NoSQL-Datenbanken

Dokumentenorientierte Datenbanken

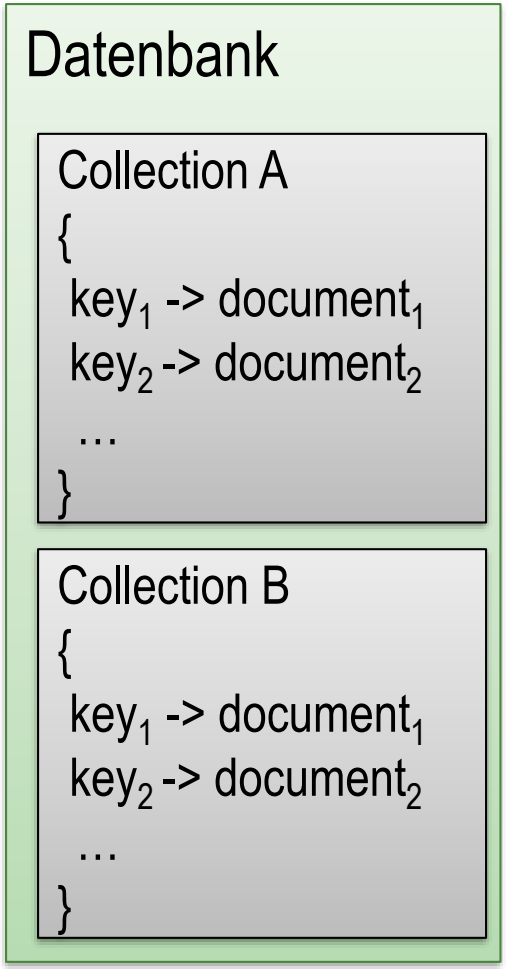
Johannes Zschache
Sommersemester 2019

Abteilung Datenbanken, Universität Leipzig
http://dbs.uni-leipzig.de

Inhaltsverzeichnis: Dokumentenorientierte DB

Dokumentenorientierte Datenbanken

- Key-Value Stores mit Dokumenten als Werten
  - Jedes Dokument wird über einen eindeutigen (Dokumenten-)Schlüssel referenziert
  - Zugriff nicht nur über Schlüssel
  - Struktur der Dokumente wird von DB berücksichtigt
- Dokument = Liste von Attributen
  - Attribut = Schlüssel-Wert-Paar
  - Schlüssel: eindeutige Zeichenkette
  - Wert: beliebiger Datentyp, inkl. Listen und Dokumente
  - Eindeutiges ID-Attribut (Primary Key)
  - Semistrukturiert
  - z.B. JSON/BSON, XML
- Dokumente werden in *Collections* zusammengefasst
  - Gleicher Entitätstyp und/oder ähnliche Struktur
  - Abstraktion je nach Anfragemuster
  - Unterstützung von Indexierung



JSON - Java Script Object Notation

- Standardisiert durch Ecma International
- Lesbare Datenstruktur
- JSON Objekt: { key: value, key: value, …}
- Key: String
- Value:
  - Zahl, String, Boolean
  - Array: [Value1, Value2, …
  - JSON Objekt
  - null

```
{ "_id": 1,
  "name": "fish.jpg",
  "user": "bob“,
  "compressed": FALSE,
  "info": { "width": 100, "height": 200, "size": 12345},
  "tags": ["tuna","shark"]
}
```

- Traversal: pfadbasiert, entlang den Keys, z.B. “info.size”
- Binary JSON (BSON): Übertragung über Netzwerk, effiziente Speicherung
- JSON Schema: Validierung

Dokumentenorientierte DB: Eigenschaften

- Effiziente Verarbeitung beliebig verschachtelter Daten
- Schemafrei
  - Unterschiedliche Attribute pro Dokument
  - Mehr Verantwortung auf Seiten der Anwendung
- Flexible Skalierbarkeit
- Zugriff:
  - Aktualisierung des gesamten Dokuments oder einzelner Attribute
  - Anfragen nach (Teilen von) Dokumenten über Dokumentenschlüssel oder Attribute
  - Spezielle Anfragesprachen
  - Löschen: Keine referentielle Integrität
- Unterschiedliche Herangehensweisen bzgl. Designprinzipien wie Indexierung, Abfragen, Beständigkeit, Replikation, Konsistenz, …

Denormalisierung

- Einfügen redundanter Daten
- Über Arrays und eingebettete Dokumente
- Vorteile:
  - Geringere Latenzzeiten
  - Partitionierung einfacher
- Nachteile:
  - Gewährleistung der Correctness (C in ACID) durch Anwendung
  - Gefahr des Datenverlusts
- Abwägung des Ausmaßes an Denormalisierung über erwartete Nutzeranfragen

```
{ "_id" : 3,
  "first_name" : "Danyette",
  "last_name" : "Flahy",
  "orders" : [
    { "orderId" : 1,
      "orderDate" : "2017-03-09",
      "lineItems" : [
        { "productId" : 158,
          "productName" : "Cup - 8oz",
          "price" : 56.92},
        { "productId" : 82,
          "productName" : "Cup - 16oz",
          "price" : 62.54}]},
    { "orderId" : 32,
      "orderDate" : "2018-01-03",
      "lineItems" : [
        { "productId" : 158,
          "productName" : "Cup - 8oz",
          "price" : 34.12}]}]}
```

DB Ranking

Quelle: http://db-engines.com/en/ranking/

☐ include secondary database models

47 systems in ranking, May 2019

Rank May 2019	Rank Apr 2019	Rank May 2018	DBMS	Database Model	Score May 2019	Score Apr 2019	Score May 2018
1.	1.	1.	MongoDB	Document	408.07	+6.10	+65.96
2.	2.	2.	Amazon DynamoDB	Multi-model	55.93	-0.08	+11.74
3.	3.	3.	Couchbase	Document	34.67	-1.61	+2.26
4.	4.	↑ 5.	Microsoft Azure Cosmos DB	Multi-model	27.59	+1.32	+10.06
5.	5.	↓ 4.	CouchDB	Document	19.11	-1.32	-0.31
6.	6.	6.	MarkLogic	Multi-model	14.05	-0.42	+3.66
7.	7.	7.	Firebase Realtime Database	Document	11.28	+0.28	+4.79
8.	8.	8.	OrientDB	Multi-model	6.37	+0.18	+1.12
9.	↑ 11.	↑ 16.	Google Cloud Firestore	Document	4.99	+0.68	+2.76
10.	↑ 12.	↑ 11.	ArangoDB	Multi-model	4.79	+0.50	+1.09
11.	↓ 9.	↑ 13.	RavenDB	Document	4.74	+0.09	+1.82
12.	↓ 10.	12.	Google Cloud Datastore	Document	4.72	+0.28	+1.55
13.	13.	↓ 9.	RethinkDB	Document	4.42	+0.25	+0.19
14.	14.	↓ 10.	Cloudant	Document	3.95	+0.05	+0.05
15.	15.	15.	PouchDB	Document	3.03	+0.17	+0.59
16.	16.	↓ 14.	Apache Drill	Multi-model	2.97	+0.27	+0.18
17.	17.	17.	CloudKit	Document	2.56	+0.37	+0.80
18.	18.	↑ 20.	Mnesia	Document	1.50	-0.17	+0.55
19.	19.	↓ 18.	Datameer	Document	1.41	-0.08	+0.23
20.	20.	↓ 19.	FoundationDB	Multi-model	1.01	-0.13	-0.11

Inhaltsverzeichnis: Dokumentenorientierte DB

MongoDB

- First Release: 2009, aktuelle Version: 4.0
- Unterstützung eines „hu**mongo**us“ Umfangs an Daten und Anfragen
- Skalierbarkeit über Partitionierung; Master-Slave Replikation
- Linearisierbarkeit (kausale Konsist. oder Eventual Consistency möglich)
- ACID Garantien (auch über mehrere Dokumente **innerhalb Partition**)
- Flexibles Datenmodell
  - JSON-Dokumente, gespeichert als BSON
  - Denormalisiert und schemafrei (Datentypen und Validierung möglich)
  - Referenzen möglich, aber keine referentielle Integrität
- Vielseitige Anfragesprache
  - Einfache Anfragen
  - Aggregation Framework
  - JavaScript für komplexe Anfragen mit MapReduce
- Anbindungen für viele Programmiersprachen

MongoDB: Schema

- MongoDB ist schemafrei
- Vorteile eines Schemas:
  - Klare Struktur und Dokumentation
  - Weniger Fehler/Aufwand durch Anwender
- ab Version 3.6 : JSON-Schema-Validator

```
db.createCollection( "people" , {
  validator: { $jsonSchema: {
    bsonType: "object",
    required: [ "name", "email" ],
    properties: {
      name: { bsonType: "string",
              description: "required and a string" },
      email: { bsonType: "string",
               pattern: "^.+\@.+$",
               description: "required and a valid email" },
      year_of_birth: { bsonType: "int",
                       minimum: 1900,
                       maximum: 2018,
                       description: "in range 1900-2018" },
      gender: { enum: [ "M", "F" ],
                description: "can be only M or F" } } } } } )
```

Quelle: https://www.percona.com/blog/2018/08/16/mongodb-how-to-use-json-schema-validator/

MongoDB: Demo

```
db.towns.insert({
  name: "New York",
  population: 22200000,
  last_census: ISODate("2009-07-31"),
  famous_for: [ "statue of liberty", "food" ],
  mayor : { name : "Michael Bloomberg",
            party : "I"
  }})
...
db.countries.insert({
  _id : "us",
  name : "United States",
  exports : {
    foods : [
      { name : "bacon", tasty : true },
      { name : "burgers" }
    ]
  }})
...
```

MongoDB: Anfragen

- Mongo Shell: `mongo`
- Kollektionen: `show collections`
- Inhalt einer Kollektion:

```
db.towns.find()
db.towns.find().pretty()
```

- Alle verfügbaren Funktionen:

```
db.help()
db.towns.help()
```

```
db.towns.count()
db.towns.find({ _id : ObjectId("5d00c237e7742bf2cb37da59") })
db.towns.find({ name : "New York"})
db.towns.find({ name : "New York" }, { name : 1 })
db.towns.find({ name : "New York" }, { name : 1, _id:0 })
db.towns.find({ name : "New York" }, { name : 0 })
db.towns.find(
  { name : /^P/, population : { $lt : 10000 } },
  { name : 1, population : 1 })
```

MongoDB: Anfragen

Arrays

```
db.towns.find(
  { famous_for : 'food' },
  { _id : 0, name : 1, famous_for : 1 }
)
db.towns.find(
  { famous_for : { $all : ['food', 'beer'] } },
  { _id : 0, name:1, famous_for:1 }
)
db.towns.find(
  { famous_for : { $nin : ['food', 'beer'] } },
  { _id : 0, name : 1, famous_for : 1 }
)
```

MongoDB: Anfragen

Verschachtelte Attribute

```
db.towns.find(
  { 'mayor.party' : 'I' },
  { _id : 0, name : 1, mayor : 1 }
)
db.countries.find(
  { 'exports.foods.name' : 'bacon',
    'exports.foods.tasty' : true },
  { _id : 0, name : 1 }
)
db.countries.find(
  { 'exports.foods' : {
      $elemMatch : { name : 'bacon', tasty : true }
    } }, { _id : 0, name : 1 }
)
```

MongoDB: Anfragen

Logische Operatoren

```
db.towns.find(
  { 'mayor.party' : { $exists : false } },
  { _id : 0, name : 1, mayor : 1 }
)

db.countries.find(
  { $or : [
      { _id : "mx" },
      { name : "United States" }
    ]
  }, { _id:1 }
)
```

$not, $ne, $lt, $lte, $gt, $gte, $exists, $all, $in, $nin, $or, $size, ...

MongoDB: Anfragesprache

Teilmenge von SQL kann durch find-Funktion abgebildet werden

SQL	Mongo
`SELECT name, status FROM users WHERE age=33;`	
`SELECT * FROM users WHERE age>33 AND age <= 40;`	
`SELECT * FROM users WHERE status != "A" OR age = 50;`	
`SELECT * FROM users WHERE name LIKE "%Joe%";`	
`SELECT COUNT(age) from users;`	

MongoDB: Updates

```
db.towns.update(
  { _id : ObjectId("5a5f2") },
  { $set : { state : "OR" } }
)
db.towns.update(
  { _id : ObjectId("5a5f2") },
  { $inc : { population : 1000} }
)
```

Weitere Operatoren	Beschreibung
$unset	Entfernen eines Attributs
$mul	Multiplikation mit einem Faktor
$max, $min	Aktualisierung falls größer/kleiner als aktueller Wert
$push, $pushAll, $addToSet	Hinzufügen eines Elements zu Array
$pop, $pull, $pullAll	Entfernen eines Elements von Array

MongoDB: Updates

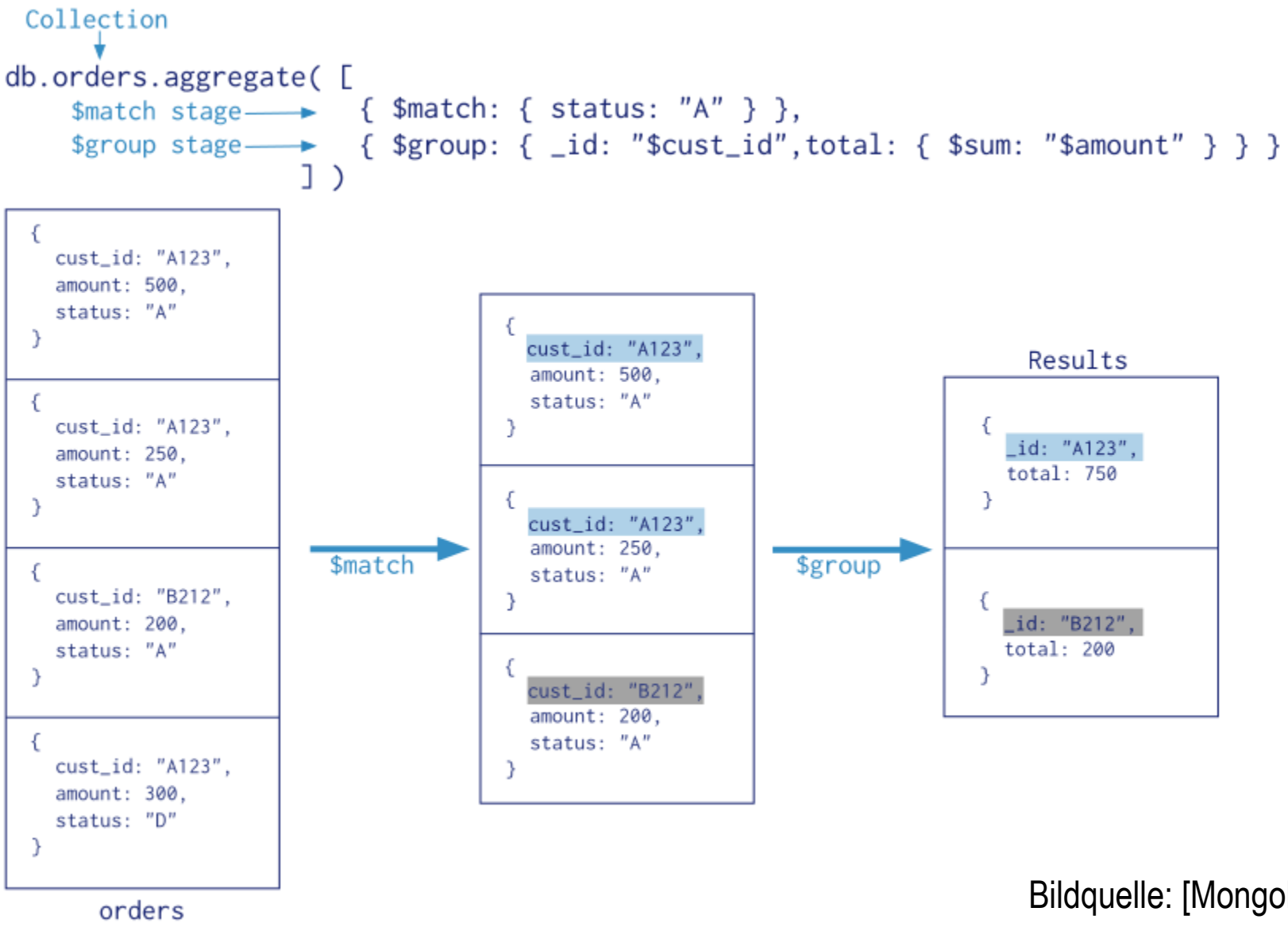
Aktualisierung mehrerer Elemente eines Array

```
db.countries.update({_id: "us"}, {
	$set: {
		"exports.foods.$[].price": 10
	}})

db.countries.update({_id: "us"}, {
	$inc: {
		"exports.foods.$[item].price": 5
	}}, {
	arrayFilters: [{
		"item.tasty": true
	}]})
```

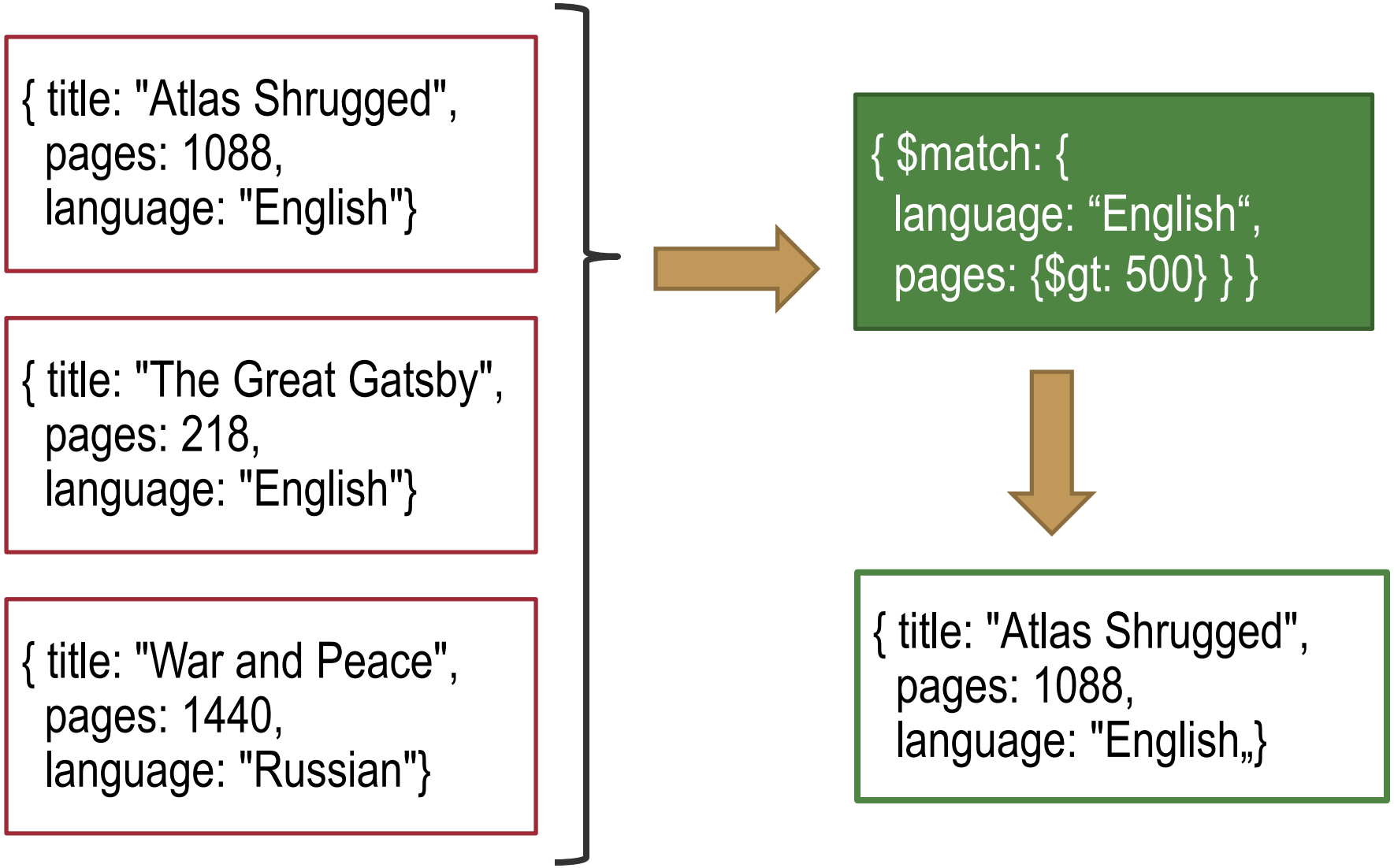
Inhaltsverzeichnis: Dokumentenorientierte DB

MongoDB: Aggregation Framework



Bildquelle: [MongoDB]

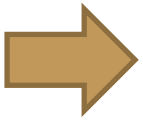
Filter: $match



[Hou14]

Umformung: $project

```
{ _id: 375,
  title: "Great Gatsby",
  ISBN: "9781857150193",
  available: true,
  pages: 218,
  chapters: 9,
  subjects: [
    "Long Island",
    "New York",
    "1920s"
  ],
  language: "English" }
```

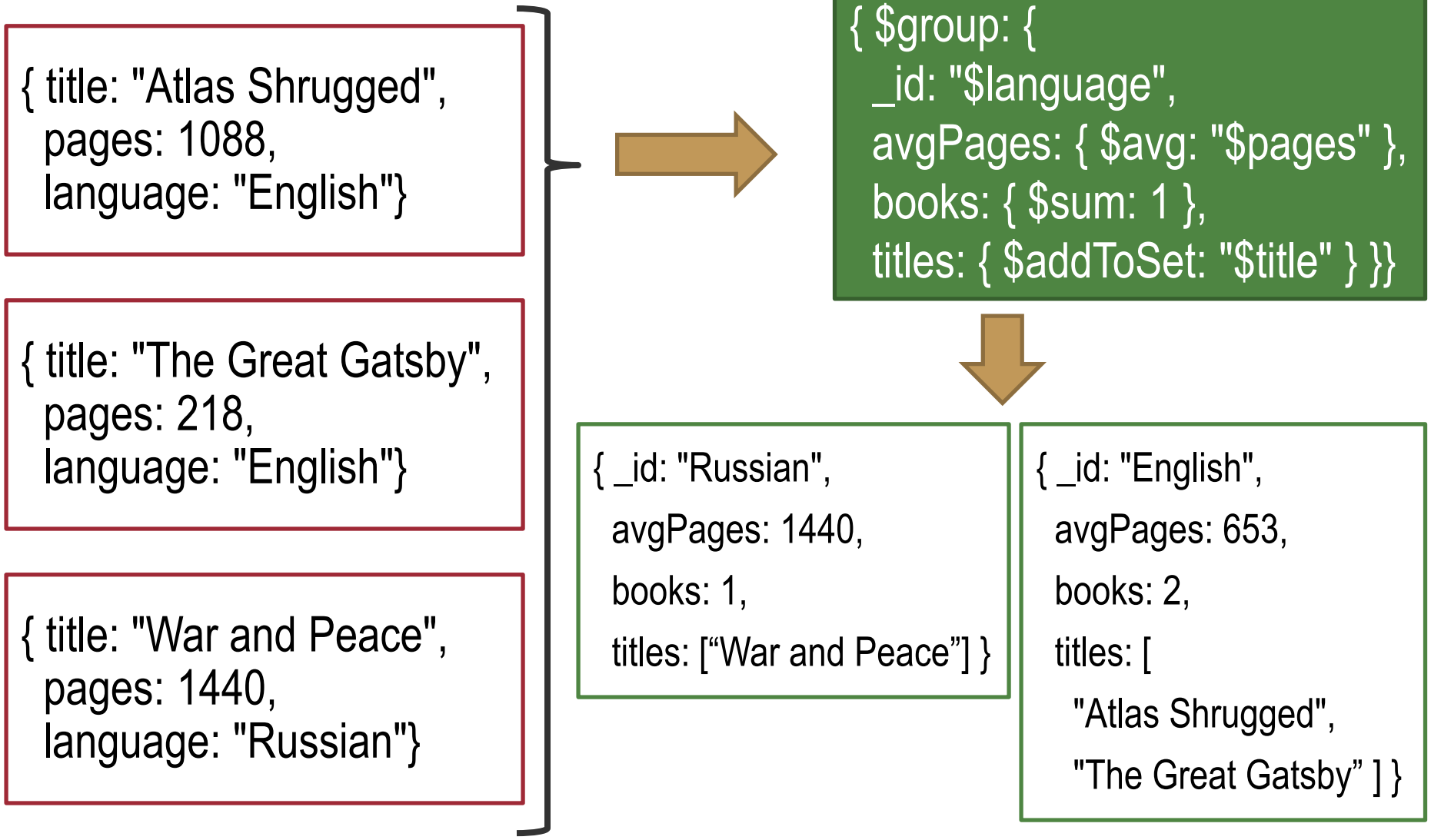


```
{ $project: {
  _id: 0,
  title: 1,
  avgChapterLength: {
    $divide: ["$pages",   "$chapters"] },
  stats: { pages: "$pages",
           lang: "$language", }}}
```

```
{ title: "Great Gatsby",
  avgChapterLength: 24.2222,
  stats: { pages: 218,
           lang: "English“ }}
```

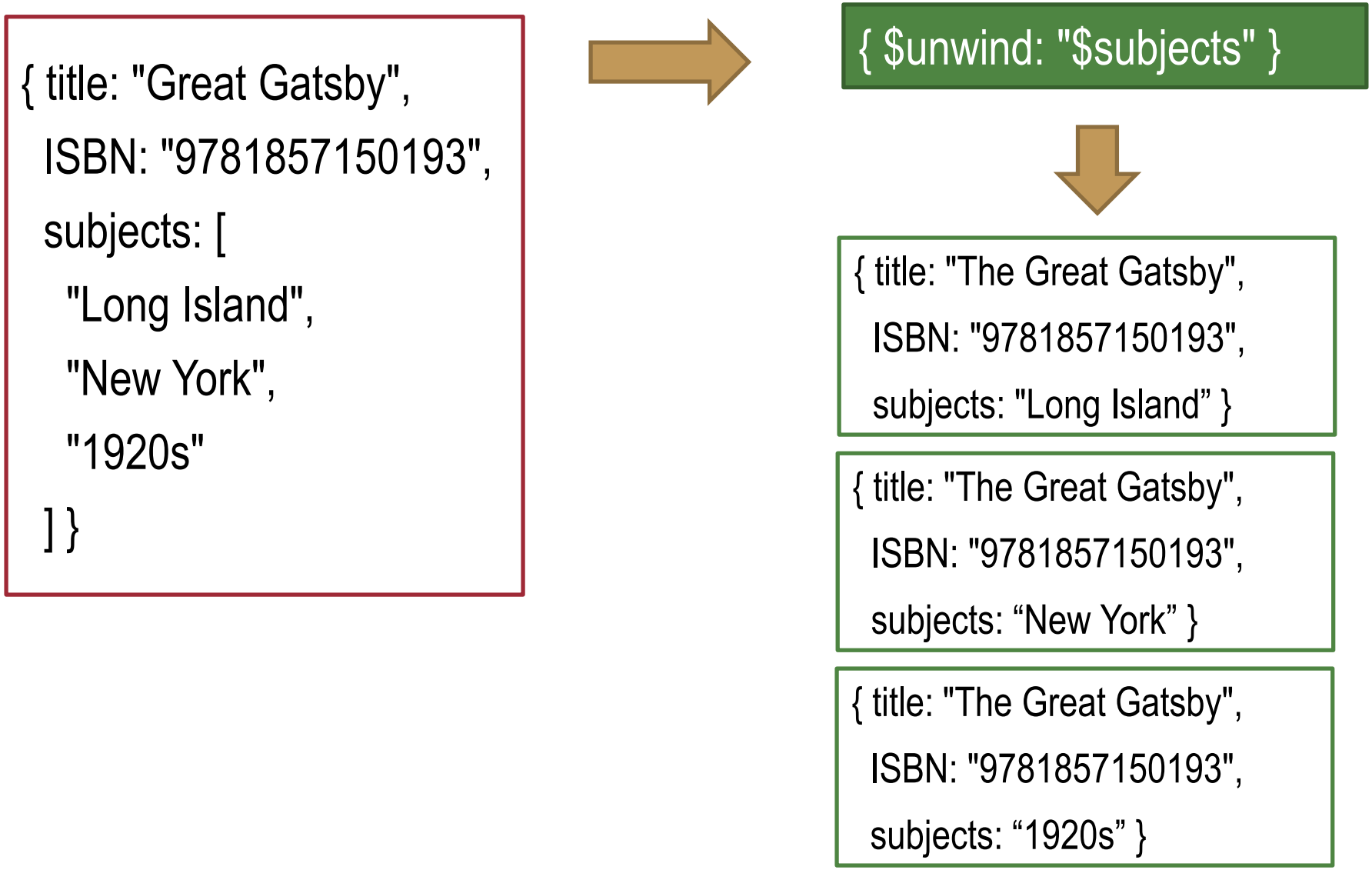
[Hou14]

Gruppierung: $group



[Hou14]

Expandieren: $unwind

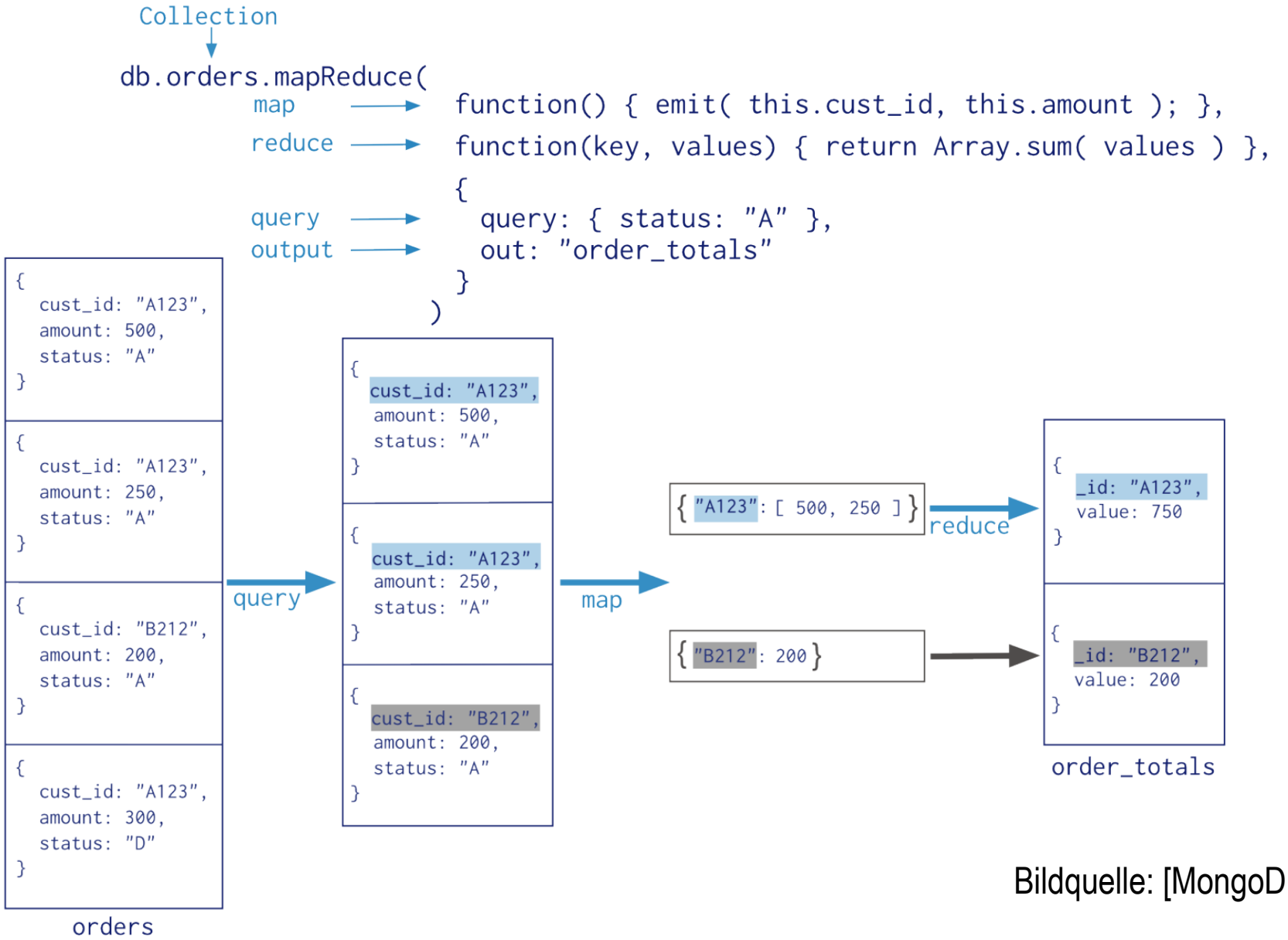


[Hou14]

MongoDB: Aggregation Framework

SQL	Mongo
SELECT user_id AS _id, avg(score) AS avg_score FROM users WHERE status = "A" GROUP BY user_id HAVING avg_score < 10 ORDER BY avg_score;	

MongoDB: Anfragen mit MapReduce

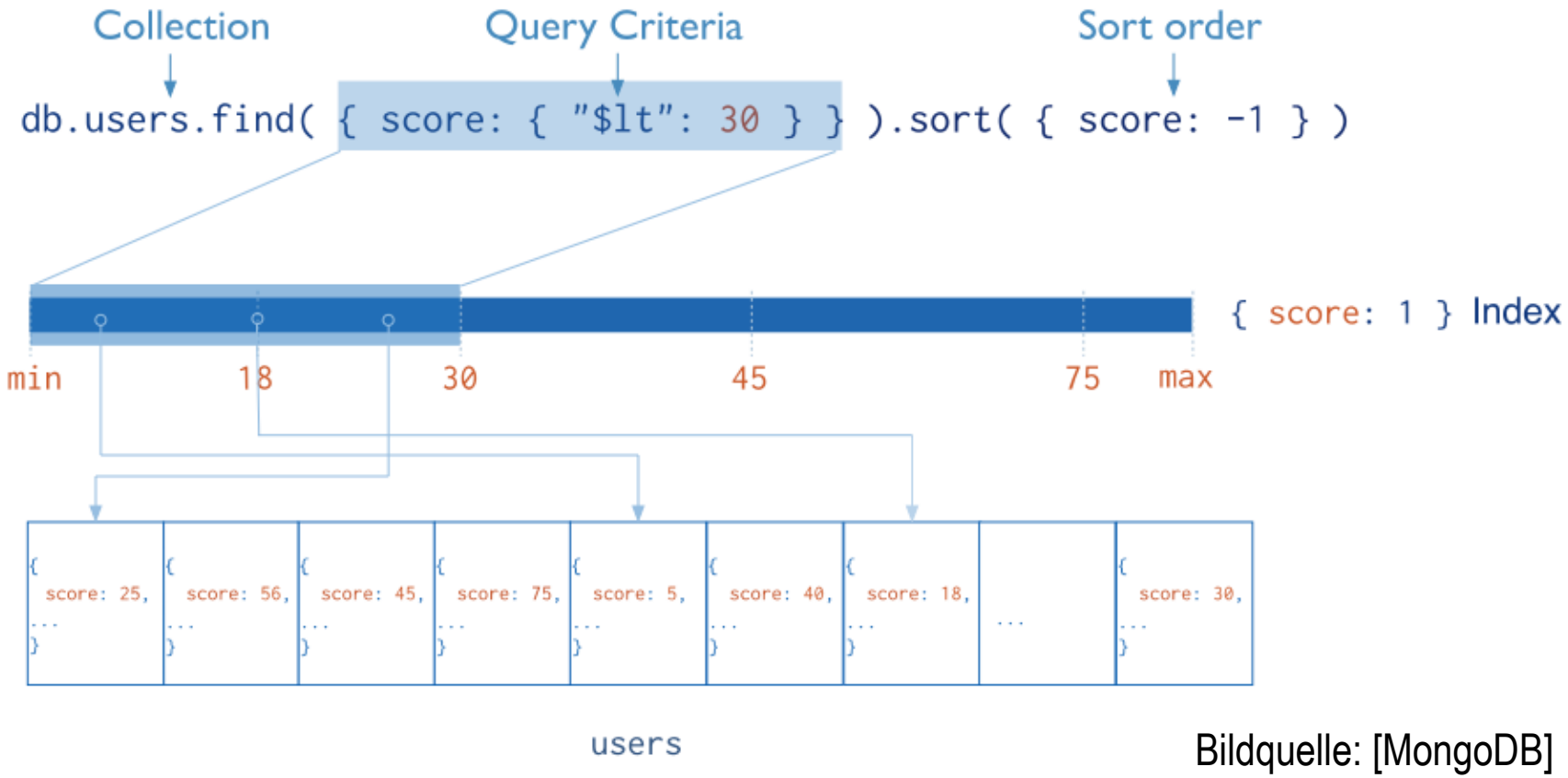


Bildquelle: [MongoDB]

Inhaltsverzeichnis: Dokumentenorientierte DB

MongoDB: Index (1)

- Indizes: mehr Leistung für Leseanfragen möglich
- z.B. B-Bäume, Aufbau und Verwendung analog zu RDBS
- Schnelleres Suchen, *Sortieren* und *Joins*



Bildquelle: [MongoDB]

MongoDB: Index (2)

```
db.phones.getIndexes()
// langsam, falls kein Index auf "display":
db.phones.find({display: "+4 800-5550001"})
db.phones.createIndex(
  { display : 1 }, { unique : true}
)
// nun viel schneller:
db.phones.find({display: "+4 800-5550001"})
```

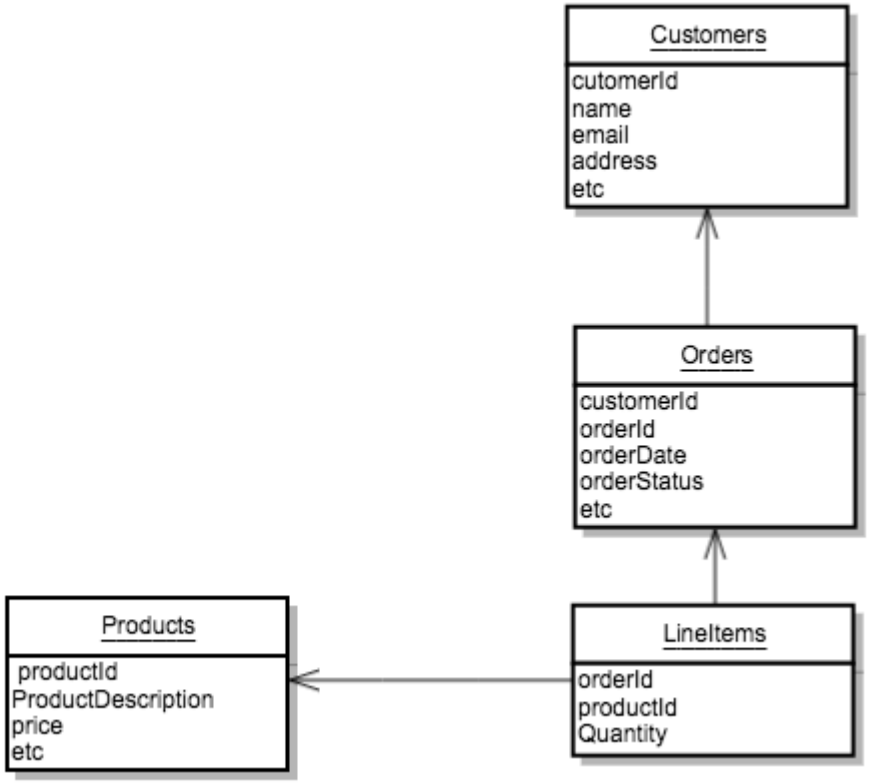
- Ausführungsplan und -zeit über EXPLAIN-Operator:

```
db.phones.find(...).explain(verbosity="executionStats")
```

- MongoDB Profiler für Messungen in Test-/Produktionsumgebung: https://studio3t.com/whats-new/mongodb-query-performance/

Design des Schemas

- Entscheidend für Leistung
- Orientierung an Anwendung
- Bücher: [Kval14], [Cop13]
- **Beispiel**: Leistungsunterschiede zwischen zwei Extremen [Har18]



Design des Schemas: Normalisierung (Linked)

- Customer:

```
{ "_id" : 3,
  "first_name" : "Danyette",
  "last_name" : "Flahy",
  "email" : "dflahy2@nsl.com"
}
```

- Order:

```
{ "_id" : 1,
  "orderDate" : "2017-03-09",
  "orderStatus" : 0,
  "customerId" : 3
}
```

- LineItem:

```
{ "_id" : "5a7935",
  "orderId" : 1,
  "prodId" : 158,
  "itemCount" : 48
}
```

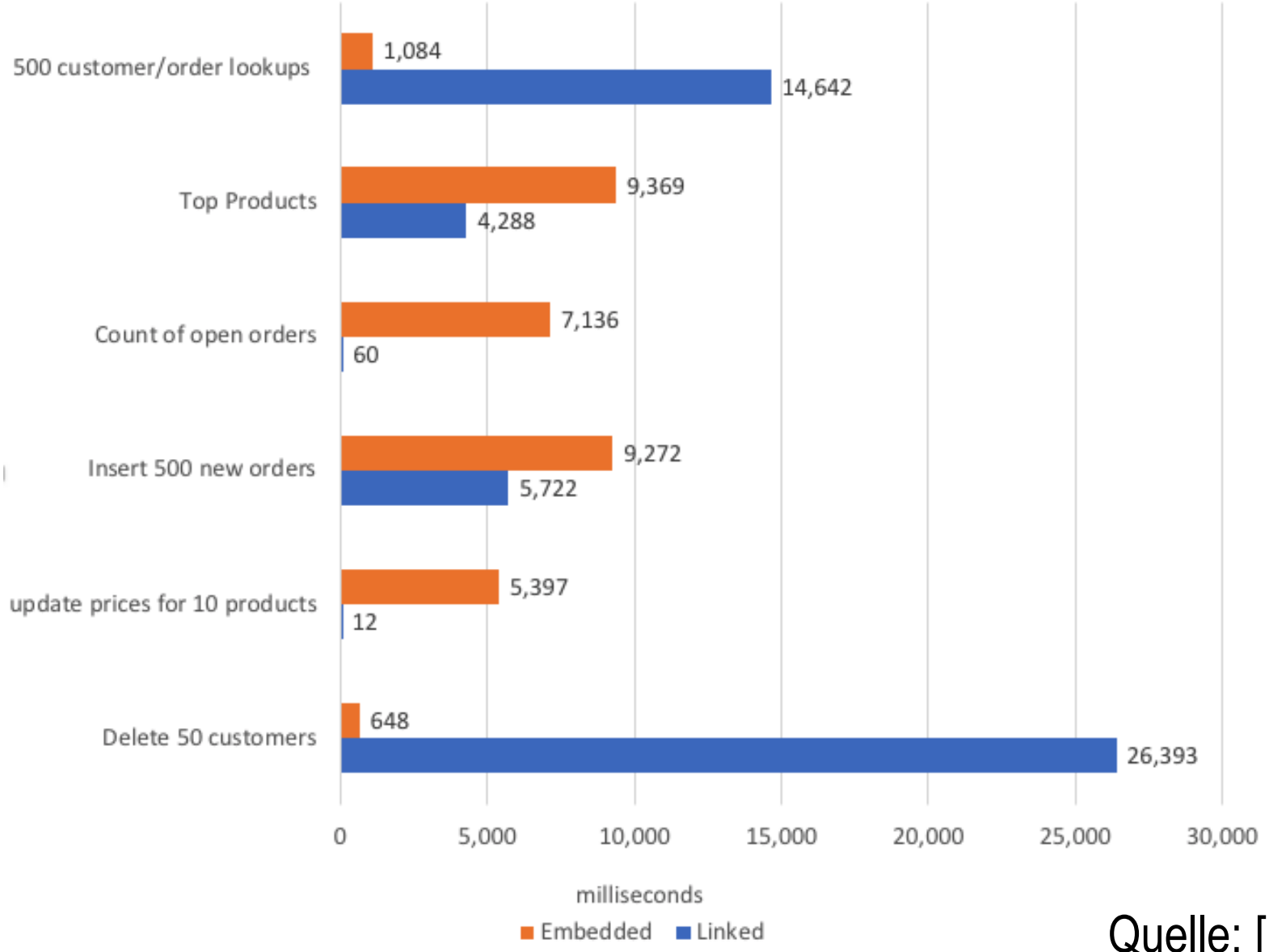
- Product

```
{ "_id" : 158,
  "productName" : "Cup - 8oz",
  "price" : 56.92,
  "color" : "Turquoise",
  "Image" : "dummyimage.jpg"
}
```

Design des Schemas: Denormalisierung (Embedded)

```
{ "_id" : 3,
  "first_name" : "Danyette",
  "last_name" : "Flahy",
  "email" : "dflahy2@nsl.com",
  "orders" : [
        { "orderId" : 1,
          "orderDate" : "2017-03-09",
          "orderStatus" : 0,
          "lineItems" : [
                { "productId" : 158,
                  "productName" : "Cup - 8oz",
                  "price" : 56.92,
                  "color" : "Turquoise",
                  "Image" : "dummyimage.jpg",
                  "itemCount" : 48
                }, ... ]
        }, ... ]
}
```

Design des Schemas: Vergleich der Leistung



Quelle: [Har18]

Übung

- Installation MongoDB: https://docs.mongodb.com/manual/installation/
- Importieren Sie die Datensätze *business2006.json*, *review2006.json* und *user2006.json* mittels mongoimport als drei separate *Collections* (normalisierte Form). Alternativ ist auch die Arbeit mit dem vollständigen Datensatz möglich.
- Erstellen Sie über das Aggregation Framework (Joins über $lookup) eine weitere *collection*, welche alle drei Dokumente "embedded" bzw. in denormalisierter Form enthält.
  - Wählen Sie z.B. die Unternehmen als Wurzelelemente, welche die jeweils zugehörigen Reviews als Array enthalten. Jedes Review enthält dann auch ein Attribut mit allen Informationen des Nutzers.
  - Diese Variante ist evtl. nicht möglich, da Dokumente in MongoDB bezüglich ihrer Größe beschränkt sind. Wählen Sie in diesem Fall die Reviews als Wurzelelemente.
- Siehe auch: https://medium.com/dbkoda/studies-in-mongodb-schema-design-pt-1-9ebef72ae7b9

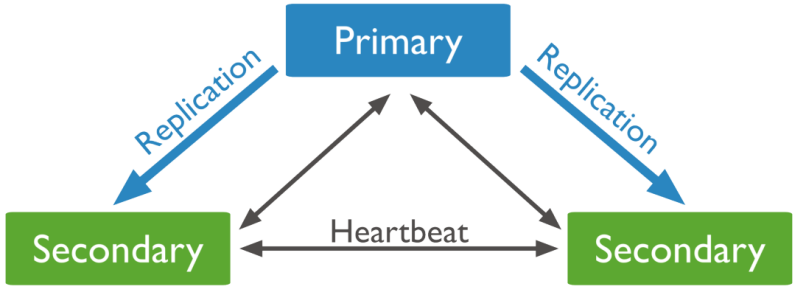
Übung

- Vergleichen Sie die beiden Entwürfe (normalisiert und denormalisiert) in Bezug auf deren Ausführungszeiten der folgenden Anfragen.

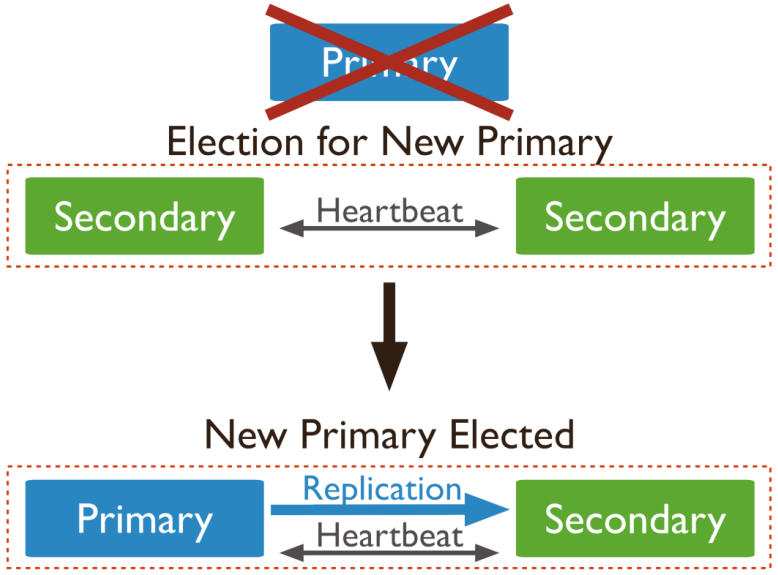
1. Sortieren Sie die Städte nach Anzahl der Unternehmen!
2. Laden Sie die Texte der jeweils neuesten Rezension der, in "Las Vegas" vorkommenden, Unternehmen der Kategorie "Burgers„!
3. Berechnen Sie für jede Kategorie die Anzahl der Rezensionen! Welche Kategorie wird am häufigsten bewertet?
4. Rufen Sie Informationen zu den 10 bestbewerteten Unternehmen ab: deren Name, Stadt und Bewertung (durchschnittliche Anzahl der Sterne)!
5. Finden Sie für jedes Unternehmen die Namen der Nutzer, welche die Rezension mit der schlechtesten Bewertung geschrieben haben!
6. Erweitern Sie den Bereich der möglichen Sterne von 5 auf 10! Verdoppeln Sie dazu die Sterne aller Rezensionen. Aktualisieren Sie auch die durchschnittliche Anzahl der Sterne in den Dokumenten der Unternehmen und Nutzer.
7. Ersetzen Sie, in allen Einträgen, die Kategorie "Burgers" durch "Burger".

Inhaltsverzeichnis: Dokumentenorientierte DB

MongoDB: Replikation



- Asynchrone Master-Slave Replikation
- Replica Set: mind. 3 Knoten (max. 50)
- Schreib- **und** Lesezugriffe über “Primary”
  - Oplog (operations log) leitet Daten weiter an “Secondaries”
  - „Write/Read Concern“: Linearisierbarkeit, kausale Konsistenz oder Lesen über Secondary einstellbar
- Wahl eines „Primary”
  - Zu Beginn und bei Ausfall
  - Unterbrechung der Schreiboperationen („Retryable Writes“ ab MongoDB 3.6)
  - Mehrheit wird benötigt
  - Wahl über Mehrheit der Stimmen, aktuellste „Optime“ und höchste „Priority“
  - Neue/wiederkehrende Knoten werden durch Replikation aktualisiert



Bildquelle: [MongoDB]

MongoDB: Setup Replikation (lokal)

```
mkdir ./mongo1 ./mongo2 ./mongo3
```

```
mongod --replSet rs1 --dbpath ./mongo1 --port 27011
```

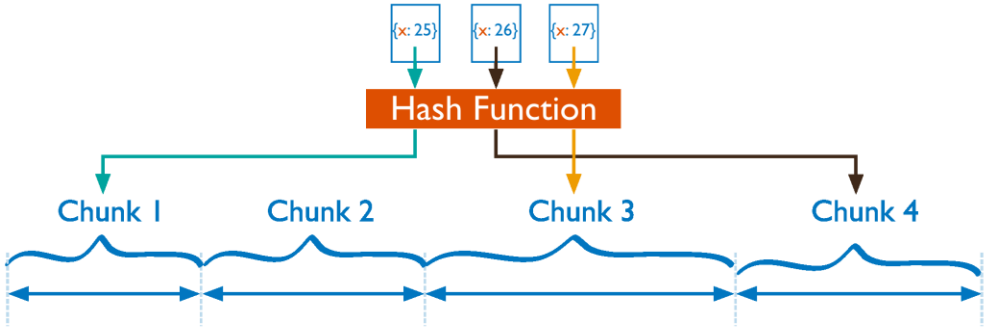
```
mongod --replSet rs1 --dbpath ./mongo2 --port 27012
```

```
mongod --replSet rs1 --dbpath ./mongo3 --port 27013
```

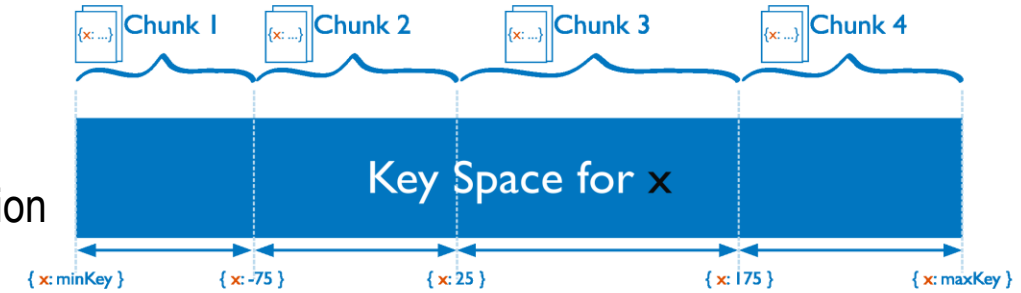
```
mongo localhost:27011
rs.initiate({
  _id: 'rs1',
  members: [
    {_id: 1, host: 'localhost:27011'},
    {_id: 2, host: 'localhost:27012'},
    {_id: 3, host: 'localhost:27013'}
  ]
})
rs.status()
```

MongoDB: Sharding (1)

- Aufteilung einer Collection über mehrere Server
- Sharding-Attribut
  - Immutable, hohe Kardinalität
  - Einfach oder zusammengesetzt
  - Existenz in allen Dokumenten



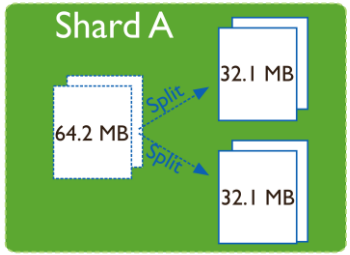
- Hash-basiert
  - Anwendung einer Hash-Funktion auf Sharding-Attribut
  - Gleichmäßige Verteilung
  - Ineffiziente Bereichsabfragen
- Range-basiert
  - Dokumente mit ähnlichem Sharding-Attribut befinden sich in gleicher Partition
  - Effiziente Bereichsabfragen
  - Gefahr der Entstehung von Hotspots
  - Günstige Wahl des Sharding-Attributs entscheidend



Bildquelle: [MongoDB]

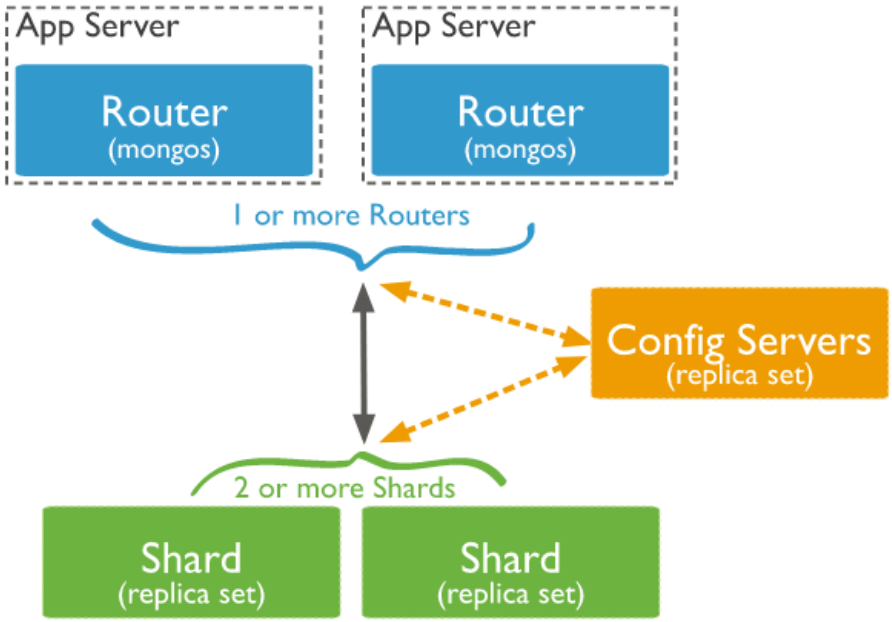
MongoDB: Sharding (2)

- Aufspaltung der Daten in **Chunks**
  - Default: 64 MB
  - Autom. Anpassung durch „Range-Split“



- **Shard** = Alle Chunks eines Servers (i.d.R. ein Replica Set, d.h. mehrere Server)

- ***Config-Server***:
  - Verwaltung der Metadaten (Speicherorte)
  - Lastbalancierung: gleich viele Chunks pro Shard
- Anfragen über ***Routing-Server*** (*mongos*)



Bildquelle: [MongoDB]

MongoDB: Setup Sharding (lokal)

```
mongod –-shardsvr --replSet rs1 --dbpath ./mongo1 --port 27011
```

```
mongod –-shardsvr --replSet rs1 --dbpath ./mongo2 --port 27012
```

```
mongod –-shardsvr --replSet rs1 --dbpath ./mongo3 --port 27013
```

```
mkdir ./mongoconfig1 ./mongoconfig2 ./mongoconfig3
```

```
mongod --configsvr --replSet configReplSet --dbpath ./mongoconfig1 --port 27019
```

```
mongod --configsvr --replSet configReplSet --dbpath ./mongoconfig2 --port 27020
```

```
mongod --configsvr --replSet configReplSet --dbpath ./mongoconfig3 --port 27021
```

```
mongo localhost:27019
rs.initiate( {
   _id: "configReplSet",
   configsvr: true,
   members: [
      { _id: 1, host: "localhost:27019" },
      { _id: 2, host: "localhost:27020" },
      { _id: 3, host: "localhost:27021" }
   ]
} )
```

```
mongos --configdb configReplSet/localhost:27019,localhost:27020,localhost:27021 --port 27022
```

```
mongo localhost:27022/admin
sh.addShard( "rs1/localhost:27011,localhost:27012,localhost:27013" )
```

MongoDB: Setup Sharding (lokal)

```
mkdir ./mongo4 ./mongo5 ./mongo6
```

```
mongod --shardsvr --replSet rs2 --dbpath ./mongo4 --port 27014
```

```
mongod –-shardsvr --replSet rs2 --dbpath ./mongo5 --port 27015
```

```
mongod –-shardsvr --replSet rs2 --dbpath ./mongo6 --port 27016
```

```
mongo localhost:27014
rs.initiate({
  _id: 'rs2',
  members: [
    {_id: 1, host: 'localhost:27014'},
    {_id: 2, host: 'localhost:27015'},
    {_id: 3, host: 'localhost:27016'}
  ]
})
```

```
mongo localhost:27022/admin
sh.addShard( "rs2/localhost:27014,localhost:27015,localhost:27016" )
sh.enableSharding( "test" )
use test
db.cities.createIndex( { name : 1 } )
sh.shardCollection( "test.cities", { name : 1 } )
db.stats()
db.printShardingStatus()
```