## Admin matters

- **next 2 weeks**

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Day</th>
<th>Topic</th>
</tr>
</thead>
</table>
| 8    | 8 Sept   | Tues| L15: Neo4j graph database  
Essay presentation: topic 9                                         |
| 9    | 9 Sept   | Wed | L16: Neo4j graph database  
Essay presentation: topic 10                                         |
|      | 11 Sept  | Fri | **Practical 6b: MongoDB document database**                          |
| 9    | 15 Sept  | Tues| L17: Essay presentations: topics 11,12,13,14                         |
| 16 Sept | Wed | L18: No lecture  
**Semester test: evening**                                         |
| 18 Sept | Fri | **Practical 7a: Tutorial for prac7 Neo4j database**                 |
Outline

1. Neo4j functions & database operations

2. Aspects of graph databases
   a. Data models for graph databases
   b. problems solved by graph DBs (Homework)
   c. Database design

Reference:
(1) Neo4j 2.1.3 manual (documentation)
(2) other documents
### CRUD operations: READ - FUNCTIONS (1)

- Functions (ch. 12, pages 187-213)

<table>
<thead>
<tr>
<th>Function category</th>
<th>Description &amp; example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>predicates</strong></td>
<td>return true or false for a given set of input. used to filter out subgraphs in the WHERE part of a query</td>
</tr>
<tr>
<td>ALL, ANY, NONE, SINGLE, EXISTS</td>
<td></td>
</tr>
<tr>
<td><strong>scalar functions</strong></td>
<td>Scalar functions return a single value e.g.</td>
</tr>
<tr>
<td>LENGTH, TYPE, ID, etc</td>
<td></td>
</tr>
<tr>
<td><strong>collection functions</strong></td>
<td>Collection functions return collections of things: nodes in a path, etc. e.g.</td>
</tr>
<tr>
<td>NODES, RELATIONSHIPS, LABELS, etc.</td>
<td></td>
</tr>
<tr>
<td><strong>mathematical &amp; statistical functions:</strong> trigonometric, statistical, other</td>
<td>operate on numerical expressions</td>
</tr>
<tr>
<td><strong>string functions</strong></td>
<td>These functions all operate on string expressions</td>
</tr>
<tr>
<td>SUBSTRING, TRIM, LOWER, UPPER etc.</td>
<td></td>
</tr>
</tbody>
</table>
CRUD operations: functions (1)

predicates: ALL, ANY, NONE, SINGLE, EXISTS

List all the nodes with a name property and true/false indicating if they follow someone

MATCH (n)
WHERE EXISTS(n.name)
RETURN n.name AS name,
EXISTS((n)-[:FOLLOW]->()) AS isAfollower

name: “Thandi”
from: “Durban”
hobby: “singing”
age: 20

name: “Neo”
from: “Tshwane”
hobby: “soccer”

name: “Johan”
from: “Pretoria”
hobby: “surfing”

name: “Melanie”
from: “Joburg”
hobby: “dance”
scalar functions: LENGTH, TYPE, ID, etc

For all paths of length 2, list the node names and path length:

MATCH p=(a)-->(b)-->(c)
WHERE a.name="Thandi"
RETURN a.name, b.name, c.name, length(p)
CRUD operations: functions (3)

collection functions: NODES, RELATIONSHIPS, LABELS, etc.

For all paths of length 2, show the nodes in the path p

MATCH p=(a)-->(b)-->(c)
WHERE a.name="Melanie" AND c.name="Thandi"
RETURN nodes(p)
CRUD operations: functions (4)

mathematical & statistical functions:

MATCH (a),(b)
WHERE a.name = “Thandi” AND b.name = “Neo”
RETURN a.age, b.age, abs(a.age - b.age)

abs(..) is the absolute value of the age difference is returned.
MATCH (psn)
RETURN upper(psn.name)

returns the original strings in uppercase.

name: “Thandi”
from: “Durban”
hobby: “singing”
age: 20

name: “Neo”
from: “Tshwane”
hobby: “soccer”

name: “Johan”
from: “Pretoria”
hobby: “surfing”

name: “Melanie”
from: “Joburg”
hobby: “dance”

name: “Melanie”
from: “Joburg”
hobby: “dance”

name: “Neo”
from: “Tshwane”
hobby: “soccer”

name: “Johan”
from: “Pretoria”
hobby: “surfing”

name: “Melanie”
from: “Joburg”
hobby: “dance”
variable length patterns for paths:
e.g. all Persons that are 1 or 2 links way from Melanie

MATCH (psM:Person {name:"Melanie"})-[*1..2]-(psn)
RETURN DISTINCT psn.name

returns the names for nodes node1 and node 2 where
Melanie → node1 → node2

name: “Thandi”
from: “Durban”
hobby: “singing”
age: 20

name: “Neo”
from: “Tshwane”
hobby: “soccer”

name: “Johan”
from: “Pretoria”
hobby: “surfing”

name: “Melanie”
from: “Joburg”
hobby: “dance”

FOLLOWS since: 2010
FOLLOWS since: 2009
FOLLOWS since: 2012
variable length patterns for paths:
e.g. shortest path from Johan to Neo:

```
MATCH p = shortestPath ( 
  (psJ:Person {name:"Johan"})-[*]-(psN:Person {name:"Neo"}) )
RETURN p
```

returns the nodes & relationships in the shortest path: 

- **Thandi**
  - from: "Durban"
  - hobby: "singing"
  - age: 20

- **Neo**
  - from: "Tshwane"
  - hobby: "soccer"

- **Johan**
  - from: "Pretoria"
  - hobby: "surfing"

- **Melanie**
  - from: "Joburg"
  - hobby: "dance"

- **FOLLOWS** since: 2010
- **FOLLOWS** since: 2009
- **FOLLOWS** since: 2012
CRUD operations: patterns & paths (3)

variable length patterns for paths:
e.g. shortest path from Johan to Neo (limit search to max 5):

MATCH p = shortestPath (  
  (psJ:Person {name:"Johan"})-[*..5]-(psN:Person {name:"Neo"})  )  
RETURN psJ.name, psN.name, length(p)

returns the names for nodes & length of shortest path
Graph algorithms (Refer to manual – chapter 18)

- Neo4j graph algorithms
  - is a component that contains Neo4j implementations of some common algorithms for graphs. It includes algorithms like:
    - Shortest paths,
    - all paths,
    - all simple paths,
    - Dijkstra
    - A*

Reading for the very keen student
Neo4j graph example: HOMEWORK (1)

Calculating the **clustering coefficient of a network** (pg.61)

MATCH (a { name: "startnode" })--(b)
WITH a, count(DISTINCT b) AS n
MATCH (a)--()-[r]()->(a)
RETURN n, count(DISTINCT r) AS r
What is the difference in meaning between the patterns in the following MATCH clauses?

- MATCH (psn:Person)-->(psm)
- MATCH (psn:Person)-[:FOLLOWS]-(follows)
- MATCH (psn:Person)-[:FOLLOWS]->(follows)
- MATCH (psn:Person)<-[[:FOLLOWS]]-(follows)
Data models for graph databases

Property graph model

- information is modeled using three basic building blocks:
  - node (a.k.a. vertex)
  - relationship (a.k.a edge, with direction & type. labeled & directed)
  - property (a.k.a attribute) on nodes and relationships

More specifically:

- the graph model is a labeled and directed multigraph.
  - a labeled graph has a label for each edge (type for edge)
  - a directed graph uses edges with fixed direction
  - a multigraph allows multiple edges between two nodes

Many graph algorithms (from graph theory) are available
  - e.g shortest path, closeness, betweenness, etc.

Neo4j uses the property graph model
consider the entity-relationship model of the domain TV shows (page 39-40 of manual)

Relationship type in Neo4j
- HAS_SEASON
- HAS_EPISODE
- FEATURED_CHARACTER
- PLAYED_CHARACTER
- HAS-REVIEW
- WROTE_REVIEW

NODE TYPES:
- TV_SHOW, SEASON,
- USER, REVIEW,
- EPISODE, ACTOR,
- CHARACTER
Neo4j graph data modeling (1)

• example of entities & relationships in the graph database
NoSQL data models

Diagram for the positioning of the main NoSQL categories in terms of complexity & scalability (Neotechnology):

- Key-value stores
- Bigtable stores
- Document DBs
- Graph DBs (billions of nodes & relationships)
Weaknesses of Relational DB systems

- **Normalization** leads to relations that do not correspond to entities in the “real world”. Information is obtained from the DB through performing a **join of 2 or more tables**

- **Semantic overloading**: No mechanism to distinguish between relations & relationships or between different types of relationships
  
  - Difficulty handling recursive queries (relationships that a relation has with itself)

- **Schema changes are difficult**: Database change requires program changes

- **Designed for content-based associative access**
  
  - Poor for navigational access: Movement between records is not supported (except through stored procedures)

**Task**: explain how NoSQL dbs: MongoDB & Neo4j solve address the above weaknesses
Class presentations

- Essay topic 10