# Hadoop, Impala and Neo4J

Back in December, I wrote about some ways of moving data from Hadoop into Neo4J using Pig, Py2Neo and Neo4J. Overall, it was successful although maybe not at the scale I would have liked. So this is really attempt number two at using Hadoop technology to populate a Neo4J instance.

In this post, I'll use a combination of Neo4J's batchInserter plus JDBC to query against Cloudera's Impala which in turn queries against files on HDFS. We'll use a list of 694,221 organizations that I had from previous docGraph work.

**What is Cloudera Impala**
A good overview of Impala is in this presentation on SlideShare. Impala provides interactive SQL, nearly ANSI-92 standard SQL queries and provides a JDBC connector allowing external tools to access Impala. Cloudera Impala is the industry's leading massively parallel processing (MPP) SQL query engine that runs natively in Apache Hadoop. The Apache-licensed, open source Impala project combines modern, scalable parallel database technology with the power of Hadoop, enabling users to directly query data stored in HDFS and Apache HBase without requiring data movement or transformation.

**Installing a Hadoop Cluster**
For this demonstration, we will use the Cloudera QuickStart VM running on VirtualBox. You can download the VM and run it in VMWare, KVM and VirtualBox. I chose VirtualBox. The VM gives you access to a working Impala instance without manually installing each of the components.

**Sample Data**
As mentioned above, the dataset is a list of 694,221 organizations that I had from previous docGraph work. Each organization is a single name that we will use to create organization nodes in Neo4J. An example is shown below:

**Creating a table**
After starting the Cloudera VM, I logged into Cloudera Manager and made sure Impala was starting. Once Impala was up and running, I used the Impala Query Editor to create the table.

This creates an *organizations* table with a single c_orgname column.

**Populating the Table**
Populating the Impala table is really easy. After copying the source text file to the VM and dropping it into an HDFS directory, I ran the following command in the Impala query interface:

I had three organization list files. When I was done, I did a "Select * from organizations" resulting in a count of 694,221 records.

**Querying the table**
Cloudera provides a page on the JDBC driver. This page enables you to download the JDBC driver as well as points you to a GitHub page with a sample Java program for querying against Impala.

**Populating Neo4J with BatchInserter and JDBC**
While there are a multitude of ways of populating Neo4J, in this case I modified an existing program that I had written around the BatchInserter to populate Neo4J. The code can be browsed on Github. The java code creates an empty Neo4J data directory. It then connects to Impala through JDBC and runs a query to get all of the organizations. The code then loops over the result set, determines if a node with the name exists and creates a node with the organizational name if the node did not previously exist.

Running the program over the 694,221 records in Impala, the Neo4J database was created in approximately 20 seconds. Once that is done, I copied the database files into my Neo4J instance and restarted.

Once inside of Neo4J, I can query on the data or visualize the information as shown below.

**Summary**

Impala's JDBC driver combined with Neo4J's BatchInserter provides an easy and fast way to populate a Neo4J instance. While this example doesn't show relationships, you can easily modify the BatchInserter to create the relationships.