Hands-On: Importing Data Into Neo4j

In this activity, we are going to learn how to load data to Neo4j. Before starting, make sure to have your Neo4j container running.

Step 1. Review the "test.csv" spreadsheet to be used for importing into Neo4j. Open the *test.csv* file located inside your *big-data-5/neo4j* directory. The file is the following:

This is a simple spreadsheet consisting of only a few rows and three columns in an adequate format for importing into a graph database.

To illustrate this example better, you can imagine that this data might represent a simple road network in which the Source and Target values represent towns, and the distance values represent the actual distance in miles between the towns.

Step 2. Review the neo4j CYPHER script and describe the syntax. Before reviewing the script, let's "clean the slate" in Neo4j:

```
match (a)-[r]->() delete a,r;
match (a) delete a
```

The script to import test.csv (simple road network) would be the following:

```
LOAD CSV WITH HEADERS FROM "file:////datasets/test.csv" AS line
MERGE (n:MyNode {Name: line.Source})
MERGE (m:MyNode {Name: line.Target})
MERGE (n) -[:TO {dist: toInteger(line.distance)}]-> (m)
```

The first line of code performs the actual import, specifying the file format and that our file contains headers. In our Docker container (which operates in Linux) we placed the test.csv file within the path *////datasets/test.csv*.

The other three lines of code provide constraints on the formatting of this data, specifying which nodes will be the source nodes, which nodes will be the target nodes, and the properties we will attach to them, as well as defining the relationships and the properties we will attach to the relationships.

As we read each line of data n, we're going to use keyword *line*, to specify the individual line we're currently working on. We use that keyword at the end of our load command and so we'll have to continue to use it in the subsequent merge commands.

Our source node variable is going to be *n* and we'll use the *MyNode* type, which we've made up ourselves. We're going to add a *Name* property to each of our source nodes. and we're going to attach the value in the *Source* column to that particular node.

Likewise for our target nodes, we'll use a variable *m*. We'll define them as the same type (*MyNode*), we will give them a *Name* and we will extract that name from the *Target* column on the particular line we're working with.

Finally, we need to define our edge relationships. We're going to give each edge a label of the word *TO*, and we're going to add a property called *dist*, which represents the distance, and we'll attach the values in the distance column from the particular line we're currently working on.

Step 3. Run the script and validate the resulting graph network. Now that we have reviewed the code, execute it on Neo4J:

```
LOAD CSV WITH HEADERS FROM "file:////datasets/test.csv" AS line
MERGE (n:MyNode {Name: line.Source})
MERGE (m:MyNode {Name: line.Target})
MERGE (n) -[:TO {dist: toInteger(line.distance)}]-> (m)
```

Added 11 labels, created 11 nodes, set 25 properties, created 14 relationships, completed after 188 ms.

Table

Code

To validate your graph, click on the *Database Information* logo on the left side of your screen, followed by clicking on *MyNode* (this is a more straightforward way to visualize your graphs).

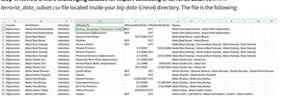
The graph should be displayed as following:

We can see that the nodes are listed all as *MyNode* and there's our graph. The edge relationships should all

have different distance values and each node should be named a different letter.

Now we will try a more complex dataset.

Step 4. Review a more challenging dataset for import consisting of terrorist data. Open the *terrorist_data_subset.csv* file located inside your *big-data-5/neo4j* directory. The file is the following:



This subset contains around 1000 rows of terrorists information, taken from a bigger dataset of around 100,000 rows.

As you can notice, this dataset contains more information for each particular observation.

Step 5. Review the script commands for importing the terrorist dataset. The script that we are going to use is the following:

```
LOAD CSV WITH HEADERS FROM "file:////datasets/terrorist_data_subset.csv" AS row
MERGE (c:Country {Name:row.Country})
MERGE (a:Actor {Name: row.ActorName, Aliases: row.Aliases, Type: row.ActorType})
MERGE (o:Organization {Name: row.Affiliation})
MERGE (a)-[:AFFILIATED_TO {Start: row.AffiliationStartDate, End: row.AffiliationEndDate]->(o)
MERGE(o)-[:IS_FROM]->(c);
```

The first line of code is very similar to the load command we've used in previous datasets.

The second line of code will use a variable *c* and a label *Country* for the particular nodes representing the individual countries in the dataset. In this particular case, we're going to use the keyword *row* instead of *line* to read our data *n*. We could use either word as long we are consistent from command to command. Therefore, we're using the *row* and associating the value in the *Country* column with the node that we're working on in that particular row.

We will do something similar with nodes that are intended to represent the actors or the actual individual terrorists. We're going to use a variable *a* and we will associate a property called *Name*, and associate the *ActorName* with that property. We'll also associate a property named *Aliases*, and associate the value in the *Aliases* column with that property. Finally, we will define a property called *Type* and associate the values in the *ActorType* column with that property.

We're going to create nodes representing organizations as well and we will use the variable *o*, and the label *Organization* for those nodes. We will attach a single property to these nodes called *Name* and we'll assign the values in the *Affiliation*To column to that property.

In the fifth line we're going to define relationships between the Actors and the Organizations they're affiliated with. The relationship label is going to be *AFFILIATED_TO* and we'll define a property called *Start*, assigning the values in the *AffiliationStartDate* with that property. Likewise, we will define a property called *End* and assign the values in the *AffiliationEndDate* with that property.

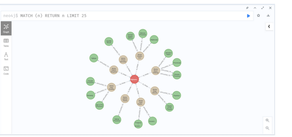
Finally, we're going to create relationships between the countries and the actors. In this case, we will define relationships with the label *IS_FROM* that will describe the fact that a particular actor is from a particular country.

Step 6. Run the script and explore the resulting graph network. Execute the previously reviewed code in Neo4j:

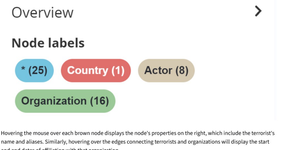
```
LOAD CSV WITH HEADERS FROM "file:////datasets/terrorist_data_subset.csv" AS row
MERGE (c:Country {Name:row.Country})
MERGE (a:Actor {Name: row.ActorName, Aliases: row.Aliases, Type: row.ActorType})
MERGE (o:Organization {Name: row.Affiliation})
MERGE (a)-[:AFFILIATED_TO {Start: row.AffiliationStartDate, End: row.AffiliationEndDate]->(o)
MERGE(o)-[:IS_FROM]->(c);
```

Click on *Database Information* and the asterisk (*) button to display all type of nodes in the network (this will display only the first 25 rows of data):

The graph should look as following:



Each brown node represents a terrorist, and the green nodes represent the organizations they're affiliated with. Since we only returned the first 25 rows, all the terrorists in this sample of the network are from Afghanistan. Also, we only returned information involving 8 actors and 16 organizations.



Hovering the mouse over each brown node displays the node's properties on the right, which include the terrorist's name and aliases. Similarly, hovering over the edges connecting terrorists and organizations will display the start and end dates of affiliation with that organization.

Click on the command that displayed the graph and change 25 to 250 to display more data:

Once executed, the returning graph should be more extensive:

Now we are displaying information involving 132 terrorists and 117 organizations. There is still only one country involved, which is Afghanistan.

To display the full network, we need to make a little adjustment to our visualization settings. On the left side of your screen, click on *Browser Settings* and set *Initial Node Display* to 1000.

Then, run the displaying command again, setting the limit to 1000:

```
MATCH (a) RETURN a LIMIT 1000
```

Zooming out, you should be able to display the full network:

Go to next item ✓ Completed

Like Dislike Report an issue