

# NoSQL Datenbanken

Vorlesung – Hochschule Mannheim

## Klassifizierungen

# Inhaltsverzeichnis

- ▶ Key-Value Stores
- ▶ **Dokumentenbasierte Datenbanken**
- ▶ Spaltenorientierte Datenbanken
- ▶ Graphendatenbanken

# Dokumentenbasierte Datenbanken


















**CouchDB**  
relax



**Couchbase**

# Weitere

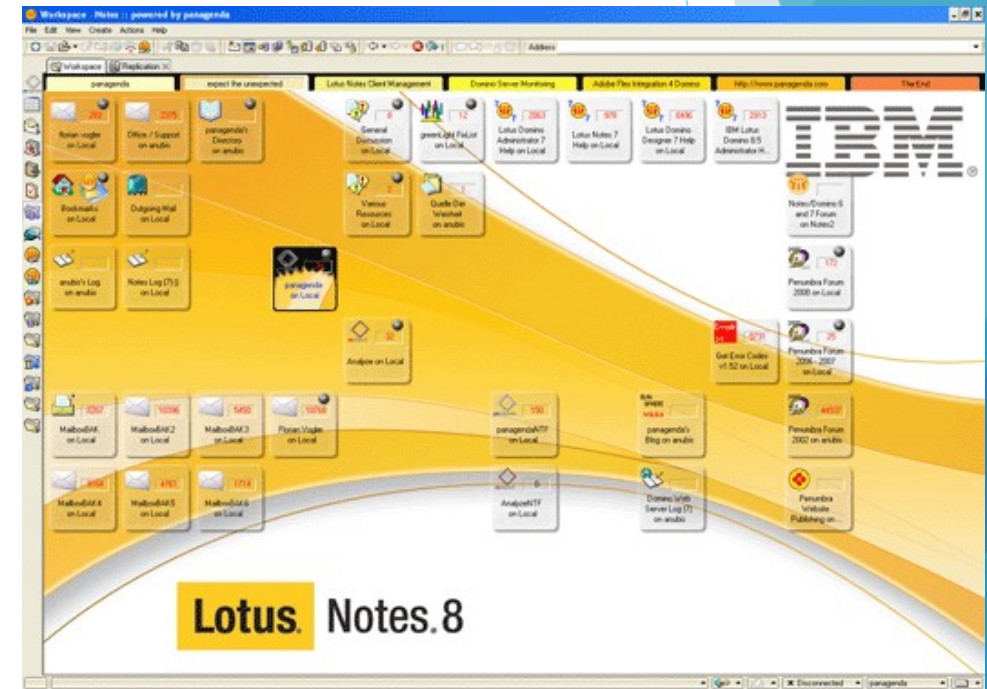
- ▶ OrientDB
- ▶ Amazon DynamoDB
- ▶ Google Cloud Datastore
- ▶ PouchDB

Rang			DBMS	Datenbankmodell	Punkte		
Nov 2015	Okt 2015	Nov 2014			Nov 2015	Okt 2015	Nov 2014
1.	1.	1.	MongoDB 	Document Store	304,61	+11,34	+59,87
2.	2.	2.	CouchDB	Document Store	26,37	-0,49	+0,57
3.	3.	3.	Couchbase 	Document Store	25,82	-0,37	+4,33
4.	4.	4.	Amazon DynamoDB	Multi-Model 	21,75	+0,42	+9,43
5.	5.	5.	MarkLogic	Multi-Model 	11,00	-0,34	+2,90
6.	6.	6.	RavenDB	Document Store	5,82	+0,00	+1,20
7.	 8.	 9.	OrientDB	Multi-Model 	5,50	+0,57	+3,48
8.	 7.	 7.	Cloudbant	Document Store	5,37	+0,18	+2,73
9.	9.	 8.	GemFire	Document Store	4,68	-0,01	+2,56
10.	10.	10.	RethinkDB	Document Store	3,68	+0,33	+2,72
11.	11.	11.	Datameer	Document Store	2,49	-0,06	+1,56
12.	12.	 13.	Microsoft Azure DocumentDB	Document Store	1,88	+0,32	+1,23
13.	13.	 17.	ArangoDB 	Multi-Model 	1,61	+0,13	+1,30
14.	14.		PouchDB	Document Store	1,51	+0,08	
15.	15.	 16.	CloudKit	Document Store	1,05	+0,00	+0,72

Quelle: <http://db-engines.com/de/ranking/document+store>

# Geschichte

- ▶ 1989 erschien die erste dokumentenorientierte Datenbank
  - Lotus Notes (nach März 2013 IBM Notes)
- ▶ Eigenes Datenbankmanagementsystem welches autark als eigener DB-Server mit einem Client reden konnte



# Dokumentenorientierte Datenbanken

- ▶ Gehören zu den zentralen NoSQL-Systemen
- ▶ Dokumente als Grundeinheit zur Speicherung
  - JSON / BSON
  - XML / YAML-Dokumente
  - Binary Large Objects
  - Textverarbeitungsprogrammdatei
- ▶ Kein festes Datenschema



# Datenmodell

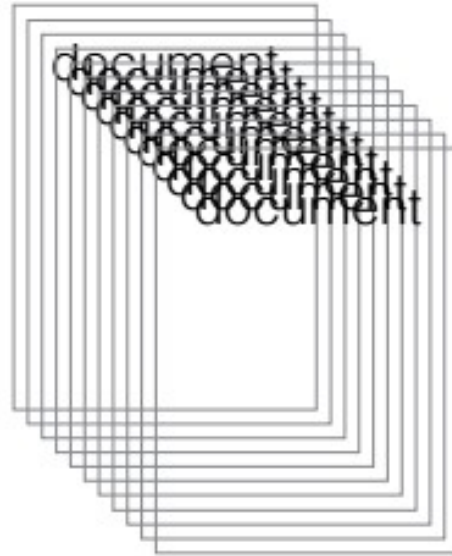
```
{  
  "_id" : 11112022  
  "Vorname" : "Christoph",  
  "Name": "Ns",  
  "Age": 28,  
  "Projects": ["Emily", "LycheeJS", "AE"],  
  "Pet" : {"name" : "Arthur", "Age": 11, "color" : "black"}  
}
```

# Datenmodell

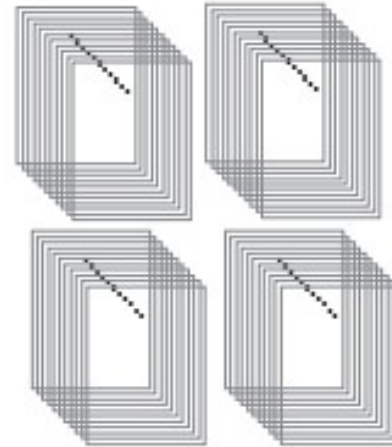
## document



## collection



## database



# Terminologien

SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row	Document (BSON)
column	field
index	index
table joins	embedded documents and linking
primary key	primary key
Specify any unique column or column combination as primary key.	the primary key is automatically set to the <code>_id</code> field.

# BSON (Binary JSON)

## ► Erweiterung des JSON Formats

- In JSON können Datenstrukturen wie *date* oder *byte* nicht dargestellt werden
- string
- integer (32- or 64-bit)
- double (64-bit)
- date (integer number of milliseconds since the Unix epoch)
- byte array (binary data)
- boolean (true and false)
- null
- BSON object
- BSON array





**FOURSQUARE**

**SOURCE** **forge**



**INTERACTIVE  
STUDIOS**

# MongoDB

- Abgeleitet von *humongous*, „gigantisch“
- Dokumentenbasierte schemafreie Datenbank
- 2009 von 10gen (später MongoDB Inc.)
- Veröffentlicht im Februar 2009 von MongoDB Inc.
- Open-Source-Datenbank (GNU AGPL v3.0)
- Entwickelt für geringe Reaktionszeiten bei großen Datenmengen

# MongoDB

- Geschrieben in C++
- Treiber für: C, C++, C#, Haskell, Java, Lisp, Perl, PHP, Python, Ruby und Scala
- Sehr gut mit NodeJS
- Große Community
- Bekannteste NoSQL Datenbank
- Eigene Abfragesprache (in JavaScript)



# MongoDB

- Eine Datenbank besteht aus „Collections“
  - Dort werden die eigentlichen Dokumente abgespeichert
- Ein Dokument ist maximal 8MB groß
  - Daten werden aufgeteilt und in separate Dokumente abgespeichert
  - IDs der Dokumente werden in Metadatendokument hinterlegt um Datenfragmente wieder zusammenzusetzen

# MongoDB

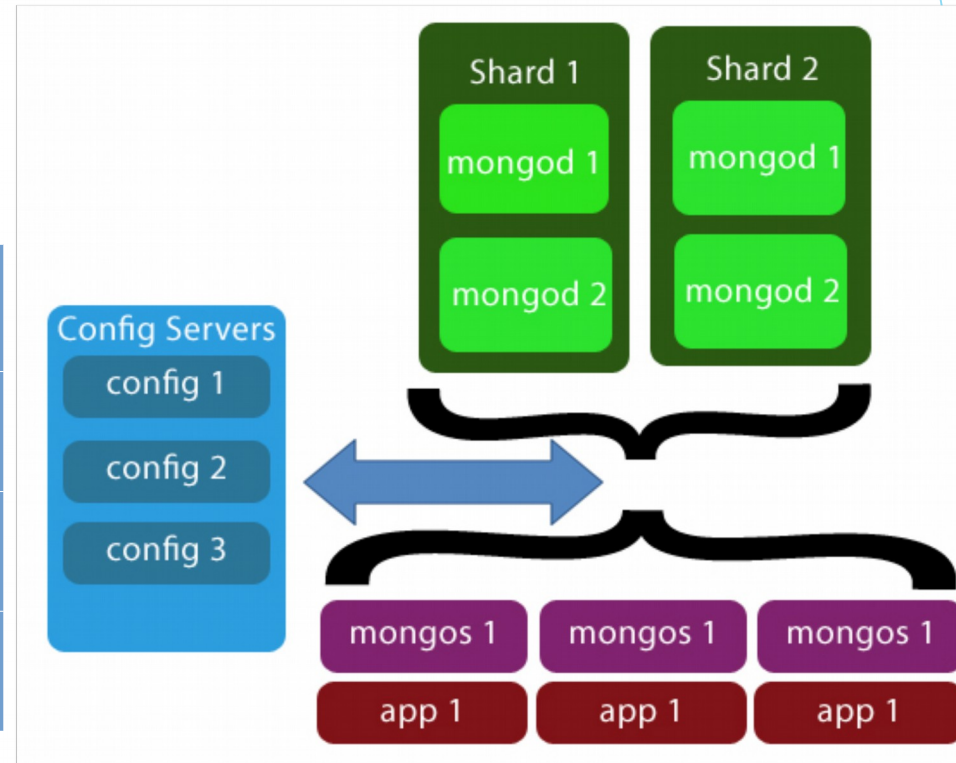
- MongoDB besitzt eine eigene Abfragesprache (In JavaScript)

## Verteilte Speichern von Daten:

- Replikation
  - Master-Slave
- Sharding
  - Daten werden anhand eines Shardingkeys verteilt
  - Range-Based und Hash-Based Sharding
  - Einzelne Datenpakete nennt man „Chunks“ und unterschiedliche Server „Shards“
  - Auf dem „Config Server“ wird hinterlegt auf welchem Shard sich die Dokumente befinden

# MongoDB

Chunk / Server	Obere Altersgrenze	Unter Altersgrenze
1	11	21
2	22	33
3	34	45






# Installation

## Current Stable Release (3.0.7)

10/13/2015: [Release Notes](#) | [Changelog](#)

Download Source: [tgz](#) | [zip](#)

 Windows

 Linux

 Mac OS X

Solaris

Select your distribution from the list or the legacy Linux 64-bit version if your distribution is unavailable. Keep in mind that this legacy Linux 64 build may lack the performance optimizations present in targeted builds.

### VERSION:

Ubuntu 14.04 Linux 64-bit ▼

The binary of this version has been compiled with SSL enabled and dynamically linked. This requires that SSL libraries be installed separately. See [here](#) for more information on installing OpenSSL.

### PACKAGE MANAGER:

[Instructions for installing with apt](#)

**BINARY:** [Installation Instructions](#) | [View Build Archive](#)

 [DOWNLOAD \(tgz\)](#)

[https://fastdl.mongodb.org/linux/mongodb-linux-x86\\_64-ubuntu1404-3.0.7.tgz](https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-ubuntu1404-3.0.7.tgz)

[Copy Link](#)

# Getting Started

- ▶ <https://www.mongodb.org/downloads>
- ▶ Windows, Mac OS X, Linux und Solaris Installer

## **Windows:**

- ▶ Mit der Kommandozeile (als Admin) in den entsprechenden Pfad
  - ▶ (...)\\MongoDB\\Server\\3.0\\bin  
mkdir data //Ordner zur Speicherung der Dokumente
  - ▶ Server starten (auf Port 27017):  
mongod --port 27017 --dbpath=./data
  - ▶ Client starten:  
mongo

# Getting Started

## Linux:

- ▶ Mit der Kommandozeile (als Admin) in den entsprechenden Pfad
  - ▶ (...)\\MongoDB\\Server\\3.0\\bin
- ```
md data/db //Ordner zur Speicherung der Dokumente
```
- ```
sudo ./mongod --dbpath= <pathtoData>
```
- ```
sudo ./mongod -dbpath=./data/db
```
- ▶ Server starten:

```
sudo ./mongod
```
- ▶ Client starten:

```
sudo ./mongo
```

# Übung

- ▶ MongoDB installieren
- ▶ Server starten
- ▶ Verbindung zum Server starten
- ▶ Eigene Datenbank *myCatsdb* anlegen
- ▶ Erstelle mehrere *Documente* und füge sie einer *Collection* “*MeineKatzen*” hinzufügen
- ▶ Attribute der Katzen: *name, color, age, high, weigth*

# Dokumente auslesen (Read)

*db.collectionName.find()*

- ▶ Zeigt alle Dokumente der entsprechende Collection an

*db.collectionName.find({alter:11})*

- ▶ Zeigt alle Dokumente der entsprechende Collection an die ein Attribut alter besitzen welches den Value 11 hat



# Dokumente auslesen (Read)

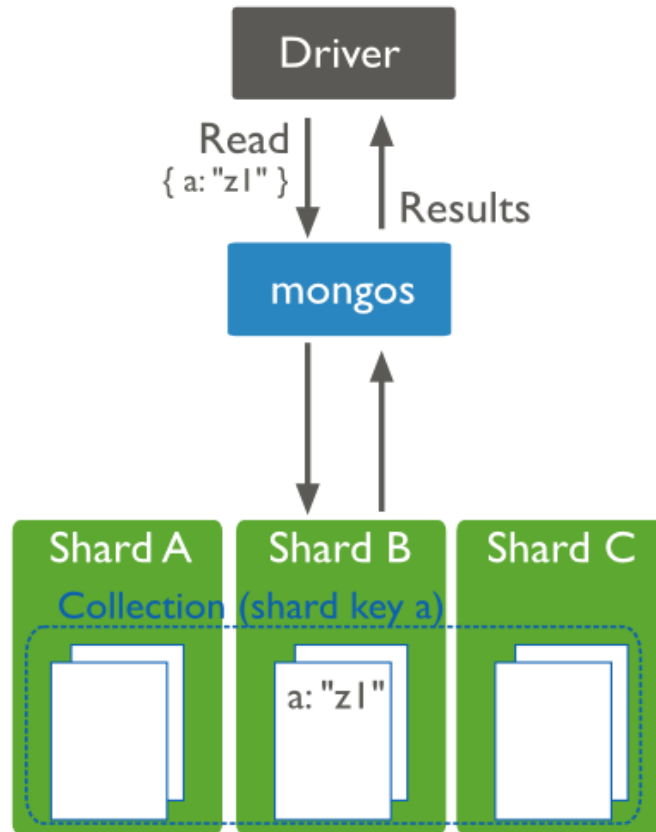
```
db.users.find(  
  { age: { $gt: 18 } },  
  { name: 1, address: 1 }  
) .limit(5)
```

← collection  
← query criteria  
← projection  
← cursor modifier

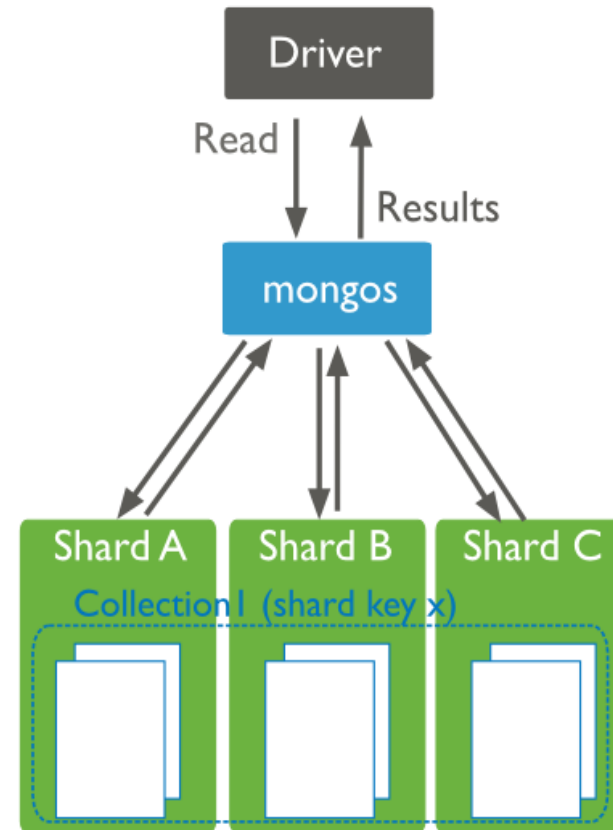
```
SELECT _id, name, address  
FROM users  
WHERE age > 18  
LIMIT 5
```

← projection  
← table  
← select criteria  
← cursor modifier

# Read über mehrere Shards



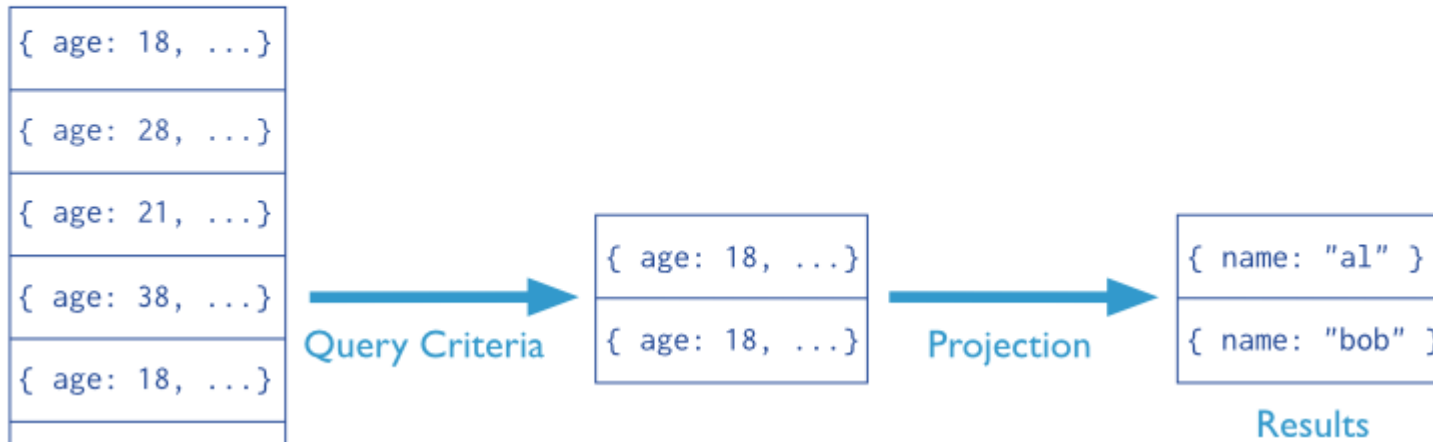
Sharding Key bekannt



Sharding Key unbekannt

# Projection

Collection                      Query Criteria                      Projection  
`db.users.find( { age: 18 }, { name: 1, _id: 0 } )`



- ▶ `_id` Feld wird als Defaultwert immer angezeigt
  - Explizit auf 0 setzen

# Dokumente manipulieren (Update)

- ▶ Mithilfe der Funktion *update()* einer Collection
- ▶ Diese Funktion hat verschiedene Operatoren um ein Dokument zu manipulieren:

\$set            Setzt einen neuen Wert für ein Feld

\$rename       Benennt ein Feld um

\$inc         Inkrementiert einen Wert

\$mul         Multipliziert einen Wert

\$unset       Löscht ein ausgewähltes Feld aus dem Dokument

```
db.people.update( { name: „Florian“ }, { name: „Florian“, rating: 1, score: 1 } )
```

<http://docs.mongodb.org/manual/reference/operator/update/>

# Dokumente manipulieren (Update)

## \$set

```
db.products.update(
  { _id: 100 },
  { $set:
    {
      quantity: 500,
      details: { model: "14Q3", make: "xyz" },
      tags: [ "coats", "outerwear", "clothing" ]
    }
  }
)
```

## \$inc

```
db.products.update(
  { sku: "abc123" },
  { $inc: { quantity: -2, "metrics.orders": 1 } }
)
```

# Dokumente manipulieren (Update)

## \$rename

```
db.students.update( { _id: 1 }, { $rename: { "nmae": "name" } } )
```

## \$mul

```
db.products.update(  
  { _id: 1 },  
  { $mul: { price: 1.25 } }  
)
```

## \$unset

```
db.products.update(  
  { sku: "unknown" },  
  { $unset: { quantity: "", instock: "" } }  
)
```

# Dokumente manipulieren (Insert)

```
db.users.insert (  ← collection
{
  name: "sue",      ← field: value
  age: 26,          ← field: value
  status: "A"       ← field: value
}                  } document
)
```

```
INSERT INTO users      ← table
( name, age, status )  ← columns
VALUES                ← values/row
( "sue", 26, "A" )
```

# Dokumente manipulieren (Insert multiple)

```
db.products.insert(  
  [  
    { _id: 11, item: "pencil", qty: 50, type: "no.2" },  
    { item: "pen", qty: 20 },  
    { item: "eraser", qty: 25 }  
  ],  
  { ordered: false }  
)  
{ "_id" : 11, "item" : "pencil", "qty" : 50, "type" : "no.2" }  
{ "_id" : ObjectId("51e0373c6f35bd826f47e9a0"), "item" : "pen", "qty" : 20 }  
{ "_id" : ObjectId("51e0373c6f35bd826f47e9a1"), "item" : "eraser", "qty" : 25 }
```

Kein Abbruch falls ein Datensatz fehlschlägt



# Dokumente manipulieren (Insert if not exist)

```
db.books.update(  
  { item: "ZZZ135" },  
  {  
    item: "ZZZ135",  
    stock: 5,  
    tags: [ "database" ]  
  },  
  { upsert: true }  
)
```

# Dokumente manipulieren (remove)

```
db.users.remove(  
  { status: "D" }  
)
```

← collection  
← remove criteria

```
DELETE FROM users  
WHERE status = 'D'
```

← table  
← delete criteria

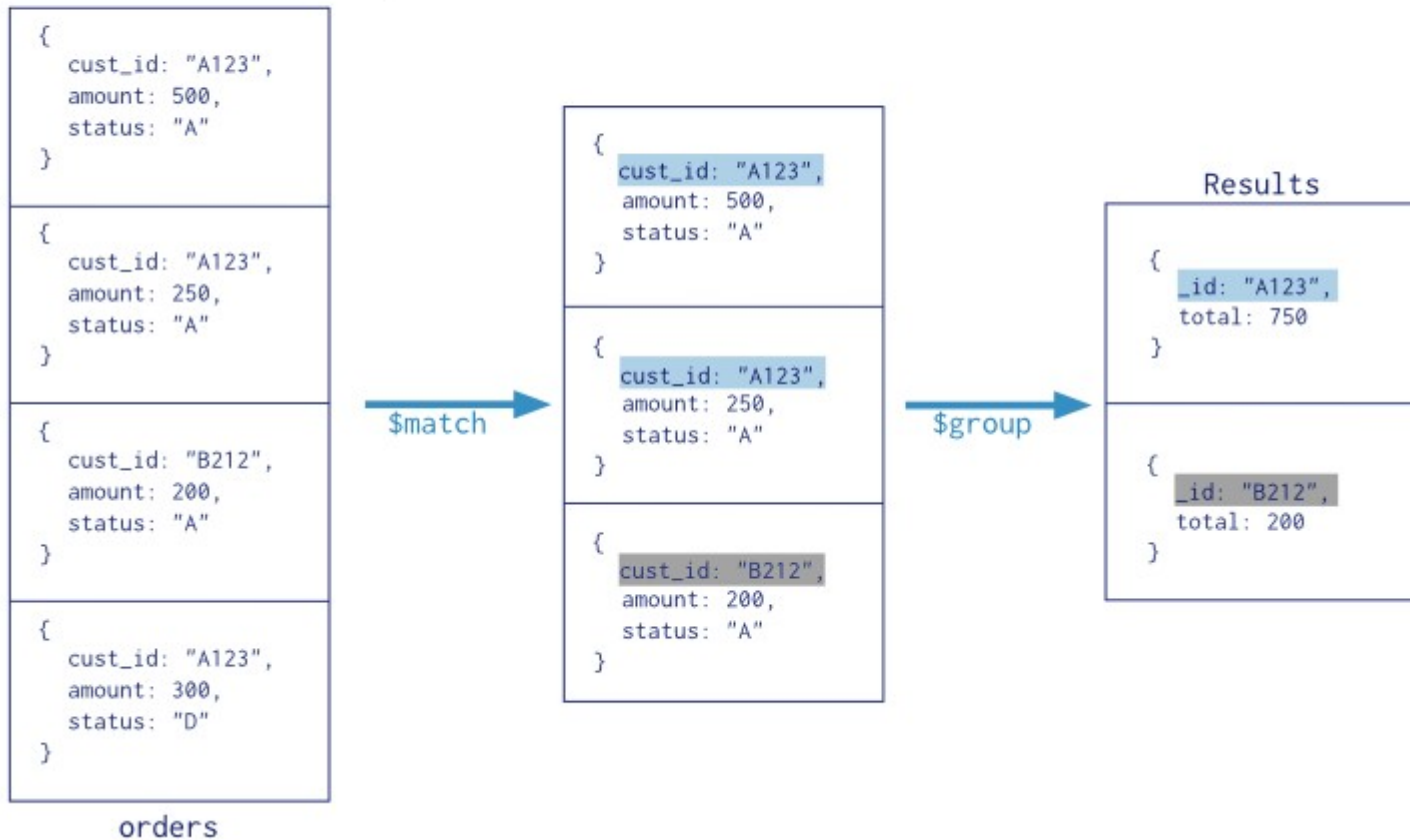
# Moodle Übungsblatt 1

# Aggregation

- ▶ Erst seit MongoDB 2.2
- ▶ Framework bietet Funktionen zur Aggregation
- ▶ Gruppierung, Count, Map-Reduce

# Aggregation

Collection  
↓  
`db.orders.aggregate( [`  
    \$match stage → `{ $match: { status: "A" } },`  
    \$group stage → `{ $group: { _id: "$cust_id", total: { $sum: "$amount" } } }`  
    `] )`



# Aggregation

## ► Weitere Operationen

`$project`     Erstellt ein neues Feld mit angegebenem Alias

`$sort`     Sortiert nach angegebenem Feld und Kriterium

`$limit`     Limitiert die Ausgabe der Dokumente

`db.users.aggregate(`

```
[  
  { $project : { name:{$toUpper:"$_id"} , _id:0 } },  
  { $sort : { name : 1 } },  
  { $limit : 5}  
]  
)
```

Collection  
↓  
`db.orders.distinct( "cust_id" )`

|                                                                |
|----------------------------------------------------------------|
| <pre>{   cust_id: "A123",   amount: 500,   status: "A" }</pre> |
| <pre>{   cust_id: "A123",   amount: 250,   status: "A" }</pre> |
| <pre>{   cust_id: "B212",   amount: 200,   status: "A" }</pre> |
| <pre>{   cust_id: "A123",   amount: 300,   status: "D" }</pre> |

orders

distinct → [ "A123", "B212" ]

# Vergleich mit Redis



# Schnittstelle

## MongoDB

- Schnittstelle zu C++
- Abfragesprache für einfach und komplexe Abfragen

## Redis

- Schnittstelle zu C++
- Keine komplexen Abfragen
- Für jeden unterstützten Datentyp existieren eigene Kommandos

# Datenmodell

## MongoDB

- Dokumente
- Befinden sich innerhalb einer Collection
- Daten frei von Strukturvorgaben speicherbar
- Verschiedene Datentypen vorhanden
  - Unter anderem zum Referenzieren von Daten

## Redis

- Key-Value-Paare
  - Schlüssel (Key) kann eine frei wählbare Zeichenkette sein
- Wert (Value) können verschiedene Datentypen beinhalten
  - (Strings, Sets, Listen und Hashes)

# Datenmodell

## MongoDB

- Intern werden Daten im BSON-Format abgespeichert
- Kann durch die Schemafreiheit gut auf Änderungen in der Datenbank reagieren
- Mehr Gestaltungsfreiheiten
  - Datentypen zum Referenzieren von Dokumenten

## Redis

- Intern werden Daten als String-/Bytearray abgelegt (typenlos)
- Manche API bieten Ausnahmen und erlauben das Speichern als JSON bzw. BSON Format
- Kann durch die Schemafreiheit gut auf Änderungen in der Datenbank reagieren

# Skalierung

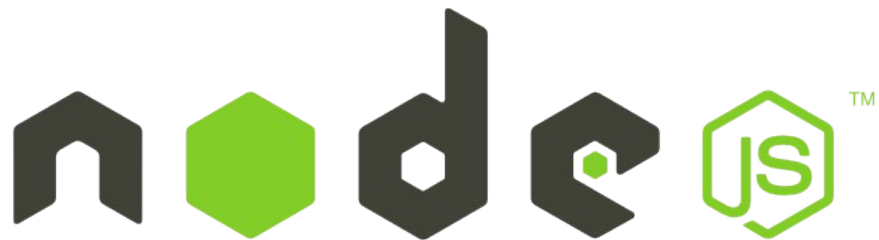
## MongoDB

- Bietet horizontales Skalieren
- Master/Slave-Replikation
- Beide Datenbanken bieten die Datenreplikation an
  - Hohe Ausfallsicherheit
- MongoDB bietet zusätzlich und mit geringerem Aufwand das Skalieren der Datenbank an

## Redis

- Kein natives Skalieren
  - Feature soll in späterer Version ergänzt werden
  - Anwender kann „Client Sharing“ implementieren
  - Programmieraufwand liegt dann beim Programmierer
- Master/Slave-Replikation

# Putting all together ...



# Treiber für NodeJS

- Den Treiber im Projektpfad via NPM installieren:

```
$ npm install mongodb
```

```
var url = 'mongodb://localhost:27017';  
var mongo = require('mongodb').MongoClient  
    mongo.connect(url, function(err, db) {  
        console.log("Connected correctly to server");  
        // db gives access to the database  
    })
```

```
makraus@MAKRAUS-PC  
$ node mongo.js  
Connected correctly to server
```

# Find Documents

- Abfrage einer Collection

```
db.collection('<collection name>')
```

- Abfrage aller Dokumente einer Collection

```
coll.find({}).toArray(function(err, documents){  
  console.log(documents);  
  db.close();  
});
```

```
$ node mongo.js  
Connected correctly to server  
[ { _id: 564dbcbfa7c6658e72d33557, name: 'morle' },  
  { _id: 564dbe3aa7c6658e72d33558, name: 'morle' },  
  { _id: 564dc03ea7c6658e72d33559, name: 'chris' } ]
```

# Find Documents

```
var url = 'mongodb://localhost:27017/mongoTest';
var mongo = require('mongodb').MongoClient
mongo.connect(url, function(err, db) {
  console.log("Connected correctly to server");
  // db gives access to the database

  var coll = db.collection('cats');

  coll.find({name: 'chris'}).toArray(function(err, documents){

    console.log(documents);
    db.close();

  });
})
```

```
$ node mongo.js
Connected correctly to server
[ { _id: 564dc03ea7c6658e72d33559, name: 'chris' } ]
```



# Find Documents

```
var value1 = parseInt(process.argv[2]);

coll.find({ age: { $gt: value1 } })
.toArray(function(err, documents){
    console.log(documents)
    db.close();
});
```

```
$ node mongo.js 5
Connected correctly to server
[ { _id: 564dc591a7c6658e72d3355a, name: 'chris', age: 11 },
  { _id: 564dc59ea7c6658e72d3355b, name: 'passo', age: 13 } ]
```

```
$ node mongo.js 12
Connected correctly to server
[ { _id: 564dc59ea7c6658e72d3355b, name: 'passo', age: 13 } ]
```

# Find Documents

```
collection.find({  
  name: 'foo'  
}, {  
  name: 1  
, age: 1  
, _id: 0  
}).toArray(function(err, documents) {  
  })
```

## Projektion

- Nur name, age werden als Attribute angezeigt
- \_id wird nicht mehr angezeigt

# Insert Documents

```
TODO: writeConcern
var coll = db.collection('parrots');
    coll.insert({name:"jascha"},
        function(err, data){
            //handle error
        });
```

```
> db.parrots.find()
{ "_id" : ObjectId("564dbcbfa7c6658e72d33557"), "name" : "morle" }
{ "_id" : ObjectId("564dbe3aa7c6658e72d33558"), "name" : "morle" }
{ "_id" : ObjectId("564dc03ea7c6658e72d33559"), "name" : "chris" }
{ "_id" : ObjectId("564dc591a7c6658e72d3355a"), "name" : "chris", "age" : 11 }
{ "_id" : ObjectId("564dc59ea7c6658e72d3355b"), "name" : "passo", "age" : 13 }
{ "_id" : ObjectId("564dcfb1203c01d01560a065"), "name" : "jascha" }
```

# Insert Documents

```
var coll = db.collection('parrots');
coll.insert(
  [
    { name:"jascha", age: 11 },
    { name:"harald", age: 13 },
    { name:"uta", age: 12 },
  ],
  function(err, data){
    //handle error
  });
```

# Update Documents

```
var coll = db.collection('parrots');
coll.update({
  name: "morle"
}, {
  $set: {
    name: "mo"
  }
})
```

```
> db.parrots.find()
{ "_id" : ObjectId("564dbcbfa7c6658e72d33557"), "name" : "morle" }
{ "_id" : ObjectId("564dbe3aa7c6658e72d33558"), "name" : "morle" }
{ "_id" : ObjectId("564dc03ea7c6658e72d33559"), "name" : "chris" }
{ "_id" : ObjectId("564dc591a7c6658e72d3355a"), "name" : "chris", "age" : 11 }
{ "_id" : ObjectId("564dc59ea7c6658e72d3355b"), "name" : "passo", "age" : 13 }
{ "_id" : ObjectId("564dcfb1203c01d01560a065"), "name" : "jascha" }
> db.parrots.find()
{ "_id" : ObjectId("564dbcbfa7c6658e72d33557"), "name" : "mo" }
{ "_id" : ObjectId("564dbe3aa7c6658e72d33558"), "name" : "morle" }
{ "_id" : ObjectId("564dc03ea7c6658e72d33559"), "name" : "chris" }
{ "_id" : ObjectId("564dc591a7c6658e72d3355a"), "name" : "chris", "age" : 11 }
{ "_id" : ObjectId("564dc59ea7c6658e72d3355b"), "name" : "passo", "age" : 13 }
{ "_id" : ObjectId("564dcfb1203c01d01560a065"), "name" : "jascha" }
```

# Update Documents

```
coll.find({name: 'jascha'}).toArray(function(err, documents){  
    //documents is an array  
    id = documents[0]._id;  
    coll.update({  
        _id: id  
    }, {  
        $set: {  
            name: "Sascha"  
        }  
    })  
    db.close();  
});
```

# Remove Documents

//Entfernt das Dokument mit dem Namen "Sascha"

```
coll.remove({  
  name: 'Sascha'  
},)
```

//Entfernt alle Dokumente deren Attribut 'age' größer als 12 ist.

```
coll.remove({  
  { age: { $gt: 12 } }  
},)
```

# Aggregation

Erstellung von Metadaten aus Daten:

- Zuordnung der Daten zu eine Gruppe
- Erstellen von allgemeinen Aussage der spezifischen Gruppe

- 1) Auswahlkriterium für Gruppierung finden (Matching)
- 2) Attribute des neuen Gruppenobjekts (Eigenschaften der Gruppe) bestimmen
- 3) Operationen für die Wertermittlung der Eigenschaften



# Aggregation

```
coll.aggregate([
```

- Match-Klausel: Selektiere Datensätze nach entsprechendem Parameter

```
{ $match: { status: 'A' } }
```

- Gruppiere ausgewählte Datensätze und erstelle ein neues Objekt daraus

```
, { $group: {  
  //neues Attribut _id mit dem Wert 'sum'  
  _id: 'sum'  
  //neues Attribut sum mit einem errechnetem Wert  
  , sum: {
```

- \$sum Operator zur Summierung. Der Wert ist hierbei der Attributkey über den summiert werden soll

```
  // $sum Operator zum summieren der Werte  
  $sum: '$value'
```

```
  }  
} }  
])
```

# Aggregation

```
.toArray(function(err, results) {  
  // handle error  
  console.log(results)  
  // => [  
  // => { _id: 'sum', sum: 12 }  
  // => ]  
})
```

```
{ "_id" : ObjectId("564de1b6a7c6658e72d3355f"), "status" : "A", "value" : 1 }  
{ "_id" : ObjectId("564de1b6a7c6658e72d33560"), "status" : "B", "value" : 10 }  
{ "_id" : ObjectId("564de1b6a7c6658e72d33561"), "status" : "A", "value" : 11 }
```

# Aggregation

Weitere Operatoren (neben \$sum)

- **\$avg**  
Bestimmt den Mittelwert
- **\$first**  
Ermittelt den Wert des ersten Document einer Gruppe
- **\$last**  
Ermittelt den Wert des letzten Document einer Gruppe
- **\$max**  
Ermittelt den größten Wert eines Documents in einer Gruppe
- **\$min**  
Ermittelt den kleinsten Wert eines Documents in einer Gruppe