Admin matters (1)

- next 2 weeks

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Day</th>
<th>Topic</th>
</tr>
</thead>
</table>
| 7    | 1 Sept | Tues | L13: MongoDB document database  
                 Essay presentation: topic 7 |
| 2 Sept | Wed | L14: Neo4j graph database  
                Essay presentation: topic 8 |
| 4 Sept | Fri | Practical 6a: Tutorial on MongoDB |
| 8    | 8 Sept | Tues | L15: Neo4j graph database  
                 Essay presentation: topic 9 |
| 9 Sept | Wed | L16: Neo4j graph database  
                Essay presentation: topic 10 |
| 11 Sept | Fri | Practical 6b: MongoDB document database |
Outline

1. Graph theory
2. Graph data structure
3. Graph database concepts
4. Neo4j
   a. Client / server model
   b. web client
   c. Cypher queries
5. Youtube presentation by Neotechnology

Reference: Neo4j 2.1.3 manual (documentation)
Graph theory: Mathematical perspective (1)

Graph theory:
- A branch of discrete mathematics, distinguished by its geometric approach to the study of various objects. **The principal object of the theory is a graph and its generalizations.**

A graph:
- can be denoted as $G = (V, E)$.
- $V$ is a set of vertices, $E$ is a set of edges.
- Each edge is a tuple $(v, w)$ where $w, v \in V$.
- A third component can be added to the edge tuple to represent a **weight**.
- A **subgraph** $s$ is a set of edges $e$ and vertices $v$ such that $e \in E$ and $v \in V$.

A Simple Example of a Directed Graph

[Diagram of a directed graph with vertices V0, V1, V2, V3, V4, V5 and edges with labels 1 to 9.]
Graph theory: Mathematical perspective (2)

- **A vertex (also called a “node”)** is a fundamental part of a graph. It can have a name, which we will call the “key.” A vertex may also have additional information.

- **An edge (also called an “arc”)** connects two vertices to show that there is a relationship between them.

- Edges may be one-way or two-way. If the edges in a graph are all one-way, we say that the graph is a **directed graph**, or a **digraph**.

- **Weight**: Edges may be weighted to show that there is a cost to go from one vertex to another. E.g. in a graph of roads that connect one city to another, the weight on the edge might represent the distance between the two cities.

- **A path** in a graph is a sequence of vertices that are connected by edges.

- **A cycle** in a directed graph is a path that starts and ends at the same vertex.

A Simple Example of a Directed Graph
The Graph Abstract Data Type (part of the CS curriculum)

The graph abstract data type (ADT) is defined as follows:

- `Graph()` creates a new, empty graph.
- `addVertex(vert)` adds an instance of Vertex to the graph.
- `addEdge(fromVert, toVert)` Adds a new, directed edge to the graph that connects two vertices.
- `addEdge(fromVert, toVert, weight)` Adds a new, weighted, directed edge to the graph that connects two vertices.
- `getVertex(vertKey)` finds the vertex in the graph named vertKey.
- `getVertices()` returns the list of all vertices in the graph.
- `in` returns True for a statement of the form `vertex in graph`, if the given vertex is in the graph, False otherwise.
Recall: Classification of NoSQL databases

1) Key-value stores
2) Document databases
3) Wide-column stores
4) Graph databases

A graph database can store any kind of data using a few simple concepts:

- **Nodes** - graph data records
- **Relationships** - connect nodes
- **Properties** - named data values
Graph databases

- use structured **relationship graphs of** interconnected key-value pairings

- **Graph** represented as an **Object-oriented network** of:
  - **nodes** (objects),
  - **edges** (node relationships),
  - **properties** (object attributes expressed as *key-value* pairings)
Graph databases

- Primary uses:
  - representing social networks,
  - generating recommendations,
  - conducting forensic investigations

- Example DBs: Neo4j, InfoGrid, AllegroGraph
Neo4j graph database: Nodes (1)

- The simplest graph
  - has just a single node
  - with some **named values called Properties**.

- **In general:**
  - Node is the name for **data records** in a graph
    - Data is stored as Properties
    - Properties are simple **key/value pairs**
      - e.g.
        - name: “Thandi”
        - from: “Durban”
Neo4j graph database: Nodes (2)

- **Labels** are used to associate (group) a set of nodes.

- Nodes can be grouped together by applying a **label** to each member
  - E.g. nodes for people can all be given the label ‘**Person**’

- A node can have zero or more labels
  - Labels do not have any properties

- **Neo4j is schema-optional**
  - nodes can have a mix of common and unique properties.

**This node can be assigned the label** **Person**

- name: “Thandi”
- from: “Durban”
Neo4j: Relationships

**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
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
The version to be used for the prac is Neo4j is version 2.1.3, released in July 2014. The installers for Neo4j are available from [http://www.neo4j.org/download](http://www.neo4j.org/download) as well as the COS326 website:

- **MSWindows:**
  - installer file: `neo4j-community.windows_2_1_3.exe`
  - Java library: `neo4j-community.windows_2_1_3.zip`

- **Debian Linux:** The instructions for installation are available from [http://debian.neo4j.org](http://debian.neo4j.org)
Windows 7:
Step 1:
Start the server process from the Start menu

Step 2:
Browse to folder for DB

Step 3:
Start the web client process from the Neo4j Community dialog by clicking on the URL or by typing the displayed URL in a web browser of your choice

To create a new database, create a folder e.g. Mydb.graphdb
Neo4j web client: http://localhost:7474

- Neo4j Browser is a command driven client, like a web-based shell environment which supports:
  - Graph visualization of Cypher query results containing nodes and relationships
  - Convenient exploration of Neo4j's REST API
- ****Use a browser which gives you the interface below***
The editor is the primary interface for entering and running commands.

Enter Cypher queries to work with graph data.

Single line editing for brief queries or commands

Switch to multi-line editing with <shift-enter>

Run a query with <ctrl-enter>

OR click on the execute button to run a query.
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
Neo4j: CREATE statement

Create a node:
CREATE ( psT:Person { name: "Thandi", from: "Durban", hobby: "singing" } )

- CREATE clause to create data
- ( ) parenthesis to indicate a node
- psT:Person a variable ‘psT’ and label 'Person' for the new node
- {} brackets to add properties to the node

$ CREATE (psT:Person { name: "Thandi", from: "Durban", hobby: "singing" })

Cypher CREATE (psT:Person { name: "Thandi", from: "Durban", hobby: "singing" })

✓ Added 1 label, created 1 node, set 3 properties, returned 0 rows in 746 ms
Neo4j: MATCH statement (1)

MATCH: Finding nodes
MATCH (psT:Person) WHERE psT.name = “Thandi" RETURN psT;

- MATCH clause to specify a pattern of nodes and relationships
- (psT:Person) a single node pattern with label 'Person' which will assign matches to the variable ‘psT'
- WHERE clause to constrain the results
- pst.name = “Thandi" compares name property to the value “Thandi" 
- RETURN clause used to request particular results
MATCH: Finding nodes
MATCH (psT:Person) WHERE psT.name = "Thandi" RETURN psT;

displayed when you click on person node 181
Neo4j: CREATE statement - again

CREATE clauses can create many nodes and relationships at once.

MATCH (psT:Person) WHERE psT.name = "Thandi"
CREATE

  ( 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)
**Neo4j: MATCH statement – again (1)**

- **Pattern matching**
  - Describe what to find in the graph
  - a pattern can be used to find Thandi’s followers

MATCH (psn:Person)-[:FOLLOWS]-(followers)
WHERE psn.name = "Thandi" RETURN psn, followers;

- **MATCH** clause to describe the pattern from known Nodes to found Nodes
- **(psn)** starts the pattern with a Person (qualified by WHERE)
- **-[:FOLLOWS]-(followers)** matches “FOLLOWS" relationships (in either direction)
- **(followers)** will be bound to Thandi’s followers
MATCH (psn:Person)-[:FOLLOWS]-(followers)
WHERE psn.name = "Thandi" RETURN psn, followers;
HOMEWORK:

watch the YouTube video on NoSQL and Neo4j:

YouTube link is:
http://www.youtube.com/watch?v=UodTzseLh04