MySQL database vervangen door MongoDB in een Java applicatie

Onderzoeksopdracht Enterprise Applications

v1.0
Rowan Paul Flynn 637588
ITA-OOSE-A-f 2020
Docent Michel Portier
2 april 2021
OOSE-DEA

Inhoudsopgave

Inleiding

Dit onderzoek is geschreven om erachter te komen hoe in een bestaande Java applicatie met MySQL MongoDB kan worden geïmplementeerd. De applicatie bevat een JAX-RS REST API voor de Spotitube applicatie. Op dit moment wordt gebruik gemaakt van MySQL als database maar in dit verslag wordt gekeken naar de mogelijkheid om MySQL te vervangen door een MySQL database.

De hoofdvraag van dit onderzoek is “Hoe kan de MySQL database vervangen worden door een MongoDB database?”. Gebaseerd op deze hoofdvraag zijn er vier (kleinere) deelvragen bedacht:

- Wat is MongoDB?
- Hoe verschilt MongoDB van MySQL?
- Hoe kan MongoDB gebruikt worden in een Java applicatie?
- Wat moet er gebeuren om MongoDB te implementeren in een al bestaande applicatie?

Er is gebruik gemaakt van bieb onderzoek, labonderzoek en werkplaats onderzoek (ICT Methodspak, z.d., pp. 1–3).

Het doel is dat na het lezen van dit verslag de lezer zelfstandig MongoDB kan implementeren in een al bestaande Java applicatie. Ook kan met behulp van dit verslag een betere keuze voor database gemaakt worden omdat de verschillen met MySQL expliciet genoemd worden in dit onderzoek.

1) Wat is MongoDB?

MongoDB is een NoSQL database (MongoDB, z.d.-e) die data opslaat in flexibele, BSON (lijkt op JSON) documenten. Dit betekent dat velden en data structuur kunnen verschillen per document.

MongoDB is ook gratis om te gebruiken onder de Server Side Public License (SSPL) v1 (MongoDB, z.d.-d).

NoSQL data modellen kunnen dingen “nested” opslaan, bijvoorbeeld een array van passagiers kunnen worden opgeslagen binnen het “trein” document (MongoDB, z.d.-f). NOSQL ontstond aan het eind van de 2000s omdat het fysiek opslaan van data vele malen goedkoper was geworden en het daarom niet meer nodig was om complexe data modellen te maken om kosten te besparen.

De opkomst van AGILE, een werkmethode met flexibele, kleine cycli (Scrum Company, z.d.), zorgde ervoor dat developers snel veranderingen wouden maken, iets wat je met NoSQL gemakkelijk kan doen omdat zoals eerder genoemd er geen database schema nodig is.

Een MongoDB database bestaat uit één of meerdere collecties, denk tabellen in SQL. Een collectie bevat documenten (denk aan rijen in SQL). Een document bestaat weer uit velden, die lijken op kolommen in SQL (Seguin, z.d., pp. 1–3). Documenten zijn dus waar je de data in op slaat.

Een veelgenoemd voordeel van MongoDB is dat het een flexibel schema heeft, dit betekent dat je niet van te voren een schema moet opstellen waarin staat wat je in het document gaat zetten. Als een attribuut niet bestaat maakt MongoDB deze gewoon aan. Dit zorgt voor een stuk minder hoofdpijn bij developers, die hierdoor veel gemakkelijker een nieuw attribuut bij een document kunnen testen zonder eerst het schema aan te passen

MongoDB is veel simpeler en veel meer rechtdoorzee dan SQL. Ook is het in het algemeen sneller en heeft het minder regels voor de developer om te onthouden.

Voor dit onderzoek wordt gebruik gemaakt van de MongoDB Java Drivers die zorgt dat MongoDB gemakkelijk kan worden aangesproken. Naast een driver heeft MongoDB ook nog drivers voor 9 andere talen. Voor meer informatie, zie ook het hoofdstuk 3) Hoe kan MongoDB gebruikt worden in een Java applicatie voor meer informatie.

2) Hoe verschilt MongoDB van SQL?

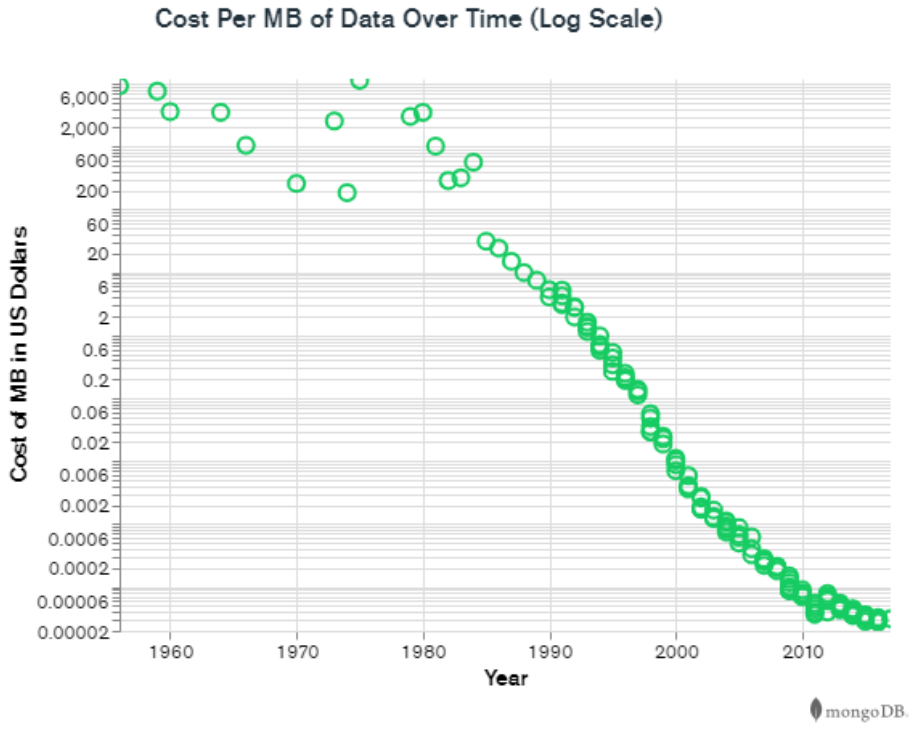
In het vorige hoofdstuk is er globaal beschreven wat MongoDB en MongoDB zijn, maar in dit hoofdstuk wordt dieper ingegaan op de verschillen tussen MongoDB en SQL (MongoDB, z.d.-c).

MongoDB heeft flexibele data modellen, wat betekent dat er gemakkelijk veranderingen kunnen worden gedaan. Hierdoor kunnen er snel veranderingen gemaakt worden in de database. MongoDB schaalt horizontaal, in plaats van naar een grotere, duurdere server moet migreren als je huidige server niet meer voldoet kan je gewoon meer kleine servers toevoegen wanneer dat nodig is.

Data in SQL databases is vaak genormaliseerd en dat zorgt ervoor dat voor één object soms meerdere tabellen moeten worden geraadpleegd. Dit zorgt voor slome *queries*. MongoDB databases slaan data op zodat ze makkelijker *queried* kunnen worden omdat data die bij elkaar hoort bij elkaar wordt opgeslagen en er dus geen joins nodig zijn.

Developers vinden MongoDB databases over het algemeen makkelijker omdat ze hun data structuur die ze in hun taal gebruiken vaak overgenomen kan worden in MongoDB. Bijvoorbeeld een array met objecten kan direct in de MongoDB database worden gevoegd. Dit zorgt voor minder bugs en meer vrijheid voor developers.\

Een nadeel van flexibele schema's is dat MongoDB databases vaak groter zijn dan een vergelijkbare SQL database. Maar tegenwoordig is data opslaan zo goedkoop, dat dit niet meer een groot impact heeft op de keuze van database, zoals te zien is in afbeelding 1.



Afbeelding 1 Kosten van het opslaan van 1 MB over tijd

3) Hoe kan MongoDB gebruikt worden in een Java applicatie?

Om erachter te komen hoe MySQL te vervangen is door MongoDB kan worden is er een nieuw Java project aangemaakt om het te testen. Er is vanuit gegaan dat je al mongodb hebt geïnstalleerd op je systeem en een database genaamd *onderzoek* hebt aangemaakt. . De code in dit hoofdstuk is ook te vinden in Bijlage A Labonderzoek MongoDB in Java in een JAX-RS project.

Maak een nieuw TomEE project aan binnen je IDE en zet in je *pom.xml* bestand het volgende (MongoDB, z.d.-a):

```
<dependencies>
    <dependency>
        <groupId>org.mongodb</groupId>
        <artifactId>mongodb-driver-sync</artifactId>
        <version>4.2.2</version>
    </dependency>
</dependencies>
```

Importeer de volgende dingen vanuit de MongoDB driver (MongoDB, z.d.-b):

```
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;

import com.mongodb.client.MongoDatabase;
import com.mongodb.client.MongoCollection;

import org.bson.Document;
import java.util.Arrays;

import static com.mongodb.client.model.Filters.*;
import static com.mongodb.client.model.Updates.*;
```

Maak op de volgende manier de connectie naar de database, selecteer de juiste database en selecteer de collectie die je wilt hebben:

```
MongoClient mongoClient = MongoClients.create();
MongoDatabase database = mongoClient.getDatabase("onderzoek");
MongoCollection<Document> collection = database.getCollection("test");
```

Een document toevoegen kan door een nieuw *Document* te maken en indien je meerdere attributen hebt, *append* te gebruiken om ze toe te voegen. Als laatste run je *collection.insertOne* om het *Document* toe te voegen:

```
Document doc = new Document("name", "A Test")
      .append("count", 1)
      .append("versions", Arrays.asList("v3.2", "v3.0", "v2.6"))
      .append("info", new Document("x", 203).append("y", 102));
collection.insertOne(doc);
```

Om een document op te vragen, kan je een nieuw document maken en *collection.find()* doen. Om altijd het eerste document op te vragen, voeg *.first()* achter deze *query*:

```
Document myDoc = collection.find().first();
```

Om alle documenten op te halen en vervolgens doorheen te *loopen* maak je eerst een *cursor* aan die je alle documenten laat ophalen en vervolgens loop je er doorheen totdat je klaar bent en de *cursor* sluit::

```
MongoCursor<Document> cursor = collection.find().iterator();

try {
    while (cursor.hasNext()) {
        System.out.println(cursor.next().toJson());
    }
} finally {
    cursor.close();
}
```

Documenten updaten kan met behulp van *collection.updateOne()* met daarin tussen de haakjes als eerste een filter en als laatste welke attributen je wilt updaten met wat:

```
collection.updateOne(eq("name", "A Test"), set("name", "A test, changed"));
```

Een document verwijderen kan door gebruik te maken van *collection.deleteOne()* met tussen de haakjes een filter welk document verwijderd moet worden.

```
collection.deleteOne(eq("name", "A test, changed"));
```

Bij het ophalen, bewerken en verwijderen van documenten kan je filters gebruiken. De meest gebruikte filter is *eq*, die checkt of een attribuut gelijk is aan het gegeven attribuut. Andere filters zijn *gt* voor groter dan, lte voor lager of gelijk aan en de *and* filter die je om twee dingen heen kan zetten zoals dit:

```
and(gt("i", 50), lte("i", 100))
```

Door dit kleine onderzoek is gebleken dat alle methoden die gebruikt worden in de applicatie, namelijk dingen ophalen, toevoegen, veranderen en verwijderen uit de database ook gedaan kunnen worden in MongoDB met de Java driver.

4) Wat moet er gebeuren om MongoDB te implementeren in een al bestaande applicatie?

De huidige applicatie bestaat uit 3 verschillende lagen:

- De DAO waarin de gegevens uit de database worden gehaald
- Het Domain, de objecten die gebruikt worden binnen de applicatie
- De Service laag waar alle logica gedaan wordt

Doordat er gebruik wordt gemaakt van *dependency injection* in de DAO is het relatief makkelijk om de MySQL DAO om te wisselen voor een MongoDB DAO zonder de andere lagen daarvoor aan te passen.

Als eerste moet, net als bij hoofdstuk 3 de MongoDB driver toegevoegd worden aan het project:

```
<dependencies>
    <dependency>
        <groupId>org.mongodb</groupId>
        <artifactId>mongodb-driver-sync</artifactId>
        <version>4.2.2</version>
    </dependency>
</dependencies>
```

Voor dit onderzoek is de *UserDAO* aangepast, deze klasse bevat 3 methoden: één om een gebruiker op te halen aan de hand van zijn/haar naam, één om een token toe te voegen aan de database en één om de token te checken.

Zet de klasse waarvan je een MongoDB versie maakt `@Alternative` voor en voeg het volgende toe aan *beans.xml* tussen de *<beans>* tags:

```
<alternatives>
    <class>com.rowanpaulflynn.dao.UserDAOMongo</class>
</alternatives>
```

Dit geeft aan dat de volgende klassen een *alternative* hebben op de *default* implementatie, bij ons de MySQL implementatie. Nu hoeft alleen de injection setter in de service nog veranderd worden:

```
@Inject
public void setUserDAO(IUserDAO userDAOMongo) {
   this.userDAO = userDAOMongo;
}
```

Dit zorgt ervoor dat de *userDAOMongo* gebruikt wordt wanneer de userDAO wordt aangeroepen in deze klasse.

```
@Override
public Token createToken(String user) {
   Token token = new Token(user);
```

```
    Document doc = new Document().append("token",
token.getToken()).append("user", user);
    tokensCollection.insertOne(doc);

    return token;
}
```

Bij het maken van een token in *createToken* zou een hele goede oplossing zijn om de token in een array op te slaan in de users collection. Dit is lekker makkelijk en maakt goed gebruik van de voordelen van MongoDB. Toch is dit niet gedaan, omdat anders de andere methodes zouden moeten worden aangepast, wat niet de bedoeling was in dit onderzoek.

Na het aanpassen van alle methoden in de *userDAO* loopt de applicatie nu gedeeltelijk op MongoDB, zonder iets van de andere lagen aan te hoeven passen (behalve de setters natuurlijk).

Conclusie

Met MongoDB, een gratis NoSQL database, kan heel gemakkelijk veranderingen worden doorgevoerd door het flexibele data schema en wijde ondersteuning aan datatypes en nesting. Een nadeel aan MongoDB is dat het veel dubbele data kan bevatten, maar doordat data goedkoop kan worden opgeslagen is dit niet een heel groot probleem.

Een MongoDB database kan gemakkelijk gebruikt worden in een Java applicatie door gebruik te maken van de mongodb-driver-sync dependency. Het is gemakkelijk om documenten toe te voegen, op te halen, te bewerken en te verwijderen. Dit is aangetoond naarmate een labonderzoek (REF BIJLAGE).

Bestaande applicaties kunnen ook gemakkelijk worden omgezet naar MongoDB als ze dependency injection gebruiken. Het enigste wat hoeft te gebeuren is de dependency setter aan te passen naar de correcte klasse en de juiste methodes te implementeren. Het is ook belangrijk om op te letten hoe een MongoDB database anders is ingericht dan een MySQL database.

MongoDB is een goed alternatief wat, vooral voor de developer, makkelijker te implementeren en op verder te bouwen is. Dankzij het gebruik van dependency injection is het aanpassen van de datasource een fluitje van een cent.

Bronnen

ICT Methodspak. (z.d.). Onbekend.

MongoDB. (z.d.-a). Installation. Geraadpleegd op 29 maart 2021, van https://mongodb.github.io/mongo-java-driver/4.2/driver/getting-started/installation/

MongoDB. (z.d.-b). MongoDB Driver Quick Start. Geraadpleegd op 29 maart 2021, van https://mongodb.github.io/mongo-java-driver/4.2/driver/getting-started/quick-start/#mongodb-driver-quick-start

MongoDB. (z.d.-c). NoSQL vs SQL Databases. Geraadpleegd op 29 maart 2021, van https://www.mongodb.com/nosql-explained/nosql-vs-sql

MongoDB. (z.d.-d). Server Side Public License (SSPL). Geraadpleegd op 29 maart 2021, van https://www.mongodb.com/licensing/server-side-public-license

MongoDB. (z.d.-e). What Is MongoDB? Geraadpleegd op 29 maart 2021, van https://www.mongodb.com/what-is-mongodb

MongoDB. (z.d.-f). What is NoSQL? NoSQL Databases Explained. Geraadpleegd op 29 maart 2021, van https://www.mongodb.com/nosql-explained

Scrum Company. (z.d.). Meer snelheid en effectiviteit door Agile werken. Maar wat is Agile precies? Geraadpleegd op 29 maart 2021, van https://www.scrumcompany.nl/wat-is-agile/

Seguin, K. (z.d.). The Little MongoDB Book (now updated for 2.6 editie) [E-book]. https://www.openmymind.net/mongodb.pdf