# Projekt NoSQL


| | |
|---|---|
| Author: | Lukas Lieb |
| | Daniel Schmid |
| | Matthias Eiholzer |
| E-Mail: | lukas.lieb@stud.hslu.ch |
| | daniel.schmid@stud.hslu.ch |
| | matthias.eiholzer@stud.hslu.ch |


| | |
|---|---|
| Erstellungsdatum: | 22. Dezember 2016 |

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# 1 Einführung

## 1.1 Aufgabenstellung

Ziel dieser Arbeit ist es, die Uni-DB, welche auf einem SQL Server liegt, in eine NoSQL-Datenbank zu implementieren. Dazu wird das ER-Schema der Uni-DB so umgezeichnet, dass es in der gewählten NoSQL-Datenbank abgebildet werden kann. Dieses Schema wird dann in der entsprechend gewählten Datenbank umgesetzt. Dazu werden die Daten aus der SQL- in die NoSQL-Datenbank migriert. Zusätzlich wird dem Benutzer eine Möglichkeit geboten, mit Hilfe eines GUI folgende SQL-Querry auf der MongoDB [1] abzufragen:

```
select ProfessorName, AnzahlStudenten, SummeSWS
from (
        select
                p.Name as ProfessorName,
        count(s.MatrNr) as AnzahlStudenten
        from Professoren p
        join Vorlesungen v on v.gelesenVon = p.PersNr
        join hoeren h on h.VorlNr = v.VorlNr
        join Studenten s on s.MatrNr = h.MatrNr
        group by p.Name
) A
join
(
        select
                p.Name as ProfessorName,
        sum(SWS) as summeSWS
        from Professoren p
        join Vorlesungen v on v.gelesenVon = p.PersNr
        group by p.Name
) B
using(ProfessorName);
```

## 1.2 Datenbankauswahl

Bei der Auswahl der Datenbank haben sich die Verfasser dieser Arbeit für die MongoDB entschieden. Ausschlaggebend waren dabei folgende Gründe:

- Geeignet um die Aufgabenstellung zu lösen.
- Wird in der Praxis oft eingesetzt
- Bekannt
- Gute Dokumentation
- Kostenlos
- Unterstützung durch Forenuser
- Erste Erfahrung vorhanden

Die MongoDB ist eine Datenbank, welche sich grundlegend im CAP Theorem in der Spalte CP aufhält. Dies bedeutet, dass die Konsitenz (Consistency) gewährleistet ist. Dies wird realisiert, indem in einem System ein Knoten als primäres Mitglied fungiert und alle Anfragen über diesen Knoten abgearbeitet werden. Zusätzlich wird die Partition-Tolerance gewährleistet. Gemäss Betreiber wird bei einem Ausfall des Primärknotens automatisch ein Sekundärknoten zum neuen Primärknoten definiert und somit eine einwandfreie Funktion gewährleistet. Ebenfalls soll es möglich sein, auf Kosten der Konsistenz (Consistency) die Verfügbarkeit (Availability) zu erhöhen, indem man in den Einstellungen die Konfiguration vornimmt, dass Daten nicht bloss über den Primär-Knoten, sondern auch über Sekundärknoten bezogen werden können.

# 2 Datenmanagement

Zwei Aktoren wirken auf die Datenbank ein. Einerseits der Benutzer, welcher die Informationen abfragen kann. Andererseits der DB-Administrator, der die Daten abfragen, aber auch verändern und neue hinzufügen kann. Diese Anwendungsfälle sind in der Abbildung 1 abgebildet.

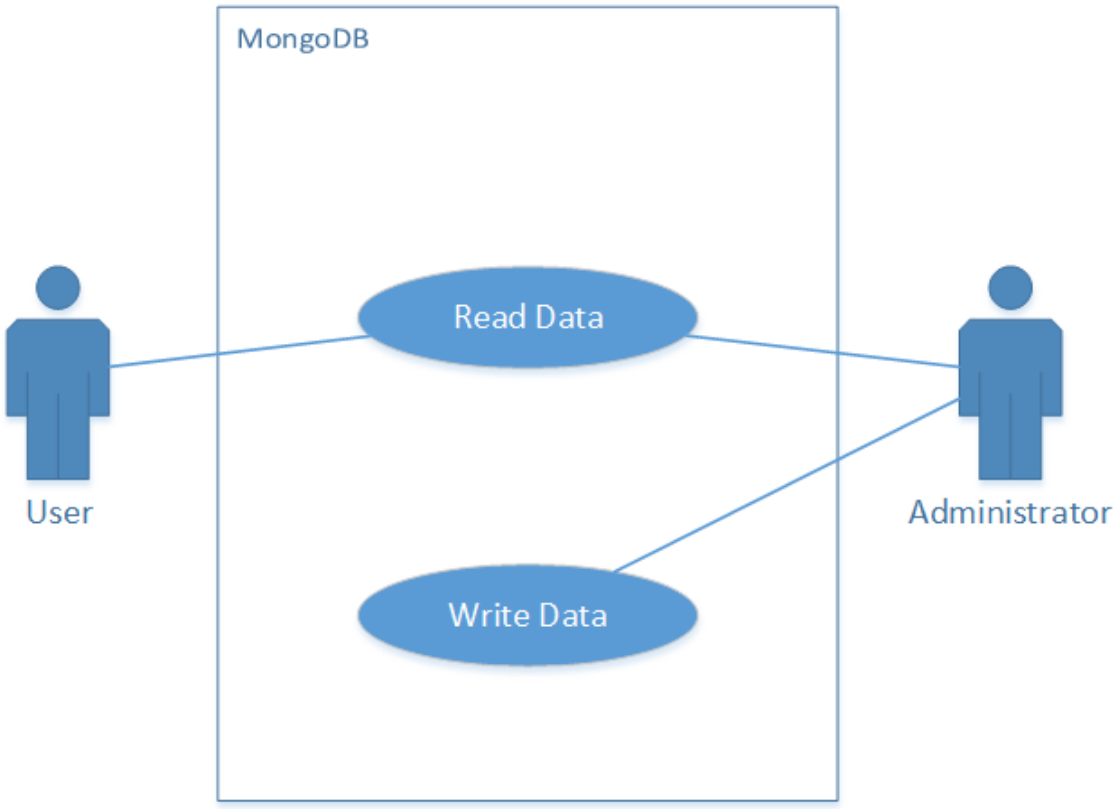


Abbildung 1: Visualisierung des Anwendungsfalls

Die Daten, welche in der MongoDB zum Einsatz kamen, wurden aus der Uni-DB, die für einige Testataufgaben im Modul DMG bereits verwendet wurde, migriert. Um diese Migration durchzuführen ist ein Java Programm geschrieben worden, welches in Kapitel 4 näher erläutert wird. Der Strukturierte Aufbau der Datenbank wird in Kapitel 3 beschrieben.

Der Benutzer hat die Möglichkeit über ein GUI Daten auf der Datenbank abzufragen: Über dieses GUI sind alle Professoren ersichtlich welche in der Datenbank

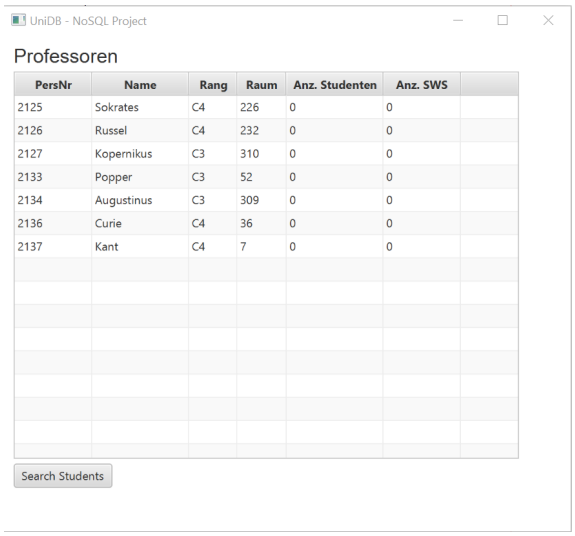


Abbildung 2: Gui für den Benutzer

eingetragen sind. Mithilfe des SearchStudents Knopf kann der Benutzer abfragen, wie viele Studenten in der Vorlesung eines Professors sitzen, und wie viele SWS Punkte er unterrichtet.

# 3 Datenmodellierung

MongoDB ist eine Dokument orientierte Datenbank. Dabei werden die Daten in JSON ähnlichen Dokumenten verwaltet. Um das Entity-Relations Schema in die MongoDB abbilden zu können, muss das Schema umgeformt werden. Dies ist auch bei relationalen Datenbanken der Fall. In Abbildung 3 ist das ER-Schema der uni-DB abgebildet. Dies ist die Ausgangslage für unser Projekt. Eine Collection der Mon-

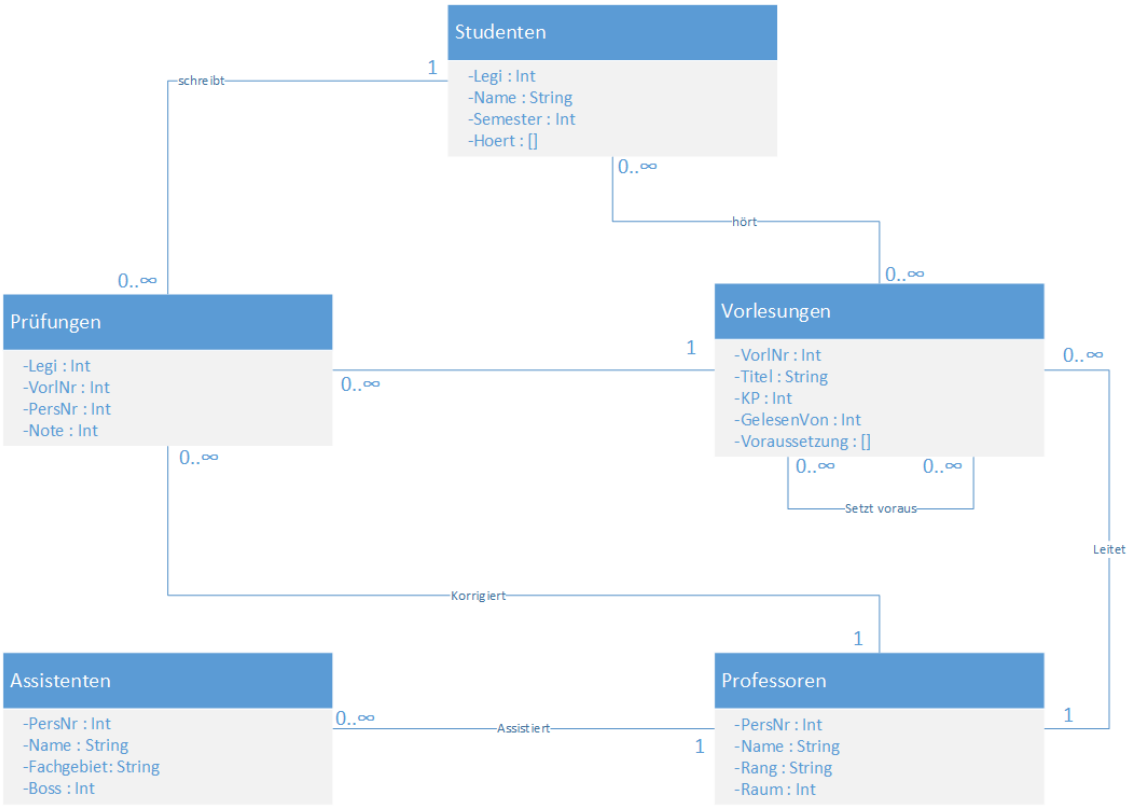


Abbildung 3: ER-Diagramm zur Uni-DB

goDB entspricht der Tabelle in relationalen Datenbanken. Das Dokument ist das Gegenstück zum Tupel. Die Attribute und die dazugehörigen Werte werden in der MongoDB als Schlüssel-Wert Paare gespeichert. Beziehungen werden in RDBMS als Fremdschlüssel abgebildet. In MongoDB können Relationen in Form von Werten eines bestimmten Keys gespeichert werden. Komplex-Komplexe Beziehungen werden in RDBMS in eigene Tabellen ausgelagert. Daraus entstehen dann zwei Einfach-Komplexe Beziehungen.

Wie bereits erwähnt, muss das ER-Schema auch für die MongoDB umgeformt wer-

den. Starke Entitäten werden als Collections abgebildet, schwache Entitäten werden als Unterteil in das dazugehörige Dokument eingebettet. Komplex-Komplexe Beziehungen werden auch im Schema für die MongoDB aufgespalten. Das Schema nach der Umformung ist in der Abbildung 4 ersichtlich. Eine Aggregation bedeutet, dass

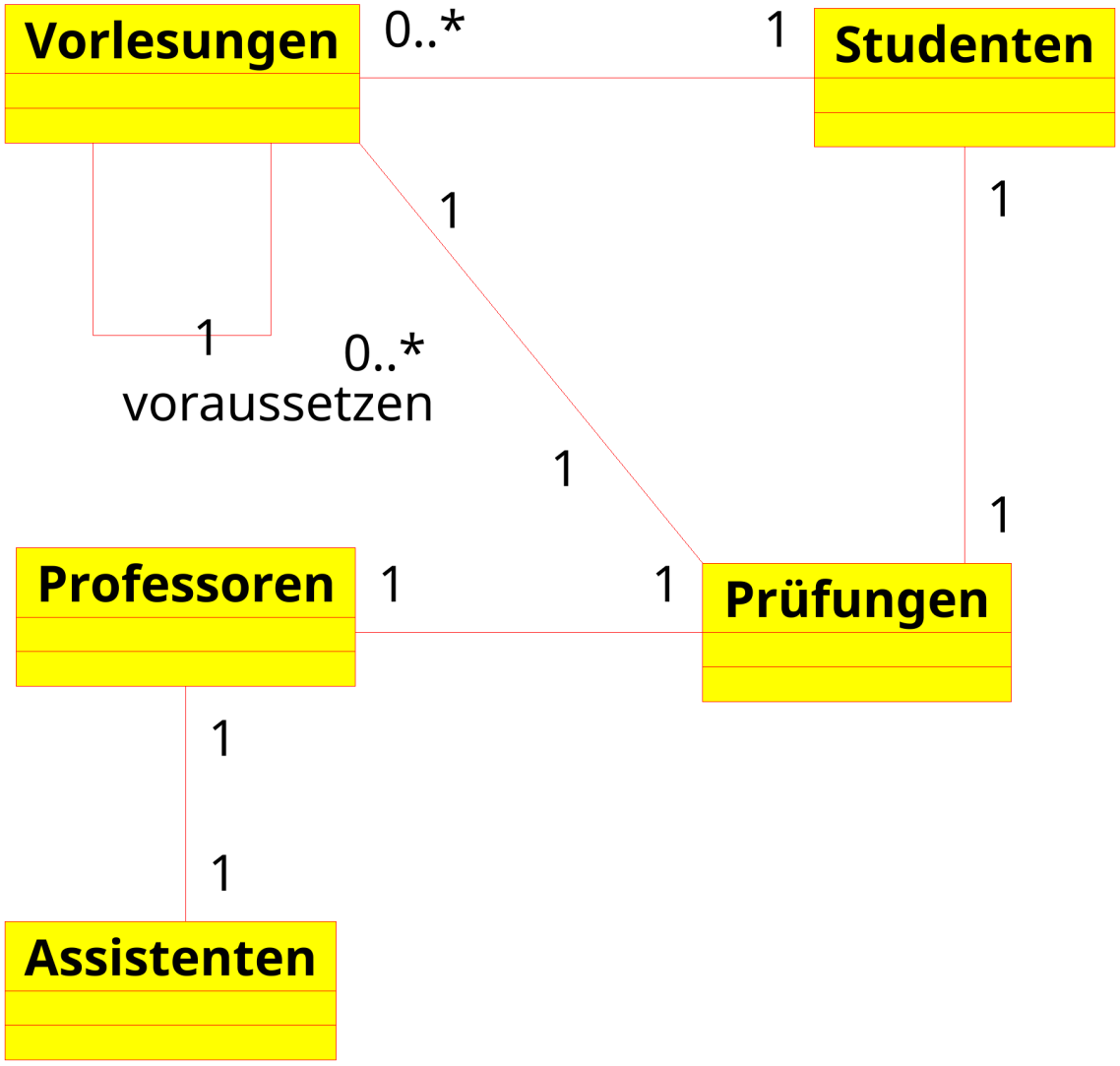


Abbildung 4: MongoDB Schema zur Uni-DB

eine schwache Entität als Unterteil abgebildet wird. Eine Assoziation bedeutet, dass eine starke Entität als Referenz abgebildet wird. In MongoDB wird die ID des Referenzierten Dokuments als Wert gespeichert.

Der folgenden JSON-Code zeigt ein Beispiel wie eine starke Entität in MongoDB abgespeichert wird. Das Code-Beispiel zeigt das Dokument eines Studenten. Die

Vorlesungen, welcher dieser Student hört, werden im Werte Array des Schlüssels Hören abgespeichert. Die Werte entsprechen dabei der ID der Vorlesung.

```
{
"Legi": 25403,
"Name" : "Jonas",
"Semester" : 12,
"Hören" : [5032, 1910],
}
```

# 4 Datenbanksprachen

MongoDB bietet verschiedene Möglichkeiten um Daten hinzuzufügen oder zu manipulieren. So können Dokumente beispielsweise über die Kommandozeile hinzugefügt werden. Zudem gibt es für viele Programmiersprachen einen Treiber, der in das jeweilige Projekt eingebunden werden kann.

Da für diese Arbeit ein Programm in Java geschrieben wurde, kam der aktuelle mongo-java-driver in der Version 3.4.0 zum Einsatz. Allerdings sind die Kommandos für die Interaktion mit den Daten nicht sehr unterschiedlich zum normalen Kommandofenster. Nach der Wahl der Datenbank kann im Java Treiber die entsprechende Collection ausgewählt und dieser anschliessend Dokumente hinzugefügt werden:

```
MongoClient mongo = new MongoClient( "localhost" , 27017 );
MongoDatabase database = mongo.getDatabase("unidb");
MongoCollection<Document> stud = database.getCollection("studenten");
stud.drop();
studs.insertOne(new Document("Legi", 25403)
        .append("Name", "Jonas")
        .append("Semester", 12)
        .append("Hoeren", 5022));
mongo.close();
```

So wurde für die Testdaten ein Javaprogramm erstellt, dass alle Daten der UniDB in entsprechende Collections der MongoDB schreibt.

Für die Erwähnte Query, die in der Einführung bereits erwähnt wurde, wurde eine separate Funktion geschrieben, die die Daten aus der Datenbank holt und aneinanderhängt. Der Join wird bei den zwei for-Schleifen ausgeführt. Darin wird für jeden Professor nachgeschlagen, welche Vorlesungen dieser hält. Anhand dieser wird die Anzahl SWS Punkte ermittelt, die der Professor unterrichtet. In der zweiten for-Schleife wird zusätzlich gezählt, wie viele Studenten dieser Vorlesungen zuhören. Die Selektion wird hier bei der Setzung der Anzahl SWS und Studenten zur Liste aller Professoren ausgeführt (p.setAnzSWS(swsCounter) und p.setAnzStud(studCounter)).

```
MongoClient mongo = new MongoClient( "localhost" , 27017 );
MongoDatabase database = mongo.getDatabase("unidb");
MongoCollection<Document> prof = database.getCollection("professoren");
MongoCollection<Document> vorl = database.getCollection("vorlesungen");
MongoCollection<Document> stud = database.getCollection("studenten");
```

```
for (Professor p: professoren){
        int swsCounter = 0;
        int studCounter = 0;

        for(Document dVorl: vorl.find(eq("GelesenVon", p.getPersNr()))){
                swsCounter += Integer.parseInt(dVorl.get("SWS").toString());
                for(Document dStud: stud.find(eq("Hoeren", dVorl.get("VorlNr")))){
                        studCounter ++;
                }
        }
        p.setAnzSWS(swsCounter);
        p.setAnzStud(studCounter);
}
```

Wenn die Query in MySQL ausgeführt wird, werden als Resultat die Anzahl Studenten und die Summe der SWS für jeden Professor angezeigt:

| ProfessorName | AnzahlStudenten | SummeSWS |
|---|---|---|
| Augustinus | 2 | 2 |
| Kant | 4 | 8 |
| Popper | 1 | 2 |
| Russel | 2 | 8 |
| Sokrates | 4 | 10 |

Abbildung 5: Query Resultat in MySQL

Beim Javacode wird diese eine Tabelle angezeigt und mit einem Click auf den Search Students Knopf wird besagte Funktion oben ausgeführt und die Tabelle aktualisiert.

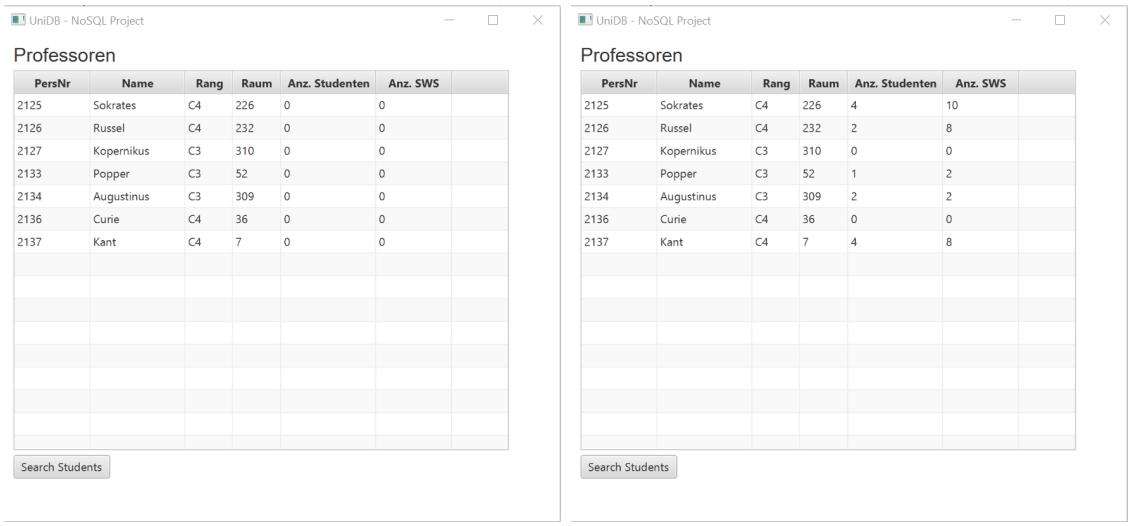


Abbildung 6: Query Resultat MongoDB

# 5 Konsistenzsicherung

Bei relationalen Datenbanken bietet die Datenbank Mechanismen zur Konsistenzsicherung an. Dies wird durch verschiedene Eigenschaften im RDBMS erreicht. Jeder Tabelle liegt ein Schema zu Grunde. In diesem wird definiert, wie die Daten strukturiert sein müssen. Zum Beispiel wird festgelegt, welches der Primärschlüssel ist, ob Felder "null" sein dürfen oder ob Tupel gelöscht werden dürfen wenn noch Referenzen darauf zeigen.

MongoDB ist Schema frei. Das bedeutet, dass sich je zwei Dokumente in einer Sammlung komplett in ihrer Struktur voneinander unterscheiden können. Des Weiteren garantiert MongoDB keine Integrität der Referenzen. Dieser Umstand wird klar, wenn verglichen wird, welchen Regeln relationale Datenbanken und die MognoDB unterliegen.

RDBMS unterliegen den ACID Regeln:

- Atomar: Die gesamte Transaktion wird ausgeführt oder die Transaktion wird rückgängig gemacht.
- Consistent: Nach jeder Transaktion muss die Datenbank widerspruchsfrei sein.
- Isolatet: Bei einer Transaktion dürfen keine Seiteneffekte auftreten. Dies garantiert, dass ein Mehrbenutzerbetrieb möglich ist.
- Dauerhaft: Die Daten werden sicher gespeichert, auch bei Systemabstürzen. Bei einem Systemabsturz wird ein Recovery durchgeführt, so dass danach die Datenbank wieder in einem konsistenten Zustand ist.

Die in unserem Projekt eingesetzte MongoDB unterliegt den BASE Regeln.

- Basically Available: Die Datenbank sollte meistens laufen.
- Eventually Consistent: Die Konsistenz der Daten wird nicht unmittelbar nach der Operation gewährleistet. Sie kann verzögert eintreten.

Da MongoDB keine Transaktionen (eine Folge von Operationen, die Atomar ausgeführt werden) unterstützt, muss diese Funktionalität auf der Anwendungsebene implementiert werden. Dies ist beim Einsatz eines RDBMS nicht notwendig, da dieses Transaktionen unterstützt. Im Falle dieses Projektes werden keine Transaktionen verwendet, da nur lesend auf die Daten zugegriffen wird. Da keine Daten geändert oder hinzugefügt/entfernt werden, kann die Datenbank durch Operationen nicht in einen inkonsistenten Zustand überführt werden. Deswegen kann auch parallel auf die Daten zugegriffen werden, ohne die Konsistenz zu verlieren. Sollen zu einem späteren Zeitpunkt zur Laufzeit der Anwendung Daten geändert oder hinzugefügt/entfernt werden, so muss eine Konsistenzsicherung auf der Anwendungsebene implementiert werden.

# 6 Systemarchitektur

MongoDB besteht aus drei verschiedenen Servern. Den Routern, Config Servern und den Replica Sets. Jeder dieser drei Servertypen hat eine bestimmte Aufgabe. Der Client sendet seine Operation an den Router. Der Router wiederum leitet die Operation an die zuständigen Replica Sets weiter. Die Replica Sets wiederum speichern die zu verarbeitenden Dokumente. Damit der Router weiss, an welche Replica Sets er die Anfrage weiterleiten muss, stellt er wiederum eine Anfrage an die Config Server. Diese antworten entsprechend. Die Config Server enthalten also die Metadaten über die Dokumentenverteilung in den Replica Sets. In der Abbildung 7 ist die Architektur von MongoDB visualisiert. In der, in diesem Projekt eingesetzte,

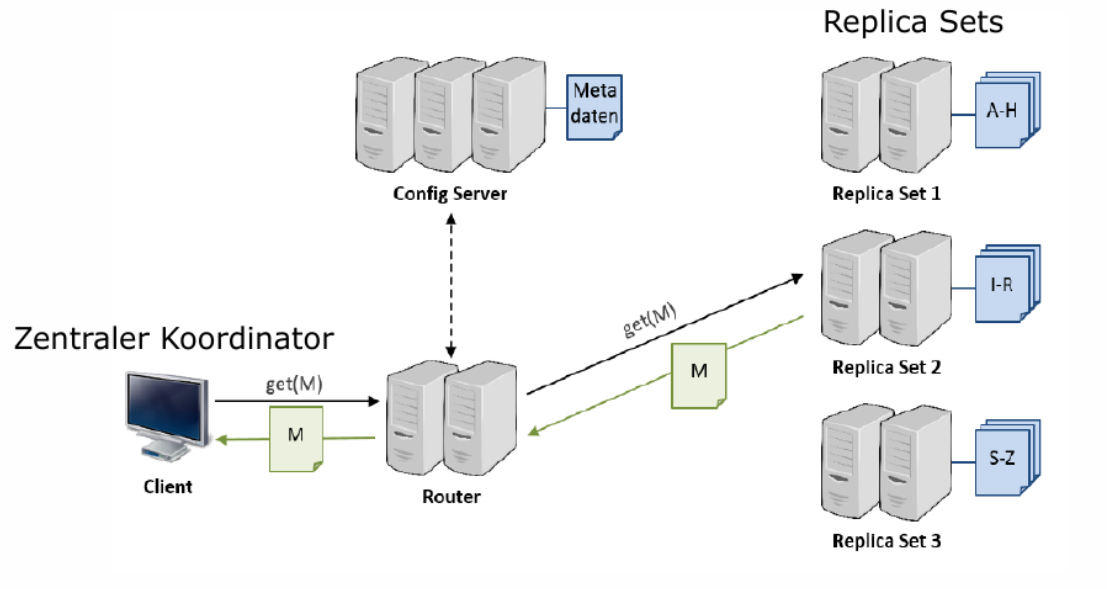


Abbildung 7: Architektur von MongoDB. [3]

MongoDB Instanz laufen alle drei Server auf demselben physischen Rechner. Dies wurde so gewählt, da die Sammlungen an Dokumenten klein ist. Des weiteren wird auch nur sporadisch auf die Daten zugegriffen. Sollte sich in Zukunft einer dieser Parameter ändern, kann darüber nachgedacht werden, die Server auf verschiedene physische Server zu verteilen. Dies hätte auch den weiteren Vorteil, dass parallel auf die Dokumente zugegriffen werden kann. Ist die Replica Set 1 mit der Verarbeitung einer Anfrage beschäftigt, kann gleichzeitig die Replica Set 2 die nächste Anfrage bearbeiten, sofern es sich bei der Anfrage um die in ihr gespeicherten Dokumente

handelt.

In RDBMS liegen die Tabellen in normalisierter Form vor. Daten die nicht funktional vom Primärschlüssel abhängen, werden als Fremdschlüssel referenziert. Die Referenzierung macht es schwierig, die Tabelle, in welcher das referenzierte Tupel abgelegt ist, auf einen anderen Server auszulagern. Denn bei jedem Zugriff auf die ausgelagerte Tabelle, wird die Abfrage der Daten verlangsamt, da zwischen den beiden Servern Kommunikation stattfindet. Dieses Problem hat MongoDB nicht, da die Daten nicht normalisiert vorliegen müssen. Bei MongoDB dürfen die bei RDBMS referenzierten Tupel als Aggretationen in die Dokumente umgesetzt werden. Dies vermindert die Anzahl der Referenzierungen und ermöglicht es damit, dass eine horizontale Skalierung einfacher zu bewerkstelligen ist, als bei einer relationalen Datenbank. Dies führt auch Nachteile mit sich. Jedes mal wenn ein referenziertes Tupel als Aggregation in ein Dokument gespeichert wird, erhöht sich die Datenmenge. Des weiteren bedeutet es mehr Aufwand, ein solches aggregiertes Tupel zu ändern, da die Änderung bei allen Dokumenten gemacht werden muss, bei denen das referenzierte Tupel als Unterteil gespeichert wird. Dies im Gegensatz zur relationalen Datenbank. Bei dieser muss die Änderung nur an einem Ort stattfinden. MongoDB kennt keine Funktionlität zum Auflösen von Referenzen. Diese muss jeweils in der Applikation implementiert werden.

Da wie bereits erwähnt, alle Server auf einem physischen Rechner laufen, ist die Datenbank bei einem Systemausfall nicht mehr erreichbar. Bei einem Festplattenausfall kann es sogar passieren, dass die komplette Datenbank verloren geht, da es in der Replica Set nur einen Server gibt, und somit ein weiter Server fehlt, der ein Replikat der Datenbank enthält.

# 7 Schlussfolgerungen

Nach dem Entscheid über die NoSQL Datenbank konnten wir erfolgreich eine Instanz von MongoDB aufsetzten. Zudem war es uns möglich per API der Datenbank über das Konsolenfenster Daten hinzuzufügen, zu ändern oder löschen. Die Robomongo [2] Applikation ermöglichte es uns die Änderungen auf einfache Art zu kontrollieren. Nach dem Einarbeiten in die Syntax des Mongo-Java Treibers war es uns auch möglich Änderungen an der Datenbank von einem Javaprogramm aus vorzunehmen.

Bei diesem Projekt ist uns klar geworden wie sehr sich NoSQL- von SQL-Datenbanken unterscheiden. Wir haben zwar schon vermehrt in der Theorie davon erfahren, allerdings ist es etwas ganz anderes dies in der Praxis zu erleben. Zudem ist es bei NoSQL Datenbanken viel wichtiger die Applikation sauber aufzubauen, da einem die API der Datenbank viel weniger Funktionen zur Verfügung stellt als beispielsweise bei einer SQL Abfragesprache.

MongoDB ist grundsätzlich sehr einfach installiert und es läuft sehr stabil. Auch die gebotene Abfragesprache ist nach einer kurzen Zeit sehr einfach und intuitiv. Grundsätzlich ist MongoDB im CP Bereich also Consistency und Partition-Tolerance angesiedelt. Die Möglichkeit auf Kosten der Consistency die Availability zu erhöhen macht sie zusätzlich für weitere Anwendungsgruppen sehr attraktiv. Die Datenbank verfügt über eine grosse Community, durch welche reichlich Hilfe bei Problemen zur Verfügung steht. Ausserdem ist sie auch sehr gut Dokumentiert.

Sie verfügt aber auch über gewisse Nachteile. So ist sie zum Beispiel nicht für alle Anwendungstypen einer NoSQL Datenbank ausgestattet. Zudem ist sie nicht abgesichert wenn mehrere Benutzer gleichzeitig auf der Datenbank Änderungen vornehmen. Bei gewissen Szenarien ist es zudem von Vorteil wenn sauber referenziert werden kann. Dies ist bei MongoDB und auch allgemein bei NoSQL Datenbanken nicht immer ideal.

Allgemein werden SQL Datenbanken häufiger in System verwenden, in denen die Konsistenz sehr hoch sein muss, wie zum Beispiel bei Banken oder auch Versicherungen. NoSQL Datenbanken werden dabei eher bei Datenmenge wie BigData oder auch bei grösseren WebServices eingesetzt. MongoDB speziell sollte bei Systemen zum Einsatz kommen, die eine hohe Daten Übereinstimmung bei den einzelnen Nodes und eine gute Ausfallsicherheit voraussetzen.

# Literatur

[1] Mongodb for giant ideas, 2016. https://www.mongodb.com/.

[2] Robomong - native mongodb management tool, 2016. https://robomongo.org//.

[3] Michael Kaufmann. Db: Dokument stores, 2016.