PRODYNA

NEO4J IN DER PRAXIS – EIN ERFAHRUNGSBERICHT
ABOUT ME

MAY I INTRODUCE MYSELF

- Darko Križić
- Chief Technology Officer (CTO)
COMMON SCENARIO

- Development of a solution for the customer/the own company
- Business domain model
- Persistence in the database
- Fast queries
- Fast and cost-effective development
- Flexibility/agility
OUR SCENARIO

- Highly semantical data
- Worldwide
  - Many translations
  - Many countries
- Incomplete
  - E.g. some translations don’t exist
- Exceptions
  - Some countries are special
- Very complex business queries
  - Comparison of products by properties
DATABASE OPTIONS

Which database?
- Classical RDBMS
- Very interesting NoSQL alternatives

Which persistence frameworks?
- Depends on the database
- JPA only covers RDBMS
NEO4J – A GRAPH DATABASE

- Open Source
- Community and Enterprise Edition
  - Same level
  - Enterprise Edition has more features (e.g. clustering, online backup)
- Pure Java
- Scalable
  - Embedded
  - Standalone
- Transactional
  - ACID
  - CAP: Consistency, availability
## COMPARISON RDBMS AND NEO4J

<table>
<thead>
<tr>
<th>Topic</th>
<th>RDBMS</th>
<th>Neo4J</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mathematical model</strong></td>
<td>Set theory</td>
<td>Graph theory</td>
</tr>
<tr>
<td><strong>Basic concepts</strong></td>
<td>Table</td>
<td>Node</td>
</tr>
<tr>
<td><strong>Relationship</strong></td>
<td>Implicit (keys)</td>
<td>Explicit (type)</td>
</tr>
<tr>
<td><strong>Query language</strong></td>
<td>SQL</td>
<td>Cypher</td>
</tr>
<tr>
<td><strong>Transactionality</strong></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td><strong>Network access</strong></td>
<td>Proprietary / JDBC API</td>
<td>REST / BOLT / (JDBC)</td>
</tr>
</tbody>
</table>
GRAPH DATABASE BASICS

- Graph database
  - There are only nodes and relationships
- No schema
  - Constrains possible
- Own query language "Cypher"
- REST API
NODES, RELATIONSHIPS AND PROPERTIES

- **Node**: Labels, properties
- **Relationship**: Direction, type, properties
RELATIONSHIPS ARE EXPLICIT

- Graph Models are much closer to the real world
- Graph Models are understandable by the business people
UNLIMITED RELATIONSHIPS BETWEEN NODES

- Nodes can be connected with unlimited number of relationships
- Relationships can be same, or differ in direction, type and/or properties
Neo4j nodes and relations are purely business driven, no need for technical constructs.
NODES CAN HAVE MULTIPLE LABELS

- **Country** Region `<id>: 19 name: Italy id: it`
- **Cluster** Region `<id>: 2 name: Southern Europe id: s-eu`
- **Cluster** Country Region `<id>: 8 name: China id: cn`
NORMALIZED STAYS NORMALIZED

- Query results can be both
  - Denormalized tables (repeating identical columns)
  - Graph (each node once, each relationship once)
- No redundant result entries
- No logic for processing data required

```json
{
  "statements": [  
    {  "statement": "match (r:Region) return r",  
       "resultDataContents": [  
         "row",  
         "graph"  
       ],  
       "includeStats": true  
     }
  ]
}
```
Platzhalter: Dieses Material darf nicht veröffentlicht werden.
NEO4J BASED SOLUTION

SCREENSHOTS

Platzhalter: Dieses Material darf nicht veröffentlicht werden.
SPRING DATA NEO4J

```java
@Query("match(tr:Treatment)-[:USING*]-(p:Product)<-[:SOLD_AS]-(f:Formulation {id:{formulationId}}), " + "(p)<-[:IN_COUNTRY]-(c:Country), " + "(tr)-[:IN_STAGE]-(cgs:CropGrowthStage)-[:ON_CROP]-(cr:Crop)-[:IN_CATEGORY{1..2}]-(cc:CropCategory {level:1}), " + "(tr)-[:ON_TARGET]-(t:Target) " + "optional match(cc)-[:TRANSLATES]-(l:Language)-[:SPOKEN]-(l:Locale {id:{localeCode}}) " + "return distinct " + "f.id, " + "cr.id as id, " + "cc.name as name, " + "null as translation, " + "cc.id as categoryId, " + "cc.name as categoryName, " + "l.translation as categoryTranslation, " + "count(distinct t) as count, " + "count(distinct c) as value"")
Set<ValueTreatmentResultImpl<Integer>> findCropTreatments(
    @Param(FORMULATION_ID) String id,
    @Param(LOCAL_CODE) String localeCode);

@Query("match(f:Formulation {id:{formulationId}})<-[:SOLD_AS]-(p:Product)<-[:USING]-(tr:Treatment) " + "<[:IN_STAGE]>(cgs:CropGrowthStage)-[:ON_CROP]-(cr:Crop)-[:IN_CATEGORY{1..2}]-(cc:CropCategory {level:1}) " + "optional match(cc)-[:TRANSLATES]-(l:Language)-[:SPOKEN]-(l:Locale {id:{localeCode}}) " + "return distinct " + "cc.id as id, " + "cc.name as name, " + "l.translation as translatedName, " + "count(distinct cr) as count")
Set<CategoryCountImpl> findCropCountForFormulation(
    @Param(FORMULATION_ID) String formulationId,
    @Param(LOCAL_CODE) String localeCode);
```
@Rule Neo4jRule launches an Embedded Neo4j
- A static graph.db is copied from a specific location
- Plugins are also part of the test
THE ACTUAL TESTS

@Test
class TestFindCropCountForFormulation{
    public void testFindCropCountForFormulation() {
        Formulation f = formulationService.findFormulationById("9501987");
        List<CategoryCount> results = formulationService.findCropCountForFormulation("de_DE", f);
        CategoryCountImpl ex = new CategoryCountImpl();
        ex.setId("2");
        ex.setName("Cereals");
        ex.setTranslatedName("Getreide");
        ex.setCount(4);
        assertThat(results, Matchers.hasItem(ex));
    }
}

@Test
class TestFindTargetTreatments{
    public void testFindTargetTreatments() {
        Formulation f = formulationService.findFormulationById("9501987");
        List<TreatmentResult<String>> results = formulationService.findTargetTreatments("de_DE", f);
        TreatmentResultImpl ex = new TreatmentResultImpl();
        ex.setId("PUCCRE");
        ex.setName("Brown Leaf Rust Of Cereals");
        ex.setCategoryId("1");
        ex.setCategoryName("Disease");
        ex.setCategoryTranslation("Krankheit");
        TreatmentResultWrapper<String> item = new TreatmentResultWrapper<>(ex, null, Target.class, null);
        assertThat(results, Matchers.hasItem(item));
    }
}
### Graph Model

**Extracting Translations**

```haskell
$ match (cr:Crop)-[:TRANSLOCATES]->(loc:Locale {id:"de_DE"}) return cr.id as cropId, cr.name as cropName,
  t.translation as translation, t.synonyms as synonyms
```

<table>
<thead>
<tr>
<th>cropId</th>
<th>cropName</th>
<th>translation</th>
<th>synonyms</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR SOS</td>
<td>Savoy Cabbage</td>
<td>Wirsing</td>
<td>[Wirsing, Welschkohl, Wirsingkohl]</td>
</tr>
<tr>
<td>BR BON</td>
<td>Chinese Cabbage</td>
<td>Spitzkohl</td>
<td>[Spitzkohl]</td>
</tr>
<tr>
<td>BR SOL</td>
<td>Head Cabbage</td>
<td>Kopfkohl</td>
<td>[Kopfkohl, Weißkohl]</td>
</tr>
<tr>
<td>BR SOR</td>
<td>Red Cabbage</td>
<td>Rotkohl</td>
<td>[Rotkohl, Blaukraut, Rotkraut]</td>
</tr>
<tr>
<td>AT CHI</td>
<td>Summer Kiwi</td>
<td>Strahlengriffel</td>
<td>[Strahlengriffel, Schlingpflaume, Chinesischer Strahlengriffel, Kiwi, Chinesischer Strahlengriffel]</td>
</tr>
<tr>
<td>MOR B O</td>
<td>Aino Mulberry</td>
<td>Japanischer Maulbeerbaum</td>
<td>[Japanischer Maulbeerbaum]</td>
</tr>
<tr>
<td>CY LAV</td>
<td>Common Hazel</td>
<td>Gewöhnliche Hasel</td>
<td>[Gewöhnliche Hasel, Waldhasel, Gewöhnliche Haselnuss]</td>
</tr>
<tr>
<td>IL ORE</td>
<td>Common Walnut</td>
<td>Echter Maulbeerbaum</td>
<td>[Echter Walnußbaum]</td>
</tr>
<tr>
<td>VA CMY</td>
<td>Blackberry</td>
<td>Heidelbeere</td>
<td>[Heidelbeere, Blaubeere, Blackberry, Blackberry]</td>
</tr>
<tr>
<td>VA CMA</td>
<td>Large Cranberry</td>
<td>Kranbeere</td>
<td>[Kranbeere, Kulturbeere, Große Moosbeere]</td>
</tr>
<tr>
<td>RU BID</td>
<td>Red Raspberry</td>
<td>Himbeere</td>
<td>[Himbeere, Rote Beere]</td>
</tr>
<tr>
<td>RI BSS</td>
<td>Ribes spp.</td>
<td>Ribes spp.</td>
<td>[Ribes spp.]</td>
</tr>
<tr>
<td>AE OSS</td>
<td>Asselius sp.</td>
<td>Asselius sp.</td>
<td>[Asselius sp.]</td>
</tr>
<tr>
<td>SY PAL</td>
<td>Common Snowberry</td>
<td>Traubige Schneebeere</td>
<td>[Traubige Schneebeere, Knallerbeere, Schneebeere]</td>
</tr>
</tbody>
</table>

Returned 138 rows in 24 ms.
GRAPH MODEL

FALBACK TO ENGLISH IF NO TRANSLATION
CSV files can be imported easily

- Header names are taken as line names and can be addressed directly
SAMPLE: LOCALE, MAPPING TO LANGUAGE

```plaintext
id, language, countries
pt_BR, pt, br
zh_CN, zh, cn
lt_LT, lt, lt
da_DK, da, dk
sv_SE, sv, se
et_EE, et, ee
lvLV, lv, lv
ru_BY, ru, by
de_AT, de, at
en_NZ, en, nz
en_ZA, en, za
ro_RO, ro, ro
it_IT, it, it
ru_RU, ru, ru
en_CA, en, ca
es_PE, es, pe
ko_KR, ko, kr
hu_HU, hu, hu
uk_UA, uk, ua

// Locale
return "Importing Locale";
create index on :Locale{id};
load csv with headers
from "file:import/locale.csv"
as line
create (:Locale {id: line.id, name: line.id, countries: split(line.countries, ";"), language: line.language});

return "Relating Locale-SPOKEN-Language";
match (:Locale), (lang:Language)
where l.language = lang.id
create (l)-[:SPOKEN]->(lang);

// Remove unneeded attributes
match (loc:Locale)
remove loc.countries, loc.language;
```
EXPERIENCE WITH NEO4J

- Clear separation of business properties and relational information
  - Relations are explicit; in SQL they are implicit

- Very fast, but
  - Always define direction and type of relations (cr:Crop)-[:EXISTS]->(c:Country)

- Integrated fulltext search
FULLTEXT SEARCH
INTEGRATED LUCENE

- Lucene is integrated in Neo4j
  - Used by Neo4j itself for indexes
  - Can be used for own indexes
- Lucene query integrated in Cypher
- Limitations for configuration via Cypher
  - Partly via REST
  - Fully via local API (Plugin!)

- Own indexes must be updated manually
  - Not suitable
- Two exceptions
  - node_auto_index
  - relationship_auto_index
Plugin in Neo4j that configures
- Properties to be indexed
- Enable fulltext index
- Enable lower case
- Enable special character handling (ñ/n, ä/a, ß/ss, ø/o, ...)

```java
public class IndexConfigurationPlugin implements SPPluginLifecycle {
    private static final String NAME_AUTO_INDEX = "node-auto-index";
    public static final String RELATIONSHIP_AUTO_INDEX = "relationship-auto-index";
    private static final String INDEX_TYPE = "type";
    private static final String INDEX_LENGTH = "length";
    private static final String FULLTEXT = "fulltext";
    private static final String INDEX_LOWER_CASE = "to_lower_case";
    private static final String INDEX_SPECIAL_CHARACTERS = "analyser";
    private static final String INDEX_ANALYZER_NAME = "AnalyzerCustomAnalyzer.class.getName()";
    private final String[] nodeProperties = {"id", "name", "enlishName", "enlishSynonyms", "latimName/longName", "latimSynonyms", "productCode/local"};
    private final String[] relationshipProperties = {"relationship", "synonyms"};
    private final Logger log = LoggerFactory.getLogger(IndexClass.class); 

    @Override public Collection<Collection<? extends Indexer>> start(final Neo4jServer neoServer) {
        final Database database = neoServer.getDatabase();
        final RelationshipDatabase relationshipDatabase = database.getRelationshipDatabase();
        final IndexManager indexManager = relationshipDatabase.getIndexManager();
        final Node relationshipNode = relationshipDatabase.createNode();
        final IndexNode relationshipIndex = indexManager.createIndexNode(indexManager.createNodeNodeIndexer(relationshipNode, relationshipIndex));
        final IndexNode nodeIndex = indexManager.createIndexNode(indexManager.createNodeAutoIndexer(relationshipNode, nodeIndex));
        final IndexNode relationshipIndexer = relationshipIndex.getIndexManager().createIndexNode(relationshipIndexer);
        final IndexNode nodeIndexer = indexManager.createIndexNode(nodeIndexer);
        if (nodeIndexer != null) {
            log.info("NodeAutoIndexer is already enabled - good!");
            return Collections.EMPTY_LIST;
        } else {
            log.info("NodeAutoIndexer is NOT enabled - need it is!");
        }
    }

    // ensure all required node properties are enabled
    final Set<String> indexNodeProperties = nodeIndexer.getAutoIndexProperties();
    for (final String nodeProperty : nodeProperty) {
        if (!indexNodeProperties.contains(nodeProperty)) {
            log.info("Node property " + nodeProperty + " not indexed - add it!");
        } else {
            nodeIndexer.setAutoIndexingProperty(nodeProperty);
            log.info("Node property " + nodeProperty + " already indexed - ok!");
        }
    }
}
```
Combination of Lucene fulltext and Cypher

Find "id" starting with "trz" in any node

Filter it out by being a Crop
FULLTEXT SEARCH

USING IT

Find all relationships where property translation contains “hase”

Filter relationship to be between Crop and Locale, Locale in a Country and Country part of a specified region
FULLTEXT SEARCH

*0..2 FOR RUNNING THROUGH TAXONOMIES

```
$ start t=relationship:relationship.auto_index("translation:*bre*") match (c:Crop)-[:TRANSLATES]->
  (loc:locale)-[:SPoken_IN]->(c:Country)-[:PART_OF*0..2]->(r:Region {id:"nan"}) match (c:)[:EXISTS]->()<
return c.id, c.translation, c.id
```

<table>
<thead>
<tr>
<th>id</th>
<th>translation</th>
<th>c.id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Baby's Bleach</td>
<td>us</td>
</tr>
<tr>
<td>2</td>
<td>Pink Beach-Ohawa</td>
<td>us</td>
</tr>
<tr>
<td>3</td>
<td>Umbrella Tree</td>
<td>us</td>
</tr>
<tr>
<td>4</td>
<td>Breadfruit</td>
<td>us</td>
</tr>
</tbody>
</table>

Returned 4 rows in 42 ms.

```
$ start t=relationship:relationship.auto_index("translation:*bre*") match (c:Crop)-[:TRANSLATES]>
  (loc:locale)-[:SPoken_IN]->(c:Country)-[:PART_OF*0..2]->(r:Region {id:"n-eu"}) match (c:)[:EXISTS]->()<
return c.id, c.translation, c.id
```

<table>
<thead>
<tr>
<th>id</th>
<th>translation</th>
<th>c.id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Concordia</td>
<td>bc</td>
</tr>
</tbody>
</table>

Returned 1 row in 61 ms.
FULLTEXT SEARCH

LIMITATIONS OF INTEGRATED LUCENE

- Score not available externally
- Found text not available (markings)
- Promise by Neo Technologies: Will be better in future versions
ISSUES WITH NEO4J

- Missing some powerful tools for import/export of partial graphs
- Database growing (no reorganization)
  - Workarounds possible
  - Will be fixed in future releases
NEUTRAL FACTS

- Query language "Cypher" is propriatery
  - Open sourced as “OpenCypher”
  - Same with most other NoSQL databases

- Clustering possible, but no sharding
  - One single graph
  - Due to nature of graph

- Only one "schema" per instance
  - Will be supported in future releases
  - Workaround: Run multiple instances
ADVANTAGES OF NEO4J

- Easy import of CSV data
- User friendly frontend
- Excellent Cypher query language
- Easy cooperation with business department
- Flexible Graph Model features
- Easy integration
- Good performance
- Very powerful testing possibility (@NeoRule)
SUMMARY

A POWERFUL DATABASE, THAT WE WILL CONSIDER FOR EVERY FUTURE PROJECT
ANY QUESTIONS?

VISIT US AT OUR PRODyna BOOTH
PRODYNA IN YOUR REGION

DARKO KRIŽIĆ (CTO) – PRODYNA FRANKFURT

- PRODYNA AG
- Ludwig-Erhard-Straße 12-14  65760 Eschborn
- T +49 69 597 724 - 175  F +49 69 597 724 - 700  M +49 176 178 70 175
- darko.krizic@prodyna.com  prodyna.com
CONTACT US!

+49 69 597 724 - 0
info@prodyna.com
Facebook/prodynaag
Twitter/prodynaag