NDBI040: PRACTICAL CLASS 6

NE04J

Based on NDBI040 practical class materials created by Martin Svoboda; Tutor: Pavel Čontoš; November 25th 2020

(RECOMMENDED) REQUIREMENTS

- Database concepts
- Java programming

- Java 8 JDK or newer installed
- Maven
- NetBeans IDE (or another IDE of yours choice)
- macOS / Linux command line or PuTTy / WinSCP on Windows

SERVER ACCESS

CONNECT TO NOSQL SERVER

- `ssh` on macOS / Linux
- `PuTTy` on Windows

- nosql.ms.mff.cuni.cz:42222
- Login and password send by e-mail
- Change your initial password (if not yet changed) by `passwd`

TRANSFER FILES

- `scp` on macOS / Linux
- `WinSCP` on Windows

MAVEN PROJECT

- Make sure that Java 8 or newer is installed on your workstation
- Choose your preferred JAVA IDE
- Apache NetBeans IDE, IntelliJ IDEA, Eclipse, ... even Notepad works

- Download `ndbi040-neo4j` project skeleton from practical class website
- `Open` or `Import` downloaded (maven) project into your IDE's workspace

- (Clean and) `Build` the project in order to download libraries (so code completion works)

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.bignTx();
try {
  /* Your code */
  tx.success();
} catch (Exception ex) {
  tx.failure();
} finally {
  tx.close();
}
```

EXERCISE 1: NODES

- Create graph nodes for a few actors, 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
}
```

EXERCISE 2: 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

- 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

- `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

EXERCISE 3: GRAPH TRAVERSALS (SOLVED)

- Find all friends (even indirect) of actor Ivan Trojan
  - Print full actor names

```
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"));
}
```

```
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;
```

EXERCISE 4: NODES AND RELATIONSHIPS

- Create nodes for movies into our graph, add `MOVIE` labels, add properties
  - samotari, Samotáři, 2000
  - medvidek, Medvídek, 2007
  - vratnelahve, Vratné lahve, 2006
- Remember node references

- Create relationships between movies and actors of `PLAY` type
  - samotari → trojan
  - samotari → machacek
  - samotari → schneiderova
  - medvidek → trojan
  - vratnelahve → sverak

EXERCISE 5: GRAPH TRAVERSALS

- Find all actors who played in Medvidek 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) {
    /* Your code */
  }
}

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

EXERCISE 6: 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/cypher-manual/current/
- Cypher reference card
  - https://neo4j.com/docs/cypher-refcard/current/