

## **MDK: Modern Database Concepts**

<http://www.ksi.mff.cuni.cz/~svoboda/courses/192-MDK/>

Practical Class 12

# **Neo4j**

**Martin Svoboda**

svoboda@ksi.mff.cuni.cz

6. 6. 2020

**Charles University**, Faculty of Mathematics and Physics

**OTH Regensburg**, Faculty of Computer Science and Mathematics

# First Steps

## Download Neo4j distribution

- From our NoSQL server...
  - nosql.ms.mff.cuni.cz:42222
  - /home/MDK/neo4j/
    - neo4j-community-3.0.7-unix.tar.gz
    - neo4j-community-3.0.7-windows.zip

# 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 MyType implements RelationshipType {  
    KNOW  
}
```

# Relationships

## Create relationships between our actors

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

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

```
actor1.createRelationshipTo(actor2, MyType.KNOW);
```



# 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 Description

## Components of a **traversal description**

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

# Graph Traversals

## Find all friends (even indirect) 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(MyType.KNOW, 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 PLAY type
  - samotari → trojan
  - samotari → machacek
  - samotari → schneiderova
  - medvidek → trojan
  - vratnelahve → sverak

# Graph Traversals

**Find all actors** who 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/3.0/>

JavaDoc

- <https://neo4j.com/docs/java-reference/3.0/javadocs/>

Cypher query language

- <https://neo4j.com/docs/developer-manual/3.0/cypher/>

Cypher reference card

- <https://neo4j.com/docs/cypher-refcard/3.0/>