MongoDB

MongoDB ist eine NoSQL-Datenbank, die zu den dokumentenorientierten Systemen zählt. Entwickelt wird sie von 10gen, einem Unternehmen in New York. Der Name leitet sich vom englischen Wort „humongous“ für „gigantisch“ ab. MongoDB bietet u.a. mit Dokumenten ein flexibles Datenmodell, gute Performance durch optimale Hauptspeichernutzung, dynamische Konsistentz bei Schreibzugriffen, horizontale Skalierbarkeit, sowie automatisches Sharding.

Es wird sowohl eine kostenlose Open-Source-Variante, als auch eine Enterprise-Version mit Support angeboten.

Aufbau

MongoDB ist eine dokumentenorientierte Datenbank, und verwaltet Datensätze entsprechend in Dokumenten. Diese sind BSON-Files (Binary JSON), eine binäre Form von JSON-Dokumenten, die es um einige Datentypen erweitert und ebenfalls von 10gen entwickelt wurde.

Auf einem Server können mehrere Datenbanken betrieben und unabhängig voneinander verwaltet werden. Diese werden dabei in **Collections** organisiert, die wiederum eine Sammlung von **Dokumenten** ist.

Ein **Dokument** entspricht einer Zeile in einer relationalen Datenbank, bestehend aus Schlüsselfeldern mit einem Wert. Anders als in relationalen Systemen folgen diese jedoch keinem festen Schema. Jedes Dokument kann seine eigene Struktur mit unterschiedlichen Attributen haben. Es ist jedoch sinnvoll, einem indirekten Schema zu folgen und die Dokumente einer Collection ähnlich aufzubauen.

Jedes Dokument besitzt eine **ObjectId** („_id“), die als Primärschlüssel der eindeutigen Identifizierung dient und auch bei Abfragen als Index hilfreich sein kann. Sie ist innerhalb einer Collection einzigartig und wird automatisch generiert, kann aber auch manuell gesetzt werden.

Die Größe eines Dokuments beschränkt sich auf 16MB. Das soll verhindern, dass ein File zu viel RAM- oder Netzwerk-Kapazitäten beansprucht. Dokumente, die größer sind, werden mit GridFS in kleinere Teile aufgespalten. (MongoDB o. J.)

Beispiel für ein Dokument:

```
{
    „_id“: 4711,
    „vorname“: “Max“,
    „nachname“: “Mustermann“,
    „adresse“: {    „plz“: 54321,
            „ort“: “Musterstadt“,
```

```
            „strasse“: “Musterweg 12“
           },
    „mag“: [„Katze“, „Hund“]
}
```

Um nun ein Dokument anzulegen, muss dafür die Datenbank und die Collection angegeben werden: `db.collection.insert(<Dokument oder Array von Dokumenten>)`. Dabei wird das **_id-Feld** automatisch angelegt und ein Wert generiert (u.a. abhängig von Maschine und dem Zeitpunkt), kann aber auch (wie in diesem Beispiel) manuell angelegt werden, dann muss aber auch darauf geachtet werden, dass dieser Wert einzigartig innerhalb der Collection bleibt. (MongoDB o. J.a)

Bei dem Beispiel handelt es sich um ein verschachteltes Dokument. Das „adresse“-Feld beinhaltet ein Adressobjekt mit eigenen Attributen. Das Feld „mag“ beinhaltet eine Liste aus beliebig vielen Elementen. Eine Liste kann auch aus mehreren Objekten bestehen und z.B. die Liefer- und Rechnungsadresse (ähnlich aufgebaut wie „adresse“) einer Person speichern. Ebenso gut könnte ein anderes Dokument an dieser Stelle aber auch nur „Hund“ beinhalten oder das Feld gar nicht erst angelegt haben, da es kein fest definiertes Schema gibt, was im Gegensatz zu RDBMS flexiblere Gestaltungsmöglichkeiten bietet, da man dort vor dem Erstellen einer Tabelle genaue Angaben darüber braucht, welche Felder angelegt sein müssen (die im Zweifelsfall mit NULL aufgefüllt werden müssten).

Beim Einfügen bietet MongoDB die Möglichkeit, den Grad der Konsistenz zu bestimmen. Durch `db.collection.insert( document, { writeConcern: { w: „majority“, wtimeot: 5000} } )` lässt sich festlegen, dass eine Schreiboperation erst dann als vollendet gilt, wenn die „Mehrheit“ der Replika geupdatet wurden oder alternativ 5 Sekunden verstrichen sind. (MongoDB o. J.a)

Abfragen

Ein Vorteil gegenüber den simplen Key-Value-Stores ist, dass bei dokumentorientierten Datenbanken auch eine Abfrage über Dokumenteninhalte möglich ist, ohne zuvor das ganze Objekt über den Key laden zu müssen. Das ermöglicht Abfragen, die an SQL-Konstrukte der RDBMS erinnern.

Z.B. Eine Abfrage um alle Personen einer Collection abzurufen:

In SQL: `SELECT * FROM personen`

In MongoDB: `db.personen.find()`

Oder bestimmte Person über ihren Namen:

SQL: `SELECT * FROM personen WHERE vorname = ”Max”`

MongoDB: `db.personen.find( {“vorname”:”Max”} )`

Dieselbe Abfrage mit Nachnamen:

SQL: `SELECT vorname, nachname FROM personen WHERE vorname = “Max”`

MongoDB: `db.personen.find( {„vorname“:“Max“} , {vorname:1, nachname:1} )`

Interessant wird es, wenn z.B. um Bestellungen geht, die ein bestimmtes Produkt beinhalten. Während bei RDBMS die Informationen über Kunde, Anschrift und Produkt in verschiedenen Tabellen gespeichert werden würde (was eine entsprechend lange SQL-Abfrage bedeutet), würde man in dokumentenbasierten Systemen alle Informationen in einem Dokument speichern und hätte eine entsprechend knappe Abfrage (z.B.: `db.bestellungen.find( {„produkt“:“Bleistift“} )`).

Des Weiteren gibt es Optionen mit `$lt` (<) und/oder `$gt` (>) Abfragen in definierbaren Bereichen zu machen oder Befehle wie `update()` zum Manipulieren eines bestehenden Dokuments, etc.

Eine praktische Übersicht an Befehlen findet sich z.B. hier.

Darüber hinaus lassen sich größere Aufgaben auch in MapReduce-Jobs erledigen, die in JavaScript geschrieben werden (Edlich et al. 2010: S. 124).

Skalierung

MongoDB bietet die Möglichkeit, Replikationen nach dem Master-Slave-Prinzip zu erstellen, die dabei helfen, die Lese-Last zu verteilen und eine gewisse Ausfallsicherheit zu garantieren. Dabei besteht ein Replica Set aus einem Master-Knoten (auch Primary genannt) und Reihe von Slave-Knoten (auch Secondary genannt). Alle Schreibzugriffe gehen über den Primary, der diese in einem „Oplog“ protokolliert, welches er dann an die Secondary-Knoten überträgt, damit diese ihre Daten aktualisieren können. Bei der Erweiterung eines Replica Sets sorgt MongoDB automatisch für die Datenverteilung. Sollte ein Primary ausfallen, wird er durch den Secondray ersetzt, der den aktuellsten Oplog hat. (Beitter/Brödel 2015)

Zur Entlastung des Systems bei Schreibzugriffen und um zusätzlichen Speicher bereitzustellen, bietet MongoDB horizontale Skalierung auch in Form von Sharding an. Bei der Verteilung der Daten wird nach einem Shard-Schlüssel vorgegangen, der so festgelgt werden sollte, dass sich Daten einer Collection möglichst zusammen auf einem Server befinden (z.B. Personendaten nach Heimatland verteilt). Dabei wird eine Collection in sogenannte Chunks aufgeteilt, die Daten in einem Bereich des Shard-Keys beinhalten. Diese Chunks werden dann auf Server verteilt, die man Shards nennt. Dies alles geschieht automatisiert und bedeutet keine Ausfallzeit des Systems, unter Umständen aber eine gedrosselte Performance, abhängig von der Masse an Daten, die verschoben werden. (vgl. Edlich et al. 2010: S.125f; Sadalage/Fowler 2012: S. 96f).

Verwendung

Dokumentenorientierte Datenbanken wie MongoDB bieten sich an, wenn ein flexibles Schema gefragt ist. Anwendungsgebiete wären etwa Event-Logging-Szenarien, als Content-Management-Systeme für Blogs und generell für dynamische Internetanwendungen, die ohnehin mit JSON-Dateien arbeiten. So wird MongoDB u.a. von EA Sports FIFA für die Verwaltung der Spielerdaten und Spielstati oder von eBay für die Speicherung aller Mediadaten (Produktbilder, etc.) verwendet (MongoDB o. J.b).

Weniger geeignet ist es, wenn es um Transaktionen geht, die mehrere Dokumente umfassen. Auch die Schemafreiheit kann zum Problem werden, wenn sich Dokumente zu sehr unterscheiden und so Abfragen jedes Mal angepasst werden müssen.