



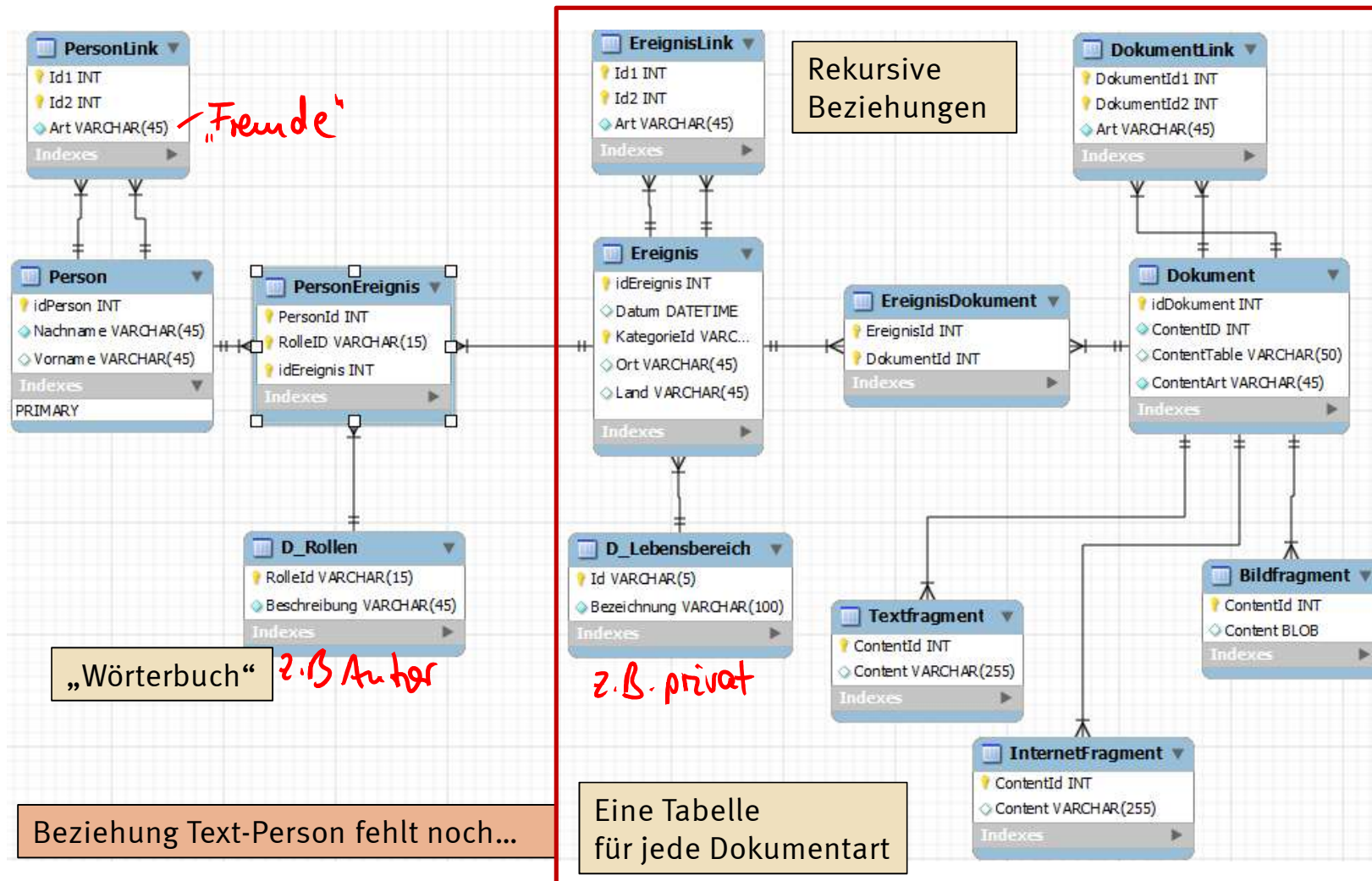
Moderne Datenbanken

Dokumentenorientierte Datenbanken

Agenda

1	Dokumentenorientierte Datenbanken	2
2	Abfragen	12
2.1	Query-by-Example	13
2.2	Map-Reduce-Algorithmus	18
2.3	Beispiel	33
2.4	Aggregation Pipeline	40
3	Diskussion	49

Relationales Modell – Kollaboratives Tagebuch



Relationale Datenbank

- Philosophie:
Zuerst DB-Schema erstellen,
dann Daten einfügen
- **Datenzentriert**
- Basis:
 - Relationales Modell
 - Normalisierung
 - Kein Standard zur
Datenbeschreibung
 - DB-Schema ist stabil

Dokumentenorientierte Datenbank

- Philosophie:
Daten sind verfügbar mit und ohne
Datenschema (z.B. DTD)
- **Dokumentenorientiert**
- Basis:
 - Syntax zur Datenbeschreibung
 - Keine Normalisierung
 - Standard zur Datenbeschreibung
vorhanden
 - Datenschema ist flexibel
- Beispiele
 - JSON-basiert:
 - CouchDB, MongoDB, Riak
 - XML-basiert:
 - eXist-DB, Oracle Berkley DB

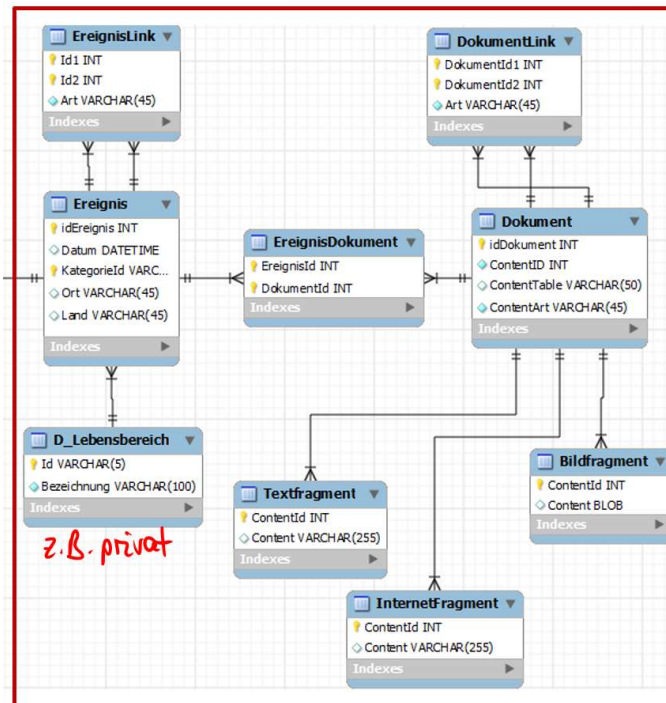
Dokumentenformate XML und JSON

XML	JSON
Document Type Definition (DTD)	JavaScript Notation
Daten sind nicht typisiert	Daten sind typisiert
xs:date, xs:time	Kein Datetime-Datentyp
Auszeichnungssprache → größerer Anwendungsbereich	valides JavaScript
Daten verschachtelt	einfacher lesbar

```
<Kunde
  Name="Meitner"
  Vorname="Lise"
  weiblich="true"
  Alter="42">
<Hobby>Physik</Hobby>
<Hobby>Lesen</Hobby>
</Kunde>
```

```
{
  „Name": "Meitner",
  "Vorname": "Lise",
  "weiblich": true,
  "Hobbys": [ "Physik", "Lesen" ],
  "Alter": 42
}
```

Das Datenmodell besteht aus strukturierten Datensammlungen (z.B. XML-, JSON- (JavaScript Object Notation), oder RDF (Ressource Description Framework)-Dokumenten). Einzelne Dokumente werden zusammen mit einer ID abgelegt.

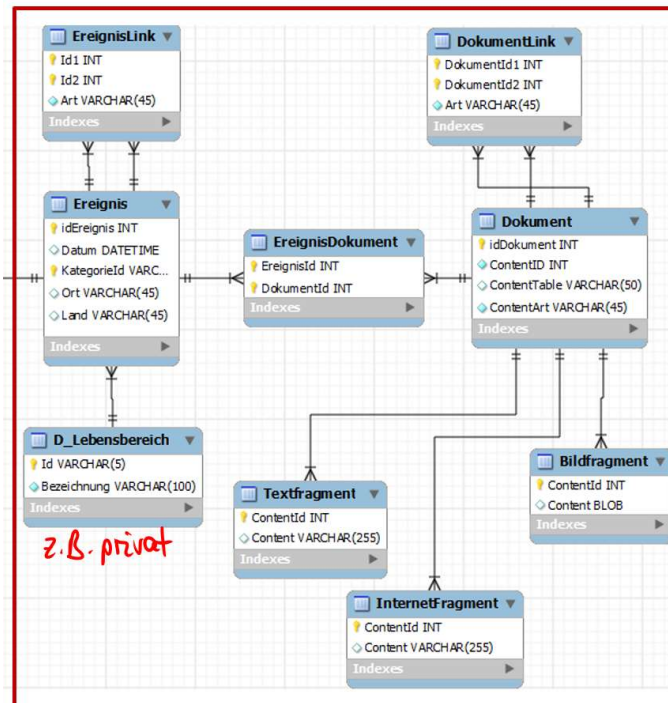


XML

```

<Ereignis id="Ereignis1">
  <Titel>Moderne Datenbanken</Titel>
  <Ort> EF42</Ort>
  <Datum>2018.05.05 </Datum>
  <Dokumente id="1">
    <Dokument id="1" typ="text" autor="1">
      <Beschreibung>Vorlesung MDB </Beschreibung>
    </Dokument>
    <Dokument id="2" typ="link" autor="1">
      <a href="www.fh-dortmund.de"> FHDo </a>
    </Dokument>
  </Dokumente>
</Ereignis>
    
```


Das Datenaustauschformat **JSON** bietet eine einfach lesbare Textform in JavaScript-Notation. Jedes gültige JSON-Dokument soll ein gültiges JavaScript Fragment sein.



JSON

```
{
  "_id": "Ereignis1",
  "Titel": "Moderne Datenbanken",
  "Datum": "2018-05-05",
  "Ort": "Dortmund, EF 42",
  "Dokumente": "[
    {
      "Typ": "text",
      "Autor": "Autor1",
      "Beschreibung": "Vorlesung MDB",
      "Typ": "link",
      "Autor": "Autor1",
      "href": "www.fh-dortmund.de",
      "text": "FHDo"
    }
  ]"
}
```

Verschachtelung

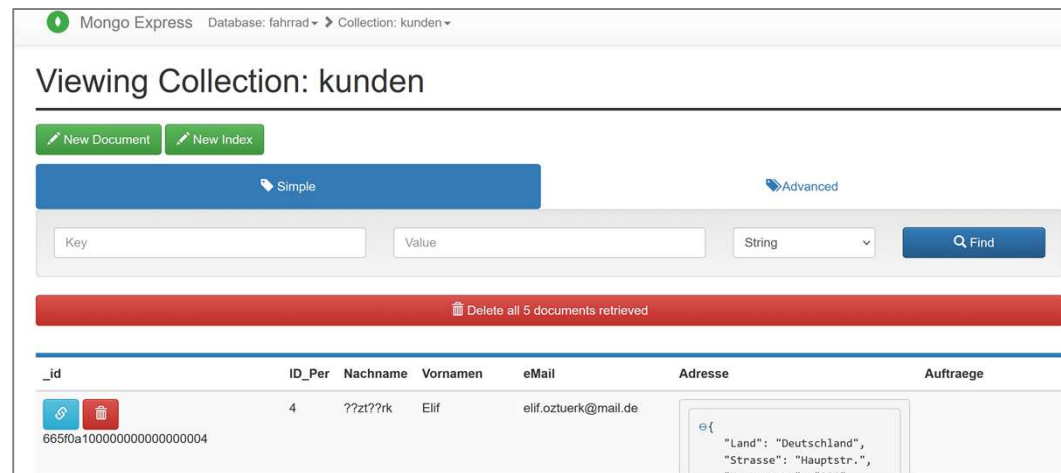
Beispiel: Dokumentenorientierte Datenbank MongoDB

MongoDB bietet eine REST-Schnittstelle und Verwaltungsoberflächen an.
Beim Einfügen und Ändern werden die Dokumente validiert.

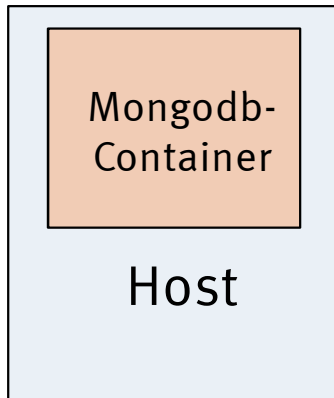
Mongodb Compass
(lokal installiert)



GUI MongoExpress
(mit Docker)



Verwendung von mongoDB im Docker-Container



- Öffnen der mongodb-Shell im Container:
admin ist die Datenbank, in der die Credentials des Root-Users gespeichert sind
`docker exec -it mongodb mongosh -u admin --authenticationDatabase admin`
- Listen der Datenbanken und Collections
`show dbs`
`show collections`
- Explizites anlegen von Datenbanken und Collections (implizit durch Create)
`use shop`
`db.createCollection("kunden")`
- Anlegen von Nutzer-Credetials
Anlegen eines Nutzers in der Datenbank Fahrrad
`use shop`
`db.createUser({`
 `user: "fahrrad_user",`
 `pwd: "geheim",`
 `roles: [{ role: "readWrite", db: "fahrrad" }]`
`})`

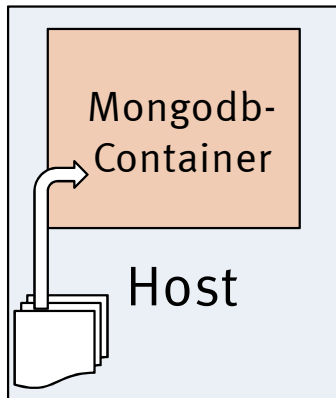
Import von Daten in die Datenbank

Prof. Dr. I. M. Saatz

Moderne Datenbanken

Fachbereich Informatik

11



Import über die Konsole

```
docker exec -i mongodb mongoimport \  
--uri "mongodb://login:pw@localhost:27017" \  
--authenticationDatabase admin \  
--db shop\  
--collection kunden \  
--jsonArray \  
--drop \  
--file - < kunden.json
```

```
# Aufruf mongoimport im Container  
# URI  
# Datenbank mit den User-Credentials  
# Ausgewählte Datenbank  
# Ausgewählte Collection  
# Datei enthält die Daten als JSON-Array  
# Löschen der vorhandenen Collection  
# Einlesen der Dateiinhalte in stdin
```

Alternative Powershell (Windows)

- *# Powershell Probleme: \ ersetzen durch ` oder alles in eine Zeile schreiben, < kunden.json durch Get-Content -Raw ersetzen*

```
Get-Content -Raw .\kunden.json |
```

```
docker exec -i mongodb mongoimport --username admin --password mongodb --  
authenticationDatabase admin --db fahrrad --collection kunden --jsonArray --drop
```

Agenda

1	Dokumentenorientierte Datenbanken	2
2	Abfragen	12
2.1	Query-by-Example	13
2.2	Map-Reduce-Algorithmus	18
2.3	Beispiel	33
2.4	Aggregation Pipeline	40
3	Diskussion	49

Agenda

1	Dokumentenorientierte Datenbanken	2
2	Abfragen	12
2.1	Query-by-Example	13
2.2	Map-Reduce-Algorithmus	18
2.3	Beispiel	33
2.4	Aggregation Pipeline	40
3	Diskussion	49

Beispiel: Mongo Express

The screenshot shows the Mongo Express web interface. At the top, it says 'Mongo Express' with a green logo, followed by 'Database: vorlesung' and 'Collection: test'. Below this, there are two tabs: 'Simple' and 'Advanced'. The 'Simple' tab is active, showing a query editor with the following JSON query:

```
{
  $or: [{ Nachname: 'Saatz'}, {Nachname:
    'Mustermann'}],
  Vornamen: { $all: ["Inga"] }
}
```

To the right of the query editor, there is a checkbox labeled 'Aggregate query' which is checked. Below the checkbox is a blue button labeled 'Find'. Below the query editor, there is a red bar with a trash icon and the text 'Delete all 1 documents retrieved'. Below this, there is a table with the following columns: '_id', 'Nachname', and 'Vornamen'. The table contains one row with the following data:

_id	Nachname	Vornamen
66350e5ba717b4c8d5102317	Saatz	Inga, Marina

Äquivalente SQL-Anfrage:

SELECT Nachname, Vornamen FROM test

WHERE „Inga“ IN Vornamen AND

Nachname=„Saatz“ OR Nachname=„Mustermann“

- Query-by-Example
 - Idee: Anhand eines Musterobjektes werden alle hierzu „passenden“ persistenten Objekte zurückgeliefert
 - Erstellung des **Musterobjektes**
 - Belegung des Musterobjektes mit den gewünschten Attributwerten
 - Felder mit Defaultwerten werden bei der Anfrage ignoriert

```
Person muster = new Person();  
muster.setName("Meitner");  
ArrayList<Person> result = index.queryByExample(muster);
```

- Bereichsabfragen sind möglich über den Methodenaufruf
index.queryByExample(musterVon, musterBis)
- Nachteile:
 - Nur einfache Abfragen (ohne logische Verknüpfungen, ...)
 - Abfrage von NULL-Werten ist nicht möglich

Auswahl von Operatoren zur Verwendung in QbE-Anfragen. Anfragen mit Gruppierungen und Aggregationen können mit QbE nicht formuliert werden.

Befehl	Beschreibung
\$regex	Matching eines PCRE-konformen Regulären Ausdrucks (oder verwenden Sie einfach die //-Trennzeichen, wie vorhin gezeigt)
\$ne	Nicht gleich
\$lt	Kleiner als
\$lte	Kleiner oder gleich
\$gt	Größer als
\$gte	Größer oder gleich
\$exists	Prüft, ob ein Feld existiert
\$all	Matching aller Elemente im Array
\$in	Matching beliebiger Elemente im Array
\$nin	Kein Matching beliebiger Element im Array
\$elemMatch	Matching aller Felder in einem Array verschachtelter Dokumente
\$or	ODER
\$nor	NICHT ODER
\$size	Matching eines Arrays gegebener Größe
\$mod	Modulo
\$type	Matching eines Feldes auf angegebenen Datentyp
\$not	Negation des angegebenen Operators

Befehl	Beschreibung
\$set	Setzt das angegebene Feld auf den angegebenen Wert
\$unset	Entfernt das Feld
\$inc	Addiert den angegebenen Wert zum angegebenen Feld hinzu
\$pop	Entfernt das letzte (oder erste) Element aus einem Array
\$push	Fügt einen Wert in ein Array ein
\$pushAll	Fügt alle Werte in ein Array ein
\$addToSet	Wie push, aber ohne doppelte Werte
\$pull	Entfernt passende Werte aus einem Array
\$pullAll	Entfernt alle passenden Wert aus einem Array

Quelle: [RW2012, S. 159, 161]

In der MongoDB-Shell können mit der find-Methode QbE-Anfragen formuliert werden.
Als Parameter können JSON-Dokumente verwendet werden.
Eine Collection ist eine Sammlung von Dokumenten.

```
find(<SEL>, <PRO>, <weitereOptionen>).sort(<Attribute> ).limit(<Anzahl>).skip(<Anzahl>)
```

MongoDB:

```
db.vorlesung.find({Titel:"Moderne Datenbanken"},  
                 {Titel:1, Ort:1, Datum:1, id=0}  
                 ).sort({Titel:1}).limit(5).skip(0)
```

Collection ↑

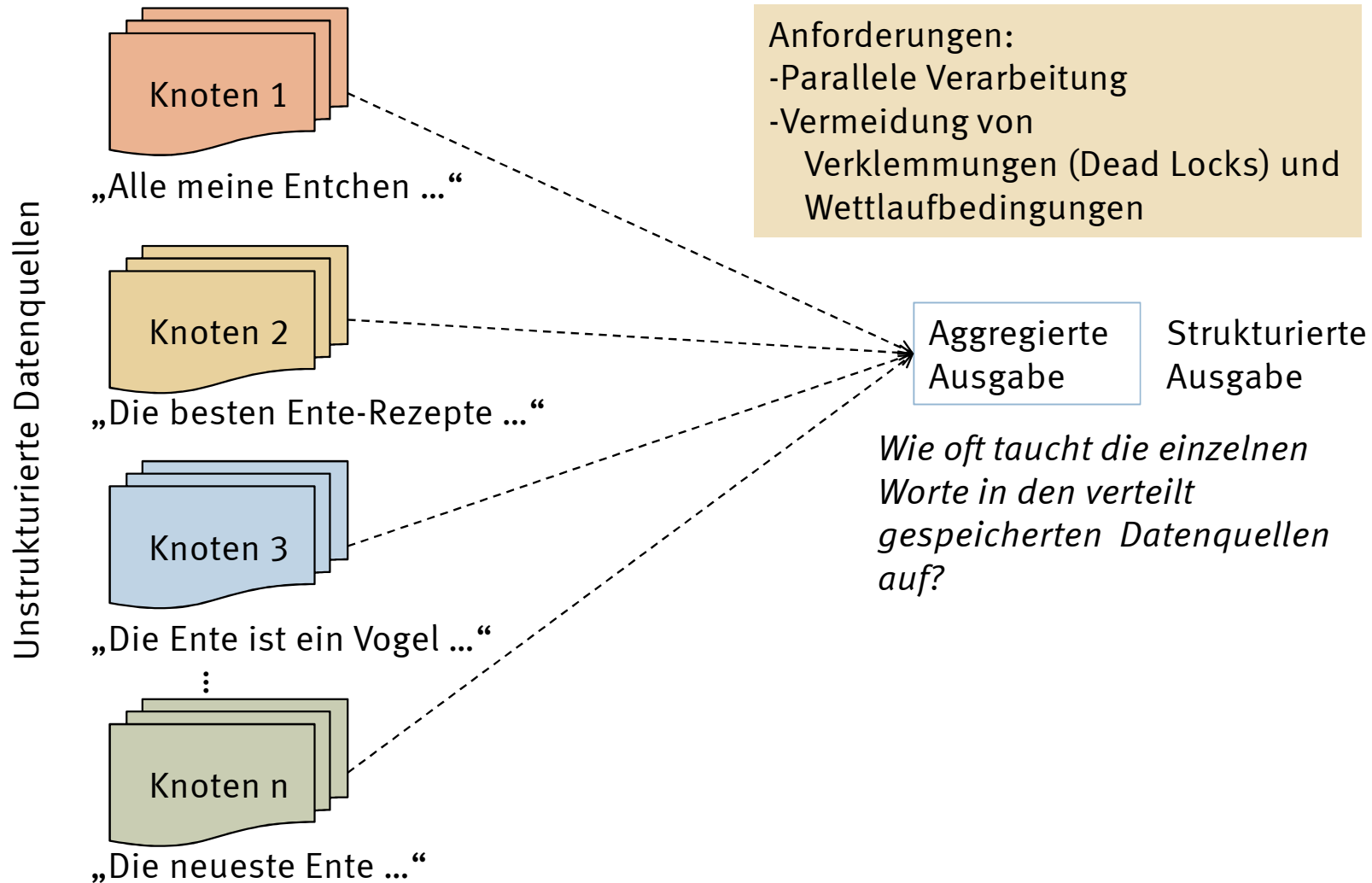
ASC=1
DESC=-1 ↑

Anzeigen=1
Ausblenden=0 ↓

Agenda

1	Dokumentenorientierte Datenbanken	2
2	Abfragen	12
2.1	Query-by-Example	13
2.2	Map-Reduce-Algorithmus	18
2.3	Beispiel	33
2.4	Aggregation Pipeline	40
3	Diskussion	49

Abfrage schemafreier, skalierbarer Datenbanken

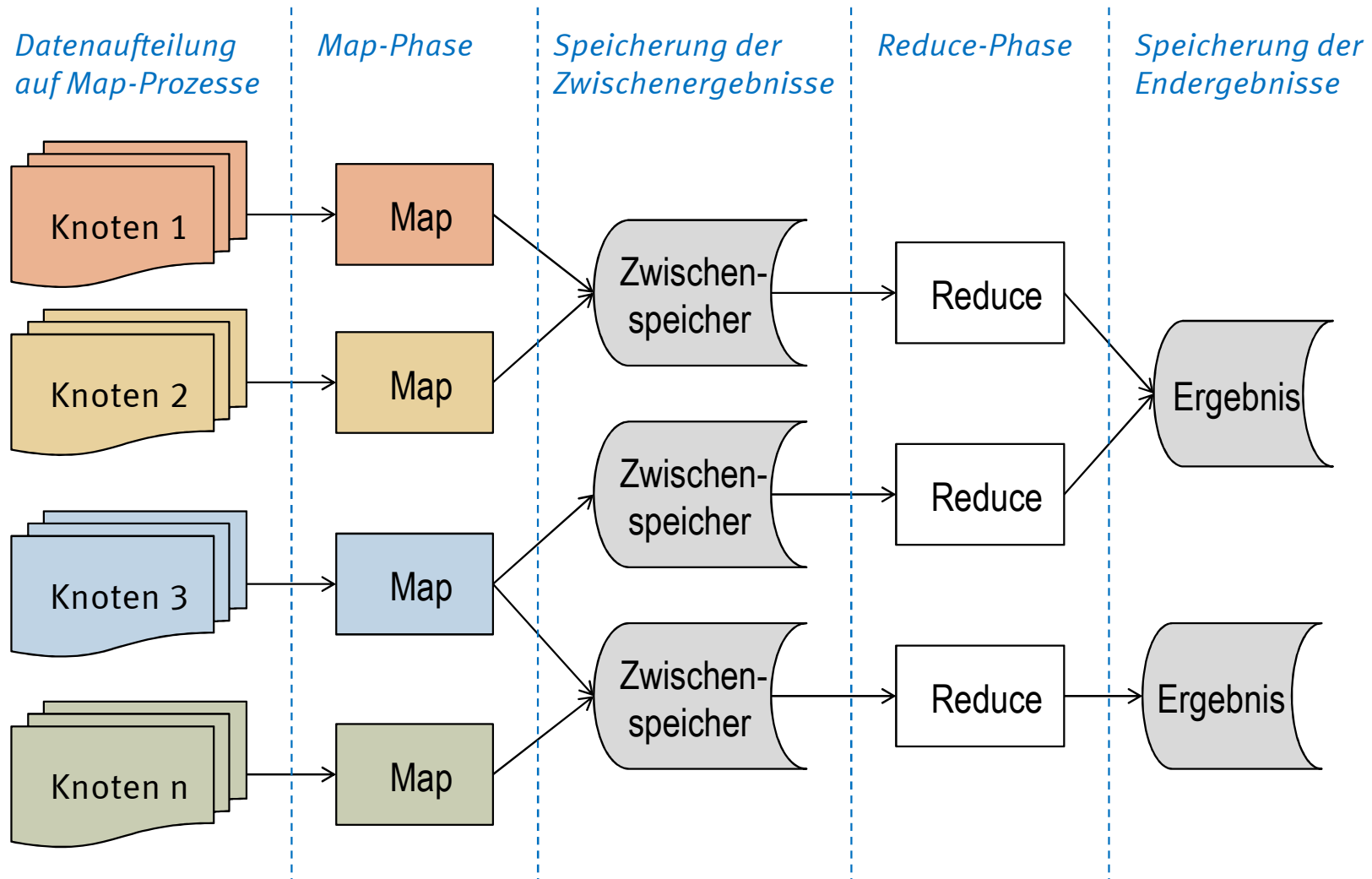


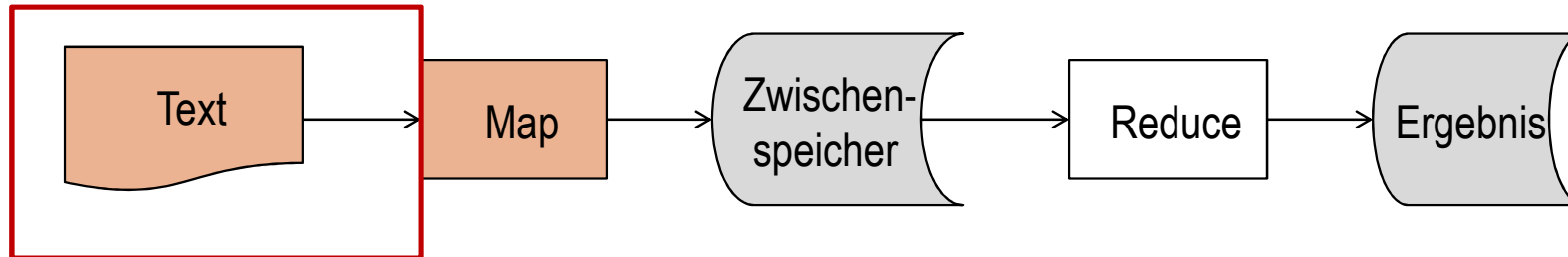
- Anwendung:
 - Effizientes paralleles Durchsuchen von großen verteilten Datenmengen
- Verwendung
 - Datenbankabfrage (bspw. CouchDB, MongoDB, Riak, Hbase)
- Historie
 - Entwickelt durch Jeffrey Dean und Sanjay Ghemawat bei der Google Inc.



- Veröffentlichung in 2004 <http://labs.google.com/papers/mapreduce.html>
- 2010 Patentiert durch Google Inc.

Ablauf des Algorithmus





Schritt 1: Unstrukturierte Texte parallel einlesen

„Alle meine Entchen schwimmen auf dem See, schwimmen auf dem See, ...“

Schritt 2: Zuordnung von Dateiinhalt zu Positionen

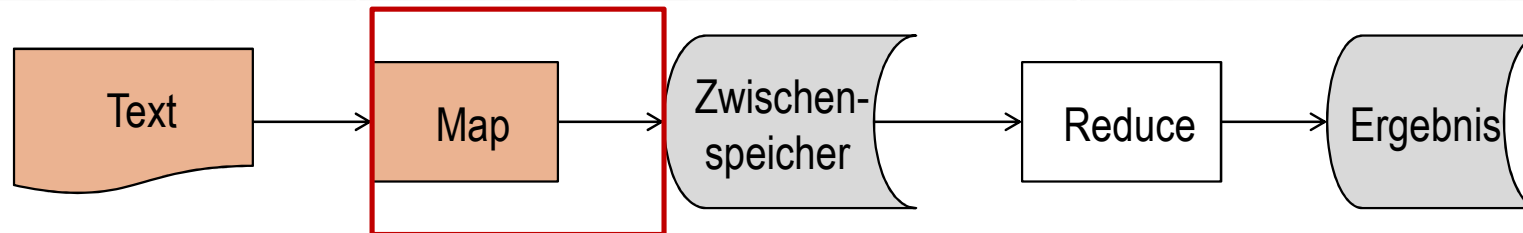
- a) Normalisierung des Textes
- b) Aufteilung des Textes in gleichlange Sätze
- c) Erzeugung von Schlüssel-Wert-Paaren

Eingabeliste = [(satz1, „alle meine entchen schwimmen auf dem see“),
(satz2, „schwimmen auf dem see köpfchen in das wasser“),
(satz3, „schwänzchen in die höh“)]

Key Value

Schritt 3: Starten eines Mapprozesses für jedes Schlüssel-Wert-Paar der Eingabeliste

P1 = Map(satz1, „alle meine entchen schwimmen auf dem see “)



Schritt 1: Extrahierung der relevanten Daten aus der Eingabe

Wort	Anzahl	Wort	Anzahl
alle	1	auf	1
meine	1	dem	1
entchen	1	see	1
schwimmen	1		

Schritt 2: Lokale Speicherung von Zwischenergebnispaaren (pro Map-Prozess)

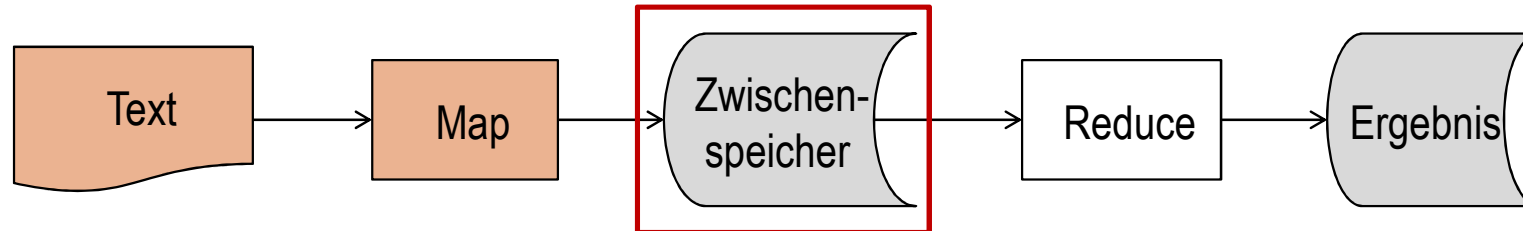
Map-Funktion

P1 = Map(satz1, „alle meine entchen schwimmen auf dem see“)

= [(„alle“, 1), („meine“, 1), („entchen“, 1), („schwimmen“, 1), („auf“, 1), („dem“, 1), („see“, 1)]

Die Map-Funktion ist in JavaScript geschrieben. Alle map-Funktionen haben den Parameter doc, der das jeweilige Dokument repräsentiert. Durch die emit-Funktion werden die Schlüssel-Wert-Paare in die View-Ausgabe geschrieben. Die emit-Funktion kann mehrfach für ein Dokument aufgerufen werden.

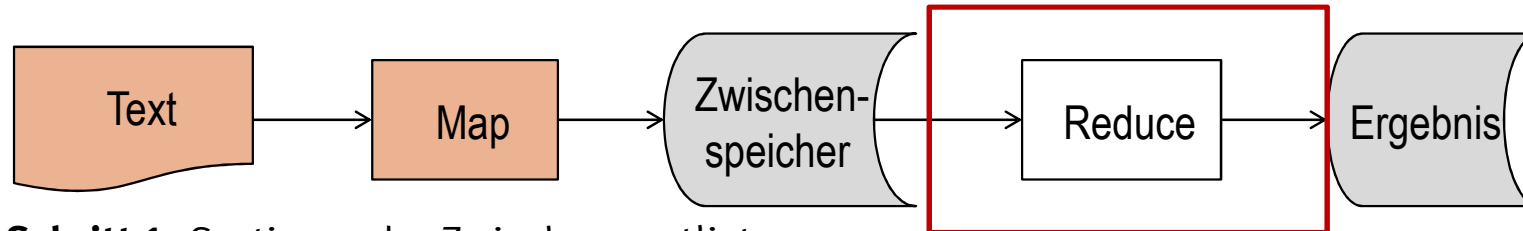
```
var mapFunction = function () {  
    if (typeof this.text === "string") {  
        var match = this.text.match(/Enten|Entchen/g);  
        if (match) {  
            var felder = this.text.split(/\s+/);  
            felder.forEach(function (item) { emit(item, 1); });  
        }  
    }  
}
```



Schritt: Sammlung der Zwischenergebnispaare der einzelnen Map-Prozesse

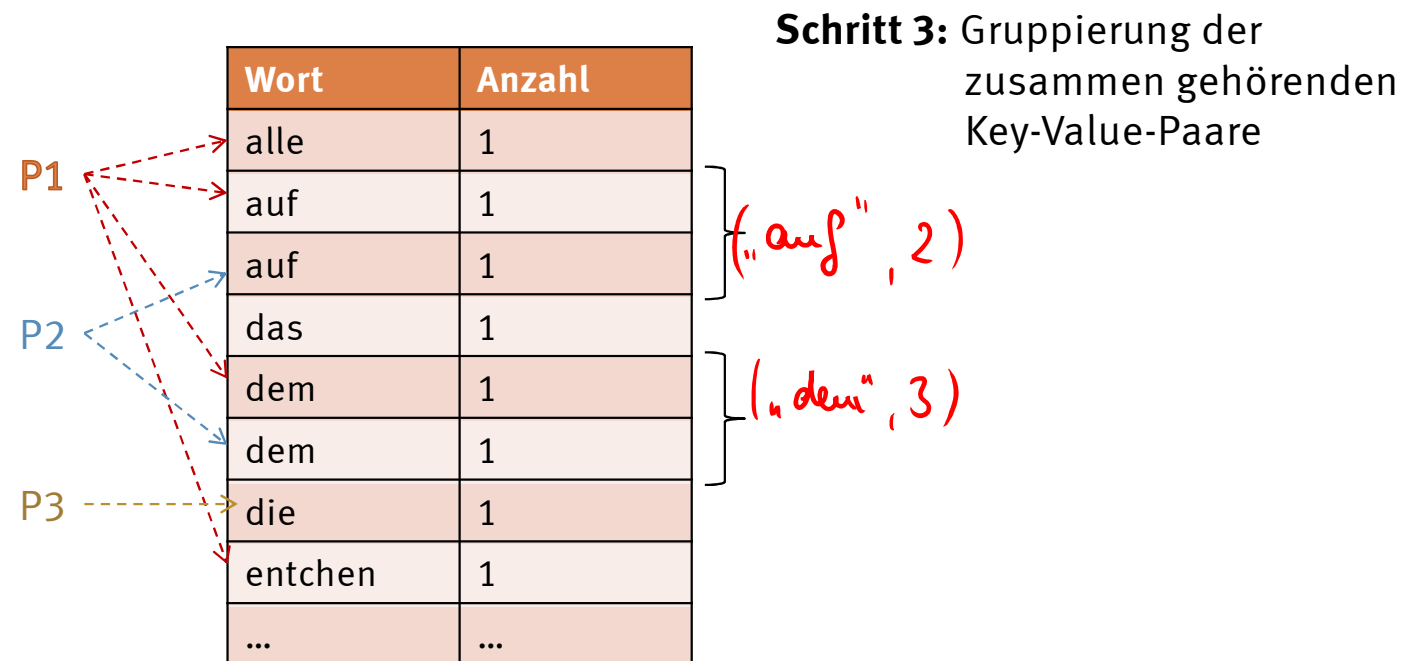
	Wort	Anzahl	Wort	Anzahl
P1	alle	1	auf	1
	meine	1	dem	1
	entchen	1	see	1
	schwimmen	1		
P2	schwimmen	1	köpfchen	1
	auf	1	in	1
	dem	1	das	1
	see	1	wasser	1
P3	schwänzchen	1	die	1
	in	1	höh'	1

Reduce



Schritt 1: Sortieren der Zwischenwertlisten

Schritt 2: Zusammenführen der (verteilten) Zwischenwertlisten



Sortierte Ergebnisse der Map-Phase

Anklicken des generierten Views in der CouchDB-Weboberfläche Futon liefert das Ergebnis der Map-Phase.



http://localhost:5984/vorlesung/_design/testView/_view/new-view?reduce=false

Table Metadata JSON		
id	key	value
373847c32b6cfe3d05efce92a6004822	Alle	1
373847c32b6cfe3d05efce92a60018c4	auf	1
373847c32b6cfe3d05efce92a6004822	auf	1
373847c32b6cfe3d05efce92a602303b	deluxe	1
373847c32b6cfe3d05efce92a60018c4	dem	1
373847c32b6cfe3d05efce92a6004822	dem	
373847c32b6cfe3d05efce92a6004822	Entchen	
373847c32b6cfe3d05efce92a60018c4	Enten	
373847c32b6cfe3d05efce92a602303b	Entenbraten	
373847c32b6cfe3d05efce92a60018c4	Land	
373847c32b6cfe3d05efce92a6004822	meine	
373847c32b6cfe3d05efce92a6004822	schwimmen	
373847c32b6cfe3d05efce92a6004822	See	

JSON-Ansicht

```
id "373847c32b6cfe3d05efce92a6004822"
{
  "id": "373847c32b6cfe3d05efce92a6004822",
  "key": "Alle",
  "value": 1,
  "doc": {
    "_id": "373847c32b6cfe3d05efce92a6004822",
    "_rev": "4-33ce00426886b6bafbbec005ef63516a",
    "text": "Alle meine Entchen schwimmen auf dem See"
  }
}
```

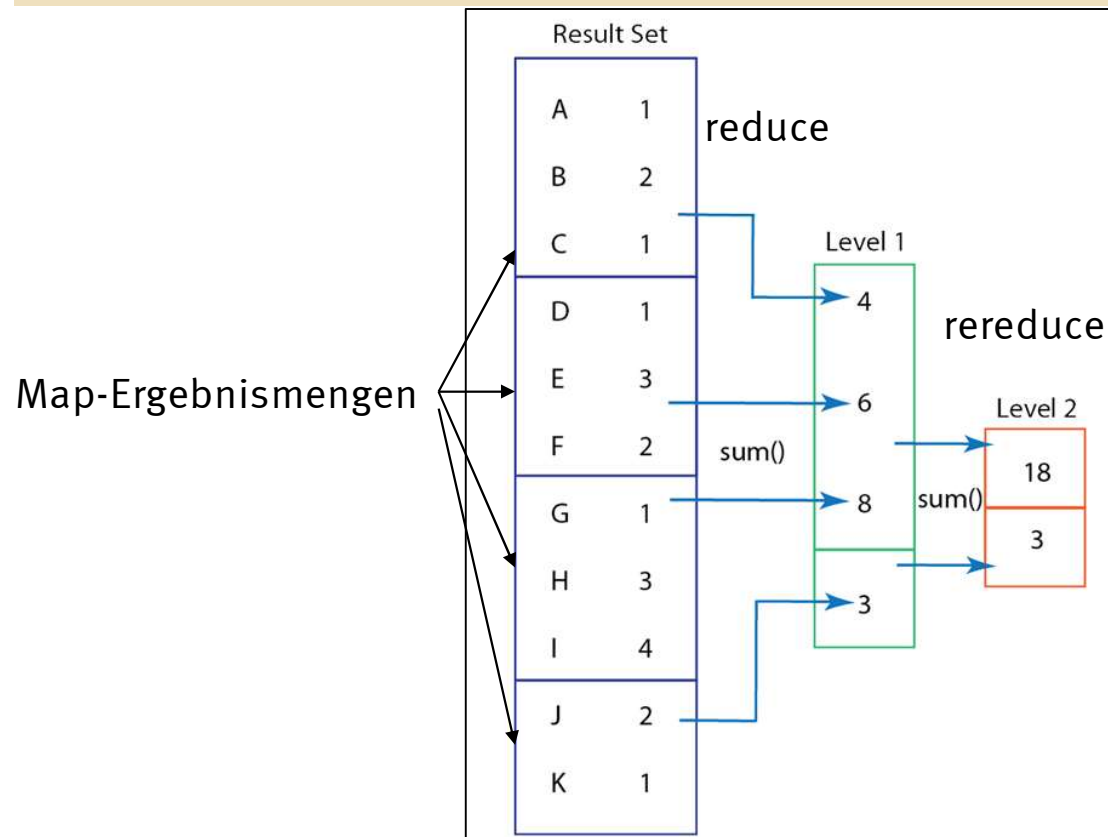
Auswahl einer vordefinierten Reduce-Funktion oder Implementierung einer Reduce-Funktion in JavaScript.

```
var reduceFunction = function(key, values) {  
  return Array.sum(values)  
}
```

```
// Map-Reduce-Algorithmus ausführen und  
// Ergebnis in Collection schreiben  
db.enten.mapReduce(  
  mapFunction,  
  reduceFunction,  
  { out: "mr_output" }  
)  
  
// Ausgabe  
db.mr_output.find()
```

Rerreduce: Mehrfaches anwenden des Reduce-Schritts

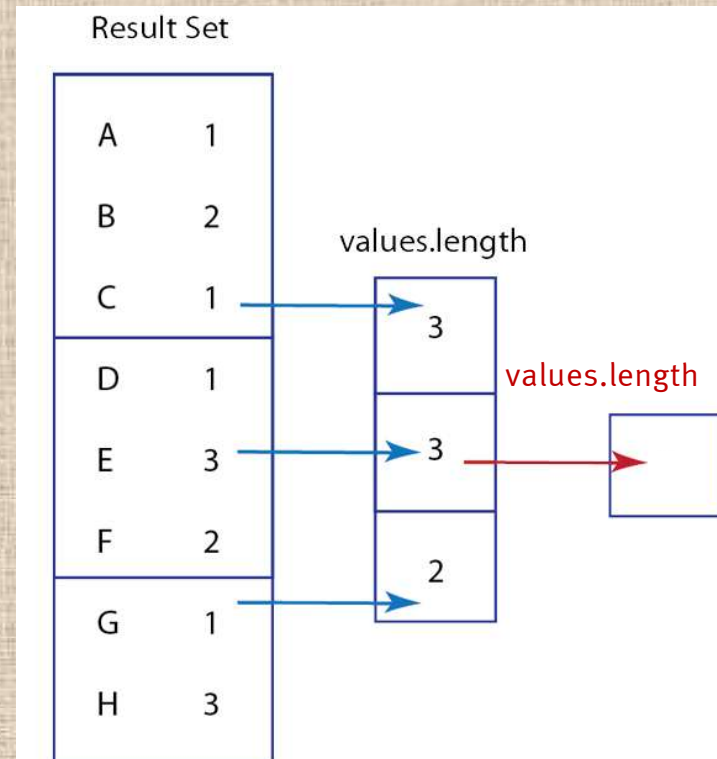
Die mehrfache Anwendung der Reduce-Funktion wird als Re-Reduce bezeichnet. Es handelt sich um eine Aggregation, die beispielsweise beim Zusammenführen von Ergebnissen von verschiedenen DB-Knoten auftritt.

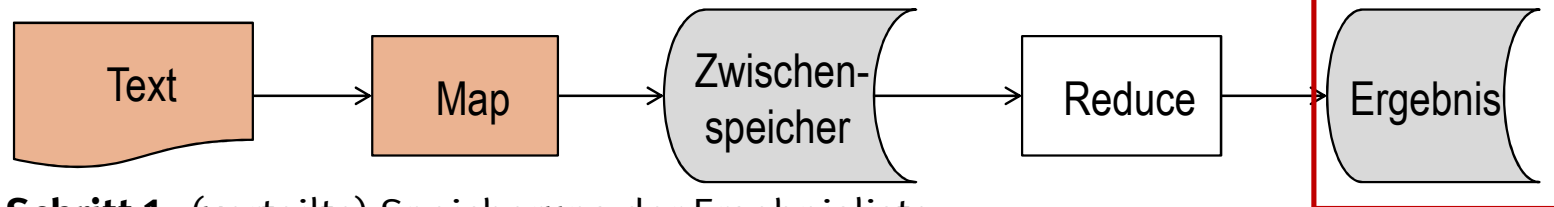


Vorsicht Falle!

Was geht hier schief?

Ermittle die Anzahl der Keys





Schritt 1: (verteilte) Speicherung der Ergebnisliste

Wort	Anzahl	Wort	Anzahl
alle	1	auf	2
meine	1	dem	2
entchen	1	see	2
schwimmen	2	köpfchen	1
in	2	wasser	1
das	1	schwänzchen	1
die	1	die	1

Schritt 2: Generierung der Ergebnisliste

Reduce-Funktion
↙
Ausgabeliste = Reduce(P1, P2, P3)
= {(„alle“, 1), („meine“, 1), („entchen“, 1), („schwimmen“, 2),
(„auf“, 2), („dem“, 2), („see“, 2), ...}

Agenda

1	Dokumentenorientierte Datenbanken	2
2	Abfragen	12
2.1	Query-by-Example	13
2.2	Map-Reduce-Algorithmus	18
2.3	Beispiel	33
2.4	Aggregation Pipeline	40
3	Diskussion	49

Beispiel: Nutzerbewertungen

Ereignisse im kooperativen Tagebuch werden durch andere Nutzer bewertet:

Ereignis	Gut	Mittel	Schlecht	weißNicht
Ereignis 1	3	1	3	3
Ereignis 2	2	2	0	2

Wie wäre das JSON-Dokument zu ergänzen, so dass die Bewertungen erfasst werden können?

```
{
  "_id": "Ereignis1",
  "Titel": "Moderne Datenbanken",
  "Datum": "2018-05-05",
  "Ort": "Dortmund, EF 42",
  "Dokumente": [
    {"Typ": "text",
     "Autor": "Autor1",
     "Beschreibung": "Vorlesung MDB"},
    {"Typ": "link",
     "Autor": "Autor1",
     "href": "www.fh-dortmund.de",
     "text": "FHDo"}]
}
```

Beispiel: Nutzerbewertungen

Ereignisse im kooperativen Tagebuch werden durch andere Nutzer bewertet:

Ereignis	Gut	Mittel	Schlecht	weißNicht
Ereignis 1	3	2	1	0
Ereignis 2	0	1	2	3

Wie wäre das JSON-Dokument zu ergänzen, so dass die Bewertungen erfasst werden können?

Bewertung: {"gut": 3, "mittel": 2, "schlecht":1, "weißnicht":0}
oder
Bewertung: [3, 2,1,0]

```
"Ort": "Dortmund, EF 42",  
"Dokumente": [  
  {  
    "Typ": "text",  
    "Bewertung": [  
      3,  
      2,  
      1,  
      0  
    ],  
    "Autor": "Autor1",
```


Beispiel: Nutzerbewertungen

Ereignisse im kooperativen Tagebuch werden durch andere Nutzer bewertet:

Ereignis	Gut	Mittel	Schlecht	weißNicht
Ereignis 1	3	2	1	0
Ereignis 2	0	1	2	3

Es soll der gewichtete Mittelwert der Bewertungen zu den Ereignissen berechnet werden.

```
{"_id": "Ereignis1",  
  "Bewertung": [3, 2, 1, 0]  
}
```

Map-Funktion

Reduce-Funktion

Beispiel: Nutzerbewertungen

Ereignisse im kooperativen Tagebuch werden durch andere Nutzer bewertet:

Ereignis	Gut: 3	Mittel: 2	Schlecht: 1	weißNicht: 0
Ereignis 1	3	2	1	0
Ereignis 2	0	1	2	3

Gewicht



Map function ?

```
1 function (doc) {
2   if ("Dokumente" in doc ){
3     var docs = doc.Dokumente;
4     docs.forEach(
5       function (item){
6         if ("Bewertung" in item){
7           var bewertungen = item.Bewertung;
8           var avg = bewertungen[0]*3+bewertungen[1]*2+bewertungen[2]*1;
9           var anz= bewertungen[0]+bewertungen[1]+bewertungen[2];
10          if (anz>0)
11            emit(doc._id, avg/anz);
12          else
13            emit(doc._id, 0);
14        }
15      })
16    }
17 }
```

Reduce-Funktion

Custom Reduce function

```
1 function (keys, values, rereduce) {  
2   if (rereduce)  
3     return {  
4       'anz': values.reduce(function (a,b) {return a+b.anz},0),  
5       'sum': values.reduce(function (a,b) {return a+b.sum},0),  
6       'avg': values.reduce(function (a,b) {return a+b.sum},0)/values.reduce(function (a,b) {return a+b.anz},0)  
7     }  
8   else  
9     return {  
10      'anz': values.length,  
11      'sum': sum(values),  
12      'avg': sum(values)/values.length  
13    }  
14 }
```

rows: group_level=0

0:

key: null

value:

anz: 2

sum: 3.666666666666667

avg: 1.8333333333333335

rows: group_level=1

0:

key: "373847c32b6cfe3d05efce92a600d85e"

value:

anz: 1

sum: 2.3333333333333335

avg: 2.3333333333333335

1:

key: "373847c32b6cfe3d05efce92a600f6d3"

value:

anz: 1

sum: 1.3333333333333333

avg: 1.3333333333333333



- Suche von Mustern über verteilte Daten
 - Map-Funktion: Liefert Zeilen, die dem Suchmuster entsprechen
 - Reduce-Funktion: Sammeln der Ergebnisse

- Zählen von Zugriffen auf eine URL
 - Map-Funktion: Logfiles auswerten
Ergebnis: (URL, Anzahl der Zugriffe)
 - Reduce-Funktion: Addition der Zwischenergebnisse pro URL

- Graph über verlinkte Webseiten zu einem Ziel erstellen
 - Map-Funktion: Analyse der Quell-URL
Ergebnis: (Ziel-URL, Quell-URL)
 - Reduce-Funktion: Zusammenführen der Listen aller Quell-URL zu einer Ziel-URL
Ergebnis: (Ziel-URL, liste(Quell-URL))

- Sortieren verteilter Daten
- Aufbereitung von Daten für häufige Aufrufe
- Wortindex-Erstellung

Agenda

1	Dokumentenorientierte Datenbanken	2
2	Abfragen	12
2.1	Query-by-Example	13
2.2	Map-Reduce-Algorithmus	18
2.3	Beispiel	33
2.4	Aggregation Pipeline	40
3	Diskussion	49

Aggregation Pipeline - Beispiel

Zähle alle Dokumente des Autors 1.
Was ist hierfür zu tun?

```
▼ _id: Object
  autor: "Autor2"
▼ dokumente: Array
  ▼ 0: Object
    titel: "Privat"
    typ: "text"
anzahl: 1
```

```
▼ _id: Object
  autor: "Autor1"
▼ dokumente: Array
  ▼ 0: Object
    titel: "Moderne Datenbanken"
    typ: "text"
  ▼ 1: Object
    titel: "Moderne Datenbanken"
    typ: "text"
```

Aggregation Pipeline - Beispiel

Prof. Dr. I. M. Saatz

Moderne Datenbanken

Fachbereich Informatik

42



Zähle alle Dokumente des Autors 1.

```
db.vorlesung.aggregate([
  {$match: {id.autor: „Autor1“}},
  {$count: {Documente}}])
```

\$match

```
1 /**
2  * query - The query in MQL.
3  */
4 {
5   "_id.autor": "Autor1"
6 }
```

Output after \$match stage (Sample of 1 document)

```
{
  "_id": {
    "autor": "Autor1"
  },
  "dokumente": [
    {
      "id": "1",
      "autor": "Autor1",
      "titel": "The query in MQL."
    },
    {
      "id": "2",
      "autor": "Autor1",
      "titel": "The query in MQL."
    },
    {
      "id": "3",
      "autor": "Autor1",
      "titel": "The query in MQL."
    }
  ],
  "anzahl": 3
}
```

\$count

```
1 /**
2  * Provide the field name for the count.
3  */
4 'dokumente'
```

Output after \$count stage (Sample of 1 document)

```
{
  "dokumente": 1
}
```

X 

Aggregation Pipeline - Beispiel

Prof. Dr. I. M. Saatz

Moderne Datenbanken

Fachbereich Informatik

43

Mit der Funktion unwind werden Arrays „ausgepackt“.



```
db.vorlesung.aggregate([
  {$match: {id.autor: „Autor1“}},
  {$unwind: „$dokumente“},
  {$count: {Documente}}])
```

anzahldokumente SAVE PIPELINE ... COMMENT MODE SAMPLE MODE AUTO PREVIEW

```
1 /**
2  * path - Path to the array field.
3  * includeArrayIndex - Optional name for index.
4  * preserveNullAndEmptyArrays - Optional
5  *   toggle to unwind null and empty values.
6  */
7 {
8   path: '$dokumente'
9 }
```

▼ _id: Object
 autor: "Autor1"
▼ dokumente: Object
 titel: "Moderne Datenbanken"
 typ: "text"
 anzahl: 3

▼ _id: Object
 autor: "Autor1"
▼ dokumente: Object
 titel: "Moderne Datenbanken"
 typ: "link"
 anzahl: 3

dokumente: 3

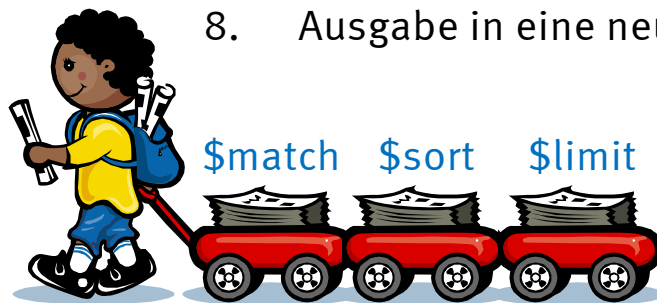


Bei einer Aggregation Pipeline wird eine Anfrage durch Angabe der nacheinander (z.T. parallel) ausgeführten Berechnungsschritte formuliert.

Anm.: Eine Aggregation Pipeline ist auch die Basis von LINQ-Abfragen unter .NET

Verarbeitung der Anfrage (wie in einer Programmiersprache üblich):

1. Filtern/Selektion (\$match)
2. Sortieren (\$sort)
3. Pagination (\$limit und \$skip)
4. Projektion und Umbenennen (\$project)
5. Gruppierung und Aggregation (\$group)
6. Arrays aufteilen (\$unwind)
7. Verbundoperationen (\$lookup)
8. Ausgabe in eine neue Kollektion schreiben (\$out)



Kollektion

```
db.vorlesung.aggregate(  
  {$match: {Titel: "Moderne Datenbanken"}},  
  {$sort: {Titel:1}},  
  {$limit: 2},  
)
```

Aggregation Pipeline: Invertierung

Invertierung der Dokumentenstruktur:

```
db.test.aggregate([
```

1. Schritt – Dokumente-Array in einzelne Tupel auflösen

```
Privat, 2018-05-05, at home, Autor2, Text
Privat, 2018-05-05, at home, Autor1, Link
```

```
{ $unwind: "$Dokumente",
```

2. Schritt – Gruppierung nach dem Autor

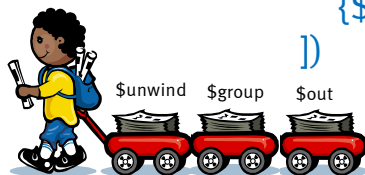
```
{ $group: {
  _id: { autor: "$Dokumente.Autor" },

  dokumente: { $push:
    { titel: "$Titel",
      typ: "$Dokumente.Typ" } }
},
```

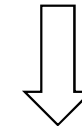
3. Schritt – Einfügen des Abfrageergebnisses in die DB

```
{ $out: "Autor-Dokumente" }
```

```
)
```



```
_id: ObjectId("5adb410728cb8...")
Titel: "Privat"
Datum: "2018-05-05"
Ort: "at home"
Dokumente: Array
  0: Object
    Typ: "text"
    Bewertung: Array
    Autor: "Autor2"
```



```
_id: Object
  autor: "Autor2"
dokumente: Array
  0: Object
    titel: "Privat"
    typ: "text"
```


Aggregation Pipeline: Verbundoperationen

Vorsicht: Es gibt keine referentielle Integrität, d.h. Referenzen werden nicht überprüft!
Nicht jede dokumentenorientierte Datenbank unterstützt das Verbundoperationen.



1. Schritt – Gruppierung nach dem Autor

```
{ $lookup: {  
  from: "test",  
  localField: "dokumente.titel",  
  foreignField: "Titel",  
  as: "ereignis" } },
```



2. Schritt – Projektion der gewünschten Felder

```
{ $project: {  
  autor: "$autor",  
  ereignis: "$ereignis.Ort"  
 } }
```

Kollektion autoren

```
{  
  "_id": Object,  
  "autor": "Autor2",  
  "dokumente": Array  
    {  
      "0": Object  
        {  
          "titel": "Privat",  
          "typ": "text",  
          "anzahl": 1  
        }  
      }  
}
```



Objekt-Id

Kollektion test

```
{  
  "_id": ObjectId("5adb410728cb8"),  
  "Titel": "Privat",  
  "Datum": "2018-05-05",  
  "Ort": "at home"  
}
```

```
> db.autoren.aggregate([ { $lookup: { from: "test", localField: "dokumente.titel", foreignField: "Titel", as: "ereignis" } }, { $project: { autor: "$autor", ereignis: "$ereignis.Ort" } } ])  
{ "_id" : { "autor" : "Autor2" }, "ereignis" : [ "at home" ] }  
{ "_id" : { "autor" : "Autor1" }, "ereignis" : [ "Dortmund, EF 42", "at home" ] }
```

Aggregation Pipeline: Gruppierung

Ermittle die Anzahl und den Mittelwert der Bewertungen:

Ereignis	Gut	Mittel	Schlecht	weißNicht
Titel "Vorlesung"	3	2	1	0
Titel "Privat"	0	1	2	3



1. Schritt – \$unwind: Umwandlung eines Arrays in einzelne Datensätze

{ Titel: "Vorlesung",
Bewertung: [3, 2, 1, 0]}
{ Titel: "Privat",
Bewertung: [0, 1, 2, 3]}



{ Titel: "Vorlesung", Bewertung:3}
{ Titel: "Vorlesung", Bewertung:2}
{ Titel: "Privat", Bewertung:0}
{ Titel: "Vorlesung", Bewertung:1}
...
Bewertungskategorie?



2. Schritt - Gruppierung

Gruppierung (_id):
Anzahl Datensätze:

Mittelwert:

nach Titel
Rückführung auf eine Summe,
da kein Count verfügbar
nur **Mittelwert der Anzahl** der Bewertungen

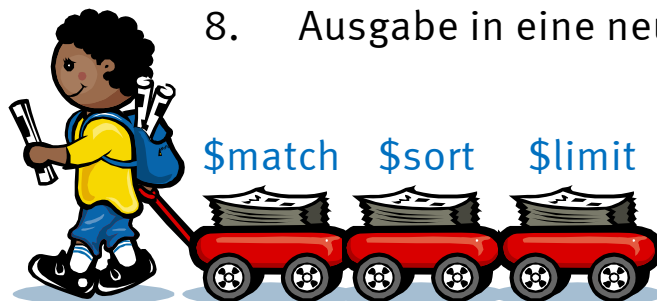
```
> db.test.aggregate([{$unwind:"$Bewertung"},{$group:{_id:{titel:"$Titel"},  
anzahl:{$sum:1}, mittelwert:{$avg:"$Bewertung"}}}])  
{ "_id" : { "titel" : "Privat" }, "anzahl" : 4, "mittelwert" : 1.5 }  
{ "_id" : { "titel" : "Vorlesung MDB" }, "anzahl" : 4, "mittelwert" : 1.5 }
```


Bei einer Aggregation Pipeline wird eine Anfrage durch Angabe der nacheinander (z.T. parallel) ausgeführten Berechnungsschritte formuliert.

Anm.: Eine Aggregation Pipeline ist auch die Basis von LINQ-Abfragen unter .NET

Verarbeitung der Anfrage (wie in einer Programmiersprache üblich):

1. Filtern/Selektion (\$match)
2. Sortieren (\$sort)
3. Pagination (\$limit und \$skip)
4. Projektion und Umbenennen (\$project)
5. Gruppierung und Aggregation (\$group)
6. Arrays aufteilen (\$unwind)
7. Verbundoperationen (\$lookup)
8. Ausgabe in eine neue Kollektion schreiben (\$out)



Kollektion

```
db.vorlesung.aggregate(  
  {$match: {Titel: "Moderne Datenbanken"}},  
  {$sort: {Titel:1}},  
  {$limit: 2},  
)
```

Agenda

1	Dokumentenorientierte Datenbanken	2
2	Abfragen	12
2.1	Query-by-Example	13
2.2	Map-Reduce-Algorithmus	18
2.3	Beispiel	33
2.4	Aggregation Pipeline	40
3	Diskussion	49

- Vorteile des Map-Reduce-Algorithmus:
 - Map- und Reduce-Phasen eignen sich sehr gut für die parallele Verarbeitung, da die Berechnung der Map-Funktionswerte voneinander unabhängig sind (→ Divide & Conquer-Strategie)
 - Im verteilten Fall kann (optional) noch eine Combine-Phase vorgeschaltet werden, bei der auf den lokalen Knoten die Zwischenergebnislisten sortiert und zusammengefasst werden.
 - Parallele Verarbeitung in einem Cluster
 - beschleunigt die Berechnung der Ergebnisliste
 - Ist erforderlich, wenn die Datenmengen für einen einzelnen Prozess (und das ausführende Rechnersystem) zu groß sind.
- Implementierung
 - Map-Reduce-Framework übernimmt
 - Effiziente Ausführung des Algorithmus zum Durchsuchen großer Datenmengen
 - Parallelisierung
 - Fehlertoleranz
 - Daten- und Lastverteilung
 - Monitoring
 - Der Anwender muss die Map- und Reduce-Funktionen definieren

Map Reduce ist nicht geeignet ...

- für Echtzeitanwendungen,
 - die Sortieralgorithmen benötigen bei hohen Trefferzahlen der Map-Funktion **VIEL** Zeit (Stunden!)
- für Verbundoperationen zwischen großen Datenmengen und komplexen Bedingungen
 - hoher I/O Aufwand, da MapReduce Daten-intensive ist
- für komplexen Algorithmen mit vielen Iterationen
 - da bereits ein hoher Implementierungsaufwand für einfache Aggregationen (z.B. Count) erforderlich ist.
- für KI-Algorithmen, welche die Daten mehrfach verarbeiten
- für die Traversierung von Graphen

- Typische Anwendungsfälle
 - Event Logging,
d.h. Ereignisse unterschiedlicher Art können gespeichert werden, z.B. Kaufereignisse, Suchereignisse,...
 - [Content Management Systeme](#) (CMS), Blogging Plattformen,
diese haben oftmals kein vordefiniertes Datenschema und unterstützen unterschiedliche Dokumentenformate
 - Anwendungen, die ein flexibles Datenschema benötigen,
z.B. für Web Analytics oder E-Commerce Anwendungen,

- Nicht geeignet ...
 - Für komplexe Transaktionen,
die unterschiedliche Operationen beinhalten oder verschiedene Dokumente miteinander verknüpfen,
 - Für ad-hoc Suche innerhalb von Dokumenten,
da die Dokumentenstruktur nicht festgelegt (normalisiert) ist ändern sich die ad-hoc Anfragen laufend.

- 1) Wodurch unterscheiden sich das XML- vom JSON-Datenformat?
- 2) Welche Möglichkeiten bieten Dokumentenorientierte Datenbanken, um Daten zu speichern, zu ändern und abzufragen?
- 3) Wie funktioniert eine Aggregation Pipeline und der Map-Reduce-Algorithmus?
- 4) Wie und wann wird der Map-Reduce-Algorithmus verwendet?
- 5) Welche Vor- und Nachteile bieten der Einsatz von dokumentenorientierten Datenbanken?

Vielen Dank für Ihre Aufmerksamkeit

- [EFHB 2010]
S. Edlich, A. Friedland, J. Hampe, B. Brauer
NoSQL – Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken
Hanser Verlag 2010
- [RW 2012] E. Redmond, J. Wilson, Sieben Wochen, sieben Datenbanken,
o'Reilly-Verlag
- [SF 2013] P. Sadalage, M. Fowler, NoSQL Distilled, Addison-Wesley