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 (METHODEKAARTEN REF).

Het doel is dat na het lezen van dit verslag de lezer zelfstandig MongoDB kan implementeren in een al bestaande Java applicatie.

1 Wat is MongoDB?

MongoDB is een NoSQL database (REF What Is MongoDB? | MongoDB) 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 (REF Server Side Public License (SSPL) | MongoDB).

NoSQL data modellen kunnen dingen “nested” opslaan, bijvoorbeeld een array van passagiers kunnen worden opgeslagen binnen het “trein” document (REF What is NoSQL? NoSQL Databases Explained | MongoDB). 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 (REF https://www.scrumcompany.nl/wat-is-agile), 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 (REF mongodb.pdf (openmymind.net). 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 driver (REF 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 (REF HOOFDSTUK) voor meer informatie.

2) Hoe verschilt MongoDB van SQL?

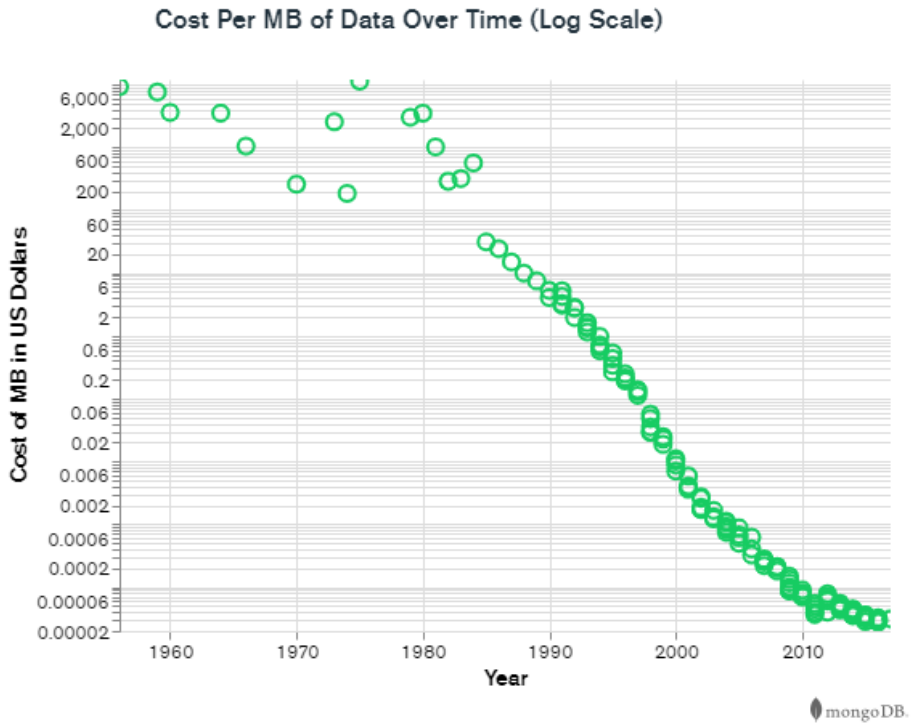
In het vorige hoofdstuk (REF HOOFDSTUK) is er globaal beschreven wat MongoDB en MongoDB zijn, maar in dit hoofdstuk wordt dieper ingegaan op de verschillen tussen MongoDB en SQL (REF NoSQL vs SQL Databases | MongoDB).

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 REF AFBEELDING.



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 (REF Quick Start (mongodb.github.io)).

Maak een nieuw TomEE project aan binnen je IDE en zet in je *pom.xml* bestand het volgende:

```
<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:

```
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();
```

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"));
```

something something over het verschil tussen 1 document en meerdere documenten en de verschillende dingen over filters. ook iets over al bestaand database genaamd onderzoek te hebben anders big sad hij niet werkt

Conclusie