

# Projekt NoSQL

---

Author: Lukas Lieb

Daniel Schmid

Matthias Eiholzer

E-Mail: [lukas.lieb@stud.hslu.ch](mailto:lukas.lieb@stud.hslu.ch)

[daniel.schmid@stud.hslu.ch](mailto:daniel.schmid@stud.hslu.ch)

[matthias.eiholzer@stud.hslu.ch](mailto:matthias.eiholzer@stud.hslu.ch)

Erstellungsdatum: 20. Dezember 2016

---

# Inhaltsverzeichnis

1	Einführung	5
2	Datenmanagement	7
3	Datenmodellierung	8
4	Datenbanksprachen	10
5	Konsistenzsicherung	12
6	Systemarchitektur	14
7	Schlussfolgerungen	17

# Abbildungsverzeichnis

1	ER-Diagramm zur Uni-DB ? . . . . .	5
2	Visualisierung des Anwendungsfalls . . . . .	7
3	Datenmodell . . . . .	9
4	Query Resultat in MySQL ? . . . . .	11
5	Query Resultat MongoDB ? . . . . .	11
6	Architektur von MongoDB. ? . . . . .	15

## Tabellenverzeichnis

# 1 Einführung

Im Modul DMG wurde zu Übungszwecken zu relationalen Datenbanken die Uni-DB verwendet. Das dabei verwendete Entity-Relation Diagramm sieht folgendermassen aus: Um mit den NoSQL Datenbanken Erfahrung zu sammeln, wird dieses Schema

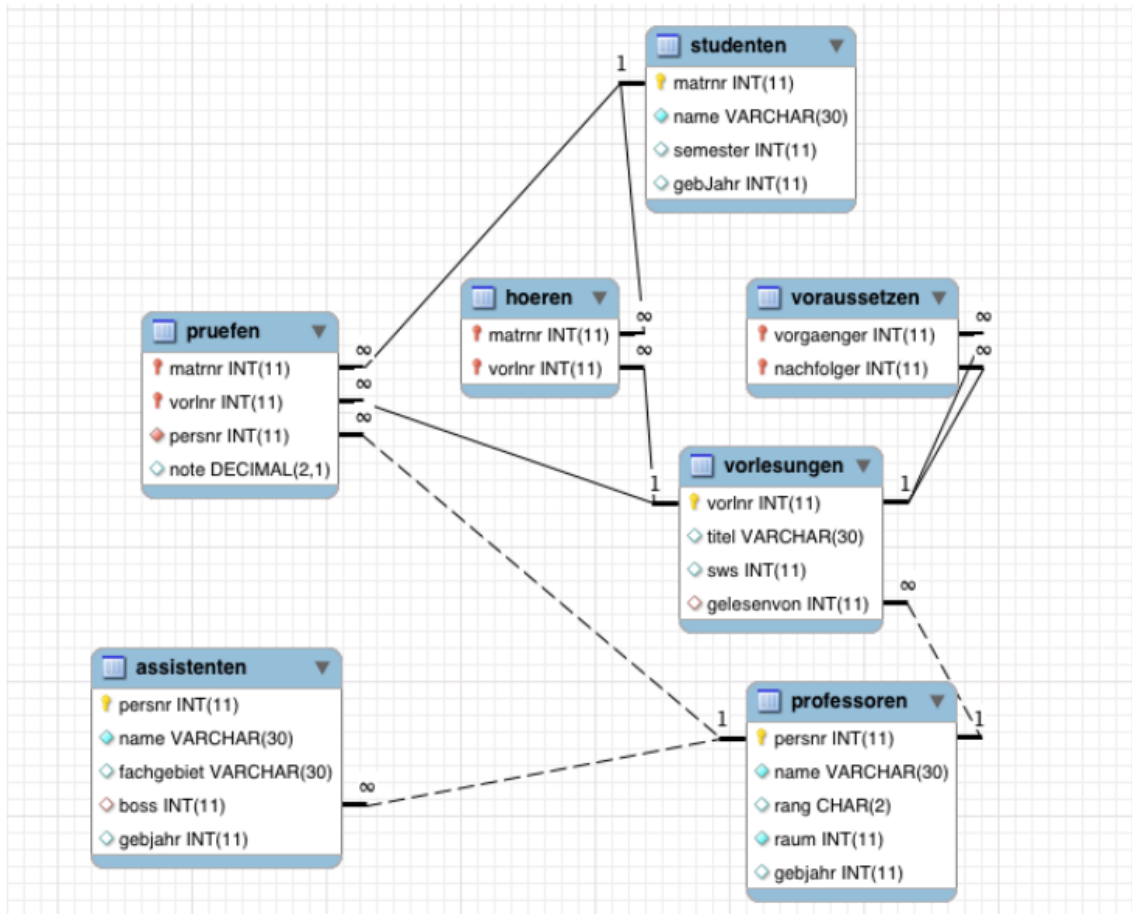


Abbildung 1: ER-Diagramm zur Uni-DB ?

in die NoSQL Datenbank MongoDB überführt. Die MongoDB als wird aus folgenden Gründen verwendet:

- Wird in der Praxis eingesetzt
- Gute Dokumentation
- Kostenfrei
- Unterstützung durch Forenuser

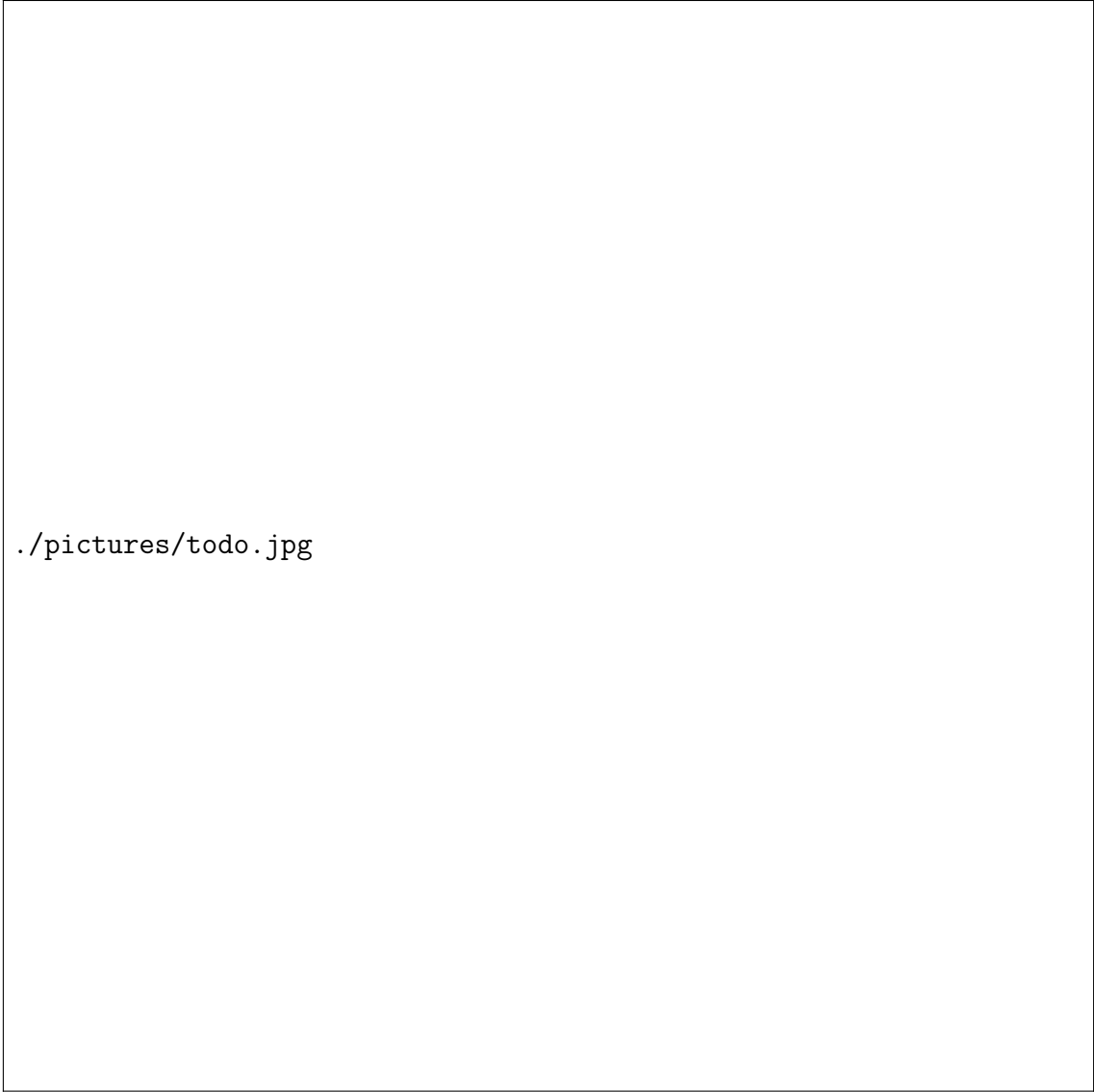
- Erste Erfahrung vorhanden

An die MonogDB wird eine Abfrage definiert, welche das selbe Resultat ergibt, wie das folgende SQL-Query:

```
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)
```

## 2 Datenmanagement

Zwei Akteure wirken auf die Datenbank ein. Einerseits der Benutzer, welcher die Informationen abfragen kann. Andererseits der DB-Administrator, welcher die Daten abfragen, aber auch verändern und neue hinzufügen kann. Diese Anwendungsfälle sind in der Abbildung 3 abgebildet.



./pictures/todo.jpg

Abbildung 2: Visualisierung des Anwendungsfalls

Welche Daten werden integriert migriert und wie werden sie aufbereitet? Wie interagiert der Benutzer mit der Datenbank?

### 3 Datenmodellierung

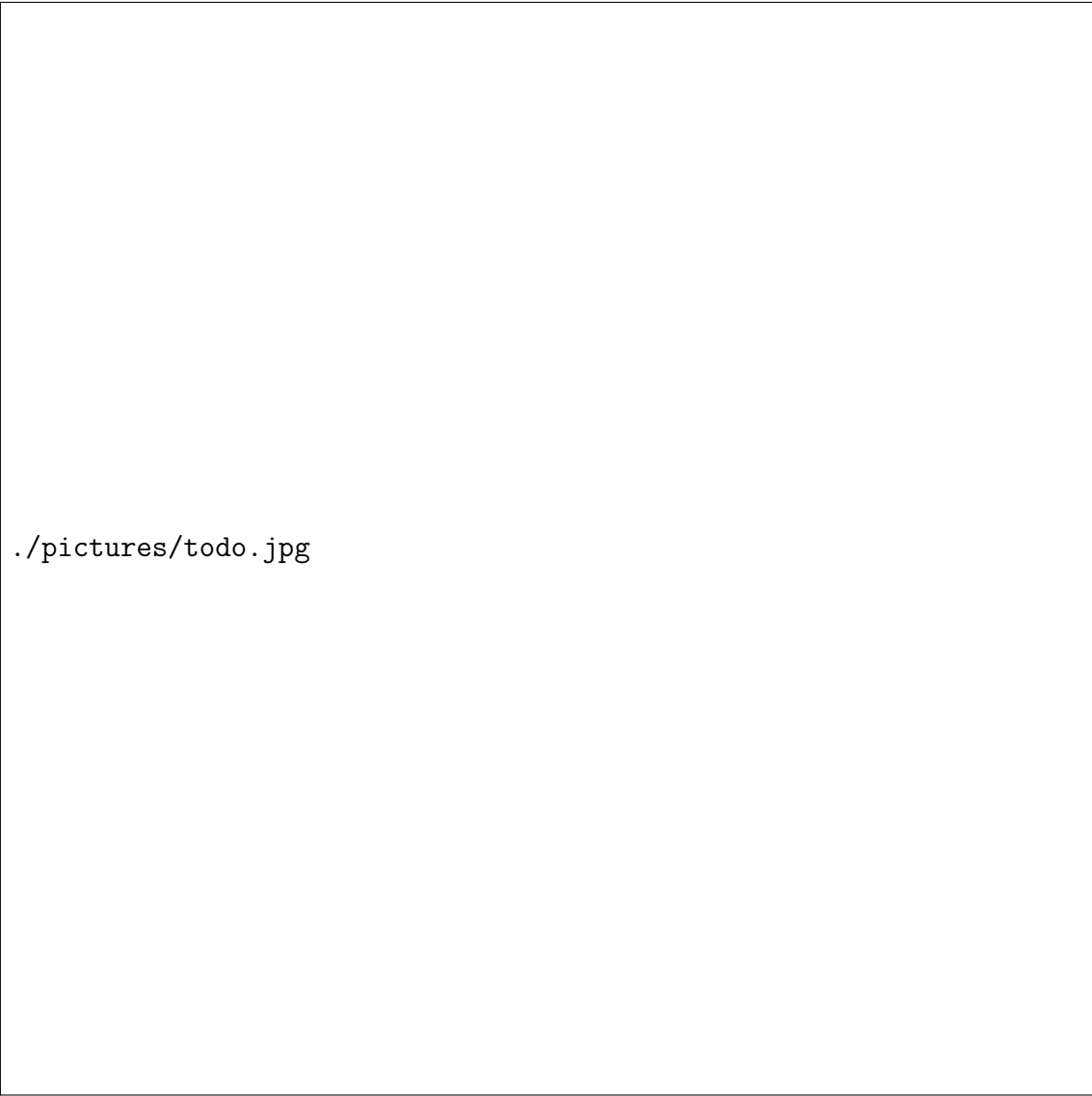
MongoDB ist eine dokumentorientierte Datenbank. Dabei werden die Daten in JSON ähnlichen Dokumenten verwaltet. Das bedeutet, dass die Daten nicht relational verwaltet werden. So kann zum Beispiel ein Tupel einer relationalen Datenbank als Dokument in der dokumentorientierten Datenbank abgebildet werden. Die Attribute und die dazugehörigen Werte werden dabei in Schlüssel-Wert Paare abgebildet.

Unserem Projekt liegt das ER-Schema aus 1 zugrunde. Da MongoDB eine NoSQL und keine relationale Datenbank ist, kann das ER-Schema nicht direkt in dieser Form in der Datenbank abgebildet werden.

Eine Komposition bedeutet, dass z.B. in der NoSQL Datenbank als "teil von" abgebildet wird. Eine Aggregation bedeutet, dass diese in Beziehung als Referenz abgebildet wird.

Im ER-Diagramm wird zum Beispiel die Komplex-Komplexe-Beziehung zwischen Vorlesungen und Studenten als eigene Tabelle abgebildet. In unserer Datenbank wird diese Beziehung in zwei einfach-komplexe Beziehung abgebildet. Somit besitzen Studenten eine Einfach-Komplexe Beziehung zu den Vorlesungen. Diese Beziehung wird im JSON -Format im Feld Hören ersichtlich. Die vom Studenten besuchten Vorlesungen werden als Vorlesungsnummer in einem Array referenziert.





`./pictures/todo.jpg`

Abbildung 3: Datenmodell

```
{  
  "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. Darin wird für jeden Professor nachgeschlagen, welche Vorlesungen dieser hält. Anhand dieser wird die Anzahl SWS Punkte ermittelt, die der Professor unterrichtet. Zusätzlich wird gezählt wie viele Studenten dieser Vorlesungen zuhören. Wenn die Query in MySQL ausgeführt wird, ist das Resultat folgendes:

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

Abbildung 4: Query Resultat in MySQL ?

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

PersNr	Name	Rang	Raum	Anz. Studenten	Anz. SWS
2125	Sokrates	C4	226	0	0
2126	Russel	C4	232	0	0
2127	Kopernikus	C3	310	0	0
2133	Popper	C3	52	0	0
2134	Augustinus	C3	309	0	0
2136	Curie	C4	36	0	0
2137	Kant	C4	7	0	0

PersNr	Name	Rang	Raum	Anz. Studenten	Anz. SWS
2125	Sokrates	C4	226	4	10
2126	Russel	C4	232	2	8
2127	Kopernikus	C3	310	0	0
2133	Popper	C3	52	1	2
2134	Augustinus	C3	309	2	2
2136	Curie	C4	36	0	0
2137	Kant	C4	7	4	8

Abbildung 5: 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. Desweiteren garantiert MongoDB keine Integrität der Referenzen. Dies wird klar, wenn man vergleicht, welchen Regeln die relationale Datenbank und die MongoDB 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.

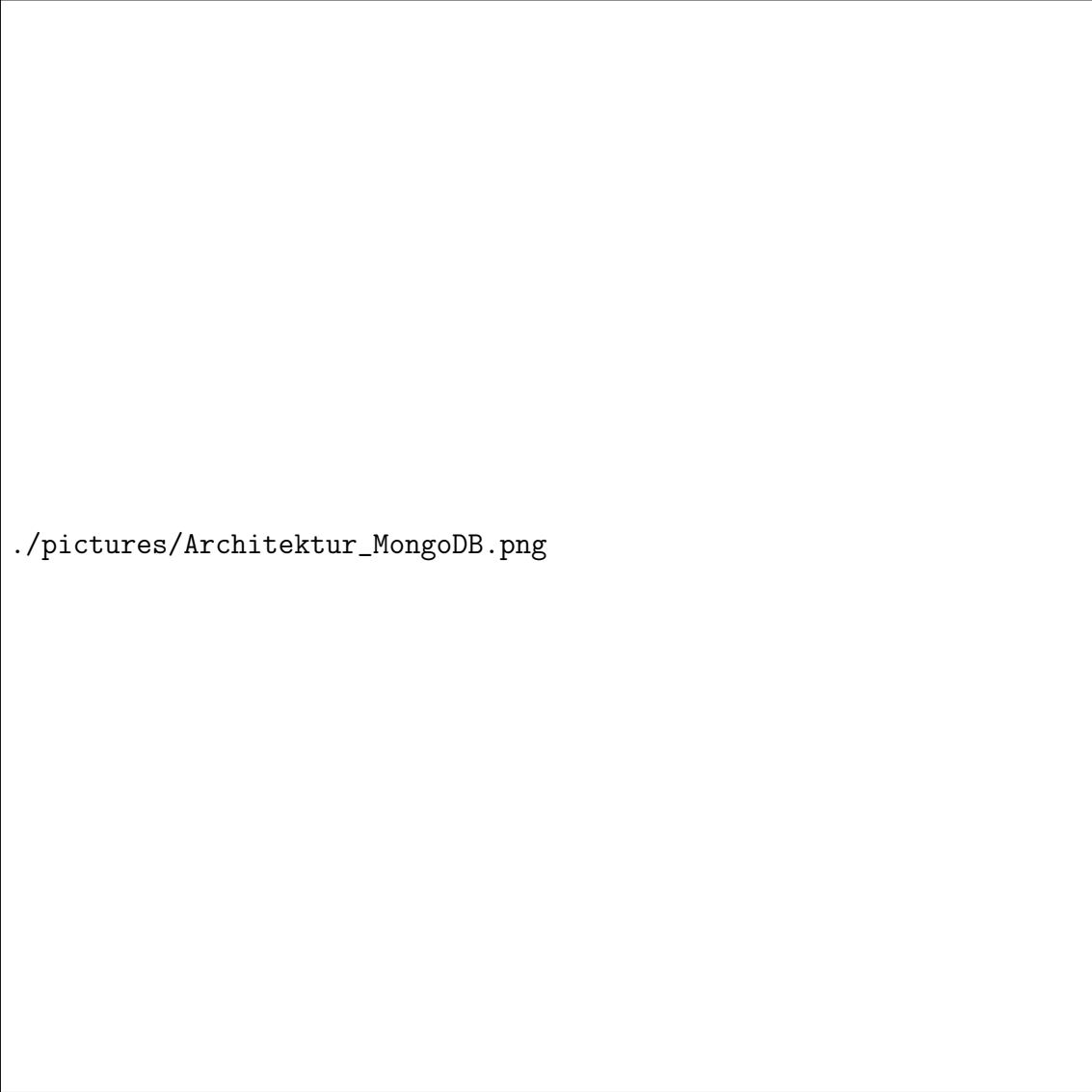
- Basically Available: Die Datenbank sollte meistens laufen.
- Eventually Consistent: Die Konsistenz der Daten wird nicht unmittelbar nach der Operation gewährleistet. Sie kann verzögert eintreten.

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. In unserem Fall werden keine Transaktionen verwendet,

da nur lesend auf die Daten zugegriffen werden. Da nur gelesen wird, und 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 Dokumenten. Damit der Router weiss, an welche Replica Sets er die Anfrage weiterleiten muss, stellt er wiederum eine Anfrage an die Config Server. Diese Antworten entsprechen. Die Config Server enthalten also die Metadaten über die Dokumentenverteilung in den Replica Sets. In der Abbildung 6 ist die Architektur von MongoDB visualisiert. In der in diesem Projekt eingesetzte MongoDB Instanz, laufen alle drei Server auf dem selben physischen Rechner. Dies wurde so gewählt, da Sammlung an Dokumenten klein ist. Desweiteren wird auch nur sporadisch auf die Daten zugegriffen. Sollte sich in Zukunft einer dieser Parameter ändern, kann darüber nachgedacht werden, die Server auf verschiedene physischen 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 währenddem 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 sehr schwierig, dass 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 Aggregationen in die Dokumente umgesetzt werden. Dies vermindert die Anzahl der Referenzierungen und ermöglicht es damit, dass eine horizontale Skalierung viel einfacher zu bewerkstelligen ist, als bei einer relationalen Datenbank. Dies führt natürlich auch Nachteile mit sich. Jedes mal wenn ein referenziertes Tupel als Aggregation in ein Dokument gespeichert wird, erhöht sich die



`./pictures/Architektur_MongoDB.png`

Abbildung 6: Architektur von MongoDB. ?

Datenmenge. Desweiteren 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 Funktionalität 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 weiterer Server fehlt, der ein Replikat der Datenbank enthält.



## 7 Schlussfolgerungen