An Overview of Neo4j

Domenico Lembo
Dipartimento di Ingegneria Informatica, Automatica e Gestionale A. Ruberti
NEO4J: Overview

- **Intuitive**, using a graph model for data representation.
- **Reliable**, with full ACID transactions.
- **Totally Transactional**, optimized for transactional performance.
- **Expressive**, with a powerful, human readable graph query language.
- **Fast**, with a powerful traversal framework for high-speed graph queries.
- **Embedded mode**, the db is incorporated in the application.
- **Server mode**, the db is a process in itself which can be accessed through REST Interface.
- No **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 is a Graph DB 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 used for an entity (label).
- **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
• 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
A Traversal is how you query a Graph, navigating from starting nodes to related nodes according to an algorithm.
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

A

B

C
Example: Relationships

A

B

C

KNOWS

DIRECTED

ACTED_IN
Example: Properties

A

\{ name: Tom Hanks \}

B

\{ name: Ron Howard \}

C

\{ title: The DaVinci Code \}

KNOWS
\{ since: 1987 \}

ACTED_IN
\{ roles: [Robert Langdon] \}

DIRECTED
Example: Labels

A \(\xrightarrow{\text{ACTED_IN}}\) A

B \(\xrightarrow{\text{KNOWS}}\) C

C \(\xrightarrow{\text{DIRECTED}}\) B

- **A**: Tom Hanks (name: Tom Hanks, roles: [Robert Langdon])
- **B**: Ron Howard (name: Ron Howard)
- **C**: The DaVinci Code (title: The DaVinci Code)
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.
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)
The translation in cypher is:

(A)
(B)
(C)
The translation in Cypher is:

- (B)-[:DIRECTED]->(C)
- (A)-[:ACTED_IN]->(C)
- (A)-[:KNOWS]->(B)
The translation in cypher is:

(A {name: "Tom Hanks"})
(B {name: "Ron Howard"})
(C {title: "The DaVinci Code"})
(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.
EXAMPLE QUERY IN CYPER

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:Person {name:"Ron Howard"})
RETURN node2.title as title
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=58:
MATCH (n { name:"Tom Hanks" }) SET n.age=58
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)=4 RETURN s
MATCH (n:Person) RETURN count(*)
MATCH (n:Person) RETURN avg(n.age)
MATCH (n:Person) RETURN collect(n.born)
Let’s now quickly look at Neo4J Internals....
Architecture NEO4J

- **Traversal**: Algorithms to traverse the graphs implemented on top of the Core API.
- **Cypher**: Builds on the same infrastructure as Traversal.
- **Record files**: Manage the on-disk storage.
- **File System Cache**: Caches regions of the store files; Regions are evicted based on a LFU-like policy.
- **Object cache**: Optimized for traversal: the nodes hold references to all its relationships (whereas on the disk they just refer their first relationship).
- Transactions involve two parts: The (thread local) changes being done by an active transaction, and the transaction **replay log** for recovery.
NEO4J: Store Files

• Record File manages storage of data on Disk
• Neo4j stores graph data in a number of different store files.
• Each store file contains the data for a specific part of the graph (e.g., nodes, relationships, properties)
  – neostore.nodestore.db
  – neostore.relationshipstore.db
  – neostore.propertystore.db
  – neostore.propertystore.db.index
  – neostore.propertystore.db.strings
  – neostore.propertystore.db.arrays
Simple sample graph. It all boils down to linked lists of fixed size records on disk. Properties are stored as a linked list of property records, each holding key-value.

Each node/relationship references its first property record.
The Nodes also reference the first node in its relationship chain.
Each Relationship references its start and end node.
It also references the prev/next relationship record for the start/end node respectively.
Simple sample graph. It all boils down to linked lists of fixed size records on disk. Properties are stored as a linked list of property records, each holding key+value. Each node/relationship references its first property record. The Nodes also reference the first node in its relationship chain. Each Relationship references its start and end node. It also references the prev/next relationship record for the start/end node respectively.
Simple sample graph. It all boils down to linked lists of fixed size records on disk. Properties are stored as a linked list of property records, each holding key/value. Each node/relationship references its first property record. The Nodes also reference the first node in its relationship chain. Each Relationship references its start and end node. It also references the prev/next relationship record for the start/end node respectively.
Simple sample graph. It all boils down to linked lists of fixed size records on disk. Properties are stored as a linked list of property records, each holding key-value. Each node/relationship references its first property record. The Nodes also reference the first node in its relationship chain. Each Relationship references its start and end node. It also references the prev/next relationship record for the start/end node respectively.

Name
Alistair
Age
34

Name
Tobias
Age
27
Nationality
Swedish

Name
Jim
Age
37
Stuff
good
Your graph on disk

Name: Alistair
Age: 34

Name: Tobias
Age: 27
Nationality: Swedish

Name: Jim
Age: 37
Stuff: good

Simple sample graph. It all boils down to linked lists of fixed size records on disk. Properties are stored as a linked list of property records, each holding key/value. Each node/relationship references its first property record. The Nodes also reference the first node in its relationship chain. Each Relationship references its start and end node. It also references the prev/next relationship record for the start/end node respectively.
Your graph on disk

Simple sample graph. It all boils down to linked lists of fixed size records on disk. Properties are stored as a linked list of property records, each holding key-value. Each node/relationship references its first property record. The Nodes also reference the first node in its relationship chain. Each Relationship references its start and end node. It also references the prev/next relationship record for the start/end node respectively.
Simple sample graph. It all boils down to linked lists of fixed size records on disk. Properties are stored as a linked list of property records, each holding key-value. Each node/relationship references its first property record. The nodes also reference the first node in its relationship chain. Each relationship references its start and end node. It also references the prev/next relationship record for the start/end node respectively.
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

Slides on Ne4J Internals are mainly taken from a presentation by Tobias Lindaaker Hacker, Neo4J Technology (https://www.slideshare.net/thobe/an-overview-of-neo4j-internals )