Using AsciiArt to Analyse your SourceCode with Neo4j and OSS Tools
Who the hell is this guy?

- Michael Hunger
- Developer Advocate Neo Technology
- Love People and Graphs
- @mesirii | michael@neotechnology.com
What will he talk about?

- What is this Neo4j Graphdatabase thing?
- Ascii-Art Rocks
- Graphs in Your Code - The Idea
- Having Fun with your Code and jQAssistant
- Gimme More
What is a Graph Database?

- labeled Nodes
- directed, typed Relationships
- arbitrary Properties on each
Property Graph Model
What makes it special?

- close to the object model
- prematerialize relationships
- traversals in linear time
- sparse, heterogenous data + schema free
- local queries - explore the neighbourhood
- whiteboard-friendly
Where can/should I use it?

- Impact Analysis (Network, Software)
- Routing / Logistics
- Recommendation, Dating, Job-Search
- Science (Metadata, Drug Research)
- Masterdata, Hierarchy-Mgmt
- Fraud-Detection, Market-Analysis
- Social, .... and many more
(Graphs)-[:LOVE]-(Whiteboards)
Ascii-Art Rocks

- Turn text into pictures
- Turn picture into text
- The Power of Symbols
- Graph Patterns Made easy
- Hacker and Mudder Friendly
- Diffs, VCS
Welcome to the Mages Lair Multiple User Dungeon
Welcome to the Addiction.
Cypher
(Cypher)-[:USES]->(Ascii-Art)

- Declarative Graph Query Language
- Graph Pattern Matching
- Humane, Readable
- Expressive
- Read and Write Graphs
- Tabular Results
A - [:LOVES] -> B
Cypher Query - Example Geekout
Cypher Query - Example Geekout

Setup

1. CREATE (:Year {year:2014})<-[:IN_YEAR]-(geekout:Conference {name:"Geekout"})
2. -[:LOCATION]->(:City {name:"Tallinn"})
3. CREATE (track:Track {name:"Room 1"})-[:TRACK_OF]->(geekout)
4. MERGE (speaker:Attendee:Speaker {name:"Hadi Hariri"})
5. MERGE (geekout)<-[:ATTENDS]-(speaker)
6. CREATE (speaker)-[:PRESENTS]->(session:Session {title:"Mouseless IDE"})<-[:IN_TRACK]-(track)
7. FOREACH (name in ["Java","IDE","Development"] |
8. MERGE (topic:Topic {name:name})
9. CREATE (session)-[:HAS_TOPIC]->(topic))
Cypher Query - Example Geekout

Setup

1. CREATE (:Year {year:2014})-[:IN_YEAR]->(geekout:Conference {name:"Geekout"})
2. -[:LOCATION]->(:City {name:"Tallinn"})
3. CREATE (track:Track {name:"Room 1"})-[:TRACK_OF]->(geekout)
4. MERGE (speaker:Attendee:Speaker {name:"Hadi Hariri"})
5. MERGE (geekout)<-[:ATTENDS]-(speaker)
6. CREATE (speaker)-[:PRESENTS]->(session:Session {title:"Mouseless IDE"})<-[:IN_TRACK]-(track)
7. FOREACH (name in ["Java","IDE","Development"] |
8. MERGE (topic:Topic {name:name})
9. CREATE (session)-[:HAS_TOPIC]->(topic))

Query

1. // Which Speakers entertain you with Java?
2. MATCH (t:Topic {name:"Java"})<-[:HAS_TOPIC]-(session)<-[:PRESENTS]-(speaker),
3. (session)<-[:IN_TRACK]-(track)
4. RETURN speaker.name as speaker, {time:session.time, session:session.title} as session,
5. track.name as track
6. ORDER BY session.time
Let me graph that for you: GeekOut 2014

What better way to show what a graph database is good at, is to show how it helps you to connect people, sessions, topics, time and more.

Query 1 — This query has been used to initialize the console

Which Speakers entertain you with Java?

Query 2

MATCH (t:Topic {name: "Java"})-[HAS_TOPIC]->(p:PRESENTS)<-[HAS_TO]-(s:session)<-[IN_TRACK]-(t:track)
RETURN speaker.name as speaker, [time:session.time, session:session.title] as session
ORDER BY session.time
Software Analytics - Approach

1. look at one interesting aspect
2. which insights would be cool?
3. model it as a graph
4. get data
5. import into graph model
6. enrich graph model with concepts / structure
7. query for insights
(Code)-[:IS_A]->(Graph)

- AST, ByteCode, Source-Code
- Inheritance, Composition, Dependencies
- Transitive Module and Library dependencies
- Dependency injection config
- Data model (db) <-> object model
- Runtime characteristics: call graph, heap
- Version control, repositories, issues
jQAssistant

- Open Source Software Analytics Tool
- Plugins for Java, JEE, JPA, Maven, Gradle, Sonarj ...
- All Cypher based
- Technical and Domain Concept Definitions
- Compute Software Metrics
- Declare and Validate Architectural Rules
- Integrated in Build Process
Actively Looking for Contributions

http://github.com/buschmais/jqassistant
public class Customer extends Person {
    private int number;

    public int getNumber() {
        return this.number;
    }
}
Java software graph model: Nodes

- Artifact
- Package
- Type, Class, Interface, Annotation, Enum
- Method, Constructor, Parameter
- Field
- Value, Class, Annotation, Enum, Primitive, Array
Java software graph model: Relationships

- CONTAINS, DECLARES
- EXTENDS, IMPLEMENTS
- RETURNS, THROWS, INVOKES, HAS, IS
- ANNOTATED_BY, OF_TYPE
Approach

1. Scan your project with Plugins for Code (Java-ASM), Config, Metadata
2. Import into Neo4j
3. Enrich with declared technical and domain concepts
4. On top of those concepts
5. Software-Metrics queries
6. Architectural-Rules queries
Pattern matching is the core principle of Cypher!

MATCH
  (c1:Class)-[:EXTENDS]->(c2:Type)
RETURN
  c1.fqn, c2.fqn
Demo
Analyzing A Maven Repository

Rickard Öberg

This little nifty tool will allow you to import your local Maven repository information into a Neo4j graph, in particular dependencies between artifacts.

You can then take this graph and put it into a Neo4j server, and perform Cypher queries on it.

Or whatever else awesome you want to do.

```bash
mvn compile exec:java -Dexec.mainClass=com.github.rickardoberg.neomvn.Main \
-Dexec.arguments="$HOME/.m2/repository"
```

https://github.com/rickardoberg/neomvn#example-queries
NeoMVN: Example Queries
NeoMVN: Example Queries

Find all transitive dependencies of all artifacts with "org.neo4j" groups

```cypher
MATCH (group:Group {groupId:'org.neo4j'}),
(group)-[:HAS_ARTIFACT]->(artifact)-[:HAS_VERSION]->(version)<-[HAS_DEPENDENCY]->(dependent)
WHERE left(dependent.groupId, 9)<group.groupId
RETURN DISTINCT dependent.artifactId, dependent.groupId
```
NeoMVN: Example Queries

Find all transitive dependencies of all artifacts with "org.neo4j" groups

```
MATCH (group:Group {groupId:'org.neo4j'}),
  (group)-[:HAS_ARTIFACT]->(artifact)-[:HAS_VERSION]->(version)<-[HAS_DEPENDENCY]-(dependent)
WHERE left(dependent.groupId,9)<>group.groupId
RETURN DISTINCT dependent.artifactId, dependent.groupId
```

Which version of JUnit is the most popular

```
MATCH (group:Group {groupId:'junit'})
MATCH (group)-[:HAS_ARTIFACT]->(artifact)-[:HAS_VERSION]->(version)<-[HAS_DEPENDENCY]-(dependent)
RETURN version.version, count(dependent) as depCount
ORDER BY depCount DESC
```
Query a JVM Heapdump

1. Get a heap-dump with `jmap`
   
   ```
   jmap -dump:format=b,file=dump.hprof <pid>
   ```

2. Use `jhat dump.hprof` find the /oql endpoint

3. run the OQL script to generate Cypher code

4. Import into Neo4j

5. Run arbitrary queries

6. Visualize
```javascript
select (function () {
    var entries = filter(p.waitingTxMap.data.table, function(it) { return it; });
    var resources = ""
    var idSeq = 0;

    for ( var i = 0; i < entries.length; i++ ) {
        var resource = entries[i].value.resource;
        var index = resource.index.toString();
        var key = resource.key.toString();
        var indexNameIndexKey = index + "-" + key;
        var resourceName = "n" + idSeq++;

        var resourceName = "";
        var tx = entries[i].key;
        var txName = "n" + idSeq++;

        var cypher = "MERGE (" + txName + ":Tx {txId:" + tx.eventIdentifier + "}<br />";
        cypher += "MERGE (" + indexName + ":IndexLock {indexLockId:" + indexNameIndexKey + "}<br />";
        cypher += "CREATE (" + txName + ")-[:WAITING_ON]->(" + indexName + ")<br />";

        resources += cypher + "<br />"
    }

    ...

    return resources;
})()
from org.neo4j.kernel.impl.transaction.RagManager p
```
Visualization of a Deadlock Scenario
JVM heap analysis using Neo4j

Nat Pryce, James Richardson (Software Engineers, Sky)

- Use Neo4J for ad-hoc analysis of heap use in a proprietary embedded JVM that’s deployed in one of the most widely used consumer products in the UK (Sky Box).
- Used Cypher queries that uncovered surprising aspects of their code, platform and the Java compiler.
- And finding a memory leaking JSON parser with a Cypher query.
Heap Model

From Nat's Graph Schema Modeling Approach
Heap Model

From Nat’s Graph Schema Modeling Approach
Source Control, Issues, Social Coding

- What can you learn from commits about the code
- Class Toxicity, Frequency of Change (Feathers) about the people
- checkin times, collaboration, commit-size, commit-frequency
- Issues
- bug-rich classes (separation of concerns?)
Example: Import Git Commit Logs into Neo4j

- `git log --format` emits CSV
- Graph Model
- LOAD CSV with Cypher
- Create / Update complex Graph Structure

Blog Post
Finally: Some Eye Candy

Isaac & Nash (Software Engineers at Leap Motion)

- Leap Motion Software
- Inheritance Hierarchy, Call Graph
- Render to .dot file
- Use dotparse.js to read it in
- WebGL enabled Three.js rendering
- LeapMotion SDK 2.x beta

Let's have a look: Demo Source
Questions ? Thank You!