Document oriented access to a graph database

Using a Neo4j Server Extension

Michael Hunger - @mesirii

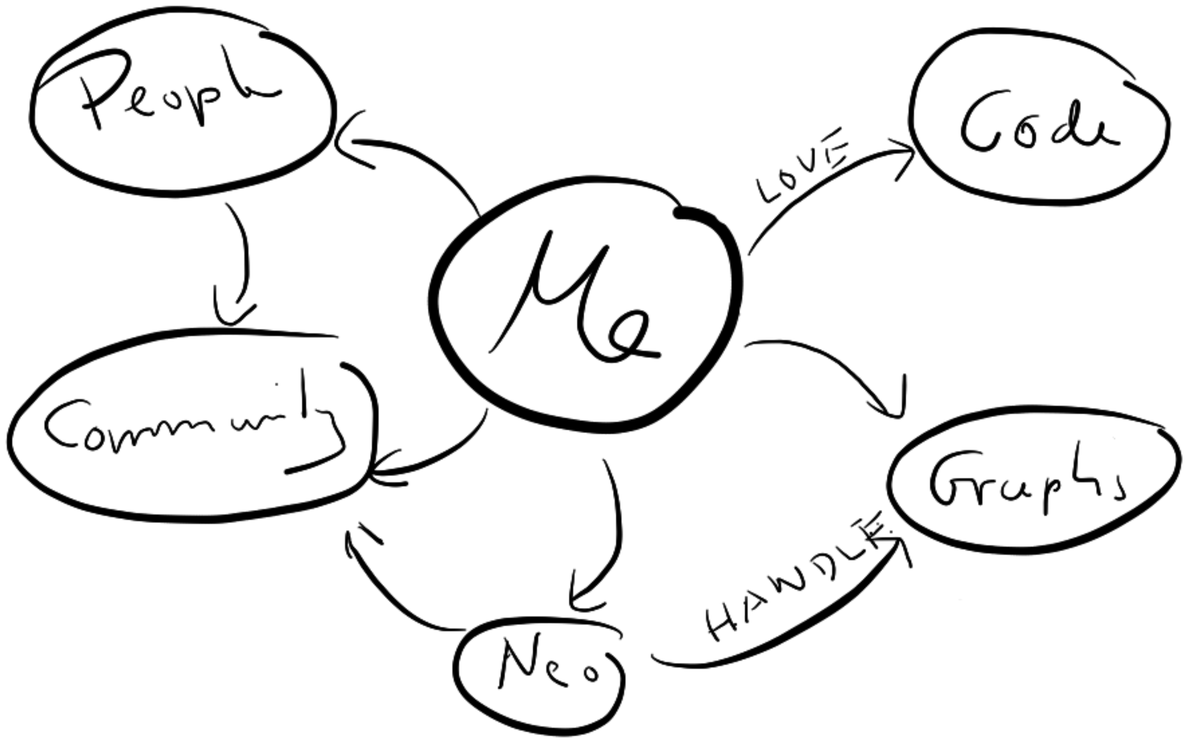
http://github.com/jexp/cypher-rs

Graphs

Graphs are great

- for modeling highly connected domains
- finding patterns to answer questions
- high performance traversals

Graphs are great



Graphs suck

- for compact access patterns
- use-case specific views
- easy integration in modern webapps

Graphs suck

```
$ match (movie:Movie)<-[role]-(person:Person) return movie.title, type(role), person.name
```

CYPHER `match (movie:Movie)<-[role]-(person:Person) return movie.title, type(role), person.name`

movie.title	type(role)	person.name
The Replacements	ACTED_IN	Keanu Reeves
The Matrix	ACTED_IN	Keanu Reeves
The Matrix	ACTED_IN	Carrie-Anne Moss
The Matrix	ACTED_IN	Laurence Fishburne
Cloud Atlas	ACTED_IN	Hugo Weaving
The Matrix	ACTED_IN	Hugo Weaving
Cloud Atlas	DIRECTED	Andy Wachowski
The Matrix	DIRECTED	Andy Wachowski
Cloud Atlas	DIRECTED	Lana Wachowski
The Matrix	DIRECTED	Lana Wachowski
The Matrix	PRODUCED	Joel Silver
Jerry Maguire	ACTED_IN	Tom Cruise

✔ Returned 47 rows in 44 ms

Documents

Documents are great

- for everything that eats JSON
- especially modern JavaScript based webapps
- as dedicated projections of a domain
- compact representation

Documents are great

```
{
  name:"Michael",
  age:39,
  children:[„Selina“,“Rana“,"Selma"],
  address: { street: „Holbeinstr“, city: „Dresden“}
  hobbies: [„Coding“,„Buchbar“,“MUD MorgenGrauen“]
  blog: „http://jexp.de/blog“
}
```

Documents suck

- for different projections besides the main use-case
- for connected data
- for cross-document linking
- for less redundancy / duplication

Documents suck

```
{
  name:"Michael",
  age:39,
  children:[„Selina“,“Rana“,"Selma"],
  address: { street: „Holbeinstr“, city: „Dresden“}
  hobbies: [„Coding“,„Buchbar“,“MUD MorgenGrauen“]
  blog: „http://jexp.de/blog“
}
```

„Show people by hobby“

Documents suck

```
{ _id: 1,
  name:“Michael“,
  works_at: dbref:2
}
```

```
{ _id: 2,
  name:“Neo Technology“,
  employees: [dbref:1]
}
```

„Link Employees to Companies“

What can we do?

- Use a **document oriented access layer** for a graph database
- Use graph queries whose results **project into documents**
- Make them **easily accessible**

How can we do that?

- Use Cypher in **Neo4j 2.0**
  - Support for literal maps
  - Parameters, including maps

Cypher Maps

Cypher & Maps - Results

```
MATCH (n:User)

WHERE n.name={name}

RETURN n

// returns a map of the node's properties

RETURN {id: id(n), labels: labels(n), data : n}
```

Cypher & Maps - Results (II)

```
MATCH (n:User)-[:PARENT_OF]->(child)

WHERE n.name={name}

RETURN {name:n.name, children:collect(child.name)}
```

Cypher & Maps - Params

```
CREATE (u:User { name: {p}.name, age: {p}.age})

RETURN u

CREATE (u:User {params})

RETURN u
```

Some Questions

How can we do that?

- Create a **RESTful API** to
  - Provide endpoints that represent on Cypher queries
  - Use **GET, POST** with parameters to execute those endpoints
  - Return only **JSON** documents or lists of those

How do we implement it?

- Add a **Neo4j-Server-Extension**
  - Jersey JAX-RS
  - Save queries in **graph properties**
  - Execute queries for **reads on GET**
  - Execute queries for **reads and writes on POST**
  - Support **JSON** and **CSV** as payload
  - Batch transactional write operations

What does it look like?

What does it look like?

- Examples for
  - Managing endpoints
  - Querying Endpoints
  - Writing to Endpoints

Create Endpoint

PUT /cypher-rs/**users**

Body: **match (n:User) where n.name={name} return n**

--> 201 Location: /cypher-rs/users

PUT /cypher-rs/**create-user**

Body: **create (n:Node {name:{name},age:{age}})**

--> 201 Location: /cypher-rs/create-user

GET, POST - Reads

GET Request

```
GET /cypher-rs/users?name=Andres

--> 200

{"name":"Andres",

 "age":21,"children":["Cypher","L.","N."]}

GET /cypher-rs/users?name=NotExists

--> 204
```

POST Request - Read

```
POST /cypher-rs/users

Content-type: application/json

Body: {"name":"Andres"}

--> 200

{"name":"Andres","age":21,

 "children":["Cypher","L.","N."]}
```

POST Request - Read (II)

POST /cypher-rs/**users**

Content-type: application/json

Body: **[{"name":"Andres"},{"name":"Peter"},{"name":"NotExists"}]**

--> 200

```
[{"name":"Andres","age":21,"children":["Cypher","L.","N."]},
 {"name":"Peter" ,“age“:32,"children":["Neo4j","O.","K."]},
 null]
```

POST - Write

POST Request - Write JSON

```
POST /cypher-rs/create-user

Content-type: application/json

Body: {"name":"Andres","age":21}

--> 201
```

POST Request - Write JSON (II)

```
POST /cypher-rs/create-user

Content-type: application/json

Body: [{"name":"Andres","age":21},

       {"name":"Peter", "age":40},

--> 201
```

POST Request - Write CSV (III)

```
POST /cypher-rs/create-user[?batch=20000&delim=,]

Content-type: text/plain

Body: name,age\nAndres,21\nPeter,40

--> 201
```

More Management

Delete Endpoint

```
DELETE /cypher-rs/users

--> 200
```

List Endpoints

```
GET /cypher-rs

--> 200

[„create-users“, „users“]
```

Inspect Endpoint

```
GET /cypher-rs/users/query

--> 200

match (n:User) where n.name={name} return n
```

Results

Possible Results

single column, single row

```
{"name":"Andres","age":21,"children":["Cypher","L.","N."]}
```

single column, multiple rows

```
[
 {"name":"Andres","age":21,"children":["Cypher","L.","N."]},
 {"name":"Peter", "age":40,"children":["Neo4j","O.","K."]}
]
```

Possible Results (II)

multiple columns, single row

```
{"user": "Andres", "friends": ["Peter","Michael"]}
```

multiple columns, multiple rows

```
[
  {"user": "Andres", "friends": ["Peter","Michael"]},
  {"user": "Michael","friends": ["Peter","Andres"]},
  {"user": "Peter",  "friends": ["Andres","Michael"]}
]
```

Implementation

Implementation - Init

```
@Path("/")
public class CypherRsService {

    private final ExecutionEngine engine;
    private final GraphDatabaseAPI db;
    private final GraphProperties props;

    public CypherRsService(@Context CypherExecutor executor, @Context Database database) {
        engine = executor.getExecutionEngine();
        db = database.getGraph();
        props = db.getDependencyResolver().resolveDependency(NodeManager.class).getGraphProperties();
    }
```

Implementation - PUT

```
@PUT
@Path("/{key}")
@Consumes(MediaType.TEXT_PLAIN)
public Response createEndpoint(@PathParam("key") String key, String body, @Context UriInfo uriInfo) {
    try (Transaction tx = db.beginTx()) {
        props.setProperty(key, body);
        tx.success();
        return Response.created(uriInfo.getRequestUri()).build();
    }
}
```

Implementation - GET

```
@GET
@Path("/{key}")
@Produces(MediaType.APPLICATION_JSON)
public Response readEndpoint(@PathParam("key") String key, @Context UriInfo uriInfo) {
    try (Transaction tx = db.beginTx()) {
        if (props.hasProperty(key)) {
            String query = (String) props.getProperty(key);
            if (Utils.isWriteQuery(query)) return Response.status(Response.Status.NOT_ACCEPTABLE).build();
            Map<String, Object> params = Utils.toParams(uriInfo.getQueryParameters());
            ExecutionResult result = engine.execute(query, params);
            String json = Utils.toJson(result);
            tx.success();
            return Response.ok(json).build();
        }
    } catch(Exception e) {
        e.printStackTrace();
        return Response.serverError().entity(e.getMessage()).build();
    }
    return notFound();
}
```

Usage

Build with `mvn clean install dependency:copy-dependencies`

Copy files

```
cypher-rs-2.0-SNAPSHOT.jar opencsv-2.3.jar to path/to/server/plugins
```

Add this line to `path/to/server/conf/neo4j-server.properties`

```
org.neo4j.server.thirdparty_jaxrs_classes=org.neo4j.cypher_rs=/cypher-rs
```

Restart

Next Steps

- Add streaming
- Rule the world