CouchDB: importeren van RDF bestanden

Steven Raaijmakers

February 2017

1 Inleiding

Relationele databases zoals MySQL en PostgreSQL genieten veel bekendheid. Er bestaan echter ook andere soort databases, die men NoSQL databases noemt. Zoals de naam al doet vermoeden wordt hierbij geen gebruik gemaakt van SQL.

Een voorbeeld van een NoSQL database is CouchDB. Couchdb is een open-source database die zich focust op gebruiksvriendelijkheid en is gebouwd op een architectuur die volledig op het web rust.

2 Probleemstelling

RDF bestanden worden de standaard voor het opslaan van data op het internet. Zo bestaat ook de Britse Nationale Bilbiotheek database (BNB) compleet uit .rdf bestanden, die samen meer dan 10 GB aan geheugen in beslag nemen.

CouchDB heeft echter een andere structuur dan RDF bestanden, en daarom zal een dergelijk bestand eerst qua structuur moeten worden aangepast om zo de informatie juist weer te geven in de CouchDB. In CouchDB is er geen standaard mogelijkheid om RDF bestanden in te lezen.

Wanneer de boeken uit BNB in een CouchDB worden gezet is het daarom nodig om zelf een script te zetten die deze .rdf bestanden om zet naar bruikbare informatie in een CouchDB.

3 Theorie

3.1 CouchDB

CouchDB is een document-oriënted database. Zo′n database bestaat uit een tabel met daarin verschillende documenten. Een document bevat op zijn beurt een verzameling van key-value-eenheden, waarbij verder geen vaste structuur aanwezig hoeft te zijn. Dit betekent dat men niet verplicht is om bepaalde key-velden in te vullen bij het toevoegen van een nieuw document. In één database kunnen documenten dus verschillende structuren hebben, in tegenstelling tot de PostgreSQL database, waar men per tabel verplicht is vaste kolommen bepaalde waardes te geven.

CouchDB is geschreven in Erlang en werkt met opslag van JSON-documenten. Het is gebouwd om zijn gehele architectuur "op het web te laten rusten"[2]. Iets toevoegen aan CouchDB kan door het gebruik van de webinterface Futon, maar ook door gebruik van cURL. Hiernaast biedt CouchDB in tegenstelling tot veel andere NoSQL-databases (zoals MongoDB) ondersteuning voor ACID-transacties.

Een ander belangrijk feature van CouchDB zijn de revisions. Bij het veranderen van een document in CouchDB wordt de oude versie volledig vervangen door een nieuwe, zonder hierbij de oude versie weg te gooien. Hierdoor zijn veranderingen aan documenten te herstellen.

3.2 RDF

Resource Description Framework (RDF) is een standaard, gecreëerd door W3C, en is manier om data te beschrijven. Met RDF wordt data door een driedelige structuur (n-triple) weergegven:

- Subject: de resource die omschreven wordt
- Predicate: geeft aan welk kenmerk van de resource omschreven wordt
- Object: de waarde van het kenmerk

Een voorbeeld van een object in RDF n-triples ziet er als volgt uit:

```
<http://example.org/#m> #type <http://xmlns.com/foaf/0.1/Person> .
<http://example.org/#m> #label "Michael" .
```

De eerste twee links omschrijven de resource, maar omdat de links identiek zijn verwijzen ze naar dezelfde resource, en dus omschrijven ze samen één en hetzelfde object:

```
<http://example.org/#m> = {type: person, label: michael}
```

4 Onderzoek

CouchDB accepteert JSON bestanden als input en daarom is een logische stap om het geleverde RDF bestand om te zetten naar een JSON object. Voor het onderzoek werd er (zoals gesuggereerd door de sllabus) gebruik gemaakt van de programmeertaal Python.

4.1 Python

Om een .rdf bestand in te lezen en als daadwerkelijk RDF bestand te representeren (in formaat van n-triples dus), kan gebruik gemaakt worden van de Python library *rdflib*. In rdflib bestaat de mogelijkheid .rdf bestanden te serializen naar JSON bestanden, wat ideaal is voor CouchDB. Dit is te downloaden via:

```
sudo pip install rdflib
sudo pip install rdflib-jsonld
```

Om het bestand nu uit te lezen, werd de volgende code gebruikt:

```
g = Graph()
g.parse("bnbbasic.rdf")
g = g.serialize(format="json/ld")
```

Het probleem hierbij is echter dat het .rdf bestand nu gepresenteerd wordt als een graph in JSON. In deze graph ontstaan relaties door te verwijzen naar de IDs van nodes. Dit is niet ideaal voor CouchDB, omdat dit een Document-DB is, en geen Graph-DB.[1]

De RDF structuur in het bestand wordt omgeschreven zodra deze wordt geparsed door rdflib. Hierdoor verandert de structuur nadelig omdat deze zich in eerste instantie al in geschikt vorm bevind voor CouchDB - zonder relaties tussen verschillende documenten. Er werd daarom gekozen om het .rdf bestand niet meer uit te lezen naar n-triples, maar regel voor regel te parsen. Op aanraden van een medestudent werd daarom overgestapt naar de programmeertaal Ruby.

4.2 Ruby

In het script wordt het gehele bestand regel voor regel uitgelezen. Regels die "xml" en "rdf:rdf" bevatten worden overgeslagen, aangezien deze verder geen object (behalve het gehele document) beschrijven.

Vervolgens worden alle ingelezen regels onthouden totdat de eerst volgende regel een lege betreft. In het RDF document wordt er tussen verschillende boeken onderscheid gemaakt door middel van meerdere blanco regels. Wanneer een lege regel wordt aangetroffen worden alle eerder gevonden regels doorgestuurd naar een functie die de zinnige informatie uit deze regels onttrekt.

4.2.1 Parsen

Bij het parsen van de lines wordt een hash gemaakt (een soort python dictionary) met daarin alle (key, value pairs) van één boek. De keys ontstaan door de tags van het formaat $< TAG : key >$ te splitten, waardoor je enkel de het tweede deel van de tag overhoudt als key. De value bestaat uit de informatie die zich binnen de tag bevindt. Er wordt hierbij gekozen enkel de buitenste tag mee te nemen als key, omdat het RDF formaat verder veel irrelevante informatie voor de CouchDB bevat:

```
<dcterms:contributor>
    <rdf:Description>
           <rdfs:label>Cronquist, Arthur</rdfs:label>
           <rdf:type rdf:resource="xmlns.com/foaf/0.1/Person" />
    </rdf:Description>
</dcterms:contributor>
```

Wordt dus:

```
"contributor" = "Cronquist, Arthur"
```

Hierdoor gaan de rdf:description en rdf:type verloren, maar blijft er zinnige informatie over.

4.2.2 Hash

Nu de (key, value)-pairs clean zijn, kunnen ze worden toegevoegd aan de hash. Eerst wordt er gekeken of de betreffende key al een entry heeft in de hash. Is dit niet het geval dan wordt het gehele (key,value)-pair aan de hash toegevoegd. Komt de key al wel voor in de hahs zal er worden gekeken of de key enkel één value waarde heeft. Als dat zo is zal deze oude value samen met de nieuwe value een lijst vormen. Voor een contributor kan dat er als volgt uitzien.

```
contributor = ["Cronquist, Arthur", "Isely, Duane, 1918"]
```

4.2.3 Couchrest

Nu een boek uit de BNB als een volledig hash object omschreven wordt, is het zaak deze toe te voegen aan de database. Dit kan door middel van couchrest, een library voor ruby die er voor zorgt dat er CouchDB functies in ruby mogelijk zijn. Deze is te verkrijgen via:

```
sudo gem install couchrest
```

Er kan nu als volgt verbinding worden gemaakt met de CouchDB:

```
@db = CouchRest.database!("http://localhost:5984/ruby_books")
```

Het uploaden van documenten gaat door het correcte hash object op te slaan in de database:

```
db.save_doc(HASH_OBJECT)
```

4.3 MapReduce

De BNB database bestaat uit 10 bestanden, waarbij een enkel BNB bestand bestaat uit ongeveer 1 gb, wat neerkomt op een kleine 300.000 verschillende boeken en een totaal van 3 miljoen boeken. De MapReduce functie is daarom erg belangrijk, omdat hiermee specifieke informatie uit deze grote verzameling kan worden onttrokken.

4.3.1 Authors per Country

Om bijvoorbeeld de "authors" per "country" te verkrijgen kan een MapReduce functie geschreven worden. Hierbij wordt "country" als "placeOfPublication" gezien. De placeOfPublication bevat echter de MARC codes voor landen. Zo slaat xxu bijvoorbeeld op Amerika.

Een boek bevat meestal slechts één country, maar kan meerdere schrijvers hebben in de velden van "creator" en "contributors". In de map functie wordt daarom gekeken of een boek een creator heeft, is dit het geval dan wordt er geemit met (language, author). Als een boek (ook) contributors bevat, wordt er per contributor een nieuwe emit gestuurd: (lanuage, contributor). Uit een enkel boek met 1 creator en 2 contributors zulen er dus 3 emits worden gestuurd.

```
function(doc) {
    if (doc.placeOfPublication){
        if(doc.creator){
            emit(doc.placeOfPublication, doc.creator);
        }
        if(doc.contributor){
            (doc.contributor || []).forEach(function(contributor) {
                        emit(doc.placeOfPublication, contributor);
                });
        }

    }
}
```

In de reduce-functie worden deze emits samengevoegd op basis van hun key. Door nu het aantal unieke elementen in een value-field-lijst te tellen verschijnt het aantal unieke schrijvers dat in dit land een boek gepubliceerd heeft.

```
function(key, values, rereduce){
       var tmp = [];
       (values || []).forEach(function(val) {
              if(tmp.indexOf(val) == -1){
                     tmp.push(val);
              }
       });
       return tmp.length;
}
```

4.3.2 Languages per Author

Een document kan meerdere talen bevatten, maar slechts een creator. Om te kijken of een document enkel één taal bevat, of juist meerdere, kan gekeken worden of het veld een string of een array bevat. Wanneer het een string is zal

deze meteen worden ge-emit als (creator, language). In het geval van een array zal elk item ge-emit worden als (creator, language):

```
function(doc) {
    if (doc.language){
        if(doc.creator){
                if(typeof doc.language === 'string'){
                        emit(doc.creator, doc.language);
                }
                else{
                        (doc.language || []).forEach(function(lan) {
                           emit(doc.creator, lan);
                         });
                }
        }
    }
}
```

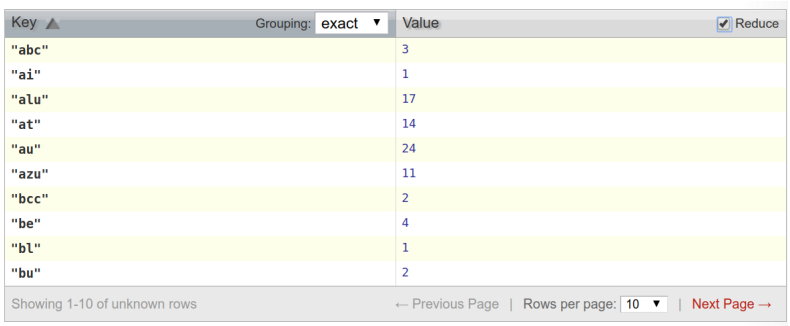
Uiteindelijk zal er per author zo een lijst als values ontstaan met alle talen waarin hij/zij boeken geschreven heeft. Hieruit moeten enkel de distinct waardes gehaald worden om zo een resultaat weer te geven:

```
function(key, values, rereduce){
        var tmp = [];
        (values || []).forEach(function(val) {
                if(tmp.indexOf(val) == -1){
                        tmp.push(val);
                }
        });
        return tmp;
}
```

5 Resultaat

Het geschreven ruby script is redelijk gemiddeld qua snelheid. Het doet over het inladen van $BNBBasic\_201701\_f01.rdf$, (1.1 GB), ongeveer 2 uur. Dit lijkt traag, maar aangezien het script 260.622 verschillende boeken laadt, komt het gemiddelde uit op een kleine 40 boeken per seconde. Voor het gemak is alleen *BNBBasic_201701_f01.rdf* ingeladen.

In figuur 1 zijn de unieke authors per country te zien, gesorteerd op de MARC code van het bijbehorende land. De MARC codes kunnen via `http://id.loc.gov/vocabulary/countries` gekoppeld worden aan een land.

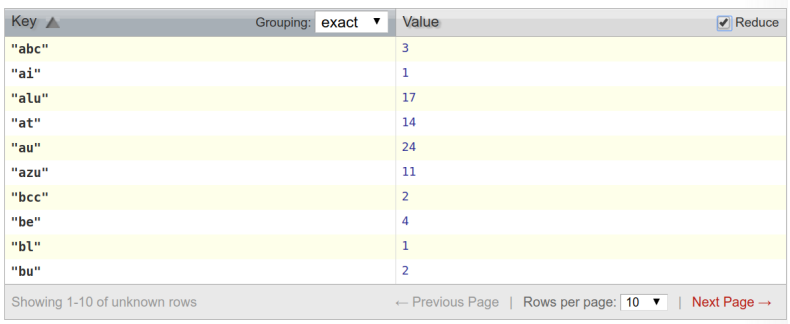


Key	Value
"abc"	3
"ai"	1
"alu"	17
"at"	14
"au"	24
"azu"	11
"bcc"	2
"be"	4
"bl"	1
"bu"	2

Figuur 1: Unieke authors / country

Ook het laden van de resultaten uit figuur 1 kost wat tijd, maar alsnog sneller dan verwacht wanneer men een SQL database gewend is. Het mappen en reduce van deze gegevens kost zo ongeveer 2 minuten, wat vrij rap is voor meerdere schrijvers per 300.000 boeken.

De query voor de uieke talen per author kost nog meer tijd, omdat hier lijsten moeten worden opgebouwd in plaasts van geteld te worden:



Key	Value
"abc"	3
"ai"	1
"alu"	17
"at"	14
"au"	24
"azu"	11
"bcc"	2
"be"	4
"bl"	1
"bu"	2

Figuur 2: Unieke talen / author

6 Conclusie en Discussie

Bij dit onderzoek is er veel tijd verloren gegaan aan het programmeren in Python. In eerste instantie werd hierin geprogrammeerd door de suggestie van de syllabus. Verder was het ook niet meteen duidelijk dat een .rdf bestand iets anders is dan een bestand bestaande uit meerdere n-triples.

Toen eenmaal gekozen was om het bestand niet te parsen naar n-triples, maar om de structuur te behouden kwamen de resultaten snel naar boven.

Hoewel .rdf een standaard wordt is het niet aan te raden gebruik te maken van dit soort bestanden wanneer je met CouchDB werkt. Veel informatie die in een .rdf bestand (zoals het predicate) is onzinnig voor een CouchDB, en daarom kost het veel werkt een script te maken die toch iets zinnigs kan breien bij de verandering van deze structuur.

Het gebruik van MapReduce in CouchDB is ook niet al te simpel omdat er niet makkelijk gedebugged kan worden. Doordat queries enkele minuten kosten

om te laden wordt dit proces bemoeilijkt. Het resultaat van de MapReduce-functie is echter wel boven verwachting snel.

References

[1] mhausenblas. *Mapping RDF to JSON documents*. `https://github.com/mhausenblas/ld-in-couch/wiki/Mapping-RDF-to-JSON-documents`. 2014.

[2] Wikipedia. *CouchDB*. `https://en.wikipedia.org/wiki/CouchDB`. 2017.