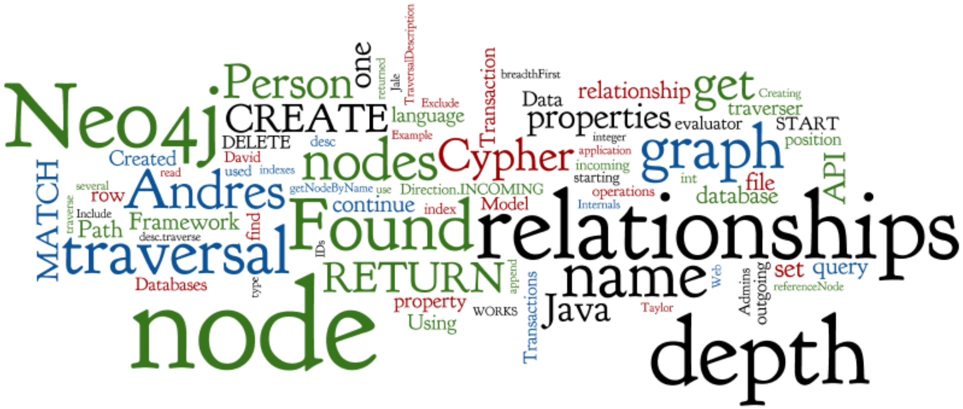
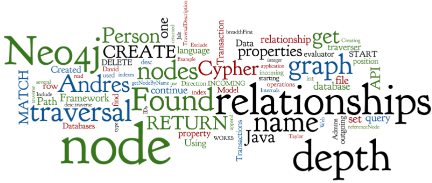


Neo4j Graph Database

Seminar 5 of *NoSQL Databases* (PA195)

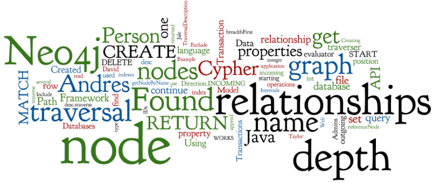
David Novak & Vlastislav Dohnal
Faculty of Informatics, Masaryk University, Brno

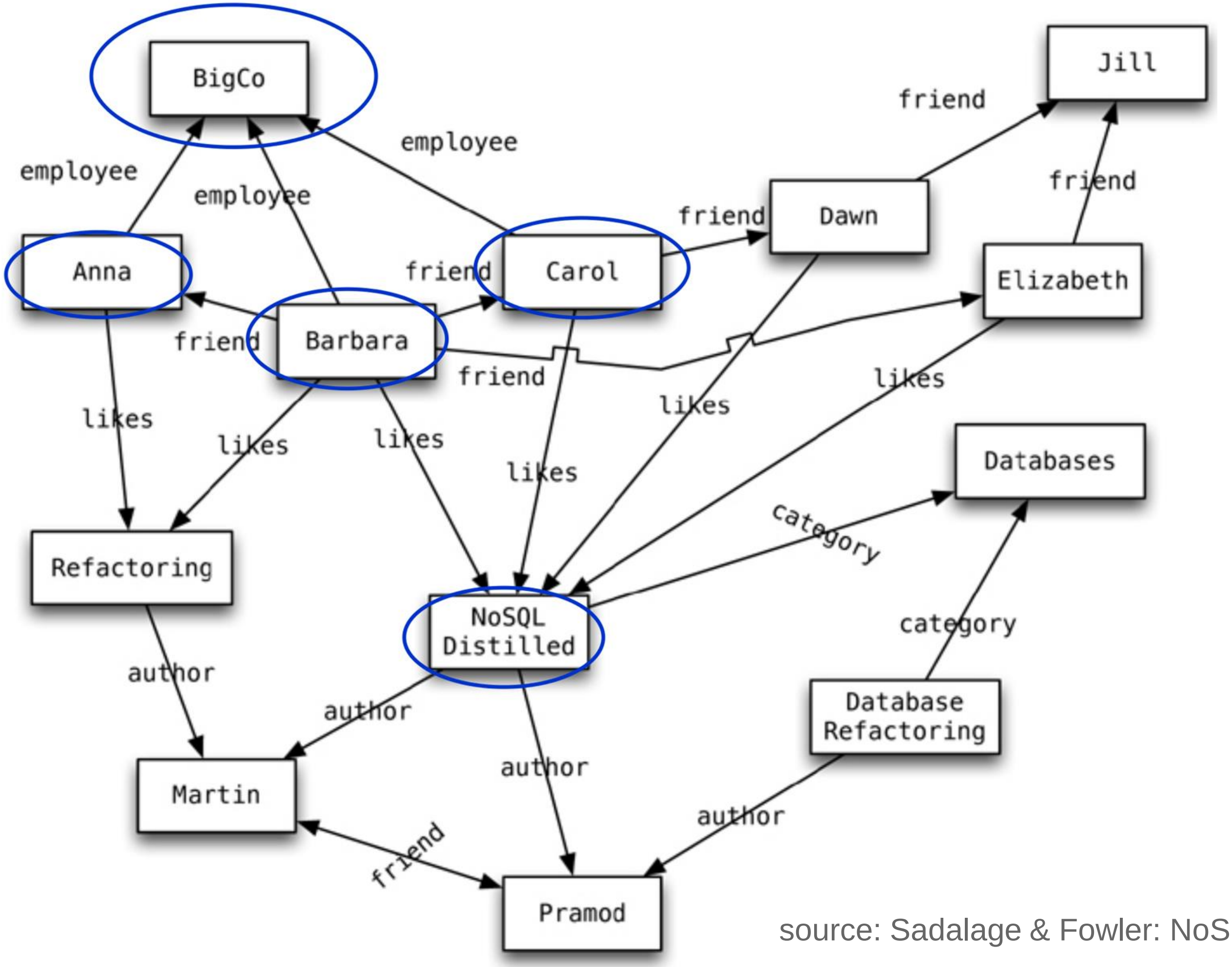
Agenda



- Graph Databases
- Neo4j
  - Basic information
  - Data model
  - Cypher query language
    - Structure and examples
    - Other interfaces: Experience with Web UI
  - Java API (embedded database)
  - Traversal of the graph
    - Traversal framework
    - Examples

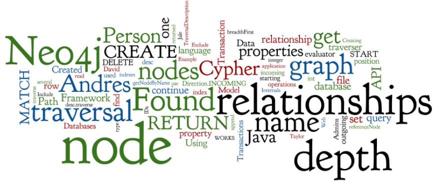
Graph Databases: Example





source: Sadalage & Fowler: NoSQL Distilled, 2012

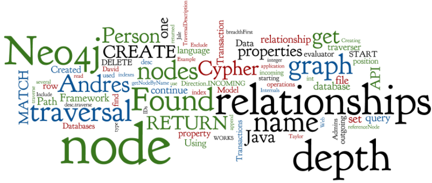
Graph Databases: Mission



- To store entities and relationships between them
  - Nodes are instances of objects
  - Nodes have properties, e.g., name
  - Edges have directional significance
  - Edges have types e.g., likes, friend, …

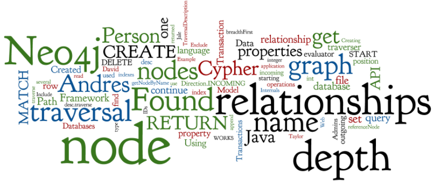
- Nodes are organized by relationships
  - Allows finding interesting patterns
  - **Example:** Get all nodes that are “employee” of “Big Company” and that “likes” “NoSQL Distilled”

Graph Databases: Representatives

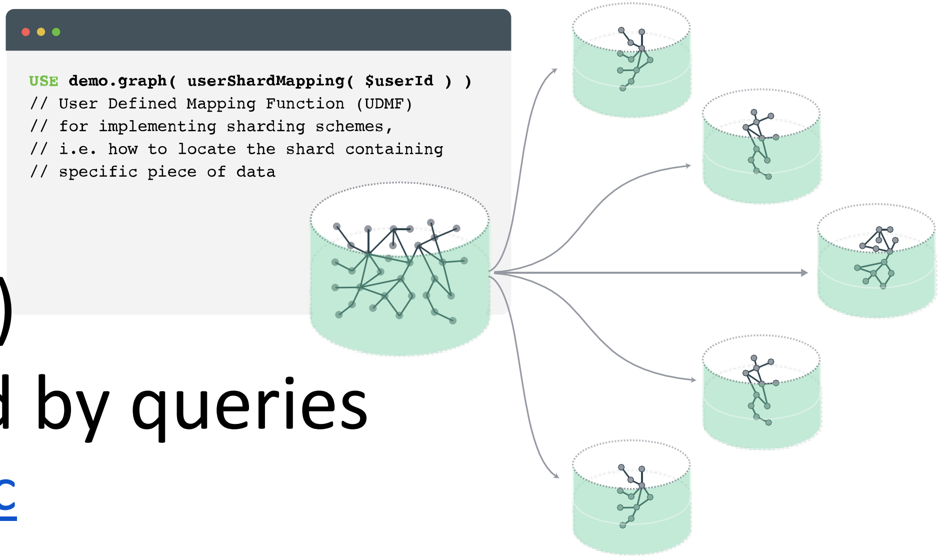


Ranked list: http://db-engines.com/en/ranking/graph+dbms

Neo4j: Basic Info



- Open source graph database
- Initial release: 2007
  - Current version 4.2
- Written in: Java
- OS: cross-platform
- Full transactions (ACID)
- Partitioning: supported by queries
  - since 4.0, by Neo4j Fabric
- Replication: Master-slave
  - Eventual consistency

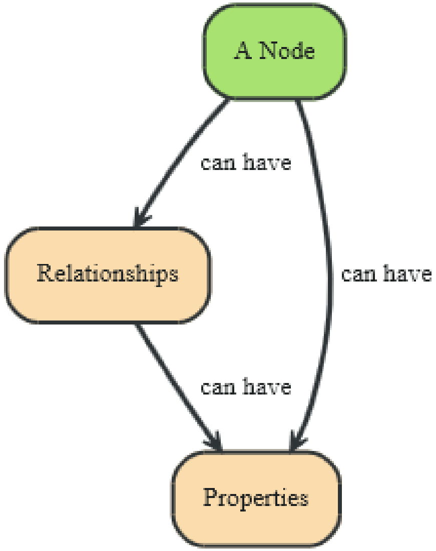


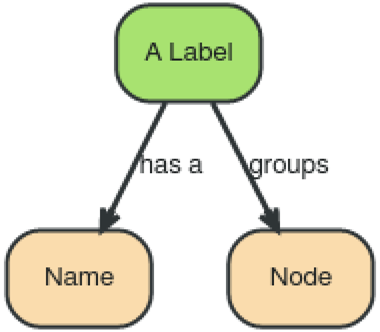
```
USE demo.graph( userShardMapping( $userId ) )
// User Defined Mapping Function (UDMF)
// for implementing sharding schemes,
// i.e. how to locate the shard containing
// specific piece of data
```

Source: neo4j.com

Data Model: Nodes

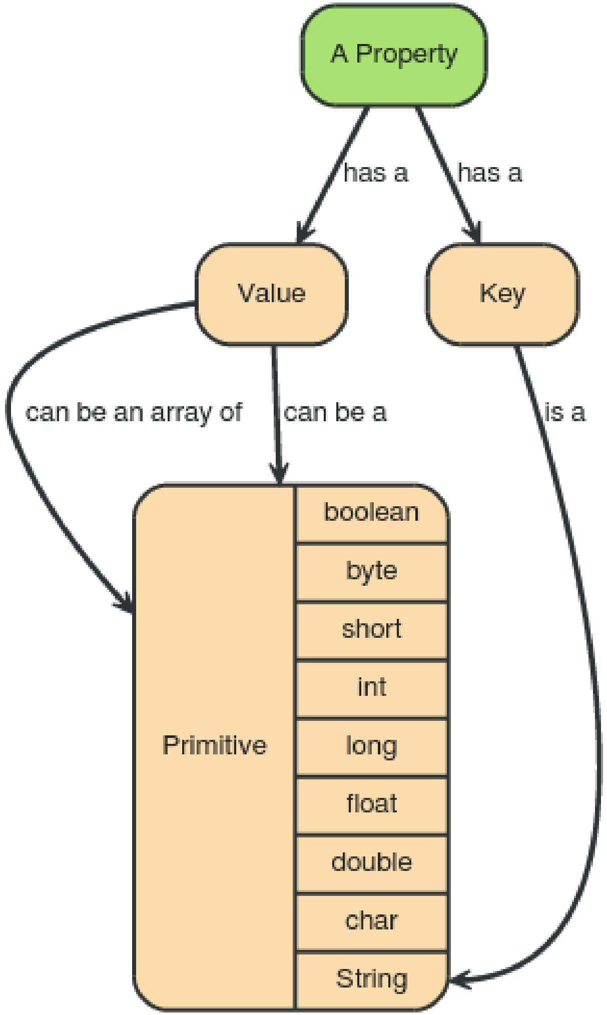
- Fundamental unit: node
- Nodes have properties
  - Key-value pairs
  - null is not a valid property value
    - nulls can be modelled by the absence of a key
- Nodes have labels
  - labels typically express "type of node"





http://db-engines.com/en/system/Neo4j

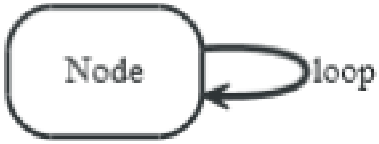
Data Model: Properties

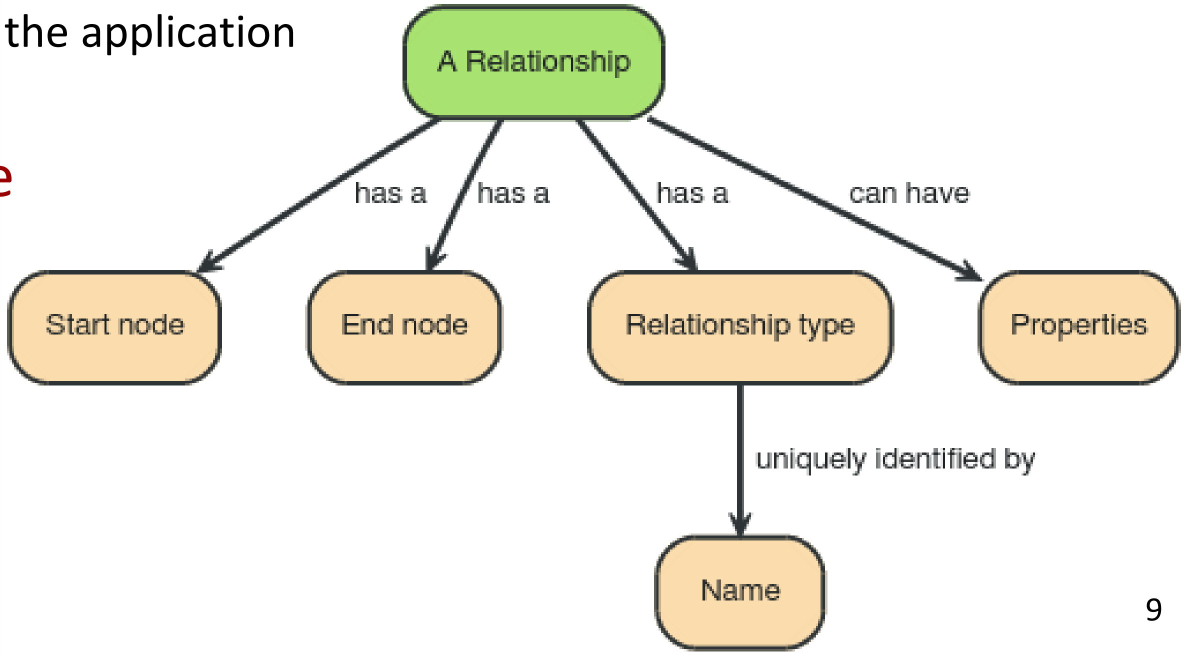


Type	Description
boolean	true/false
byte	8-bit integer
short	16-bit integer
int	32-bit integer
long	64-bit integer
float	32-bit IEEE 754 floating-point number
double	64-bit IEEE 754 floating-point number
char	16-bit unsigned integers representing Unicode characters
String	sequence of Unicode characters

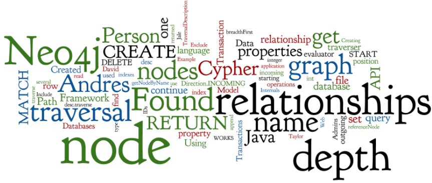
Data Model: Relationships

- Directed relationships (edges)
  - Incoming and outgoing edge
    - Equally efficient traversal in both directions
    - Direction can be ignored
      if not needed by the application
  - Always a start
    and an end node
    - Can be recursive





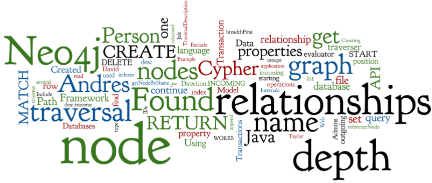
Use of Neo4j



- Two ways to use Neo4j:
  - Embedded: Used directly within a Java application
  - Self-standing server + connections

- Various types of connections
  - Neo4j Cypher Shell
  - HTTP API
    - uses Cypher query language
  - Web GUI
    - using Cypher query language
  - Standard Java API
  - Gremlin graph traversal language (plugin), etc.

Neo4j in Server mode



- Virtual machine http://stratus.fi.muni.cz

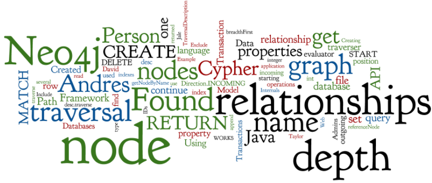
stratus

  - ○ Template “PA195 - Neo4j”, ID 255

```
$ ssh root@... -L 7474:localhost:7474
               -L 7687:localhost:7687

# neo4j-community-3.1.4/bin/neo4j start
```

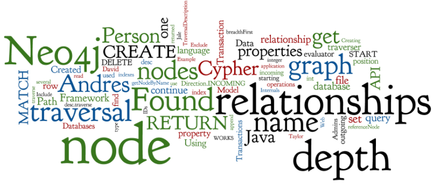
- or Install on your own:
  - download from https://neo4j.com/ to `/var/tmp`
  - `tar xvzf neo4j-community-*.tar.gz`
    - `module add jdk`
  - `./bin/neo4j start`

Neo4j Command-line Querying



- Cypher shell
  - `./bin/cypher-shell`
  - can also be installed separately, but shipped with the server

HTTP API

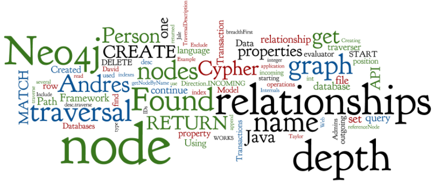


- Query/update operations using HTTP protocol
  - GET, POST methods
  - data sent/received in JSON
- Fully transactional in the latest version
- Example: create node with “name” property

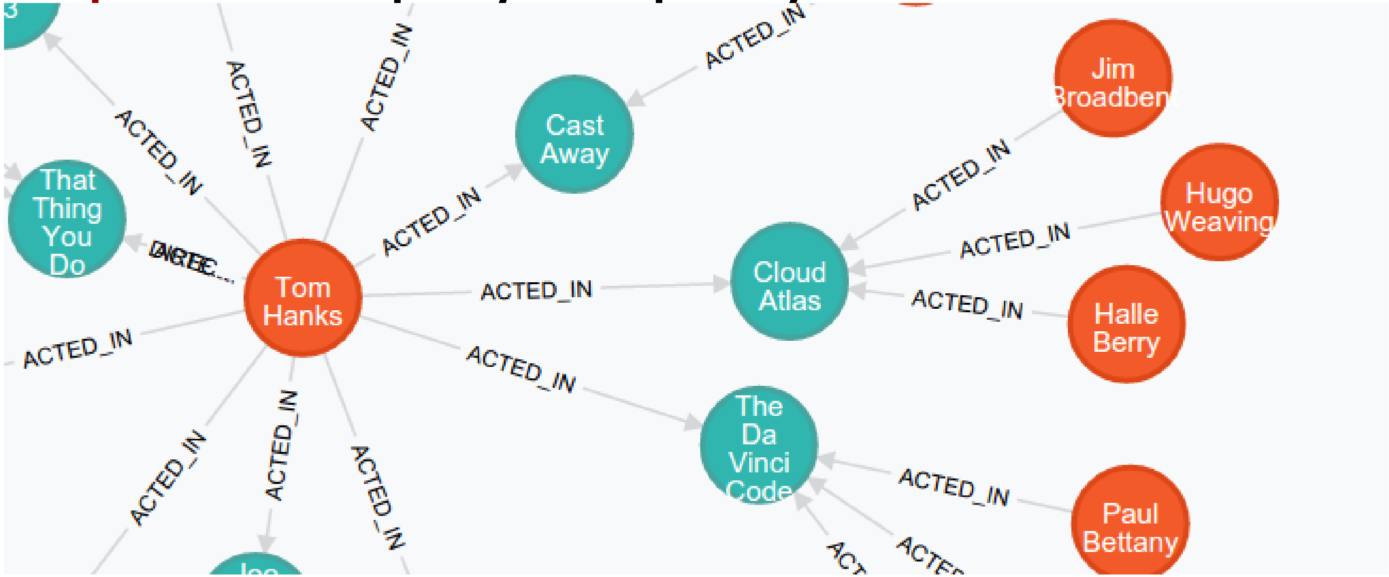
```
curl -i -X POST http://localhost:7474/db/neo4j/tx/commit
-H "Content-Type: application/json; charset=UTF-8" --user
"neo4j" -d '{ "statements": [ { "statement": "CREATE (n
$props) RETURN n", "parameters": { "props": { "name":
"John Doe" } } } ] }'
```

https://neo4j.com/docs/http-api/current/

Neo4j Web Interface



- By default, running on http://localhost:7474/
  - default credentials: neo4j/neo4j
- Online interpreter of Cypher
- Graphical display of query results



Cypher Language

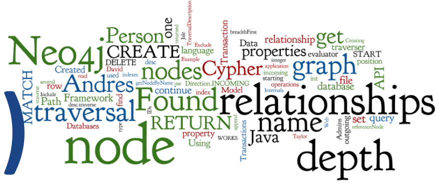
- Neo4j graph query language
    - For querying and updating
- Declarative – we say what we want
    - Not how to get it
    - Not necessary to express traversals
- Human-readable
- Inspired by SQL and SPARQL
- Still growing = syntax changes are often

http://neo4j.com/docs/stable/cypher-query-lang.html

Cypher: Clauses

- **MATCH**: The graph pattern to match
- **WHERE**: Filtering criteria
- **RETURN**: What to return
- ~~**START**: Starting points in the graph~~
  - ~~by explicit index lookups or by node IDs (both deprecated)~~
- **WITH**: Divides a query into multiple parts
- **CREATE**: Creates nodes and relationships.
- **DELETE**: Remove nodes, relationships, properties
- **SET**: Set values to properties

Cypher: Creating Nodes (Examples)



CREATE (n);

*(create a node, assign to var **n**)*

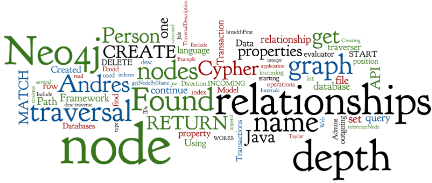
```
Created 1 node, returned 0 rows
```

CREATE (a: Person {name : 'Jan'}) **RETURN** a;

(create a node with label ‘Person’ and ‘name’ property Jan’)

```
Created 1 node, set 1 property, returned
1 row
```

Cypher: Creating Relationships



```
MATCH (a {name:'John'}), (b {name:'Jack'})
CREATE (a)-[r:FRIEND]->(b)
RETURN r ;
```

(create a relation FRIEND *between John and Jack)*

```
Created 1 relationship, returned 1 row
```

~~**START** a=node(1), b=node(2)~~

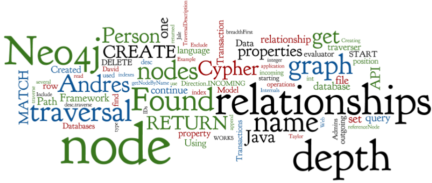
~~**CREATE** (a)-[r:RELTYPE {name : a.name + '->' + b.name }]->(b)~~

~~**RETURN** r~~

~~*(set property 'name' of the relationship)*~~

~~Created 1 node, set 1 property, returned 1 row~~

Cypher: Creating Paths

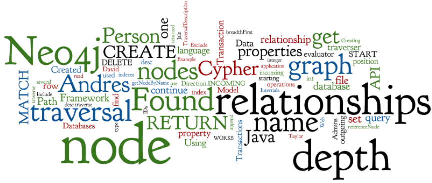


```
CREATE p = (andres: Person {name: 'Andres'})
-[:WORKS_AT]->
(neo)
<-[:WORKS_AT]-
(michael: Person {name:'Michael'})
RETURN p ;
```

To create just a relationship, use MATCH and WHERE

(all parts of the pattern are created)

```
P [Node[4]{name:"Andres"},:WORKS_AT[2]
{},Node[5]{},:WORKS_AT[3] {},Node[6]{name:"Michael"}]
1 row
Nodes created: 3
Relationships created: 2
Properties set: 2
```



Cypher: Changing Properties

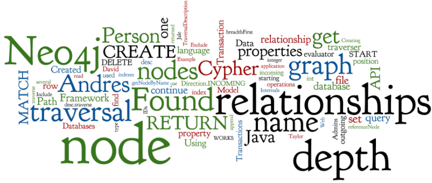
MATCH (n: Person {name: 'Andres'})
SET n.surname = 'Taylor'
RETURN n

(find a node with name ‘Andres’ and set it surname ‘Taylor’)

```
n
Node[0]{name:"Andres",surname:"Taylor"}
1 row
Properties set: 1
```

Task 1: Update Queries

Attendance Check



Modify the nodes so that these queries return something:

```
MATCH (p: Person)
WHERE p.age > 18 AND p.age < 30
RETURN p.name
```

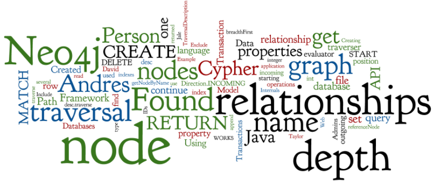
(return names of all adult people under 30)

```
MATCH (user: Person {name: 'Andres'})-[:FRIEND]->(follower)
RETURN user.name, follower.name
```

(find all ‘friends’ of 'Andres')

Copy & paste the queries and their responses to the file **`task1.txt`** and upload to the IS’s vault.

Cypher: Queries (2)



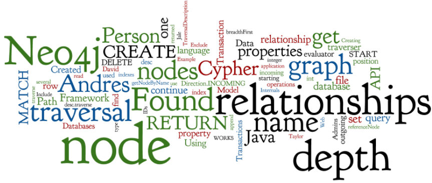
```
MATCH (andres: Person {name: 'Andres'})-[*1..3]-(node)
RETURN andres, node ;
```

(find all ‘nodes’ within three hops from ‘Andres’)

```
MATCH p=shortestPath(
  (andres:Person {name: 'Andres'})-[*]-(david {name:'David'})
)
RETURN p ;
```

(find the shortest connection between ‘Andres’ and ‘David’)

Cypher: Delete



```
MATCH (n: Person {name: 'Andres'})
DELETE n
```

(delete all Persons with name 'Andres')

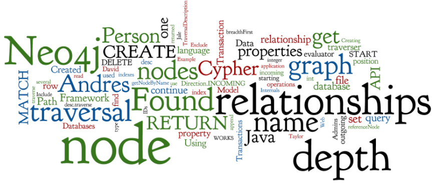
```
Cannot delete node<3>, because it still has relationships.
```

```
MATCH (n: Person {name: 'Andres'}), ((n)-[r]-())
DELETE r,n
```

(first, we must delete all relationships of node with name 'Andres')

```
Nodes deleted: 1
Relationships deleted: 1
```

Task 2: Movies Database



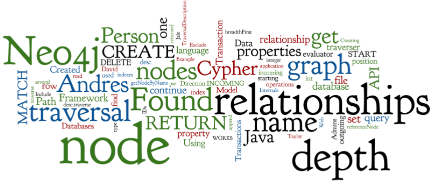
- Go over the “Movies” demo prepared by Neo4j
  - localhost
  - Download the data from the course page (movies-insert.cypher)
  - Copy the file to Stratus VM
  - stratus
  - Import by cypher shell

```
neo4j-community-3.1.4/bin/cypher-shell -u neo4j -p test
<movies-insert.cypher
```

```
Added 171 nodes, Created 253 relationships, Set 564
properties, Added 171 labels
```

Task 2: Query Movies

Attendance Check



- Find all actors who played in a movie with Keanu Reeves.
- Find all directors of movies where acted Tom Hanks.
- Find the oldest director
  - It ain’t “Max von Sydow”
- Print distinct first names of all persons

Copy & paste the queries and their responses to the file **`task2.txt`** and upload to the IS’s vault! *In case you fail to form any query, make a notice there!*

Neo4j as Embedded Database

CONTINUE INDIVIDUALLY…

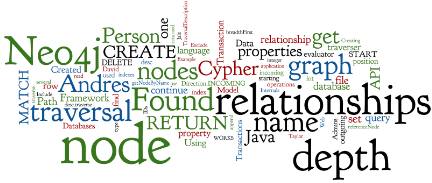
- either use .jar packages from the distribution
- …or download packages from Maven repository
  - package `org.neo4j:neo4j:3.0.0`
    - dependencies automatically loaded
  - newest versions available in repository http://repo.maven.apache.org/maven2/

localhost

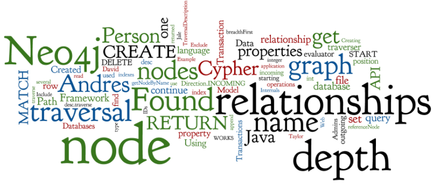
- …or download project from the course web

```
$ unzip neo4j-excercise.zip
$ module add idea-2019.2
$ idea.sh
```

Neo4j: “Hello World” – Java API

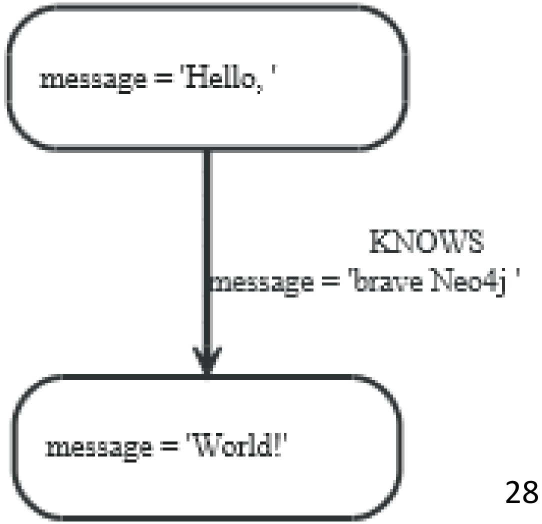


```
String PATH="some_directory";
GraphDatabaseService graphDb;

// starting a database
graphDb = new GraphDatabaseFactory().newEmbeddedDatabase(new
File(PATH));

Node firstNode, secondNode;
Relationship relationship;
```

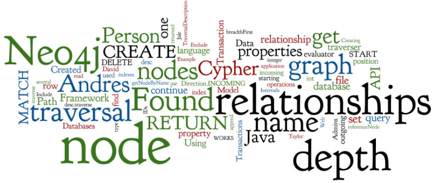


Neo4j: “Hello World” (2)

```
// create a small graph:
firstNode = graphDb.createNode();
firstNode.setProperty( "message", "Hello, " );
secondNode = graphDb.createNode();
secondNode.setProperty( "message", "World!" );

relationship = firstNode.createRelationshipTo
       (secondNode,
     RelationshipType.withName("KNOWS"));

relationship.setProperty
       ("message", "brave Neo4j ");
```



Neo4j: Transactions

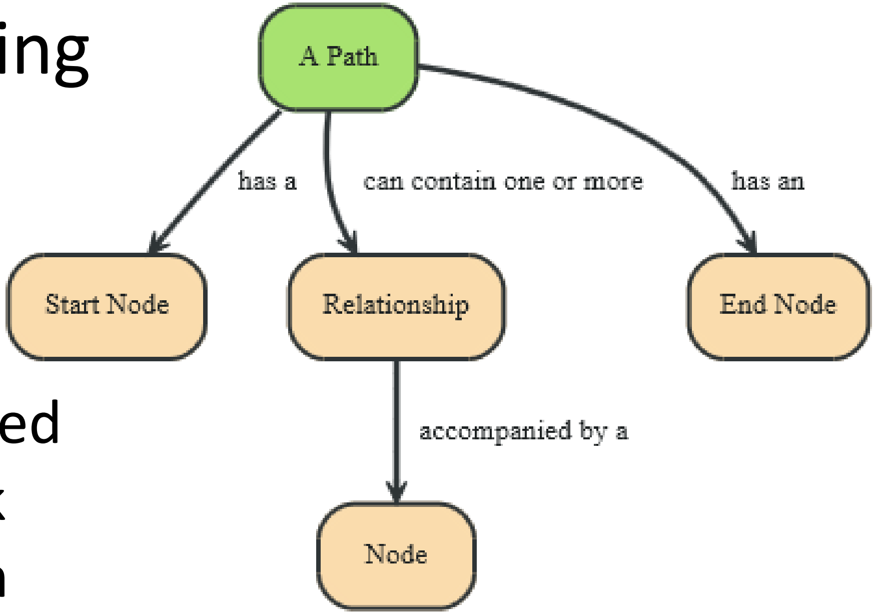


```
// all writes (creating, deleting and updating any data)
// have to be performed in a transaction:
try (Transaction tx = graphDb.beginTx()) {

       (…)

   // print the result:
   System.out.print(firstNode.getProperty("message"));
   System.out.print(relationship.getProperty("message"));
   System.out.println(secondNode.getProperty("message"));


       // transaction operations
   tx.success();
}
```

Data Model: Path & Traversal

- Path = specific nodes + connecting relationships
  - Path can be a result of a query or a traversal

- Traversing a graph = visiting its nodes, following relationships according to some rules
  - Typically, a subgraph is visited
  - Neo4j: Traversal framework in Java API, Cypher, Gremlin


A Path
has a
can contain one or more
has an
Start Node
Relationship
End Node
accompanied by a
Node

Traversal Framework

- A traversal is influenced by
  - Starting node(s) where the traversal will begin
  - Expanders – define what to traverse
    - i.e., relationship direction and type
  - Order – depth-first / breadth-first
  - Uniqueness – visit nodes (relationships, paths) only once
  - Evaluator – what to return

    and whether to stop or continue beyond current position

Traversal = TraversalDescription + starting node(s)

Traversal Framework – Java API

- `org.neo4j...TraversalDescription`
  - The main interface for defining traversals
    - Can specify branch ordering `breadthFirst()` / `depthFirst()`

- `.relationships()`
  - Specify the relationship types to traverse
    - e.g., traverse only edge types: FRIEND, RELATIVE
    - Empty (default) = traverse all relationships
  - Can also specify direction
    - `Direction.BOTH`
    - `Direction.INCOMING`
    - `Direction.OUTGOING`

Traversal Framework – Java API (2)

- `org.neo4j...Evaluator`
  - Used for deciding at each node: should the traversal continue, and should the node be included in the result
    - `INCLUDE_AND_CONTINUE`: Include this node in the result and continue the traversal
    - `INCLUDE_AND_PRUNE`: Include this node, do not continue traversal
    - `EXCLUDE_AND_CONTINUE`: Exclude this node, but continue traversal
    - `EXCLUDE_AND_PRUNE`: Exclude this node and do not continue

  - Pre-defined evaluators:
    - `Evaluators.toDepth(int depth)` / `Evaluators.fromDepth(int depth)`,
    - `Evaluators.excludeStartPosition()`
    - …

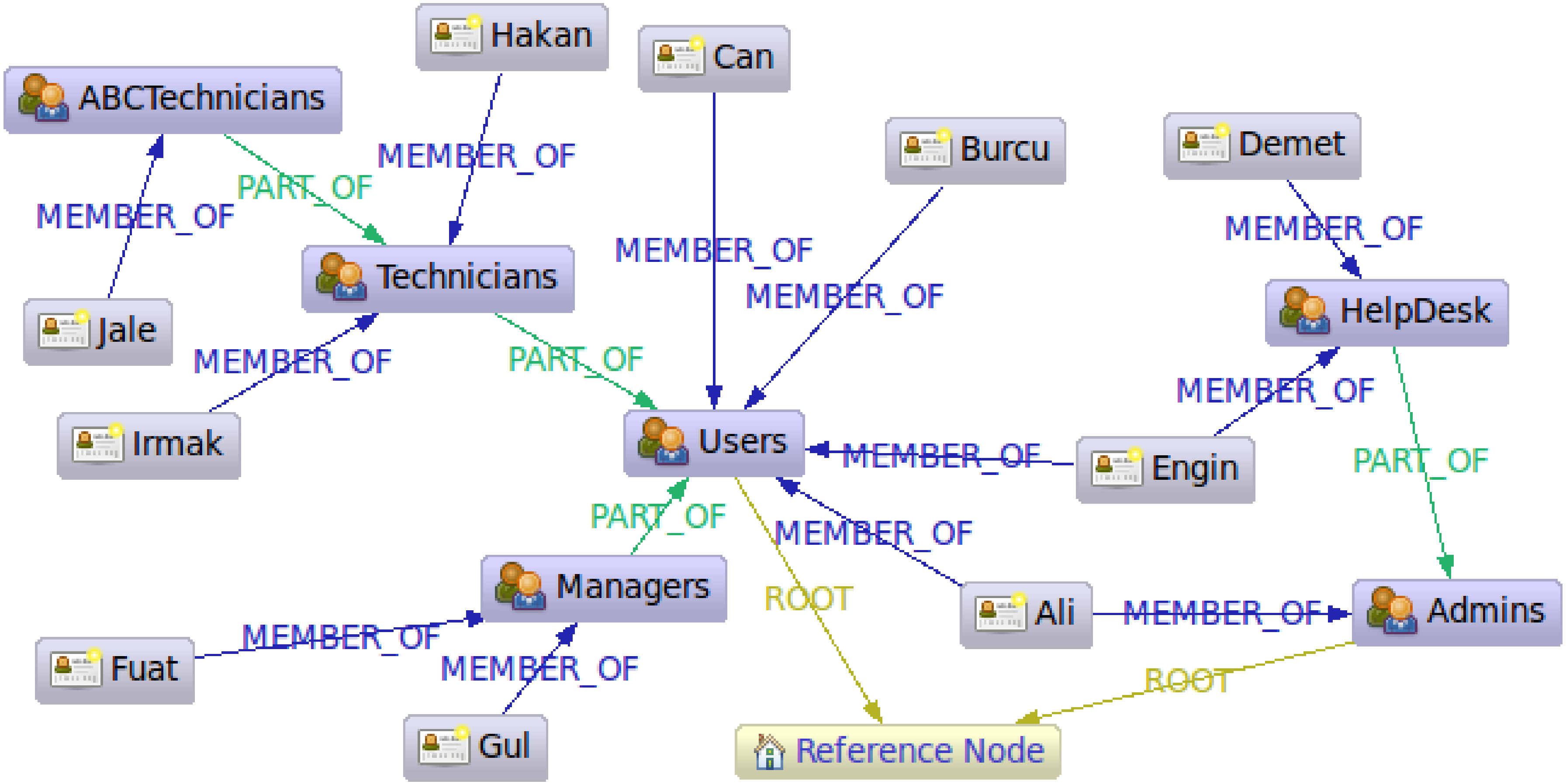
Traversal Framework – Java API (3)

- `org.neo4j...Uniqueness`
  - Indicates under what circumstances a traversal may revisit the same position in the graph

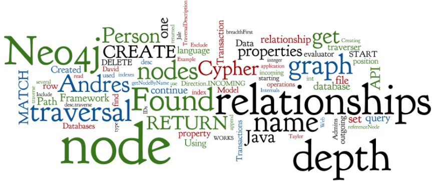
- `Traverser`
  - Starts actual traversal given a TraversalDescription and starting node(s)
  - Returns an iterator over “steps” in the traversal
    - Steps can be: Path (default), Node, Relationship
  - The graph is actually traversed “lazily” (on request)

Sample Data

graph
nodes
vertices
relationships
Adjacency
node
database
Matrix
Example
data
Partitioning
connected
Neo4j
edges
Bandwidth


Hakan
Can
ABCTechnicians
MEMBER_OF
PART_OF
Burcu
Demet
MEMBER_OF
Technicians
MEMBER_OF
MEMBER_OF
MEMBER_OF
Jale
HelpDesk
MEMBER_OF
PART_OF
Irmak
Users
MEMBER_OF
MEMBER_OF
Engin
PART_OF
PART_OF
MEMBER_OF
Managers
ROOT
Ali
MEMBER_OF
Admins
MEMBER_OF
Fuat
MEMBER_OF
ROOT
Gul
Reference Node

Query: Find All "Admins"



```
Node admins = getNodeByName( "Admins" );
TraversalDescription desc = graphDb.traversalDescription()
    .breadthFirst()
    .evaluator( Evaluators.excludeStartPosition() )
    .relationships(RoleRels.PART_OF, Direction.INCOMING)
    .relationships(RoleRels.MEMBER_OF, Direction.INCOMING);
Traverser traverser = desc.traverse(admins);
StringBuilder output = new StringBuilder();

for ( Node node : traverser.nodes() ) {
     output.append("Found: ")
           .append(node.getProperty(NAME))
           .append(" at depth: ")
           .append(path.length()).append("\n");
}
```

```
Found: HelpDesk at depth: 1
Found: Ali at depth: 1
Found: Engin at depth: 2
Found: Demet at depth: 2
```

Query: Get Group Membership of a User

```
Node jale = getNodeByName( "Jale" );
desc = graphDb.traversalDescription()
    .depthFirst()
    .evaluator( Evaluators.excludeStartPosition() )
    .relationships(RoleRels.MEMBER_OF, Direction.OUTGOING)
    .relationships(RoleRels.PART_OF, Direction.OUTGOING);

traverser = traversalDescription.traverse( jale );
```

```
Found: ABCTechnicians at depth: 1
Found: Technicians at depth: 2
Found: Users at depth: 3
```

Query: Get All Groups

```
Node referenceNode = getNodeByName( "Reference_Node" ) ;
desc = graphDb.traversalDescription()
     .breadthFirst()
     .evaluator( Evaluators.excludeStartPosition() )
     .relationships(RoleRels.ROOT, Direction.INCOMING )
     .relationships(RoleRels.PART_OF, Direction.INCOMING);

traverser = desc.traverse( referenceNode );
```

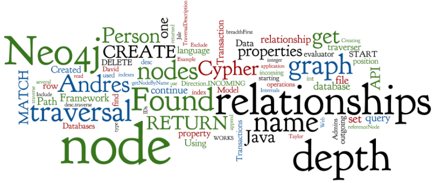
```
Found: Admins at depth: 1
Found: Users at depth: 1
Found: HelpDesk at depth: 2
Found: Managers at depth: 2
Found: Technicians at depth: 2
Found: ABCTechnicians at depth: 3
```

Query: Get All Members in the Database

```
Node referenceNode = getNodeByName( "Reference_Node" ) ;
desc = graphDb.traversalDescription()
  .breadthFirst()
  .evaluator(Evaluators.includeWhereLastRelationshipTypeIs
             (RoleRels.MEMBER_OF ));

traverser =
desc.traverse( referenceNode );
```

```
Found: Ali at depth: 2
Found: Engin at depth: 2
Found: Burcu at depth: 2
Found: Can at depth: 2
Found: Demet at depth: 3
Found: Gul at depth: 3
Found: Fuat at depth: 3
Found: Hakan at depth: 3
Found: Irmak at depth: 3
Found: Jale at depth: 4
```

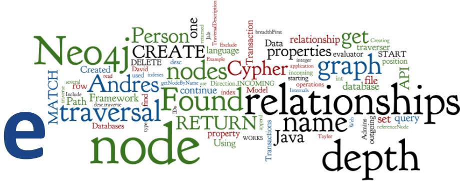
Access to Nodes



- How to get to the starting node(s) before traversal
  1. Using internal identifiers (unique generated IDs)
     - not recommended because Neo4j does reuse freed IDs
  2. Using specified properties
     - one of the properties is typically “ID” (natural user-specified ID)
     - recommended, properties can be indexed
       - automatic indexes
  3. Using “labels”
     - group nodes into “subsets” (named graph)
     - a node can have more than one label
       - belong to more subsets

```
Node ali =
   graphDb.findNode(Label.label("Person"), "name", "Ali");
```

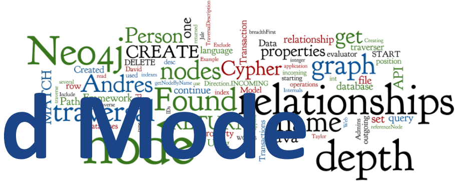
Task 3: Movies in Embedded Mode



localhost

- Use the Movie database in the embedded mode
  - download the Java Maven project from course page
  - insert the Movie database using Cypher
    - The code is prepared in `MoviesBuild.java`
    - source data in `movies-insert.cypher`

Task 3: Query Movies in Embedded Mode

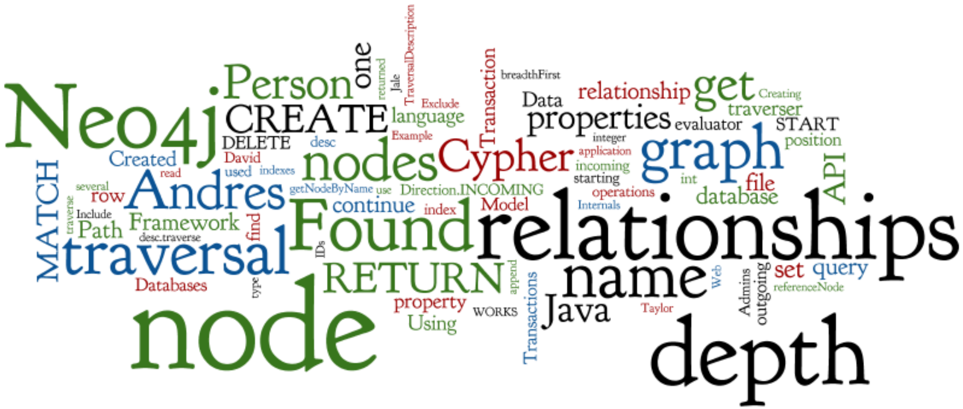


- Find all actors who played in a movie with Keanu Reeves.

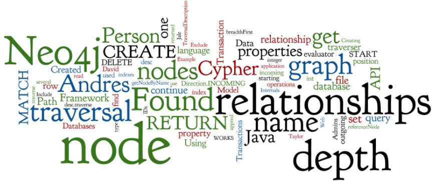
- Find all directors of movies where acted Tom Hanks.

Questions?

Please, any questions? Good question is a gift...



References



- RNDr. Irena Holubova, Ph.D. MMF UK course NDBI040: Big Data Management and NoSQL Databases

- Neo4j http://www.neo4j.org/
- Neo4j Manual http://neo4j.org/docs/stable/
- Neo4j Download http://www.neo4j.org/download
- Cypher Query Language http://neo4j.com/docs/stable/cypher-query-lang.html