Neo4j

Steven Raaijmakers

March 2017

1 Inleiding

Naast de relationele database (Postgres) en document-based databases (CouchDB) bestaan er nog meer type databases, zoals de graph database. Ook dit type valt onder de NoSQL soorten, en is uniek in zijn soort. Het verschilt namelijk op veel vlakken van zijn collega's. Door deze grote verschillen is het importeren data voor andere type databases naar graph databases ingewikkeld.

2 Probleemstelling

Het importeren van bijvoorbeeld SQL data naar een graph database is lastig, omdat er gebruik gemaakt wordt van een compleet andere structuur. Relaties en informatie zijn gescheiden in een graph database, in tegenstelling tot bij vele andere databases, waar deze informatie zich vaak op dezelfde plek bevind.

Voor het importeren van een SQL database (zoals de musicbrainz database) zal daarom een speciaal script geschreven moeten worden, dat hier rekening mee houdt.

3 Theorie

Een graph database is een database waarin de tabellen nodes zijn. Hierin kunnen de kolommen van de tabel weergegeven worden als waardes van de node. Relaties ontstaan door meerdere nodes te verbinden via edges. Informatie vanuit de graph database wordt opgevraagd door gebruik van semantische queries. De nodes bevatten dus de informatie, en de edges bevatten de relaties tussen nodes.

3.1 Neo4j

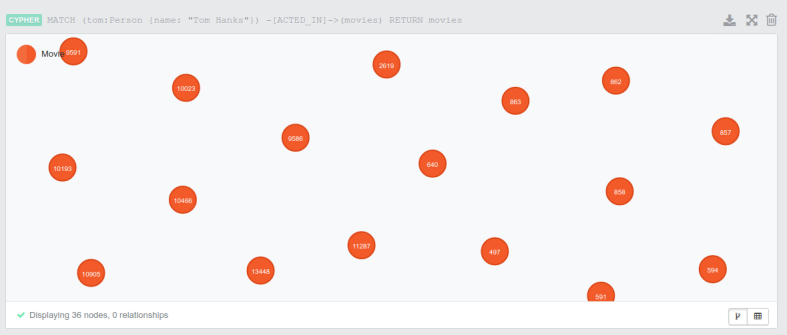
Neo4j, geschreven in Java, is een graph database, ontwikkeld door Neo Technology. In Neo4j wordt gebruik gemaakt van het ACID-principe, en is de meest gebruikte graph-database software. Queries worden geschreven in de Cyper Query Language, en kunnen ingevoerd worden in het webinterface, of door het gebruik van libraries in bijvoorbeeld Python.

4 Experimenten

Na het downloaden van Neo4j (volgens de syllabus), wordt een sample dataset van de IMDB ingelezen in Neo4j via `neo4j.com/developer/movie-database/`.

Na het installeren van de IMDB kan aan de database opgevraagd worden in welke films de acteur "Tom Hanks" voorkomt:

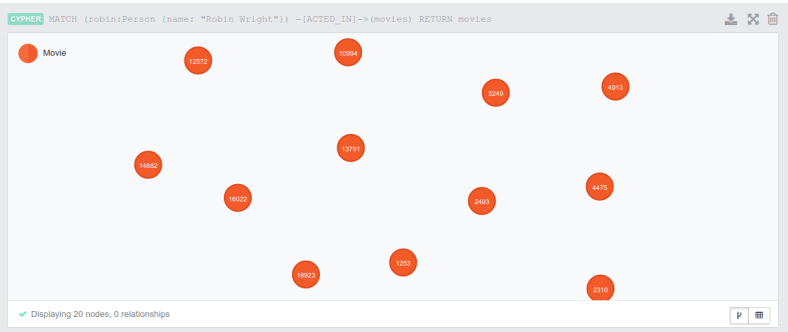
```
MATCH (tom:Person {name: "Tom Hanks"}) -[ACTED_IN]->(movies)
RETURN movies
```



In de figuur zijn de nodes gelabeld met hun ID. Wanneer je hierop klikt zie je meer informatie over de node, zoals de titel. Als geswitched wordt naar de tabel-weergave zal er per node alle beschikbare informatie verschijnen.

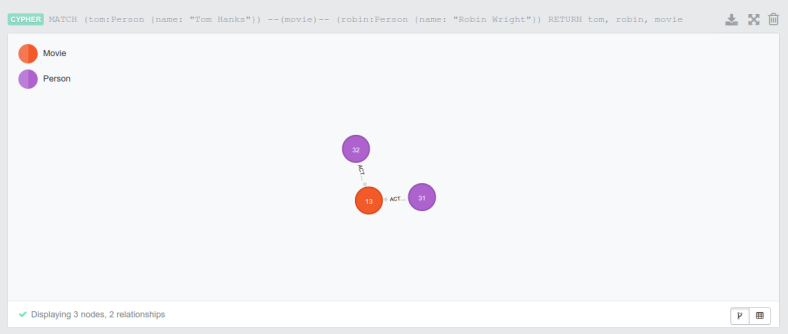
Eenzelfde query kan worden uitgevoerd voor de actrice "Robin Wright":

```
MATCH (robin:Person {name: "Robin Wright"}) -[ACTED_IN]->(movies)
RETURN movies
```

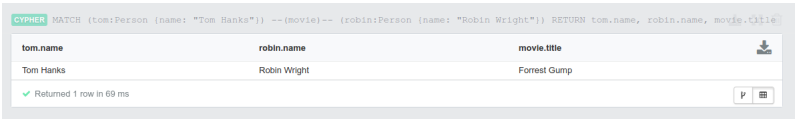


Ook kan gekeken worden of er een bepaalde relatie bestaat tussen deze acteur en actrice:

```
MATCH (tom:Person {name: "Tom Hanks"})
--(movie)--
(robin:Person {name: "Robin Wright"})
RETURN tom.name, robin, movie
```

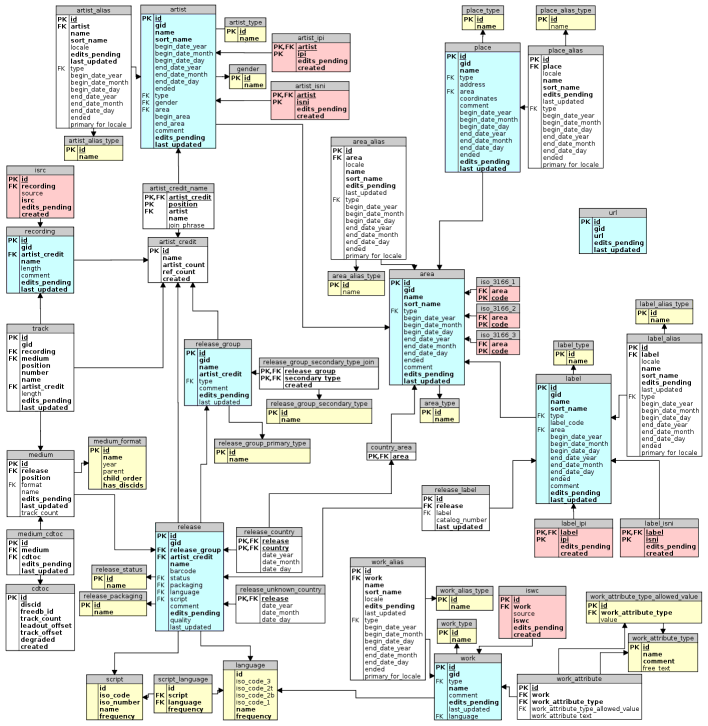


Het resultaat laat zien dat Tom Hanks en Robin Wright één relatie hebben in de database. Wanneer geswitched wordt op de tabel-weergave, wordt duidelijk dat het de film “Forest Gump” betreft, waarin beiden acteurs een rol speelden:



5 Onderzoek

Het doel is het implementeren van het volgende schema in Neo4j:



Figuur 1: Musicbrainz database schema

Het importeren van de gehele musicbrainz database (MBDB) is al eerder gedaan, via de "sql2graph" van *redapple*: `github.com/redapple/sql2graph`. Hierin wordt gebruik gemaakt van de library py2neo in python. Deze is te downloaden via pip:

```
sudo pip install py2neo
```

Bij het gebruik van het script van redaple wordt de gehele MBDB overgezet. Omdat het de bedoeling is dat slechts het schema overgenomen wordt, is gekozen om zelf een script te schrijven dat hierin voorziet. Wel is een deel van het MBDB schema terug te vinden in het script: `github.com/redapple/sql2graph/blob/master/musicbrainz_schema.py`.

Hierin staan alle tabel-namen met hun verplichte kolommen gedefinieerd. Van deze file is gebruik gemaakt zodat de tabel-namen en relaties uit figuur 1 niet meer handmatig ingevoerd hoefden te worden.

5.0.1 Structuur

Elke tabel wordt een *entity* genoemd. De entity bestaat uit een *name* en enkele *properties* (oftewel de attributen). Een enkele property bestaat uit een *type* en een *column*. Een type kan bijvoorbeeld aangeven of de kolom een foreign key bevat.

Een kolom bestaat op zijn beurt uit een “name”, en wanneer het een foreign key betreft, bevat de kolom ook informatie over naar welke *foreign column* deze kolom verwijst.

Naast de properties kan een entity ook een of meerdere relations bevatten. Hierin staat een naam die de relatie omschrijft, de begin positie en de eind positie van de relatie.

De entity “Artist” heeft bijvoorbeeld de volgende relatie:

```
Relation(
    'FROM_AREA',
    start=Reference('artist', Column('id')),
    end=Reference('area', Column('area')),
)
```

De relatie “FROM_AREA” verwijst in dit geval van de artist-node naar de area-node.

Door het gebruik van de vele objecten staat er een genestelde structuur:

- Entity
  - Name
  - Properties:
    - type
    - name (+ eventuele verwijzing)
  - Relations:
    - name
    - start
    - end

Dat ziet er in het de file van redapple als volgt uit:

```
Entity('artist_alias',
    [
        IntegerProperty('pk', Column('id')),
        Property('name', Column('name')),
        Property('type', Column('type',
            ForeignColumn('artist_alias_type', 'name', null=True))
        ),
    ],
    [
        Relation(
            'HAS_ALIAS',
            start=Reference('artist', Column('artist')),
            end=Reference('artist_alias', Column('id')),
        ),
    ]
)
```

Met deze informatie is gekozen een structuur te maken als volgt: elke tabel krijgt zijn eigen node, de table-node. Alle kolommen uit dezelfde tabel worden ook een node in de graph, de attribute-nodes. Een table-node verwijst vervolgens naar meerdere attribute-nodes om aan te geven welke attributen een tabel bevat. Deze relatie wordt aangegeven als "has attribute". Hiernaast bestaan er ook relaties tussen twee tabellen.

5.1 Script

Zie script.py

In python kan verbinding gemaakt worden met de Neo4j database:

```
from py2neo import Graph
graph = Graph("localhost:7474/db/data")
```

Ons script zal alle entities uitlezen, en voor elke unieke entity een nieuwe node maken met de bijbehorende naam:

```
for entity in schema:
    add_node(entity, type=table)
```

Vervolgens leest het script van alle entities alle properties uit en maakt voor elke unieke property een nieuwe node:

```
for entity in schema:
    for property in entity.properties:
        add_node(property, type=attribute)
```

Nu alle nodes aangemaakt zijn, zal het script relaties gaan aanmaken. Er wordt hierbij onderscheid gemaakt tussen twee soorten relaties, de "has attribute"-relatie: de relatie tussen een table-node; en een attribute-node: en de relatie tussen twee table-nodes.

Het script maakt eerst alle relaties tussen de table-nodes en hun attribute-nodes:

```
for entity in schema:
    for property in entity.properties:
        u1 = graph.find(entity)
        u2 = graph.find(property)
        if property.fk:
            graph.create(relation(u1, 'has attribute', u2,
                         fk=property.fk)
        else:
            graph.create(relationship(u1, 'has attribute', u2))
```

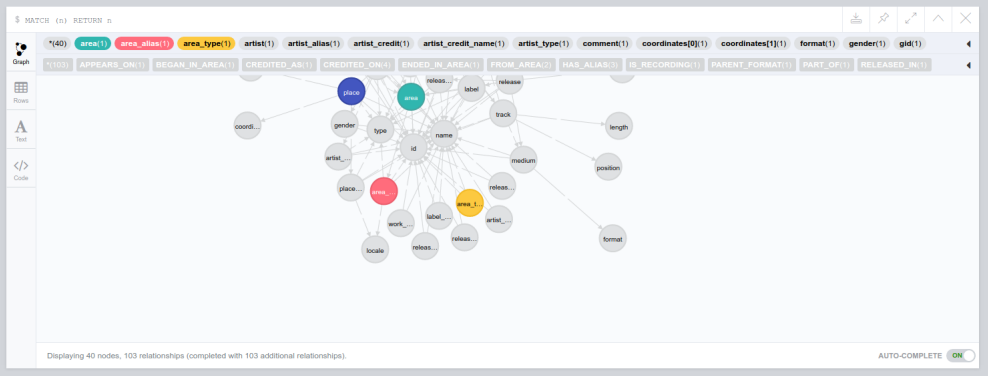
Wanneer een bepaalde property (attribute) naar een foreign key verwijst, zal deze informatie mee worden genomen bij het creëren van de relatie, zodat deze informatie niet verloren gaat. Dit kan handig zijn wanneer er straks data in de graph komt te staan.

Tot slot worden de relaties tussen twee table-nodes gemaakt:

```
for entity in schema:
    if entity.relations:
        for relation in entity.relations:
            u1 = graph.find_one(relation.start)
            u2 = graph.find_one(relation.end)
            graph.create(relation(u1, relation.name, u2))
```

6 Resultaat

De gehele database ziet er als volgt uit:

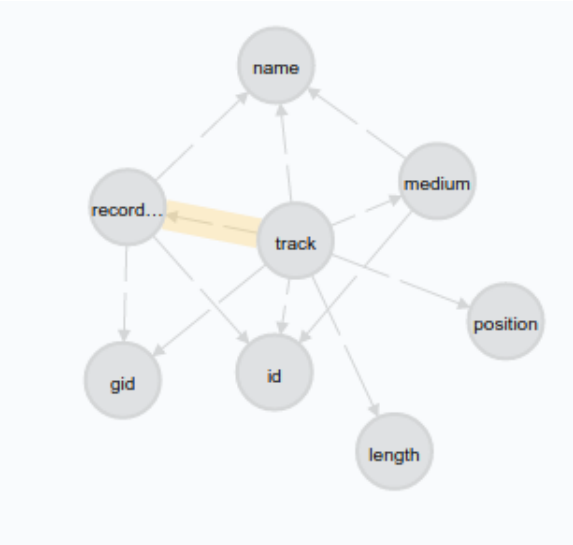


Hierin is te zien dat de attribute-node “id” relatief veel inkomende relaties heeft. Dit komt overeen met het MBDB schema uit figuur 1, waarin te zien is dat elke tabel een id heeft.

Een deel van de informatie over de track-tabel, kan in de graph worden opgevraagd door:

```
MATCH (n)-->(b)
WHERE n.name="track"
RETURN n, b
```

En dat ziet er als volgt uit:



In het schema is de “track”-tabel/node te zien, en al zijn uitgaande relaties. Sommige van deze relaties betreffen relaties tussen table-nodes en de attribute-nodes, zoals de verwijzing van track naar gid en id. Ook zijn andere type relaties te vinden: tussen twee table-nodes, zoals bijvoorbeeld de relatie tussen “track” en “recording”. Als op de relatie geklikt is, wordt de naam “IS_RECORDING” weergegeven.

7 Conclusie en Discussie

Het importeren van het musicbrainz schema naar Neo4j is gedaan door gebruik te maken van de python module py2neo. Om dit schema te importeren is gebruik gemaakt van een stuk code van redapple. Hiervan is alleen de informatie over het schema (tabel-namen, kolom-namen) gebruikt. Om deze informatie in te lezen is een script geschreven, dat de informatie omzet naar python objecten. Vervolgens kan het script deze python objecten in Neo4j zetten.