# Analysis of Large Scale Socil Networks

# Exercise Session 2: Neo4J - Graph Database

## Software

* <http://neo4j.org>

## Cypher Reference

* <http://neo4j.com/docs/developer-manual/current/cypher>

## Data Sets

* GTFS (General Transit Feed Specification)-file of Belgian Railway company, see: <https://hello.irail.be/gtfs/>

## Preprocessing

*See:* [*http://blog.bruggen.com/2015/11/loading-general-transport-feed-spec.html*](http://blog.bruggen.com/2015/11/loading-general-transport-feed-spec.html)

* *Prepare graph database: create constraints and indexes*

create constraint on (a:Agency) assert a.id is unique;

create constraint on (r:Route) assert r.id is unique;

create constraint on (t:Trip) assert t.id is unique;

create index on :Trip(service\_id);

create constraint on (s:Stop) assert s.id is unique;

create index on :Stoptime(stop\_sequence);

create index on :Stop(name);

* *Upload Data*

//add the agency

load csv with headers from

'file:///nmbs/agency.txt' as csv

create (a:Agency {

id: toInt(csv.agency\_id),

name: csv.agency\_name,

url: csv.agency\_url,

timezone: csv.agency\_timezone});

Agency = Label

Id, name, url, timezone = Properties

load csv with headers from

'file:///nmbs/routes.txt' as csv

match (a:Agency {id: toInt(csv.agency\_id)})

create (a)-[:OPERATES]->(r:Route {

id: csv.route\_id,

short\_name: csv.route\_short\_name,

long\_name: csv.route\_long\_name,

type: toInt(csv.route\_type)});

Match = This specifies a pattern that will be searched in the database.

-[:OPERATES]-> Defines a directed relationship of type OPERATES

// add the trips

load csv with headers from

'file:///nmbs/trips.txt' as csv

match (r:Route {id: csv.route\_id})

create (r)<-[:USES]-(t:Trip {

id: csv.trip\_id,

service\_id: csv.service\_id,

headsign: csv.trip\_headsign,

direction\_id: csv.direction\_id,

short\_name: csv.trip\_short\_name,

block\_id: csv.block\_id,

bikes\_allowed: csv.bikes\_allowed,

shape\_id: csv.shape\_id});

//add the stops

load csv with headers from

'file:///nmbs/stops.txt' as csv

create (s:Stop {

id: csv.stop\_id,

name: csv.stop\_name,

lat: toFloat(csv.stop\_lat),

lon: toFloat(csv.stop\_lon),

platform\_code: csv.platform\_code,

parent\_station: csv.parent\_station,

location\_type: csv.location\_type,

timezone: csv.stop\_timezone,

code: csv.stop\_code});

//connect parent/child relationships to stops

load csv with headers from

'file:///nmbs/stops.txt' as csv

with csv

where not (csv.parent\_station is null)

match (ps:Stop {id: csv.parent\_station}),

(s:Stop {id: csv.stop\_id})

create (ps)<-[:PART\_OF]-(s);

with = This allows u to manipulate the data before it passed to the following query part

where = defines a filter for the with statement or adds constraints to the pattern to be matched.

//add the stoptimes

using periodic commit

load csv with headers from

'file:///nmbs/stop\_times.txt' as csv

match (t:Trip {id: csv.trip\_id}), (s:Stop {id: csv.stop\_id})

create (t)<-[:PART\_OF\_TRIP]-(st:Stoptime {

arrival\_time: csv.arrival\_time,

departure\_time: csv.departure\_time,

stop\_sequence: toInt(csv.stop\_sequence)})

-[:LOCATED\_AT]->(s);

Using Periodic Commit = This avoids fails due to memory constraints

//connect the stoptime sequences

match (s1:Stoptime)-[:PART\_OF\_TRIP]->(t:Trip),

(s2:Stoptime)-[:PART\_OF\_TRIP]->(t)

where s2.stop\_sequence=s1.stop\_sequence+1

create (s1)-[:PRECEDES]->(s2);

## Access Neo4J Database

Running an a EC2 instance:

<http://ec2-34-245-42-122.eu-west-1.compute.amazonaws.com:7474/browser/>

User: neo4j

Passwod: ALSN-neo4j

## Exercise 1:

## Run this query:

match (leuven:Stop)-[PART\_OF]-(ls:Stop)--(lst:Stoptime)

where upper(leuven.name)='LEUVEN' and lst.departure\_time >="07:00:00" and lst.departure\_time <="08:00:00"

with leuven,lst,ls

match (brugge:Stop)-[PART\_OF]-(bs:Stop)--(bst:Stoptime)

where upper(brugge.name)='BRUGGE' and lst.departure\_time < bst.arrival\_time

with leuven,lst, brugge, bst,ls,bs

match p = shortestpath((lst)-[:PRECEDES\*]-(bst))

with leuven,ls,brugge,bs, lst,bst,nodes(p) as n

unwind n as nodes

match (nodes)-[:PART\_OF\_TRIP]->(t:Trip)-[:USES]-(route)

return bst,lst,route,leuven,ls,brugge,bs,t

Describe the output of this query:

Move the nodes so in order to create a good visualization of the paths between the different train stops. Export this network as a png file.

## Exercise 2:

## Complete the Graph Data Model

|  |  |  |
| --- | --- | --- |
| Label Source Node | Relation Type | Label Target Node |
| Stoptime | PRECEDES | Stoptime |
| Stoptime | LOCATED\_AT |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Exercise 3:

## Write Cypher queries to calculate these properties:

1. *Number of nodes and relations.*
2. *Number of nodes per Label*
3. *The number of stops being part of main stops ‘Leuven’ and ‘Brugge’*
4. *The number of trains arriving in Leuven between 7 and 8 AM.*
5. *The number of stops on the trip with id 7415*

## Use a cypher query to list the following information:

1. *The name of all stops on trip with id 7415 ordered by stop sequence*
2. *All names of the distinct routes from or to Brussels*
3. *Shortest Path between ‘Leuven’ and ‘Kiewit’*
4. *Network with all trips and routes leaving Leuven at 7:19 AM*
5. *A shortest paths (limit to 50 ) between ‘Leuven’ and ‘Brugge’*
6. *All shortest paths (limit to 50 ) between ‘Leuven’ and ‘Brugge’ leaving Leuven between 7 and 8 AM*
7. *Network of all direct routes between Leuven and Brugge, leaving Leuven between 7 and 8 AM with time and platform of departure and arrival*