Web Data Management

Salim Salmi, Bojana Urumovska, Jari Blom

June 2017

1 Introduction

This report describes the procedure followed to create a restful API, using three different types of databases as required for the course Web Data Managment. The data models will be described in section 2, the integration with the API in section 3 and lastly section 4 describes the experience we had with the different datastores and their integration throughout the project.

2 Data models

2.1 PostgreSQL

PostgreSQL uses tables to store data. In the tables (movies, actors, genres) we have rows representing an object, with attributes in its columns (id, lname). Relations between the rows of two different tables are defined in another look-up table (acted_in, movies_genres). This way of defining relationships is depicted in fig. 1. PostgreSQL is the most efficient for simple queries involving only one object, like: find every movie with Star Wars included in the title.

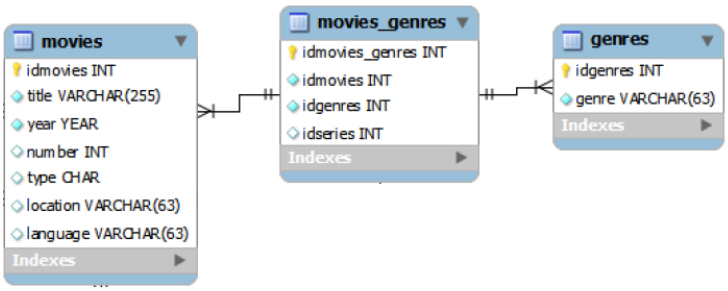

Figure 1: Visualization of relations in PosgreSQL

2.2 Neo4J

In Neo4J every object is a node, which can have properties and relations to other nodes implemented as edges, as visualized in fig. 2. As it skips the step of matching properties in the two tables, queries using two or more different objects, for example finding the title of every movie with Arnold Schwarzenegger, will be faster using Neo4J than for PostgreSQL. Neo4J's consistency however is not an important feat in this application.

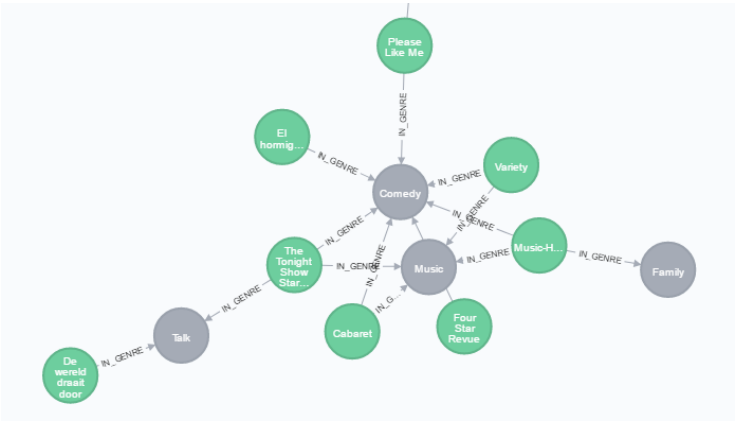

Figure 2: Visualization of relations in Neo4J

2.3 MongoDB

MongoDB is a document store data base, every object is stored as a document in a collection of similar objects. Any connection of an object to other objects needs to be defined in the document itself, which means every relation has to be defined in two directions. For example the genre_id of a movie has to be defined in the movie document, but also the movie_id has to be included in the document of that genre. MongoDB is the slowest of the three, but has the advantage that data can be easily distributed through sharding. It can be done in PostgreSQL too, but MongoDB has query routers that can identify the right shard to read from.

However in the IMDb application sharding does not make that much sense, since there is not really a way in which we could split the data, minimizing the access of multiple shards. Accessing multiple shards, makes the proces slow.

A comment on the structure of these documents though: not only genreid, but also genre is in the movie documents, which saves a transaction when asking for the genre of a certain movie.

3 Service Interfaces

For the project a python based restful API was made using Django. Django can run a PostgreSQL database when using the "postgresql_psycopg2"-engine. For running the other databases "neomodel" and "mongoengine" and "pymongo" were installed.

The structure of the database is defined in the "models.py" file of every one of the three API's. Specifying the node structure of the Neo4J, the table structure of the PostgreSQL and the document structure of the MongoDb database. Where Neo4J uses properties instead of fields as specified for the other two. All 3 databases can be queried by adding "/object/?attribute=value" to the published url.

All json objects look the same for the three databases: {"Movies":{"idmovies": "idmovies" , "title":"title","year:year","number":"number", "type":"type", "location":"location", "language":"language","actors":"actors", "genres":"genres"}}. The only difference is in the class of the last two entries: actors and genres are foreign keys in PostgreSQL, relations in Neo4J and a reference field in MongoDB.

4 Experience with the data stores

Versions of engines, database programs, python and django not being compatible was the main source of errors throughout the project. Other difficulties were in the vast amount of programs needed to convert the data from one to the other type of database.