Onderzoeksrapport Spotitube

Leander ten Hoedt
634442
ITA-OOSE-A-f
Michel Portier
2-4-2021
v1

Inhoud

Inleiding

In dit verslag ga ik vertellen over een onderzoek op de applicatie spotitube. Dit onderzoek heb ik onder andere gedaan om erachter te komen of een onderdeel van de applicatie geschikt is om te vervangen met een andere techniek. Dit onderzoek is opgedeeld in 3 onderzoeken; literatuur onderzoek, labonderzoek en werkplaatsonderzoek. Bij het literatuuronderzoek ga ik allerlei aspecten van de nieuwe techniek bekijken. Daarna in het labonderzoek ga ik een kleine test applicatie maken om de nieuwe techniek uit te proberen. Als laatst in het werkplaats onderzoek ga ik de nieuwe techniek toevoegen aan de Spotitube applicatie.

Onderzoeksvraag

Spotitube gebruikt op dit moment een relationele ofwel een SQL database. De mogelijkheid is er om gebruik te maken van een non-relationele database ofwel een NOSQL database zoals bijvoorbeeld mongoDB. Dit wekt de vraag:

Hoe implementeer ik een niet-relationele db in spotitube?

Deelvragen

- Wat is een relationele database?
- Wat is een non-relationele database?
- Welke type non-relationele database is het meest geschikt?
- Hoe verbind ik de NoSQL database in java?
- Hoe voer ik queries uit op de NoSQL database?

Doelstelling

Doelstelling van deze onderzoeksopdracht is om een deel van de Spotitube applicatie te vervangen door een alternatieve technologie. Deze technologie kan ingepast worden in de bestaande architectuur, maar kan ook een grote architecturale wijziging vereisen.

Literatuuronderzoek

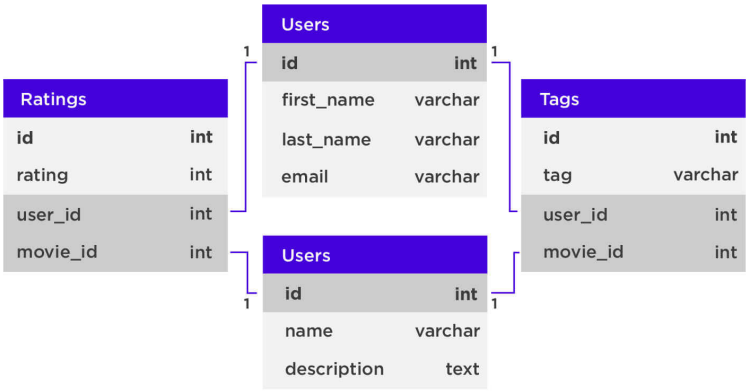
2.1 Onderzoeksgebied

Het onderzoek van het implementeren van een non-relationele database gaat uiteraard over de database maar ook over de implementatie met de Spotitube backend waarbij de DAO's moeten worden aangepast.
In dit hoofdstuk ga ik een aantal aspecten bekijken over beide database types daarna ga ik een aantal aspecten bekijken van een NoSQL database.

2.2 Onderzoek

2.2.1 Relationele database

Een relationele database is een database die rijen bevat (ook wel tupels genoemd) waar relaties tussen liggen. Dit wil zeggen dat rijen relaties kunnen hebben ofwel gelinkt kunnen zijn aan andere rijen door middel van een kolom in beide rijen, doordat deze relaties er kunnen zijn heet het een relationele database.
De meest voorkomende taal die gebruik maakt van een relationele database is SQL, de overige talen gebruiken eigenlijk altijd een SQL querytaal.

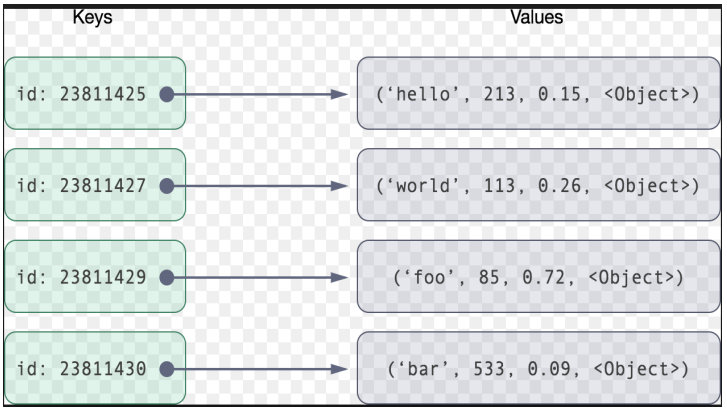


2.2.2 Niet-relationele database

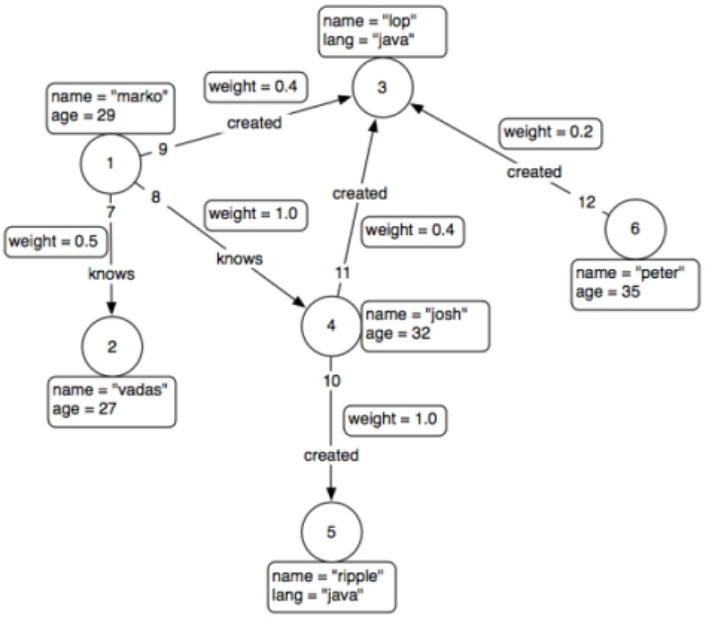
Een niet relationele database ook wel een NoSQL database genoemd, bestaat uit een 4 soorten databases namelijk:

1. key-value store
2. Graph store
3. Column store
4. Document store

Key-value store slaat data op als associative array dat wil zeggen zeggen een lijst met koppels van naam en waarde.



Graph store slaat data op als zogeheten ‘nodes’ waarbij het vooral draait om de onderlinge relaties. Dit wordt vaak weergeven in de vorm van een kaart zoals je hier kunt zien.



Column store slaat data op in blokken van kolommen die aan elkaar gelinkt kunnen worden door waarden uit die kolommen. Waarbij elke kolom belangrijk is en het minder belangrijk is dat ze bij elkaar horen.

Column-oriented

Name	ID
John	001
Karen	002
Bill	003

Grade	ID
Senior	001
Freshman	002
Junior	003

GPA	ID
4.00	001
3.67	002
3.33	003

Document store slaat data op als een object (vaak als JSON of XML) waarbij elk object onderscheiden wordt een door een unieke sleutel.

```
{
   "id": "Customer:1",
   "firstName": "John",
   "lastName": "Wick",
   "age: 25,
   "address": {
        "country": "US",
        "city": "New York",
        "state": "NY"
        "street": "21 2nd Street",
      },
   "hobbies": [ Football, Hiking ],
   "phoneNumbers": [
      {
        "type": "Home",
        "number": "212 555-1234"
      },
      {
        "type": "Office",
        "number": "616 565-6789"
      }
    ]
}
```

2.2.3 Welke type NoSQL is het meest geschikt?

De data van spotitube bestaat uit het volgende:

- User
    - username
    - password
    - token
- Playlist
    - naam
    - tracks
- Tracks
    - titel
    - etc
- Album
    - naam
    - tracks

Waar rekening mee gehouden moet worden is het volgende

- Een user heeft niet per se bezit over een playlist.
- Tracks kunnen bestaan zonder in een playlist te zitten dat wil zeggen dat playlist geen bezit heeft over tracks
- Een album kan bezit hebben over een track. maar een track kan ook zonder album bestaan.

Deze contraints zorgen ervoor dat **key-value** af valt omdat niet elke key volledig uniek is er moet values kunnen worden hergebruikt. **Column** store valt ook af omdat die data te los is. Alle kolommen bestaan los wat niet zo handig omdat een aantal kolommen altijd bij elkaar moeten horen.
Hierdoor blijven **Document-** en **Graph store** over.
Bij **document store** gaat het heel erg om wie er bezit heeft over de data wat bij Spotitube niet helemaal mooi past omdat bijvoorbeeld een playlist van iemand is maar andere mensen moeten hem ook kunnen zien dus het moet duidelijk zijn dat de user eigenaar is van de playlist maar de playlist moet wel op zichzelf kunnen bestaan, ook kunnen tracks bestaan zonder playlist maar ook in een playlist zitten en een track kan in meerdere playlists zitten.
Bij **graph store** zijn de verbindingen heel belangrijk wat heel goed past bij Spotitube; een user bezit over een playlist maar de playlist bestaat los van de user, een track bestaat los maar kan verbonden worden aan meerdere playlists en een track kan uit een album zijn of los bestaan. Hierdoor kies ik ervoor om een graph store te gaan gebruiken.
Ik heb gekozen om neo4j te gaan gebruiken als database omdat dat de meest populaire graph database is en omdat deze database goed werkt met java. De query taal die gebruikt wordt voor neo4j heet cypher.

2.3 Belangen

Het wisselen naar een NoSQL database heeft als voordeel dat NoSQL een stuk sneller is dan SQL. Wat belangrijk is omdat er veel requests worden verstuurd. Daarnaast is graph store meer geschikt bij Spotitube dan SQL omdat er veel relaties tussen de data zijn.

2.4 Design patterns

Aan het design pattern wordt niks veranderd; de data wordt nog steeds opgehaald vanuit een DAO.

Experiment

In dit hoofdstuk staan de twee onderzoeken die ik heb gedaan om te testen hoe ik NoSQL database kan toevoegen aan Spotitube.

3.1 Het labonderzoek

3.1.1 Beschrijving

Allereerst het labonderzoek in dit onderzoek ga ik een hele kleine test applicatie maken om te testen hoe ik een NoSQL database aanmaak en daar data uit haal / in stop in java.

Het verbinden van een neo4j database kan op een aantal manieren maar ik heb gekozen om het te doen via een jdbc driver omdat we dat al gebruikte waardoor er in de code weinig veranderd. De dependency ziet er als volgt uit:

```
<dependency>
    <groupId>org.neo4j</groupId>
    <artifactId>neo4j-jdbc-driver</artifactId>
    <version>4.0.1</version>
</dependency>
```

Dus het uitvoeren van queries gaat zoals het al ging met de SQL database; eerst de query preparen, eventueel parameters zetten en als laatst executen.

In de test roep ik een aantal query's aan; als eerste query maak ik een user en een playlist aan die verbonden zijn via de relatie "owns".

```
String queryInsert = "CREATE (user:Users {username:\"henk\", password: \"henk\", token: \"1234-1234-1234\"} ) "
                   + "-[:owns]-> (playlist:Playlists {name: \"SRV\"}) "
                   + "RETURN user, playlist";
PreparedStatement stmt = con.prepareStatement(queryInsert);
stmt.executeUpdate();
```

Daarna maak ik nog een playlist aan die ik koppel aan de user.

```
String queryAddPlaylist = "MATCH (user:Users) "
                        + "WHERE user.username='henk' "
                        + "CREATE (user)-[:owns]->(playlist:Playlists {name: \"kaas\"}) "
                        + "RETURN user";
PreparedStatement stmt2 = con.prepareStatement(queryAddPlaylist);
stmt2.executeUpdate();
```

Waarna ik een aantal tracks aanmaak en die koppel aan 1 playlist.

```
for ( int i = 0; i < 10; i++ ) {
    String sql = "MATCH (pl:Playlists) "
               + "WHERE pl.name='kaas' "
               + "CREATE (pl)-[:contains]->(track:Tracks {name: \"track" + i + "\"}) "
               + "RETURN pl";
    PreparedStatement s = con.prepareStatement(sql);
    s.executeUpdate();
}
```

Als laatste haal ik de user met playlists en tracks op.

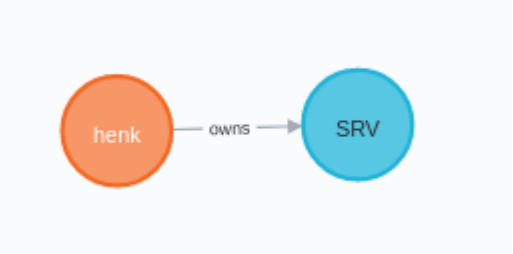
```
String queryRetrieve = "MATCH (user:Users)-[:owns]->(playlists:Playlists) "
                     + "OPTIONAL MATCH (playlists)-[:contains]->(tracks:Tracks) "
                     + "with user, playlists, collect(tracks) AS ts "
                     + "with user, collect({name: playlists.name, tracks: ts}) AS playlists "
                     + "WHERE user.username='henk' "
                     + "RETURN user.username, {playlists: playlists} AS playlists";
PreparedStatement stmtRetrieve = con.prepareStatement(queryRetrieve);
ResultSet resultSet = stmtRetrieve.executeQuery();
while (resultSet.next()) {
    System.out.println(resultSet.getString("user.username"));
    String jsonString = resultSet.getString("playlists");

    JSONArray playlists = new JSONObject(jsonString).getJSONArray("playlists");

    for ( int i = 0; i < playlists.length(); i++) {
        JSONObject playlist = playlists.getJSONObject(i);
        JSONArray tracks = playlist.getJSONArray("tracks");

        System.out.println( playlist.getString("name") );

        for ( int j = 0; j < tracks.length(); j++) {
            JSONObject track = tracks.getJSONObject(j);
            System.out.println(track.getString("name"));
        }
    }
}
```

Wat er als volgt uitziet in de console

```
henk
SRV
kaas
track9
track8
track7
track6
track5
track4
track3
track2
track1
track0
```

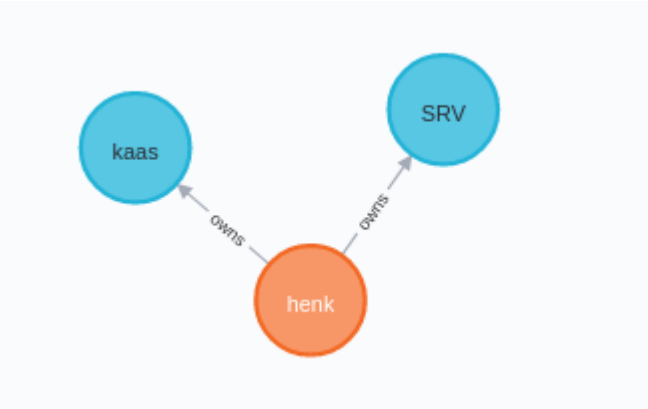
3.1.2 Resultaten

Stap 1 geeft het volgende resultaat in de database:



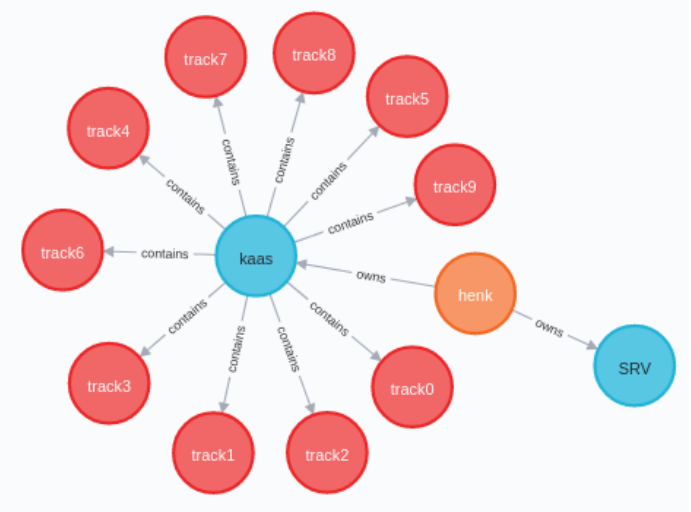
Je zit hier dus de user node die een playlist node heeft.

Stap 2 geeft het volgende resultaat in de database:



Je zit hier dus dat er nog een playlist is aangemaakt die ook van de user is.

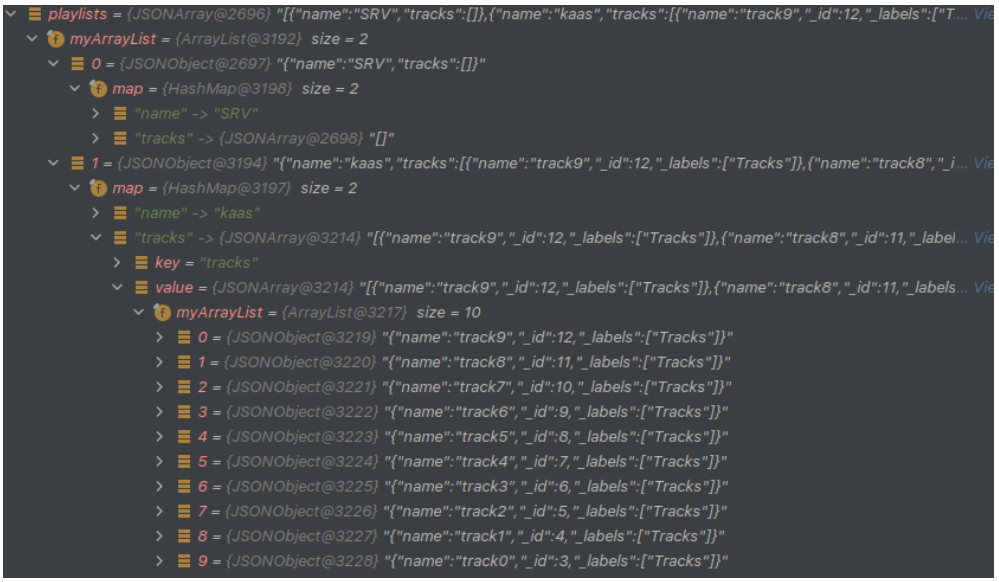
Stap 3 geeft het volgende resultaat in de database:



Je ziet hier dat er 10 tracks zijn aangemaakt die aan playlist “kaas” zijn toegevoegd.

Voor **stap 4** gebruik ik een JSON library om de json te parse het resultaat zie je hieronder.

Je zit hier dus een variabele playlists die 2 playlists bevat waarvan playlist “kaas” 10 tracks heeft.



3.2 Het werkplaatsonderzoek

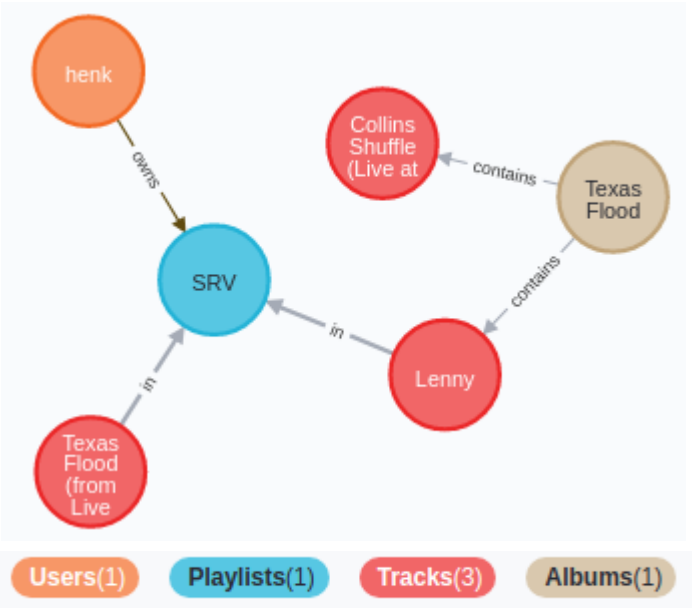
3.2.1 Beschrijving

In dit werkplaatsonderzoek ga ik de trackDAO aanpassen zodat methode getTracks() gebruik maakt van de neo4j database.

3.2.2 Resultaten

Na het maken van een create script (database/create.cypher) ziet de database er als volgt uit:

Je ziet hier dat de user “henk” een playlist heeft genaamd “SRV”, in die playlist zitten 2 tracks waarvan 1 track in een album genaamd “Texas Flood” zit. De derde track zit ook in dat album maar niet in de playlist



De query om alle tracks op te halen die niet in de gegeven afspeellijst zitten op te halen, ziet er als volgt uit:

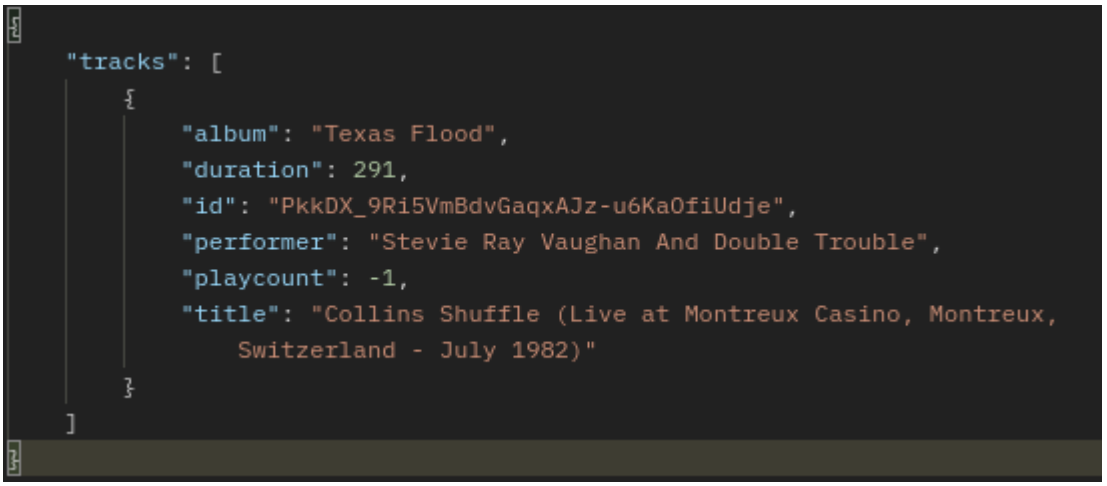
```
MATCH (ts:Tracks), (pl:Playlists)
OPTIONAL MATCH (album:Albums)-[:contains]→(ts)
WITH ts, pl, album
WHERE NOT (ts)-[:in]→(pl { id: "RtUtzbPwzN1rds0qEGtSvsmcvtIT3Rpxg0" })
RETURN {album_name: album.name, track: ts} AS data
```

Met als resultaat:

De nieuwe code voor getTracks() ziet er nu als volgt uit:

```
@Override
public ArrayList<Track> getTracks(String token, String forPlaylist) {
    try (Connection con = DB.getNoSQLURL()) {
        String queryRetrieve = "MATCH (ts:Tracks), (pl:Playlists) "
                             + "OPTIONAL MATCH (album:Albums)-[:contains]->(ts) "
                             + "WITH ts, pl, album "
                             + "WHERE NOT (ts)-[:in]->(pl { id: ? }) "
                             + "RETURN {album_name: album.name, track: ts} AS data";

        PreparedStatement stmtRetrieve = con.prepareStatement(queryRetrieve);
        stmtRetrieve.setString(1, forPlaylist);
        ResultSet resultSet = stmtRetrieve.executeQuery();

        ArrayList<Track> tracks = new ArrayList<>();

        while (resultSet.next()) {
            String dataString = resultSet.getString("data");
            JSONObject dataJSON = new JSONObject(dataString);

            JSONObject trackJSON = dataJSON.getJSONObject("track");

            Track track = null;

            if ( trackJSON.getString("type").equals("song") ) {
                track = new Song( trackJSON.getString("id") );
                parseTrackDataJSON(track, trackJSON);

                if ( dataJSON.has("album_name") )
                    track.setAlbum( new Album(dataJSON.getString("album_name")) );

            } else {
                track = new Video( trackJSON.getString("id") );
                parseTrackDataJSON(track, trackJSON);

                if ( trackJSON.has("playcount") )
                    track.setPlaycount(trackJSON.getInt("playcount"));

                if ( trackJSON.has("publication_date") )
                    track.setPublicationDate(trackJSON.getString("publication_date"));

                if ( trackJSON.has("description") )
                    track.setDescription( trackJSON.getString("description") );
            }

            tracks.add(track);
        }

        return tracks;

    } catch ( SQLException e ) {
        // TODO: errorhandling
        e.printStackTrace();
    }

    return null;
}

private void parseTrackDataJSON( Track track, JSONObject trackJSON ) {
    track.setDuration(trackJSON.getInt("duration"));
    track.setPerformer(trackJSON.getString("performer"));
    track.setTitle(trackJSON.getString("title"));
    track.setUrl(trackJSON.getString("url"));
}
```

In blok 1 wordt de query aangeroepen en wordt er een resultSet aangemaakt daarna in blok 2 wordt de resultSet omgezet in een JSONObject omdat de data vanuit neo4j wordt gegeven in JSON formaat. Als laatste in blok 3 wordt er een track object aangemaakt die dan gevuld wordt met data uit de database. Het eindresultaat in postman ziet er als volgt uit:



Conclusie

Nadat ik heb onderzocht wat een SQL- en een NoSQL database is, heb getest in een kleine test applicatie en neo4j heb geïmplementeerd in Spotitube kan ik antwoord geven op de onderzoeksvraag: **Hoe implementeer ik een niet-relationele db in spotitube?**
Als eerste is moet je de NoSQL dependency toevoegen aan de pom.xml. Wanneer de dependency succesvol is toegevoegd moet je in de DAO nieuwe queries aanmaken om te kunnen werken met de NoSQL database.

Bronnen

Wikipedia-bijdragers. (2021, 8 maart). Relationeel model. Wikipedia. https://nl.wikipedia.org/wiki/Relationeel_model (geraadpleegd op 29-03-2021).

Wikipedia contributors. (2021, 17 maart). NoSQL. Wikipedia. https://en.wikipedia.org/wiki/NoSQL (geraadpleegd op 29-03-2021).

Serra, J. (2020, 20 augustus). Relational databases vs Non-relational databases | James Serra's Blog. James Serra's Blog | Big Data and Data Warehousing. https://www.jamesserra.com/archive/2015/08/relational-databases-vs-non-relational-databases/ (geraadpleegd op 29-03-2021).

Vishwakarma, R. (2020, 2 juni). The Different Types of NoSQL Databases. Open Source For You. https://www.opensourceforu.com/2017/05/different-types-nosql-databases/ (geraadpleegd op 29-03-2021).

B. (2020, 20 juli). A Guide to Neo4J with Java. Baeldung. https://www.baeldung.com/java-neo4j (geraadpleegd op 31-03-2021).

Neo4j. (2021, 31 maart). Neo4j Graph Platform – The Leader in Graph Databases. Neo4j Graph Database Platform. https://neo4j.com/ (geraadpleegd op 2-04-2021).