## 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   | 11 Sept  | Fri   | Practical 6b: MongoDB document database                               |
| 9    | 15 Sept  | Tues  | L17: Essay presentations: topics 11,12,13,14                         |
| 16   | 16 Sept  | Wed   | L18: No lecture
      |          |       | Semester test: evening                                               |
| 18   | 18 Sept  | Fri   | Practical 7a: Tutorial for prac7 Neo4j database                      |
Outline

1. RECAP: Neo4j DB environment

2. Neo4j CRUD operations
   a. WRITE clauses
   b. READ clauses
   c. AGGREGATION clauses

Reference: Neo4j 2.1.3 manual (documentation)
Recap: Neo4j graph database

Client / server.

connection URL: http://localhost:7474
Recap: Neo4j Cypher query language

Cypher: Neo4j's graph query language
• purpose built for working with graph data.
• uses patterns to describe graph data
• uses familiar SQL-like clauses
• is declarative (like SQL) i.e. describes what to do, not how to do it

A Neo4j graph consists of:
• Nodes which store data as Properties
• Properties are simple key/value pairs e.g. name: “Thandi” from: “Durban”
• Plus optional components:
  • Constraints & indexes
• System assigns unique node IDs
**Nodes and Relationships (1)**

**Relationships:** Connect nodes in the graph

- The real power of a graph database e.g. Neo4j is in connected data.
- **Relationships describe how the records are related.** e.g. the **FOLLOWS** relationship
  - Thandi FOLLOWS Neo and Johan
  - Johan FOLLOWS Thandi
- Relationships **always have direction & a type** & form patterns of data

- Thandi
  - Name: "Thandi"
  - From: "Durban"
  - Follows: Johan and Neo

- Johan
  - Name: "Johan"
  - From: "Pretoria"
  - Follows: Thandi and Neo

- Neo
  - Name: "Neo"
  - From: "Tshwane"
  - Follows: Thandi
**Relationship properties**

- Store information shared by two nodes.
- In a property graph, **relationships are data records** that can also contain properties. E.g.
  - Thandi FOLLOWS Neo since 2009
  - Thandi FOLLOWS Johan since 2012
  - Johan FOLLOWS Thandi since 2010
## CRUD operations: WRITING clauses

- Writing clauses (ch. 11, pages 160-181)

<table>
<thead>
<tr>
<th>Writing clause</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CREATE</strong></td>
<td>Creating graph elements: nodes, relationships, constraints</td>
</tr>
<tr>
<td><strong>MERGE</strong></td>
<td>a combination of MATCH and CREATE that additionally allows you to specify what happens if the data was matched or created.</td>
</tr>
<tr>
<td><strong>SET</strong></td>
<td>for <strong>updating labels</strong> on nodes and <strong>properties</strong> on nodes and relationships</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>Deleting graph elements: nodes and relationships</td>
</tr>
<tr>
<td><strong>REMOVE</strong></td>
<td>for removing properties and labels from graph elements</td>
</tr>
<tr>
<td><strong>FOREACH</strong></td>
<td>use for updating commands on elements in a collection: a path, or a collection created by aggregation.</td>
</tr>
<tr>
<td><strong>CREATE UNIQUE</strong></td>
<td><strong>Create unique nodes. Create node if missing.</strong> If the pattern described needs a node, and it can’t be matched, a new node will be created.</td>
</tr>
</tbody>
</table>
A CREATE clause can create many nodes and relationships at once.

**CREATE**

( psT:Person { name: "Thandi", from: "Durban", hobby: "singing" } ),
( psJ:Person { name: "Johan", from: "Pretoria", hobby: "surfing" } ),
( psN:Person { name: "Neo", from: "Tshwane", hobby: "soccer" } ),
(psT)-[:FOLLOWS { since: 2012 }]->(psJ),
(psT)-[:FOLLOWS { since: 2009 }]->(psN),
(psJ)-[:FOLLOWS { since: 2010 }]->(psT)
A CREATE clause can create a **constraint** and an **index**

CREATE CONSTRAINT ON ( person: Person)
ASSERT person.name IS UNIQUE

**Note:** adding the unique constraint will add an index on that property

Alternatively:

CREATE INDEX ON : Person(name)

**and relationships:**

CREATE UNIQUE (<pattern1)-[RELATIONSHIP]->(<pattern2>)  //pages 25, 30

- **Johan** from: **Pretoria** hobby: “surfing”
  - FOLLOWED by **Thandi** from: **Durban** hobby: “singing” since: 2010
  - FOLLOWED by **Neo** from: **Tshwane** hobby: “soccer” since: 2012
  - FOLLOWED by **Johan** from: “Pretoria” hobby: “surfing”
CRUD operations: SET

for **updating labels** on nodes and **properties** on nodes & relationships

MATCH (psT)
WHERE psT.name = "Thandi"  SET psT.age = 20  RETURN psT

BUT for relationships:

MATCH (psN), (psT)
WHERE psN.name = "Neo" AND psT.name = "Thandi"
CREATE (psN)-[:FOLLOWS { since: 2014 } ]->(psT )
CRUD operations: MERGE

either matches existing node and binds it, or creates new node and binds it. (Can use only ON CREATE or only ON MATCH or both)

```mermaid
graph LR
  person --> person3["name: "Johan", from: "Pretoria", hobby: :surfing"]
  person3 --> person2
  person3 --> person4["name: "Neo", from: "Tshwane", hobby: "soccer"]
  person4 --> person
```

```
MERGE (psT:Person { name: "Thandi"})
ON CREATE SET psT.name = "Thandi", psT.from = "Durban",
psT.hobby = "singing", psT.age = 21
ON MATCH SET psT.age = 21
RETURN psT
```
CRUD operations: DELETE (1)

- Deleting graph elements: nodes and relationships
- e.g. Delete a node and connected relationships

MATCH (psZ:Person {name: "Zorro"})-[relZ]-( )
DELETE psZ, relZ
CRUD operations: DELETE (2)

- Deleting graph elements: nodes and relationships
- e.g. Delete all nodes and connected relationships

```
MATCH (psn:Person)
OPTIONAL MATCH (psn)-[rel]-( )  //equiv to outer join
DELETE psn, rel
```
CRUD operations: REMOVE

- for removing **properties** and labels from graph elements (nodes & relationships) e.g.

MATCH (psZ:Person) WHERE psZ.name = "Zorro"
REMOVE psZ.hobby
RETURN psZ

name: "Thandi"
from: "Durban"
hobby: "singing"
age: 21

FOLLOWS since: 2010

name: "Johan"
from: "Pretoria"
hobby: "surfing"

FOLLOWS since: 2012

name: "Zorro"
from: "California"
hobby: "swords"
FOLLOWS since: 2000

name: "Neo"
from: "Tshwane"
hobby: "soccer"
FOLLOWS since: 2009

name: "Neo"
from: "Tshwane"
hobby: "soccer"
CRUD operations: READING clauses (1)

- Reading clauses (ch. 10, pages 131-157)

<table>
<thead>
<tr>
<th>Reading clause</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MATCH</strong></td>
<td>The MATCH clause allows you to specify the patterns Cypher will search for in the database.</td>
</tr>
<tr>
<td><strong>OPTIONAL MATCH</strong></td>
<td>if no matches are found, OPTIONAL MATCH will use NULLs for missing parts of the pattern. OPTIONAL MATCH could be considered the Cypher equivalent of the outer join in SQL.</td>
</tr>
<tr>
<td><strong>WHERE</strong></td>
<td><strong>WHERE is not a clause in it’s own right</strong>, rather, it’s part of MATCH, OPTIONAL MATCH, START and WITH. In the case of WITH and START, WHERE simply filters the results.</td>
</tr>
<tr>
<td><strong>AGGREGATION</strong></td>
<td>To calculate aggregated data, Cypher offers aggregation, much like SQL’s aggregates and GROUP BY.</td>
</tr>
</tbody>
</table>
### Patterns for MATCH

<table>
<thead>
<tr>
<th>Pattern</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATCH (&lt;var&gt;)</td>
<td>any node</td>
</tr>
<tr>
<td>MATCH (&lt;var&gt;: &lt;label&gt; )</td>
<td>a single node pattern with label &lt;label&gt; which will assign matches to the variable &lt;var&gt;</td>
</tr>
<tr>
<td>MATCH (&lt;var1&gt;:&lt;label&gt;)-[:&lt;relationship&gt;]-(&lt;var2&gt;)</td>
<td></td>
</tr>
<tr>
<td>MATCH (&lt;var1&gt; { &lt;key&gt;: &lt;value&gt; })--&gt;(&lt;var2&gt;)</td>
<td></td>
</tr>
<tr>
<td>etc</td>
<td></td>
</tr>
</tbody>
</table>
CRUD operations: MATCH

MATCH (psT:Person) RETURN psT;  //returns nodes

MATCH (psT:Person) RETURN psT.name;  //returns names

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

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

FOLLOWS since: 2010

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

FOLLOWS since: 2009

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

FOLLOWS since: 2012
CRUD operations: MATCH & WHERE

specify **selection criteria** for **patterns**
e.g. show Melanie’s details

MATCH (psM)  WHERE psM.name = "Melanie"
RETURN psM.name,  psM.from,  psM.hobby

e.g. find the people that Thandi follows

MATCH (psn:Person)-[:FOLLOWS]->(follows)
WHERE psn.name = "Thandi"
RETURN psn.name,  follows.name
General clauses (ch. 10, pages 115-129)

- **ORDER BY** is used to sort the output.

```
MATCH (psn) RETURN psn.name
ORDER BY psn.name
```

- Sorted list of names is returned

```
name: "Thandi"
from: "Durban"
hobby: "singing"
age: 21
```

```
name: "Neo"
from: "Tshwane"
hobby: "soccer"
```

```
name: "Melanie"
from: "Joburg"
hobby: "dance"
```

```
name: "Johan"
from: "Pretoria"
hobby: "surfing"
```

```
FOLLOWS since: 2010
```

```
FOLLOWS since: 2012
```

```
FOLLOWS since: 2009
```

```
FOLLOWS since: 2010
```
To calculate aggregated data, Cypher offers aggregation, much like SQL’s aggregates and GROUP BY.

<table>
<thead>
<tr>
<th>Aggregation function</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘GROUP BY’</td>
<td>MATCH (n { name: 'A' })--&gt;(x) RETURN n.name, count(x)</td>
</tr>
<tr>
<td>count</td>
<td>RETURN count(*)</td>
</tr>
<tr>
<td>sum</td>
<td>RETURN sum(n.property)</td>
</tr>
<tr>
<td>avg</td>
<td>RETURN avg(n.property)</td>
</tr>
<tr>
<td>stdev</td>
<td>RETURN stdev(n.property)</td>
</tr>
<tr>
<td>max</td>
<td>RETURN max(n.property)</td>
</tr>
<tr>
<td>min</td>
<td>RETURN min(n.property)</td>
</tr>
<tr>
<td>collect</td>
<td>RETURN collect(n.property)</td>
</tr>
<tr>
<td>distinct</td>
<td>RETURN count(DISTINCT b.eyes)</td>
</tr>
</tbody>
</table>
e.g. 1 count the people that Thandi follows:

MATCH (psT {name: "Thandi"})-[:FOLLOWS]-> follows
RETURN psT.name, count(*) AS follows

name: "Thandi"
from: "Durban"
hobby: "singing"
age: 21
FOLLOWS since: 2010
FOLLOWS since: 2012

name: "Johan"
from: "Pretoria"
hobby: "surfing"
FOLLOWS since: 2012

name: "Neo"
from: "Tshwane"
hobby: "soccer"
FOLLOWS since: 2009

name: "Melanie"
from: "Joburg"
hobby: "dance"
FOLLOWS since: 2010
CRUD operations: AGGREGATION (3)

e.g.2 Count the number of relationships in the graph

MATCH (psn)-[relation]->( )
RETURN type(relation), count(*)