

B4M36DS2: Database Systems 2

<http://www.ksi.mff.cuni.cz/~svoboda/courses/2016-1-B4M36DS2/>

Practical Class 7

Neo4j Graph Database

Martin Svoboda

svoboda@ksi.mff.cuni.cz

28. and 29. 11. 2016

Charles University in Prague, Faculty of Mathematics and Physics

Czech Technical University in Prague, Faculty of Electrical Engineering

Data Model

Database system structure

Instance → single **graph**

Property graph = directed labeled multigraph

- Collection of vertices (**nodes**) and edges (**relationships**)

Graph **node**

- Has a unique (internal) **identifier**
- Can be associated with a **set of labels**
 - Allow us to categorize nodes
- Can also be associated with a **set of properties**
 - Allow us to store additional data together with nodes

Data Model

Graph **relationship**

- Has a unique (internal) **identifier**
- Has a **direction**
 - Relationships are equally well traversed in either direction!
 - Directions can be ignored when querying
- Always has a **start** and **end node**
 - Can be recursive (i.e. loops are allowed)
- Is associated with **right one** **type**
- Can also be associated with a **set of properties**

Data Model

Node and relationship **property**

- **Key-value pair**
 - Key is a string
 - Value is an **atomic value** of any primitive data type, or an **array of atomic values** of one primitive data type

Primitive **data types**

- boolean – **boolean** values true and false
- byte, short, int, long – **integers** (1B, 2B, 4B, 8B)
- float, double – **floating-point numbers** (4B, 8B)
- char – one Unicode character
- String – sequence of **Unicode characters**

Traversal Framework

Traversal framework

- Allows us to express and execute graph traversal queries
- Based on callbacks, executed lazily

Traversal description

- **Defines rules and other characteristics of a traversal**

Traverser

- Initiates and **manages a particular graph traversal** according to...
 - the provided traversal description, and
 - graph node / set of nodes where the traversal starts
- Allows for the **iteration over the matching paths**, one by one

Traversal Framework

Components of a **traversal description**

- **Expanders**
 - What relationships should be considered
- **Order**
 - Which graph traversal algorithm should be used
- **Uniqueness**
 - Whether nodes / relationships can be visited repeatedly
- **Evaluators**
 - When the traversal should be terminated
 - What paths should be included in the query result

Tutorial: Neo4j



First Steps

Download Neo4j distribution

- From our NoSQL server...
 - 147.32.83.196:22
 - /home/NOSQL/neo4j/
 - neo4j-community-3.0.7-unix.tar.gz
 - neo4j-community-3.0.7-windows.zip
- From Neo4j website...
 - <https://neo4j.com/download/other-releases/>
 - Neo4j 3.0.7
 - Community edition
 - ZIP/TAR distribution

First Steps

Unzip Neo4j distribution file

- `tar -zxvf neo4j-community-3.0.7-unix.tar.gz`

Create a new NetBeans project

- Select *Java application* as a project type
- Add all the libraries from Neo4j `lib` directory
 - Use *Add JAR/Folder* in the project context menu

Database

Create a new embedded database

```
import org.neo4j.graphdb.GraphDatabaseService;  
import org.neo4j.graphdb.factory.GraphDatabaseFactory;  
import java.io.File;
```

```
GraphDatabaseService db = new GraphDatabaseFactory()  
    .newEmbeddedDatabase(new File("MyNeo4jDB"));
```

Close the database connection

```
db.shutdown();
```

Transactions

Start a new database transaction

```
import org.neo4j.graphdb.Transaction;
```

```
Transaction tx = db.beginTx();  
try {  
    ...  
    tx.success();  
} catch (Exception e) {  
    tx.failure();  
} finally {  
    tx.close();  
}
```

Nodes

Create graph nodes for a few actors

- Create nodes, add actor labels, add properties
 - trojan, Ivan Trojan, 1964
 - machacek, Jiří Macháček, 1966
 - schneiderova, Jitka Schneiderová, 1973
 - sverak, Zdeněk Svěrák, 1936
- Remember node references

```
import org.neo4j.graphdb.Node;  
import org.neo4j.graphdb.Label;
```

```
Node actor = db.createNode();  
actor.setProperty("id", "trojan");  
actor.setProperty("name", "Ivan Trojan");  
actor.setProperty("year", 1964);  
actor.addLabel(Label.label("actor"));
```

Relationships

Define relationship types for our graph

```
import org.neo4j.graphdb.RelationshipType;
```

```
private static enum MyTypes implements RelationshipType {  
    KNOWS  
}
```

Relationships

Create relationships between our actors

- Create relationships of KNOWS type
 - trojan → machacek
 - trojan → schneiderova
 - machacek → trojan
 - machacek → schneiderova
 - sverak → machacek
- Consider these relationships as symmetric

```
import org.neo4j.graphdb.Relationship;
```

```
actor1.createRelationshipTo(actor2, MyTypes.KNOWS);
```

Graph Traversals

Find all friends of actor *Ivan Trojan*

- Print full actor names

```
import org.neo4j.graphdb.traversal.TraversalDescription;
import org.neo4j.graphdb.traversal.Evaluators;
import org.neo4j.graphdb.traversal.Uniqueness;
import org.neo4j.graphdb.traversal.Traverser;
import org.neo4j.graphdb.Direction;
import org.neo4j.graphdb.Path;
```

```
TraversalDescription td = db.traversalDescription()
    .breadthFirst()
    .relationships(MyTypes.KNOWS, Direction.BOTH)
    .evaluator(Evaluators.excludeStartPosition())
    .uniqueness(Uniqueness.NODE_GLOBAL);
Traverser t = td.traverse(actor);
for (Path p : t) {
    System.out.println(p.endNode().getProperty("name"));
}
```

Nodes and Relationships

Add nodes for movies into our graph

- Create nodes, add `movie` labels, add properties
 - `samotari`, `Samotáři`, 2000
 - `medvidek`, `Medvídek`, 2007
 - `vratnelahve`, `Vratné lahve`, 2006
- Remember node references

Create relationships between movies and actors

- Create relationships of `PLAYS` type
 - `samotari` → `trojan`
 - `samotari` → `machacek`
 - `samotari` → `schneiderova`
 - `medvidek` → `trojan`
 - `vratnelahve` → `sverak`

Graph Traversals

Find all actors that played in *Medvídek* movie **together with all their friends** and friends of friends as well

- Use a single graph traversal, implement a custom evaluator
- Print full actor names

```
import org.neo4j.graphdb.traversal.Evaluator;  
import org.neo4j.graphdb.traversal.Evaluation;
```

```
public static class MyEvaluator implements Evaluator {  
    @Override  
    public Evaluation evaluate(Path path) {  
        return ...;  
    }  
}
```

```
td.evaluator(new MyEvaluator());
```

Cypher Queries

Find all movies

- Express and execute a Cypher query
- Return movie nodes, print movie titles

```
import org.neo4j.graphdb.Result;  
import java.util.Map;
```

```
Result result = db.execute("MATCH (n:movie) RETURN n");  
while (result.hasNext()) {  
    Map<String, Object> row = result.next();  
    Node n = (Node)row.get("n");  
    System.out.println(n.getProperty("title"));  
}
```

References

Embedded database and traversal framework

- <https://neo4j.com/docs/java-reference/current/>

JavaDoc

- <https://neo4j.com/docs/java-reference/current/javadocs/>

Cypher query language

- <https://neo4j.com/docs/developer-manual/current/cypher/>

Cypher reference card

- <https://neo4j.com/docs/cypher-refcard/current/>