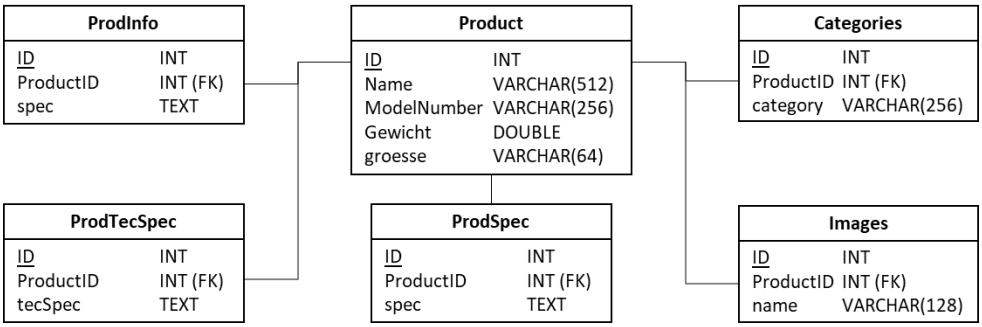
1 Konzept hinter MongoDB

MongoDB ist eine NoSQL (also „Not only SQL") Datenbank und gehört zu der Kategorie der „Dokumentenbasierten Datenbanken". Der Name wird vom englischen Wort „*humongous*" (engl. für riesig, enorm) ab. Die Idee hinter dokumentenbasierten Datenbanken ist, dass Informationen nicht, wie in einer relationalen Datenbank, in der 3. Normalform abgelegt – also alle Daten in einzelne Tabellen aufgeteilt – werden, sondern zusammengehörige Daten auch in einer zusammengehörigen Struktur – dem Dokument – abgelegt werden.

Hierzu sehen wir uns das Datenmodell für die Produktinformationen im relationalen Datenmodell an:



Um nun alle Produktrelevanten Daten zu ermitteln, müssen wir die einzelnen 1:n Beziehungen über einen Join auflösen und die einzelnen Informationen zusammensetzen. Das vergleichbare Dokument in einer MongoDB könnte wie folgt aussehen:

```
{
  "Name":"myName",
  "ModelNumber":"12345",
  "Gewicht":123.45,
  "Groesse":"123",
  "Categories":["Category A","Category B"],
  "Info":["Info 1","Info 2"],
  "Images":["ImgMod12345_01.jpg","ImgMod12345_02.jpg","ImgMod12345_03.jpg"],
  "Spec":["Bruttoweight:123.45","Nettoweight:100.00","NormId:23221"],
  "TecSpec":["Keep it try","Need special tools for assembly"]
}
```

Es handelt sich um ein simples JSON Format. Das Besondere ist nun, dass wir in JSON Arrays und auch verschachtelte Objekte realisieren können. In einer MongoDB können nun beliebig viele solcher Datensätze hinterlegt werden.

Anmerkung: Intern nutzt MongoDB das sogenannte „BSON" Format, was für „Binary encoded JSON" steht. Somit ist es auch möglich, in einer MongoDB auch Binärdaten zu speichern, bzw. über JSON hinaus gehende Datentypen zu definieren. Für uns ist dies erstmal nicht relevant, weshalb ich beim Begriff „JSON" bleibe.

2 Installation

Prinzipiell benötigt man erstmal keine lokale Installation einer MongoDB, da es für erste Gehversuche kostenlose Onlineangebote gibt[1]. Ansonsten ist es problemlos möglich, den kostenfreien MongoDB Community Server von https://www.mongodb.com/try/download/community herunterzuladen und lokal zu installieren. Bei der Installation (v. 6.0.2) wird zugleich noch das Administrationswerkzeug MongoDB Compass hinzugefügt, welches für allgemeine Wartungs- und Administrationsaufgaben sinnvoll ist.

[1] Bspw.: https://www.mongodb.com/pricing

Weiterhin wird empfohlen, noch die MongoDB Shell unter https://www.mongodb.com/try/download/shell und die Datenbanktools unter https://www.mongodb.com/try/download/database-tools2 zu laden und zu installieren (idealerweise jeweils die Windows 64-bit (MSI) Version wählen). Mit mongosh können wir MongoDB Befehle direkt in einer Konsole ausführen und die Tools erlauben einen Export/Import. Sollte man mit VSCode arbeiten, ist das zugehörige Plugin von MongoDB ein weiteres hilfreiches Werkzeug, wobei der automatisch installierte MongoDB Compass im Regelfall komfortabler ist.

3 Begriffsdefinition

Folgende Begriffe sind bei der Nutzung von MongoDB wichtig zu unterscheiden:

Begriff:	**Bedeutung:**
Dokument	Ein einzelner Datensatz, welcher aus zusammengehörigen Informationen besteht.
Collection	Eine Sammlung Dokumenten. Sie müssen nicht zwingend einem Schema folgen, wenngleich dies meist sinnvoll ist. Bei relationalen Datenbanken wäre dies am ehesten mit einer Tabelle zu vergleichen.
Datenbank	Eine Sammlung von Collections.
Schema	Vorgegebene Struktur der Daten. Diese kann, muss aber nicht erzwungen werden.

4 Unterstützte Datentypen

Da wir in unseren Projekten die MongoDB aus JavaScript speisen, sind die von uns genutzten Datentypen auf die JavaScript Datentypen begrenzt, wobei wir hier erstmal nur `String`, `Number` und `boolean` nutzen. Intern kann MongoDB sehr viel mehr[2]. Darüber hinaus können wir noch existierende andere Dokumente in der MongoDB referenzieren[3]. Wir arbeiten in unserem Projekt jedoch lediglich mit einer einzigen Collection mit voneinander unabhängigen Dokumenten.

5 Wichtige Befehle in MongoDB

Die hier aufgezeigten Befehle konzentrieren sich auf die Nutzung der MongoDB Shell. Später werden wir mit der Mongoose API arbeiten, wobei sich die dort realisierten Befehle weitestgehend mit den Shell Befehlen decken.

5.1 Grundkonzept

Anders als bei SQL, sind die MongoDB Befehle objektorientiert aufgebaut. Hierbei geht die MongoDB nach dem folgenden Schema vor:

`db.`*`Collection`*`.`*`Aktion`*`([`*`Parameter`*`])[.`*`Aktion`*`([`*`Parameter`*`])]`

`db`	Muss immer „`db`“ lauten und steht für die aktuell „geöffnete“ Datenbank.
`Collection`	Name der Collection, in der die Aktion ausgeführt werden soll.
`Aktion`	Aufruf der Aktion, welche in der Collection, bzw. auf dem Ergebnis der ersten Aktion, ausgeführt werden soll.
`Parameter`	Die einzelnen Aktionen benötigen jeweils [optionale] Parameter

Bevor wir also eine Aktion durchführen können, müssen wir eine Datenbank auswählen:

```
show dbs     // Listet die existierenden Datenbanken auf
db           // zeigt die aktuell geöffnete Datenbank an
use myDb     // öffnet die DB (bzw. erzeugt sie, wenn nicht existent)
show collections  // Zeigt die existierenden Collections der akt. DB
```

Collections werden automatisch erstellt, sobald der erste Datensatz eingefügt wird.

5.2 Einfügen von Daten

Daten werden in MongoDB über die Befehle `insertOne()` bzw. `insertMany()` eingefügt, wobei `insertMany()` die Dokumente als Array von JSON Objekten erwartet:

[2] https://www.mongodb.com/docs/manual/reference/bson-types/
[3] https://www.mongodb.com/docs/v5.3/reference/database-references/#dbrefs

```
db.books.insertOne({title:"C Language", pages: 479,
authors:["Kernighan", "Ritchie"]});

db.books.insertMany([{title: "C++ Programming", pages: 911,
authors: ["Stroustrup"]},
{title: "UML 2.5", pages: 450, onlinecode: "XY1234",
authors: ["Kecher", "Salvanos", "Hoffmann-Elbern"]}]);
```

Jeder neu geschriebene Datensatz erhält automatisch eine über die gesamte Datenbank eindeutige ID. Der Datensatz der C-Language würde somit bspw. wie folgt aussehen:

```
{ "_id": {"$oid": "634593fce2a7b4be613c954a"},
  "title": "C Language",
  "pages": 479,
  "authors": ["Kernighan","Ritchie"]
}
```

Eine weiter Möglichkeit zum Laden von Daten ist es, ein JSON Array File über das Importtool zu importieren:

```
mongoimport --db warpshop --collection products --type=json
   --file C:\temp\wpshpProduct.json --jsonArray
```

5.3 Lesen von Daten

Der Hauptbefehl für das Lesen von Daten ist `find()`, welche als Parameter die Suchbedingungen akzeptiert (bzw. alle ausgibt, wenn kein Parameter angegeben wird):

```
db.books.find({pages: 479});
```

Hier gilt zu beachten, dass der Parameter von `find()` ein Objekt für die Suchbedingungen ist, also in `{}` stehen muss. Weiterhin unterscheidet MongoDB `pages:"479"` und `pages:479`. Bei den Suchbedingungen gibt es nun eine Vielzahl von Möglichkeiten – hier eine Auswahl:

Suchbedingung:	**Funktion:**
`{title:" UML 2.5", pages: 450}`	Alle Dokumente mit title = „UML 2.5“ **und** pages = 450
`{$or:[{title:" UML 2.5"}, {pages:450}]}`	Alle Dokumente mit title = „UML 2.5“ **oder** pages = 450
`{pages:{$gt:200}}`	Alle Dokumente mit pages > 200

Auflistung der wichtigen Operatoren:

Operator:	**Funktion:**
`{onlinecode: {$exists: true}}`	Dokumente, deren Eigenschaft „onlinecode“ existiert
`{title: {$type: "int"}}`	Dokumente, bei denen in „title“ eine Ganzzahl steht
`{pages:{$gt:200}}`	Dokumente, bei denen in „pages“ > 200 ist
`{pages:{$gte:200}}`	Dokumente, bei denen in „pages“ >= 200 ist
`{pages:{$lt:200}}`	Dokumente, bei denen in „pages“ < 200 ist
`{pages:{$lte:200}}`	Dokumente, bei denen in „pages“ <= 200 ist
`{pages:{$ne:200}}`	Dokumente, bei denen in „pages“ != 200 ist
`{title:{$in:["SQL","UML 2.5"]}}`	Dokumente, die „SQL“ oder „UML 2.5“ im title haben
`{authors:{$in:["Kecher", "Berners-Lee"]}}`	Dokumente, bei denen „Kecher“ oder „Berners-Lee“ in der Autorenliste zu finden sind
`{authors:{$all:["Kecher", "Berners-Lee"]}}`	Dokumente, bei denen „Kecher“ und „Berners-Lee“ in der Autorenliste zu finden sind
`{authors:{$nin:["Kecher", "Berners-Lee"]}}`	Invertierung von $in, also Dokumente, bei weder „Kecher“ noch „Berners-Lee“ in der Autorenliste zu finden sind
`{$or:[{title:" UML 2.5"}, {pages:450}]}`	Dokumente, die den Titel „UML 2.5“ haben, oder 450 Seiten
`{$and:[{title:" UML 2.5"}, {pages:450}]}`	Dokumente, die den Titel „UML 2.5“ und 450 Seiten haben

Operator:	Funktion:
{**$nor**:[{title:" UML 2.5"}, {pages:450}]}	Dokumente, die weder den Titel „UML 2.5“ noch 450 Seiten haben
{**$not**:{pages:450}}	Dokumente, die keine 450 Seiten haben

Wenn bei `find()` die Existenz gewisser Spalten gefordert ist, dann können diese auch explizit angegeben werden:

```
db.books.find({}, {authors:true, title:true});
```

Weiterhin ist noch zu erwähnen, dass ein eindeutiges finden am besten über die ID erfolgt:

```
db.books.find({_id:ObjectId("634593fce2a7b4be613c954a")});
```

Wird nur die Anzahl der Fundstellen benötigt, so wird statt `find()`, `countDocuments()` verwendet:

```
db.books.countDocuments({authors:{$in:["Kecher"]}});
```

5.4 Verkettung

Die Ergebnismenge von `find()` kann wiederum an weitere Aktionen weitergereicht werden. Diese sind:

Aktion:	Funktion:
`.limit(n)`	Limitiere die Ausgabemenge auf die ersten `n` Elemente.
`.skip(n)`	Überspringe die ersten `n` Elemente.
`.sort({field:1})`	Sortiert nach `field` absteigend (`1`) oder aufsteigend (`-1`)

Aktion:	Funktion:
`db.books.find().limit(3)`	Gibt die ersten 3 Dokumente von books zurück
`db.books.find().skip(3)`	Gibt ab dem 3. Dokument alle weiteren zurück
`db.books.find().skip(3).limit(2)`	Gibt die Dokumente 4 und 5 zurück
`db.books.find().sort({title: 1})`	Gibt die Bücher nach Namen sortiert zurück

5.5 Ändern von Daten

Für das Ändern von Daten gibt es ebenfalls eigene Funktionen. Die wichtigsten sind `updateOne()` und `updateMany()`. Beide Funktionen erwarten einen Filter, so wie er auch bei `find()` angewendet wird, wobei `updateOne()` immer nur den ersten Treffer verändert und `updateMany()` entsprechend alle. Das folgende Statement sucht das erste Buch mit dem Titel „UML 2.5“ und nennt es in „UML 2.6“ um:

```
db.books.updateOne({title:"UML 2.5"}, {$set:{title:"UML 2.6"}})
```

Arrayinhalte werden mit `$push` hinzugefügt:

```
db.books.updateOne({title:"UML 2.5"}, {$push:{authors:"Einstein"}})
```

Mit `$pull` würden wir ein Arrayelement löschen. Da in MongoDB die Feldnamen auch ein Teil der Daten sind, werden diese auch über `updateOne()` bzw. `updateMany()` geändert. Hier am Beispiel einer Umbenennung eines Feldes:

```
db.books.updateOne({_id:ObjectId("634593fce2a7b4be613c954a")}, {$rename:
{title:"bookTitle"}});
```

Nun fehlt noch das komplette Entfernen eines Feldes – wobei hier das zu löschende Element als „Objekt“ übergeben werden muss, weshalb aufgrund der Konvention ein Leerstring notwendig ist:

```
db.books.updateOne({_id:ObjectId("634593fce2a7b4be613c954a")}, {$unset: {
bookTitle:""}});
```

Eine weitere Option ist `replace()`. Hier wird zuerst wieder ein Filter benötigt, jedoch danach ein komplettes Objekt. Dieses ersetzt das alte Objekt vollständig – also alle Eigenschaften, welche im alten Objekt vorhanden waren, jedoch nicht mehr im neuen Objekt, gehen verloren.

5.6 Löschen von Daten

Das Löschen von Daten wird folgerichtig ebenfalls über eine Funktion – `deleteOne()` und `deleteMany()` erledigt.

```
db.books.deleteOne({_id:ObjectId("634593fce2a7b4be613c954a")});
```

6 Erzeugung eines Index

Die Daten in den einzelnen Collections sind mit einem Index auf das Feld _ID versehen, so dass der Zugriff hierauf extrem schnell durchgeführt werden kann. Möchte man auf ein anderes Feld einen Index legen, so kann dies entweder im MongoDB Compass erfolgen, oder über die Shell, mit folgendem Statement:

```
db.products.createIndex({"DB_ID": 1}, {unique: true});
```

Hiermit wird ein Index auf das Feld `DB_ID` erstellt, welcher darüber hinaus noch eindeutig sein muss. Dies ist bspw. sinnvoll, wenn wir Daten sowohl in einer MongoDB, als auch in einer MySQL Datenbank haben.