Assigment: Data Migration from relational database to NoSQL database

CUNY MSDS DATA 607

Duubar Villalobos Jimenez mydvtech@gmail.com

April 30, 2017

Figure 1:

Data migration from MariaDB to Neo4j

Introduction

Since this is my first time performing this kind of work, I am going to proceed to explain step by step on how I have successful achieved to Migrate my `flights` data base from my `MariaDB` home based server to my `Neo4j` home based server as well (these steps should not be much different for any other option you might want to use).

Seting up `Neo4j` Server

For this, I have already successfully setup an Ubuntu 16.04 LTS Home Web server with Nginx, MariaDB and ISPConfig already installed.

The original link from Neo4j provide a series of instructions but lack of detail and their version employed is not even available on their repository. Hence I used that as a starter point.

After some encountered problems, I have done some extra research and followed Panagiotis Katsaroumpas useful post with some modifications due to Ubuntu OS version and Neo4j version as well.

After some adjustments I am finally able to log in remotely to my home based `Neo4j Server` by firing up my desired web browser of preference and following this link:

http://localhost:7474

Please note that I am not providing my remote address but by providing some tweaks on your home router you will be able to do the same as well.

Adding `flights` to MariaDB

Since my home server doesn't has the flights database in it, I will add it with the following commands.

Function to Write Tables into MariaDB

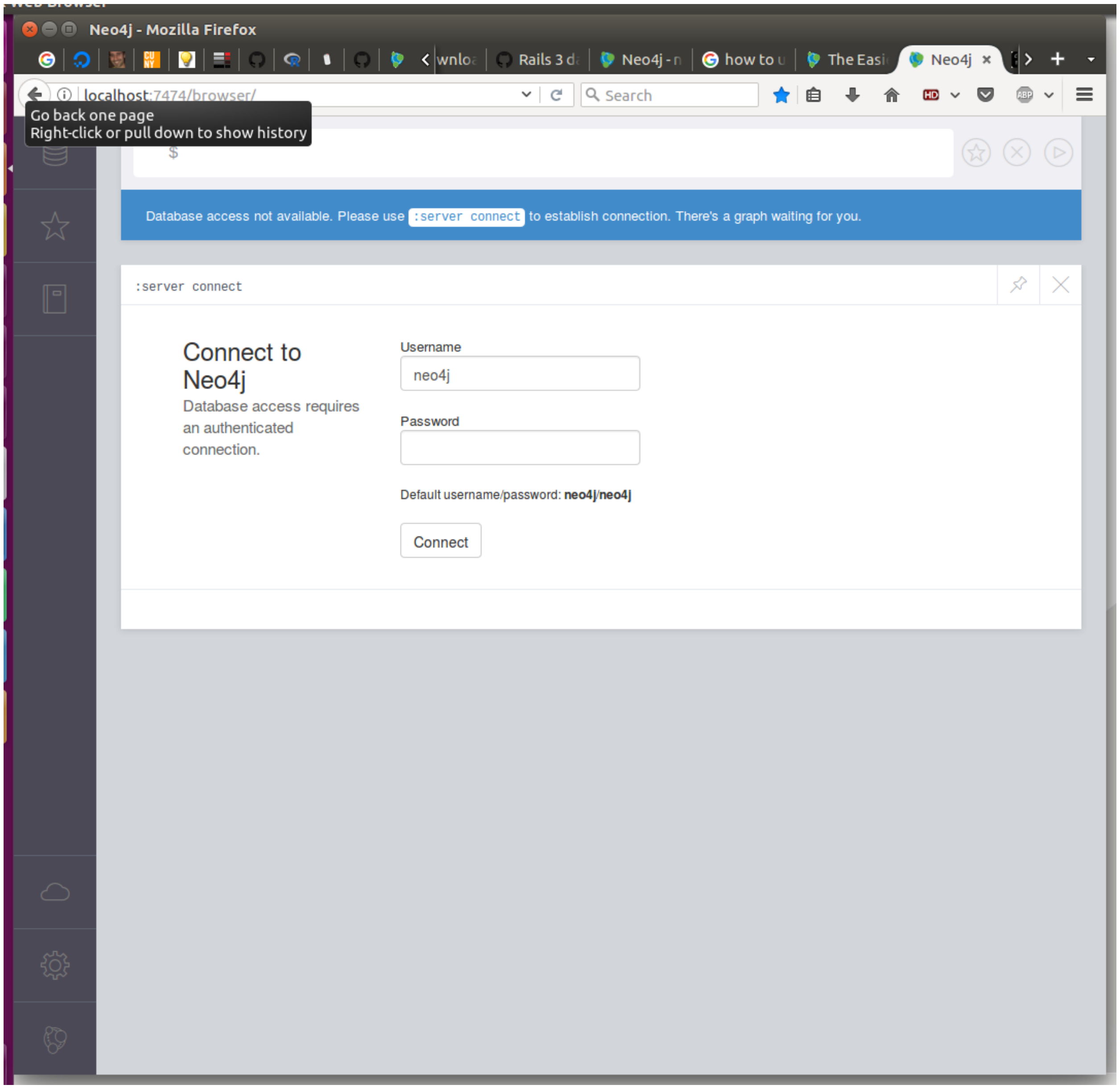
```
# Writing data into MariaDB
# This code connects to our remote MariaDB server and writes a data frame into a table.
library(DBI)
writeMariaDBTable <- function(my.data = NULL, myLocalMariaDBSchema = NULL, myLocalTableName = NULL){

    # Create a RMySQL Connection
    mydbConnection <- dbConnect(RMySQL::MySQL())

    # Creating a schema if it doesn't exist by employing RMySQL() in R
    MySQLcode <- paste0("CREATE SCHEMA IF NOT EXISTS ",myLocalMariaDBSchema,";",sep="")
    dbSendQuery(mydbConnection, MySQLcode)

    # Drop  table if it exists
    myLocalTableName <- tolower(myLocalTableName)
    MySQLcode <- paste0("DROP TABLE IF EXISTS ",myLocalTableName,";",sep="")
    dbSendQuery(mydbConnection, MySQLcode)
    dbDisconnect(mydbConnection)

    # Write our data frame into MariaDB
    mydbConnection <- dbConnect(RMySQL::MySQL(), dbname = myLocalMariaDBSchema)
    dbWriteTable(mydbConnection, myLocalTableName, my.data)
```



Database access not available. Please use `:server connect` to establish connection. There's a graph waiting for you.

```
:server connect
```

Connect to Neo4j

Database access requires an authenticated connection.

Username: neo4j

Password:

Default username/password: **neo4j/neo4j**

Figure 2:

```
    # Closing connection with local MariaDB Schema
    dbDisconnect(mydbConnection)

}
```

Function to read data from MariaDB

```
readMariaDBTable <- function(myLocalMariaDBSchema = NULL, myLocalTableName = NULL){

    # Create a RMySQL Connection
    mydbConnection <- dbConnect(RMySQL::MySQL(), dbname = myLocalMariaDBSchema)

    # Check to see if our table exists? and read our data
    myLocalTableName <- tolower(myLocalTableName)
    if (dbExistsTable(mydbConnection, name = myLocalTableName)  == TRUE){
        my.data <- dbReadTable(mydbConnection, name = myLocalTableName)
    }

    # Closing connection with local MAriaDB Schema
    dbDisconnect(mydbConnection)

    # Return Data
    return(my.data)

}
```

Adding data into MariaDB

Obtaining flights data

```
#install.packages("nycflights13")
library(nycflights13)
```

Calling functions

Write and Read diverse flights data tables one by one.

```
# Write Table into MariaDB
writeMariaDBTable(nycflights13::airlines, myLocalMariaDBSchema = 'flights', myLocalTableName = 'tbl_airl
writeMariaDBTable(nycflights13::airports, myLocalMariaDBSchema = 'flights', myLocalTableName = 'tbl_airp
writeMariaDBTable(nycflights13::flights,  myLocalMariaDBSchema = 'flights', myLocalTableName = 'tbl_flig
writeMariaDBTable(nycflights13::planes,   myLocalMariaDBSchema = 'flights', myLocalTableName = 'tbl_plan
writeMariaDBTable(nycflights13::weather,  myLocalMariaDBSchema = 'flights', myLocalTableName = 'tbl_weat
```

Please note that the above setup took hours to successfully run; so, plan ahead of time.

```
# Read Tables from MariaDB
airines  <- readMariaDBTable(myLocalMariaDBSchema = 'flights', 'tbl_airlines')
airports <- readMariaDBTable(myLocalMariaDBSchema = 'flights', 'tbl_airports')
flights  <- readMariaDBTable(myLocalMariaDBSchema = 'flights', 'tbl_flights')
planes   <- readMariaDBTable(myLocalMariaDBSchema = 'flights', 'tbl_planes')
weather  <- readMariaDBTable(myLocalMariaDBSchema = 'flights', 'tbl_weather')
```

Loading `MariaDB` to `Neo4j` Like Magic (failed attempt)

As far as I know, this is the *simplest SQL to Neo4j* import process possible as described by Neo4j. The catch is that you have to install alternative software to do so and many extra steps need to be taken into place, once this software is properly set and installed all you need is a simple line of code in order to do all the job.

Simply use RubyGems

RubyGems is a package management framework for Ruby.

For this, I had to install as follows:

    sudo apt-get install ruby-dev

Install or upgrade RubyGems

    gem update –system

    gem install rubygems-update

Install Rails using RubyGems

    gem install rails

    update_rubygems

Then install the neo4apis-activerecord

    gem install neo4apis-activerecord

Then install the database adapter gem:

    gem install mysql2

Creating Your Rails App

To create a Rails app configured for MariaDB, run this command:

    rails new flights –database=mysql

This creates a directory called **flights** which houses an app called 'flights' (you can name it anything you like when running the command). Rails expects the name of the database user to match the name of the application, but you can easily change that if need be.

We will now configure which database Rails will talk to. This is done using the database.yml file, located at:

RAILS_ROOT/config/database.yml

Note: RAILS_ROOT is the Rails root directory. In the above example, it would be at /flights (relative to your current location).

Find **MySQL** socket location

    mysqladmin -uroot -p variables | grep socket

In my case this returns:

```
| socket       | /var/run/mysqld/mysqld.sock
```

Since we need to create a file a *config/database.yml* file which looks something like this:

From ~/**flights** location, edit this file in order to look similar to the one below:

    nano /config/database.yml

```
development
    adapter: mysql2
    encoding: utf8
    reconnect: false
    database: flights
    pool: 5
    username: root
    password: Password
    socket: /var/run/mysqld/mysqld.sock
```

After that, Migrate as follows:

    sudo rake db:migrate

Data Import: Making sure you are at the root of the **flights** directory.

    bundle update spring

Need to install NodeJS in Ubuntu

    sudo apt-get install nodejs

Uglifier is a JS wrapper and it needs a JS runtime running or JS interpreter. I would choose to install NodeJS.

After that, we can start rails as follows:

http://localhost:3000/

Then to import all your data it is as simple as:

    neo4apis activerecord all_tables –identify-model –import-all-associations

If everything went well, that simple line is all I needed to do!

The problem I got stuck with is that on April 27, 2017 a new version of rails was released and the `neo4apis-activerecord` install `activesupport 4.2.8` but my Gemfile requires `activesupport 5.1.0` creating a conflict. After a lot of time trying to get a fix or work around on this issue, I had to abandon this idea but definitely will come back with more time due to it's simplicity once accomplished.

Neo4j from R

After seeking a different approach, I found Neo4j for R Developers and Data Scientists.

The documentation can be found here: https://neo4j.com/developer/r/

Goals

If you are an R developer or data scientist, this guide provides an overview of options for connecting from R to Neo4j and even using Neo4j from within R-Studio.

Prerequisites

You should be familiar with graph database concepts and the property graph model. You should have installed Neo4j and made yourself familiar with our Cypher Query language.

Build the Database

Establish a connection (make sure Neo4j is running), clear the graph, and add the necessary uniqueness constraints with addConstraint.

```
graph = startGraph("http://localhost:7474/db/data/",
                   username = myneo4jUser,
                   password = myneo4jPassword)

clear(graph, input = FALSE)

addConstraint(graph, "Airlines", "carrier")
addConstraint(graph, "Airports", "faa"    )
addConstraint(graph, "Flights",  "flight" )
addConstraint(graph, "Planes",   "tailnum")
addConstraint(graph, "Weather",  "origin" )
```

Now, I need to write a function through which I will pass each status object in order to add the flights to the graph.

I only need to use `RNeo4j` functions `getOrCreateNode` and `createRel` to **build the database**. I have to use getOrCreateNode because Airlines, Airports, Planes and Flights will occur more than once and I don't want to create any duplicates. So, getOrCreateNode either creates the node if it doesn't exist or retrieves it from the graph. The syntax is getOrCreateNode(graph, label, . . .) where . . . are the node properties in the form key = value. It is necessary that uniqueness constraints exist to use this function.

Then, `createRel` creates a relationship between two nodes with the syntax createRel(fromNode, type, toNode, . . .), where . . . are optional relationship properties in the form key = value.

The following function takes all objects, as an input and adds it to the graph database.

```
create_db = function(a, b, c, d, e) {

airlines_raw <- a
airports_raw <- b
flights_raw <- c
planes_raw <- d
weather_raw <- e

flights_raw[is.na(flights_raw)] <- "0"
planes_raw[is.na(planes_raw)] <- "0"
weather_raw[is.na(weather_raw)] <- "0"

airlines = getOrCreateNode(graph, "Airlines", carrier = airlines_raw$carrier, names = airlines_raw$name
airports = getOrCreateNode(graph, "Airports", faa = airports_raw$faa, name = airports_raw$name, lat = ai
                           lon = airports_raw$lon, alt = airports_raw$alt, tz = airports_raw$tz, dst = a
                           tzone = airports_raw$tzone)
flights = getOrCreateNode(graph, "Flights", flight = flights_raw$flight, year = flights_raw$year, month
                          dep_time = flights_raw$dep_time, sched_dep_time = flights_raw$sched_dep_time,
                          arr_time = flights_raw$arr_time, sched_arr_time = flights_raw$sched_arr_time,
                          carrier = flights_raw$carrier, tailnum = flights_raw$tailnum, origin = flights
                          air_time = flights_raw$air_time, distance = flights_raw$distance, hour = fligh
                          time_hour = flights_raw$time_hour)
planes = getOrCreateNode(graph, "Planes", tailnum = planes_raw$tailnum, year = planes_raw$year, type = p
                         model = planes_raw$model, engines = planes_raw$engines, seats = planes_raw$sea
weather = getOrCreateNode(graph, "Weather", origin = weather_raw$origin, year = weather_raw$year, month
                            hour = weather_raw$hour, temp = weather_raw$temp, dewp = weather_raw$dewp, hu
                            wind_speed = weather_raw$wind_speed, wind_gust = weather_raw$wind_gust, preci
                            visib = weather_raw$visib, time_hour = weather_raw$time_hour)

    createRel(airports, "Airlines", airlines)
    createRel(airlines, "Flights", flights)
    createRel(airports, "Airport_Temp", weather)
    createRel(planes, "Planes", flights)

}
```

```
create_db(nycflights13::airlines, nycflights13::airports, nycflights13::flights, nycflights13::planes, 
```

Delete the Whole Database

You can take the DETACH DELETE a step further and delete the whole database.

Simply remove any filtering criteria and it will delete all nodes and all relationships.

Go ahead and execute the following statement from within Neo4j.

MATCH (n) DETACH DELETE n

Summary Graph()

```
summary(graph)
```

```
##       This           To     That
## 1 Airports     Airlines Airlines
## 2   Planes       Planes  Flights
## 3 Airlines      Flights  Flights
## 4 Airports Airport_Temp  Weather
```

Screenshots

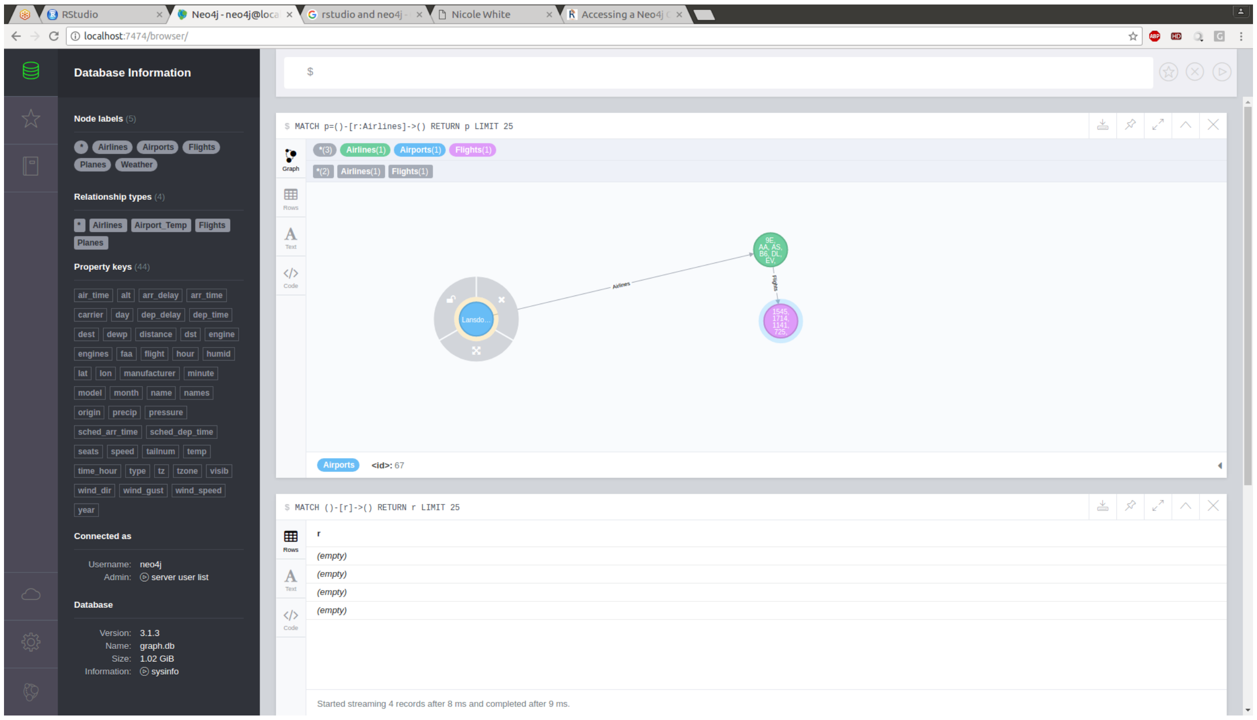
Observations

This project is quite insightful and made me realize how important is to gain knowledge for these great tools available.

Something that struck me was the size for the Neo4j database size after import, it came to about 1.02 Gb.

Conclussions

- Need to do extra research in order to be available to import databases the "Easy Way" even tough in my case became the "difficult way" due to being unable to resolve conflicting version on available packages.
- Need to perform a attain a better understanding on how to use R, MariaDB and Neo4j in a better fashion for larger data-sets (The bigger the data set the most time will be require to process).



Database Information

Node labels (5)

* Airlines Airports Flights Planes Weather

Relationship types (4)

* Airlines Airport_Temp Flights Planes

Property keys (44)

air_time alt arr_delay arr_time carrier day dep_delay dep_time dest dewp distance dst engine engines faa flight hour humid lat lon manufacturer minute model month name names origin precip pressure sched_arr_time sched_dep_time seats speed tailnum temp time_hour type tz tzone visib wind_dir wind_gust wind_speed year

Connected as

Username: neo4j
Admin: server user list

Database

Version: 3.1.3
Name: graph.db
Size: 1.02 GiB
Information: sysinfo

```
$ MATCH p=()-[r:Airlines]->() RETURN p LIMIT 25
```

Airports <id>: 67

```
$ MATCH ()-[r]->() RETURN r LIMIT 25
```

```
r
(empty)
(empty)
(empty)
(empty)
```

Started streaming 4 records after 8 ms and completed after 9 ms.

Figure 3:

References

http://neo4j.com/docs/operations-manual/current/installation/linux/debian/

http://www.codiply.com/blog/standalone-neo4j-server-setup-on-ubuntu-14-04/

https://neo4j.com/blog/loading-sql-neo4j-like-magic/

https://rubygems.org/pages/download

https://www.digitalocean.com/community/tutorials/how-to-setup-ruby-on-rails-with-postgres

https://gist.github.com/erichurst/961978

http://railsapps.github.io/rails-release-history.html

https://rubygems.org/gems/neo4apis-activerecord/versions/0.9.1

https://neo4j.com/developer/r/

https://nicolewhite.github.io/2014/05/30/demo-of-rneo4j-part1.html