Neo4j

Mimanshu Maheshwari

October 3, 2024

Contents

List of Figures

Listings

Chapter 1

Introduction

1.1 Setting up Neo4j with docker

Using official docker setup link for neo4j.

```
1  docker pull neo4j
```

Listing 1.1: get neo4j image

```
1  docker run \
2     --publish=7474:7474 --publish=7687:7687 \
3     --volume=$HOME/neo4j/data:/data \
4     neo4j
```

Listing 1.2: Starting neo4j container

1.2 Creating Nodes

```
1  CREATE( var: label{key1:value1, key2:value2, key3:value3,...,keyn:valuen})
```

Listing 1.3: Create Node

- Properties of nodes present between { and }
- Node present between *and*
  1. var is the variable name such as John
  2. label is the entity type such as Customer
  3. key:value are the attributes of the properties such as `contact_num: 1234567890`

Example:

```
1  CREATE (b:Bank{name:"Oziku"})
```

Listing 1.4: Create Node Example

1.3 Creating Relationships

```
CREATE (node1) -[var:Rel_type{key1:value1, key2:value2, ...keyn:valuen}] -> (node2)
```

Listing 1.5: Create Relationships

Example:

```
CREATE (BG:Customer { cust_id: 675489, cust_name: "BlackGreyTechnologies",
    ↪ contact_num: 1298764592 })-
[:Owns]->(AcBG:Account { acc_num: 3498761, cust_id: 675489, type: "checking",
    ↪ balance:958990 })
```

Listing 1.6: Create Relation Example

Nodes, relationships can be created with/without label, properties Nodes or relationships can have single or multiple labels

1.4 Defining a constraint

Create a unique constraint on `cust_id` property of the Customer node as shown below:

```
CREATE CONSTRAINT ON (c:Customer) ASSERT c.cust_id IS UNIQUE
```

Listing 1.7: constraint

1.5 MATCH

Is used to search for a pattern. To return all the nodes created:

```
MATCH (n) RETURN n
```

Listing 1.8: Match syntax

```
MATCH (n:Bank)
RETURN n
```

Listing 1.9: Match example

```
MATCH (Customer { cust_name: 'BlackGreyTechnologies' })--(Account)
RETURN Account.acc_num, Account.balance
```

Listing 1.10: Match example

```
MATCH (city:City {name:"Raleigh"})
MERGE (state:State{name:"North Carolina"})
MERGE (city)-[:LOCATED_IN]->(state)
RETURN city,state
```

Listing 1.11: Match example

1.6 Optional Match

```
MATCH (c:Customer { cust_id: 675489 })
OPTIONAL MATCH (c)-[r:having] -()
RETURN c.cust_name, r.name
```

1.7 WHERE

```
MATCH (c:Customer), (a:Account)
WHERE a.type='checking'
RETURN c.cust_name,a.acc_num,a.balance
```

1.8 Merge

```
MERGE (c:Customer { cust_name: 'Charlie Sheen' })
RETURN c
```

1.9 UNION, UNION ALL

Union combines the results of multiple queries and removes duplicates whereas Union All performs the same operation but retains duplicates.

```
MATCH (c:Customer), (a:Account)
WHERE a.type='checking'
RETURN c.cust_name,a.acc_num,a.balance
UNION
MATCH (c:Customer { cust_name: 'BlackGreyTechnologies' })--(a:Account)
RETURN c.cust_name, a.acc_num, a.balance
```

NOTE: All sub queries must have the same column names.

1.10 ORDER BY, SKIP, LIMIT, DISTINCT

ORDER BY - specifies how the output of RETURN or WITH should be sorted. SKIP - defines from which record to start including the records in the output. LIMIT - constraints the number of output records. DISTINCT - retrieves only unique rows. The below query sorts the Customer node in the ascending order of its customer names, skips the first record and limits the output to only one record:

```
ORDER BY, SKIP, LIMIT, DISTINCT
```

To retrieve the unique relationships present in the database:

```
MATCH (n)-[r]-()
RETURN distinct type(r)
```

1.11 SET

It is used to update node labels and properties of nodes and relationships. For example, adding new properties such as email and country to the existing Customer BlackGreyTechnologies:

```
MATCH (c:Customer {cust_name: 'BlackGreyTechnologies'})
SET c.email= 'BGT@blackgrey.com', c.country = 'France'
RETURN c
```

Listing 1.12: Match example

1.12 FOREACH

It updates data within a list which can be components of a path* or result of an aggregation Path is a directed sequence of nodes and relationships. Assume you want to track funds transfer between two accounts suspected to be laundering funds illegally between intermediary accounts.

```
CREATE p = (:Account {acc_num:65178})-[:Funds_transfer]->(:Account
{acc_num:98567})-[:Fundstransfer]->(:Account{acc_num:46378})-[:Fundstransfer]->(:
    ↪ Account
{acc_num:95648})-[:Fundstransfer]->(:Account{acc_num:46897})
RETURN p

MATCH p = (:Account{acc_num:65178})-[*]->(:Account{acc_num:46897})
FOREACH (n IN nodes(p)| SET n.marked = "flaggedFraud")
```

Note: In the above code, [*] is used to define any number of intermediary relationships `acc_num=65178 to acc_num=46897` In the output shown below, you can observe that a new property `marked:"flaggedFraud"` has been added:

1.13 REMOVE

Is used to remove labels and properties of nodes and relationships.

To remove the email property from Customer having `cust_name=BlackGreyTechnologies`:

```
MATCH (c:Customer {cust_name: 'BlackGreyTechnologies'})
REMOVE c.email
RETURN c
```

1.14 DELETE

Is used to delete nodes, relationships or paths. Node cannot be deleted without deleting its associated relationships. Either delete relationships explicitly or use DETACH DELETE that is discussed next.

```
MATCH (n { acc_num: 65178 })-[r:Funds_transfer]->()
DELETE r
```

1.15 DETACH DELETE

Is used to delete nodes along with their relationships.

To delete Customer node with `name:"BlackGreyTechnologies"` and all its associated links:

```
MATCH (a {cust_name: "BlackGreyTechnologies"})
DETACH DELETE a
```

Delete all the nodes and relationships from the database:

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

1.16 Bulk Load of Data

Step 1: To load data from csv file, the below configuration property needs to be added to the neo4j.conf configuration file: `dbms.security.allow_csv_import_from_file_urls=true`

Step 2: Create Bank node, Customer node and ensure that the customer ID is unique as discussed previously.

Step 3: Load data from Customer.csv file to Customer node

```
load csv with headers from 'file:///C:/Users/Sahana_Basavaraja/Desktop/Customer.csv
    ↪ ' as cust
merge (c:Customer{cust_id:toInteger(cust.cid),cust_name:cust.cname,contact_num:
    ↪ toInteger(cust.phnum)})
```

Note: By default, the row retrieved from the file is always string, for example cust. You need to explicitly convert to the appropriate datatypes if required, for example `toInteger()` as shown above. Establish the relationship between Customer and Bank as `Customer_of`:

```
MATCH (c:Customer),(b:Bank) merge (c)-[b1:Customer_of]->(b)
MATCH (n) RETURN n
CREATE CONSTRAINT ON (a:Account) ASSERT a.acc_num IS UNIQUE

// load data present in account csv
load csv with headers from 'file:///C:/Users/Sahana_Basavaraja/Desktop/Account.csv'
    ↪  as acc
merge(a:Account{acc_num:toInteger(acc.acc_num)})
set
a.cust_id=toInteger(acc.cid),
a.balance=toInteger(acc.balance)
```

Note: Use SET to map columns of the file to properties of the node as illustrated above. Establish the relationship between Customer and Account as Owns:

```
load csv with headers from 'file:///C:/Users/Sahana_Basavaraja/Desktop/Account.csv'
    ↪  as acc
match(a:Account{acc_num:toInteger(acc.acc_num)}),(c:Customer{cust_id:toInteger(acc.
    ↪ cid)})
merge (a)<-[o:Owns]-(c)
```

```
load csv with headers from 'file:///C:/Users/Sahana_Basavaraja/Desktop/txn.csv' as
    ↪ txn
match (a:Account), (b:Account)
where a.acc_num=toInteger(txn.from_acc) AND b.acc_num=toInteger(txn.to_acc) AND
    ↪ toInteger(a.balance)>=toInteger(txn.amount_acc)
```

```
4 create (a)-[:Funds_transfer{txn_id:toInteger(txn.tid),from_acc:toInteger(txn.
      ↪ from_acc),to_acc:toInteger(txn.to_acc),amount:toInteger
5 (txn.amount_acc),txntime:txn.txntime}]->(b)
```