1.	Start Neo4J
1.1	make sure vagrant is installed: http://www.vagrantup.com/downloads
1.2	install virtualbox: https://www.virtualbox.org/wiki/Downloads
1.3	install git: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
1.4	download the project - go to a folder of your choice via command line window - execute the command: **git clone https://github.com/tomvdbulck/training-no-sql.git**
1.5	start up the vagrant box - navigate to the redis/vagrant folder: ***cd neo4j/vagrant*** - enter the command: ***vagrant up***
This will now download and install the vagrant box, this might take a while	
1.6	start up the redis server via a **command prompt** - enter the command: **vagrant ssh** - you will now go the the vagrant box => this is just to verify that the machine is up
1.7	Open your web browser and go to: http://localhost:7474
2.	Basic operations (CRUD, MATCH, WHERE)
2.1	Enter: **CREATE (tom:Person { name:"Tom", address: "Essen, Belgium"})** This will create a node “tom” with as label “Person” with the properties “name” and “address” The client will return: Added 1 label, created 1 node, set 2 properties, statement executed in 1410 ms.
2.2	Lets retrieve the person: **MATCH (variable:Person) WHERE variable.name = "Tom" RETURN variable;** This will retrieve a Person and put it in the variable “variable”. Where the “name” of that “Person” equals “Tom” With RETURN you will get back the results MATCH only allows key/value pairs to be supplied as filter clauses.

2.4	The create statement can handle many nodes and relations at once: **CREATE (js:Person { name: "Johan", from: "Kalmthout", learn: "surfing" }),** **(ir:Person { name: "Ian", from: "Brussels", title: "data scientist" }),** **(rvb:Person { name: "Rik", from: "Antwerp", beer: "Orval" }),** **(kris:Person { name: "Kris", from: "Gent", database: "cassandra" }),** **(js)-[:KNOWS]->(ir),(js)-[:KNOWS]->(rvb),** **(ir)-[:KNOWS]->(js),(ir)-[:KNOWS]->(ally),** **(rvb)-[:KNOWS]->(kris)** Click on the play button to execute the statement.
2.5	With **MATCH person:Person RETURN person** you will get an overview of all the persons
2.6	Now we shal link Tom to the other persons: **MATCH(tom:Person) WHERE (tom.name = "Tom")** **MATCH(js:Person) WHERE (js.name = "Johan")** **MATCH(kris:Person) WHERE (kris.name = "Kris")** **CREATE (tom)-[:KNOWS {since: 2001}]->(js),(tom)-[:KNOWS {rating: 5}]->(kris);** return: Set 2 properties, created 2 relationships, statement executed in 116 ms.
2.7	Now lets retrieve Tom with his Friends: **MATCH (pers:Person)-[:KNOWS]-(friends) WHERE pers.name = "Tom" RETURN pers, friends** This will show you the relations
2.8	If we want to see who Tom can introduce to Kris to learn Cassandra We can run: **MATCH (js:Person)-[:KNOWS]-(connector)-[:KNOWS]-(databaseDude)** **WHERE databaseDude.database = "cassandra"** **RETURN js, connector, databaseDude** This will show us that Johan can be introduced to Kris by Tom Please note that the WHERE must always be used together with a MATCH. WHERE simply filters the results and allows extra options to be carried out (like functions: ID(node) => retrieves the ID

2.9	Lets try a delete. **MATCH(js:Person) WHERE (js.name = "Johan") DELETE js;** This will throw an error: `org.neo4j.kernel.api.exceptions.TransactionFailureException: Node record Node[1,used=false,rel=14,prop=-1,labels=Inline(0x0:[]),light] still has relationships` Relations must also be deleted before you can delete the nodes For that we have to add the relations as well: **MATCH(js:Person)-[r]-()** **WHERE (js.name = "Johan")** **DELETE js,r** This will delete the statement.
3.	Pre provided exercise by Neo4J
	Neo4J provides some exercises by themselves in the console
3.1	Run: **:play movie graph**
3.2	Notably: retrieve information 4 hops away: **MATCH (bacon:Person {name:"Kevin Bacon"})-[*1..4]-(hollywood)** **RETURN DISTINCT hollywood**
3.3	And how to find the shortest path between 2 nodes: **MATCH p=shortestPath(** **(bacon:Person {name:"Kevin Bacon"})-[*]-(meg:Person {name:"Meg Ryan"})** **)** **RETURN p**

3.4	Find actors that Tom Hanks hasn't yet worked with, but his co-actors have. **MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors),** **(coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(cocoActors)** **WHERE NOT (tom)-[:ACTED_IN]->(m2)** **RETURN cocoActors.name AS Recommended, count(*) AS Strength ORDER BY Strength DESC**

If you want to clear your data completely:

```
MATCH (n)
OPTIONAL MATCH (n)-[r]-()
DELETE n,r
```

=> only use this on small test data sets ;-)