Assignment\_12

Priya Shaji

4/24/2019

## NoSQL migration

NoSQL database: Neo4j

relational database: flights database

## Neo4j process

### Data Preparation

1. Make sure your database is empty by issuing the query

match (n) return (n)

and checking that no nodes are found.

If you need to empty a database that has nodes and relationships, you can use the query:

match (a) optional match (a)-[r]-() delete a,r

1. Use Load CSV to create the airport nodes. Note that each airport node should have a label of “Airport” and a set of three attributes: code, city, and state.

load csv with headers from "https://raw.githubusercontent.com/PriyaShaji/Data607/master/Assignment\_12/neo4j-airport-csv-raw(1).csv" as airports create (a1:Airport {label: airports.label, city: airports.city, state: airports.state})

1. Use Load CSV to create the flight nodes. Each flight node should have a label of “Flight” and should have three attributes: number, capacity, and airline.

load csv with headers from "https://raw.githubusercontent.com/PriyaShaji/Data607/master/Assignment\_12/neo4j-flight-lab-data.csv" as flights create (n:Flight {number: flights.flight, airline: flights.airline, capacity: flights.capacity})

1. Use Load CSV to create the arrival relationships. Each arrival relationship should have a label of “Arrive” and should have a single attribute: landing. (This attribute gives the scheduled landing time.)

load csv with headers from "https://raw.githubusercontent.com/PriyaShaji/Data607/master/Assignment\_12/neo4j-flight-lab-data.csv" as flights match (a: Flight {number: flights.flight}),(b:Airport {label: flights.arrive}) create (a) -[r:Arrives {landing: flights.landing}]-> (b)

1. Use Load CSV to create the departure relationships. Each departure relationship should have a label of “Depart” and should have a single attribute: takeoff. (This attribute gives the scheduled takeoff time.)

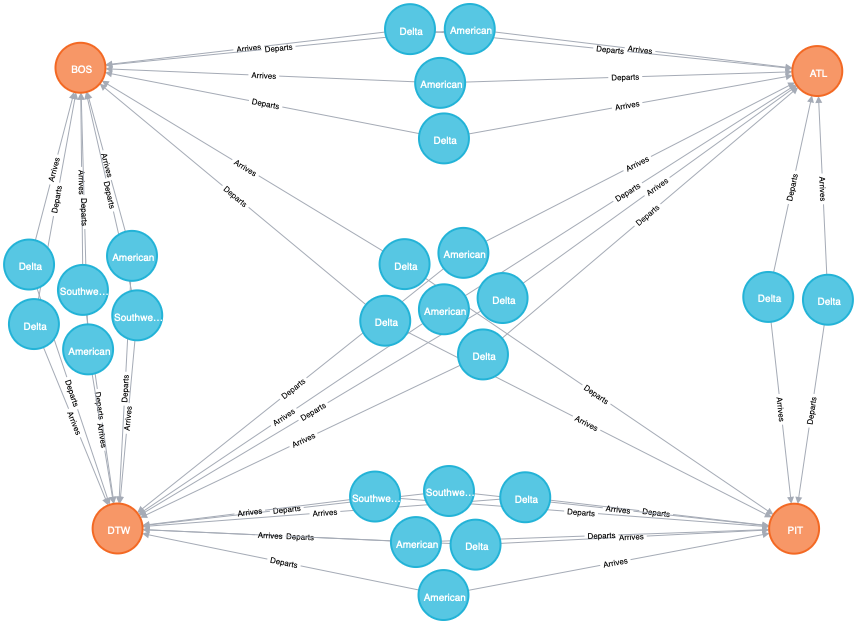
load csv with headers from "https://raw.githubusercontent.com/PriyaShaji/Data607/master/Assignment\_12/neo4j-flight-lab-data.csv" as flights match (a: Flight {number: flights.flight}),(b:Airport {label: flights.depart}) create (a) -[r:Departs {takeoff: flights.takeoff}]-> (b)

1. Run the query

match (n) return (n)

to ensure that the data have loaded correctly. You should see four airport nodes, 24 flight nodes, and the various attributes assigned correctly. Organize the graph in a way that makes these features clear.

You should see a graph that looks something like this.

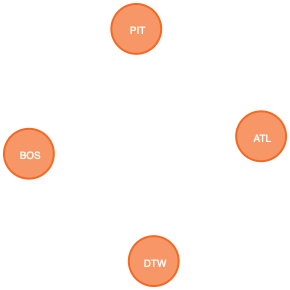


### Data Analysis

Now let’s write Cypher queries to answer each of the following:

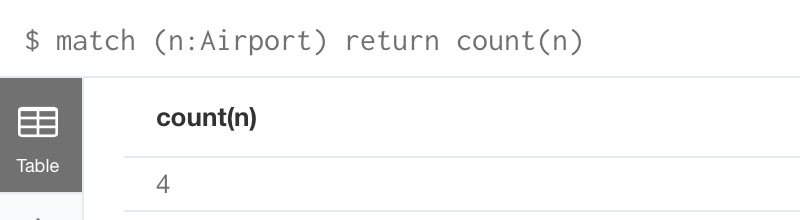
1. Write a query that returns all airports.

match (n:Airport) return n



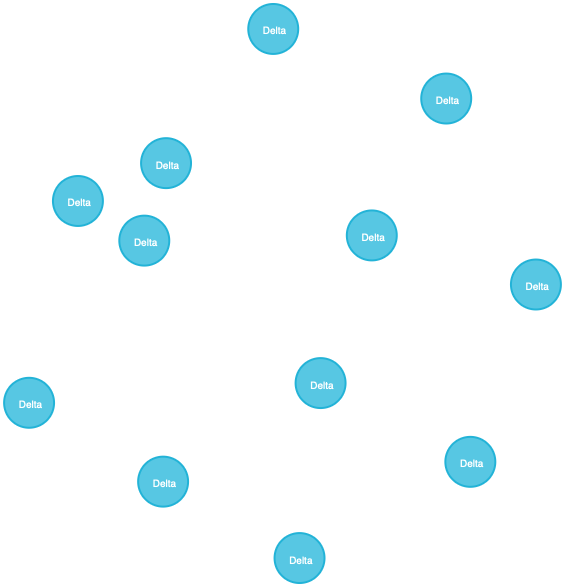
1. Write a query that returns a count of all airports.

match (n:Airport) return count(n)



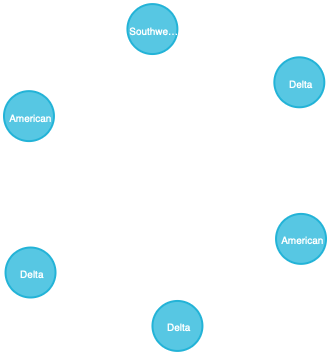
1. Write a query that returns all Delta flights.

match (f:Flight {airline: "Delta"}) return f



1. Return all flights that depart from Boston.

match (f:Flight)-[d:Departs]-(a:Airport {label: 'BOS'}) return f



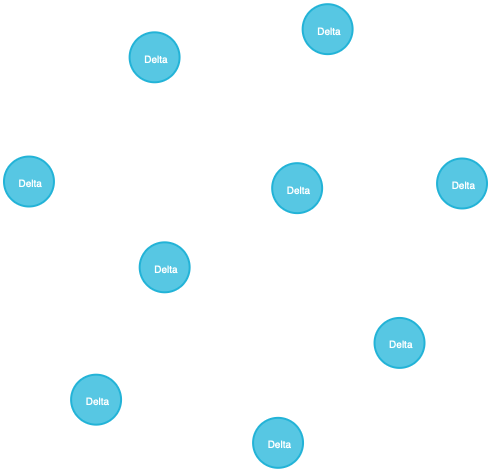
1. Return all flights that run from Detroit to Atlanta.

match (b:Airport {label: 'ATL'})-[r:Arrives]-(f:Flight)-[d:Departs]-(a:Airport {label: 'DTW'}) return f



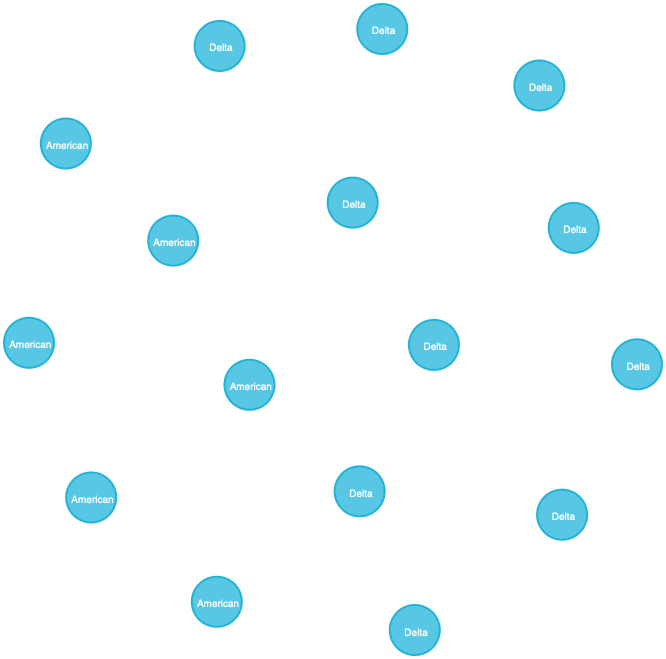
1. Return all flights that take off before 11 a.m.

match (f:Flight) -[d:Departs]-() where toInt(d.takeoff) < 1100 return f



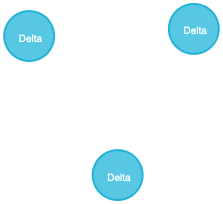
1. Return all flights with a capacity greater than 150 passengers.

match (f:Flight) where toInt(f.capacity) > 150 return f



1. Return all flights on Delta that arrive in Boston.

match (f:Flight {airline: 'Delta'}) -[r:Arrives]-(a:Airport {label: 'BOS'}) return f



### Data Modification

Let’s make a few simple changes to our data. Perform the following tasks:

1. Suppose the Pittsburgh airport designation code has been changed from PIT to PGH. Write a query to update the airport’s code.

match (a:Airport {label: 'PIT'}) set a.label = 'PGH' return a



1. Delta has decided to renumber some of its flights. Write a query to change flight 28 to flight 29.

match (f:Flight {number: '28'}) set f.number = '29' return f



1. Southwest has cancelled all flights between Detroit and Boston. Write a query that removes all such flights. You can remove flights in each direction with separate queries as follows:

match (b:Airport {label: 'BOS'})-[r:Arrives]-(f:Flight {airline: 'Southwest'})-[d:Departs]-(a:Airport {label: 'DTW'}) delete f,r,d

### In R

Connect the neo4j database with R and access the flight and airport database

1. load the library neo4j

#devtools::install\_git("https://github.com/nicolewhite/RNeo4j")  
  
library(RNeo4j)

1. Connect R with neo4j

graph=startGraph("http://127.0.0.1:7474/db/data/", username = "neo4j", password = "data607")

1. clear all the existing database

clear(graph, input = FALSE)

1. The three-character airport code (unique)

addConstraint(graph, "Airport","label")

1. The flight identification number (this is unique)

addConstraint(graph, "Flight", "flight" )

1. load the filght and airport dataset

Flight<-read.csv("https://raw.githubusercontent.com/PriyaShaji/Data607/master/Assignment\_12/neo4j-flight-lab-data.csv",header = T)  
Airport<-read.csv("https://raw.githubusercontent.com/PriyaShaji/Data607/master/Assignment\_12/neo4j-airport-csv-raw(1).csv",header = T)

for ( i in 1: nrow(Airport) ) {  
createNode (graph, "Airport",   
 label = Airport[[i,"label" ]],  
 city = Airport[[i,"city"]],   
 state = Airport[[i,"state"]]   
 )  
}  
  
  
for ( j in 1: nrow(Flight) ) {  
createNode (graph, "Flight",   
 flight = Flight[[j,"flight"]],   
 airline = Flight[[j,"airline"]] ,   
 capacity = Flight[[j,"capacity" ]],  
 arrive = Flight[[j,"arrive" ]]  
 )  
}

query1 = "MATCH (a:Flight), (b:Airport) where a.arrive =b.label RETURN a.flight,a.airline, a.capacity, a.arrive, b.label, b.city, b. state"  
cypher(graph,query1)

## a.flight a.airline a.capacity a.arrive b.label b.city  
## 1 38 Delta 160 ATL ATL Atlanta  
## 2 46 Delta 160 BOS BOS Boston  
## 3 104 Southwest 136 DTW DTW Detroit  
## 4 1258 American 160 ATL ATL Atlanta  
## 5 1292 American 160 BOS BOS Boston  
## 6 27 Delta 160 BOS BOS Boston  
## 7 37 Delta 160 BOS BOS Boston  
## 8 45 Delta 160 PIT PIT Pittsburgh  
## 9 103 Southwest 136 PIT PIT Pittsburgh  
## 10 1257 American 128 ATL ATL Atlanta  
## 11 1291 American 128 DTW DTW Detroit  
## 12 35 Delta 128 PIT PIT Pittsburgh  
## 13 43 Delta 160 PIT PIT Pittsburgh  
## 14 23 Delta 160 DTW DTW Detroit  
## 15 44 Delta 160 ATL ATL Atlanta  
## 16 102 Southwest 136 DTW DTW Detroit  
## 17 1232 American 160 PIT PIT Pittsburgh  
## 18 1278 American 160 DTW DTW Detroit  
## 19 24 Delta 160 ATL ATL Atlanta  
## 20 36 Delta 128 DTW DTW Detroit  
## 21 101 Southwest 136 BOS BOS Boston  
## 22 1231 American 160 BOS BOS Boston  
## 23 1277 American 160 DTW DTW Detroit  
## 24 28 Delta 160 DTW DTW Detroit  
## b. state  
## 1 Georgia  
## 2 Massachussetts  
## 3 Michigan  
## 4 Georgia  
## 5 Massachussetts  
## 6 Massachussetts  
## 7 Massachussetts  
## 8 Pennsylvania  
## 9 Pennsylvania  
## 10 Georgia  
## 11 Michigan  
## 12 Pennsylvania  
## 13 Pennsylvania  
## 14 Michigan  
## 15 Georgia  
## 16 Michigan  
## 17 Pennsylvania  
## 18 Michigan  
## 19 Georgia  
## 20 Michigan  
## 21 Massachussetts  
## 22 Massachussetts  
## 23 Michigan  
## 24 Michigan

### Pros n Cons

Question:

Briefly describe the advantages and disadvantages of storing the data in a relational database vs. your NoSQL database.

Answer)

Relational database

Relational db nomally require us to infer connection between entities suing speical properties such as foreign keys, or out-of-band processing like map-reduce. Relational db can also be called relational database management systems (RDBMS) or SQL databases. The most popular of these are Microsoft SQL Server, Oracle Database, MySQL, and IBM DB2.

Pros:

1. Expressivge query language & secondary Indexes: Users should be able to access and manipulate their data in sophisticated ways to support both operational and analytical applications. Indexes play a critical role in providing efficient access to data, supported natively by the database rather than maintained in application code.
2. Strong consistency: It is much more complex to build applications around an eventually consistent model, imposing significant work on the developer, even for the most sophisticated engineering teams.
3. Enterprise Management and Integrations: Organizations need a database that can be secured, monitored, automated, and integrated with their existing technology infrastructure, processes, and staff, including operations teams, DBAs, and data analysts.

Cons:

1. Performance problems associated with re-assembling simple data structures into their more complicated real-world representations.
2. Lack of support for complex base types, e.g., drawings.
3. SQL is limited when accessing complex data.
4. Knowledge of the database structure is required to create ad hoc queries.
5. Locking mechanisms defined by RDBMSs do not allow design transactions to be supported, e.g., the “check in” and “check out” type of feature that would allow an engineer to modify a drawing over the course of several working days.

NoSql database

Graph db is assembling the simple abstractions of nodes and relationships into connected structures. It enable to bnuild sophisticated models that map closely to the problem domain.Graph db also called NoSQL databases, the most popular being MongoDB, DocumentDB, Cassandra, Coachbase, HBase, Redis, and Neo4j.

Pros:

1. Flexible data model:Whether document,graph, key-value, or wide-column, all offer a flexible data model making it easy to store and combine data of any structure and allow dynamic modification of the schema without downtime or performance impace.
2. Scalability and performance:This allows the database to scale out on commodity hardware deployed on-premises or in the cloud, enabling almost unlimited growth with higher throughput and lower latency than relational databases.
3. Always-on global deployments:They are designed to run across many nodes, including replication to automatically synchronize data across servers, racks, and data centers.

Cons:

1. It is a growing technology that will be mature in a few years, so, you have to bet for one and pray for it becoming successful.
2. Most of them do not have a declarative language, and those that have it lack the capability to optimise queries in a proper way.
3. You have to use an API.
4. Many lack native implementations for different platforms, except for Sparksee high-performance graph database, which has native implementations for Linux, Windows, MacOS, iOS, Android and BB10.

Reference

“<http://www.jamesserra.com/archive/2015/08/relational-databases-vs-non-relational-databases/>”