MongoDB

dokumentbasierte, NoSQL Datenbank

Einführung

MongoDB ist eine beliebte NoSQL-Datenbank, die Daten in einem flexiblen, JSON-ähnlichen Format namens BSON speichert. Sie ist bekannt für ihre Skalierbarkeit und Benutzerfreundlichkeit.

BSON (Binary JSON) ist ein offener Standard für die Codierung oder die Darstellung von JSON-ähnlichen Dokumenten in einem Binärformat. Während JSON in erster Linie für die Übertragung von Daten als Austauschformat gedacht ist, ist BSON für die Speicherung von Daten in binärer Form gedacht.

Unterschiede von MongoDB zu RDBMS

Während relationale Datenbanken auf einem tabellenbasierten Datenmodell mit vordefinierten Spalten und Beziehungen aufbauen, werden in MongoDB Daten in Dokumenten (BSON) abgelegt, die unterschiedliche Felder und Strukturen haben können, was hohe Flexibilität bei der Datenspeicherung ermöglicht.

Beispiel eines Nutzers im BSON-Format:

```
{
 "name": "Alice",
  "email": "alice@example.com",
  "age": 28
}
```

BSON unterstützt im Gegensatz zu JSON weitere Datentypen wie Datumsobjekte oder Binäardaten. So lassen sich zum Beispiel Machine-Learning-Modelle mit BSON speichern.

Schemafreiheit

MongoDB ist im Gegensatz zu relationalen Datenbanken schemafrei. Es lassen sich also pro Dokument neue Felder hinzufügen, die in anderen Dokumenten nicht vorhanden sind. Optional lassen sich aber Schema-Validation-Regeln einführen.

Personen-Dokument	anderes Personen-Dokument

{
"name": "Alice",
"email": "alice@example.com",
"age": 28
}
	{
	"name": "Bob",
	"email": "bob@example.com",
	"age": 89,
	"hobbies": ["Swimming", "Dancing"]
	}

Datenbanken und Collections

Datenbank (Database): Eine Datenbank in MongoDB ist eine logische Containerstruktur, die verschiedene Collections und deren zugehörige Dokumente enthält.

Collection: Eine Collection in MongoDB ist eine Gruppe von Dokumenten, die in der gleichen logischen Containerstruktur innerhalb einer Datenbank gespeichert werden. Collections entsprechen in etwa Tabellen in relationalen Datenbanken. In einer Collection werden Dokumente in BSON (Binary JSON) gespeichert. So werden zum Beispiel alle User in einer Collenction namens "user" verwaltet.

Erste Schritte in der MongoDB Shell

Datenbank erstellen

use heroDatabase

Die Datenbank heroDatabase muss noch nicht exisiteren, um sie nutzen zu können! Die neue Datenbank wird erst angelegt, wenn eine Sammlung angelegt wird.

vorhandene Datenbanken anzeigen

show dbs

aktuell ausgewählte Datenbank anzeigen

db

aktuell ausgewählte Datenbank löschen

db.dropDatabase()

Eine neue Sammlung anlegen

use heroDatabase

db ist das Datenbank-Objekt auf die ausgewählte Datenbank. Mit createCollection kann eine neue Sammlung angelegt werden.

```
db.createCollection("Heroes")
```

Es empfiehlt sich, den Namen der Sammlung ohne Leerzeichen zu schreiben, obwohl das möglich wäre.
Außerdem sollten keine speziellen Zeichen wie Punkt oder Dollarzeichen im Namen angegeben werden.

Alle Collections der aktuellen Datenbank anzeigen:

```
show collections
```

Items in die Collection eintragen

Mit insertOne lässt sich ein neues Item-Objekt in die Collection der aktuellen Datenbank eintragen. Das Objekt entspricht einer JSON-Datenstruktur.

```
db.heroes.insertOne({
  name: "Superman",
  alias: "Clark Kent",
  superpowers: ["Fliegen", "Superstärke", "Hitzestrahl"],
  city: "Metropolis"
})
```

Es wurde jetzt ein Heroes-Dokument mit einer internen _id erstellt. Um alle Dokumente einer Collection anzuzeigen, gibt es den Befehl:

```
db.heroes.find()
```

wichtige Datentypen von Items

Datentyp	Anmerkung
String	
Integer	
Boolean	
Double	
Object ID	Dokument ID
Arrays	
Binary Data	
Null	
Code	Java Script Code
Date	Datumsobjekt

insertMany

Es lassen sich auch viele Objekte auf einen Schlag eintragen:

```
heroes = [
  {
    name: "Spider-Man",
    alias: "Peter Parker",
    superpowers: ["Spinnennetz-Schwingen", "Wandschleichen"],
    city: "New York"
  },
  {
    name: "Wonder Woman",
    alias: "Diana Prince",
    superpowers: ["Superstärke", "Lasso der Wahrheit"],
    city: "Themyscira"
  }
]
db.Heroes.insertMany(heroes)
```

die wichtigsten Operatoren

Operatoren sind spezielle Symbole, die Compilern helfen, mathematische oder logische Aufgaben auszuführen. MongoDB bietet mehrere Arten von Operatoren, um mit der Datenbank zu interagieren.

Die wichtigsten Operatoren sind die boolschen Operatoren, arithmetischen Vergleichsoperatoren, Elementoperatoren, Array-Operatoren und Update-Operatoren. Es existieren noch spezielle Operatoren wie zum Beispiel geospatiale Operaten, Aggregate oder Projektionsoperatoren.

UpdateOne und UpdateMany

Mit updateOne und updateMany lassen sich Dokumente updaten. Die Filter-Bedingung gibt an, welche Dokumente editiert werden sollen. Mit dem $set Operator wird definiert, welchen Wert die Felder haben sollen.

UpdateOne

```
db.heroes.updateOne({alias: "Diana Prince"}, {$set: {points: 89, city: "Los Angeles"}})
```

entspricht SQL: Update Heroes set points=89, city="Los Angeles" where alias="Diana Prince"

UpdateMany

```
db.heroes.updateMany({}, {$set: {points: 40}})
```

entspricht SQL: Update Heroes set points=40

In der Collection suchen (selektieren)

Mit find lässt sich nach Attributen der Objekte suchen

```
db.Heroes.find({name: "Superman"})
db.Heroes.find({city: "New York"})
```

Mit Limit die Anzahl begrenzen:

```
db.Heroes.find({city: "New York"}).limit(5)
```

Ausgabe nach Feld aufsteigend sortieren

```
db.Heroes.find().sort({city: 1})
```

Absteigend sortiert wird mit -1

Elemente löschen

Mit updateOne und updateMany lassen sich Dokumente updaten. Die Filter-Bedingung gibt an, welche Dokumente editiert werden sollen. Mit dem $unset Operator wird definiert, welche Attribute gelöscht werden.

```
db.heroes.updateOne(
  { name: "Wonder Woman" },  // Filter
  { $unset: { powers: "" } } // Feld löschen
)
```

Elemente in einen Array speichern

Elemente in einem existierenden Array anzusprechen. Hier sind der $push, $pop und $pull - Operator zu sehen.

Einen neuen Eintrag zum superpowers-Array hinzufügen

```
db.Heroes.updateOne({name: "Wonder Woman"}, {$push: {superpowers: "Night Vision"}})
```

Duplikate verhindern:

```
db.Heroes.updateOne({name: "Wonder Woman"}, {$addToSet: {superpowers: "Night Vision"}})
```

Das letzte Element des Superpowers-Arrays löschen

```
db.Heroes.updateOne({name: "Wonder Woman"}, {$pop: {superpowers: 1}})
```

Das erste Element des Superpowers-Arrays löschen

```
db.Heroes.updateOne({name: "Wonder Woman"}, {$pop: {superpowers: -1}})
```

Ein spezifisches Element aus dem Superpowers Array löschen

```
db.Heroes.updateOne({name: "Superman"}, {$pull: {superpowers:"Hitzestrahl"}})
```

In einem Array suchen

Einträge suchen, die ein bestimmtes Element im Array haben. In dem Beispiel selektieren wir alle Heroes, die als Superpower eine Night Vision haben.

$elemMatch prüft, ob mindestens ein Element im Array die Bedingung erfüllt.

$eq prüft auf Gleichheit. Alternativen wären z.b. $gt, $gte, $lt, $lte

```
db.heroes.find(
        {superpowers: {$elemMatch: {$eq: "Night Vision"}}}
)
```

arrayFilter

Mit arrayFilters kann man granular steuern, welche Elemete in einem Array upgedated werden sollen. Mit dem $set - Operator lässt sich spezifizieren, was upgedated werden soll.

```
db.Heroes.updateOne(
  { name: "Superman" },
  {
    $set: { "superpowers.$[element]": "Flying" } // $set operator with the positional operator $
  },
  {
    arrayFilters: [{ "element": "Fliegen" }] // Specify the condition to match the element in the array
  }
)
```

Ein Dokument aus einer Collection löschen

Mit deleteOne lässt sich genau ein Dokument löschen. Bei mehrmaligem Vorkommen wird nur eins gelöscht.

```
db.Heroes.deleteOne({name:"Superman"})
```

mit deleteMany lassen sich auch mehrere Dokumente löschen

```
db.Heroes.deleteMany({city: "Metropolis"})
db.Heroes.deleteMany({city: "New York", points: {$lt: 30}})
db.Heroes.deleteMany({$or: [{ points: { $gt: 50 } },{ city: "New York" } ] })
db.Heroes.deleteMany({})
```

Daten werden bei beiden Befehlen dauerhaft aus der Collection entfernt.

Query Dokuments

In Collections lässt sich ähnlich wie mit SQL nach Items filtern.

Suche nach allen Vorkommen von "om" im Namen (regular expression)

```
db.Heroes.find({name: /om/i})
```

oder mit

```
db.Heroes.find({name: {$regex: ".om."}})
```

Alle Heroes mit Punkten > 50

```
db.Heroes.find({ points: { $gt: 50 } })
```

Alle Heroes mit Punkten > 50 oder aus New York

```
db.Heroes.find({$or: [{ points: { $gt: 50 } },{ city: "New York" }]})
```

_id

In MongoDB wird die eindeutige Kennzeichnung eines Dokuments durch ein spezielles Feld mit dem Namen _id gewährleistet. Dieses Feld ist vom Typ ObjectId und wird automatisch bei der Erstellung eines Dokuments generiert. Um nach einem Dokument anhand seiner ObjectId zu suchen. ObjectId ist eine 12-Byte-Zahl, die normalerweise aus einer Kombination von Informationen wie einem Zeitstempel, einer Maschinen-ID, einer Prozess-ID und einem Zähler besteht. Dies gewährleistet die Eindeutigkeit der ObjectId innerhalb einer MongoDB-Instanz.

```
var objectIdToSearch = ObjectId("6540df2628ada77d00fe611c")
db.Heroes.find({ _id: objectIdToSearch })
```

Eigene _id anlegen

Man kann die _id auch selber anlegen, muss dann allerdings selbst dafür sorgen tragen, dass die ID eindeutig innerhalb der Collection ist.

```
db.Heroes.insertOne({_id: 2, name:"Bob"})
```

Nun kann man sich die Konvertierung in eine ObjectID sparen und direkt nach dem Integer 2 suchen:

```
db.Heroes.find({_id:2})
```

Eine Collection löschen

Eine Collection lässt sich mit drop löschen

db.Heroes.drop()

Dieser Vorgang kann nicht rückgängig gemacht werden, die Collection ist unwiderruflich verloren. Es ist überdies selten nötig, eine Collection zu löschen.

Beispiel einer Transaktion mit pymongo

```
def perform_transaction():
    session = client.start_session()

    try:
        with session.start_transaction():
            db = session.client.mydatabase  # Ihre Datenbank auswählen
            collection = db.mycollection
            collection.insert_one({"name": "John", "age": 30}, session=session)
            collection.update_one({"name": "John"}, {"$set": {"age": 31}}, session=session)

    except Exception as e:
        print("Fehler in der Transaktion:", e)
        session.abort_transaction()
    else:
        session.commit_transaction()
        print("Transaktion erfolgreich abgeschlossen.")
    finally:
        session.end_session()
```