tgm
Die Schule der Technik

JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

INSY Unterricht

4CHIT 2017/18

Mitschrift NoSQL

Mario Fentler

15. Juni 2018

Bewertung:

Betreuer: Michael Borko, Christopher Roschger

Version: 1.1

Begonnen: 25.05 2018

Beendet: 15.06.2018

Inhaltsverzeichnis

1 Einführung

In diesem Protokoll geht es um NoSQL Datenbanken. Dieses Protokoll beschäftigt sich mitunter mit MongoDB und Couchbase.

1.1 Ziele

Das Ziel dieser Übung ist die Implementierung von je einer Übung zu den zwei Datenbanksystemen.

1.2 Voraussetzungen

- Grundlagen zu JSON
- VM mit Ubuntu (Couchbase)

1.3 Aufgabenstellung

- **Aufgabe MongoDB:**
  Es soll ein Programm erstellt werden, dass sich mit der Datenbank verbindet und 5 Schüler einfügt. Weiters sollen spezielle Queries erstellt werden.
- **Aufgabe Couchbase:**
  Es soll ein Programm für die CRUD Funktionalitäten von Couchbase erstellt werden.

2 Overview

"NoSQL steht für „Not only SQL“ und bezeichnet Datenbanksysteme, die einen nicht-relationalen Ansatz verfolgen. Diese Datenbanken, denen verschiedene Datenbankmodelle zugrunde liegen können, sind horizontal skalierbar und lassen sich für Big-Data-Anwendungen einsetzen."[1]

2.1 Verschiedene Arten von Datenbänken

Bestimmte Settings erfordern unterschiedliche Datenbänke

- Document Oriented (Couchbase, MongoDB,...)
- Column/Big Table (Cassandra, HiBase)
- Graphical (Neo4J)
- Key-Value (Redis)
- Time-Based

2.2 Vorteile NoSQL

- Bei Massendatenverarbeitung sehr schnell.
- Tabellen besitzen kein fixes Schema
- Weniger Design aufwand (ERD,RM,..)
- Geeignet für Anwendungen, die spezielle Datenstrukturen verwenden
- Einfache Verteilung auf mehrere Systeme

2.3 Nachteile NoSQL

- Nicht strukturiert
- Folgen nicht dem ACID Schema
- Keine standardisierte Abfragesprache
- Können keine Datenkonsistenz auf Datenbankebene sicherstellen

3 MongoDB

3.1 Übung

Für die Verbindung und die Inserts wurde ein von mir geschriebenes Java Programm benutzt. Dieses kann so aussehen:

```
public static void main(String[] args){
        //Verbindung mit MongoDB
        Mongo mongo = new Mongo("localhost",27017);
        //Datenbank holen
        DB db = mongo.getDB("test");

        if(db.collectionExists("insy_mongo")==true){
            //Loescht mit die Collection, damit sie nachher wieder
                ganz neu ist.
            db.getCollection("insy_mongo").drop();
        }

        //Collection holen
        DBCollection collection = db.getCollection("insy_mongo");

        //Json String bilden und in ein DBObject umwandeln
        String jsonString = KlassenGenerator.generate();
        DBObject dbObject = (DBObject)JSON.parse(jsonString);

        //DBObject in die Datenbank einfuehren.
        collection.insert(dbObject);
    }
```

Listing 1: Java Programm

Den JSON String erstelle ich mir in der Methode "KlassenGenerator.generate()". Wichtig dabei ist, das man bedenken sollte, dass es auch hierbei ein Root Element gibt (Schueler).

```
{ "schueler":[
{"name":"Mario", "jahr":"1999", "klasse": "4chit", "ampeln":[{"fach
    ":"AM","farbe":"gruen"},{"fach":"INSY","farbe":"gruen"}]},
{"name":"Julian", "jahr":"1998", "klasse": "4ahit", "ampeln":[{"fach
    ":"AM","farbe":"rot"},{"fach":"INSY","farbe":"gelb"}]}
...
]}
```

Listing 2: JSON String

3.1.1 Queries

Um die Daten dann wieder abzufragen wird die MongoShell benutzt. Um die Ergebnisse dann auch für den Menschen schön lesbar angezeigt zu bekommen wird die Funktion ”pretty()” verwendet.

Wenn man nur die Ergebnisse angezeigt bekommen will, die auch der Query entsprechen, muss man die Funktion ”aggregate()” benutzen. Denn die Funktion ”find()” gibt das ganze Objekt aus.

- **Alle Schueler:**

```
db.insy_mongo.find().pretty()
```

Listing 3: Alle Schüler

- **Alle Schueler, die 2000 gebohren sind:**

```
db.insy_mongo.aggregate([
    {$match: {'schueler.jahr': '2000'}},
    {$project: {
        Schueler: {$filter: {
            input: '$schueler',
            as: 'temp',
            cond: {$eq: ['$$temp.jahr', '2000']}
        }},
        _id: 0
    }}
]).pretty()
```

Listing 4: Alle Schüler 2000

- **Alle Schueler, die 2000 gebohren sind, oder aelter:**

```
db.insy_mongo.aggregate([
    {$match: {'schueler.jahr': {$lte:'2000'}}},
    {$project: {
        Schueler: {$filter: {
            input: '$schueler',
            as: 'temp',
            cond: {$lte: ['$$temp.jahr', '2000']}
        }},
        _id: 0
    }}
]).pretty()
```

Listing 5: Alle Schüler >= 2000

- **Alle Schueler, die 2000 geboren sind und in die Klasse 4dhit gehen:**

```
db.insy_mongo.aggregate([
    {$match: {'schueler.jahr': '2000'}},
    {$project: {
        Schueler: {$filter: {
            input: '$schueler',
            as: 'temp',
            cond: {$and: [ {$eq: ['$$temp.jahr', '2000']}, {
                $eq: ['$$temp.klasse', '4dhit']}]}
        }},
        _id: 0
    }}
]).pretty()
```

Listing 6: Alle Schüler 2000, 4dhit

- **Alle Schueler, die in die Klasse 4dhit gehen und in "AM" eine rote Ampel haben:**

```
db.insy_mongo.aggregate([
    {$match: {'schueler.jahr': '2000'}},
    {$project: {
        Schueler: {$filter: {
            input: '$schueler',
            as: 'temp',
            cond: {$and:[{$and: [ {$eq: ['$$temp.ampeln.farbe
                ', 'rot']},{$eq: ['$$temp.ampeln.fach', 'AM
                ']}]}, {$eq: ['$$temp.klasse', '4dhit']}]}
        }},
        _id: 0
    }}
]).pretty()
```

Listing 7: Alle Schüler 4dhit, AM rot

4 Couchbase

4.1 Installation

Couchbase Download über Docker in einer Ubuntu VM.

```
sudo apt-get install docker
sudo apt-get install docker.io
```

Listing 8: Installation Docker

Der Dockercontainer mit der Couchbase Instanz kann so heruntergeladen werden.

```
docker run -d --name db -p 8091-8094:8091-8094 -p
    11210-11211:11210-11211 couchbase
```

Listing 9: Docker Couchbase download

Sollte man den Couchbase Server neustarten wollen (VM neustart, ...), dann muss man sich zuerst die Container ID holen und dannach den Server manuell neustarten:

```
//Get Container ID
sudo docker ps -a
//Start Server
sudo docker start <Container-ID>
```

Listing 10: Couchbase Server neustarten

Den Server kann man dann unter "http://localhost:8091" oder "http://<vm-ip-dockerInterface>:8091" erreichen.

4.2 Couchbase Server einrichten

Auf der oben genannten Adresse kann man sich nun am Server mit den Zugangsdaten (Administrator,Couch123) verbinden.
Dort wird ein neues Cluster erstellt. In diesem werden nun die Sample Buckets installiert.(Buckets <=> Datenbank+Tables -> JSON Docs)

4.3 Benutzer erstellen

Einen User kann man auf der Webseite unter "Securtity/AddUser" hinzufügen. Diesem werden für die Übung alle Rechte auf die Buckets zugeteilt.

4.4 Buckets erstellen

Buckets sind soetwas wie eine Datenbank und die Tabellen in MySQL. Man erstellt einen in dem man auf der Webseite "Add new" drückt.

4.5 Working with the SDK

Ich erstelle ein Java Maven Program in IntelliJ. (File/new Project/Maven Project) Wichtig: Create a simple Project sollte angeklickt sein.
Folgende Dependencies müssen in das ”pom.xml” File gefügt werden:

```
<dependencies>
  <dependency>
    <groupId>com.couchbase.client</groupId>
    <artifactId>java-client</artifactId>
    <version>2.5.8</version>
  </dependency>
</dependencies>
```

Listing 11: pom.xml

4.5.1 CRUD Programm

Hier habe ich ein Example Project, dass ich von der Couchbase Seite entnommen und angepasst habe. Es verbindet sich mit dem, von mir vorher erstellten Bucket (user-bucket), fügt mir ein neues Dokument ein und macht dann eine Abfrage.

```
    //this tunes the SDK (to customize connection timeout)
    CouchbaseEnvironment env = DefaultCouchbaseEnvironment.builder()
            .connectTimeout(20000) //20000ms = 20s, default is 5s
            .build();

    // Initialize the Connection
    Cluster cluster = CouchbaseCluster.create("172.17.0.1");
    cluster.authenticate("couchUser", "Couch123");
    System.out.println("Try to open bucket");
    Bucket bucket = cluster.openBucket("user-bucket");
```

Listing 12: Verbindung mit dem Server

So werden Dokumente erstellt und in den Bucket eingefügt:

```
    // Create a JSON Document
    JsonObject mario = JsonObject.create()
            .put("name", "Mario")
            .put("email", "mfentler@student.tgm.ac.at")
            .put("interests", JsonArray.from("INSY", "Programmieren")
                );

    // Store the Document
    bucket.upsert(JsonDocument.create("u:mario", mario));
```

Listing 13: Dokument erstellen

Und so kann man dann die Abfragen schreiben:

```
// Load the Document and print it
// Prints Content and Metadata of the stored Document
System.out.println(bucket.get("u:mario"));

// Create a N1QL Primary Index (but ignore if it exists)
bucket.bucketManager().createN1qlPrimaryIndex(true, false);

// Perform a N1QL Query
N1qlQueryResult result = bucket.query(
        N1qlQuery.parameterized("SELECT␣name␣FROM␣ user-bucket ␣
            WHERE␣$1␣IN␣interests",
                JsonArray.from("INSY"))
);

// Print each found Row
for (N1qlQueryRow row : result) {
    // Prints {"name":"Arthur"}
    System.out.println(row);
}
```

Listing 14: Abfragen

5 Quellen

[1] https://www.bigdata-insider.de/was-ist-nosql-a-615718/
https://api.mongodb.com/python/current/tutorial.html
https://docs.mongodb.com/manual/tutorial/insert-documents/
https://stackoverflow.com/questions/26214587/create-collection-in-mongodb-using-java
https://forums.couchbase.com/t/unable-to-connect-to-db-java-util-concurrent-timeoutexception/4471
https://developer.couchbase.com/documentation/server/5.1/sdk/java/start-using-sdk.html

Auflistungsverzeichnis