MongoDB-Grundlagen

Einsatzzwecke, Stärken und Unterschiede zu MySQL

MongoDB in Stichworten

Einige der folgenden Begriffe werden später weiter erläutert.

- Dokumentenorientiert
- Dynamische Schemata
- NoSQL-Datenbank
- In C++ geschrieben
- Verwendet JSON-Format
- JavaScript verwendbar
- Indizes auf Attribute setzbar
- Replikation für Hochverfügbarkeit
- “Sharding” für horizontale Skalierbarkeit
- Speicherung großer Dateien mit GridFS
- “Pipelines” für Aggregationen
- Unterstützt geografische Suche
- Hohe Performanz

MongoDB vs. RDBMS

MongoDB	**RDBMS**
Database	Database
Collection	Table
Document	Row
Field	Column
Embedded/Linked documents	Table join
Primary key	Primary key

Unterstützte Datentypen

MongoDB unterstützt unter anderem folgenden Datentypen:

- String
- Integer
- Boolean
- Min/Max-Schlüssel
- Array
- Timestamp
- Object
- Null
- Symbol
- Date
- Object ID
- Binary
- Code
- Regular expressions

Dokumentenorientiert

MongoDB

```
{
  name: ‘John’,
  age: 21,
  jobs: [
    ‘Dancer’,
    ‘Jumper’,
    ‘Sitter’
  ]
}
```

MySQL

```
ID   Name      Age
1    John      21

ID   Jobs
1    Dancer
2    Jumper
3    Sitter

Person    Job
1         1
1         2
1         3
```

Dynamische Schemata

Erste Einfügung

```
{
  name: ‘John’,
  age: 21,
  jobs: [
    ‘Dancer’,
    ‘Jumper’,
    ‘Sitter’
  ]
}
```

Zweite Einfügung

```
{
  name: Jenny’,
  age: 36,
  jobs: [
    ‘Spinner’
  ],
  title: ‘Doctor’
}
```

Vorteile dynamische Schemata

Keine Schemata einhalten zu müssen können einige Vorteile ergeben.

- Weniger Datenbankmigrationen
- Keine Ausfallzeit
- Weniger Gefahr des Datenverlusts
- Kürzere Entwicklungszeit
- Keine leeren oder “NULL”-Werte

Beziehungen in MongoDB

Bei der Planung von Beziehungen gilt es im Gegensatz zu MySQL zusätzliche Überlegungen vorzunehmen.

- 1-zu-wenigen
- 1-zu-vielen
- 1-zu-Übermengen
- n-zu-m

1-zu-n-Beziehung

1-zu-wenigen (“embedding”)
Eine Abfrage

Person

```
{
  name: 'Mike’,
  pets: [
    'cat’,
    'dog’
  ]
}
```

1-zu-n-Beziehung

1-zu-vielen (“referencing”)
Zwei Abfragen

Husband

```
{
  name: ‘Peter’,
  wives: [
    ObjectID(‘1‘),
    ObjectID(‘2‘),
    ObjectID(‘3‘)
  ]
}
```

Wife

```
{
  _id: ‘1’,
  name: ‘Jenny’,
  gender: ‘Female’
}
```

1-zu-n-Beziehung

1-zu-Übermengen (“referencing”)
Zwei Abfragen

Pc

```
{
  _id: ‘it-pc-1’,
  ip: ‘127.66.66.66’
}
```

Log

```
{
  time: (‘09:42:41.382’),
  message: ‘Error while booting’,
  pc: ObjectID(‘it-pc-1’)
}
```

n-zu-m-Beziehung

“referencing”

Product

```
{
  name: ‘Car’,
  parts: [
    ObjectID(‘1’),
    ObjectID(‘2’),
    ObjectID(‘3’)
  ]
}
```

Part

```
{
  _id: 1,
  name: ‘Wheel’
}
```

Abfragesprache

Abfragen werden wie “chained methods” in JavaScript gehandhabt. Die meistens konventionellen JavaScript-Funktionen sind auch innerhalb der Abfrage verfügbar.

Beispiel

```
$ mongo < script.js
```

Einfügen

MongoDB

Eine

```
db.person.insert({
  name: ‘Mike’,
  job: ‘Dancer’
})
```

Mehrere

```
db.person.insert({
  name: ‘Kevin’,
  job: ‘Smoker’
}, {
  name: ‘John’,
  job: ‘Fighter’
})
```

MySQL

Eine

```
INSERT INTO person
VALUES (‘Mike’, ‘Dancer’);
```

Mehrere

```
INSERT INTO person
VALUES (‘Mike’), (‘John’);
```

Auswählen

MongoDB

Alle
```
db.person.find()
```

Alle mit Limit
```
db.person.find().limit(10)
```

Eine
```
db.person.findOne({
  name: ‘Peter’
})
```

MySQL

Alle
```
SELECT * FROM person;
```

Alle mit Limit
```
SELECT * FROM person LIMIT 10;
```

Eine
```
SELECT * FROM person
WHERE name = Peter;
```

Auswählen (Spezialisierung)

MongoDB

Und-Operator

```
db.person.findOne({
  $and: [
    { name: ‘Peter’ },
    { job: ‘Murder’ }
  ]
})
```

MySQL

Und-Operator

```
SELECT * FROM person
WHERE name = Peter
AND job = Murder;
```

Entfernen

MongoDB

Eine

```
db.person.remove({
  name: ‘Peter’
})
```

Alle

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

MySQL

Eine

```
DELETE FROM person
WHERE name = Peter;
```

Alle

```
DELETE FROM person;
```

Validierung

Ähnlich wie bei MySQL lassen sich Dokumente validieren. Anders als bei MySQL können komplexere Regeln definiert werden.

Syntax

Vordefinierte Validatoren

Kollektion

```
db.createCollection(‘contacts’, {
  validator: { $or:
    [
      { phone: { $type: ‘string’ } },
      { email: { $regex: /@web*/ } },
      { status: { $in: [1, 2, 3] } }
    ]
  }
})
```

Replikation und “Sharding”

Während Replikation Redundanz und höhere Datenverfügbarkeit gewährleistet sorgt “Sharding” für eine einfache horizontale Skalierung.

Vorteile Replikation

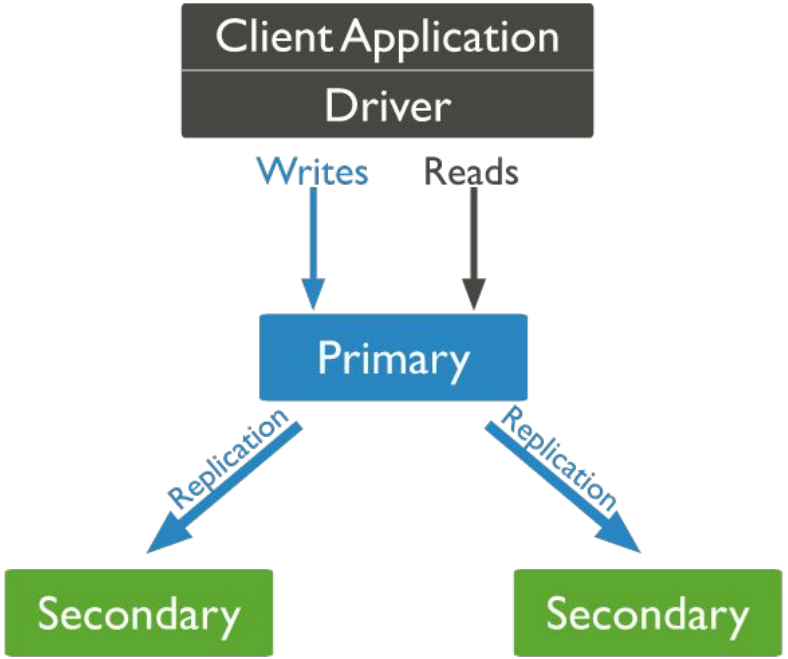
- Höhere Datensicherheit
- 24 Stunden Verfügbarkeit der Daten
- Disasterwiederherstellung
- Keine Ausfallzeit bei Wartungsarbeiten
- Skaliert beim Lesen

Vorteile “Sharding”

- Aufteilung großer Datensätze
- Paralleles Lesen
- Vermeidet vertikales Skalieren

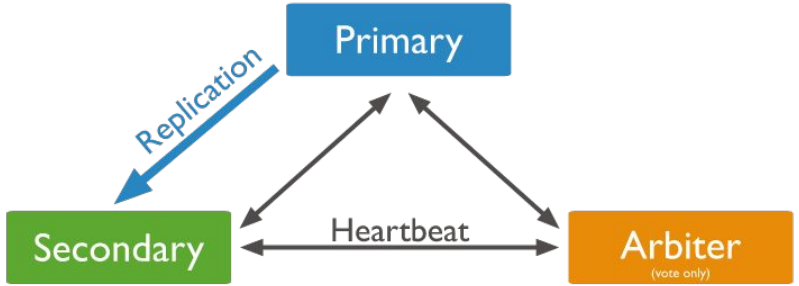
Replikation

“Primaries” und “Secondories”



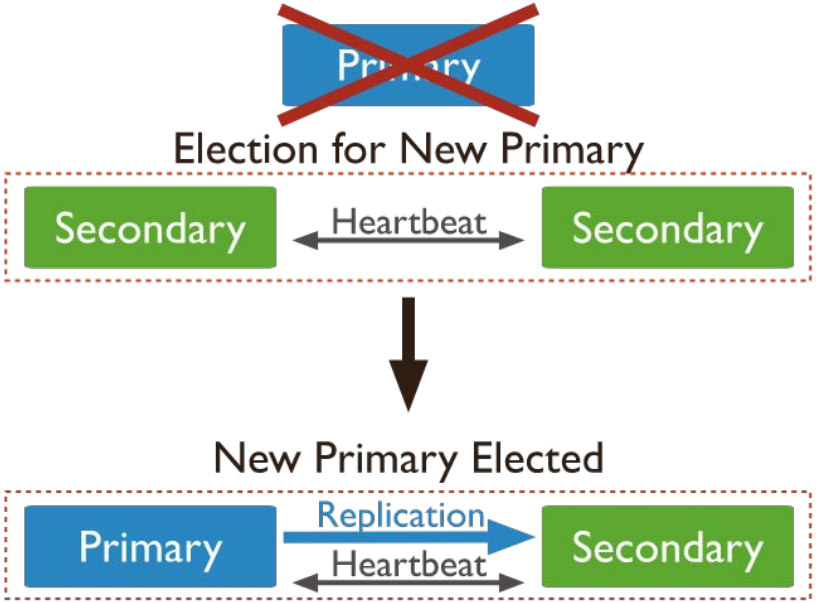
Replikation

“Heartbeat”



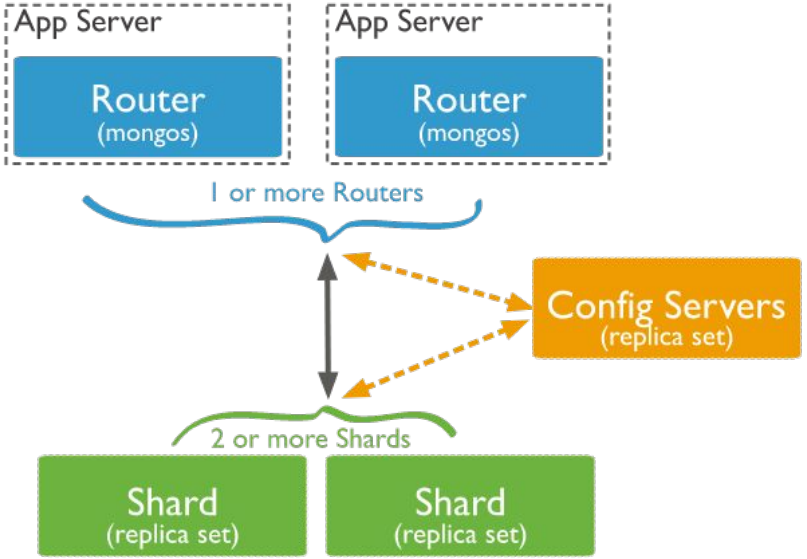
Replikation

“Election”



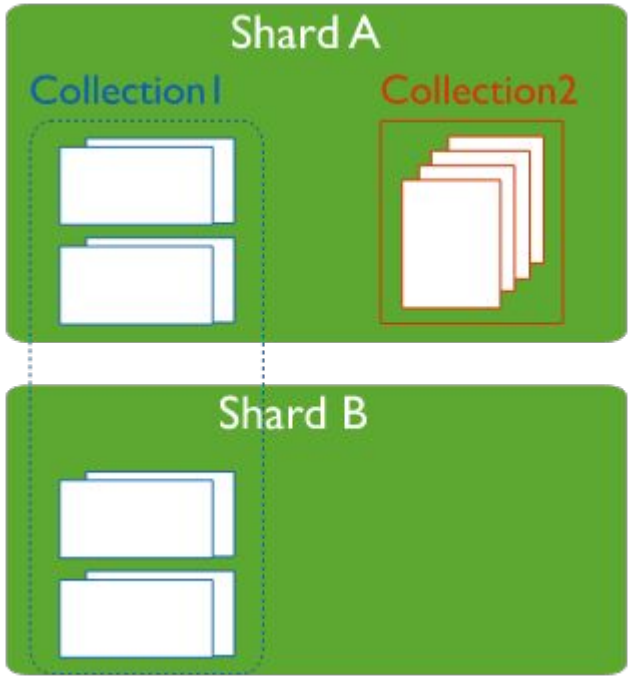
“Sharding”

“Sharded Cluster”



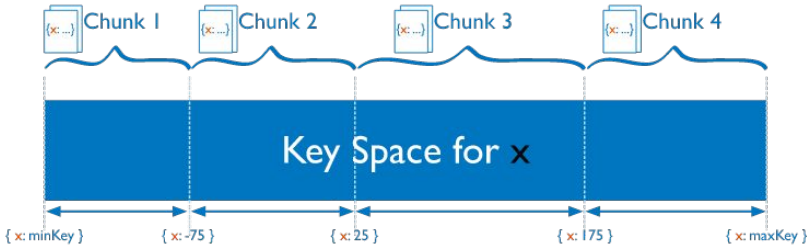
“Sharding”

“Sharded collection”



“Sharding”

“Ranged sharding”



Aggregation

Anders als bei MySQL werden Aggregationen in MongoDB durch “Pipelines” realisiert. Jede “Pipeline” verändert/selektiert die Daten auf eine bestimmte Weise.

Aggregation (einfach)

MongoDB

```
db.orders.aggregate([
  {
    $group: {
      _id: null,
      total: { $sum: ‘$price’ }
    }
  }
])
```

MySQL

```
SELECT SUM(price) AS total
FROM orders;
```

Aggregation (komplex)

MongoDB

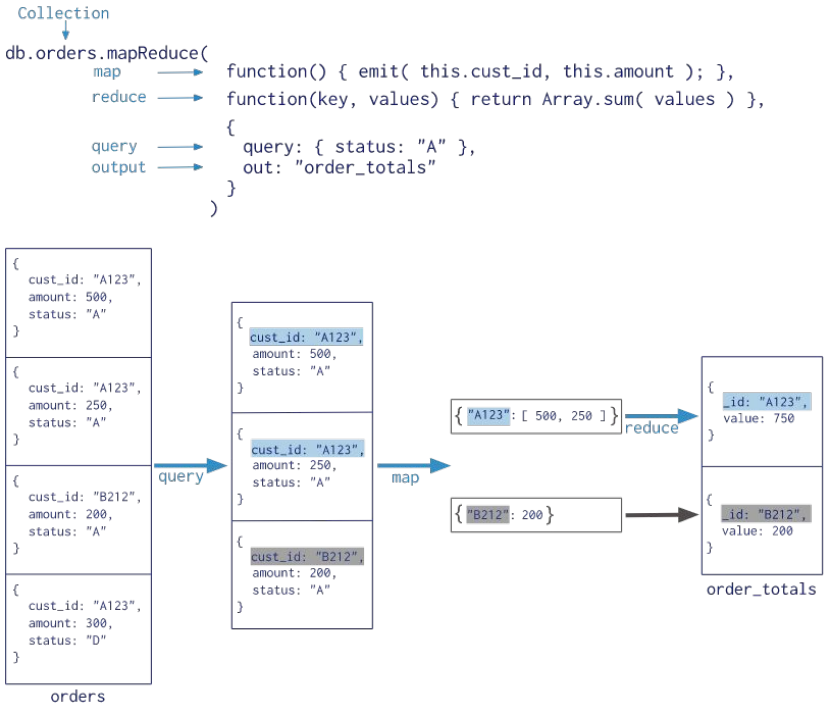
```
db.orders.aggregate([
  {$match: { status: 'open' }},
  {
    $group: {
      _id: "$cust_id",
      total: { $sum: "$price" }
    }
  },
  {$match: { total: { $gt: 250 } }}
])
```

MySQL

```
SELECT cust_id,SUM(price) as total
FROM orders
WHERE status = 'A'
GROUP BY cust_id
HAVING total > 250;
```

Aggregation

“Map-Reduce”



Indizes

Im Unterschied zu MySQL lassen sich Indizes nicht nur auf Dokumenten- sondern auch auf Schlüsselebene definieren. Die Unterschiede bezüglich der Performanz sind enorm!

Indizes

Indizes können auch aus zwei Feldern bestehen

Kollektion

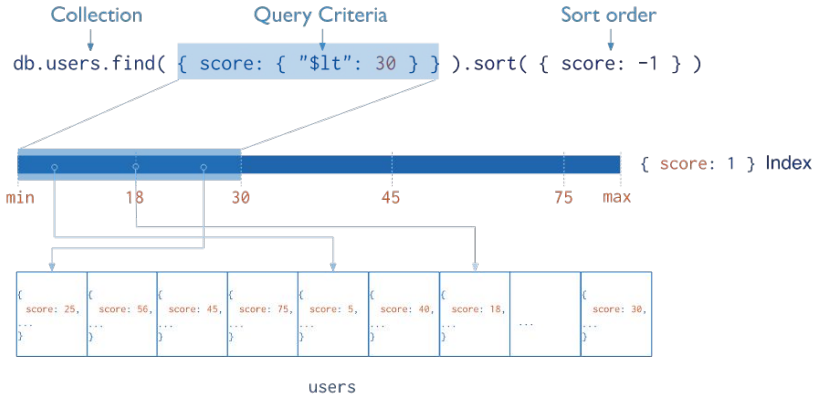
```
{
  name: ‘Mike’,
  age: 23,
  gender: ‘Male’
}
```

Index setzen

```
db.person.ensureIndex({age: 1})
```

Indizes

Indizes enthalten kleine Portionen der Daten



Raumbezogene Suche (Geospatial)

In MongoDB lassen sich raumbezogene Suchen realisieren. MongoDB kann Längen- und Breitengrade interpretieren und beispielsweise Umkreissuchen durchführen.

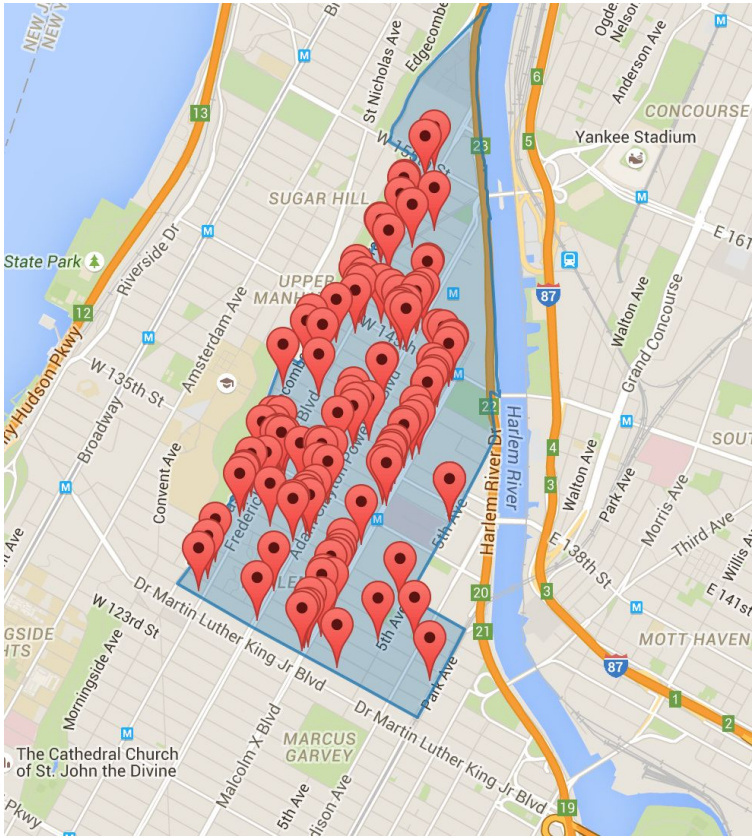
In der Nähe-Suche

Neben Punkten können auch Sphären angegeben werden

```
db.places.find({
  location: {
    $nearSphere: {
      $geometry: {
        type: "Point",
        coordinates: [-73.4, 40.1]
      },
      $minDistance: 1000,
      $maxDistance: 5000
    }
  }
})
```

In der Nähe-Suche

Außer Punkt möglich: “LineString”, “Polygon”, “Multipoint”, “MultiLineString”, “Multipolygon”, “GeometryCollection”



Einsatzzwecke

MongoDB ist nicht immer relationalen Datenbanken vorzuziehen. Hier einige Punkte, bei denen MongoDB die bessere Wahl ist.

- Hohes Besucheraufkommen
- Hohe Verfügbarkeit von Daten
- Georedundanz
- Skalierung
- Große Datensätze ab einem Gigabyte
- “Single-Page-Applications”
- Keine festen Schemata
- Echtzeitanalysen