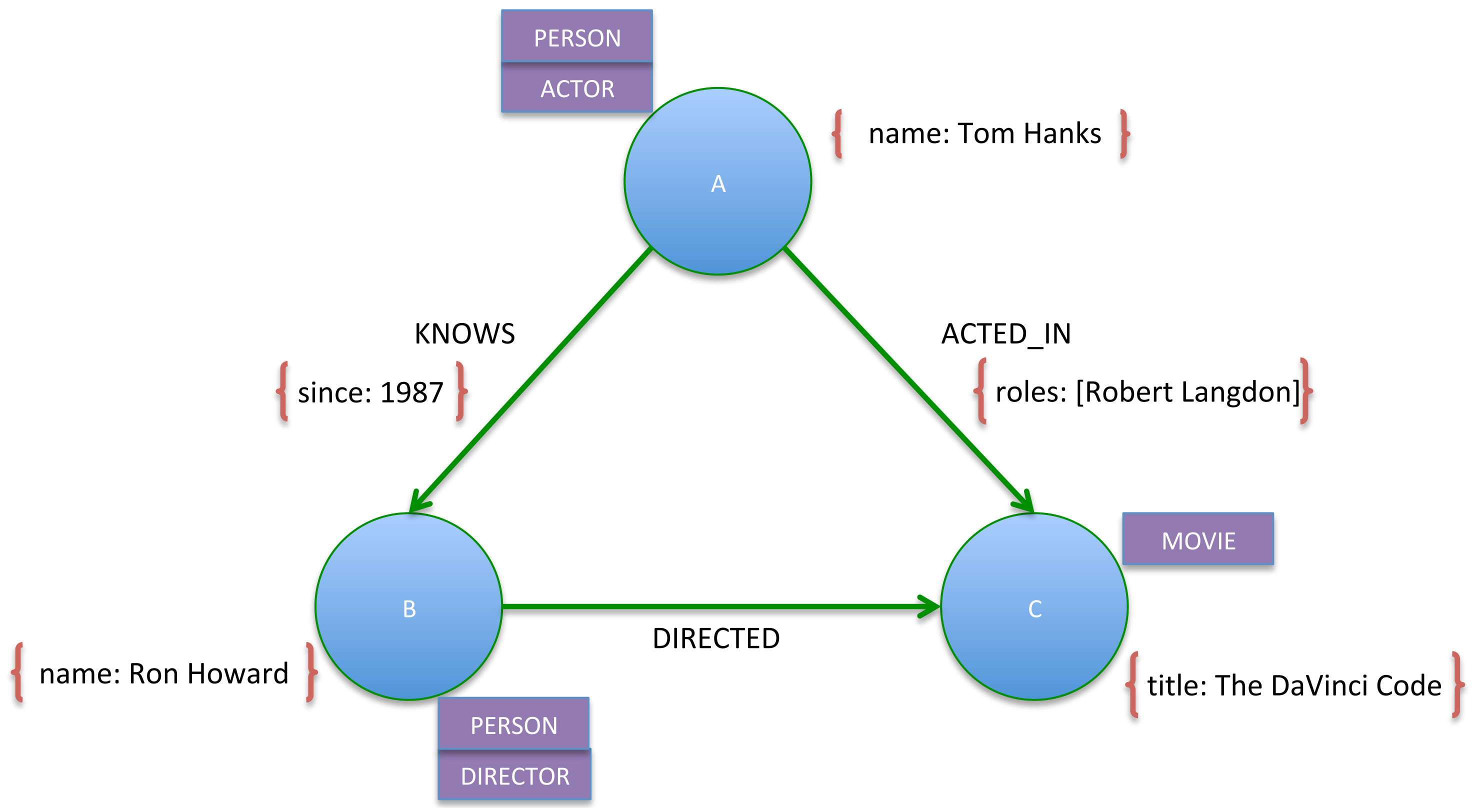
Example: Relationships



Example: Properties



Example: Labels



NEO4J: Storage

- NEO4J uses **native graph storage**, which is optimized and designed for storing and managing graphs. Coherently, it adopts a native graph processing: it leverages index-free adjacency, meaning that connected nodes physically “point” to each other in the database.

- Neo4j integrates an indexing service based on Lucene that allows to store nodes referring to a label, and then access to the iterator of nodes. There are server plugins that allow to automatically index nodes.

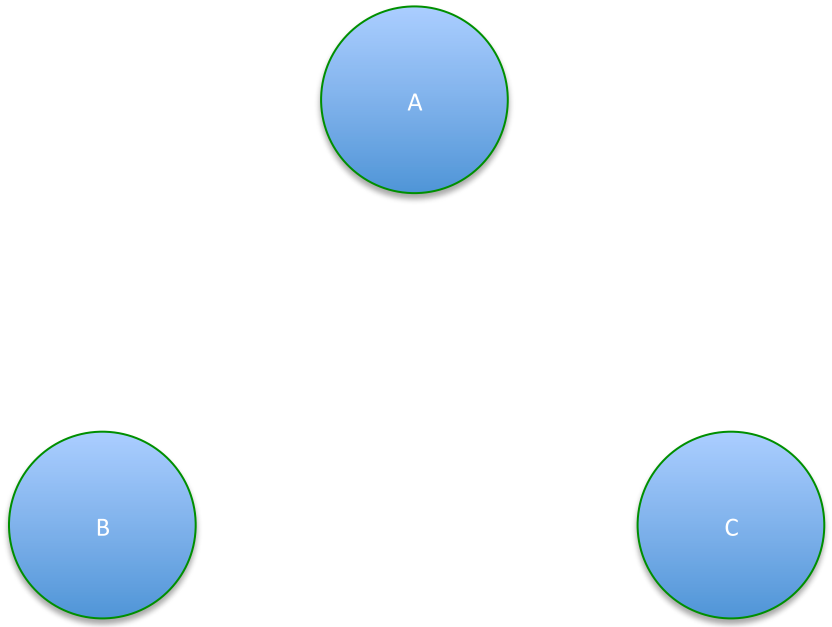
- It is finally provided with an indexing service based on the timestamp that allows to obtain the nodes corresponding to a time and a date included in a certain range

NEO4J: Cypher's introduction

Cypher is a declarative, SQL inspired language for describing patterns in graphs. It allows us to describe *what* we want to select, insert, update or delete from a graph database without requiring us to describe exactly *how* to do it. Cypher uses ASCII-Art* to represent patterns.

*ASCII-Art is a graphic design technique that uses computers for presentation and consists of pictures pieced together from the 95 printable (from a total of 128) characters defined by the ASCII - American Standard Code for Information Interchange (from Wikipedia)

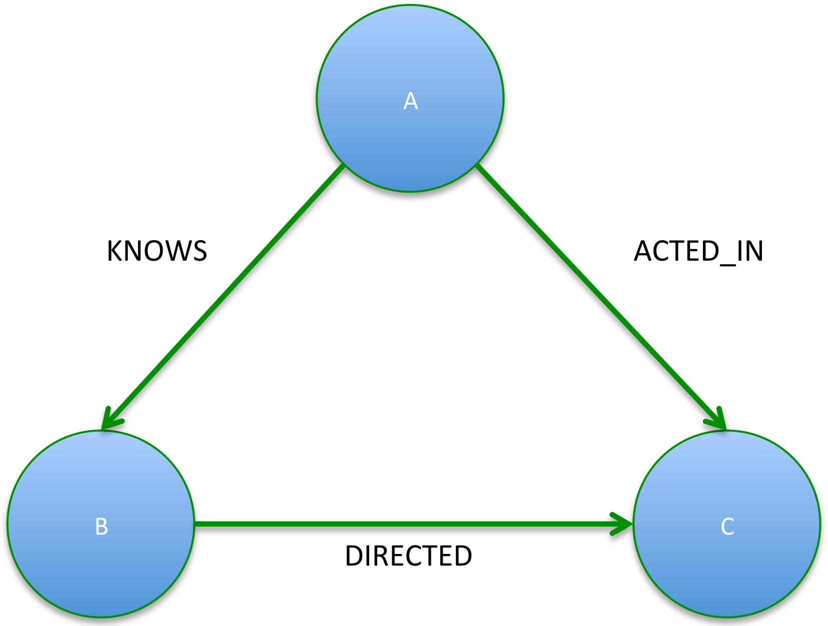
NO4J: Nodes in Cypher



The translation in cypher is:

(A)
(B)
(C)

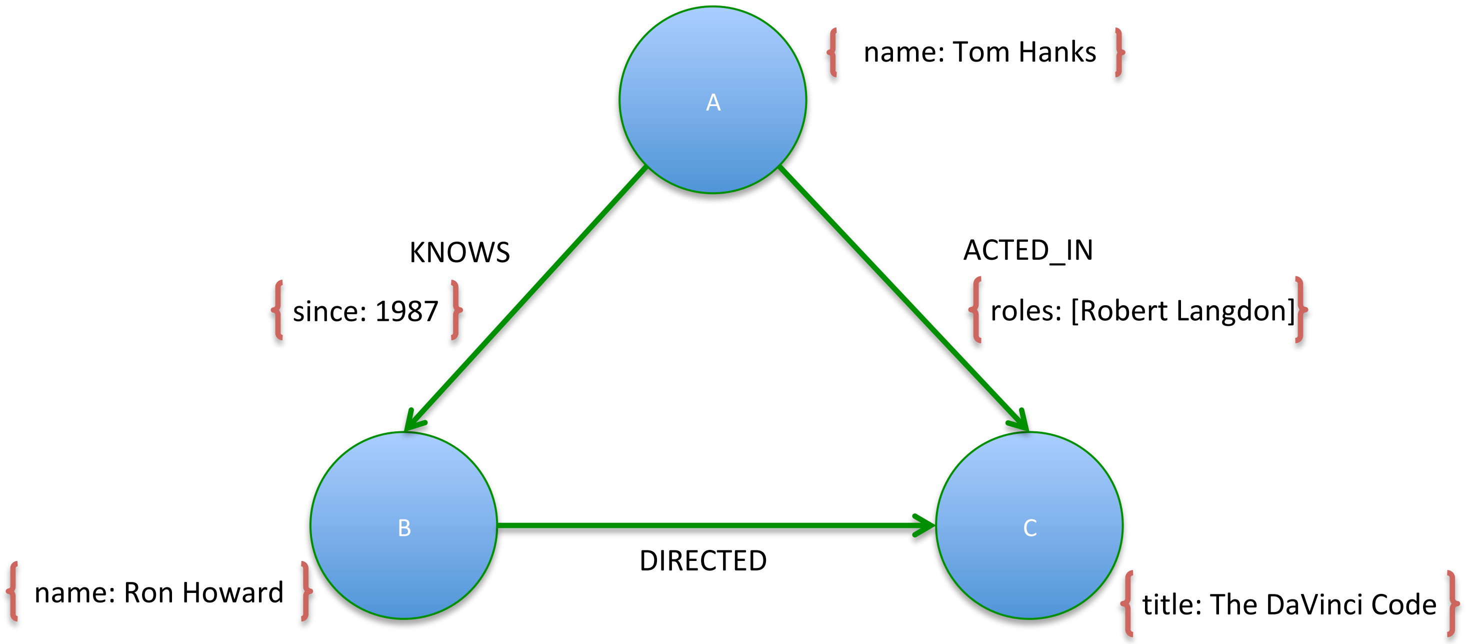
NEO4J: Relationships in Cypher



The translation in cypher is:

```
(B)-[:DIRECTED]->(C)
(A)-[:ACTED_IN]->(C)
(A)-[:KNOWS]->(B)
```

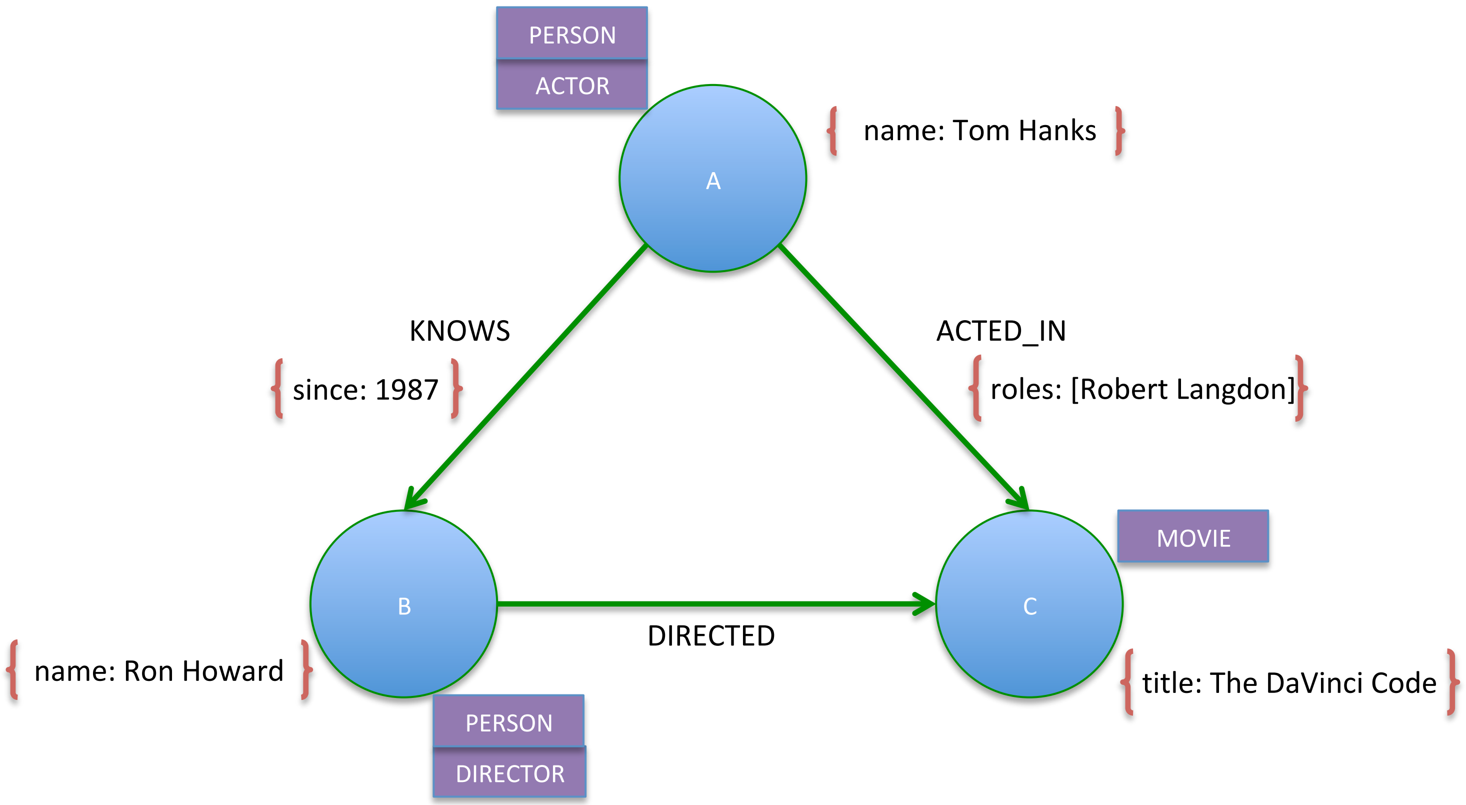
NO4J: Properties in Cypher



The translation in cypher is:

```
(A {name:"Tom Hanks"} )
(B {name:"Ron Howard"} )
(C {title:"The DaVinciCode"} )
(A)-[:ACTED_IN {roles:["Robert Langdon"]}]->(C)
(A)-[:KNOWS {since:1987}]->(B)
```

NEO4J: Labels in Cypher



The translation in cypher is:

```
(A :PERSON)
(B :PERSON)
(C :MOVIE)
(A :ACTOR)
(B :DIRECTOR)
```

NEO4J: Cypher’s query structure

Querying the graph

- **MATCH**: Primary way of getting data from the database.
  WHERE: Filters the results.
  RETURN: Returns and projects result data.
  ORDER BY: Sorts the query result.
  SKIP/LIMIT: Paginates the query result.

Updating the graph

- **CREATE**: Creates nodes and relationships.
  DELETE: Removes nodes, relationships.
  SET: Updates properties and labels.
  REMOVE: Removes properties and labels.
  FOREACH: Performs updating actions once per element in a list, e.g., returned by a match.

CYPHER SCRIPT

```
CREATE (TheDaVinciCode:Movie {title:'The Da Vinci Code', released:2006,
        tagline:'Break The Codes'})
CREATE (TomH:Person:Actor {name:'Tom Hanks', born:1956})
CREATE (RonH:Person:Director {name:'Ron Howard', born:1954})
CREATE (TomH)-[:ACTED_IN {roles:['Dr. Robert Langdon']}]->(TheDaVinciCode)
CREATE (RonH)-[:DIRECTED]->(TheDaVinciCode)
CREATE (TomH)-[:KNOWS {since:1987}]->(RonH)
```

EXAMPLE QUERY IN CYPHER

Return the titles of the films where Tom Hanks acted in and directed by Ron Howard

```
MATCH  (node1)-[:ACTED_IN]->(node2)<-[:DIRECTED]-(node3)

WHERE   node1.name="Tom Hanks" AND node3.name="Ron Howard"

RETURN  node2.title as title
```

Alternative Formulation

```
MATCH  (node1:Person  {name:"Tom Hanks"})-[:ACTED_IN]->(node2)<-
[:DIRECTED]-(node3 {name:"Ron Howard"})

RETURN  node2.title as title
```

WHERE CLAUSE (basics)

You can use the boolean operators AND, OR, XOR and NOT

```
MATCH (n)
WHERE n.name = 'Peter' XOR (n.age < 30 AND n.name = 'Timothy')
        OR NOT (n.name = 'Timothy' OR n.name = 'Peter')
RETURN n.name, n.age
```

To filter nodes by label, write a label predicate after the WHERE keyword using WHERE n:foo.

```
MATCH (n)
WHERE n:Swedish
RETURN n.name, n.age
```

EXAMPLE UPDATING in NEO4J

Create a node Person for Tom Hanks with name attribute:

```
 CREATE (n:Person { name:"Tom Hanks" });
```

Delete a node with name attribute="Tom Hanks" if it exists:

```
MATCH (n { name:"Tom Hanks" }) DELETE n
```

Update a node with name attribute="Tom Hanks" with the attribute age=63:

```
MATCH (n { name:"Tom Hanks" }) SET n.age=63
```

Othe commands in Cypher

ID: allows to retrieve a node with a certain neo4j assigned identifier
count(rel/node/prop): add up the number of occurrences
min(n.prop): get the lowest value
max(n.prop): get the highest value
sum(n.prop): get the sum of numeric values
avg(n.prop): get the average of a numeric value
DISTINCT: remove duplicates
collect(n.prop): collects all the values into a list

Examples:

```
MATCH (s) WHERE ID(s)=100 RETURN s
MATCH (n:Person) RETURN count(*)
MATCH (n:Person) RETURN avg(n.age)
MATCH (n:Person) RETURN collect(n.born)
```

Credits

These Slides for the most are adapted by the original slide of a student project carried out by Giulio Ganino.

The main bibliographic sources used for their preparation are:

www.neo4j.org/

Ian Robinson, Jim Webber, and Emil Eifrem, Graph Databases
Jonas Partner, Aleksa Vukotic, and Nicki Watt. *Neo4j in Action*. 2012