

MongoDB

Was ist MongoDB?

- Namen abgeleitet vom engl. humongous (=gigantisch)
- Dokumentenorientierte **NoSQL-Datenbank**
- Speichert Daten im **JSON-Format**
- Entwickelt in der Programmiersprache C++
- Ursprünglich entwickelt von der Firma 10gen, 2009 als Open-Source bereitgestellt und 2013 in MongoDB umbenannt
- Aktuell die meistverbreitetste noSQL-DB

Relationale Datenbanken vs. MongoDB

SQL

Datenbank 

Tabelle

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	228
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Datensatz / Row

2134	Augustinus	C3	309
------	------------	----	-----

MongoDB

Datenbank 

Collection 

Dokument 

Relationale Datenbanken vs. MongoDB

SQL

MongoDB

Artikel			
ArtikelNr	Name	Preis	HerstellerID
1001	Monitor 22"	159.00	1
1002	Monitor 24"	199.00	1
1003	Laptop XY	478.00	2

Hersteller		
HerstellerID	Bezeichnung	...
1	Acer	...
2	Toshiba	...

In Dokument gruppierte Daten

ArtikelNr: 1001

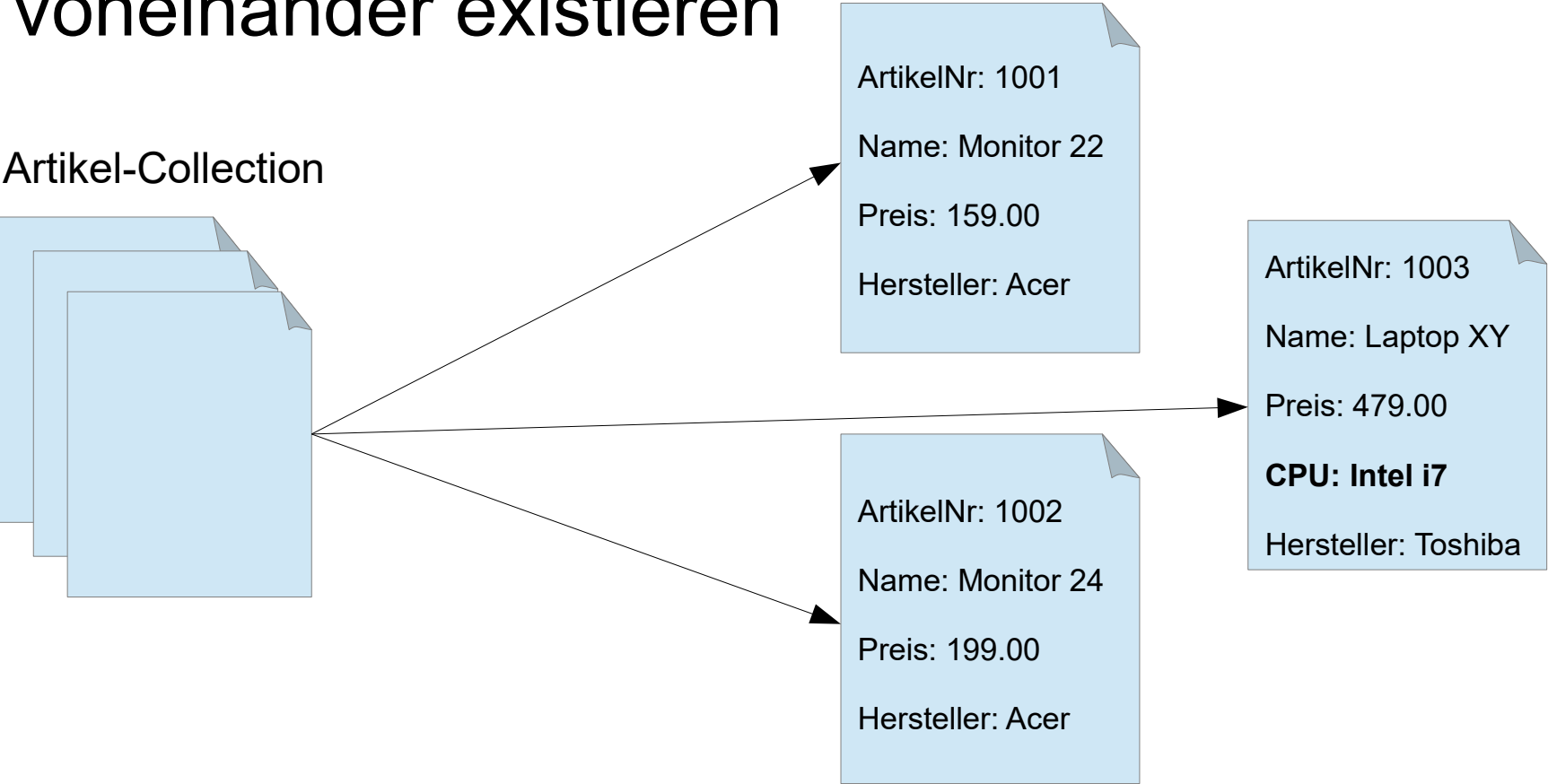
Name: Monitor 22

Preis: 159.00

Hersteller: Acer

Collections

- Sind Gruppierungen von Dokumenten
- Dokumente können jedoch unterschiedliche Felder haben, da diese unabhängig voneinander existieren



Installation von MongoDB



Installieren Sie nun MongoDB auf Ihrem Rechner.
Folgen Sie dazu den Schritten in der Moodle-Lektion!

Erste Kommandos in MongoDB

Zeigt verfügbare Kommandos

```
> help
db.help()
db.mycoll.help()
....
```

Listet vorhandene Datenbanken auf

```
> show dbs
admin    0.000GB
local    0.000GB
```

Wechselt zur angegebenen Datenbank. Wenn nicht vorhanden wird diese beim ersten Schreibvorgang angelegt

```
> use shopdb
switched to db shopdb

> db
shopdb
```

Zeigt die aktuelle Datenbank

Dokumente einfügen

Dokumente müssen immer in eine **Collection** gespeichert werden! Existiert diese noch nicht, wird sie automatisch angelegt.

Collection „artikel“ wird
autom. angelegt!

```
> show collections
artikel
```

```
> db.artikel.insert(
... {
...   "ArtikelNr": 1001,
...   "Name": "Monitor 22",
...   "Preis": 159.00,
...   "Hersteller": "Acer"
... }
... )
WriteResult({ "nInserted" : 1 })
```

Bestätigung über 1
eingefügtes Dokument

Vorhandene Dokumente abrufen

Die Methode `find()` liefert alle Dokumente der Collection zurück.

Eindeutige ID, die autom. generiert wird, sofern sie nicht übergeben wird

```
> db.artikel.find()
{
  "_id" : ObjectId("59c29fd5abb1c1d26f101f21"),
  "ArtikelNr" : 1001,
  "Name" : "Monitor 22",
  "Preis" : 159,
  "Hersteller" : "Acer"
}
```



Fügen Sie die 2 restlichen Artikel in die Collection „artikel“ ein!

Einfache Suche

Der Methode `find()` kann ein Dokument mit Suchparameter übergeben werden.

Query mit
Key/Value - Paaren

```
> db.artikel.find({"Preis": 199})
{
  "_id" : ObjectId("59c2a575abb1c1d26f101f23"),
  "ArtikelNr" : 1002,
  "Name" : "Monitor 24",
  "Preis" : 199,
  "Hersteller" : "Acer"
}
```



Fragen Sie alle Artikel vom Hersteller „Acer“ ab.
Was passiert, wenn mehrere Treffer gefunden werden?

Datum einfügen – JavaScript Date

Dates können mittels JavaScript Date-Objekt hinzugefügt werden.

```
{  
  "ArtikelNr" : 1002,  
  "Name" : "Monitor 24",  
  "Preis" : 199,  
  "Hersteller" : "Acer",  
  "Verkaufsstart": new Date(2017, 09, 21)  
}
```

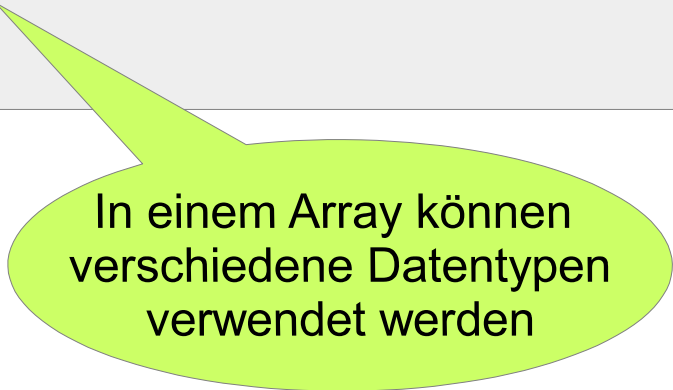
Datum wird im
ISO-Format
abgespeichert

```
"Verkaufsstart" : ISODate("2017-10-20T22:00:00Z")
```

Arrays

In MongoDB ist es möglich, Listen als Arrays im Dokument abzuspeichern.

```
{
  "ArtikelNr" : 1002,
  "Name" : "Monitor 24",
  "Preis" : 199,
  "Hersteller" : "Acer",
  "Verkaufsstart": new Date(2017, 09, 21),
  "Tags" : ["Monitor", 24, "Acer", "3D"]
}
```



In einem Array können verschiedene Datentypen verwendet werden

Embedded Documents

Dokumente können verschachtelt werden. Kind-Dokumente benötigen keine eigene ID, da sie Bestandteil des Hauptdokuments sind!

```
{
  "ArtikelNr" : 1002,
  "Name" : "Monitor 24",
  "Preis" : 199,
  "Hersteller" : "Acer",
  "Verkaufsstart": new Date(2017, 09, 21),
  "Tags": ["Monitor", 24, "Acer", "3D"],
  "Spezifikation": {
    "Spannung": 240, "Leistung": 35
  }
}
```

Aktualisieren der Artikelliste



Löschen Sie alle Artikel aus der Collection:

```
db.artikel.remove({})
```

Fügen Sie nun die 3 Artikel erneut ein, wobei folgende Daten ergänzt werden:

- Verkaufsstart
- Tags
- Spezifikation

ArtikelNr	Verkaufsstart	Tags	Spezifikation (V, W)
1001	1.1.2015	Monitor, 22, Acer	240, 25
1002	21.9.2017	Monitor, 24, Acer, 3D	240, 35
1003	1.1.2017	Laptop, i7, SSD	240, 55

Suchen in Arrays

Array-Einträge werden wie separate Werte abgefragt.

```
> db.artikel.find({ "Tags" : "Monitor" })

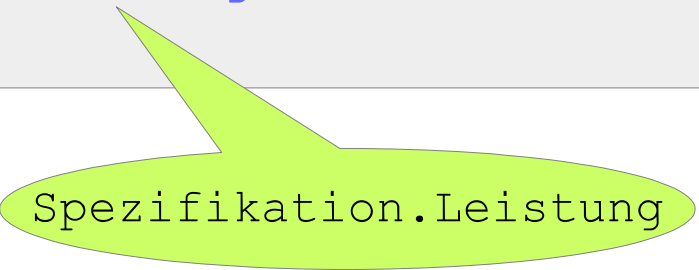
{ "_id" : ObjectId("59c2b2d4abb1c1d26f101f26"),
  "ArtikelNr" : 1001,
  ...,
  "Tags" : [ "Monitor", 22, "Acer" ],
  ... }
{ "_id" : ObjectId("59c2b2deabb1c1d26f101f27"),
  "ArtikelNr" : 1002,
  ...
  "Tags" : [ "Monitor", 24, "Acer", "3D" ],
  ... }
```

Suchen in Embedded Documents

Felder in Embedded Documents können über die Punkt-Notation spezifiziert werden.

```
> db.artikel.find({"Spezifikation.Leistung":25})

{ "_id" : ObjectId("59c2b2d4abb1c1d26f101f26"),
  "ArtikelNr" : 1001,
  ...
  "Spezifikation" : {
    "Spannung" : 240, "Leistung" : 25 }
}
```



Spezifikation.Leistung

Dokumente löschen/updaten

Dokumente löschen

Mittels **remove ()** können Dokumente aus einer Collection gelöscht werden. Als Parameter muss ein Dokument mit Sucheigenschaften übergeben werden. Der zweite (optionale) Parameter gibt an, ob nur der erste Treffer gelöscht werden soll.

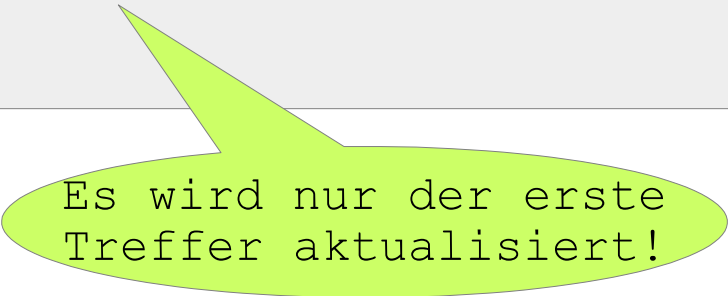
```
> db.artikel.remove({"Hersteller":"Acer", true})  
WriteResult({ "nRemoved" : 1 })
```

```
> db.artikel.remove({"Hersteller":"Acer"})  
WriteResult({ "nRemoved" : 2 })
```

Dokumente updaten

Mittels `update(query, update)` können Dokumente verändert werden. Der erste Parameter spezifiziert die Query, der zweite Parameter ist der Update-Parameter. Update-Operatoren beginnen immer mit \$.

```
> db.artikel.update({"Tags": "Monitor"},  
                    {"$set": {"Preis": 179}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0,  
              "nModified" : 1 })
```

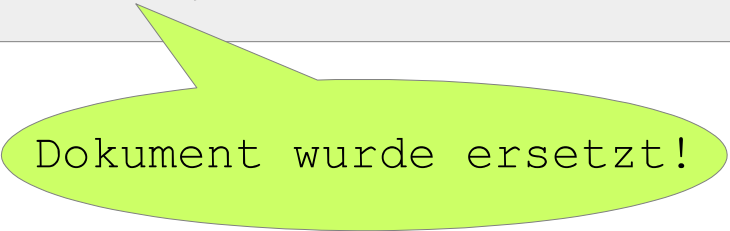


Es wird nur der erste
Treffer aktualisiert!

Dokumente updaten

Wird kein Update-Operator angegeben sondern stattdessen ein Key/Value-Dokument, wird das Dokument mit diesem ersetzt (ausgenommen ID).

```
> db.artikel.update( {"Tags": "Monitor"},  
                     {"Preis": 179})  
  
...  
{ "_id" : ObjectId("59c679a8e93494268e62ed04"),  
  "Preis" : 179 }
```



Dokument wurde ersetzt!

Mehrere Dokumente updaten

Wird der update-Methode als dritter Parameter `{"multi": true}` mitgegeben, so werden alle Treffer der Query aktualisiert!

```
> db.artikel.update({"Tags": "Monitor",  
                    {"$set": {"Preis": 179}},  
                    {"multi": true})  
  
WriteResult({ "nMatched" : 2,  
              "nUpserted" : 0,  
              "nModified" : 2 })
```

upsert

Wird der `update`-Methode als dritter Parameter `{"upsert": true}` mitgegeben, so wird das Dokument angelegt, sofern es noch nicht existiert.

```
> db.likes.update({"ArtikelNr": 1001},
                  {"$inc": {"anzahl": 1}},
                  {"upsert": true})

WriteResult({
  "nMatched" : 0,
  "nUpserted" : 1,
  "nModified" : 0,
  "_id" : ObjectId("59c6....")
})
```

\$unset – Felder löschen

Mittels `$unset` können Felder aus Dokumenten gelöscht werden.

```
> db.artikel.update({},  
                    {"$unset": {"verkauft": ""}},  
                    {"multi": true})  
WriteResult({ "nMatched" : 3,  
              "nUpserted" : 0,  
              "nModified" : 1 })
```

\$rename – Felder umbenennen

Mittels **\$rename** können Felder umbenannt werden.

```
> db.artikel.update({},  
    {"$rename": {"Name": "Bezeichnung"}},  
    {"multi": true})  
WriteResult({ "nMatched" : 3,  
    "nUpserted" : 0,  
    "nModified" : 3 })
```


Werte in einem Array aktualisieren

Auf die Werte kann mittels Punktnotation zugegriffen werden, wobei der Index des Array-Elements anzugeben ist.

```
{
  "ArtikelNr" : 1003,
  ...
  "Tags": ["Laptop", "i7", "SSD"],
  ....
}
```

0 1 2

```
> db.artikel.update({"ArtikelNr": 1003},
  {"$set": {"Tags.0": "Notebook"}},
  {"multi": true})
WriteResult({ "nMatched" : 1,
  "nUpserted" : 0,
  "nModified" : 1 })
```

Werte in einem Array aktualisieren

Sollen alle Vorkommen eines Wertes in dem Array aller Dokumente ersetzt werden, kann der Positional-Operator (\$) verwendet werden.

```
> db.artikel.update({"Tags": "Monitor"},  
    {"$set": {"Tags.$": "LED-Monitor"}},  
    {"multi": true})  
WriteResult({ "nMatched" : 2,  
    "nUpserted" : 0,  
    "nModified" : 2 })
```

Werte aus einem Array löschen

Mittels `$pop` kann der erste oder letzte Werte aus einem Array gelöscht werden. Dabei gilt:

- 1 löscht das letzte Element
- 1 löscht das erste Element

"Tags" : ["LED-Monitor", 22, "Acer"]

```
> db.artikel.update( {"ArtikelNr": 1001},  
  {"$pop": {"Tags": 1}})
```

"Tags" : ["LED-Monitor", 22, ~~"Acer"~~]

```
> db.artikel.update( {"ArtikelNr": 1001},  
  {"$pop": {"Tags": -1}})
```

"Tags" : [~~"LED-Monitor"~~, 22]

Werte zu einem Array hinzufügen

Mittels `$push` kann ein neuer Wert an ein Array ergänzt werden.

`$addToSet` ergänzt den Eintrag nur, wenn er nicht schon vorhanden ist.

```
> db.artikel.update({"ArtikelNr": 1001},
  {"$push": {"Tags": "Acer"}})

> db.artikel.update({"ArtikelNr": 1001},
  {"$addToSet": {"Tags": "LED-Monitor"}})
```

Werte aus einem Array löschen

Mittels `$pull` können alle Vorkommen eines Wertes aus dem angegebenen Array entfernt werden – unabhängig von seiner(n) Position(en)

```
> db.artikel.update({"ArtikelNr": 1001},  
                    {"$pull": {"Tags": "Acer"}})
```

Queries

Suche mit mehreren Kriterien

Bei einer Suchanfrage können im Suchdokument mehrere Kriterien angegeben werden.

```
> db.artikel.find({"Tags": "LED-Monitor",
                  "Hersteller": "Acer"})

{ "_id" : ObjectId("59c2b2d4abb1c1d26f101f26"),
  "ArtikelNr" : 1001,
  ...,
  "Hersteller" : "Acer",
  "Tags" : [ "LED-Monitor", 22, "Acer" ],
  ... }
```

Vergleichsoperatoren

\$gt greater than

\$gte greater then or equal

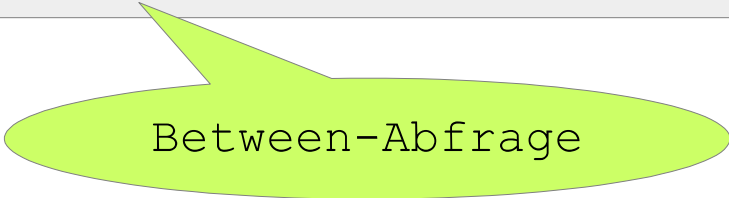
\$lt lower than

\$lte less then or equal

\$ne not equal

```
> db.artikel.find({"Preis": {"$lte": 179}})
```

```
> db.artikel.find({"Preis":  
    {"$gt": 160, "$lt": 200}})
```



Between-Abfrage

\$elemMatch - Bereichssuche in Arrays

```
{
  Bezeichnung: "LED-TV",
  Diagonalen: [40, 47, 48],
  ....
}
```

40 entspricht
der Suche

```
{
  Bezeichnung: "Plasma-TV",
  Diagonalen: [32, 34, 47],
  ....
}
```

```
> db.artikel.find({"Diagonalen":
  {"$elemMatch": {"$gt":34, "$lt":47}}})
```

Mind. 1 Eintrag muß größer
sein als 34 **und** kleiner
als 47!

Bereichssuche in Arrays

```
{
  Bezeichnung: "LED-TV",
  Diagonalen: [40, 47, 48],
  ....
}
```

```
{
  Bezeichnung: "Plasma-TV",
  Diagonalen: [32, 34, 47],
  ....
}
```

32	<	47	-> OK
34	<	47	-> OK
47	>	34	-> OK

```
> db.artikel.find({"Diagonalen":
  {"$gt":34, "$lt":47}})
```

Jeder Eintrag im Array wird geprüft:
Wenn für jede Bedingung mind. 1
Treffer gefunden wurde, wird
das Dokument geliefert!

Projektionen

Der Funktion `find()` kann als zweiter Parameter eine Projektionsdokument mitgegeben werden. Darin werden die zu liefernden Attribute angegeben mit dem jeweiligen Wert „true“.

```
> db.artikel.find({},  
  {"Bezeichnung": true, "Preis": true})  
  
{ "_id" : ObjectId("59..."),  
  "Preis" : 179, "Bezeichnung" : "Monitor 24" }  
{ "_id" : ObjectId("59..."),  
  "Preis" : 479, "Bezeichnung" : "Laptop XY" }  
{ "_id" : ObjectId("59..."),  
  "Preis" : 179, "Bezeichnung" : "Monitor 22" }
```

Projektionen

Wird ein Feld mit „true“ angegeben, werden alle anderen ausgenommen die ID mit false vorbelegt.

Wird ein false-Wert übergeben werden alle anderen mit true vorbelegt. Die ID muss immer explizit deaktiviert werden, wenn nicht gewünscht.

```
> db.artikel.find({},
  {"Hersteller": false, "Spezifikation": false})

{ "_id" : ObjectId("59..."),
  "ArtikelNr" : 1002,
  "Preis" : 179,
  "Verkaufsstart" : ISODate("2017-10-20..."),
  "Tags" : [ "LED-Monitor", 24, "Acer", "3D" ],
  "Bezeichnung" : "Monitor 24"
}
```

count()

`count()` kann nach dem `find()`-Aufruf angegeben werden und liefert die Anzahl an Treffer zurück.

```
> db.artikel.find({},  
  {"Hersteller": false, "Spezifikation": false})  
  .count()
```

5

sort()

`sort()` kann nach dem `find()`-Aufruf angegeben werden und sortiert nach dem angegebenen Kriterium.

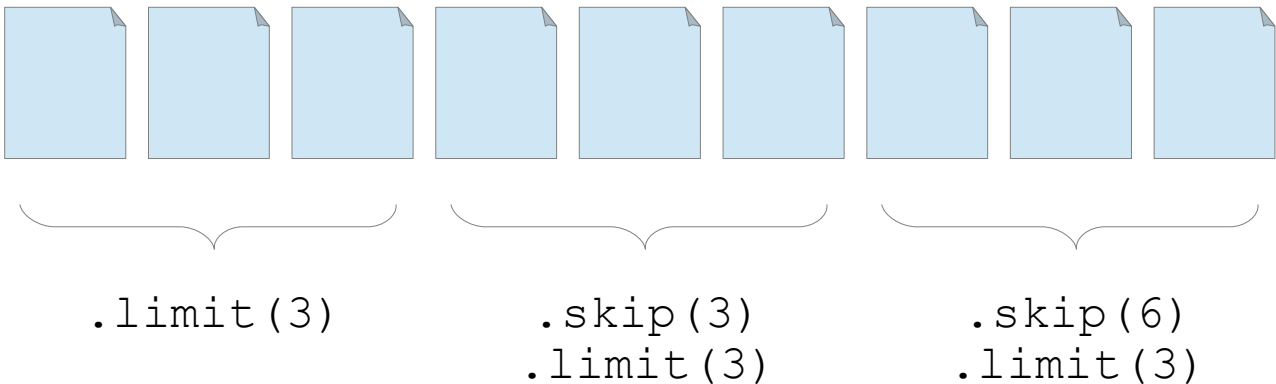
1 ascending

-1 descending

```
> db.artikel.find({},  
  {"Bezeichnung": true})  
  .sort({"Bezeichnung": 1})  
{ "_id" : ObjectId("59..."),  
  "Bezeichnung" : "Laptop XY" }  
{ "_id" : ObjectId("59..."),  
  "Bezeichnung" : "Monitor 22" }  
{ "_id" : ObjectId("59..."),  
  "Bezeichnung" : "Monitor 24" }
```

skip() und limit()

Notwendig für „Pagination“



```
> db.artikel.find().skip(3).limit(3)
```

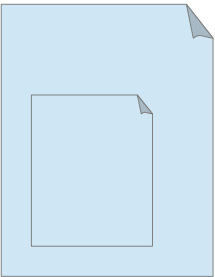
Datenstrukturen Embed or Reference?

Dokumente referenzieren

Embedded Documents bieten gute Performance, sofern die Daten häufig gemeinsam benötigt werden. Allerdings entstehen dadurch Redundanzen, die zu inkonsistenten Daten führen können.

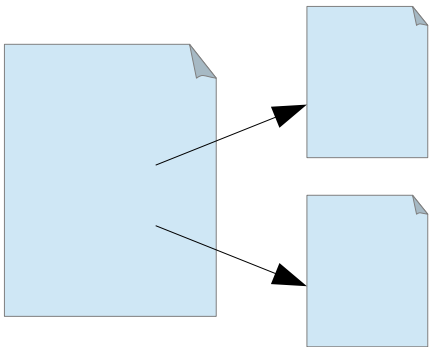
Deshalb ist es manchmal sinnvoll, Dokumente getrennt abzuspeichern und zu referenzieren.

Embedding



- + Single query
- + Document wird mit Hauptdokument mitgelesen
- + Atomares Schreiben

Referencing



- benötigt 2 Queries
- +/- Dokumente existieren unabhängig voneinander

Dokumente referenzieren

Anlegen der Hersteller-Dokumente:

```
{
  "_id": "Acer",
  "Anschrift": ...
  ....
}
```

```
{
  "_id": "Toshiba",
  "Anschrift": ...
  ....
}
```

Referenzieren im Artikel:

```
{
  "ArtikelNr": 1001,
  "Hersteller_id":
    "Acer",
  ....
}
```

```
{
  "ArtikelNr": 1003,
  "Hersteller_id":
    "Toshiba",
  ....
}
```

```
db.artikel.insert({
  "ArtikelNr": 1001,
  "Hersteller_id":
    "Acer",
  ....
})
```

Embed oder Reference?

Werden die Daten oft zusammen benutzt?

	Immer	Gelegentlich	Selten
Embed	✓	✓	
Reference		✓	✓

Wie groß sind die Daten?

	< 100	Einige 100	Tausende
Embed	✓	✓	
Reference		✓	✓

Wie oft ändern sich die Daten?

	Nie/Kaum	Gelegentlich	Häufig
Embed	✓	✓	
Reference		✓	✓

Aggregatfunktionen

aggregate()

Mittels der aggregate()-Funktion können Aggregatfunktionen wie Gruppierungen ausgeführt werden.

```
db.collection.aggregate(  
  [{"<operator>": {"<key>": "<value>"} }]
```

\$group

```
> db.artikel.aggregate([
  {"$group": {"_id": "$Hersteller"}}])

{ "_id" : "Toshiba" }
{ "_id" : "Acer" }
```

Sog. Field-Paths müssen mit einem \$-Zeichen beginnen und stellen Links dar

Akkumulatoren - \$sum

Alle Angaben nach dem Group-Key sind Akkumulatoren, die Berechnungen für die jeweilige Gruppe durchführen.

Werte, die mit \$ beginnen, repräsentieren Field-Paths!

```
> db.artikel.aggregate([
  {"$group": {"_id": "$Hersteller",
    "Anzahl": {"$sum": 1}}}
])
```

Felder, die mit \$ beginnen, sind Operatoren!

```
{ "_id" : "Toshiba", "Anzahl" : 1 }
{ "_id" : "Acer", "Anzahl" : 2 }
```

Akkumulatoren - \$avg

Berechnet den Durchschnitt pro Gruppe.

```
> db.artikel.aggregate([
  { "$group": { "_id": "$Hersteller",
    "Anzahl": { "$sum": 1 },
    "DS-Verbrauch": {
      { "$avg": "$Spezifikation.Leistung" } }
    }
  }
])

{ "_id" : "Toshiba", "Anzahl" : 1,
  "DS-Verbrauch:" : 55 }
{ "_id" : "Acer", "Anzahl" : 2,
  "DS-Verbrauch:" : 30 }
```


Akkumulatoren - \$max

Ermittelt den Maximalwert pro Gruppe.

```
> db.artikel.aggregate([  
  { "$group": { "_id": "$Hersteller",  
    "Max-Verbrauch":  
      { "$max": "$Spezifikation.Leistung" } }  
  }  
])
```

```
{ "_id" : "Toshiba", "Max-Verbrauch:" : 55 }  
{ "_id" : "Acer", "Max-Verbrauch:" : 35 }
```

Akkumulatoren - \$min und \$max

Das selbe Feld kann in mehreren Akkulatoren verwendet werden.

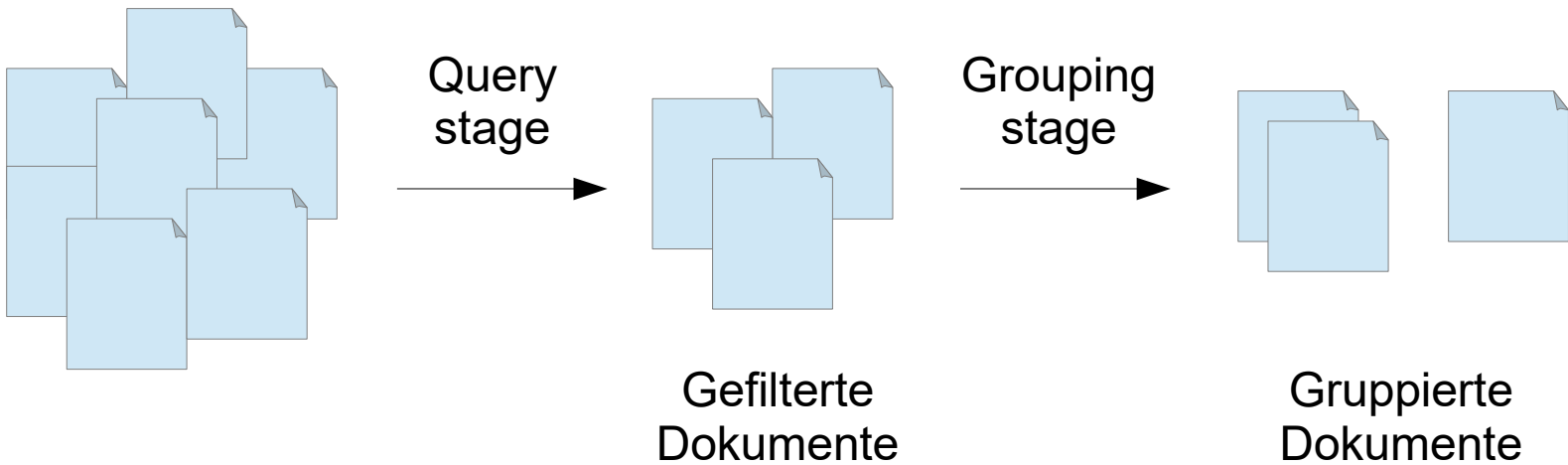
```
> db.artikel.aggregate([
  { "$group": { "_id": "$Hersteller",
    "Min-Verbrauch": { "$min": "$Spezifikation.Leistung" },
    "Max-Verbrauch": { "$max": "$Spezifikation.Leistung" } }
  }
])

{ "_id" : "Toshiba", "Min-Verbrauch:" : 55,
  "Max-Verbrauch:" : 55 }
{ "_id" : "Acer", "Min-Verbrauch:" : 25,
  "Max-Verbrauch:" : 35 }
```

Aggregation Pipeline

Die aggregate-Funktion arbeitet wie eine Pipeline, durch die die Daten durchlaufen.

```
db.collection.aggregate([stage, stage, stage])
```



\$match

`$match` arbeitet wie eine normale `find()`-Query. Es werden nur gewählte Dokumente an die nächste Stage weitergereicht.

```
> db.artikel.aggregate([  
  { "$match": { "Hersteller": "Acer" } }  
])
```

\$match/\$project/\$group/\$sort/\$limit

Die Filterung mittels \$match soll so früh wie möglich gesetzt werden, um die Datenmenge früh zu reduzieren. Weiters sollen so wenige Felder wie möglich weitergereicht werden, was mittels \$project möglich ist.

```
> db.artikel.aggregate([
  {"$match": {"Hersteller": "Acer"}},
  {"$project": {"_id": false, "Bezeichnung": true,
                "Preis": true}},
  {"$group": {"_id": "$Bezeichnung",
              "MinPreis": {"$min": "$Preis"}}},
  {"$sort": {"MinPreis": 1}},
  {"$limit": 3}
])
```