Kostenloses eBook

LERNEN MongoDB

Free unaffiliated eBook created from
Stack Overflow contributors.

#mongodb

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: mongodb

It is an unofficial and free MongoDB ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official MongoDB.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit MongoDB

Bemerkungen

- Die Daten in der Welt begannen enorm zu wachsen, nachdem mobile Anwendungen auf den Markt kamen. Diese riesige Datenmenge wurde mit der traditionellen relationalen Datenbank - SQL - fast unmöglich verarbeitet. NoSQL-Datenbanken werden eingeführt, um diese Daten zu verarbeiten, bei denen die Flexibilität einer variablen Spaltenanzahl für die einzelnen Daten sehr viel größer war.
- MongoDB ist eine der führenden NoSQL-Datenbanken. Jede Sammlung enthält eine Reihe von JSON-Dokumenten. Jedes Datenmodell, das in einem JSON-Dokument ausgedrückt werden kann, kann problemlos in MongoDB gespeichert werden.
- MongoDB ist eine Server-Client-Datenbank. Der Server läuft normalerweise mit der Binärdatei `mongod` und der Client mit `mongo` .
- In MongoDB gibt es aus verschiedenen philosophischen und pragmatischen Gründen keine Join-Operation vor v.3.2. Die Mongo-Shell unterstützt jedoch Javascript. Wenn also $ Lookup nicht verfügbar ist, können Sie Verknüpfungsoperationen für Dokumente in Javascript simulieren, bevor Sie sie einfügen.
- Um eine Instanz in der Produktionsumgebung auszuführen, wird dringend empfohlen, die Operations-Checkliste zu befolgen.

Versionen

Ausführung	Veröffentlichungsdatum
3.4	2016-11-29
3.2	2015-12-08
3,0	2015-03-03
2.6	2014-04-08
2.4	2013-03-19
2.2	2012-08-29
2,0	2011-09-12
1.8	2011-03-16
1.6	2010-08-31
1.4	2010-03-25
1.2	2009-12-10

Examples

Installation

Führen Sie die folgenden Schritte aus, um MongoDB zu installieren:

- **Für Mac OS:**
  - Für Mac OS gibt es zwei Optionen: manuelle Installation oder Homebrew .
  - **Installation mit Homebrew :**
    - Geben Sie den folgenden Befehl in das Terminal ein:

      ```
      $ brew install mongodb
      ```

  - **Manuell installieren:**
    - Laden Sie die neueste Version hier herunter. Stellen Sie sicher, dass Sie die entsprechende Datei herunterladen. Überprüfen Sie insbesondere, ob Ihr Betriebssystem 32-Bit oder 64-Bit ist. Die heruntergeladene Datei hat das Format `tgz` .
    - Wechseln Sie in das Verzeichnis, in das diese Datei heruntergeladen wird. Geben Sie dann den folgenden Befehl ein:

      ```
      $ tar xvf mongodb-osx-xyz.tgz
      ```

      Anstelle von `xyz` gäbe es Informationen zu Version und Systemtyp. Der extrahierte Ordner hätte denselben Namen wie die `tgz` Datei. Innerhalb des Ordners befindet sich ein Unterordner mit dem Namen `bin` der mehrere Binärdateien sowie `mongod` und `mongo` .
    - Standardmäßig speichert der Server die Daten im Ordner `/data/db` . Also müssen wir dieses Verzeichnis erstellen und dann den Server mit den folgenden Befehlen ausführen:

      ```
      $ sudo bash
      # mkdir -p /data/db
      # chmod 777 /data
      # chmod 777 /data/db
      # exit
      ```

    - Um den Server zu starten, sollte der folgende Befehl vom aktuellen Speicherort aus gegeben werden:

      ```
      $ ./mongod
      ```

      Der Server wird standardmäßig am Port 27017 gestartet.

- Um den Client zu starten, muss ein neues Terminal mit demselben Verzeichnis wie zuvor geöffnet werden. Dann würde der folgende Befehl den Client starten und eine Verbindung zum Server herstellen.

  ```
  $ ./mongo
  ```

  Standardmäßig stellt es eine Verbindung zur `test` . Wenn Sie sehen, dass sich die Leitung wie `connecting to: test` . Dann haben Sie MongoDB erfolgreich installiert. Glückwunsch! Jetzt können Sie Hello World testen, um sicherer zu sein.

- **Für Windows:**

  - Laden Sie die neueste Version hier herunter. Stellen Sie sicher, dass Sie die entsprechende Datei herunterladen. Überprüfen Sie insbesondere, ob Ihr Betriebssystem 32-Bit oder 64-Bit ist.
  - Die heruntergeladene Binärdatei hat die Erweiterung `exe` . Starte es. Es wird ein Installationsassistent angezeigt.
  - Klicken Sie auf **Weiter** .
  - **Akzeptieren Sie** die Lizenzvereinbarung und klicken Sie auf **Weiter** .
  - Wählen Sie **Complete** Installation aus.
  - Klicken Sie auf **Installieren** . Möglicherweise werden Sie in einem Fenster aufgefordert, die Erlaubnis des Administrators einzuholen. Klicken Sie auf **Ja** .
  - Nach der Installation klicken Sie auf **Fertig stellen** .
  - Das mongodb wird jetzt unter dem Pfad `C:/Program Files/MongoDB/Server/3.2/bin` installiert. Anstelle von Version 3.2 könnte es auch eine andere Version für Ihren Fall geben. Der Pfadname würde entsprechend geändert.
  - `bin` Verzeichnis enthält mehrere Binärdateien sowie `mongod` und `mongo` . Um es aus einem anderen Ordner auszuführen, können Sie den Pfad im Systempfad hinzufügen. Es zu tun:
    - Klicken Sie mit der rechten Maustaste auf **Arbeitsplatz,** und wählen Sie **Eigenschaften aus** .
    - Klicken Sie im linken Bereich auf **Erweiterte Systemeinstellungen** .
    - Klicken Sie auf der Registerkarte **Erweitert** auf **Umgebungsvariablen** .
    - Wählen Sie den **Pfad** aus Abschnitt **Systemvariablen** und klicken Sie auf **Bearbeiten ....**
    - Hängen Sie vor Windows 10 ein Semikolon an und fügen Sie den oben angegebenen Pfad ein. In Windows 10 gibt es die Schaltfläche **Neu** , um einen neuen Pfad hinzuzufügen.
    - Klicken Sie auf **OK** , um die Änderungen zu speichern.

- Erstellen Sie nun einen Ordner namens `data` mit einem Unterordner namens `db` in dem Sie den Server ausführen möchten.
- Starten Sie die Eingabeaufforderung über ihre. Ändern Sie entweder den Pfad in cmd oder klicken Sie auf das **Befehlsfenster Öffnen,** das sichtbar **ist** , wenn Sie mit der rechten Maustaste auf den leeren Bereich der Ordner-Benutzeroberfläche klicken, indem Sie die Umschalt- und die Strg-Taste gleichzeitig drücken.
- Schreiben Sie den Befehl zum Starten des Servers:

```
> mongod
```

  Der Server wird standardmäßig am Port 27017 gestartet.

- Öffnen Sie eine andere Eingabeaufforderung, und geben Sie Folgendes ein, um den Client zu starten:

```
> mongo
```

- Standardmäßig stellt es eine Verbindung zur `test` . Wenn Sie sehen, dass sich die Leitung wie `connecting to: test` . Dann haben Sie MongoDB erfolgreich installiert. Glückwunsch! Jetzt können Sie Hello World testen, um sicherer zu sein.

- **Für Linux:** Fast gleich wie Mac OS, nur dass ein entsprechender Befehl benötigt wird.
  - Für Debian-basierte Distributionen (mit `apt-get`):
    - Importieren Sie den MongoDB-Repository-Schlüssel.

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
gpg: Total number processed: 1\
gpg:               imported: 1  (RSA: 1)
```

    - Repository zur Paketliste auf **Ubuntu 16.04 hinzufügen** .

```
$ echo "deb http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.2
multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list
```

    - auf **Ubuntu 14.04** .

```
$ echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.2
multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list
```

    - Paketliste aktualisieren

```
$ sudo apt-get update
```

    - Installieren Sie MongoDB.

```
$ sudo apt-get install mongodb-org
```

- Für Red Hat basierte Distributionen (mit `yum`):
  - Verwenden Sie einen Texteditor, den Sie bevorzugen.

    $ vi /etc/yum.repos.d/mongodb-org-3.4.repo
  - Fügen Sie folgenden Text ein.

```
[mongodb-org-3.4]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-
org/3.4/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-3.4.asc
```

  - Paketliste aktualisieren

```
$ sudo yum update
```

  - Installieren Sie MongoDB

```
$ sudo yum install mongodb-org
```

Hallo Welt

Nach dem Installationsvorgang sollten die folgenden Zeilen in der Mongo-Shell (Client-Terminal) eingegeben werden.

```
> db.world.insert({ "speech" : "Hello World!" });
> cur = db.world.find();x=cur.next();print(x["speech"]);
```

Hallo Welt!

Erläuterung:

- In der ersten Zeile haben wir eingeführt , um eine `{ key : value }` gepaart Dokument in der Standarddatenbank `test` und in der Sammlung namens `world` .
- In der zweiten Zeile rufen wir die Daten ab, die wir gerade eingefügt haben. Die abgerufenen Daten werden in einer Javascript-Variablen namens `cur` gespeichert. Dann haben wir mit der `next()` Funktion das erste und einzige Dokument abgerufen und in einer anderen js-Variablen namens `x` gespeichert. Dann wurde der Wert des Dokuments gedruckt, das den Schlüssel enthält.

Ergänzende Bedingungen

SQL-Begriffe	MongoDB-Begriffe
Datenbank	Datenbank
Tabelle	Sammlung
Einheit / Zeile	Dokumentieren
Säule	Schlüssel / Feld
Tabelle beitreten	Eingebettete Dokumente
Primärschlüssel	Primärschlüssel (Standardschlüssel `_id` bereitgestellt von mongodb

Ausführung einer JavaScript-Datei in MongoDB

```
./mongo localhost:27017/mydb myjsfile.js
```

Erläuterung: Diese Operation führt das Skript `myjsfile.js` in einer `mongo` Shell aus, die eine Verbindung zur `mydb` Datenbank der `mongod` Instanz herstellt, auf die über die Schnittstelle `localhost` an Port `27017` . `localhost:27017` ist nicht obligatorisch, da dies der Standardport ist, den `mongodb` verwendet.

Sie können auch eine `.js` Datei in der `mongo` Konsole `.js` .

```
>load("myjsfile.js")
```

Die Ausgabe von find in der Shell lesbar machen

Wir fügen unserem Erfassungstest drei Datensätze hinzu als:

```
> db.test.insert({"key":"value1","key2":"Val2","key3":"val3"})
WriteResult({ "nInserted" : 1 })
> db.test.insert({"key":"value2","key2":"Val21","key3":"val31"})
WriteResult({ "nInserted" : 1 })
> db.test.insert({"key":"value3","key2":"Val22","key3":"val33"})
WriteResult({ "nInserted" : 1 })
```

Wenn wir sie über find sehen, werden sie sehr hässlich aussehen.

```
> db.test.find()
{ "_id" : ObjectId("5790c5cecae25b3d38c3c7ae"), "key" : "value1", "key2" : "Val2
", "key3" : "val3" }
{ "_id" : ObjectId("5790c5d9cae25b3d38c3c7af"), "key" : "value2", "key2" : "Val2
1", "key3" : "val31" }
{ "_id" : ObjectId("5790c5e9cae25b3d38c3c7b0"), "key" : "value3", "key2" : "Val2
2", "key3" : "val33" }
```

Um dies zu umgehen und lesbar zu machen, verwenden Sie die Funktion **pretty** ().

```
> db.test.find().pretty()
{
        "_id" : ObjectId("5790c5cecae25b3d38c3c7ae"),
        "key" : "value1",
        "key2" : "Val2",
        "key3" : "val3"
}
{
        "_id" : ObjectId("5790c5d9cae25b3d38c3c7af"),
        "key" : "value2",
        "key2" : "Val21",
        "key3" : "val31"
}
{
        "_id" : ObjectId("5790c5e9cae25b3d38c3c7b0"),
        "key" : "value3",
        "key2" : "Val22",
        "key3" : "val33"
}
>
```

Grundlegende Befehle zur Mongo-Shell

Alle verfügbaren Datenbanken anzeigen:

```
show dbs;
```

Wählen Sie eine bestimmte Datenbank aus, auf die `mydb` , z. B. `mydb` . Dies erstellt `mydb` falls noch nicht vorhanden:

```
use mydb;
```

Alle Sammlungen in der Datenbank anzeigen (unbedingt zuerst auswählen, siehe oben):

```
show collections;
```

Alle Funktionen anzeigen, die mit der Datenbank verwendet werden können:

```
db.mydb.help();
```

Verwenden Sie zum Überprüfen der aktuell ausgewählten Datenbank den Befehl `db`

```
> db
mydb
```

`db.dropDatabase()` Befehl `db.dropDatabase()` wird zum `db.dropDatabase()` einer vorhandenen Datenbank verwendet.

```
db.dropDatabase()
```

Erste Schritte mit MongoDB online lesen: https://riptutorial.com/de/mongodb/topic/691/erste-

schritte-mit-mongodb

Kapitel 2: 2dsphere Index

Examples

Erstellen Sie einen 2dsphere-Index

`db.collection.createIndex()` Methode `db.collection.createIndex()` wird zum Erstellen eines `2dsphere` Indexes verwendet. Der Entwurf eines `2dsphere` Indexes:

```
db.collection.createIndex( { <location field> : "2dsphere" } )
```

Hier ist das `location field` der Schlüssel und `2dsphere` der Typ des Index. Im folgenden Beispiel erstellen wir einen `2dsphre` Index in der `places` Sammlung.

```
db.places.insert(
{
  loc : { type: "Point", coordinates: [ -73.97, 40.77 ] },
  name: "Central Park",
  category : "Parks"
})
```

Die folgende Operation wird erstellen `2dsphere` Index auf dem `loc` Bereich der `places` Sammlung.

```
db.places.createIndex( { loc : "2dsphere" } )
```

2dsphere Index online lesen: https://riptutorial.com/de/mongodb/topic/6632/2dsphere-index

Kapitel 3: Aktualisieren der MongoDB-Version

Einführung

So aktualisieren Sie die Version von MongoDB auf verschiedenen Plattformen und Versionen auf Ihrem Computer.

Bemerkungen

Wenn Sie eine ältere Version von MongoDB haben, müssen Sie den gesamten Pfad auf die neueste Version aktualisieren. Wenn Sie beispielsweise Version 3.0 ausführen und Version 3.4 abrufen möchten, müssen Sie ein Upgrade von 3.0 -> 3.2 -> 3.4 durchführen.

Examples

Upgrade auf Ubuntu 16.04 auf 3.4 mit apt

Sie müssen über 3.2 verfügen, um ein Upgrade auf 3.4 durchführen zu können. In diesem Beispiel wird davon `apt` dass Sie `apt` .

0. `sudo service mongod stop`
1. `sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 0C49F3730359A14518585931BC711F9BA15703C6`
2. `echo "deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list`
3. `sudo apt-get update`
4. `sudo apt-get upgrade`
5. `sudo service mongod start`

Stellen Sie sicher, dass die neue Version mit `mongo` . Die Shell gibt die MongoDB-Serverversion aus, die jetzt 3.4 sein sollte.

Aktualisieren der MongoDB-Version online lesen:
https://riptutorial.com/de/mongodb/topic/9851/aktualisieren-der-mongodb-version

Kapitel 4: Aktualisieren Sie die Operatoren

Syntax

- {$ set: {<Feld1>: <Wert1>, <Feld2>: <Wert2>, ...}}

Parameter

Parameter	Bedeutung
Feldname	Feld wird aktualisiert: { ***Name*** : 'Tom'}
targetVaule	Der Wert wird dem Feld zugewiesen: {name: ***'Tom'*** }

Bemerkungen

Referenz für $ set operator: $ set auf der offiziellen Website

Examples

$ set-Operator zum Aktualisieren von angegebenen Feldern in Dokumenten

I.Übersicht

Ein wesentlicher Unterschied zwischen MongoDB und RDBMS ist, dass MongoDB viele Arten von Operatoren hat. Einer von ihnen ist der Aktualisierungsoperator, der in Aktualisierungsanweisungen verwendet wird.

II. Was passiert, wenn wir keine Update-Operatoren verwenden?

Angenommen, wir haben eine **Studentensammlung** zum Speichern von Studentendaten (Tabellenansicht):

age	name	sex
20	Tom	M
25	Billy	M
18	Mary	F
40	Ken	M

Eines Tages bekommen Sie einen Job, der Toms Geschlecht von "M" in "F" ändern muss. Das ist einfach, richtig? Sie schreiben also die folgende Anweisung sehr schnell, basierend auf Ihrer RDBMS-Erfahrung:

```
db.student.update(
    {name: 'Tom'}, // query criteria
    {sex: 'F'} // update action
);
```

Mal sehen, was das Ergebnis ist:

age	name	sex
		F
25	Billy	M
18	Mary	F
40	Ken	M

Wir haben Toms Alter und Namen verloren! In diesem Beispiel können wir wissen, dass **das gesamte Dokument überschrieben wird,** wenn in der Aktualisierungsanweisung kein Aktualisierungsoperator vorhanden ist. Dies ist das Standardverhalten von MongoDB.

III. $ Set Operator

Wenn wir nur das "Geschlecht" -Feld in Toms Dokument ändern möchten, können Sie mit `$set` angeben, welche Felder Sie aktualisieren möchten:

```
db.student.update(
    {name: 'Tom'}, // query criteria
    {$set: {sex: 'F'}} // update action
);
```

Der Wert von `$set` ist ein Objekt, seine Felder stehen für die Felder, die Sie in den Dokumenten aktualisieren möchten, und die Werte dieser Felder sind die Zielwerte.

Das Ergebnis ist also jetzt korrekt:

age	name	sex
20	Tom	F
25	Billy	M
18	Mary	F
40	Ken	M

Wenn Sie "Geschlecht" und "Alter" gleichzeitig ändern möchten, können Sie sie an `$set` anhängen:

```
db.student.update(
    {name: 'Tom'}, // query criteria
    {$set: {sex: 'F', age: 40}} // update action
);
```

Aktualisieren Sie die Operatoren online lesen:
https://riptutorial.com/de/mongodb/topic/5880/aktualisieren-sie-die-operatoren

Kapitel 5: Anhäufung

Einführung

`Aggregations` verarbeiten Datensätze und geben berechnete Ergebnisse zurück. Aggregationsvorgänge gruppieren Werte aus mehreren Dokumenten zusammen und können verschiedene Vorgänge an den gruppierten Daten ausführen, um ein einzelnes Ergebnis zurückzugeben. MongoDB bietet drei Möglichkeiten für die Aggregation: die Aggregationspipeline, die Map-Reduction-Funktion und Einzelaggregationsmethoden.

Aus dem Mongo-Handbuch https://docs.mongodb.com/manual/aggregation/

Syntax

- **db.collection.aggregate (Pipeline, Optionen)**

Parameter

Parameter	Einzelheiten
Pipeline	Array (Eine Folge von Datenaggregationsvorgängen oder -stufen)
Optionen	Dokument (optional, nur verfügbar, wenn Pipeline als Array vorhanden ist)

Bemerkungen

Das Aggregations-Framework in MongoDB wird verwendet, um die allgemeine `GROUP BY` Funktionalität von SQL zu erreichen.

Berücksichtigen Sie für jedes Beispiel die folgenden Einfügungen in der Sammlung benannte `transactions` .

```
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 100, fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 10,  fee : 2});
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 4});
> db.transactions.insert({ cr_dr : "D", amount : 10,  fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 10,  fee : 4});
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 2});
```

Examples

Anzahl

Wie erhalten Sie die Anzahl der Debit- und Kredittransaktionen? Eine Möglichkeit, dies zu tun, ist die Verwendung der Funktion `count()` wie unten beschrieben.

```
> db.transactions.count({cr_dr : "D"});
```

oder

```
> db.transactions.find({cr_dr : "D"}).length();
```

Was aber, wenn Sie die möglichen Werte von `cr_dr` Voraus nicht kennen. Hier kommt das Aggregations-Framework zum Einsatz. Siehe unten stehende Aggregatabfrage.

```
> db.transactions.aggregate(
      [
          {
              $group : {
                  _id : '$cr_dr', // group by type of transaction
                // Add 1 for each document to the count for this type of transaction
                  count : {$sum : 1}
              }
          }
      ]
  );
```

Und das Ergebnis ist

```
{
    "_id" : "C",
    "count" : 3
}
{
    "_id" : "D",
    "count" : 5
}
```

Summe

Wie erhält man die Summe der `amount` ? Siehe unten stehende Aggregatabfrage.

```
> db.transactions.aggregate(
     [
         {
             $group : {
                 _id : '$cr_dr',
                 count : {$sum : 1},    //counts the number
                 totalAmount : {$sum : '$amount'}    //sums the amount
             }
         }
     ]
 );
```

Und das Ergebnis ist

```
{
    "_id" : "C",
    "count" : 3.0,
    "totalAmount" : 120.0
}
{
    "_id" : "D",
    "count" : 5.0,
    "totalAmount" : 410.0
}
```

Eine andere Version, die `amount` und `fee` summiert.

```
> db.transactions.aggregate(
      [
          {
              $group : {
                  _id : '$cr_dr',
                  count : {$sum : 1},
                  totalAmount : {$sum : { $sum : ['$amount', '$fee']}}
              }
          }
      ]
  );
```

Und das Ergebnis ist

```
{
    "_id" : "C",
    "count" : 3.0,
    "totalAmount" : 128.0
}
{
    "_id" : "D",
    "count" : 5.0,
    "totalAmount" : 422.0
}
```

Durchschnittlich

Wie erhalte ich den durchschnittlichen Betrag von Lastschriften und Gutschriften?

```
> db.transactions.aggregate(
    [
        {
            $group : {
                _id : '$cr_dr', // group by type of transaction (debit or credit)
                count : {$sum : 1},    // number of transaction for each type
                totalAmount : {$sum : { $sum : ['$amount', '$fee']}},    // sum
                averageAmount : {$avg : { $sum : ['$amount', '$fee']}}   // average
            }
        }
    ]
)
```

Das Ergebnis ist

```
{
    "_id" : "C", // Amounts for credit transactions
    "count" : 3.0,
    "totalAmount" : 128.0,
    "averageAmount" : 40.0
}
{
    "_id" : "D", // Amounts for debit transactions
    "count" : 5.0,
    "totalAmount" : 422.0,
    "averageAmount" : 82.0
}
```

Operationen mit Arrays.

Wenn Sie mit den Dateneinträgen in Arrays arbeiten möchten, müssen Sie das Array zunächst abwickeln . Beim Abwickelvorgang wird für jeden Eintrag im Array ein Dokument erstellt. Wenn Sie viele Dokumente mit großen Arrays haben, wird die Anzahl der Dokumente explodieren.

```
{ "_id" : 1, "item" : "myItem1", sizes: [ "S", "M", "L"] }
{ "_id" : 2, "item" : "myItem2", sizes: [ "XS", "M", "XL"] }

db.inventory.aggregate( [ { $unwind : "$sizes" }] )
```

Ein wichtiger Hinweis ist, dass ein Dokument, das das Array nicht enthält, verloren geht. Ab Mongo 3.2 wird eine Abrolloption "preserveNullAndEmptyArrays" hinzugefügt. Diese Option stellt sicher, dass das Dokument erhalten bleibt, wenn das Array fehlt.

```
{ "_id" : 1, "item" : "myItem1", sizes: [ "S", "M", "L"] }
{ "_id" : 2, "item" : "myItem2", sizes: [ "XS", "M", "XL"] }
{ "_id" : 3, "item" : "myItem3" }

db.inventory.aggregate( [ { $unwind : { path: "$sizes", includeArrayIndex: "arrayIndex" } }] )
```

Spiel

Wie schreibe ich eine Abfrage, um alle Abteilungen zu erhalten, in denen das Durchschnittsalter der Beschäftigten unter oder 70000 USD liegt oder über 35 liegt?

Um dies zu erreichen, müssen wir eine Anfrage schreiben, die den Mitarbeitern entspricht, deren Gehalt weniger als oder gleich 70000 USD beträgt. Fügen Sie dann die Aggregatstufe hinzu, um die Mitarbeiter nach Abteilungen zu gruppieren. Fügen Sie dann einen Akkumulator mit einem Feld mit dem Namen "average_age" hinzu, um das Durchschnittsalter pro Abteilung mithilfe des $ avg-Akkumulators zu ermitteln. Unterhalb der vorhandenen $ match- und $ group-Aggregate fügen Sie ein weiteres $ match-Aggregat hinzu, sodass wir nur Ergebnisse mit einem average_age-Wert abrufen größer als oder gleich 35.

```
db.employees.aggregate([
  {"$match": {"salary": {"$lte": 70000}}},
  {"$group": {"_id": "$dept",
              "average_age": {"$avg": "$age"}
             }
```

```
  },
  {"$match": {"average_age": {"$gte": 35}}}
])
```

Das Ergebnis ist:

```
{
  "_id": "IT",
  "average_age": 31
}
{
  "_id": "Customer Service",
  "average_age": 34.5
}
{
  "_id": "Finance",
  "average_age": 32.5
}
```

Dokumente entfernen, die ein doppeltes Feld in einer Sammlung enthalten (Deduplizierung)

Beachten Sie, dass die Option allowDiskUse: true optional ist, dass jedoch Probleme mit dem Arbeitsspeicher vermieden werden, da diese Aggregation bei großer Sammlungsgröße einen speicherintensiven Vorgang darstellen kann.

```
var duplicates = [];

db.transactions.aggregate([
 { $group: {
   _id: { cr_dr: "$cr_dr"},
   dups: { "$addToSet": "$_id" },
   count: { "$sum": 1 }
 }
},
{ $match: {
  count: { "$gt": 1 }
}}
],allowDiskUse: true}
)
.result
.forEach(function(doc) {
  doc.dups.shift();
  doc.dups.forEach( function(dupId){
    duplicates.push(dupId);
  }
)
})
// printjson(duplicates);

// Remove all duplicates in one go
db.transactions.remove({_id:{$in:duplicates}})
```

Anhäufung online lesen: https://riptutorial.com/de/mongodb/topic/3852/anhaufung

Kapitel 6: Aufbau

Parameter

Parameter	Standard
systemLog.verbosity	0
systemLog.quiet	falsch
systemLog.traceAllExceptions	falsch
systemLog.syslogFacility	Nutzer
systemLog.path	-
systemLog.logAppend	falsch
systemLog.logRotate	umbenennen
systemLog.destination	stdout
systemLog.timeStampFormat	iso8601-local
systemLog.component.accessControl.verbosity	0
systemLog.component.command.verbosity	0
systemLog.component.control.verbosity	0
systemLog.component.ftdc.verbosity	0
systemLog.component.geo.verbosity	0
systemLog.component.index.verbosity	0
systemLog.component.network.verbo	0
systemLog.component.query.verbosity	0
systemLog.component.replication.verbosity	0
systemLog.component.sharding.verbosity	0
systemLog.component.storage.verbosity	0
systemLog.component.storage.journal.verbosity	0
systemLog.component.write.verbosity	0

Parameter	Standard
processManagement.fork	falsch
processManagement.pidFilePath	keiner
net.port	27017
net.bindIp	0,0,0,0
net.maxIncomingConnections	65536
net.wireObjectCheck	wahr
net.ipv6	falsch
net.unixDomainSocket.enabled	wahr
net.unixDomainSocket.pathPrefix	/ tmp
net.unixDomainSocket.filePermissions	0700
net.http.enabled	falsch
net.http.JSONPEnabled	falsch
net.http.RESTInterfaceEnabled	falsch
net.ssl.sslOnNormalPorts	falsch
net.ssl.mode	deaktiviert
net.ssl.PEMKeyFile	keiner
net.ssl.PEMKeyPassword	keiner
net.ssl.clusterFile	keiner
net.ssl.clusterPassword	keiner
net.ssl.CAFile	keiner
net.ssl.CRLFile	keiner
net.ssl.allowConnectionsWithoutCertificates	falsch
net.ssl.allowInvalidCertificates	falsch
net.ssl.allowInvalidHostnames	falsch
net.ssl.disabledProtocols	keiner

Parameter	Standard
net.ssl.FIPSMode	falsch

Examples

Mongo mit einer bestimmten Konfigurationsdatei starten

Verwenden Sie das Flag `--config` .

```
$ /bin/mongod --config /etc/mongod.conf
$ /bin/mongos --config /etc/mongos.conf
```

Beachten Sie, dass `-f` das kürzere Synonym für `--config` .

Aufbau online lesen: https://riptutorial.com/de/mongodb/topic/5985/aufbau

Kapitel 7: Authentifizierungsmechanismen in MongoDB

Einführung

Bei der Authentifizierung wird die Identität eines Clients überprüft. Wenn die Zugriffskontrolle (Autorisierung) aktiviert ist, verlangt MongoDB von allen Clients, dass sie sich authentifizieren, um ihren Zugriff zu bestimmen.

MongoDB unterstützt eine Reihe von Authentifizierungsmechanismen, mit denen Clients ihre Identität überprüfen können. Diese Mechanismen ermöglichen die Integration von MongoDB in Ihr vorhandenes Authentifizierungssystem.

Examples

Authentifizierungsmechanismen

MongoDB unterstützt mehrere Authentifizierungsmechanismen.

Client- und Benutzerauthentifizierungsmechanismen

- SCRAM-SHA-1
- X.509-Zertifikatauthentifizierung
- MongoDB Challenge und Antwort (MONGODB-CR)
- LDAP-Proxy-Authentifizierung und
- Kerberos-Authentifizierung

Interne Authentifizierungsmechanismen

- Schlüsseldatei
- X.509

Authentifizierungsmechanismen in MongoDB online lesen:
https://riptutorial.com/de/mongodb/topic/8113/authentifizierungsmechanismen-in-mongodb

Kapitel 8: CRUD-Operation

Syntax

- Einfügen (*Dokument oder Array von Dokumenten*)
- insertOne ('document', {writeConcern: 'document'})
- insertMany ({[Dokument 1, Dokument 2, ...]}, {writeConcern: Dokument, sortiert: boolean})
- find (*Abfrage* , *Projektion*)
- findOne (*Abfrage* , *Projektion*)
- Update (*Abfrage* , *Update*)
- updateOne (*Abfrage* , *Aktualisierung* , {upsert: boolean, writeConcern: document})
- updateMany (*Abfrage* , *Aktualisierung* , {upsert: boolean, writeConcern: document})
- replaceOne (*Abfrage* , *Ersetzung* , {upsert: boolean, writeConcern: document})
- entfernen (*Abfrage* , *justOne*)
- findAndModify (*Abfrage* , *Sortierung* , *Aktualisierung* , *Optionen [optional]*)

Bemerkungen

Das Aktualisieren und Löschen eines Dokuments sollte sorgfältig durchgeführt werden. Da kann sich der Vorgang auf mehrere Dokumente auswirken.

Examples

Erstellen

```
db.people.insert({name: 'Tom', age: 28});
```

Oder

```
db.people.save({name: 'Tom', age: 28});
```

Der Unterschied zum `save` ist, dass, wenn das übergebene Dokument ein `_id` Feld enthält, wenn ein Dokument bereits mit dieser `_id` es aktualisiert wird, anstatt als neu hinzugefügt zu werden.

Zwei neue Methoden zum Einfügen von Dokumenten in eine Sammlung in MongoDB 3.2.x: -

Verwenden Sie `insertOne` , um nur einen Datensatz einzufügen: -

```
db.people.insertOne({name: 'Tom', age: 28});
```

Verwenden Sie `insertMany` , um mehrere Datensätze einzufügen: -

```
db.people.insertMany([{name: 'Tom', age: 28},{name: 'John', age: 25}, {name: 'Kathy', age:
23}])
```

Beachten Sie, dass die `insert` seit Version 3.0 in jedem offiziellen Sprachentreiber als veraltet markiert ist. Der volle Unterschied besteht darin, dass die Shell-Methoden bei der Implementierung der Methode tatsächlich hinter den anderen Treibern zurückgeblieben sind. Das gleiche gilt für alle anderen CRUD-Methoden

Aktualisieren

Aktualisieren Sie das **gesamte** Objekt:

```
db.people.update({name: 'Tom'}, {age: 29, name: 'Tom'})

// New in MongoDB 3.2
db.people.updateOne({name: 'Tom'},{age: 29, name: 'Tom'}) //Will replace only first matching
document.

db.people.updateMany({name: 'Tom'},{age: 29, name: 'Tom'}) //Will replace all matching
documents.
```

Oder aktualisieren Sie einfach ein einzelnes Feld eines Dokuments. In diesem Fall `age` :

```
db.people.update({name: 'Tom'}, {$set: {age: 29}})
```

Sie können auch mehrere Dokumente gleichzeitig aktualisieren, indem Sie einen dritten Parameter hinzufügen. Diese Abfrage aktualisiert alle Dokumente, bei denen der Name `Tom` entspricht:

```
db.people.update({name: 'Tom'}, {$set: {age: 29}}, {multi: true})

// New in MongoDB 3.2
db.people.updateOne({name: 'Tom'},{$set:{age: 30}) //Will update only first matching document.

db.people.updateMany({name: 'Tom'},{$set:{age: 30}}) //Will update all matching documents.
```

Wenn ein neues Feld zur Aktualisierung erscheint, wird dieses Feld dem Dokument hinzugefügt.

```
db.people.updateMany({name: 'Tom'},{$set:{age: 30, salary:50000}})// Document will have
`salary` field as well.
```

Wenn ein Dokument ersetzt werden muss,

```
db.collection.replaceOne({name:'Tom'}, {name:'Lakmal',age:25,address:'Sri Lanka'})
```

kann verwendet werden.

Hinweis : Felder, die Sie zur Identifizierung des Objekts verwenden, werden im aktualisierten Dokument gespeichert. Felder, die nicht im Aktualisierungsabschnitt definiert sind, werden aus dem Dokument entfernt.

Löschen

Löscht alle Dokumente, die dem Abfrageparameter entsprechen:

```
// New in MongoDB 3.2
db.people.deleteMany({name: 'Tom'})

// All versions
db.people.remove({name: 'Tom'})
```

Oder nur eine

```
// New in MongoDB 3.2
db.people.deleteOne({name: 'Tom'})

// All versions
db.people.remove({name: 'Tom'}, true)
```

MongoDBs `remove()` Methode. Wenn Sie diesen Befehl ohne ein Argument oder ohne ein leeres Argument ausführen, werden alle Dokumente aus der Sammlung entfernt.

```
db.people.remove();
```

oder

```
db.people.remove({});
```

Lesen

Abfrage für alle Dokumente in der `people` Sammlung , die einen haben `name` Feld mit einem Wert von `'Tom'` :

```
db.people.find({name: 'Tom'})
```

Oder nur der erste:

```
db.people.findOne({name: 'Tom'})
```

Sie können auch angeben, welche Felder zurückgegeben werden sollen, indem Sie einen Feldauswahlparameter übergeben. Folgendes schließt das `_id` Feld aus und enthält nur das `age` Feld:

```
db.people.find({name: 'Tom'}, {_id: 0, age: 1})
```

Hinweis: Standardmäßig wird das Feld `_id` zurückgegeben, auch wenn Sie nicht danach fragen. Wenn Sie die `_id` zurückbekommen `_id` , folgen Sie einfach dem vorherigen Beispiel und fordern Sie die `_id` unter Angabe von `_id: 0` (oder `_id: false`) aus , Stadt usw.

```
db.people.find({'address.country': 'US'})
```

& bei Bedarf auch Feld angeben

```
db.people.find({'address.country': 'US'}, {'name': true, 'address.city': true})Remember that
the result has a `.pretty()` method that pretty-prints resulting JSON:

db.people.find().pretty()
```

Weitere Update-Operatoren

Sie können bei der Aktualisierung eines Dokuments neben `$set` andere Operatoren verwenden. Mit dem `$push` Operator können Sie einen Wert in ein Array verschieben. In diesem Fall fügen Sie dem `nicknames` Array einen neuen Kurznamen hinzu.

```
db.people.update({name: 'Tom'}, {$push: {nicknames: 'Tommy'}})
// This adds the string 'Tommy' into the nicknames array in Tom's document.
```

Der `$pull` Operator ist das Gegenteil von `$push` . Sie können bestimmte Elemente aus Arrays ziehen.

```
db.people.update({name: 'Tom'}, {$pull: {nicknames: 'Tommy'}})
// This removes the string 'Tommy' from the nicknames array in Tom's document.
```

Mit dem Operator `$pop` können Sie den ersten oder den letzten Wert aus einem Array entfernen. Angenommen, Toms Dokument hat eine Eigenschaft namens Geschwister, die den Wert `['Marie', 'Bob', 'Kevin', 'Alex']` .

```
db.people.update({name: 'Tom'}, {$pop: {siblings: -1}})
// This will remove the first value from the siblings array, which is 'Marie' in this case.


db.people.update({name: 'Tom'}, {$pop: {siblings: 1}})
// This will remove the last value from the siblings array, which is 'Alex' in this case.
```

"multi" Parameter beim Aktualisieren mehrerer Dokumente

Um mehrere Dokumente in einer Sammlung zu aktualisieren, setzen Sie die Multi-Option auf true.

```
db.collection.update(
   query,
   update,
   {
     upsert: boolean,
     multi: boolean,
     writeConcern: document
   }
)
```

multi ist optional. Wenn der Wert auf true gesetzt ist, werden mehrere Dokumente aktualisiert, die die Abfragekriterien erfüllen. Bei Festlegung auf "false" wird ein Dokument aktualisiert. Der Standardwert ist false.

db.mycol.find () {"_id": ObjectId (598354878df45ec5), "title": "MongoDB Overview"} {"_id": ObjectId (59835487adf45ec6), "title": "NoSQL Overview"} {"_id": ObjectId (59835487adf45ec7), "title": "Übersicht über Lernprogramme"}

db.mycol.update ({'title': 'MongoDB Overview'}, {$ set: {'title': 'Neues MongoDB-Tutorial'}}, {multi: true})

Update eingebetteter Dokumente.

Für das folgende Schema:

```
{name: 'Tom', age: 28, marks: [50, 60, 70]}
```

Tom's Marken auf 55 aktualisieren, wobei Marken 50 sind (Verwenden Sie den Positionsoperator $):

```
db.people.update({name: "Tom", marks: 50}, {"$set": {"marks.$": 55}})
```

Für das folgende Schema:

```
{name: 'Tom', age: 28, marks: [{subject: "English", marks: 90},{subject: "Maths", marks: 100},
{subject: "Computes", marks: 20}]}
```

Aktualisieren Sie die englischen Markierungen von Tom auf 85:

```
db.people.update({name: "Tom", "marks.subject": "English"},{"$set":{"marks.$.marks": 85}})
```

Erklärendes Beispiel oben:

Wenn Sie {name: "Tom", "marks.subject": "English"} verwenden, erhalten Sie die Position des Objekts im Markierungen-Array, wobei das Subjekt Englisch ist. In "marks. $. Marks" wird $ verwendet, um an dieser Position des Marken-Arrays zu aktualisieren

Werte in einem Array aktualisieren

Der positional $ -Operator identifiziert ein Element in einem Array, das aktualisiert werden soll, ohne die Position des Elements im Array explizit anzugeben.

Betrachten Sie eine Sammlung Schüler mit den folgenden Dokumenten:

```
{ "_id" : 1, "grades" : [ 80, 85, 90 ] }
{ "_id" : 2, "grades" : [ 88, 90, 92 ] }
{ "_id" : 3, "grades" : [ 85, 100, 90 ] }
```

Verwenden Sie zum Aktualisieren von 80 bis 82 im Notenarray im ersten Dokument den Operator positional $, wenn Sie die Position des Elements im Array nicht kennen:

```
db.students.update(
```

```
    { _id: 1, grades: 80 },
    { $set: { "grades.$" : 82 } }
)
```

CRUD-Operation online lesen: https://riptutorial.com/de/mongodb/topic/1683/crud-operation

Kapitel 9: Daten abfragen (Erste Schritte)

Einführung

Grundlegende Abfragebeispiele

Examples

Finden()

alle Dokumente in einer Sammlung abrufen

```
db.collection.find({});
```

Abrufen von Dokumenten in einer Auflistung mithilfe einer Bedingung (ähnlich WHERE in MYSQL)

```
db.collection.find({key: value});
example
  db.users.find({email:"sample@email.com"});
```

Abrufen von Dokumenten in einer Sammlung unter Verwendung boolescher Bedingungen (Abfrageoperatoren)

```
//AND
db.collection.find( {
    $and: [
     { key: value }, { key: value }
    ]
})
//OR
db.collection.find( {
    $or: [
     { key: value }, { key: value }
    ]
})
//NOT
db.inventory.find( { key: { $not: value } } )
```

Weitere boolesche Operationen und Beispiele finden Sie hier

HINWEIS: *find ()* durchsucht die Sammlung auch dann weiter, wenn eine Dokumentübereinstimmung gefunden wurde. Daher ist es bei Verwendung in einer großen Sammlung ineffizient. Wenn Sie jedoch Ihre Daten sorgfältig modellieren und / oder Indizes verwenden, können Sie die *Suche* effizienter gestalten *( )*

Einen finden()

```
db.collection.findOne({});
```

Die Abfragefunktion ist ähnlich wie bei find (). Die Ausführung wird jedoch in dem Moment beendet, in dem ein Dokument gefunden wird, das mit seiner Bedingung übereinstimmt. Wenn es mit einem leeren Objekt verwendet wird, wird es das erste Dokument abrufen und zurückgeben. findOne () Mongodb-API-Dokumentation

Dokument abfragen - Verwenden von AND-, OR- und IN-Bedingungen

Alle Dokumente aus der `students` .

```
> db.students.find().pretty();

{
    "_id" : ObjectId("58f29a694117d1b7af126dca"),
    "studentNo" : 1,
    "firstName" : "Prosen",
    "lastName" : "Ghosh",
    "age" : 25
}
{
    "_id" : ObjectId("58f29a694117d1b7af126dcb"),
    "studentNo" : 2,
    "firstName" : "Rajib",
    "lastName" : "Ghosh",
    "age" : 25
}
{
    "_id" : ObjectId("58f29a694117d1b7af126dcc"),
    "studentNo" : 3,
    "firstName" : "Rizve",
    "lastName" : "Amin",
    "age" : 23
}
{
    "_id" : ObjectId("58f29a694117d1b7af126dcd"),
    "studentNo" : 4,
    "firstName" : "Jabed",
    "lastName" : "Bangali",
    "age" : 25
}
{
    "_id" : ObjectId("58f29a694117d1b7af126dce"),
    "studentNo" : 5,
    "firstName" : "Gm",
    "lastName" : "Anik",
    "age" : 23
}
```

Ähnliche `mySql` Abfrage des obigen Befehls.

```
SELECT * FROM students;
```

```
db.students.find({firstName:"Prosen"});
```

```
{ "_id" : ObjectId("58f2547804951ad51ad206f5"), "studentNo" : "1", "firstName" : "Prosen",
"lastName" : "Ghosh", "age" : "23" }
```

Ähnliche `mySql` Abfrage des obigen Befehls.

```
SELECT * FROM students WHERE firstName = "Prosen";
```

UND Abfragen

```
db.students.find({
    "firstName": "Prosen",
    "age": {
        "$gte": 23
    }
});

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen",
"lastName" : "Ghosh", "age" : 25 }
```

Ähnliche `mySql` Abfrage des obigen Befehls.

```
SELECT * FROM students WHERE firstName = "Prosen" AND age >= 23
```

Oder Abfragen

```
db.students.find({
     "$or": [{
         "firstName": "Prosen"
     }, {
         "age": {
             "$gte": 23
         }
     }]
 });

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen",
"lastName" : "Ghosh", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcb"), "studentNo" : 2, "firstName" : "Rajib",
"lastName" : "Ghosh", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcc"), "studentNo" : 3, "firstName" : "Rizve",
"lastName" : "Amin", "age" : 23 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcd"), "studentNo" : 4, "firstName" : "Jabed",
"lastName" : "Bangali", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dce"), "studentNo" : 5, "firstName" : "Gm",
"lastName" : "Anik", "age" : 23 }
```

Ähnliche `mySql` Abfrage des obigen Befehls.

```
SELECT * FROM students WHERE firstName = "Prosen" OR age >= 23
```

Und ODER-Anfragen

```
db.students.find({
```

```
        firstName : "Prosen",
        $or : [
            {age : 23},
            {age : 25}
        ]
});

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen",
"lastName" : "Ghosh", "age" : 25 }
```

Ähnliche mySql-Abfrage des obigen Befehls.

```
SELECT * FROM students WHERE firstName = "Prosen" AND age = 23 OR age = 25;
```

IN-Abfragen Diese Abfragen können die mehrfache Verwendung von ODER-Abfragen verbessern

```
db.students.find(lastName:{$in:["Ghosh", "Amin"]})

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen",
"lastName" : "Ghosh", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcb"), "studentNo" : 2, "firstName" : "Rajib",
"lastName" : "Ghosh", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcc"), "studentNo" : 3, "firstName" : "Rizve",
"lastName" : "Amin", "age" : 23 }
```

Ähnliche mySql-Abfrage zum obigen Befehl

```
select * from students where lastName in ('Ghosh', 'Amin')
```

find () -Methode mit Projektion

Die grundlegende Syntax der `find()` Methode mit Projektion lautet wie folgt

```
> db.COLLECTION_NAME.find({},{KEY:1});
```

Wenn Sie alle Dokumente ohne das Feld Alter anzeigen möchten, lautet der Befehl wie folgt

```
db.people.find({},{age : 0});
```

Wenn Sie alle Dokumente im Feld Alter anzeigen möchten, lautet der Befehl wie folgt

Find () -Methode mit Projektion

In MongoDB bedeutet Projektion, nur die erforderlichen Daten auszuwählen, anstatt die gesamten Daten eines Dokuments auszuwählen.

Die grundlegende Syntax der `find()` Methode mit Projektion lautet wie folgt

```
> db.COLLECTION_NAME.find({},{KEY:1});
```

Wenn Sie alle Dokumente ohne das Feld Alter anzeigen möchten, lautet der Befehl wie folgt

```
> db.people.find({},{age:0});
```

Wenn Sie nur das Alter-Feld anzeigen möchten, lautet der Befehl wie folgt

```
> db.people.find({},{age:1});
```

Hinweis: `_id` Feld `_id` wird bei der Ausführung der `find()` -Methode immer angezeigt. Wenn Sie dieses Feld nicht möchten, müssen Sie es auf `0` .

```
> db.people.find({},{name:1,_id:0});
```

Hinweis: Mit `1` wird das Feld angezeigt, mit `0` werden die Felder ausgeblendet.

beschränken, überspringen, sortieren und zählen Sie die Ergebnisse der find () -Methode

Ähnlich wie bei Aggregationsmethoden haben Sie auch bei find () die Möglichkeit, die Ergebnisse einzuschränken, zu überspringen, zu sortieren und zu zählen. Nehmen wir an, wir haben folgende Sammlung:

```
db.test.insertMany([
    {name:"Any", age:"21", status:"busy"},
    {name:"Tony", age:"25", status:"busy"},
    {name:"Bobby", age:"28", status:"online"},
    {name:"Sonny", age:"28", status:"away"},
    {name:"Cher", age:"20", status:"online"}
])
```

Um die Sammlung aufzulisten:

```
db.test.find({})
```

Wird zurückkehren:

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b0"), "name" : "Any", "age" : "21", "status" :
"busy" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b1"), "name" : "Tony", "age" : "25", "status" :
"busy" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b2"), "name" : "Bobby", "age" : "28", "status" :
"online" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" :
"away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" :
"online" }
```

Erste 3 Dokumente überspringen:

```
db.test.find({}).skip(3)
```

Wird zurückkehren:

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" :
"away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" :
"online" }
```

Absteigend nach Feldnamen sortieren:

```
db.test.find({}).sort({ "name" : -1})
```

Wird zurückkehren:

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b1"), "name" : "Tony", "age" : "25", "status" :
"busy" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" :
"away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" :
"online" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b2"), "name" : "Bobby", "age" : "28", "status" :
"online" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b0"), "name" : "Any", "age" : "21", "status" :
"busy" }
```

Wenn Sie aufsteigend sortieren möchten, ersetzen Sie -1 durch 1

Um die Ergebnisse zu zählen:

```
db.test.find({}).count()
```

Wird zurückkehren:

```
5
```

Auch Kombinationen dieser Methoden sind zulässig. Erhalten Sie zum Beispiel 2 Dokumente aus einer absteigend sortierten Sammlung, wobei die erste 1 übersprungen wird:

```
db.test.find({}).sort({ "name" : -1}).skip(1).limit(2)
```

Wird zurückkehren:

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" :
"away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" :
"online" }
```

Daten abfragen (Erste Schritte) online lesen: https://riptutorial.com/de/mongodb/topic/9271/daten-abfragen--erste-schritte-

Kapitel 10: Daten sichern und wiederherstellen

Examples

Mongoimport mit JSON

Beispiel für einen ZIP-Datensatz in zipcodes.json, gespeichert in c: \ Users \ yc03ak1 \ Desktop \ zips.json

```
{ "_id" : "01001", "city" : "AGAWAM", "loc" : [ -72.622739, 42.070206 ], "pop" : 15338,
"state" : "MA" }
{ "_id" : "01002", "city" : "CUSHMAN", "loc" : [ -72.51564999999999, 42.377017 ], "pop" :
36963, "state" : "MA" }
{ "_id" : "01005", "city" : "BARRE", "loc" : [ -72.10835400000001, 42.409698 ], "pop" : 4546,
"state" : "MA" }
{ "_id" : "01007", "city" : "BELCHERTOWN", "loc" : [ -72.41095300000001, 42.275103 ], "pop" :
10579, "state" : "MA" }
{ "_id" : "01008", "city" : "BLANDFORD", "loc" : [ -72.936114, 42.182949 ], "pop" : 1240,
"state" : "MA" }
{ "_id" : "01010", "city" : "BRIMFIELD", "loc" : [ -72.188455, 42.116543 ], "pop" : 3706,
"state" : "MA" }
{ "_id" : "01011", "city" : "CHESTER", "loc" : [ -72.988761, 42.279421 ], "pop" : 1688,
"state" : "MA" }
```

Importieren dieses Datensatzes in die Datenbank "test" und die Auflistung "zips"

```
C:\Users\yc03ak1>mongoimport --db test --collection "zips" --drop --type json --host
"localhost:47019"  --file "c:\Users\yc03ak1\Desktop\zips.json"
```

- --db: Name der Datenbank, in die Daten importiert werden sollen
- --collection: Name der Sammlung in der Datenbank, in der Daten veröffentlicht werden sollen
- --drop: löscht die Sammlung vor dem Importieren
- --type: Dokumenttyp, der importiert werden muss. Standard-JSON
- --host: Mongodb-Host und -Port, auf dem Daten importiert werden sollen.
- --file: Pfad, in dem sich die Json-Datei befindet

Ausgabe :

```
2016-08-10T20:10:50.159-0700    connected to: localhost:47019
2016-08-10T20:10:50.163-0700    dropping: test.zips
2016-08-10T20:10:53.155-0700    [################........] test.zips    2.1 MB/3.0 MB (68.5%)
2016-08-10T20:10:56.150-0700    [########################] test.zips    3.0 MB/3.0 MB (100.0%)
2016-08-10T20:10:57.819-0700    [########################] test.zips    3.0 MB/3.0 MB (100.0%)
2016-08-10T20:10:57.821-0700    imported 29353 documents
```

Mongoimport mit CSV

Beispiel einer CSV-Datei für Testdatensätze, die im Verzeichnis c: \ Users \ yc03ak1 \ Desktop \ testing.csv gespeichert ist

```
_id    city    loc    pop    state
1    A    [10.0, 20.0]    2222    PQE
2    B    [10.1, 20.1]    22122    RW
3    C    [10.2, 20.0]    255222    RWE
4    D    [10.3, 20.3]    226622    SFDS
5    E    [10.4, 20.0]    222122    FDS
```

Importieren dieses Datensatzes in die Datenbank "test" und die Auflistung "sample"

```
C:\Users\yc03ak1>mongoimport --db test --collection "sample" --drop --type csv --headerline --
host "localhost:47019"  --file "c:\Users\yc03ak1\Desktop\testing.csv"
```

- --headerline: Verwenden Sie die erste Zeile der CSV-Datei als Felder für das Json-Dokument

Ausgabe :

```
2016-08-10T20:25:48.572-0700    connected to: localhost:47019
2016-08-10T20:25:48.576-0700    dropping: test.sample
2016-08-10T20:25:49.109-0700    imported 5 documents
```

ODER

```
C:\Users\yc03ak1>mongoimport --db test --collection "sample" --drop --type csv --fields
_id,city,loc,pop,state --host "localhost:47019"  --file "c:\Users\yc03ak1\Desktop\testing.csv"
```

- --fields: kommagetrennte Liste der Felder, die in das json-Dokument importiert werden müssen. Ausgabe:

```
2016-08-10T20:26:48.978-0700    connected to: localhost:47019
2016-08-10T20:26:48.982-0700    dropping: test.sample
2016-08-10T20:26:49.611-0700    imported 6 documents
```

Daten sichern und wiederherstellen online lesen:
https://riptutorial.com/de/mongodb/topic/6290/daten-sichern-und-wiederherstellen

Kapitel 11: Daten sichern und wiederherstellen

Examples

Grundlegender Mongodump der lokalen Standard-Mongod-Instanz

```
mongodump --db mydb --gzip --out "mydb.dump.$(date +%F_%R)"
```

Mit diesem Befehl wird ein von bson gzipped erstelltes Archiv Ihrer lokalen Mongod-Datenbank "mydb" im Verzeichnis "mydb.dump. {Timestamp}" abgelegt

Grundlegender Mongorestore des lokalen Standard-Mongod-Dumps

```
mongorestore --db mydb mydb.dump.2016-08-27_12:44/mydb --drop --gzip
```

Dieser Befehl löscht zuerst Ihre aktuelle 'mydb' -Datenbank und stellt dann Ihren gzippten bson-dump aus der 'mydb mydb.dump.2016-08-27_12: 44 / mydb'-Archiv-Dump-Datei wieder her.

Daten sichern und wiederherstellen online lesen:
https://riptutorial.com/de/mongodb/topic/6494/daten-sichern-und-wiederherstellen

Kapitel 12: Datenbankinformationen abrufen

Examples

Alle Datenbanken auflisten

```
show dbs
```

oder

```
db.adminCommand('listDatabases')
```

oder

```
db.getMongo().getDBNames()
```

Alle Sammlungen in der Datenbank auflisten

```
show collections
```

oder

```
show tables
```

oder

```
db.getCollectionNames()
```

Datenbankinformationen abrufen online lesen:
https://riptutorial.com/de/mongodb/topic/6397/datenbankinformationen-abrufen

Kapitel 13: Indizes

Syntax

- `db.collection.createIndex({ <string field> : <1|-1 order> [, <string field> : <1|-1 order>] });`

Bemerkungen

Performance Auswirkungen: Beachten Sie, dass Indizes Lese Leistungen zu verbessern, kann aber schlechte Auswirkungen auf die Schreibleistung haben, wie das Einfügen eines Dokuments verlangt , dass alle Indizes zu aktualisieren.

Examples

Einzelfeld

```
db.people.createIndex({name: 1})
```

Dadurch entsteht eine aufsteigende einzelnes Feld Index auf den *Feldnamen.*

Bei dieser Art von Indizes ist die Sortierreihenfolge irrelevant, da Mongo den Index in beide Richtungen durchlaufen kann.

Verbindung

```
db.people.createIndex({name: 1, age: -1})
```

Dadurch wird ein Index für mehrere Felder erstellt, in diesem Fall für die Felder `name` und `age` . Es wird im `name` aufsteigend und im `age` absteigend.

Bei diesem Indextyp ist die Sortierreihenfolge relevant, da er bestimmt, ob der Index eine Sortieroperation unterstützen kann oder nicht. Die umgekehrte Sortierung wird für jedes Präfix eines zusammengesetzten Index unterstützt, sofern die Sortierung in umgekehrter Sortierrichtung für **alle** Schlüssel in der Sortierung erfolgt. Andernfalls muss die Sortierung nach zusammengesetzten Indizes mit der Reihenfolge des Index übereinstimmen.

Die Reihenfolge der Felder ist ebenfalls wichtig. In diesem Fall wird der Index zuerst nach `name` und innerhalb jedes Namenswerts nach den Werten des Felds `age` sortiert. Dies ermöglicht es der Index von Abfragen auf dem zu verwendenden `name` Feld oder auf `name` und `age` , aber nicht auf `age` allein.

Löschen

Um einen Index zu löschen, können Sie den Indexnamen verwenden

```
db.people.dropIndex("nameIndex")
```

Oder das Indexspezifikationsdokument

```
db.people.dropIndex({name: 1})
```

Liste

```
db.people.getIndexes()
```

Dadurch wird ein Array von Dokumenten zurückgegeben, die jeweils einen Index der *Personensammlung beschreiben*

Grundlagen zur Indexerstellung

Siehe die untenstehende Transaktionssammlung.

```
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 100, fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 10,  fee : 2});
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 4});
> db.transactions.insert({ cr_dr : "D", amount : 10,  fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 10,  fee : 4});
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 2});
```

`getIndexes()` Funktionen zeigen alle für eine Collection verfügbaren Indizes an.

```
db.transactions.getIndexes();
```

Lassen Sie die Ausgabe der obigen Anweisung sehen.

```
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    }
]
```

Es gibt bereits einen Index für die Transaktionserfassung. Dies liegt daran, dass MongoDB während der Erstellung einer Sammlung einen *eindeutigen Index* für das Feld `_id` . Der `_id` Index verhindert, dass Clients zwei Dokumente mit demselben Wert für das `_id` Feld `_id` . Sie können diesen Index nicht im Feld `_id` .

Nun fügen wir einen Index für das cr_dr-Feld hinzu.

```
db.transactions.createIndex({ cr_dr : 1 });
```

Das Ergebnis der Indexausführung lautet wie folgt.

```
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

> Die createdCollectionAutomatically gibt an, ob der Vorgang eine Auflistung erstellt hat. Wenn keine Sammlung vorhanden ist, erstellt MongoDB die Sammlung als Teil des Indexierungsvorgangs.

Lassen Sie `db.transactions.getIndexes();` nochmal.

```
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : 1
        },
        "name" : "cr_dr_1",
        "ns" : "documentation_db.transactions"
    }
]
```

Jetzt sehen Sie, dass die Transaktionssammlung zwei Indizes hat. Standard- `_id` Index und `cr_dr_1` die wir erstellt haben. Der Name wird von MongoDB vergeben. Sie können Ihren eigenen Namen wie folgt festlegen.

```
db.transactions.createIndex({ cr_dr : -1 },{name : "index on cr_dr desc"})
```

Jetzt `db.transactions.getIndexes();` gibt Ihnen drei Indizes.

```
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : 1
```

```
        },
        "name" : "cr_dr_1",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : -1
        },
        "name" : "index on cr_dr desc",
        "ns" : "documentation_db.transactions"
    }
]
```

Beim Erstellen des Index `{ cr_dr : -1 }` 1 wird der Index in `ascending` Reihenfolge und -1 in `descending` Reihenfolge `{ cr_dr : -1 }` .

2.4

Hash-Indizes

Indizes können auch als *Hash* definiert werden. Dies ist bei *Gleichheitsabfragen* performanter, bei *Bereichsabfragen* jedoch nicht effizient. Sie können jedoch sowohl gehashte als auch aufsteigende / absteigende Indizes für dasselbe Feld definieren.

```
> db.transactions.createIndex({ cr_dr : "hashed" });

> db.transactions.getIndexes(
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : "hashed"
        },
        "name" : "cr_dr_hashed",
        "ns" : "documentation_db.transactions"
    }
]
```

Index löschen / löschen

Wenn der Indexname bekannt ist,

```
db.collection.dropIndex('name_of_index');
```

Wenn der Indexname nicht bekannt ist,

```
db.collection.dropIndex( { 'name_of_field' : -1 } );
```

Holen Sie sich Indizes einer Sammlung

```
 db.collection.getIndexes();
```

Ausgabe

```
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : 1
        },
        "name" : "cr_dr_1",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : -1
        },
        "name" : "index on cr_dr desc",
        "ns" : "documentation_db.transactions"
    }
]
```

Eindeutiger Index

```
db.collection.createIndex( { "user_id": 1 }, { unique: true } )
```

Eindeutigkeit für den definierten Index erzwingen (entweder einzeln oder zusammengesetzt). Das Erstellen des Index schlägt fehl, wenn die Sammlung bereits doppelte Werte enthält. Die Indizierung schlägt auch fehl, wenn das Feld bei mehreren Einträgen fehlt (da sie alle mit dem Wert `null` indiziert werden), sofern nicht `sparse: true` angegeben ist.

Sparse-Indizes und Teilindizes

Sparse-Indizes:

Dies kann besonders nützlich für Felder sein, die optional sind, aber auch eindeutig sein sollten.

```
{ "_id" : "john@example.com", "nickname" : "Johnnie" }
{ "_id" : "jane@example.com" }
```

```
{ "_id" : "julia@example.com", "nickname" : "Jules"}
{ "_id" : "jack@example.com" }
```

Da für zwei Einträge kein "Kurzname" angegeben ist und die Indexierung nicht festgelegte Felder als Nullwert behandelt, schlägt die Indexerstellung mit zwei Dokumenten mit "Nullwert" fehl.

```
db.scores.createIndex( { nickname: 1 } , { unique: true, sparse: true } )
```

werden Sie immer noch 'null' Spitznamen haben.

Sparse-Indizes sind kompakter, da sie Dokumente überspringen / ignorieren, in denen dieses Feld nicht angegeben ist. Wenn Sie also über eine Sammlung verfügen, in der nur weniger als 10% der Dokumente dieses Feld angeben, können Sie wesentlich kleinere Indizes erstellen. So können Sie den begrenzten Speicher besser nutzen, wenn Sie Abfragen wie:

```
db.scores.find({'nickname': 'Johnnie'})
```

Teilindizes:

> Teilindizes stellen eine Obermenge der von Sparse-Indizes gebotenen Funktionalität dar und sollten gegenüber Sparse-Indizes bevorzugt werden. (*Neu in Version 3.2*)

Teilindizes bestimmen die Indexeinträge basierend auf dem angegebenen Filter.

```
db.restaurants.createIndex(
  { cuisine: 1 },
  { partialFilterExpression: { rating: { $gt: 5 } } }
)
```

Ist die `rating` größer als 5, wird die `cuisine` indexiert. Ja, wir können auch eine Eigenschaft angeben, die basierend auf dem Wert anderer Eigenschaften indiziert werden soll.

Unterschied zwischen Sparse- und Teilindizes:

Sparse-Indizes wählen die zu indizierenden Dokumente ausschließlich auf der Grundlage des Vorhandenseins des indizierten Feldes oder bei zusammengesetzten Indizes das Vorhandensein der indizierten Felder aus.

Teilindizes bestimmen die Indexeinträge basierend auf dem angegebenen Filter. Der Filter kann andere Felder als die Indexschlüssel enthalten und kann andere Bedingungen als nur eine Existenzprüfung angeben.

Ein Teilindex kann jedoch dasselbe Verhalten wie ein Index mit geringer Dichte implementieren

Z.B:

```
db.contacts.createIndex(
   { name: 1 },
   { partialFilterExpression: { name: { $exists: true } } }
)
```

Anmerkung: Die Option *partialFilterExpression* und die Option " *Sparse* " können nicht gleichzeitig angegeben werden.

Indizes online lesen: https://riptutorial.com/de/mongodb/topic/3934/indizes

Kapitel 14: Java-Treiber

Examples

Erstellen Sie einen anpassbaren Cursor

```
find(query).projection(fields).cursorType(CursorType.TailableAwait).iterator();
```

Dieser Code gilt für die `MongoCollection` Klasse.

CursorType ist eine Aufzählung und hat folgende Werte:

```
Tailable
TailableAwait
```

Entsprechend den alten (<3.0) DBCursor-Typen addOption Bytes:

```
Bytes.QUERYOPTION_TAILABLE
Bytes.QUERYOPTION_AWAITDATA
```

Erstellen Sie einen Datenbankbenutzer

So erstellen Sie einen Benutzer **dev** mit dem Kennwort **Kennwort123**

```
MongoClient mongo = new MongoClient("localhost", 27017);
MongoDatabase db =  mongo.getDatabase("testDB");
Map<String, Object> commandArguments = new BasicDBObject();
commandArguments.put("createUser", "dev");
commandArguments.put("pwd", "password123");
String[] roles = { "readWrite" };
commandArguments.put("roles", roles);
BasicDBObject command = new BasicDBObject(commandArguments);
db.runCommand(command);
```

Sammeln von Daten mit Bedingung

`testcollection` von Daten aus der `testcollection` Sammlung in der `testdb` Datenbank, wobei `name=dev`

```
import org.bson.Document;
import com.mongodb.BasicDBObject;
import com.mongodb.MongoClient;
import com.mongodb.ServerAddress;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;

MongoClient mongoClient = new MongoClient(new ServerAddress("localhost", 27017));
MongoDatabase db = mongoClient.getDatabase("testdb");
```

```
MongoCollection<Document> collection = db.getCollection("testcollection");

BasicDBObject searchQuery = new BasicDBObject();
searchQuery.put("name","dev");

MongoCursor<Document> cursor = collection.find(searchQuery).iterator();
try {
    while (cursor.hasNext()) {
        System.out.println(cursor.next().toJson());
    }
} finally {
    cursor.close();
}
```

Java-Treiber online lesen: https://riptutorial.com/de/mongodb/topic/6286/java-treiber

Kapitel 15: Massenvorgänge

Bemerkungen

Erstellen einer Liste von Schreibvorgängen, die für eine einzelne Sammlung in Massen ausgeführt werden sollen.

Examples

Konvertieren eines Felds in einen anderen Typ und Aktualisieren der gesamten Sammlung in Massen

In der Regel ist es der Fall, wenn ein Feldtyp in einen anderen geändert werden soll. In der Originalsammlung können beispielsweise "numerische" oder "Datums" -Felder als Zeichenfolgen gespeichert sein:

```
{
    "name": "Alice",
    "salary": "57871",
    "dob": "1986-08-21"
},
{
    "name": "Bob",
    "salary": "48974",
    "dob": "1990-11-04"
}
```

Das Ziel wäre, eine riesige Sammlung wie die oben genannte zu aktualisieren

```
{
    "name": "Alice",
    "salary": 57871,
    "dob": ISODate("1986-08-21T00:00:00.000Z")
},
{
    "name": "Bob",
    "salary": 48974,
    "dob": ISODate("1990-11-04T00:00:00.000Z")
}
```

Bei relativ kleinen Daten können Sie dies erreichen, indem Sie die Sammlung mithilfe einer `snapshot` mit der `forEach()` Methode des Cursors iterieren und jedes Dokument wie folgt aktualisieren:

```
db.test.find({
    "salary": { "$exists": true, "$type": 2 },
    "dob": { "$exists": true, "$type": 2 }
}).snapshot().forEach(function(doc){
    var newSalary = parseInt(doc.salary),
        newDob = new ISODate(doc.dob);
```

```
    db.test.updateOne(
        { "_id": doc._id },
        { "$set": { "salary": newSalary, "dob": newDob } }
    );
});
```

Während dies für kleine Sammlungen optimal ist, wird die Leistung bei großen Sammlungen erheblich reduziert, da das Durchlaufen einer großen Datenmenge und das Senden jedes Aktualisierungsvorgangs pro Anforderung an den Server eine Rechenstrafe nach sich zieht.

Die `Bulk()` API hilft dabei und verbessert die Leistung erheblich, da Schreibvorgänge nur einmal an den Server gesendet werden. Die Effizienz wird erreicht, da die Methode nicht jede Schreibanforderung an den Server sendet (wie bei der aktuellen Aktualisierungsanweisung innerhalb der `forEach()` Schleife), sondern nur einmal pro 1000 Anfragen, wodurch Aktualisierungen effizienter und schneller als derzeit gemacht werden.

Wenn Sie dasselbe Konzept wie bei der `forEach()` Schleife verwenden, um die Stapel zu erstellen, können Sie die Sammlung wie folgt in großen Mengen aktualisieren. In dieser Demonstration verwendet die in den MongoDB-Versionen `>= 2.6` und `< 3.2` verfügbare `Bulk()` API die Methode `initializeUnorderedBulkOp()` , um die Schreibvorgänge in den Batches parallel und in nicht-deterministischer Reihenfolge auszuführen.

Es aktualisiert alle Dokumente in der `dob` indem die `salary` und `dob` Felder in `numerical` bzw. `datetime` Werte `datetime` werden:

```
var bulk = db.test.initializeUnorderedBulkOp(),
    counter = 0; // counter to keep track of the batch update size

db.test.find({
    "salary": { "$exists": true, "$type": 2 },
    "dob": { "$exists": true, "$type": 2 }
}).snapshot().forEach(function(doc){
    var newSalary = parseInt(doc.salary),
        newDob = new ISODate(doc.dob);
    bulk.find({ "_id": doc._id }).updateOne({
        "$set": { "salary": newSalary, "dob": newDob }
    });

    counter++; // increment counter
    if (counter % 1000 == 0) {
        bulk.execute(); // Execute per 1000 operations and re-initialize every 1000 update
statements
        bulk = db.test.initializeUnorderedBulkOp();
    }
});
```

Das nächste Beispiel gilt für die neue MongoDB-Version `3.2` die die `Bulk()` API seitdem nicht mehr unterstützt und einen neueren Satz von Apis mithilfe von `bulkWrite()` .

Es verwendet die gleichen Cursor wie oben, erstellt jedoch die Arrays mit den Massenvorgängen unter Verwendung der gleichen `forEach()` Cursor-Methode, um jedes Massenschreibdokument in

das Array zu verschieben. Da Schreibbefehle nicht mehr als 1000 Operationen akzeptieren können, müssen Operationen gruppiert werden, um höchstens 1000 Operationen zu haben und das Array neu zu initialisieren, wenn die Schleife die 1000-Iteration erreicht:

```
var cursor = db.test.find({
        "salary": { "$exists": true, "$type": 2 },
        "dob": { "$exists": true, "$type": 2 }
    }),
    bulkUpdateOps = [];

cursor.snapshot().forEach(function(doc){
    var newSalary = parseInt(doc.salary),
        newDob = new ISODate(doc.dob);
    bulkUpdateOps.push({
        "updateOne": {
            "filter": { "_id": doc._id },
            "update": { "$set": { "salary": newSalary, "dob": newDob } }
         }
    });

    if (bulkUpdateOps.length === 1000) {
        db.test.bulkWrite(bulkUpdateOps);
        bulkUpdateOps = [];
    }
});

if (bulkUpdateOps.length > 0) { db.test.bulkWrite(bulkUpdateOps); }
```

Massenvorgänge online lesen: https://riptutorial.com/de/mongodb/topic/6211/massenvorgange

Kapitel 16: Mongo als Nachbildung

Examples

Mongodb als Nachbildung

Wir würden mongodb als Replikatsatz mit 3 Instanzen erstellen. Eine Instanz wäre primär und die anderen zwei Instanzen wären sekundär.

Der Einfachheit halber werde ich ein Replikatset mit drei Instanzen von mongodb auf demselben Server haben. Um dies zu erreichen, würden alle drei mongodb-Instanzen auf unterschiedlichen Portnummern laufen.

In einer Produktionsumgebung, in der eine dedizierte mongodb-Instanz auf einem einzelnen Server ausgeführt wird, können Sie dieselben Portnummern wiederverwenden.

1. Erstellen Sie Datenverzeichnisse (Pfad, in dem die Mongodb-Daten in einer Datei gespeichert werden).

```
- mkdir c:\data\server1 (datafile path for instance 1)
- mkdir c:\data\server2 (datafile path for instance 2)
- mkdir c:\data\server3 (datafile path for instance 3)
```

2. ein. Starten Sie die erste Mongod-Instanz

- Öffnen Sie die Eingabeaufforderung und geben Sie die folgende Eingabetaste ein.

```
mongod --replSet s0 --dbpath c:\data\server1 --port 37017  --smallfiles --oplogSize 100
```

Der obige Befehl ordnet die Instanz von mongodb einem Replikatsatznamen "s0" zu und startet die erste Instanz von mongodb auf Port 37017 mit oplogSize 100 MB

2. b. Starten Sie auf ähnliche Weise die zweite Instanz von Mongodb

```
mongod --replSet s0 --dbpath c:\data\server2 --port 37018  --smallfiles --oplogSize 100
```

Der obige Befehl ordnet die Instanz von mongodb einem replicaSet-Namen "s0" zu und startet die erste Instanz von mongodb an Port 37018 mit oplogSize 100 MB

2. c. Starten Sie nun die dritte Instanz von Mongodb

```
mongod --replSet s0 --dbpath c:\data\server3 --port 37019  --smallfiles --oplogSize 100
```

Der obige Befehl ordnet die Instanz von mongodb einem Replikatsatznamen "s0" zu und startet die erste Instanz von mongodb auf Port 37019 mit oplogSize 100 MB

Wenn alle 3 Instanzen gestartet sind, sind diese 3 Instanzen derzeit unabhängig voneinander. Wir

müssen diese Instanzen jetzt als Replikatsatz gruppieren. Wir machen dies mit Hilfe eines Konfigurationsobjekts.

3.a Verbinden Sie sich über die Mongo-Shell mit einem der Mongod-Server. Öffnen Sie dazu die Eingabeaufforderung und geben Sie ein.

```
mongo --port 37017
```

Sobald Sie mit der Mongo-Shell verbunden sind, erstellen Sie ein Konfigurationsobjekt

```
   var config = {"_id":"s0", members[]};
```

Dieses Konfigurationsobjekt hat 2 Attribute

- 1. _id: der Name des Replikatsatzes ("s0")
  2. Members: [] (Members ist ein Array von Mongod-Instanzen. Lassen Sie dieses Feld zunächst leer, fügen Sie Mitglieder über den Push-Befehl hinzu.)

3.b Push (Hinzufügen) von Mongod-Instanzen zum Members-Array im Config-Objekt. Auf dem Mongo-Muscheltyp

```
config.members.push({"_id":0,"host":"localhost:37017"});
config.members.push({"_id":1,"host":"localhost:37018"});
config.members.push({"_id":2,"host":"localhost:37019"});
```

Wir weisen jeder Mongod-Instanz eine _id und einen Host zu. _id kann eine eindeutige Nummer sein, und der Host sollte der Hostname des Servers sein, auf dem er ausgeführt wird, gefolgt von der Portnummer.

4. Initiieren Sie das config-Objekt mit dem folgenden Befehl in der Mongo-Shell.

```
rs.initiate(config)
```

5. Nach ein paar Sekunden haben wir einen Replikatsatz von 3 Mongod-Instanzen, die auf dem Server ausgeführt werden. Geben Sie den folgenden Befehl ein, um den Status des Replikatsatzes zu überprüfen und um festzustellen, welcher primärer und welcher sekundärer ist.

```
rs.status();
```

Mongo als Nachbildung online lesen: https://riptutorial.com/de/mongodb/topic/6603/mongo-als-nachbildung

Kapitel 17: Mongo als Nachbildung

Examples

Überprüfen Sie die Status der MongoDB-Replikatsätze

Verwenden Sie den folgenden Befehl, um den Status der Replikatsätze zu überprüfen.

Befehl : rs.status ()

Verbinden Sie ein beliebiges Replikatsmitglied und geben Sie diesen Befehl aus, um den vollständigen Status des Replikatsatzes anzuzeigen

Beispiel:

```
{
 "set" : "ReplicaName",
 "date" : ISODate("2016-09-26T07:36:04.935Z"),
 "myState" : 1,
 "term" : NumberLong(-1),
 "heartbeatIntervalMillis" : NumberLong(2000),
 "members" : [
        {
                "_id" : 0,
                "name" : "<IP>:<PORT>,
                "health" : 1,
                "state" : 1,
                "stateStr" : "PRIMARY",
                "uptime" : 5953744,
                "optime" : Timestamp(1474875364, 36),
                "optimeDate" : ISODate("2016-09-26T07:36:04Z"),
                "electionTime" : Timestamp(1468921646, 1),
                "electionDate" : ISODate("2016-07-19T09:47:26Z"),
                "configVersion" : 6,
                "self" : true
        },
        {
                "_id" : 1,
                "name" : "<IP>:<PORT>",
                "health" : 1,
                "state" : 2,
                "stateStr" : "SECONDARY",
                "uptime" : 5953720,
                "optime" : Timestamp(1474875364, 13),
                "optimeDate" : ISODate("2016-09-26T07:36:04Z"),
                "lastHeartbeat" : ISODate("2016-09-26T07:36:04.244Z"),
                "lastHeartbeatRecv" : ISODate("2016-09-26T07:36:03.871Z"),
                "pingMs" : NumberLong(0),
                "syncingTo" : "10.9.52.55:10050",
                "configVersion" : 6
        },
        {
                "_id" : 2,
                "name" : "<IP>:<PORT>",
                "health" : 1,
```

```
                "state" : 7,
                "stateStr" : "ARBITER",
                "uptime" : 5953696,
                "lastHeartbeat" : ISODate("2016-09-26T07:36:03.183Z"),
                "lastHeartbeatRecv" : ISODate("2016-09-26T07:36:03.715Z"),
                "pingMs" : NumberLong(0),
                "configVersion" : 6
        },
        {
                "_id" : 3,
                "name" : "<IP>:<PORT>",
                "health" : 1,
                "state" : 2,
                "stateStr" : "SECONDARY",
                "uptime" : 1984305,
                "optime" : Timestamp(1474875361, 16),
                "optimeDate" : ISODate("2016-09-26T07:36:01Z"),
                "lastHeartbeat" : ISODate("2016-09-26T07:36:02.921Z"),
                "lastHeartbeatRecv" : ISODate("2016-09-26T07:36:03.793Z"),
                "pingMs" : NumberLong(22),
                "lastHeartbeatMessage" : "syncing from: 10.9.52.56:10050",
                "syncingTo" : "10.9.52.56:10050",
                "configVersion" : 6
        }
    ],
    "ok" : 1
    }
```

Aus den obigen Angaben können wir den gesamten Status des Replikatsatzes ermitteln

Mongo als Nachbildung online lesen: https://riptutorial.com/de/mongodb/topic/7043/mongo-als-nachbildung

Kapitel 18: Mongo als Scherben

Examples

Sharding-Umgebung einrichten

Mitglieder der Sharding Group:

Für das Scherben gibt es drei Spieler.

1. Config Server
2. Replikatsätze
3. Mongos

   Für einen Mongo-Shard müssen wir die drei oben genannten Server einrichten.

Config Server Setup: Fügen Sie der Mongod-Conf-Datei Folgendes hinzu

```
sharding:
  clusterRole: configsvr
replication:
  replSetName: <setname>
```

run: mongod --config

Wir können den Konfigurationsserver als Replikatsatz auswählen oder einen eigenständigen Server verwenden. Basierend auf unserer Anforderung können wir das Beste auswählen. Wenn die Konfiguration in einem Replikatsatz ausgeführt werden muss, müssen wir die Einrichtung des Replikatsatzes befolgen

Replikatsetup: Replikatsatz erstellen // Bitte beziehen Sie sich auf das Replikatsetup

MongoS-Setup: Mongos ist das Haupt-Setup im Shard. Es ist ein Abfrage-Router für den Zugriff auf alle Replikatsätze

Fügen Sie Folgendes in der Mongos-Conf-Datei hinzu

```
    sharding:
      configDB: <configReplSetName>/cfg1.example.net:27017;
```

Freigegeben konfigurieren:

Verbinden Sie die Mongos via Shell (Mongo - Host - Port)

1. sh.addShard ("/s1-mongo1.example.net:27017")
2. sh.enableSharding ("")

3. sh.shardCollection ("<Datenbank>. <Sammlung>", {<Schlüssel>: <Richtung>})
4. sh.status () // Um das Sharding zu gewährleisten

Mongo als Scherben online lesen: https://riptutorial.com/de/mongodb/topic/7044/mongo-als-scherben

Kapitel 19: MongoDB - Konfigurieren Sie ein ReplicaSet zur Unterstützung von TLS / SSL

Einführung

Wie konfiguriere ich ein ReplicaSet zur Unterstützung von TLS / SSL?

Wir werden ein ReplicaSet mit 3 Knoten in Ihrer lokalen Umgebung bereitstellen und ein selbstsigniertes Zertifikat verwenden. Verwenden Sie kein selbstsigniertes Zertifikat in PRODUCTION.

Wie verbinden Sie Ihren Client mit diesem ReplicaSet?

Wir werden eine Mongo-Shell anschließen.

Eine Beschreibung von TLS / SSL-, PKI-Zertifikaten (Public Key Infrastructure) und der Zertifizierungsstelle geht über den Umfang dieser Dokumentation hinaus.

Examples

Wie konfiguriere ich ein ReplicaSet zur Unterstützung von TLS / SSL?

Erstellen Sie das Stammzertifikat

Das Stammzertifikat (auch als CA-Datei bezeichnet) wird zum Signieren und Identifizieren Ihres Zertifikats verwendet. Führen Sie den folgenden Befehl aus, um es zu generieren.

```
openssl req -nodes -out ca.pem -new -x509 -keyout ca.key
```

Bewahren Sie das Stammzertifikat und seinen Schlüssel sorgfältig auf. Beide werden zum Signieren Ihrer Zertifikate verwendet. Das Stammzertifikat wird möglicherweise auch von Ihrem Client verwendet.

Generieren Sie die Zertifikatsanforderungen und die privaten Schlüssel

Geben Sie beim Generieren der Zertifikatsignierungsanforderung (aka CSR) **den genauen Hostnamen (oder die IP-Adresse) Ihres Knotens in das Feld Common Name (auch CN) ein. Die anderen Felder müssen exakt denselben Wert haben.** Möglicherweise müssen Sie Ihre Datei */ etc / hosts* ändern.

Mit den folgenden Befehlen werden die CSR-Dateien und die privaten RSA-Schlüssel (4096 Bit)

generiert.

```
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_node_1.key -out mongodb_node_1.csr
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_node_2.key -out mongodb_node_2.csr
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_node_3.key -out mongodb_node_3.csr
```

Sie müssen eine CSR für jeden Knoten Ihres ReplicaSet generieren. Denken Sie daran, dass der allgemeine Name von Knoten zu Knoten nicht identisch ist. Basis mehrerer CSRs nicht auf demselben privaten Schlüssel.

Sie müssen jetzt 3 CSRs und 3 private Schlüssel haben.

```
mongodb_node_1.key - mongodb_node_2.key - mongodb_node_3.key
mongodb_node_1.csr - mongodb_node_2.csr - mongodb_node_3.csr
```

Unterschreiben Sie Ihre Zertifikatsanfragen

Verwenden Sie die zuvor generierte CA-Datei (ca.pem) und ihren privaten Schlüssel (ca.key), um jede Zertifikatsanforderung zu signieren, indem Sie die folgenden Befehle ausführen.

```
openssl x509 -req -in mongodb_node_1.csr -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_node_1.crt
openssl x509 -req -in mongodb_node_2.csr -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_node_2.crt
openssl x509 -req -in mongodb_node_3.csr -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_node_3.crt
```

Sie müssen jede CSR unterzeichnen.

Sie müssen jetzt über 3 CSRs, 3 private Schlüssel und 3 selbstsignierte Zertifikate verfügen. Nur die privaten Schlüssel und die Zertifikate werden von MongoDB verwendet.

```
mongodb_node_1.key - mongodb_node_2.key - mongodb_node_3.key
mongodb_node_1.csr - mongodb_node_2.csr - mongodb_node_3.csr
mongodb_node_1.crt - mongodb_node_2.crt - mongodb_node_3.crt
```

Jedes Zertifikat entspricht einem Knoten. Erinnern Sie sich sorgfältig, welchen CN / Hostnamen Sie für jede CSR angegeben haben.

Concat jedes Knoten-Zertifikat mit seinem Schlüssel

Führen Sie die folgenden Befehle aus, um jedes Knotenzertifikat mit seinem Schlüssel in einer Datei zusammenzufassen (MongoDB-Anforderung).

```
cat mongodb_node_1.key mongodb_node_1.crt > mongodb_node_1.pem
cat mongodb_node_2.key mongodb_node_2.crt > mongodb_node_2.pem
cat mongodb_node_3.key mongodb_node_3.crt > mongodb_node_3.pem
```

Ihr muss jetzt 3 PEM-Dateien haben.

```
mongodb_node_1.pem - mongodb_node_2.pem - mongodb_node_3.pem
```

Stellen Sie Ihr ReplicaSet bereit

Wir gehen davon aus, dass sich Ihre PEM-Dateien in Ihrem aktuellen Ordner befinden sowie Daten / Daten1, Daten / Daten2 und Daten / Daten3.

Führen Sie die folgenden Befehle aus, um Ihre 3 Nodes ReplicaSet-Überwachung auf Port 27017, 27018 und 27019 bereitzustellen.

```
mongod --dbpath data/data_1 --replSet rs0 --port 27017 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_1.pem
mongod --dbpath data/data_2 --replSet rs0 --port 27018 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_2.pem
mongod --dbpath data/data_3 --replSet rs0 --port 27019 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_3.pem
```

Sie haben jetzt einen ReplicaSet mit 3 Knoten, der in Ihrer lokalen Umgebung bereitgestellt wird, und alle ihre Transaktionen werden verschlüsselt. Sie können keine Verbindung zu diesem ReplicaSet herstellen, ohne TLS zu verwenden.

Stellen Sie Ihr ReplicaSet für gegenseitiges SSL / gegenseitiges Vertrauen bereit

Um den Client zur Bereitstellung eines Client-Zertifikats (Gegenseitiges SSL) zu zwingen, müssen Sie die CA-Datei hinzufügen, wenn Sie Ihre Instanzen ausführen.

```
mongod --dbpath data/data_1 --replSet rs0 --port 27017 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_1.pem --sslCAFile ca.pem
mongod --dbpath data/data_2 --replSet rs0 --port 27018 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_2.pem --sslCAFile ca.pem
mongod --dbpath data/data_3 --replSet rs0 --port 27019 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_3.pem --sslCAFile ca.pem
```

Sie haben jetzt einen ReplicaSet mit 3 Knoten, der in Ihrer lokalen Umgebung bereitgestellt wird, und alle ihre Transaktionen werden verschlüsselt. Sie können keine Verbindung zu diesem ReplicaSet herstellen, ohne TLS zu verwenden oder ein von Ihrer Zertifizierungsstelle vertrauenswürdiges Clientzertifikat bereitzustellen.

Wie verbinden Sie Ihren Client (Mongo Shell) mit einem ReplicaSet?

Kein gegenseitiges SSL

In diesem Beispiel verwenden wir möglicherweise die CA-Datei (ca.pem), die Sie im Abschnitt "

So konfigurieren Sie ein ReplicaSet zur Unterstützung von TLS / SSL? " *Generiert haben* . Wir gehen davon aus, dass sich die CA-Datei in Ihrem aktuellen Ordner befindet.

Wir gehen davon aus, dass Ihre 3 Knoten auf Mongo1: 27017, Mongo2: 27018 und Mongo3: 27019 laufen. (Möglicherweise müssen Sie Ihre Datei */ etc / hosts* ändern.)

Wenn Ihre CA-Datei in MongoDB 3.2.6 im Trust Store des Betriebssystems registriert ist, können Sie eine Verbindung zu Ihrem ReplicaSet herstellen, ohne die CA-Datei bereitzustellen.

```
mongo --ssl --host rs0/mongo1:27017,mongo2:27018,mongo3:27019
```

Andernfalls müssen Sie die CA-Datei bereitstellen.

```
mongo --ssl --sslCAFile ca.pem --host rs0/mongo1:27017,mongo2:27018,mongo3:27019
```

Sie sind jetzt mit Ihrem ReplicaSet verbunden und alle Transaktionen zwischen Ihrer Mongo Shell und Ihrem ReplicaSet sind verschlüsselt.

Mit gegenseitigem SSL

Wenn Ihr ReplicaSet ein Client-Zertifikat verlangt, müssen Sie ein von der Zertifizierungsstelle signiertes Zertifikat bereitstellen, die von der ReplicaSet-Bereitstellung verwendet wird. Die Schritte zum Generieren des Client-Zertifikats sind fast identisch mit denen zum Generieren des Server-Zertifikats.

Sie müssen lediglich das Feld für den allgemeinen Namen während der CSR-Erstellung ändern. Anstelle eines Hostnamens mit einem Knoten im Feld "Allgemeiner Name" müssen **Sie alle durch Kommas getrennten ReplicaSet-Hostnamen angeben** .

```
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_client.key -out mongodb_client.csr
...
Common Name (e.g. server FQDN or YOUR name) []: mongo1,mongo2,mongo3
```

Wenn das Feld Common Name zu lang ist (mehr als 64 Byte), kann die Größenbeschränkung für Common Name auftreten. Um diese Einschränkung zu umgehen, müssen Sie beim Generieren der CSR den SubjectAltName verwenden.

```
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_client.key -out mongodb_client.csr
-config <(
cat <<-EOF
[req]
default_bits = 4096
prompt = no
default_md = sha256
req_extensions = req_ext
distinguished_name = dn

[ dn ]
CN = .
```

```
[ req_ext ]
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = mongo1
DNS.2 = mongo2
DNS.3 = mongo3
EOF
)
```

Dann signieren Sie die CSR mit dem CA-Zertifikat und dem Schlüssel.

```
openssl x509 -req -in mongodb_client.csr  -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_client.crt
```

Zum Schluss legen Sie den Schlüssel und das signierte Zertifikat fest.

```
cat mongodb_client.key mongodb_client.crt > mongodb_client.pem
```

Um eine Verbindung zu Ihrem ReplicaSet herzustellen, können Sie jetzt das neu generierte Clientzertifikat bereitstellen.

```
mongo --ssl --sslCAFile ca.pem --host rs0/mongo1:27017,mongo2:27018,mongo3:27019 --
sslPEMKeyFile mongodb_client.pem
```

Sie sind jetzt mit Ihrem ReplicaSet verbunden und alle Transaktionen zwischen Ihrer Mongo Shell und Ihrem ReplicaSet sind verschlüsselt.

MongoDB - Konfigurieren Sie ein ReplicaSet zur Unterstützung von TLS / SSL online lesen: https://riptutorial.com/de/mongodb/topic/9539/mongodb---konfigurieren-sie-ein-replicaset-zur-unterstutzung-von-tls---ssl

Kapitel 20: MongoDB verwalten

Examples

Auflistung der aktuell laufenden Abfragen

Der folgende Befehl listet die Abfragen auf, die derzeit auf dem Server ausgeführt werden

```
db.currentOp()
```

Die Ausgabe sieht ähnlich aus

```
{
    "inprog" : [
        {
            "opid" : "302616759",
            "active" : true,
            "secs_running" : 1,
            "microsecs_running" : NumberLong(1167662),
            "op" : "getmore",
            "ns" : "local.oplog.rs",
            "query" : {

            },
            ...
        },
        {
            "desc" : "conn48",
            "threadId" : "0x114c00700",
            "connectionId" : 48,
            "opid" : "mdss_shard00:302616760",
            "active" : true,
            "secs_running" : 1,
            "microsecs_running" : NumberLong(1169659),
            "op" : "getmore",
            "ns" : "local.oplog.rs"
            ...
        }
    ]
}
```

Das `inprog` Attribut gibt an, dass die Abfragen derzeit ausgeführt werden. Die `opid` ist die `opid` der Abfrage oder des Vorgangs. `secs_running` gibt die Zeit an, für die es läuft. Dies ist manchmal hilfreich, um lange laufende Abfragen zu identifizieren.

MongoDB verwalten online lesen: https://riptutorial.com/de/mongodb/topic/7553/mongodb-verwalten

Kapitel 21: MongoDB-Aggregation

Examples

Aggregierte Abfragebeispiele, die für Arbeit und Lernen nützlich sind

Aggregation wird verwendet, um komplexe Datensuchoperationen in der Mongo-Abfrage auszuführen, die bei normalen "find" -abfragen nicht möglich sind.

Erstellen Sie einige Dummy-Daten:

```
db.employees.insert({"name":"Adma","dept":"Admin","languages":["german","french","english","hindi"],"ag
"totalExp":10});
db.employees.insert({"name":"Anna","dept":"Admin","languages":["english","hindi"],"age":35,
"totalExp":11});
db.employees.insert({"name":"Bob","dept":"Facilities","languages":["english","hindi"],"age":36,
"totalExp":14});
db.employees.insert({"name":"Cathy","dept":"Facilities","languages":["hindi"],"age":31,
"totalExp":4});
db.employees.insert({"name":"Mike","dept":"HR","languages":["english", "hindi",
"spanish"],"age":26, "totalExp":3});
db.employees.insert({"name":"Jenny","dept":"HR","languages":["english", "hindi",
"spanish"],"age":25, "totalExp":3});
```

Beispiele nach Thema:

1. Übereinstimmung **:** Wird verwendet, um Dokumente abzugleichen (wie SQL-Klausel where)

```
db.employees.aggregate([{$match:{dept:"Admin"}}]);
Output:
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "Admin", "languages"
: [ "german", "french", "english", "hindi" ], "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fc92e9b4b54ec384a0e"), "name" : "Anna", "dept" : "Admin", "languages"
: [ "english", "hindi" ], "age" : 35, "totalExp" : 11 }
```

2. Projekt: Dient zum Auffüllen bestimmter Feldwerte

Das Projektstadium enthält automatisch das Feld _id, sofern Sie dies nicht deaktivieren.

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}]);
Output:
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "Admin" }
{ "_id" : ObjectId("54982fc92e9b4b54ec384a0e"), "name" : "Anna", "dept" : "Admin" }

db.employees.aggregate({$project: {'_id':0, 'name': 1}})
Output:
{ "name" : "Adma" }
{ "name" : "Anna" }
{ "name" : "Bob" }
{ "name" : "Cathy" }
{ "name" : "Mike" }
{ "name" : "Jenny" }
```

3. Group: $ group wird verwendet, um Dokumente nach bestimmten Feldern zu gruppieren. Hier werden die Dokumente nach dem Wert des Feldes "dept" gruppiert. Eine weitere nützliche Funktion ist, dass Sie nach Null gruppieren können. Dies bedeutet, dass alle Dokumente zu einem Dokument zusammengefasst werden.

```
db.employees.aggregate([{$group:{"_id":"$dept"}}]);


{ "_id" : "HR" }

{ "_id" : "Facilities" }

{ "_id" : "Admin" }

db.employees.aggregate([{$group:{"_id":null, "totalAge":{$sum:"$age"}}}]);
Output:
{ "_id" : null, "noOfEmployee" : 183 }
```

4. Summe: $ sum wird verwendet, um die Werte innerhalb einer Gruppe zu zählen oder zu summieren.

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfDept":{$sum:1}}}]);
Output:
{ "_id" : "HR", "noOfDept" : 2 }
{ "_id" : "Facilities", "noOfDept" : 2 }
{ "_id" : "Admin", "noOfDept" : 2 }
```

5. Durchschnitt: Berechnet den Durchschnitt des Feldwerts eines bestimmten Feldes pro Gruppe.

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"avgExp":{$avg:"$totalExp"}}}]);
Output:
{ "_id" : "HR", "noOfEmployee" : 2, "totalExp" : 3 }
{ "_id" : "Facilities", "noOfEmployee" : 2, "totalExp" : 9 }
{ "_id" : "Admin", "noOfEmployee" : 2, "totalExp" : 10.5 }
```

6. Minimum: Sucht den Mindestwert eines Feldes in jeder Gruppe.

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"minExp":{$min:"$totalExp"}}}]);
Output:
{ "_id" : "HR", "noOfEmployee" : 2, "totalExp" : 3 }
{ "_id" : "Facilities", "noOfEmployee" : 2, "totalExp" : 4 }
{ "_id" : "Admin", "noOfEmployee" : 2, "totalExp" : 10 }
```

7. Maximum: Sucht den maximalen Wert eines Feldes in jeder Gruppe.

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"maxExp":{$max:"$totalExp"}}}]);
Output:
{ "_id" : "HR", "noOfEmployee" : 2, "totalExp" : 3 }
{ "_id" : "Facilities", "noOfEmployee" : 2, "totalExp" : 14 }
{ "_id" : "Admin", "noOfEmployee" : 2, "totalExp" : 11 }
```

8. Abrufen eines bestimmten Feldwerts **aus dem ersten und dem letzten Dokument jeder Gruppe:** Funktioniert gut, wenn das Doppelergebnis sortiert wird.

```
db.employees.aggregate([{$group:{"_id":"$age", "lasts":{$last:"$name"},
"firsts":{$first:"$name"}}}]);
Output:
{ "_id" : 25, "lasts" : "Jenny", "firsts" : "Jenny" }
{ "_id" : 26, "lasts" : "Mike", "firsts" : "Mike" }
{ "_id" : 35, "lasts" : "Cathy", "firsts" : "Anna" }
{ "_id" : 30, "lasts" : "Adma", "firsts" : "Adma" }
```

9. Minimum mit maximal:

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"maxExp":{$max:"$totalExp"}, "minExp":{$min: "$totalExp"}}}]);
Output:
{ "_id" : "HR", "noOfEmployee" : 2, "maxExp" : 3, "minExp" : 3 }
{ "_id" : "Facilities", "noOfEmployee" : 2, "maxExp" : 14, "minExp" : 4 }
{ "_id" : "Admin", "noOfEmployee" : 2, "maxExp" : 11, "minExp" : 10 }
```

10. Push und addToSet: Push fügt einem Feld einen Wert aus jedem Dokument in einer Gruppe einem Array hinzu, das zum Projizieren von Daten im Array-Format verwendet wird.

```
db.employees.aggregate([{$group:{"_id":"dept", "arrPush":{$push:"$age"}, "arrSet":
{$addToSet:"$age"}}}]);
Output:
{ "_id" : "dept", "arrPush" : [ 30, 35, 35, 35, 26, 25 ], "arrSet" : [ 25, 26, 35, 30 ] }
```

11. Abwickeln: Wird verwendet, um mehrere im Arbeitsspeicher befindliche Dokumente für jeden Wert im angegebenen Feld des Array-Typs zu erstellen. Auf dieser Grundlage können wir weitere Aggregationen vornehmen.

```
db.employees.aggregate([{$match:{"name":"Adma"}}, {$unwind:"$languages"}]);
Output:
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "HR", "languages" :
"german", "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "HR", "languages" :
"french", "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "HR", "languages" :
"english", "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "HR", "languages" :
"hindi", "age" : 30, "totalExp" : 10 }
```

12. Sortierung:

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}, {$sort:
{name: 1}}]);
Output:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin" }
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin" }

db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}, {$sort:
{name: -1}}]);
Output:
```

```
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin" }
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin" }
```

13. Überspringen:

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}, {$sort:
{name: -1}}, {$skip:1}]);
Output:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin" }
```

14. Grenze:

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}, {$sort:
{name: -1}}, {$limit:1}]);
Output:

{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin" }
```

15. Vergleichsoperator in Projektion:

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1, age: {$gt:
["$age", 30]}}}]);
Output:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin", "age" :
false }
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin", "age" :
true }
```

16. Vergleichsoperator in Übereinstimmung:

```
db.employees.aggregate([{$match:{dept:"Admin", age: {$gt:30}}}, {$project:{"name":1,
"dept":1}}]);
Output:
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin" }
```

Liste der Vergleichsoperatoren: $ cmp, $ eq, $ gt, $ gte, $ lt, $ lte und $ ne

17. Boolescher Aggregationsoperator in Projektion:

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1, age: { $and: [
{ $gt: [ "$age", 30 ] }, { $lt: [ "$age", 36 ] } ] }}}]);

Output:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin", "age" :
false }
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin", "age" :
true }
```

18. Boolescher Aggregationsoperator in Übereinstimmung:

```
db.employees.aggregate([{$match:{dept:"Admin", $and: [{age: { $gt:  30 }}, {age: {$lt: 36 }} ]
}}, {$project:{"name":1, "dept":1, age: { $and: [ { $gt: [ "$age", 30 ] }, { $lt: [ "$age", 36
] } ] }}}]);
```

```
Output:
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin", "age" :
true }
```

Liste der booleschen Aggregationsoperatoren: $ und, $ oder, und $ not.

Vollständige Information: https://docs.mongodb.com/v3.2/reference/operator/aggregation/

Java und Spring Beispiel

Dies ist ein Beispielcode zum Erstellen und Ausführen der Aggregatabfrage in MongoDB mit Spring Data.

```
    try {
        MongoClient mongo = new MongoClient();
        DB db = mongo.getDB("so");
        DBCollection coll = db.getCollection("employees");

        //Equivalent to $match
        DBObject matchFields = new BasicDBObject();
        matchFields.put("dept", "Admin");
        DBObject match = new BasicDBObject("$match", matchFields);

        //Equivalent to $project
        DBObject projectFields = new BasicDBObject();
        projectFields.put("_id", 1);
        projectFields.put("name", 1);
        projectFields.put("dept", 1);
        projectFields.put("totalExp", 1);
        projectFields.put("age", 1);
        projectFields.put("languages", 1);
        DBObject project = new BasicDBObject("$project", projectFields);

        //Equivalent to $group
        DBObject groupFields = new BasicDBObject("_id", "$dept");
        groupFields.put("ageSet", new BasicDBObject("$addToSet", "$age"));
        DBObject employeeDocProjection = new BasicDBObject("$addToSet", new
BasicDBObject("totalExp", "$totalExp").append("age", "$age").append("languages",
"$languages").append("dept", "$dept").append("name", "$name"));
        groupFields.put("docs", employeeDocProjection);
        DBObject group = new BasicDBObject("$group", groupFields);

        //Sort results by age
        DBObject sort = new BasicDBObject("$sort", new BasicDBObject("age", 1));

        List<DBObject> aggregationList = new ArrayList<>();
        aggregationList.add(match);
        aggregationList.add(project);
        aggregationList.add(group);
        aggregationList.add(sort);
        AggregationOutput output = coll.aggregate(aggregationList);

        for (DBObject result : output.results()) {
            BasicDBList employeeList = (BasicDBList) result.get("docs");
            BasicDBObject employeeDoc = (BasicDBObject) employeeList.get(0);
            String name = employeeDoc.get("name").toString();
            System.out.println(name);
        }
```

```
    }catch (Exception ex){
        ex.printStackTrace();
    }
```

Sehen Sie den Wert "resultSet" im JSON-Format, um das Ausgabeformat zu verstehen:

```
[{
    "_id": "Admin",
    "ageSet": [35.0, 30.0],
    "docs": [{
        "totalExp": 11.0,
        "age": 35.0,
        "languages": ["english", "hindi"],
        "dept": "Admin",
        "name": "Anna"
    }, {
        "totalExp": 10.0,
        "age": 30.0,
        "languages": ["german", "french", "english", "hindi"],
        "dept": "Admin",
        "name": "Adma"
    }]
}]
```

Das "resultSet" enthält einen Eintrag für jede Gruppe, "ageSet" enthält die Altersliste aller Angestellten dieser Gruppe, "_id" enthält den Wert des Feldes, das zur Gruppierung verwendet wird, und "docs" enthält Daten zu jedem Angestellten dieser Gruppe, die in unserem eigenen Code und unserer Benutzeroberfläche verwendet werden kann.

Holen Sie sich Beispieldaten

Um zufällige Daten aus einer bestimmten Sammlung zu erhalten, beziehen Sie sich auf die `$sample` Aggregation.

```
db.emplyees.aggregate({ $sample: { size:1 } })
```

wobei `size` für die Anzahl der auszuwählenden Elemente steht.

Left Outer Join mit Aggregation ($ Lookup)

```
let col_1 = db.collection('col_1');
let col_2 = db.collection('col_2');
col_1 .aggregate([
    { $match: { "_id": 1 } },
    {
        $lookup: {
            from: "col_2",
            localField: "id",
            foreignField: "id",
            as: "new_document"
        }
    }
],function (err, result){
    res.send(result);
```

```
});
```

Diese Funktion wurde in der mongodb- **Version 3.2** neu veröffentlicht. Dadurch erhält der Benutzer die Möglichkeit, eine Sammlung mit den entsprechenden Attributen einer anderen Sammlung zu verbinden

Mongodb $ LookUp-Dokumentation

MongoDB-Aggregation online lesen: https://riptutorial.com/de/mongodb/topic/7417/mongodb-aggregation

Kapitel 22: MongoDB-Autorisierungsmodell

Einführung

Die Berechtigung ist die Überprüfung der Benutzerrechte. MongoDB unterstützt verschiedene Arten von Autorisierungsmodellen. 1. **Rollenbasiszugriffskontrolle**
 Rolle ist eine Gruppe von Berechtigungen, Aktionen über Ressourcen. Das ist ein Gewinn für Benutzer über einen bestimmten Namespace (Datenbank). Aktionen werden für Ressourcen ausgeführt. Ressourcen sind alle Objekte, die den Status in der Datenbank haben.

Examples

Integrierte Rollen

In jeder Datenbank sind integrierte Datenbankbenutzer- und Datenbankadministrationsrollen vorhanden.

Datenbank-Benutzerrollen

1. `read`
2. `readwrite`

MongoDB-Autorisierungsmodell online lesen:
https://riptutorial.com/de/mongodb/topic/8114/mongodb-autorisierungsmodell

Kapitel 23: Python-Treiber

Syntax

- mongodb: // [Benutzername: Kennwort @] Host1 [: Port1] [, Host2 [: Port2], ... [, HostN [: PortN]]] [/ [Datenbank] [? Optionen]]

Parameter

Parameter	Detail
hostX	Wahlweise. Sie können beliebig viele Hosts angeben. Sie würden beispielsweise mehrere Hosts für Verbindungen zu Replikatsätzen angeben.
: portX	Wahlweise. Der Standardwert ist: 27017, falls nicht angegeben.
Datenbank	Wahlweise. Der Name der zu authentifizierenden Datenbank, wenn die Verbindungszeichenfolge Authentifizierungsinformationen enthält. Wenn / database nicht angegeben ist und die Verbindungszeichenfolge Berechtigungsnachweise enthält, authentifiziert sich der Treiber bei der Verwaltungsdatenbank.
?Optionen	Verbindungsspezifische Optionen

Examples

Verbinden Sie sich über Pymongo mit MongoDB

```
from pymongo import MongoClient

uri = "mongodb://localhost:27017/"

client = MongoClient(uri)

db = client['test_db']
# or
# db = client.test_db

# collection = db['test_collection']
# or
collection = db.test_collection


collection.save({"hello":"world"})

print collection.find_one()
```

PyMongo-Abfragen

Sobald Sie ein `collection` Objekt erhalten haben, verwenden Abfragen dieselbe Syntax wie in der Mongo-Shell. Einige geringfügige Unterschiede sind:

- Jeder Schlüssel muss in Klammern stehen. Zum Beispiel:

  ```
  db.find({frequencies: {$exists: true}})
  ```

  wird in `pymongo` (beachten Sie das `True` in Großbuchstaben):

  ```
  db.find({"frequencies": { "$exists": True }})
  ```

- Objekte wie Objekt-IDs oder `ISODate` werden mit Python-Klassen `ISODate` . PyMongo verwendet seine eigene `ObjectId` Klasse, um mit Objekt-IDs zu arbeiten, während Datumsangaben das Standard- `datetime` Paket verwenden. Wenn Sie beispielsweise alle Ereignisse zwischen 2010 und 2011 abfragen möchten, können Sie Folgendes tun:

  ```
  from datetime import datetime

  date_from = datetime(2010, 1, 1)
  date_to = datetime(2011, 1, 1)
  db.find({ "date": { "$gte": date_from, "$lt": date_to } }):
  ```

Aktualisieren Sie alle Dokumente in einer Sammlung mit PyMongo

Angenommen, Sie müssen jedem Dokument in einer Sammlung ein Feld hinzufügen.

```
import pymongo

client = pymongo.MongoClient('localhost', 27017)
db = client.mydb.mycollection

for doc in db.find():
   db.update(
       {'_id': doc['_id']},
       {'$set': {'newField': 10} }, upsert=False, multi=False)
```

Die `find` Methode gibt einen `Cursor` , auf dem Sie die `for in` Syntax verwenden können. Dann rufen wir die `update` , geben die `_id` und fügen ein Feld (`$set`) hinzu. Die Parameter `upsert` und `multi` stammen von Mongodb (weitere Informationen finden Sie hier).

Python-Treiber online lesen: https://riptutorial.com/de/mongodb/topic/7843/python-treiber

Kapitel 24: Replikation

Examples

Grundkonfiguration mit drei Knoten

Der Replikatsatz ist eine Gruppe von Mongod-Instanzen, die denselben Datensatz verwalten.

Dieses Beispiel zeigt, wie ein Replikatsatz mit drei Instanzen auf demselben Server konfiguriert wird.

Datenordner erstellen

```
mkdir /srv/mongodb/data/rs0-0
mkdir /srv/mongodb/data/rs0-1
mkdir /srv/mongodb/data/rs0-2
```

Mongod-Instanzen starten

```
mongod --port 27017 --dbpath /srv/mongodb/data/rs0-0 --replSet rs0
mongod --port 27018 --dbpath /srv/mongodb/data/rs0-1 --replSet rs0
mongod --port 27019 --dbpath /srv/mongodb/data/rs0-2 --replSet rs0
```

Replikatsatz konfigurieren

```
mongo --port 27017    // connection to the instance 27017

rs.initiate();                // initilization of replica set on the 1st node
rs.add("<hostname>:27018")    // adding a 2nd node
rs.add("<hostname>:27019")    // adding a 3rd node
```

Testen Sie Ihr Setup

Um den Konfigurationstyp `rs.status()` überprüfen, sollte das Ergebnis folgendermaßen aussehen:

```
{
        "set" : "rs0",
        "date" : ISODate("2016-09-01T12:34:24.968Z"),
        "myState" : 1,
        "term" : NumberLong(4),
        "heartbeatIntervalMillis" : NumberLong(2000),
        "members" : [
                {
                        "_id" : 0,
                        "name" : "<hostname>:27017",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        ........................
                },
                {
```

```
                        "_id" : 1,
                        "name" : "<hostname>:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        ........................
                },
                {
                        "_id" : 2,
                        "name" : "<hostname>:27019",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        ........................
                }
        ],
        "ok" : 1
}
```

Replikation online lesen: https://riptutorial.com/de/mongodb/topic/6205/replikation

Kapitel 25: Sammlungen

Bemerkungen

Datenbank erstellen

Examples

Erstellen Sie eine Sammlung

Zuerst auswählen oder eine Datenbank erstellen.

```
> use mydb
switched to db mydb
```

Mit der Methode `db.createCollection("yourCollectionName")` können Sie Collection explizit erstellen.

```
> db.createCollection("newCollection1")
{ "ok" : 1 }
```

Mit dem Befehl `show collections` Sie alle Sammlungen in der Datenbank anzeigen.

```
> show collections
newCollection1
system.indexes
>
```

Die `db.createCollection()` -Methode hat die folgenden Parameter:

Parameter	Art	Beschreibung
Name	Schnur	Der Name der zu erstellenden Sammlung.
Optionen	dokumentieren	*Wahlweise.* Konfigurationsoptionen zum Erstellen einer begrenzten Sammlung oder zum Vorbelegen von Speicherplatz in einer neuen Sammlung.

Das folgende Beispiel zeigt die Syntax der `createCollection()` -Methode mit einigen wichtigen Optionen

```
>db.createCollection("newCollection4", {capped :true, autoIndexId : true, size : 6142800, max
: 10000})
{ "ok" : 1 }
```

Sowohl die `db.collection.insert()` als auch `db.collection.createIndex()` erstellen ihre jeweilige Auflistung, wenn sie noch nicht vorhanden ist.

```
> db.newCollection2.insert({name : "XXX"})
> db.newCollection3.createIndex({accountNo : 1})
```

Jetzt zeigen Sie alle Sammlungen mit dem Befehl " `show collections` anzeigen"

```
> show collections
newCollection1
newCollection2
newCollection3
newCollection4
system.indexes
```

Wenn Sie das eingefügte Dokument anzeigen möchten, verwenden Sie den Befehl `find()` .

```
> db.newCollection2.find()
{ "_id" : ObjectId("58f26876cabafaeb509e9c1f"), "name" : "XXX" }
```

Drop Collection

`db.collection.drop()` wird verwendet, um eine Sammlung aus der Datenbank zu `db.collection.drop()` .

Überprüfen Sie zuerst die verfügbaren Sammlungen in Ihre Datenbank `mydb` .

```
> use mydb
switched to db mydb

> show collections
newCollection1
newCollection2
newCollection3
system.indexes
```

`newCollection1` nun die Sammlung mit dem Namen `newCollection1` .

```
> db.newCollection1.drop()
true
```

Hinweis: Wenn die Sammlung erfolgreich gelöscht wurde, gibt die Methode `true` andernfalls `false` .

Überprüfen Sie erneut die Liste der Sammlungen in der Datenbank.

```
> show collections
newCollection2
newCollection3
system.indexes
```

Referenz: MongoDB-Methode drop () .

Sammlungen online lesen: https://riptutorial.com/de/mongodb/topic/9732/sammlungen

Kapitel 26: Steckbare Storage Engines

Bemerkungen

In MongoDB 3.0 sind MMAP (Standard) und WiredTiger die stabilen Speicher-Engines. Normalerweise verwenden Sie MMAP, wenn Ihre App lesebereit ist. Wenn es sehr schwer ist, verwenden Sie WiredTiger.

Ihre Lösung verfügt möglicherweise auch über gemischte Replikatsatzmitglieder, bei denen Sie einen Knoten mit MMAP und einen anderen mit WiredTiger konfigurieren können. Sie können eine verwenden, um umfangreiche Daten einzufügen, und die andere, um sie mit analytischen Werkzeugen zu lesen.

Nach MongoDB 3.2 wird WiredTiger zur Standard-Engine.

Examples

MMAP

MMAP ist eine steckbare Speicher-Engine, die nach dem Befehl `mmap()` Linux benannt wurde. Es ordnet Dateien dem virtuellen Speicher zu und optimiert Leseaufrufe. Wenn Sie eine große Datei haben, aber nur einen kleinen Teil davon lesen müssen, ist `mmap()` viel schneller als ein `read()` Aufruf, der die gesamte Datei in den Speicher bringt.

Ein Nachteil besteht darin, dass nicht zwei Schreibaufrufe für dieselbe Sammlung parallel verarbeitet werden können. Daher verfügt MMAP über Sperren auf Sammlungsebene (und nicht wie von WiredTiger angebotene Sperren auf Dokumentebene). Diese Auflistungssperre ist erforderlich, da ein MMAP-Index auf mehrere Dokumente verweisen kann. Wenn diese Dokumente gleichzeitig aktualisiert werden könnten, wäre der Index inkonsistent.

WiredTiger

WiredTiger unterstützt **LSM-Bäume zum Speichern von Indizes** . LSM-Bäume sind schneller für Schreibvorgänge, wenn Sie große Arbeitslasten von zufälligen Einfügungen schreiben müssen.

In WiredTiger gibt es **keine** direkten **Updates** . Wenn Sie ein Element eines Dokuments aktualisieren müssen, wird ein neues Dokument eingefügt, während das alte Dokument gelöscht wird.

WiredTiger bietet auch **Parallelität auf Dokumentebene** . Es wird davon ausgegangen, dass sich zwei Schreibvorgänge nicht auf dasselbe Dokument auswirken. Andernfalls wird ein Vorgang zurückgespult und später ausgeführt. Das ist ein großer Leistungsschub, wenn Rückspulungen selten sind.

WiredTiger unterstützt die **Algorithmen Snappy und zLib zur Komprimierung** von Daten und Indizes im Dateisystem. Bissig ist die Standardeinstellung. Es ist weniger CPU-intensiv, hat jedoch

eine niedrigere Komprimierungsrate als zLib.

So verwenden Sie die WiredTiger Engine

```
mongod --storageEngine wiredTiger --dbpath <newWiredTigerDBPath>
```

Hinweis:

1. Nach mongodb 3.2 ist die Standard-Engine WiredTiger.
2. `newWiredTigerDBPath` sollte keine Daten einer anderen Speicher-Engine enthalten. Um Ihre Daten zu migrieren, müssen Sie sie sichern und erneut in die neue Speicher-Engine importieren.

```
mongodump --out <exportDataDestination>
mongod --storageEngine wiredTiger --dbpath <newWiredTigerDBPath>
mongorestore <exportDataDestination>
```

In Erinnerung

Alle Daten werden im Arbeitsspeicher (RAM) gespeichert, um das Lesen zu erleichtern.

Mongo-Felsen

Eine Schlüsselwertmaschine, die zur Integration mit Facebooks RocksDB entwickelt wurde.

Fusion-io

Eine von SanDisk erstellte Speicher-Engine, die es ermöglicht, die Dateisystemschicht des Betriebssystems zu umgehen und direkt auf das Speichergerät zu schreiben.

TokuMX

Eine von Percona erstellte Speicher-Engine, die fraktale Baumindizes verwendet

Steckbare Storage Engines online lesen: https://riptutorial.com/de/mongodb/topic/694/steckbare-storage-engines

Kapitel 27: Upserts und Inserts

Examples

Ein Dokument einfügen

_id ist eine 12-Byte-Hexadezimalzahl, die die Eindeutigkeit jedes Dokuments gewährleistet. Sie können _id angeben, während Sie das Dokument einfügen. **Wenn Sie nicht angegeben haben, stellt MongoDB für jedes Dokument eine eindeutige ID bereit.** Diese 12 Bytes, die ersten 4 Bytes für den aktuellen Zeitstempel, die nächsten 3 Bytes für die Maschinen-ID, die nächsten 2 Bytes für die Prozess-ID des Mongodb-Servers und die restlichen 3 Bytes sind einfache inkrementelle Werte.

```
db.mycol.insert({
 _id: ObjectId(7df78ad8902c),
 title: 'MongoDB Overview',
 description: 'MongoDB is no sql database',
 by: 'tutorials point',
 url: 'http://www.tutorialspoint.com',
 tags: ['mongodb', 'database', 'NoSQL'],
 likes: 100
})
```

Mycol ist hier ein Sammlungsname. Wenn die Sammlung nicht in der Datenbank vorhanden ist, erstellt MongoDB diese Sammlung und fügt ein Dokument in die Datenbank ein. Wenn im eingefügten Dokument kein *_id*- Parameter angegeben wird, weist MongoDB diesem Dokument eine eindeutige ObjectId zu.

Upserts und Inserts online lesen: https://riptutorial.com/de/mongodb/topic/10185/upserts-und-inserts

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit MongoDB	Abdul Rehman Sayed, Amir Rahimi Farahani, Ashari, Community, ipip, jengeb, manetsus, Peter Mortensen , Prosen Ghosh, Renukaradhya, Scott Weldon, Sean Reilly, Simulant, titogeo, WAF
2	2dsphere Index	gypsyCoder
3	Aktualisieren der MongoDB-Version	Antti_M
4	Aktualisieren Sie die Operatoren	yellowB
5	Anhäufung	grape, HoefMeistert, Lakmal Vithanage, LoicM, manetsus, RaR, steveinatorx, titogeo
6	Aufbau	Matt Clark
7	Authentifizierungsmechanismen in MongoDB	Luzan Baral, Niroshan Ranapathi
8	CRUD-Operation	fracz, Himavanth, Ishan Soni, Jain, jerry, JohnnyHK, Kelum Senanayake, KrisVos130, Lakmal Vithanage, Marco, Mayank Pandeyz, Prosen Ghosh, Renukaradhya, Rick, Rotem, Shrabanee, sstyvane, Thomas Bormans, Tomás Cañibano
9	Daten abfragen (Erste Schritte)	Avindu Hewa, oggo, Prosen Ghosh, SommerEngineering
10	Daten sichern und wiederherstellen	user641887
11	Datenbankinformationen abrufen	fracz
12	Indizes	Adam Comerford, Batsu, Constantin Guay, jerry, Juan Carlos Farah, manetsus, Nic Cottrell, RaR, Rick , Sarthak Adhikari, titogeo, Tomás Cañibano
13	Java-Treiber	dev ツ, Emil Burzo
14	Massenvorgänge	chridam

15	Mongo als Nachbildung	user641887
16	Mongo als Scherben	Selva Kumar
17	MongoDB - Konfigurieren Sie ein ReplicaSet zur Unterstützung von TLS / SSL	bappr
18	MongoDB verwalten	Ravi Chandra
19	MongoDB-Aggregation	Andrii Abramov, Avindu Hewa, Erdenezul, saurav
20	MongoDB-Autorisierungsmodell	Niroshan Ranapathi
21	Python-Treiber	Derlin, sergiuz
22	Replikation	ADIMO
23	Sammlungen	Prosen Ghosh
24	Steckbare Storage Engines	Constantin Guay, Jorge Aranda, tim, Zanon
25	Upserts und Inserts	Kuhan