Vorbereitung der Portierung der Stoffdatenbank GSBL in eine NoSQL-Datenstruktur anhand der dokumentorientierten Datenbank: MongoDB

Bachelorarbeit

Zur Erlangung des akademisches Grades

Bachelor of Science (B.Sc.)

Eingereicht an der

Hochschule Anhalt

Fachbereich Informatik und Sprachen

von

Iryna Voronkina

Matrikelnummer: 4051757

Studienrichtung: Informationsmanagement

Erster Gutachter (HS Anhalt (FH)): Prof. Dr. Michael Worzyk

Zweiter Gutachter (HS Anhalt (FH)): Prof. Dr. Fissgus

Abgabetermin: 17.03.2015

Eidesstattliche Erklärung

Diese Arbeit wurde von mir selbstständig verfasst und in gleicher oder ähnlicher Fassung noch nicht in einem anderen Studiengang als Prüfungsleistung vorgelegt. Ich habe keine anderen als die angegebenen Hilfsmittel und Quellen, einschließlich der angegebenen oder beschriebenen Software, verwendet."

Dessau-Roßlau, den 09.03.2015

Iryna Voronkina

Inhalt

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Zielsetzung des Praktikums in UBA

Im Rahmen meines Praktikums am Umweltbundesamt (UBA) in Dessau-Roßlau wurde untersucht, ob ein NoSQL –Datenbanksystem[1] als Alternative zur relationalen Datenspeicherung des gemeinsamen Stoffdatenpools von Bund und Ländern (GSBL) in Frage kommt. Als ein alternatives Datenbanksystem wurde die weitverbreitete NoSQL[2]-Datenbank MongoDB vorgeschlagen. Die zu beantwortende Fragestellung für meine Praktikumsaufgabe lautete: ob man grundsätzlich die Daten des GSBL in MongoDB überführen kann, welche Möglichkeiten der webbasierten[3] Anzeige existieren, wie performant die Suche nach Daten funktioniert und welche Probleme dabei entstehen können. Aus den daraus resultierenden Lösungen und weiterführenden Fragestellungen entstand ein Leitfaden für die Erstellung eines vollständig auf MongoDB-basierten Datenbankkonzeptes, welches auf die besonderen Anforderungen des GSBL und ähnlichen Stoffdatenbanken zugeschnitten ist.

1.2 Motivation und Zielsetzung dieser Arbeit

Das Thema dieser Arbeit orientiert sich am Bedarf des Umweltbundesamtes, die Stoffdatenbank GSBL weiter zu entwickeln, die Speicherstruktur des GSBL zu modernisieren und die Möglichkeit seiner Nutzung für verschiedene Anwendergruppen zu verbessern. Die heutige Datenhaltung basiert auf einer relationalen[4] Speicherung und wird transponiert gelöst. Transponierung in Bezug auf eine relationale Speicherung bedeutet, dass Spalten und Zeilen einer Tabelle getauscht werden. Die Tabelle mit GSBL-Daten ist derart gestaltet, dass die Eigenschaften der chemischen Stoffe als Zeilenbezeichnungen statt als Spaltenüberschriften aufgefasst werden, wie es üblich bei einer relationalen Speicherung ist. Es entsteht eine sehr große Tabelle mit Stoffeigenschaften, weil viele Eigenschaften mehrmals in verschieden Zeilen gespeichert werden müssen. Andere relationale Lösungen für GSBL wurden vom UBA als nicht akzeptabel getestet.
Die Wahl eines NoSQL-Datenmodells im Allgemeinen und MongoDB als Datenbanksystem im Speziellen wurde in meinem Praktikumsbericht (Kapitel 4, Punkt 4.1 „Wahl von NoSQL-Datenmodell“ und Punkt 4.2 „Wahl von MongoDB“) erklärt. In dieser Arbeit wird ein Überblick über verschiedene Datenmodelle, ihre Vorteile und Nachteile im Hinblick auf die spezielle Problematik der Stoffdatenbanken gegeben. Die Eigenschaften von MongoDB werden in Bezug auf eine konkrete Problemstellung getestet und bewertet.

---

[1] Eine Datenbank (oder auch Datenbanksystem) ist ein System zur elektronischen Datenverwaltung. Die Hauptaufgabe eines DBS ist die dauerhafte, widerspruchsfreie und effiziente Speicherung großer Datenmengen [Wiki, 2015]

[2] Seit ca. 2009 wird NoSQL als Sammelbegriff für zum relationalen Modell "alternative" Datenbankentwicklungen verwendet [FH Köln, 2012]
Der Zugriff auf die Daten erfolgt durch Internet

[3] Der Zugriff auf die Daten erfolgt durch Internet

[4] Bei einer relationalen Speicherung werden Daten in Form von zweidimensionalen Tabellen verwaltet

Bei den Untersuchungen im UBA wurde der Anwendungsfall „des einfachen Nutzers“ in den Fokus gestellt; dabei werden die Möglichkeiten des Datenbankmanagementsystems[5] unter dem Aspekt von Recherchen der interessierten Öffentlichkeit getestet. Wenn ein Nutzer des GSBL nicht zur Gruppe der Fachleute gehört, soll es das Datenbanksystem ihm ermöglichen, nach einem oder zwei Begriffen zu suchen: einem stoffidentifizierenden Namen oder einer stoffidentifizierenden Nummer ohne zu wissen, welche Namen oder Nummern einen chemischen Stoff charakterisieren können. Da ein Stoff oft mehrere Namen und Nummern hat, die zu verschiedenen Eigenschaften gehören (bei Namen gibt es beispielsweise Registriername, Name; bei Nummern: Cas-Nummer, Index-Nummer, EG-Nummer, UN-Nummer), sollte das System mehrere Eigenschaften auf den eingegebenen Begriff performant durchsuchen zu können. Die für das Praktikum vom UBA zur Verfügung gestellten Software- und Hardwarekomponenten (einschließlich der virtuellen Umgebung) verlangten für die akzeptable Lösung dieser Aufgaben unbedingt eine Performanz-Verbesserung (die benötigten Eigenschaften wurden indiziert). Das hat weiterführende Fragen aufgeworfen:

- Wie effizient wird das System den Anforderungen von Fachleuten entsprechen, falls sie nicht nur nach Namen oder Nummern die Datenbank durchsuchen müssen, sondern nach beliebigen Stoffeigenschaften? In der Datenbank gibt es zu den Stoffen insgesamt 452 unterschiedliche Eigenschaften (Merkmale genannt), wobei jeder einzelner Stoff durch eine unterschiedliche Anzahl von Eigenschaften (Merkmalen) gekennzeichnet sein kann. Die Merkmale sind wiederum in die eigenen Eigenschaften (Felder genannt) unterteilt. Es ist ein einfaches Beispiel vorstellbar. Man soll einen oder mehrere Stoffe finden, die den Dampfdruck 100 hPa bei einer Temperatur von 20°C und einem Siedepunkt von etwa 80 °C haben, die zudem farblos und leichtentzündlich sind. In diesem Fall wird das System solche Stoffe finden müssen, die die Eigenschaften (Felder) „Dampfdruck“, “Temperatur für den Dampfdruck“, “Siedetemperatur“, „Farbe“ und „Einstufung“ mit den gesuchten Werten haben.
- Welche Möglichkeit gibt es, die bei der Untersuchung im UBA ermittelten Suchzeiten zu optimieren?
- Viele Stoffnamen sind oft als lange Zeichenketten gespeichert, die nicht leicht zu merken sind. Es ist sinnvoll, als Suchbegriff nur die ersten Buchstaben des Stoffnamens einzugeben. Kann das System in diesem Fall eine performante Suche gewährleisten?
- Welche Möglichkeiten bietet die Datenbank in Bezug auf Schnittstellen zum Datenaustausch? Wird das Datenbankmanagementsystem sich bewähren, falls Eigenschaften der Stoffe ständig geändert werden?
- Welche Infrastruktur ist in einer reellen Umgebung notwendig?

[5] Datenbankmanagementsystem ist Bestand eines Datenbanksystems, sie organisiert intern die strukturierte Speicherung der Daten und kontrolliert alle lesenden und schreibenden Zugriffe auf die Datenbank. Zur Abfrage und Verwaltung der Daten bietet ein Datenbanksystem eine Datenbanksprache an.

- Welche zusätzlichen Anwendungen werden benötigt, wenn das System von Fachleuten genutzt wird?

Zwischen diesen Fragestellungen und der Motivation zur Weiterentwicklung des GSBL besteht ein Zusammenhang. Man muss die Umstellung auf NoSQL in Betracht ziehen, wenn es ökonomisch sinnvoll ist (wenn zwei Datenbanksysteme durch ein neues NoSQL-System ersetzt werden können) oder das es einen Geschwindigkeitszuwachs gibt. Es ist dabei darauf zu achten, dass alle benötigten Module und Schnittstellen in das neue Datenbankmanagementsystem mit Rücksicht auf Benutzerfreundlichkeit eingebunden werden können.

Diese Arbeit soll in Bezug auf eine mögliche Portierung des GSBL (oder ähnlicher Stoffdatenbanken) zu NoSQL eine Hilfe zur Verfügung stellen.

2 GSBL

2.1 GSBL- Bedeutung für Umwelt

Seit 1994 wird der Gemeinsame zentrale Stoffdatenpool des Bundes und der Länder (GSBL) vom Bund und mittlerweile allen Ländern gemeinsam gepflegt, erweitert und genutzt. Der GSBL ist die wichtigste und größte Informationsquelle für chemische Stoffe, die für alle Bereiche des Umweltschutzes und zur Gefahrenabwehr von großer Bedeutung ist. Der GSBL beinhaltet auch die Information über die rechtlichen Regelungen für chemische Stoffe in den unterschiedlichsten Rechtsbereichen zum Beispiel: das Lebensmittelrecht, der Arbeitsschutz, Immissionsschutz.
In der aktuellen GSBL-Version sind ca. 63.000 Einzelinhaltsstoffe (Reinstoffe), ca. 320.000 Komponentenstoffe (Gemische und Zubereitungen) und ca. 207.000 Rechtsstoffklassen (rechtliche Regelungen) recherchierbar. [GSBL2014]

2.2 GSBL- Nutzergruppen

- Die **Umweltbehörden des Bundes und Länder** nutzen den GSBL um die möglichen Gefahren eines Stoffes rechtzeitig zu erkennen und abzuwehren.
- Neben den Umweltbehörden informieren sich **Einsatzkräfte von Feuerwehr und Polizei** über die Gefährdung von Mensch und Umwelt bei Verkehrs- und Arbeitsplatzunfällen, bei Bränden mit chemischen Substanzen.
- Auch **politische Kräfte** brauchen die Daten zur Bewertung bestehender und zur Erarbeitung neuer Rechtsvorschriften.
- Ein Teil der Datenbank ist im Internet unter www.gsbl.de öffentlich zugänglich und somit auch für interessierte Bürger nutzbar.

2.3 GSBL- Anforderungsanalyse

2.3.1 Anforderungsanalyse - Bedeutung

Die Anforderungsanalyse ist die erste Phase des Datenbankentwurfs. Sie identifiziert alle Aufgaben, die das gewünschte Datenbankmanagementsystem lösen soll. Sie

beschäftigt sich mit der Frage, welche Daten in einer Datenbank verwaltet werden, welche Zusammenhänge zwischen den Daten bestehen und welche Prozesse auf die Daten angewandt werden.

2.3.2 Daten und Zusammenhänge zwischen den Daten

Durch eine Betrachtung der vorhandenen Daten lassen sich die unten aufgeführten Kategorien feststellen:

- Stoffdaten
- Lieferantendaten
- Speziesdaten
- Zitatendaten
- Strukturdaten(ROSDAL oder MOLFILE)
- Daten über Datenmodell, zu denen die Datei DBDIR.TAB und Excel-Datei „gsbldbdir3_029_0036_2013“, Tabellenblatt „GSBL 3“ gehören
- Daten zu Nachschlagetabellen, zu denen die Datei TABLES.TAB und Excel-Datei „gsbldbdir3_029_0036_2013“,Tabellenblatt “Nachschlagetabellen“ gehören
- Daten zu verwendeten physikalischen Einheiten und ihren Umrechnungen, zu denen Datei UNITS.TAB und Excel-Datei „gsbldbdir3_029_0036_2013, Tabellenblatt „Einheitentabelle“ gehören

Existierende Dokumentation:
„Gemeinsamer zentraler Stoffdatenpool des Bundes und der Länder(GSBL). GSBL-Handbuch“. Version:3.022.0028. Stand Januar 2011“

Der aktuelle „GSBL“ nutzt ein relationales Datenbankmanagementsystem (DMBS).

Ein **Stoff** ist in den Stoffdatenbanken eine Dokumentationseinheit, die Daten aus meist unterschiedlichen Quellen umfasst. Gesammelt werden die Daten sowohl zu Eigenschaften des chemischen Stoffes und seiner Vorkommen in der Umwelt als auch zu seiner Gefährlichkeit, sicheren Umgang und benötigten Schutzmaßnahmen, sowie die stoffrelevanten Inhalte aus rechtlichen Regelungen und Verweise auf andere Stoffdatenbanken.
Das Konzept der Stoffthesaurus[6] definiert die Verwandtschaftsbeziehungen zwischen den chemischen Stoffen auf der Basis von vorhandenen Daten und macht verschiedene Arten dieser Beziehungen für Nutzer zugänglich. Die Verweise auf verwandte Stoffe gehören zu der Eigenschaft „Verwandter Stoff“ (Kurzform: „THES“).

Der GSBL enthält außerdem Informationen über **Lieferanten** der Daten und über **Spezies** - Lebewesen, die zum Zweck der Bewertung des Umweltrisikos untersucht wurden. Das Datensegment „**ROSDAL**“ und „**MOLFILE**“ enthält chemische Strukturen in jeweils einem speziellen Format.

Literaturquellen (Zeitungen, Handbücher, Gesetzessammlungen), in welchen der entsprechende Sachverhalt eines Merkmales veröffentlicht worden ist, sind unter dem Namen „**Zitate**“ in der Datenbank abgelegt.

[6] Thesaurus besteht aus einer systematisch geordneten Sammlung von Begriffen, die in thematischer Beziehung zueinander stehen [Thesaurus, 2015]

Die Datei DBDIR.TAB ebenso wie Excel-Datei „gsbldbdir3_029_0036_2013“ (Tabellenblatt „GSBL 3“) beschreibt das fachliche **Datenmodell des GSBL**. Das Datenbankkonzept des GSBL ist auf der Basis von **Merkmalen**[7] und **Feldern**[8] definiert. Ein jedes Merkmal besteht aus mindestens einem Feld, das mit einem Inhalt belegt ist. Diese zusammenfassende Merkmalstabelle ordnet den einzelnen Merkmalen Oberbegriffe zu, die jeweils eine Gruppe von Merkmalen umfassen. Die Namen von Merkmalen und Feldern sind in GSBL in Kurzform abgelegt. In der Merkmalstabelle sind zu diesen Abkürzungen auch Langnamen angegeben.

Ein Ausschnitt aus der Excel-Tabelle (Tabellenblatt „GSBL 3“) ist als Tabelle1 unten aufgeführt. Tabelle1 definiert das dem Oberbegriff „ALGM“ (Langbezeichnung: „Allgemeine Merkmale (Reale Stoffe und Stoffklassen)“) zugeordnetes Merkmal „GSBL“ (Langbezeichnung: „Stoff“) mit fünf zugehörigen Feldern (GSBL-RN, Stoffart, Strukturnummer, Sperre, Stoffpflege).

GSBL-RN ist eine Nummer, die bei der Registrierung eines Stoffes im GSBL vergeben wurde. Sie identifiziert einen Stoff, der in GSBL abgelegt wurde, eindeutig.

Stoffart charakterisiert einen Stoff unter bestimmten Gesichtspunkten.

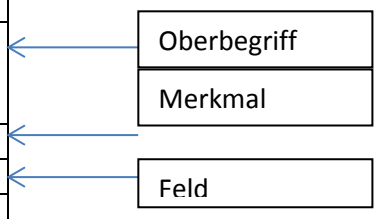
Strukturnummer ist die Verknüpfung mit der Strukturdatenbank und mit dem Datensegment „Rosdal“, wo Strukturdaten in Beilstein-Rosdal-Format[9] abgelegt sind.

Sperre zeigt, ob neue Registriernamen zu dem durch die GSBL-RN identifizierten Stoff eingeführt werden dürfen.

Stoffpflegesperre bezieht sich auf den ganzen Datensatz.

Tabelle 1: Merkmal „GSBL“ mit zugehörigen Feldern

Langbezeichnung	**Kurzbezeichnung**
Allgemeine Merkmale (Reale Stoffe und Stoffklassen)	ALGM
Stoff	GSBL
GSBL-RN	GSBL.GSBLRN
Stoffart	GSBL.STAR
Strukturnummer	GSBL.SNR
Sperre	GSBL.SPERRE
Stoffpflegesperre	GSBL.SPFSPERRE



Die Merkmalstabelle mit dem Datenmodell enthält einige zusätzliche Angaben. Zum Beispiel die Angabe über eine mögliche Multiplizität des Merkmals, was mehrmaliges Auftreten eines Merkmals im Rahmen eines Stoffes bedeutet, oder den Verweis auf eine **Einheitentabelle** (in der Excel-Datei „gsbldbdir3_029_0036_2013 Tabellenblatt „Einheitentabelle“), in welcher Standarteinheit und Umrechnungsfaktor festgelegt sind. (Beispiel: Temperaturwerte müssen in °C umgerechnet werden).

[7] Merkmal fasst Stoffinformationen zu logischen Einheiten zusammen

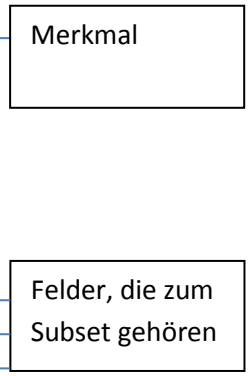
[8] Feld ist Bestandteil eines jeden Merkmals, Feld nimmt die zum Speicherung vorgesehenen Werte

[9] Das ROSDAL Format zeichnet sich durch eine sehr einfache, für einen Chemiker leicht zu erlernende Verschlüsselung der chemischen Struktur mittels alphanumerischer Symbole aus. [Gasteiger, 2003]

Innerhalb eines Merkmals können mehrere Werte auftreten, die zusammen gehören, sogenannten **Subsets**. In der Merkmalstabelle mit dem Datenmodell wird durch eintragen einer Ziffer gezeigt, welche Felder zu einer Gruppe (zu einem Subset) gehören. Tabelle2 zeigt ein solches Subset.

Tabelle 2: Merkmal LMVHPSM mit dem Subset, der durch Ziffer „1“ gekennzeichnet ist

Langbezeichnung	Kurzbezeichnung	Subset
Verordnung über Höchstmengen an Rückständen von Pflanzenschutz- und Schädlingsbekämpfungsmitteln, Düngemitteln und sonstigen Mitteln in oder auf Lebensmitteln (Rückstands-Höchstmengenverordnung - RHmV)	LMVHPSM	
Höchstmenge	LMVHPSM.HM	1
Bezugslebensmittel	LMVHPSM.BZLM	1
Anmerkung	LMVHPSM.ANM	1



Wenn zu einem Feld eine zusätzliche erläuternde **Nachschlagetabelle** mit standardisierten Werten existiert, wird der Name dieser Tabelle ebenso in der Merkmalstabelle hinterlegt. Diese erläuternden Tabellen können merkmalsübergreifend sein. In diesem Falle fassen sie Merkmale und Felder zusammen, die einen bestimmten Bereich charakterisieren (Beispiel: Ökotoxikologie).

Außer in der aktuellen Datenbank (Tabelle „T_NST“) sind die Nachschlagetabellen auch in der Excel-Datei „gsbldbdir3_029_0036_2013“(Tabellenblatt “Nachschlagetabellen“) hinterlegt. Das Tabellenblatt „Nachschlagetabellen“ enthält im Unterschied zu „T_NST“ nur die Felder, in denen die Feldinhalte komplett sind (kein neuer Feldinhalt hinzugefügt werden darf). Tabelle 3 zeigt einen solchen Eintrag. Der Namen der Nachschlagetabelle ist „T_STAR“; T_STAR bezieht sich auf das Feld GSBL.STAR (siehe Tabelle 1). GSBL.STAR kann folgende drei Feldinhalte haben: Einzelinhaltsstoff[10], Komponentenstoff[11], Stoffklasse[12]. Zu dem Einzelinhaltsstoff können Literaturdaten vorhanden sein, die an dem Stoff in seiner reinstmöglichen Form ermittelt wurden. In der Tabelle 3 ist neben der Stoffart „Einzelinhaltsstoff“ ein Kommentar „Literaturstoff“ eingetragen.

Tabelle 3: Nachschlagetabellen

Tabelle	Nummer	Text	Weiterer Text
T_STAR	1	Einzelinhaltsstoff	(Literaturstoff)
T_STAR	2	Komponentenstoff	(Gemisch)
T_STAR	3	Stoffklasse	

[10]Einzelinhaltsstoff ist ein Stoff, der exakt durch Ihren wissenschaftlichen Namen und/oder ihre Strukturformel identifiziert ist

[11] Komponentenstoff ist ein Produkt oder Gemisch, das mit herstellungsbedingten Verunreinigungen vorliegt oder aus mehreren Komponenten besteht

[12] Stoffklasse fasst Einzelinhaltsstoffe und Komponentenstoffe zusammen, stammt häufig aus gesetzlichen Regelwerken und sind durch eine Regel und eine Mitgliederliste definiert [Stoffart, 2014]

2.3.3 Prozesse und Funktionen

Stoffregistrierung:
Stoffregistrierung ist das Verfahren, das die Aufnahme von neuen Stoffen oder von Daten zu bereits existierten Stoffen in die Datenbank regelt.
Grundlage für die Stoffregistrierung hängt mit der Definition des Stoffbegriffes zusammen. Eine eindeutige Stoffdefinition ist Voraussetzung für die Registrierung von Stoffen im GSBL; Stoffe, die keinen vollständigen Identifikations-Datensatz haben, können zunächst nicht registriert werden. [GSBL-Handbuch, 2011]
Identifikationseintrag muss unbedingt einer der Stoffarten zugeordnet sein. Eine Übersicht über Stoffarten ist auf der oben dargestellten Tabelle 3 zu sehen. Es gibt Merkmale, die einen Stoff eindeutig identifizieren. Dazu gehören exakte wissenschaftliche Bezeichnungen und die Strukturformel. Es gibt Merkmale, die bei dem Identifizieren zur Kontrolle verwendet werden (z.B. Cas-Nummer oder Index-Nummer).
Bei Produkten muss zur Identifikation der Handelsname, der Name der Hersteller und der Firma benutzt werden. Da sich diese Daten ständig ändern können, muss die Identifizierung von Produkten regelmäßig aktualisiert werden. Zu den weiteren Eigenschaften, die einen Stoff identifizieren, gehört eine möglichst vollständige Beschreibung der makroskopischen[13] Zustandsform. Bestimmte Stoffe, die chemisch identisch sind, sind in GSBL als verschiedene Stoffe registriert, weil dieser Unterschied von Bedeutung im Gefahrgutrecht ist. Stoffe, bei denen die Angaben zum Aggregatzustand und zur Zustandsform nicht vollständig sind, können zunächst nicht registriert werden. Registriert werden nur die Stoffe, die alle zur Identifizierung dienenden Daten enthalten. Anhand dieser Daten soll zuerst geprüft werden, ob der zu registrierende Stoff in GSBL schon vorhanden ist. In diesem Falle und wenn es nicht um ein Duplikat handelt, wird der vorhandene Stoff aktualisiert (neue Merkmale oder Felder hinzugefügt oder geändert). Sonst wird ein neuer Stoff in der Datenbank gespeichert und dabei einer neuen eindeutigen GSBL-Nummer zugeordnet.

Erfassung der Daten:
Die lokale Erfassung von Daten geschieht durch das Standardschnittstellenformat unter Einhaltung der spezifischen Richtlinien und Regeln.

Erstellung eines Primär-Schlüssels[14] mit Auto-Inkrement-Funktion:
Der Primärschlüssel soll vom System folgendermaßen erstellt werden: Trägt man einen neuen Datensatz in die Datenbank ein, prüft das System automatisch den Wert des Primärschlüssels, der zuletzt eingetragen wurde und zählt diesen Wert um eins hoch.

***Gewährleistung von Konsistenz des Stoffthesaurus*:**
Sollten die Identitätsmerkmale eines Stoffes sich ändern, müssen Referenzen auf diesen Stoff in den anderen Stoffen auch geändert werden.

Datenmanipulationen:
Das Datenbankmanagementsystem und die eingebundenen Treiber für Programmiersprachen sollen mindestens die grundlegenden Datenmanipulationen:

[13] Makroskopische Eigenschaften sind Eigenschaften, die mit bloßem Auge sichtbar sind
[14] Ein Primärschlüssel bezeichnet ein Feld oder eine Gruppe von Feldern, die einen Datensatz (eine Zeile in der relationalen Datenbank) eindeutig identifizieren

FIND(suchen), UPDATE(aktualisieren), DELETE(löschen), INSERT(hinzufügen), SORT (sortieren), und Datendefinitionen, sowie CREATE und DELETE (erstellen Tabellen, Datenbanken oder Indexen) in Bezug auf das konkrete Datenmodell ermöglichen.

Normalisierung:
Die Wertangaben sollen standardisiert werden. Referenzen auf Einheitentabellen mit Standardeinheiten existieren in der Merkmalstabelle mit dem GSBL-Datenmodell. Zulässigkeit von Standard- oder Normtexten ist mit Hilfe von Nachschlagetabellen zu überprüfen. Das Datenfile soll in „normalisierter Form“ erstellt werden.

Austausch von Informationen:
Der Austausch von Stoffinformationen zwischen dem GSBL und anderen Stoffdatenbanken erfolgt über eine standardisierte Schnittstelle. Über diese Standartschnittstelle soll sowohl inkrementelle als auch Gesamtlieferung der Daten als Import- und Exportverfahren gewährleistet werden. Export von Daten soll auch in den anderen Formaten möglich sein (XML, CSV, Excel-Tabellen).

Sicherheit der Daten gewährleisten:
- *Authentifizierung*: einen Passwort-Schutz einrichten (Benutzername, Passwort) um die Daten vor fremdem Zugriff sichern zu können;
- *Zugriffskontrolle mit Autorisierung*: festlegen, welche Personen Zugriff auf welche Informationen und Aktionen besitzen;
- *Konsistenz der Daten:* Korrektheit der Daten garantieren;
- *Datensicherung:* Sicherungskopien von Daten erstellen um im Falle eines Datenverlustes die Originaldaten aus diesen Kopien wiederherstellen zu können;

Benutzerschnittstellen.
Der Datenzugriff über das Internet und Intranet[15] soll entsprechend den Usability[16]-Standards möglich sein.

Das Bild 1 veranschaulicht die Zustände der Datenbank mit den Ereignissen, die zu dem jeweiligen Zustandsübergang führen.

Q0-Ausganszustand. Wenn es sich bei einem neuen Stoff um ein Duplikat handelt oder die Merkmale, die diesen Stoff eindeutig identifizieren, nicht vollständig sind, bleibt die Datenbank unverändert.

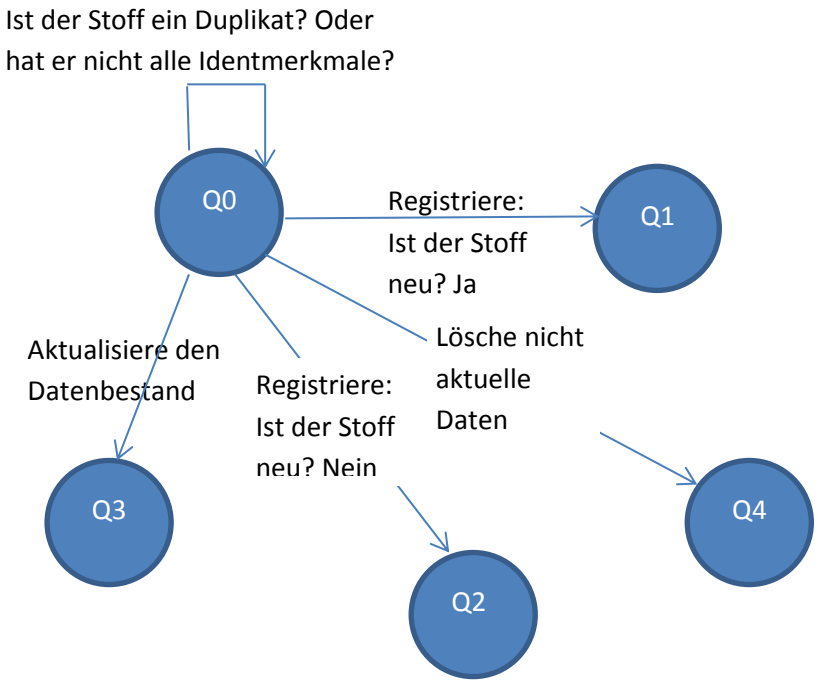
Q1-Wenn ein neuer Stoff registriert wird und er noch nicht in der Datenbank existiert, geht die Datenbank in den neuen Zustand „Einfügen eines neuen Stoffes“ über.
Q2-Wenn ein neuer Stoff registriert wird und er schon in der Datenbank existiert, geht die Datenbank in den neuen Zustand „ Aktualisiere die Daten des Stoffes“.

Q3-Nach der regelmäßigen Aktualisierung ist die Datenbank in einem neuen Zustand

Q4-Wenn die Daten veraltet sind, werden sie gelöscht

[15] Intranet ist im Gegensatz zum Internet ein nicht öffentliches Rechnernetz
[16] Usability - Benutzerfreundlichkeit

Abbildung 1: Zustände des GSBL



2.4 Datenbankmodell: Das Entity-Relationship-Modell (ERM) plus SSF-Format

Das Datenmodell beschreibt die Informationsstruktur einer Anwendung. „Ein Datenbankmodell ist die theoretische Grundlage für eine Datenbank und legt fest, auf welche Art und Weise die Daten in dem Datenbanksystem gespeichert und bearbeitet werden können“ [Datenbanken-verstehen, 2014]. Die Informationsstruktur umfasst statische Eigenschaften der Daten, wie Objekte[17] und Beziehungen zwischen diesen Objekten, dynamische Eigenschaften, wie Operationen und Beziehungen zwischen Operationen und Integritätsbedingungen. Objekte sind z.B. Stoffe und Lieferanten, die Beziehung besteht darin, dass ein Lieferant die Daten für einen Stoff liefert. Operationen auf Objekten sind Registrieren eines Stoffes oder Standardisierung der Wertangaben. Beziehungen zwischen Operationen können festlegen, dass die Registrierung eines Stoffes nur nach der Normalisierung seiner Daten abgeschlossen ist. Die Integritätsbedingung legt fest, dass GSBLRN (GSBL-Registriernummer) eindeutig sein muss oder dass die Eigenschaft von einem Stoff, die „Lieferant“ heißt, einen Lieferanten referenziert.

Eine solche Informationsstruktur wird auf einer abstrakten, implementierungsunabhängigen Ebene mit Hilfe einer graphischen Notation beschrieben. Das **Entity-Relationship-Modell (ERM)** ist ein gängiger Standard für eine graphische Darstellung des Datenbankmodells. Es veranschaulicht, welche Objekte in einer Datenbank gespeichert sind, welche Eigenschaften diese Objekte haben, welche Beziehungen zwischen den Objekten und Eigenschaften dieser

[17] Objekte sind in einer Datenbank nicht direkt sondern über ihre Eigenschaften darstellbar [Saake, 2010]

Beziehungen existieren. Bei einer NoSQL-Datenbank, im Speziellen bei der dokumentorientierten Datenbanken, ist das Modellieren mittels ERM nicht ausreichend, denn Eigenschaften (Attribute), die ein Objekt charakterisieren, werden wiederum in mehrere Attribute unterteilt. Das Objekt „Stoff“ etwa, ist durch viele Merkmale gekennzeichnet, die in Felder unterteilt sind. Ein Feld ist oft eine Folge von Attributen. Deswegen wurde hier das GSBL-Datenbankmodell für NoSQL-Datenstruktur auf der Grundlage von ERM unter Einbeziehung des standardisierten SSF-Formates gebildet.

Das **SSF-Format** dient dem Datenaustausch von Stoffinformationen zwischen GSBL und anderen Stoffdatenbanken oder Stoffanwendungen in einer einfach lesbaren Textform. Die Daten zu Objekten gleicher Art sind zu einem Segment zusammengefasst. Folglich gibt es Segmente MERKMALE, LIEFERANT, SPEZIES, ZITATE, ROSDAL. Eine Zeile der Form [Segmentname] leitet ein Segment ein. Ein Segment ist in mehrere Abschnitte, die genau ein Objekt beschreiben, unterteilt. Die folgenden fettmarkierten Zeichenfolgen dienen zu einer weiteren Untergliederung und beginnen immer eine neue Zeile:

@**.** eröffnet einen Abschnitt, der Daten zu einem Stoff, einem Lieferanten, einer Spezies oder einem Zitat enthält.

@**;Merkmalsname** eröffnet einen Abschnitt, der Daten zu einem Merkmal enthält. Der Abschnitt darf nicht leer sein.

@**:Feldname = Feldinhalt** bildet den Name des Feldes und seines Inhaltes ab. Falls das Zeichen ‘@‘ im Inhalt steht, dann ist er zu verdoppeln. Ein Feldname und ein Feldinhalt bilden eine Zeile.

@**!** Dadurch werden Felder, die zusammen gehören, eingeleitet und begrenzt. Die werden **Subsets** genannt. Der Ausschnitt aus der SSF-Datei zeigt die obengenannte Struktur.

```
@!
@:VA_LITUWRT=12800
@:VA_LITUWRTEIN=mg/kg
@:VA_LITWRTSTR=12800 mg/kg
@:WA=|LD50
@!
```

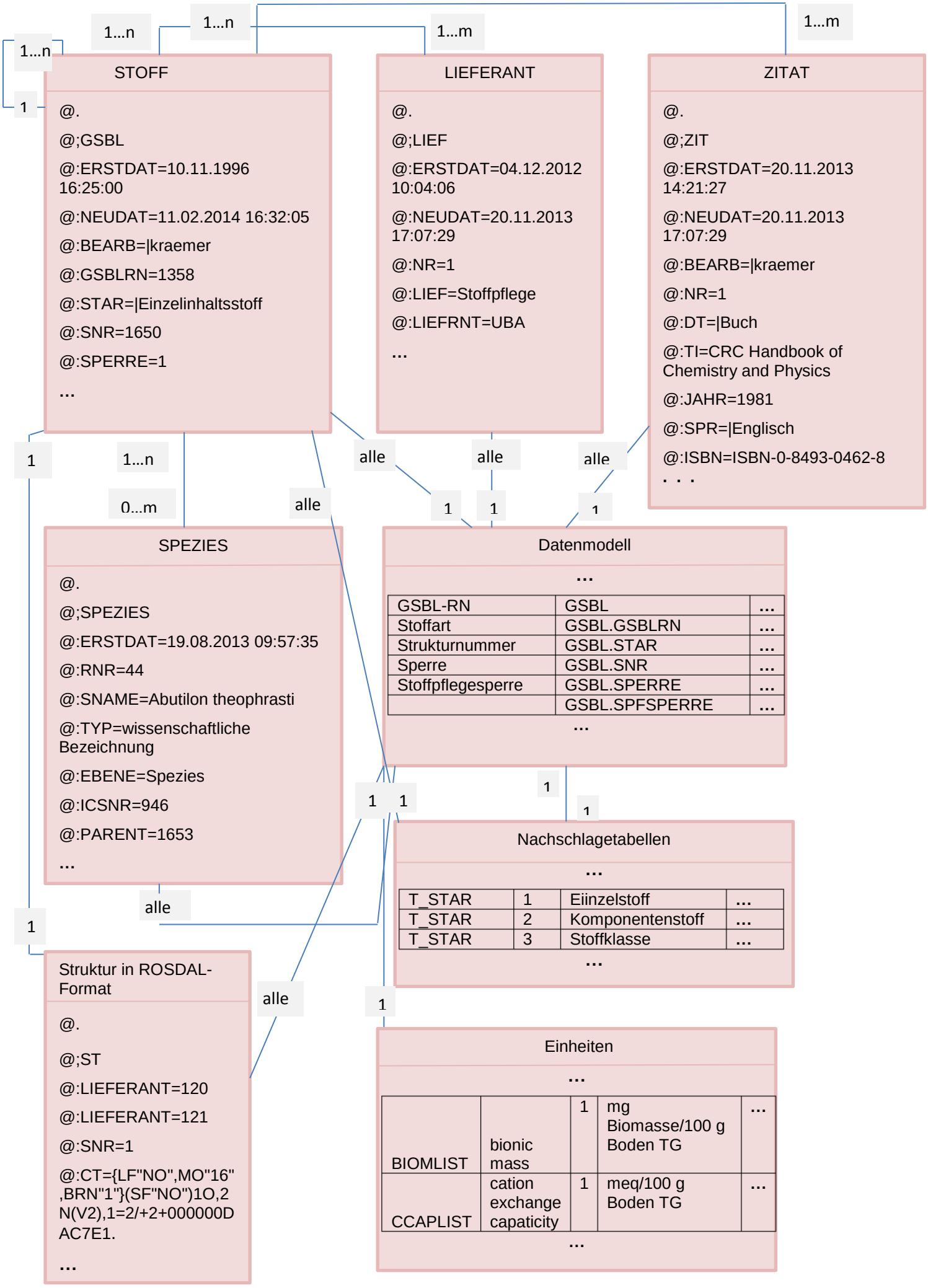
Das nachfolgende Modell (Abbildung 2) bildet alle Objekte der Datenbank und Relationen zwischen ihnen ab. Die Eigenschaften der Objekte sind mit der Notation des SSF-Formates versehen. Abgebildet sind der Anfang eines Objektes (genau genommen der Anfang der Beschreibung des Objektes, dass mit der Zeichenkette @. anfängt) und die Felder zu einem seiner Merkmale. Im Falle der umfassenden Tabellen sind Ausschnitte aus diesen Tabellen dargestellt, die den gesamten Aufbau dieser Tabellen wiedergeben.

Die Beziehungen zwischen Objekten werden mit Hilfe von Verbindungslinien dargestellt. So enthält eine zusammenfassende Tabelle mit dem Datenmodell Referenzen auf die Nachschlagetabellen und auf die Einheitentabelle. Das Objekt „Stoff“ hat Beziehung zu Objekten „Lieferanten“, „Spezies“ und „Zitate“. Der Typ dieser Beziehungen wird mit Hilfe von Kardinalitäten dargestellt. Das bedeutet, für jeden Beziehungstyp gibt man an, wie viele Objekte einer Art mit jeweils einem Objekt anderer Art in Verbindung stehen.

Zum Beispiel kann ein Lieferant Informationen zu einem oder mehreren Stoffen zur Verfügung stellen. Das wird als **1…n -** Beziehungstyp bezeichnet und wird an der Seite des Stoffes eingetragen. Ein Stoff steht wiederum in Beziehung zu einem oder mehreren Lieferanten, da die Daten eines Stoffes von einem oder mehreren Lieferanten geliefert werden. Das wird als **1…m -** Beziehungstyp bezeichnet und wird an der Seite des Lieferanten eingetragen. Das Objekt „Datenmodell“ enthält alle Merkmale mit allen möglichen Feldern, die Stoffe im GSBL (bzw. Lieferanten, Spezies, Zitaten und Strukturen) beschreiben können. Damit hat es eine Beziehung zu allen Stoffen, Lieferanten, Spezies, Zitaten und Strukturen im GSBL. Zu einem Stoff jedoch existiert nur ein „Datenmodell“-Objekt. Deswegen wird an der Seite des „Datenmodell“-Objektes der Beziehungstyp - **1** und an der Seite des Stoffes Beziehungstyp - **alle** eingetragen. Ein Stoff hat Beziehungstyp **1** zu **1…n**, weil ein Stoff Referenzen auf einen oder mehrere verwandte Stoff enthält.

Die Strukturdaten im MDL- MOLFILE-Format sind auf der Abbildung 2 nicht dargestellt, da sie nur dann ausgewertet werden, wenn keine Strukturdaten im ROSDAL-Format vorliegen.

Abbildung 2: Datenbankmodell



3 Alternativen zu einem relationalen Datenmodell und NoSQL

3.1 Auswahl einer Datenbank und das relationale Datenbank-Konzept

Das wichtigste Kriterium für die Auswahl einer Datenbank ist das zugrundeliegende Datenmodell. Welche Datenbanken sind in Bezug auf GSBL-Datenmodell in Betracht gezogen werden können? Welche Aspekte müssen dabei berücksichtigt werden? Die weiteren Faktoren, die für die Auswahl eines Datenbankmanagementsystem wichtige Rolle spielen, sind: Datenvolumen und Häufigkeit des Eintreffens neuer Daten.

Das relationale Datenbank-Konzept existiert seit langem, es wurde ständig optimiert und weiterentwickelt, es ist leicht zu nutzen, und deswegen ist auch heute oft die erste Wahl für verschiedene Datenbank-Lösungen. Ein weiterer Grund für die Befürwortung einer relationalen Datenbank ist es, dass die Tabellen sehr anschaulich sind und mit SQL[18] ein mächtiger Mechanismus für die Verwaltung der Daten benutzt wird. Klassische Anwendungsgebiete zeichnen sich durch einfach strukturierte Datensätze aus, wie etwa in der Personaldatenverwaltung, Lagerverwaltung oder Buchhaltung. Wenn sich Informationen zu den Daten in Tabellen modellieren lassen, sind für die Datenspeicherung relationale Datenbanksysteme gut geeignet. Alle ähnlichen Objekte müssen in diesem Falle die gleiche Struktur (Objekte werden durch gleiche Eigenschaften (Attribute) charakterisiert) aufweisen. Dann kann man für die Speicherung ihrer Daten eine Tabelle definieren, deren Spalten den Eigenschaften (Attributen) der Objekte entsprechen. Da die GSBL-Objekte sehr unterschiedliche Strukturen haben, war es nicht möglich ihre Daten auf solche Weise zu speichern. Folglich musste man die umständliche - derzeit benutzte - transponierte Variante implementieren. Bei dieser Lösung braucht man viele Verknüpfungen zwischen den Zeilen in einer oder mehreren Tabellen oder komplizierte Abfragen um das Ergebnis einer Suche korrekt zu gewährleisten. Für die Wahl eines relationalen Datenbanksystems ist es auch wichtig, dass nicht viele und nicht sehr große Tabellen zusammengeführt werden müssen(diese Funktion einer relationalen DMBS heißt JOIN, auf Deutsch: Verbinden). Deswegen ist die Lösung, bei der GSBL-Daten als viele kleine oder mittelgroße Tabelle abgelegt würden, nicht akzeptabel.
Es ist oft der Fall, dass zusammengehörende Informationen nicht zusammen abgelegt sind. Üblicherweise sind zu speichernde Objekte komplex, was bedeutet, dass sie sich selbst ebenfalls aus mehreren Objekten oder Listen zusammensetzen. Im GSBL sind die Stoffdaten genauso komplex. Das relationale Datenmodell kennt aber nur aus Werten zusammengesetzte Mengen (Tupel), die ein Objekt beschreiben. Dabei müssen diese Werte atomar (nicht in die andere Werte zerlegbar) sein. Die nicht in Tabellen passenden komplexen Objekte, das schlecht übersichtliche Datenschema, komplizierte Anfragen mit vielen JOINs verlangten nach einer Alternative, was zur Entwicklung objektorientierter Datenbanksysteme führte. Außerdem sind Anwendungsentwicklungen zunehmend objektorientiert und Objekte lassen sich schwer auf die Tabellen abbilden.

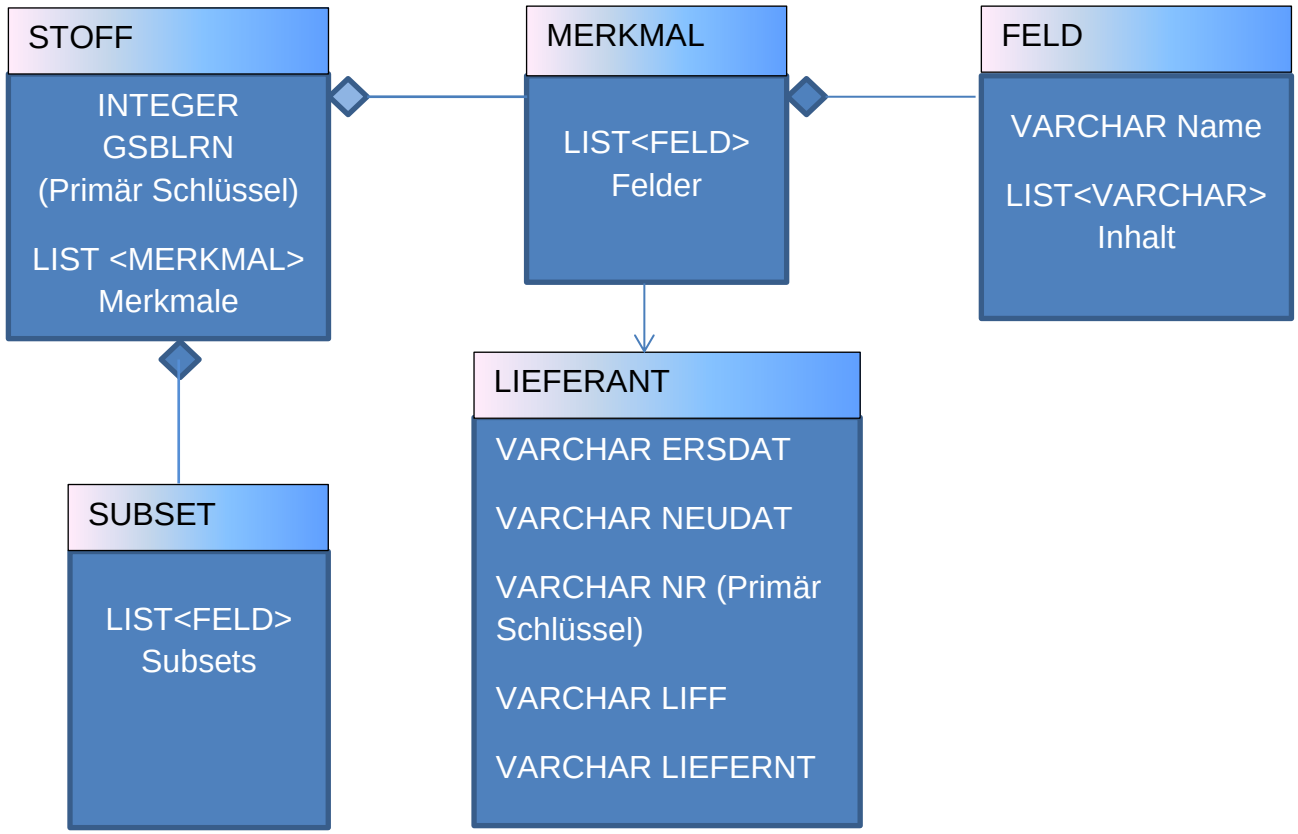
[18] SQL steht für *Structured Query Language* und mit ihr kann man Daten aus einer Datenbank verwalten

3.2 Objektorientierte Datenbanken

Objektorientierte Datenbanken können komplexe Objekte viel präziser abbilden, weil sie komplexe Datentypen speichern können. Während relationale Datenbanken starre Tabellenzeilen verwalten, besteht ein Datensatz in einer objektorientierten Datenbank aus einem komplexen Objekt. Im Gegensatz zu relationalen Datenbanksystemen ist das Ergebnis einer Abfrage nicht eine Menge von Datensätzen sondern einzelner Objekte. Das DBMS „kennt“ die Daten, welche zusammengehören. Damit wäre das Problem der Speicherung zusammengehörender Stoffdaten insofern geklärt, dass die aufwändigen Verknüpfungen wegfallen. Man könnte auf die Objektdaten direkt zugreifen. Mit dem Konzept der benutzerdefinierten Datentypen bieten Objektdatenbanksysteme ein Konzept zur Erweiterung des Datenmodells um solche anwendungsspezifischen Datentypen. Diese Datentypen definiert man nur einmal und kann sie danach wiederverwenden. Sie können auch andere benutzerdefinierte Subtypen haben. Dieses Konzept entspricht dem Klassenkonzept der objektorientierten Programmierung[19]. Es wäre vorstellbar, solche Datentypen (auch Objekttypen) im GSBL zu definieren wie Stoff, Merkmal, Feld und Subsets. Dann wären der Objekttyp MERKMAL ein Subtyp des Objekttyps STOFF und Objekttyp FELD ein Subtyp von MERKMAL. Das FELD enthält einen einfachen Datentyp[20] VARCHAR[21] und einen konstruierten Datentyp LIST (eine genauere Erläuterung erfolgt im Abschnitt objektrelationale Datenbanken). Die Abbildung 3 zeigt die möglichen Datentypen für eine objektorientierte GSBL-Lösung. Es sind fünf Objekttypen abgebildet mit den zugehörigen Attributen: STOFF, MERKMAL, FELD, SUBSET, LIEFERANT. Auf die Darstellung der Objektmethoden wurde hier verzichtet. Die anderen Datenbankobjekte wie Spezies, Strukturen usw. kann man analog aufbauen.
Die Beziehungen zwischen Objekten oder wie sie in der Unified Modeling Language (UML) heißen, Assoziationen, sind mit den Verbindungspfeilen dargestellt. Ein Pfeil mit einer ausgefüllten Raute am Ende zeigt, dass es sich um eine bestimmte Assoziation, um eine Komposition, handelt. Es beschreibt die Beziehung zwischen dem Ganzen und den Teilen, bei der die Teile von der Existenz des Ganzen abhängig sind. So existieren Merkmale nicht ohne Stoff, und Felder existieren nicht außerhalb eines Merkmals. Zwischen dem Merkmal und dem Lieferanten gibt eine unidirektionale Assoziation. Damit ist gemeint, dass man von einem Objekt MERKMAL zu einem Objekt LIEFERANT navigieren kann. Das Merkmal „kennt“ seinen Lieferanten.

[19] Unter einer **Klasse** (auch Objekttyp genannt) versteht man in der objektorientierten Programmierung ein abstraktes Modell bzw. einen Bauplan für eine Reihe von ähnlichen Objekten [Klasse, 2014].
[20] **einfache Datentypen** können nur einen Wert des entsprechenden Wertebereichs aufnehmen [Datentyp, 2014]. Im Bereich SQL-Datenbanken gehören zu den einfachen, Basisdatentypen unter anderen INTEGER, CHARACTER, VARCHAR, FLOAT, DECIMAL.
[21] **VARCHAR**-Datentyp speichert Buchstaben, Zahlen wie 1,2,3 und Sonderzeichen wie Ausrufzeichen

Abbildung 3: Objekttypen des GSBL



Zusammen mit den Objekten können Objektmethoden hinterlegt werden, die Manipulationen auf eben diesen Objekten ausführen. Es kann zum Beispiel die Methode Get_Referenzen() definiert werden, die die Eigenschaften der verwandten Stoffe liefert. Da viele Anwendungen in einer der objektorientierten Programmiersprachen, z.B. Java, erstellt sind, ist eine direkte Kommunikation zwischen den Anwendungen und der Datenbank ohne Abbildung der Objekte auf die relationale Tabellenstruktur, das sog. objektrelationale Mapping, möglich. Die schnellere Entwicklung von Anwendungen ist durch Wiederverwendung von in der Datenbank gespeicherten Funktionen gewährleistet. Verschiedene Entwicklungswerkzeuge basieren auf Objekttechnologie und gewährleisten die Entkoppelung von Anwendungsprogrammen und Benutzeroberflächen.

Nachteile: Aufgrund der geringen Verbreitung objektorientierter Datenbanken, sind Tools und Schnittstellen (JDBC/ODBC[22], ETL[23] oder OLAP[24]) für den Einsatz mit

[22] JDBC/ODBC sind Datenbankschnittstellen. Dabei handelt es sich um eine Bibliothek von Prozeduren, die aus der jeweiligen Programmiersprache heraus aufgerufen werden können und dann den Zugriff auf die Datenbank ermöglichen. ODBC stellt eine C/C++-Klassenbibliothek zur Verfügung, es lassen sich aber auch z.B. Fortran oder Visual Basic anbinden. JDBC die Standardschnittstelle für den Zugriff auf SQL-Datenbanken aus einer in Java programmierten Anwendung.

Objektdatenbanken nicht vorbereitet. Es gibt zwar verschiedene Sprachen, die für objektorientierte Datenbankverwaltungssysteme benutzt werden, aber einen allgemeinen Standard, wie ihn SQL für relationale Datenbanken bildet, gibt es noch nicht. Die nicht ausreichend standardisierte Anfragesprache wie OQL[25] kann man mit der Mächtigkeit von SQL nicht vergleichen. Falls viele Daten zu Objekten redundant gespeichert werden, kann dies zu hoher Speicherauslastung führen. Wenn bei der Implementierung objektorientierter Datenbanken viele Zeiger (Referenzen auf andere Objekte) entstehen, kann es bei den Änderungen an Objekten zur Beeinträchtigung der Performanz kommen. Besonders kritisch ist dies, wenn neben den Zeigern auch die Fremdschlüssel[26] benutzt werden.

Wichtige Vertreter sind: Versant FastObjects, Cache von Intersystem, Db4objects usw. Cache von Intersystem unterstützt sowohl JDBC/ODBC-Treiber als auch SQL. Db4objects kann in einer jar-Datei in ein Anwendungsprogramm eingebunden werden. Dann laufen das Programm und die Datenbank in einer Java-Virtuellen-Maschine. Db4objects ist unter der GPL als freie Software lizenziert solange sie nicht in den kommerziellen Projekten gebraucht wird.

3.3 Objektrelationale Datenbanken

Objektrelationale Datenbanken verschmelzen die Konzepte der relationalen Datenbanken mit denen der objektorientierten. Besonders relevant ist, dass die Daten nicht mehr atomar sein müssen, sie können aus Listen oder anderen Objekten zusammengesetzt werden, und es können eigene Typen definiert werden. Damit können komplexe Objekte in einer Tabelle gespeichert werden. Speicherverwaltung, Abfrageoptimierung, Verfügbarkeit der Daten, Zugriffskontrolle sind besser als bei der objektorientierten Datenbanken. Den Vorteil brachte auch die übernommene Standardkonformität von SQL. Eine eindeutige, unveränderliche Objektreferenz(OID) ermöglicht genauso wie in den objektorientierten Datenbanken die Trennung der Identifikation der Objekte von ihren Werten und damit die Unterscheidung zwischen identischen und gleichen Datenbankobjekten. Diese OID wird nur für interne Identifizierung verwendet und ist somit nicht sichtbar. Um die ausgereifte Datenbanktechnologie auf die Verwaltung beliebiger Arten von Daten auszudehnen, bieten die Datenbanken besondere Konzepte:

- **Typkonstruktoren**, die aus Basisdatentypen oder anderen komplexen Datentypen neue konstruierte Datentypen erzeugen, so wie Listentypkonstruktor.
- **Benutzerdefinierte Datentypen**, die genauso wie in objektorientierten Datenbanken anwendungsspezifische Objekte beschreiben. Benutzerdefinierte Datentypen sind strukturiert oder distinkt. Distinkte sind dafür da, um sinnlose Vergleiche zwischen den Objekten auszuschließen.

---

[23] ETL ist ein Prozess, bei dem Daten aus mehreren ggf. unterschiedlich strukturierten Datenquellen in einer Zieldatenbank vereinigt werden.

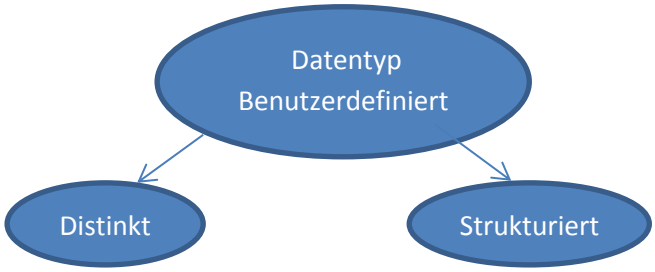
[24] Unter dem Begriff OLAP werden Technologien zusammengefasst, die Analyse von Daten unter unterschiedlichen Gesichtspunkten ermöglichen

[25] OQL (Object Query Language) ist eine an SQL angelehnte Abfragesprache für Objektdatenbanken

[26] Fremdschlüssel ist der Attribut, der auf einen Datensatz in einer anderen Tabelle verweist

Strukturierte Datentypen sollen das Konzept des Objekttyps aus der Objektorientierung im objektrelationalen Modell realisieren. Analog zum Objekttyp besitzt ein strukturierter Typ Attribute (Eigenschaften eines Objekts), und Methoden, die das Verhalten beschreiben [FH Köln, 2010].

Abbildung 4:Objekttypen des GSBL



- **Erweiterte Tupeltabellen**, die auch komplexe Werte enthalten können. Tabelle 4 illustriert dies graphisch. Neben dem Basisdatentyp VARCHAR wird einen Listentyp LIST(INTEGER) definiert. Dabei handelt es sich um eine geordnete LISTE von Integer-Datentypen, deren maximale Anzahl nicht feststeht. Auch Duplikate von Elementen sind erlaubt. Damit kann man zum Bespiel Lieferantennummer oder andere Feldinhalte, die aus mehreren Werten bestehen, speichern. In der Tabelle stehen die Felder, die zu einem Merkmal „RNAME“ gehören.

Tabelle 4: Erweiterte Tupeltabelle in einer objektrelationalen Datenbank ohne OID-Spalte

Erstdat	Neudat	Bearbeiter	Qualität	Lieferanten
VARCHAR	VARCHAR	VARCHAR	VARCHAR	LIST(INTEGER)
‘17.07.2002 10:42:18‘	‘25.11.2013 10:55:56‘	‘Franke‘	‘GSBL‘	18,359,660,728,735
Wert des strukturierten Typs RNAME				

- **Typisierte Tabellen**, auch als Objekttabellen bezeichnet, in denen die Zeilen Objekte repräsentieren. Die Zeile in der Tabelle 4 könnte man als Datentyp RNAME zusammenfassen. In Standard-SQL ist die erste Spalte einer typisierten Tabelle immer die OID-Spalte, weil ein Objekt eindeutig identifiziert werden muss. Solche Tabelle kann als Subtabelle einer anderen typisierten Tabelle sein. Der Typ der Objekte einer Subtabelle ist ein Subtyp des Typs der Objekte der Supertabelle [Türker, 2005].

So kann man in der Datenbank-GSBL eine Tabelle mit allen Merkmalen als Spaltennamen definieren. Dabei muss man prüfen, ob die Anzahl der Spalten begrenzt ist. Die Objektrelationale Datenbank PostgreSQL kann, zum Beispiel, unbegrenzt viele Datensätze enthalten, die maximal 1.6 TB groß sein können. Doch die Anzahl der Spalten ist auf 250 - 1600 begrenzt, je nachdem, welche Datentypen

verwendet wurden. Es besteht die Möglichkeit die maximale Anzahl von Spalten zu vervielfachen, indem man die Default-Blockgröße auf 32 KB heraufsetzt [PostgreSQL9, 2013].
Es ist selten, dass ein Stoff durch alle im GSBL existierenden Merkmale beschrieben wird. Wenn ein Merkmal für einen konkreten Stoff nicht relevant ist, wird in dem entsprechenden Feld der NULL-Wert stehen. Er findet dort Verwendung, wo ein Eintrag entweder nicht möglich oder nicht bekannt ist. Was für ein Eintrag soll in dem Feld stehen, wenn ein Stoff dieses Merkmal hat? Denn ein Merkmal besteht aus vielen Feldern, die nicht nur einen Wert sondern eine Liste von Werten haben können. Ein Merkmal kann auch die zusammengefassten Felder (Subsets) enthalten. Für so eine Struktur bietet es sich an, eine eingebettete Tabelle zu definieren, die den Wert des Merkmals darstellen kann. Spalten dieser Tabelle sind die zu diesem Merkmal zugehörigen Felder und Zeilen - die Merkmale mit demselben Namen, der dem Spaltennamen entspricht. Somit zeigt die Anzahl der Zeilen, wie oft das Merkmal bei einem entsprechenden Stoff auftritt (Tabelle 5).

Tabelle 5 verdeutlicht die Beziehung zwischen der Haupttabelle und einer eingebetteten Tabelle. Als eindeutige identifizierende Nummer (Primärschlüssel oder auf Englisch: Primary Key) wird GSBLNR definiert. Man sieht, dass das Merkmal RNAME bei dem Stoff mit GSBLRN=2 drei Mal auftritt (in der Wirklichkeit ist das noch mehr), weil die eingebettete Tabelle drei Zeilen hat. Zum Erstellen einer typisierten Tabelle muss man zuerst den Datentyp ihrer Zeilen definieren. Unter Nutzung von SQL sieht das folgendermaßen aus:

```
CREATE OR REPLACE TYPE RNAME_t AS OBJECT (
   ERSDAT   VARCHAR2(40),
   NEUDAT   VARCHAR2(40),
   BEARBEITER     VARCHAR2(40)
   QUALITÄT VARCHAR2 (20)
   RNAME VARCHAR2 (50)
   LIEFERANTEN LIST (INTEGER)
)
```

Danach wird die Tabelle von diesem Typ (in unserem Beispiel RNAME_t) erstellt. Der Befehl kann so aussehen:

```
CREATE OR REPLACE TYPE RNAME_nt   AS TABLE OF RNAME_t
```

Im Typ RNAME_nt deutet „nt“ darauf hin, dass es sich um eine eingebettete Tabelle handelt (auf Englisch: Nested Table).

Diese Lösung macht es nicht nur möglich, die Objekte (bei uns Felder mit Feldinhalten) direkt abzurufen und mit Hilfe der SQL-Manipulationssprache oder angelegter Prozeduren diese Objekte zu verwalten, sondern beim Import oder Überprüfen der Datenkorrektheit die fehlerhafte Zuordnung von Feldern den Merkmalen zu vermeiden. Allerdings hat man bei dieser Lösung viele mit NULL-Werten besetzte Felder.

Tabelle 5: Objektrelationale Datenmodell mit einer eingebetteten Tabelle(prüfen)

GSBLRN(Primary KEY)	*GSBL*	*RNAME*	*NAME*	*DISPNAME*	*...*
INTEGER	*TABELLE*	*TABELLE*	*TABELLE*	*TABELLE*	*...*
2					*...*

Erstdat	Neudat	Bearbeiter	Qualität	RNAME	Lieferanten
VARCHAR	VARCHAR	VARCHAR	VARCHAR	VARCHAR	LIST(INTEGER)
‘17.07.2002 10:42:18‘	‘25.11.2013 10:55:56‘	‘Franke‘	‘GSBL‘	2-Nitro-1-aminobenzol	18,359,660,728,735
‘08.09.2005 11:50:14‘	‘25.11.2013 10:55:56‘	‘Franke‘	NULL	o-(ortho)Nitro-anilin	12,18
17.07.2002 10:42:18	17.07.2002 10:42:18	NULL	GSBL	2-nitroaniline	4

Manche objektrelationale Datenbanken wie PosgreSQL können außerdem solche Datentypen wie Arrays, hstore, XML und JSON speichern, was eine Umkehr von festen relationalen Strukturen bedeutet. Der Datentyp Hstore speichert die Schlüssel-Wert-Paare an der Stelle eines Datenbankwertes.

3.4 NoSQL Datenbanken

NoSQL-Datenbanken (richtige Interpretation: Not Only SQL) traten in den Vordergrund als relationale Systeme an ihre Grenzen stießen. Zu Zeiten des Wandels der Informationsnutzung wachsen die Mengen der zu analysierenden Daten drastisch und erreichen Tera- und Peta-Datenbereich. Es entsteht höhere Anforderung an die Laufzeiten von Anfragen und an die Skalierungsfähigkeit der Daten. Unter Skalierungsfähigkeit versteht man die Eigenschaft des Datenbanksystems (oder eines anderen Softwaresystems) mit dem Wachstum an Daten selber zu wachsen. Relationale Datenbanken sind vorwiegend vertikal skalierbar. Sie versuchen die effiziente Verarbeitung der großen Datenmengen durch Aufrüsten des Servers mit besseren Softwarekomponenten zu erreichen. NoSQL-Datenbanken sind gut horizontal skalierbar, was die Verteilung der Daten auf verschiedene Server bedeutet. Das erhöht die Verfügbarkeit der Daten und macht die Datenbank robust beim Ausfall eines Servers. Obwohl es keine einheitliche Definition von NoSQL-Datenbanken gibt, weisen sie einige gemeinsame Eingenschaften auf. Außer der schon erwähnten horizontalen Skalierbarkeit haben sie kein relationales Datenbankmodel und Schemafreiheit[27]. Sie stellen keine JOINS

[27] Ein Schema einer Datenbank definiert die Struktur der Daten, z.B. Anzahl und Typ von Attributen. Schemafreiheit bedeutet nicht, dass die Daten keine Struktur haben, sondern, dass man zum Beispiel Datensätze speichern kann ohne zuvor Spaltennamen und ihren Typ zu definieren oder Anzahl der Spalten bei jedem Datensatz gleich zu halten.

zu Verfügung und fordern keine Normalisierung der Daten (Forderung der „Nichtzerlegbarkeit“ der Daten). Der Zugriff auf die Daten erfolgt nicht mehr mit Hilfe von SQL oder nicht nur mit SQL und es gibt kein Konzept Transaktionen[28]. Einige Systeme unterstützen statt Transaktionen die Idee von "Eventual Consistency". Darunter versteht man, dass die Datenbank im Normalbetrieb nach einer gewissen Zeit (typischerweise im Sekundenbereich) immer wieder einen konsistenten Zustand erreicht, das heißt alle beteiligten Knoten über den gleichen Datenstand verfügen. Im Unterschied zu den gängigen SQL-Datenbanken unterscheiden sich die NoSQL-Datenbanken untereinander sehr hinsichtlich des unterstützten Datenmodells. [Klimt, 2013].

Key-Value-Datenbanken (Schlüssel-Wert-Datenbanken)

Der Aufbau von Key-Value-Datenbanken ist ziemlich einfach. Ein bestimmter Schlüssel identifiziert eindeutig einen Wert (Key/Value Paar). Der Schlüssel kann eine strukturierte oder willkürliche Zeichenkette sein. Dabei können die Werte (Value) außer String auch Listen, Sets oder auch Hashes enthalten. Die größten Vorteile von Key-Value-Datenbanken ist ihre meist enorme Geschwindigkeit und Skalierbarkeit. So erreicht z.B. die Key-Value-Datenbank Redis Laufzeiten von O(1)[29] für Lese- und Schreiboperationen. Dafür muss man allerdings auch auf komplexe Suchalgorithmen oder Indizes verzichten [Key-Value, 2011]. Key-Value-Datenbanken halten möglichst alle Einträge neben der Sicherung auf der Festplatte auch gleich im RAM vor.

Key-Value-Stores lassen sich dadurch in zwei Gruppen unterteilen: Die In-Memory-Variante behält die Daten im Speicher und sorgt dadurch für eine hohe Performanz. Die On-Disk-Versionen speichern ihre Daten direkt auf der Festplatte. In-Memory-Datenbanken bieten sich als verteilte Cache-Speichersysteme an; On-Disk-Datenbanken werden als Datenspeicher genutzt [Redis, 2014].

Ein Vertreter von Key-Value-Datenbanken - Redis - kombiniert beide Varianten. Der Zugriff auf einen Datensatz erfolgt lediglich über dessen Schlüssel. Ist der Wert eine komplexe Struktur, wie z.B. eine Liste oder ein Objekt, analog zu den Werten von Merkmalen oder Feldern im Falle von GSBL, wird die Struktur unübersichtlich. Key-Value-Datenbanken verarbeiten schnell einfache Abfragen, eine Formulierung komplexere Abfragen ist oft nicht möglich. Die Einfachheit des Datenmodells bei großen Datenmengen und wenig Relationen zwischen den Daten sind die Hauptargumente für den Einsatz von Key-Value-Datenbanken. Wenn man zu einem Schlüssel den ganzen Datensatz haben will, also wenn es um die Ablage und Abfrage von Daten in immer derselben Form geht, ist diese Art von Datenbanken gut geeignet. Anwendungsfälle sind: Werbung auf Webseiten, wo schnell entschieden werden muss, welche Daten präsentiert werden, Verwaltung von Session-Daten in einem Websystem, Warenkörbe in E-Commerce-Applikationen[30], ein Verzeichnisdienst im Internet wie beispielsweise ein Wörterbuch mit stetig wachsender Datenmenge, das über die Webseite bereitgestellt wird. Die Recherche und Bearbeitung von Stoffdaten im GSBL erfordern hingegen verschiede Arten von

[28] Eine Transaktion bezeichnet eine Menge von Datenbankänderungen, die zusammen ausgeführt werden müssen.

[29] Unter O(1) wird f(x) =O(1) gemeint, was bedeutet, dass die Funktion f(x) für große x beschränkt bleibt.

[30] Unter E-Commerce im weiteren Sinne werden alle Formen der elektronischen Geschäftsabwicklung über öffentliche und private Computer- bzw. Telekommunikations-Netze verstanden (Hermanns/Sauter 2001, S. 8). [E-Commerce, 2013]

Abfragen, die viel komplexer sind, als nur den ganzen Stoff nach seiner Identifikationsnummer zu suchen.

Wide-Column-Stores (Spaltenorientierte Datenbanken)
Wide-Column-Stores gruppieren die Daten mehrerer Datensätzen nach ihren Spalten und nicht nach ihren Zeilen wie bei relationalen Datenbanken. Diese Art von Datenbanken eignet sich für große Datenvolumen, weil die spaltenorientierte Datenstruktur gut auf mehrere Server verteilt werden kann. Zeilenorientierte Datenbanken hängen alle Datenwerte einer Zeile aneinander, dann alle Datenwerte einer anderen Zeile aneinander und so weiter. Spaltenorientierte Datenbanken gehen hingegen analog mit den Spaltendaten vor.

Wide-Column-Stores können viele Lesezugriffe bewältigen und einen Zeitstempel haben. Der Leseprozess wird dadurch beschleunigt, dass keine unnötigen Informationen gelesen werden, weil ein Zugriff immer nur auf relevante Spalten stattfindet. Der Schreibprozess ist ebenso schnell, wenn es um eine einzelne Spalte geht.

Wenn es allerdings nötig ist eine Vielzahl von Daten zu schreiben, die nicht nur einer Spalte zugeordnet sind, muss auf alle diese Spalten zugegriffen werden, was den Schreibvorgang verlangsamt. Ein weiterer Nachteil ist, dass beim Lesen aller zusammengehörigen Daten (z.B. Stoffdaten) zwischen den Spalten hin- und her gesprungen werden muss. Das ist sehr aufwendig.

Die Spalten bestehen aus ihrem Namen, den Daten und dem Zeitstempel. Zusammenhängende Spalten (Columns) bilden eine so genannte Column-Family. So kann man sich eine Column-Family „GSBL“ vorstellen, die einem Merkmal GSBL entspricht (Tabelle 6).

Tabelle 6: Column GSBL in der spaltenorientierter Datenbank

Row ID	GSBL				
1	GSBLNR	ERSDAT	NEUDAT	STAR	BEARB
	1	20.11.2013 14:27:21	20.11.2013 17:07:29	Einzelinhalts-stoff	kraemer

Ein Vorteil: Die Zeilen müssen nicht die gleiche Anzahl von Spalten haben. Wenn für einen Datensatz zusätzliche Informationen gespeichert werden müssen, können diese hinzugefügt werden, ohne dass alle bestehenden Datensätze um einen NULL-Wert ergänzt werden, wie man es von einer relationalen Datenbank erwartet.

Super Column Families ist eine Erweiterung von Wide-Column-Stores, die die Ablage von vielen Schlüssel-Wert-Paaren ermöglicht. In der Tabelle 7 wird so eine *Super Column Familie* dargestellt. Die Identifikationsnummer der Zeile ist 1, der Name von Super Column ist GSBL, die Schlüsseln GSBLRN, ERSTDAT, NEUDAT, STAR, BEARB beinhaltet.

Tabelle 7: GSBL in der spaltenorientierter Datenbank mit den vielen Schlüssel-Wert-Paaren

Row Id	Colums	
1	GSBL	
	GSBLRN	1
	ERSDAT	20.11.2013 14:27:21
	NEUDAT	20.11.2013 17:07:29
	STAR	Einzelinhalts-stoff
	BEARB	kraemer

Spaltenorientiere Datenbanken sind gut für analytische Informationssysteme geeignet, die eine kleine Anzahl komplexer Abfragen über alle Datensätze und über bestimmte Spalten verwenden. Spaltenorientierte Speicherung hat Vorteile auch bei Datenkompression, Aggregation[31] und beim Caching[32].

Graphen-Datenbanken

Bei diesen Datenbanken werden unstrukturierte Daten in Diagrammen durch Knoten und Kanten zusammen mit ihren Eigenschaften gespeichert. Die Kanten stellen die Verbindungen zwischen den Knoten dar. Diese Datenbanken sind von Interesse für Anwendungen, bei denen die gesuchte Information nicht nur aus einzelnen, zur Suchanfrage passenden Datensätzen besteht, sondern viel mehr in der Art und Weise der Verknüpfungen dieser Datensätze untereinander relevant ist. Typische Anwendungen dafür sind die Darstellung von Nutzerbeziehungen innerhalb von sozialen Netzwerken, Bild- und Textanalysen für Wissenspräsentation, Semantisches Web[33] und Linked Open Data[34] [Edlich, 2011].

[31] Aggregation ist Zusammenfassung von Daten für statistische Auswertungen

[32] Cache bezeichnet in der EDV einen schnellen Puffer-Speicher, der (wiederholte) Zugriffe auf ein langsames Hintergrundmedium oder aufwendige Neuberechnungen zu vermeiden hilft [Cache, 2014]

[33] Im Zentrum der Idee des „semantischen Webs“ steht die Entwicklung von semantischen Technologien, mit deren Hilfe Computer die Inhalte von Musik, Bildern und Videos besser verarbeiten können sollen. Semantisch bedeutet, dass Inhalte nicht bloß eine Bedeutung haben, sondern auch in Beziehung zu anderen Bedeutungen stehen [Merschmann, 2008]

[34]Der Begriff *Linked Open Data* bezieht sich auf eine Ansammlung von *Best practices* für das Veröffentlichen von Informationen und für das *Verbinden* von strukturierten Daten im Netz. *Linked Open Data* ist begrifflich dabei eng mit dem semantischen Web verwoben, womit sich die Frage stellt worin sich diese Ausdrücke überhaupt unterscheiden. Tim Berners-Lee wird zu den Unterschied vom *semantischen Web* und Linked Data wie folgt zitiert: Linked Data is the Semantic Web done right [Linked Open Data, 2014].

Dokumentenorientierte Datenbanken
Diese Art von Datenbanken speichert die Daten in den sogenannten Dokumenten ab. Ein Dokument ist vergleichbar mit dem Datensatz in einer relationalen Datenbank, wobei nicht besetzte Attribute wegfallen. Die Daten in einem Dokument werden in Form von Schlüssel/Wert-Paaren abgelegt. Die Werte sind nicht nur atomare Daten eines einfachen Datentyps, sondern die Listen von Daten, die wiederum geschachtelte Datentypen oder eingebettete Dokumente enthalten können. Eine Normalform ist nicht erforderlich, folglich kann man zusammengehörige Daten als einen Wert ablegen. Einer der Vorteile dokumentenorientierter Datenbanken ist die ganzheitliche Speicherung der zusammengehörigen Daten. Ein anderer Vorteil ist Schemalosigkeit. Unter Schemalosigkeit versteht man, dass jedes einzelne Dokument eine andere Struktur haben kann und ein für die ganze Datenbank gleiches Schema nicht existiert. Ein schemaloses Datenmodell kann sich entsprechend der Datenbankanwendung verändern. Ein Dokument muss nicht von den anderen Dokumenten abhängen. Man kann sich leicht einen chemischen Stoff als ein Dokument vorstellen, da jeder Stoff einen individuellen Satz von Attributen besitzt und die Daten zu einem Stoff zusammengespeichert werden können. Die Merkmale zu einem bestimmten Stoff können hinzugefügt, gelöscht oder erweitert werden, ohne dabei das ganze Datenbankmodell zu verändern. Dokumente entsprechen in ihrem Aufbau sowohl dem natürlichen Objekt als auch dessen Repräsentation in einer objektorientierten Sprache, und damit wird keine besondere Transformation bei der Abbildung in Programmiersprache benötigt. Viele Dokumentendatenbanken verwenden das JSON (JavaScript Object Notation) - Format. MongoDB speichert die Daten in einem etwas abgeänderten JSON-Format, Binary JSON (BSON). BSON wird für die interne Darstellung der Daten verwendet, sodass für die Anwendungsprogramme keine Unterschiede entstehen. Im Unterschied zu JSON unterstützt BSON noch weitere Datentypen die nicht Teil der JSON-Spezifikation sind, wie z.B. einen extra Datums-Datentyp oder einen Datentyp für binäre Daten.

4 MongoDB

4.1 Allgemeine Bewertung

MongoDB gehört zu den weitverbreiteten und ausgereiften dokumentenorientierten Datenbanken. MongoDB bietet solche Funktionen an, die bei den anderen Vertretern der NoSQL-Datenbanken-Familie vermisst werden. Dazu gehören:

- umfangreiche Möglichkeiten zur Formulierung komplexerer Abfragen, was eine Annäherung an SQL darstellt,
- Bereitstellung von Treibern für eine Vielzahl von Programmiersprachen
- starke und aktive Community
- Open-Source-Projekt mit kommerziellem Support und umfangreichen Schulungen
- Nutzung von Memory-Mapped-Files für das Caching der Daten
- das Konzept des Journaling, das von relationalen Datenbanken entlehnt wurde, und das zur Reproduzierung der nicht beendeten Änderungen nach einem Systemabsturz dient
- Indexierung von eingebetteten Feldern

Im Vergleich zu dem anderen bekannten Vertreter von dokumentorientierten Datenbanken, CouchDB, bietet MongoDB mehrere Arten von Indizes, die bessere Dokumentation und die Möglichkeit des partiellen Updates. Da die Datenbank von Anfang an für den Einsatz in Webapplikationen vorgesehen war, lassen sich mit MongoDB Aufgaben wie Dokumentenmanagement, das Verwalten von Anwender- und Sitzungsdaten, Logging[35], Echtzeit-Analysen[36] und generell Aufgaben mit einem hohen Datenaufkommen einfach und bequem lösen. Nicht geeignet ist MongoDB für Systeme mit komplexen Transaktionen und für traditionelle Business-Intelligence-Projekte, da die dafür nötigen Strukturen nicht zur Verfügung stehen [Alvermann, 2011]. MongoDB kann vor allem für noch nicht ausgereifte oder oft ändernde Datenstruktur gut genutzt werden, da Änderungen an der Datenstruktur nicht auf der Datenbankebene geschehen und somit unproblematisch handhabbar sind.

4.2 Installation

Eine aktuelle Version von MongoDB ist unter der URL **http://www.mongodb.org/downloads auf der Webseite des Open-Source-Projekts** zu finden. Es stehen Versionen für verschiedene Plattformen zum Herunterladen bereit. MongoDB benutzt Memory-Mapped-Files, ein virtuelles Filesystem auf dem Dateisystem des Betriebssystems, wodurch auf 32Bit-Rechnern die Datengröße auf 2 GB begrenzt ist. Demzufolge sollten zukünftig nur die 64Bit-Version von MongoDB und ein 64Bit-Betriebssystem (Windows oder Linux) für eine Nutzung des GSBL genutzt werden. Außerdem ist stets die neueste Version von MongoDB zu nehmen, da sich die Performanz über die Versionen elementar verbessert.
Nach dem Laden der aktuellen Version aus dem Web muss die geladene Datei entpackt und ein Ordner für die Speicherung der Daten angelegt werden. Standardmäßig werden die Daten im Root-Verzeichnis unter dem Pfad „/data/db“ gespeichert. Anderenfalls benutzt man beim Starten des Servers die Option „dbpath“ um einen anderen Pfad zu spezifizieren.

4.3 Starten des Server und der Datenbank-Konsole

In den offiziellen Versionen von MongoDB ist ein Programm MongoDB JavaScript-Shell[37] enthalten. Mit Hilfe von JavaScript-Shell kann man den Server administrieren, Datenbankbefehle und Programme ausführen.
Man startet den Server mit dem Befehl **mongod** und JavaScript-Shell mit – **mongo**. Nach der Voreinstellung verwendet ein MongoDB-Server den TCP/IP-Port 27017[38]. Es gibt die Möglichkeit die HTML-basierte Administrationsoberfläche aufrufen, wenn man den Server mit der Option –httpinterface –rest startet und dann unter der URL: http: //localhost:28017 die Informationen zur Datenbank und Clients bezieht. Diese Administrationsoberfläche kann für Testzwecke genutzt werden und nicht für einen

[35] Unter Logging werden die Speicherung der Datenänderungen und Prozessen gemeint.
[36] Echtzeit-Analyse bedeutet die Analyse der Daten in demselben Moment, als sie erzeugt wurden.
[37] JavaScript-Shell ist ein im MongoDB-Paket enthaltenes Standardwerkzeug für die Datenmanipulationen und Datenbankverwaltung, die man in der Datenbankkonsole durchführen kann.
[38] Ein **Port** ist der Teil einer Netzwerk-Adresse, der die Zuordnung von Verbindungen und Datenpaketen zu Server- und Client-Programmen durch Betriebssysteme bewirkt. Ports können Netzwerkprotokolle und entsprechende Netzwerkdienste identifizieren [Port, 2014]. Ein Netzwerkprotokoll ist dient dem Austausch von Daten zwischen Computern bzw. Prozessen, die in einem Rechnernetz miteinander verbunden sind. **TCP/IP - Transmission Control Protocol / Internet Protocol** ist eine Familie von Netzwerkprotokollen.

Praxiseinsatz, da damit nur die Überprüfung einiger Informationen zum Status des Servers und zu Leseoperationen, möglich ist. Die anderen Optionen, mit denen man den MongoDB-Server starten kann, sind unter URL: http://docs.mongodb.org/manual/reference/program/mongod/ dokumentiert.

4.4 Datenmodell und Import der Daten

4.4.1. JSON-Format für die Datenspeicherung

Um die Daten zu importieren wandelt man sie in das JSON-Format um, da MongoDB nach der Voreinstellung die Daten aus diesem Format importiert. Unabhängig von dem eingesetzten Treiber werden die Daten intern in einem binären Format - BSON, ein abgeändertes JSON Format, gespeichert. Zur Ausgabe der Daten wird ebenfalls JSON verwendet. Damit hat man den Vorteil, dass beim Erzeugen des Datenmodells die Struktur der Ausgabe berücksichtigt werden kann. JSON stellt genau wie das XML-Format[39] die strukturierten Daten in einer lesbaren Textform dar. Im Unterschied zu XML erfordert JSON für seine Elemente keine Beginn- und End-Auszeichner (Tags), wodurch weniger Speicherplatz beansprucht wird. JSON basiert auf zwei Strukturen: Schlüssel-Wert-Paar und einer geordnete Liste von Werten. Für eine detaillierte Beschreibung des JSON-Formates ist die URL: http://www.json.org/json-de.html zu empfehlen. Im nächsten Abschnitt wird ein Beispiel für die importierten GSBL-Daten im JSON-Format dargestellt.

4.4.2 GSBL-Datenmodell im JSON-Format

In MongoDB darf jedes Dokument einen eigenen Aufbau aufweisen, was allerdings in Bezug auf Anwendungsprogramme nicht erwünscht wird. Das Datenmodell für die Familie der ähnlichen Objekte muss am besten einheitlich gehalten werden. Die Dokumente sind in die Collections zusammengefasst, ähnlich wie Datensätze zu einer Tabelle in einer relationalen Datenbank. Collections erfordern aber im Unterschied zu einer Tabelle kein Schema und jedes Dokument kann die eigenen Attribute haben. Doch man soll die Dokumente zu einer Collection nicht willkürlich zusammenfügen, sondern unter Berücksichtigung deren logischer Zusammengehörigkeit und des Spezifikums der erforderten Lese- und Schreibeoperationen. Zum Beispiel würde man im GSBL die Collections „Stoffe“ und „Lieferanten“ sowie „Spezies“ und „Zitate“ angelegen. Denn die Daten zu einem oder mehreren chemischen Stoffen müssen oft zusammen geliefert werden, die Lieferanten-, Spezies- oder Zitatendaten kann man mit einer extra Anfrage anfordern, wenn man die braucht.

Zwar bietet MongoDB keine JOINs, aber die Zugehörigkeit von Dokumenten kann man mit Hilfe von Referenzen festlegen. Die einfachste Form zu referenzieren ist ein Teildokument, also ein eingebettetes Dokument, einzubauen. Beispiel: Empfehlungen zu Brand-, Explosionsschutz, Evakuierungen und anderen Verhaltungsregeln sind mit den anderen Stoffdaten zusammenabzulegen, da diese Daten oft mit einer Abfrage geholt werden müssen. Die andere Form des Referenzierens ist ein Verweis in einem Dokument auf ein zusammengehöriges

[39] XML ist eine Auszeichnungssprache (extensible Markup Language), mit welcher Informationen strukturiert werden.

Dokument, ähnlich dem Fremdschlüssel in einer relationalen Datenbank. Der Unterschied liegt darin, dass ein Fremdschlüssel in einer relationalen Datenbank durch ein Datenbankmanagementsystem automatisch aufgelöst wird, in MongoDB hingegen - durch ein Anwendungsprogramm mit Hilfe mindestens einer zusätzlichen Datenbankabfrage. In der Collection „Stoffe“ werden Lieferanten-, Spezies- und Zitaten-Collection referenziert. Ein Stoff hat als Referenzen die eindeutigen Lieferanten-, Spezies- oder Zitatennummer. Es ist hier nicht erforderlich, in der umgekehrten Richtung zu referenzieren: ein Lieferant oder eine Zitat muss seine Stoffe nicht „kennen“.

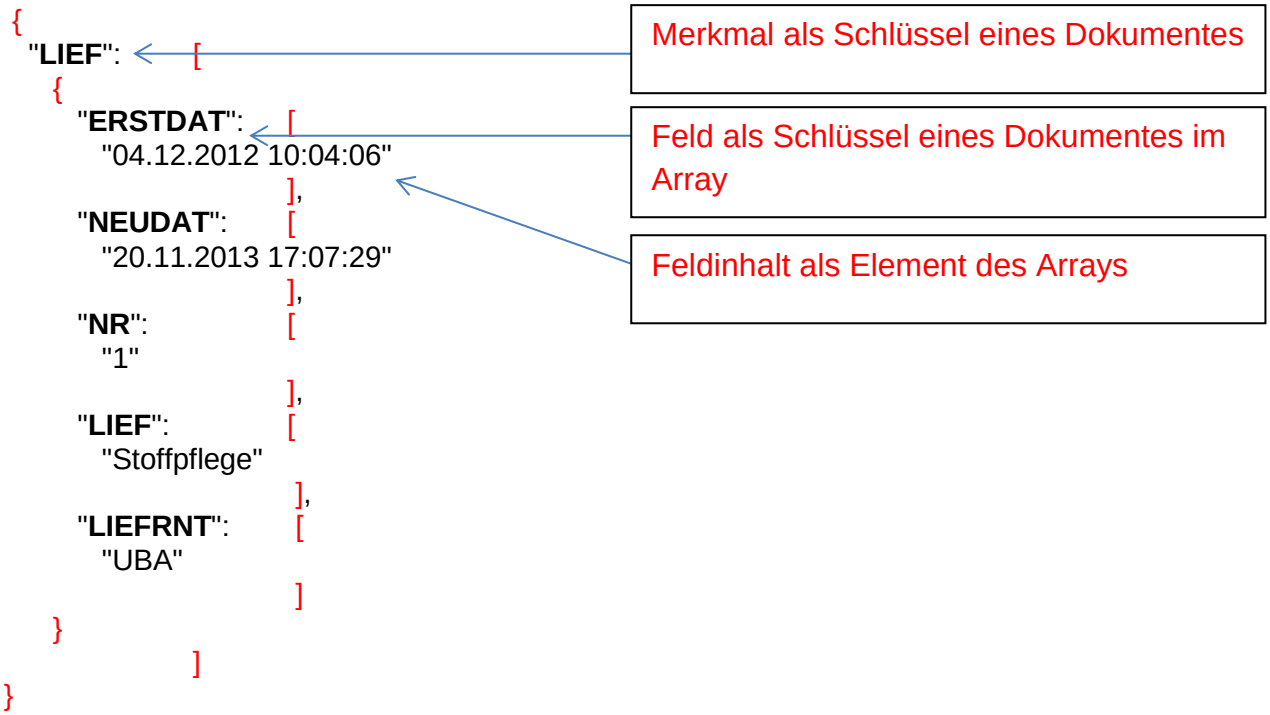
Das Einführen der Referenzen in der Collection „Stoffe“ beruht auf dem Spezifikum der erforderten Lese- und Schreibeoperationen. Mit der ersten Abfrage will man die relevantesten Stoffdaten angezeigt bekommen. Eine zusätzliche Leseoperation für die Lieferanten-, Spezies-, oder Zitatendaten soll keine Probleme bereiten, denn diese Daten werden hauptsächlich von den Fachexperten bearbeitet und nicht mit der ersten Anfrage angefordert. Dabei werden die Stoffdaten nicht unnötig redundant und noch mehr verschachtelt. Beim Update der Stoffdaten kann sich nur die Referenz auf die anderen Collections ändern ohne die Änderungsoperationen auf diesen Collections durchzuführen und Konsistenz der Daten zu verletzen. Umgekehrt muss man beim Ändern der Lieferantendaten keine Updates der Stoffdaten durchführen.

Allgemein kann man behaupten: wenn es zwischen zwei Arten von Objekten eine N:M-Beziehung existiert (ein Stoff hat mehrere Lieferanten, ein Lieferant kann Merkmale zu mehreren Stoffen liefern) und zweite Art sich ändern kann (Lieferant hat seine Adresse geändert), werden Referenzen verwendet. Man sammelt die Objekte einer Art in einer Collection und referenziert in denen die andere Collection.

Auf der Abbildung 5 ist ein Beispiel für das GSBL-Datenbankmodell in JSON-Format abgebildet. Die Daten eines Lieferanten sind analog zu den Stoffdaten aufgebaut, sie sind in die Merkmale und Felder unterteilt und stellen ein Dokument dar. Das Dokument wird immer von den geschweiften Klammern umrahmt.

Das Merkmal „LIFF“ in der Collection „Lieferanten“ ist der Schlüssel zu einem Wert, den ein Array von eingebetteten Dokumenten repräsentiert. Auf der Abbildung 5 enthält der Array nur ein Element, das in die geschweifte Klammer eingeschlossene Dokument. Warum braucht man denn ein Array? Damit beim Einfügen gleichnamiger Merkmale die Werte dieser Merkmale als neue Elemente des Arrays problemlos hinzugefügt werden können. Ein Anwendungsprogramm wird mit keinen zusätzlichen Operationen belastet, wenn man die Werte einheitlich als Arrays ablegt. Auf der Abbildung 5 enthält das eingebettete Dokument fünf Schlüssel (Felder-Namen): ERSDAT, NEUDAT, NR, LIFF und LIEFRNT mit deren Inhalten. Analog zu den Merkmalen sind die Werte der Felder auch Arrays vom Datentyp String.

Abbildung 5: JSON-Modell. Lieferantendaten



4.4.3 Umwandlung des standardisierten Formates in MongoDB-Format

Der GSBL verwendet zum Datenaustausch ein standardisiertes Format - SSF, das die Daten als ein Text darstellt, der dann mit der Dateiendung .ssf gespeichert wird. SSF-Format wurde im Abschnitt 6 beschrieben. Dieses Format bildet deutlich die Struktur der Stoffdaten ab. Die Abbildung 6 veranschaulicht die Stoffdaten in SSF-Format. Mit der Zeichenkette @. wird ein Stoff eingeleitet. Mit @; wird ein Merkmal eingeleitet (auf der Abbildung 6 ist ein Merkmal: GSBL).

Abbildung 6: die Daten in SSF_Format

```
@.
@;GSBL
@:ERSTDAT=10.11.1996 16:25:00
@:NEUDAT=09.01.2014 09:09:49
@:GSBLRN=4934
@:STAR=|Einzelinhaltsstoff
@:SNR=1614
@:SPERRE=1
```

SSF-Format unterscheidet zwischen den Namen der Merkmale und Felder, die einen jeden Stoff charakterisieren, und dessen Inhalten. Da JSON-Format Schlüssel-Wert-Paare enthält und auch einen Textformat darstellt, kann man mit einem Programm die SSF-Datei in eine JSON-Datei so umwandeln, dass die Namen als Schlüssel und Inhalte als Werte abgebildet werden. Man beachtet dabei, dass die Merkmale und Felder eines Stoffes können mehrmals auftreten, was JSON-Format nicht zulässt, da

es gegen die Eindeutigkeit des Schlüssels verstößt. Für die Umwandlung des Subsets (zusammengehörigen Daten) in JSON fehlt in der SSF-Datei der Name des Subsets, der in JSON-Datei für einen Schlüssel erforderlich ist. Da die Subsets immer zu einem Merkmal gehören, wird als Schlüssel der Name des entsprechenden Merkmals übernommen.

Die Abbildung 7 zeigt den Programmablaufplan für die Umwandlung des SSF-Formates in JSON. Rechtecke stehen für Operationen, Rauten – für Verzweigungen (Entscheidungen, wie geht das Programm weiter), Parallelogramme - für die Ein- und Ausgabe. Die Pfeile zeigen Verbindungen zwischen diesen Elementen. Das Programm liest wiederholt die Zeilen einer SSF-Datei, bis das Dateiende erreicht wird. Beim Lesen einer Zeile wird festgestellt, um welche Elemente es sich handelt (Sektions, Merkmale, Felder, Feldinhalte, Subsets) und die benötigten Elemente werden entsprechend der JSON-Syntax umgeformt, zwischengespeichert und in die jeweilige JSON-Datei ausgegeben. Auf die Darstellung der Umformung in JSON-Syntax sowie auf das Ersetzen der ungültigen Zeichen (sowie Whitespace-Zeichen, Datensatztrenner oder Anführungszeichen) wurde aus der Platzgründen auf dem Programmablaufplan verzichtet. Beim Lesen einer Zeile, die ein Merkmal oder ein Feld einleitet, wird überprüft, ob den Namen des Merkmals bzw. des Feldes der beschriebene Stoff schon enthält. In diesem Falle wird der Name nicht noch Mal gespeichert oder in die JSON-Datei ausgegeben, sondern nur der Wert des Merkmals bzw. des Feldes als ein Element des Arrays. Wenn ein Programm ein Subset durchläuft, ist Subsetcounter auf eins gesetzt und der Name des Feldes den Namen des Merkmals übernimmt. Im Unterschied zu einem Feld ohne Subset, ist in diesem Falle ein Array von Dokumenten als Typ des Feldinhaltes definiert. Die Namen der Felder und Feldinhalte werden zuerst zwischengespeichert, die Namen und Werte des Merkmals ohne das Zwischenspeichern in die JSON-Datei ausgegeben.

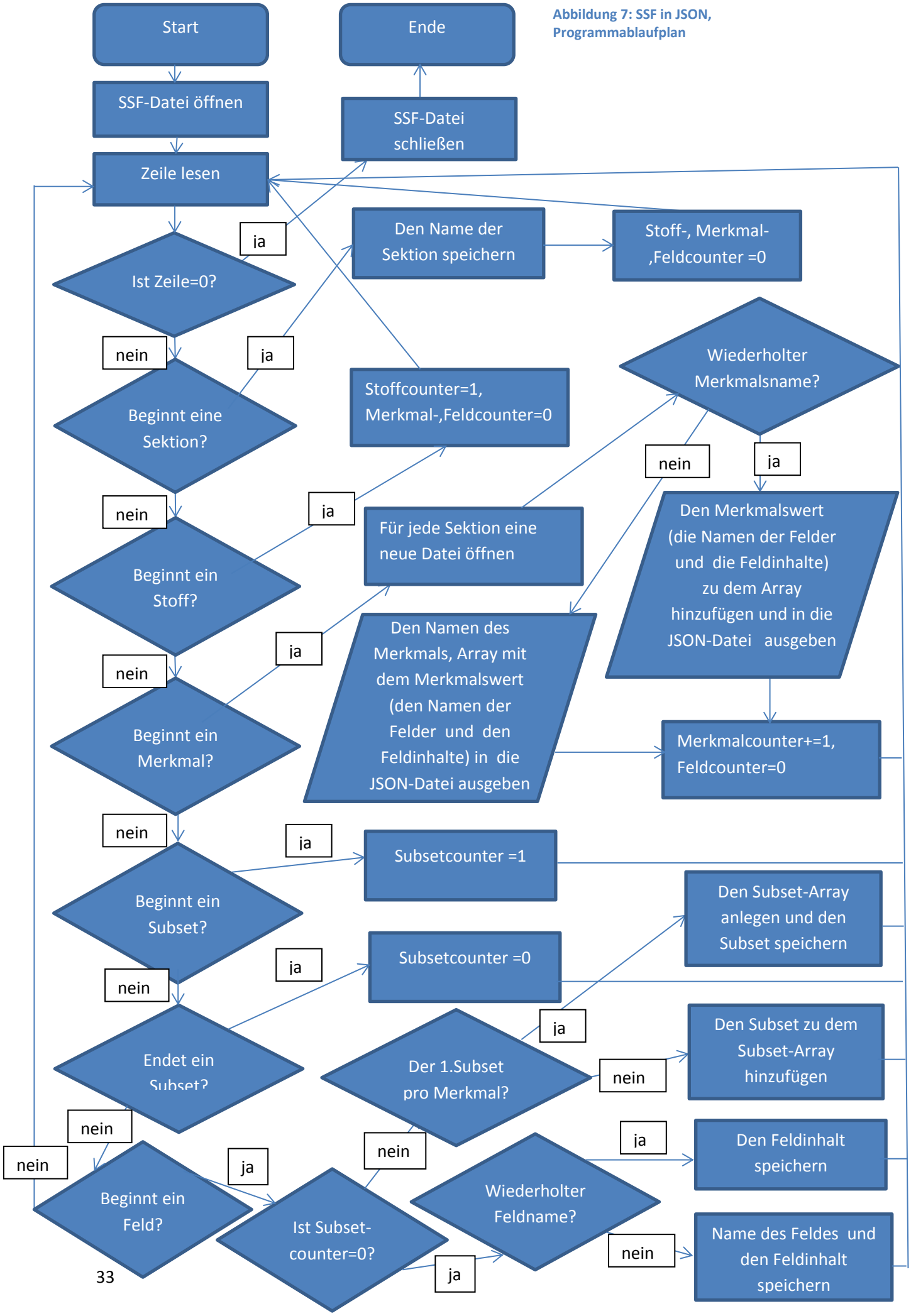


Abbildung 7: SSF in JSON, Programmablaufplan

4.4.4 Import der Stoff-, Lieferanten-, Spezies-, Zitatendaten in Mongodb

Importieren der Daten geschieht mit dem Import Tool *mongoimport*, der sich in den Installationsdateien von MongoDB befindet. Die JSON-Dateien kann man mit dem nachfolgenden Befehl

mongoimport --port 27017 --db gsbl --collection stoffe --type json --file "C:\Users\Irina\Desktop\bsp.json"

aus der Betriebsystem-Kommandozeile importieren. Mittels dieses Befehls wird die Verbindung zu einem lokalen Rechner auf dem Default-Port aufgebaut. Der Datenbankname ist „gsbl“ und der Collection-Name ist „stoffe“. Es wird JSON-File erwartet. Die Datenbank und die Collection müssen nicht vorher angelegt werden. Die werden beim Import von dem Datenbankmanagementsystem erzeugt. Wenn Anmeldedaten erforderlich sind, fügt man in den Befehl die Optionen *--username user und --password pass* nach der Angabe der Collection hinzu. Alle möglichen Optionen für den Import.exe kann man unter URL: http://docs.mongodb.org/manual/reference/program/mongoimport/ anschauen.

Falls man eine Java-Anwendung nutzt, kann man mit Hilfe von JsonParser-Klasse, die eine Schnittstelle für das Lesen JSON-Files bietet, die zu importierenden Daten aus der JSON-Datei auszulesen und danach in die Datenbank speichern. Objekte der JsonParser-Klasse werden mit der Factory-Methode *create JSON-Parser()* der JsonFactory-Klasse erstellt.

Mit dem obengenannten Befehl wurden 316477 Stoffe, 496 Lieferanten, 1428 Spezies, 121971 Zitate (insgesamt 1,6 GB) in MongoDB als JSON-Dokumente importiert.

4.4.5 Import der Tabellendaten in Mongodb

MongoDB erlaubt es auch, die Daten in CSV-Format (bzw.TSV-Format)[40] zu importieren. Das ist sinnvoll für die Daten, die in tabellenartiger Form strukturiert sind, und hilfreich um die Daten zwischen SQL-Datenbanken und NoSQL-Datenbanken auszutauschen. In GSBL sind solche Daten die Nachschlagetabellen, Einheitentabelle und die Tabelle mit dem Datenmodell. Diese Tabellen sind für das Auslesen der Stoffdaten nicht relevant. Sie dienen zur Überprüfung auf die Korrektheit oder Normalisierung der Daten. Beim Export der Stoffdaten können diese Tabellen mitgeliefert werden. Für das Datenmodell ist es eine Pflicht. Die Tabellen können ohne Umformung im selben CSV-Format, in dem sie in die Datenbank importiert wurden, wieder exportiert um danach als Excel-Tabellen an Auftraggeber geliefert werden.
Für den Import der Daten im CSV-Format benutzt man genauso wie beim JSON-Format den mongoimport-Befehl aber mit zusätzlichen Optionen. Falls es um die

[40] Dateiformat **CSV**: auf Englisch bedeutet Comma-separated values, Dateiformat **TSV**: *auf Englisch - Tab-Separated Values*. Diese Formate beschreiben einen bestimmten Aufbau für Textdateien.

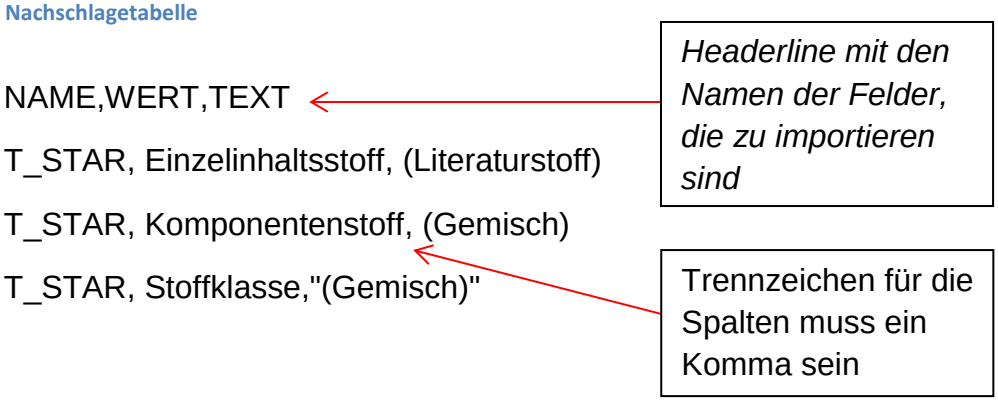
CSV- oder TSV-Dateien geht, darf man die Option *--fields* im Import-Befehl nicht vergessen. Mit dieser Option zeigt man an, welche Felder (in GSBL sind das Merkmale eines Stoffes oder Felder eines Merkmals) importiert werden sollen. Als Alternative kann man in der CSV-Datei die entsprechenden Felder in der ersten Zeile angeben und dann mit der Option *–headerline* darauf hinweisen. Falls die Daten in eine extra Datei eingeschrieben werden, braucht man mit dem Parameter – *FieldFile* diese Datei anzugeben.

Die Daten, die in ihrer Struktur einer Tabelle ähnlich sind, kann man als CSV-Datei speichern um dann in die Datenbank zu importieren. Dabei ist es wichtig, ein Komma für die Trennung der Tabellenspalten festzulegen. Mit dem nachfolgenden Befehl baut man die Verbindung zu einem lokalen Rechner auf dem Default-Port auf, erzeugt die Datenbank „gsbl“ und die Collection „stoffe“, falls sie nicht existieren, gibt das Format der Datei, den Pfad zu der und weist mit der Option *–headerline* auf die erste Zeile mit den Feldnamen hin.

mongoimport --port 27017 --db gsbl --collection stoffe --type csv –headerline –file C:\new.csv

Auf der Abbildung 8 sind Daten im CSV-Format einschließlich einer Headerline mit den Feldnamen dargestellt.

Abbildung 8: CSV-Daten für das Feld "STAR" aus der Nachschlagetabelle



Für die Normalisierungs- und Erfassungsmodule können Fachspezialisten Daten aus den Nachschlagetabellen und Einheitentabelle erfordern. Beispiel: man will rausbekommen, ob der Feldinhalt des Feldes „STAR“ für das zu erfassende Merkmal „GSBL“ zulässig ist. Dafür schaut man, ob sich das Feld in der Excel-Nachschlagetabelle befindet, und falls es der Fall ist, vergleicht den einzutragenden Feldinhalt mit den Feldinhalten für das Feld „T_STAR“ in der Nachschlagetabelle. Sollte der einzutragende Feldinhalt nicht in der Nachschlagetabelle vorhanden sein, wird das Merkmal mit diesem Inhalt nicht in die Datenbank eingetragen. In der Praxis brauchen Fachleute für das Feld „STAR“ nicht in die Tabelle zu schauen, weil es für sie bekannt ist, dass die Stoffart, wofür das Feldname „STAR“ steht, nur drei Werte in GSBL bekommen kann: Einzelinhaltsstoff, Komponentenstoff und Stoffklasse. Diese

drei Werte kann man aus der Nachschlagetabelle auslesen. Ein anderer Wert wird nicht erfasst.

Um anzuschauen, ob sich das Feld „STAR“ in der Nachschlagetabelle befindet, muss man in der Spalte „Tabelle“ des Datenmodells den Eintrag mit dem Feldnamen und der vorangestellten Zeichenkette „T_“ suchen. Dann weiß man, dass die anderen Werte, als die in der Nachschlagetabelle eingetragenen, nicht zulässig sind. Die Anfrage: „Welche Werte und Texte hat das Feld „T_STAR“ in der Nachschlagetabelle?“, wird entsprechend der MongoDB-Anfragesprache formuliert und mit dem nachfolgenden Ergebnis beantwortet. Die Abbildung 9 zeigt dieses Ergebnis.

Abbildung 9: SCV-Datenmodell für die Nachschlagetabelle, Anfrageergebnisse

{ "NAME" : "T_STAR", "WERT" : "Einzelinhaltsstoff", "TEXT" : ""

}

{ "NAME" : "T_STAR", "WERT" : "Komponentenstoff", "TEXT" : "(Literaturstoff)"

}

{ "NAME" : "T_STAR", "WERT" : "Stoffklasse", "TEXT" : "(Gemisch)"

}

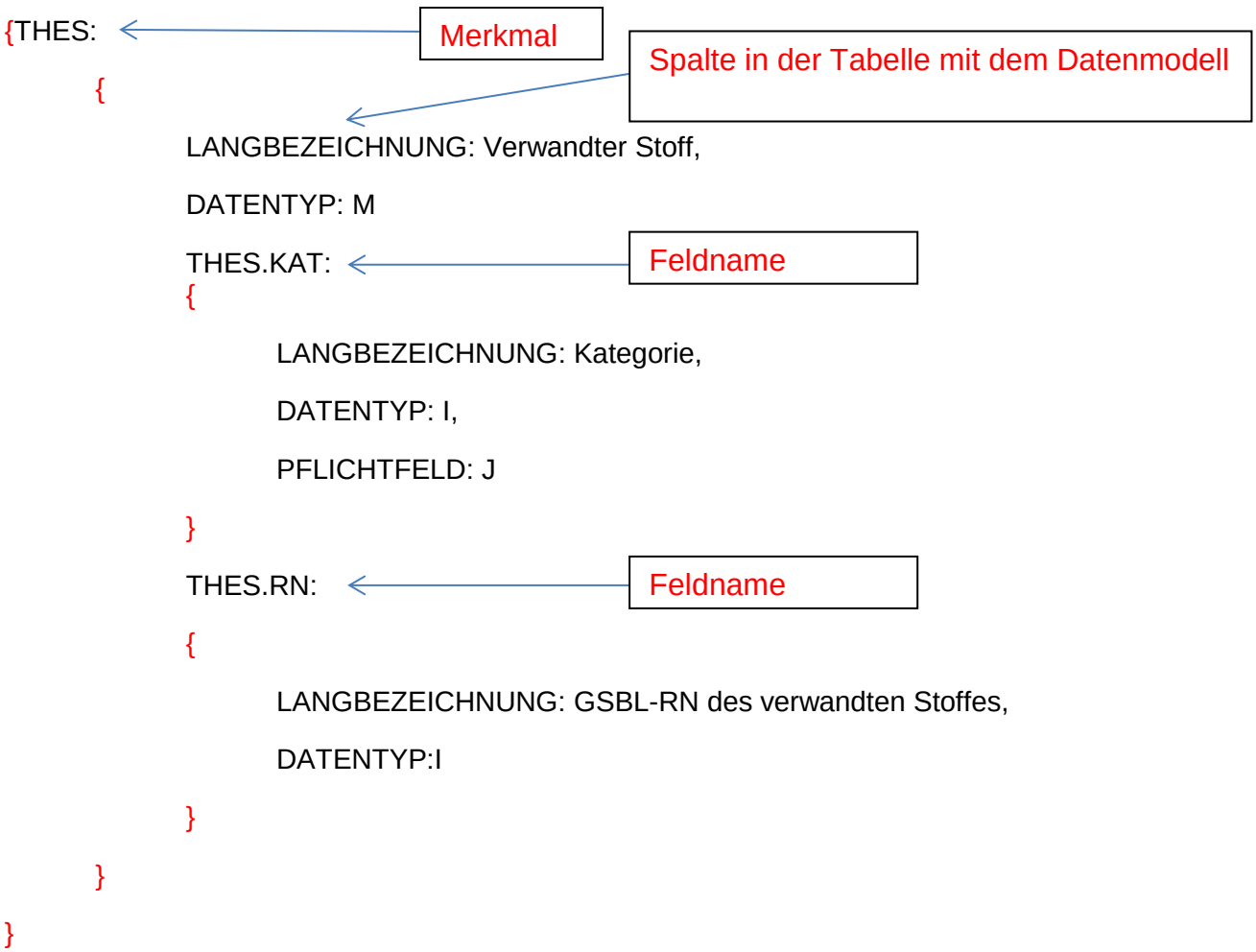
Eine andere Alternative wäre, diese Tabellen aus dem CSV- in das JSON-Format zu verwandeln. Da die Spalten einer Tabelle in CSV-Format mit einem Trennzeichen voneinander isoliert sind, kann man die einzelnen Werte mit einem Programm auslesen und entsprechend einem neuen Format zu speichern. In Java geschieht das mit der Funktion split(): *split=zeile.split(„,“);*

Die Entscheidung, welches Format zu wählen ist, hängt von den zu verarbeitenden Leseoperationen und gewünschten Ausgaben ab. Zum Beispiel, wenn man will, dass derselbe Feldname nur einmal von der Datenbank geliefert wird, speichert man alle Werte der gleichnamigen Felder in einem Array. Die Abbildung 10 veranschaulicht ein solches Array zu dem Feld „T_STAR“. Eine andere Überlegung wäre, das Datenmodell nicht in tabellenartiger Form sondern entsprechend seiner hierarchischen Struktur zu speichern. Alle möglichen Daten zu einem Merkmal (Spalten der Tabelle mit dem Datenmodell) werden in einem Dokument und die Daten zu den Feldern als eingebettete Dokumente zusammenabgelegt (Abbildung 11). Wie in dem Abschnitt „GSBL-Datenmodell in JSON-Format“ erklärt wurde, besteht JSON aus Schlüssel-Wert-Paaren. Die Schlüssel sind in diesem Falle die Spalten der Tabelle und Namen der Felder. Ein Oberbegriff, zu dem ein Merkmal zugeordnet ist, wird auch als Schlüssel eingeführt. Feldinhalte sind eingebettete Dokumente. Der Vorteil dieses Modells ist, dass mit einer Abfrage man alle einem Merkmal zugeordneten Daten bekommt oder ein Merkmal auf das Vorhandensein eines Feldes prüft. Damit man Merkmale und Felder mit den Stoffdaten vergleichen kann, muss man die in verkürzter Form speichern. Die Abbildung 11 zeigt zu dem

Abbildung 10: Datenmodell für die Nachschlagetabelle in JSON-Format. Anfrageergebnisse

```
{ "T_STAR":
        [       {
                "WERT": "Einzelinhaltsstoff", "TEXT": "(Literaturstoff)"
                },
                {
                "WERT": "Komponentenstoff", "TEXT": "(Gemisch)"
                },
                {
                "WERT": "Stoffklasse"
                }
        ]
}
```

Abbildung 11: GSBL-Datenmodell in JSON-Format



4.5 Primär-Schlüssel und Auto-Increment-Funtion

Der Primärschlüssel gekennzeichnet den Datensatz eindeutig. MongoDB erfordert für jedes Dokument in der Collection das Vorhandensein eines _id-Feldes, das einzigartig sein muss. Das _id-Feld vom Typ „ObjectID“ legt MongoDB automatisch an, falls es nicht vom Anwender selbst eingetragen wurde. Über dieses Feld wird der Primärschlüssel erzeugt. Das _id-Feld bietet einen besonderen Vorteil: es beinhaltet die Zeit des Einfügens eines Dokumentes in die Datenbank. Man kann immer diese Zeit mit einer einfachen Abfrage und der *getTimestamp()*-Funktion anzeigen lassen. Zuerst das gewünschte Dokument aus der Datenbank holen und in der Variable *var* speichern. Dann unter Benutzung von Dezimalpunktschreibweise das _id_Feld auslesen und auf dem die *getTimestamp()*-Funktion ausführen. Mit der Print-Funktion das Datum ausgeben lassen.

```
var timestamp = Dokument._id.getTimestamp();

print(timestamp);
```

Ein anderer Vorteil: wenn man Dokumente nach dem _id-Feld vom Typ ObjectId sortiert, entspricht die Reihenfolge der sortierten Dokumente derer Erstellungszeit.

Sollte für die Verarbeitung der Daten einer Collection die Vorteile des _id-Feldes mit dem ObjectId nicht relevant sein, kann man selber ein _id-Feld definieren und in die Dokumente einer Collection einfügen. Für die Collection „Stoffe“ bietet sich das Feld „GSBLRN“, weil jeder registrierter im GSBL Stoff eindeutige GSBL-Registriernummer (GSBLRN) hat. Falls man ein eigenes _id-Feld anlegt, es ist zu empfehlen, Auto-Increment-Funktion für dieses Feld zu definieren. Die Funktion wird beim Einfügen eines neuen Stoffes in die Collection „Stoffe“ den Wert der zuletzt eingetragenen GSBLRN um eins hochzählen. Damit wird es beim Einfügen der Dokumente ausgeschlossen, dass die GSBLRN doppelt vorkommen. Deklaration des Feldes, das inkrementiert werden soll, geschieht in einer extra Collection. In dem nachfolgenden Beispiel wird ein Dokument mit dem _id_Feld und seq-Feld in die Collection „counter“ eingetragen. Das _id-Feld enthält als Wert „GSBLRN“. Das seq-Feld speichert den letzten Wert des zu inkrementierenden Feldes.

```
db.counters.insert ({_id: "GSBLRN", seq: 0})
```

Falls in der Collection „counter“ seq-Feld gleich Null ist und die Collection „Stoffe“ keine Dokumente noch hat, wird beim Einfügen des ersten Dokumentes den Wert des seq-Feldes um eins erhöht und im _id-Feld des Dokumentes eine Eins gespeichert.

Beim nächsten Schritt wird eine Javascript-Funktion definiert, die als Parameter den Namen der Sequenz (sequenceName) bekommt. In unserem Fall ist die sequenceName "GSBLRN". Die Funktion erhöht den Wert der Sequenz um Eins und gibt den geänderten Wert zurück.

```
function getNextSequence(sequenceName)
{var sequenceDocument =db.counters.findAndModify
({query:{_id:sequenceName},update:{$inc:{seq:1}},new:true});
return sequenceDocument.seq;}
```

Beim Einfügen der Dokumente wird die Funktion getNextSequence() aufgerufen und den Wert, den sie zurückgibt, wird dem _id-Feld zugewiesen. Die unten aufgeführten Dokumente haben _id-Feld und „NAME“-Feld.

db.stoffe.insert ({_id: getNextSequence("GSBLRN"), "NAME:"Benzol"})
db.stoffe.insert ({_id: getNextSequence("GSBLRN"), "NAME:" Kypzin"})

Man kann die eingefügten Stoffe abfragen und bekommt das folgende Ergebnis.

{ "_id" : 1, "NAME" : "Benzol" }
{ "_id" : 2, "NAME" : " Kypzin " }

Man sieht, dass im _id-Feld die GSBLRN des Stoffes gespeichert wird und das Dokument den ObjectId nicht enthält. Hier wäre allerdings nachzudenken, ob die GSBLRN zweimal gespeichert werden soll: einmal im _id-Feld und einmal im Merkmal „GSBL“. Eine andere Lösung wäre, die Auto-Increment-Funktion nicht auf dem _id-Feld, sondern auf der „GSBLRN” zu definieren. In diesem Fall wird das _id-Feld mit dem ObjectId von System angelegt, da MongoDB für Dokumente ein _id-Feld unbedingt voraussetzt. Der Vorteil ist hier, dass das Feld „GSBLRN” wie im GSBL-Datenmodell nur im Merkmal „GSBL“ bleibt, aber eindeutig wird, was die gleiche GSBLRN in der Datenbank ausschießt. Um das Feld „GSBLRN“ automatisch zu inkrementieren, fügt man wieder die Daten in die Collection „counter“ ein, aber statt des _id-Feldes wird das „GSBLRN“-Feld eingetragen.

db.counters.insert ({"GSBLRN": "GSBLRN",seq: 0})

In dem nachfolgenden Beispiel wird ein Dokument mit dem Feld „GSBL“ sowie mit den eingebetteten Feldern „GSBLRN“ und „datum“ in die Collection „probe“ eingefügt.

db.probe.insert ({ GSBL: [{"GSBLRN" : getNextSequence ("GSBLRN"), "datum" : "11.11.2015"}]})

Lässt man die Stoffe anzeigen, bekommt man ein Dokument mit zwei eindeutigen Schlüsseln, die automatisch inkrementiert werden, das _id-Feld und die „GSBLRN“.

db.probe.find()

 "_id" : ObjectId("54cccfae224868a94c0fc0e4"), "GSBL" : [{ "GSBLRN" : 1, "datum" : "11.11.2015" }] }

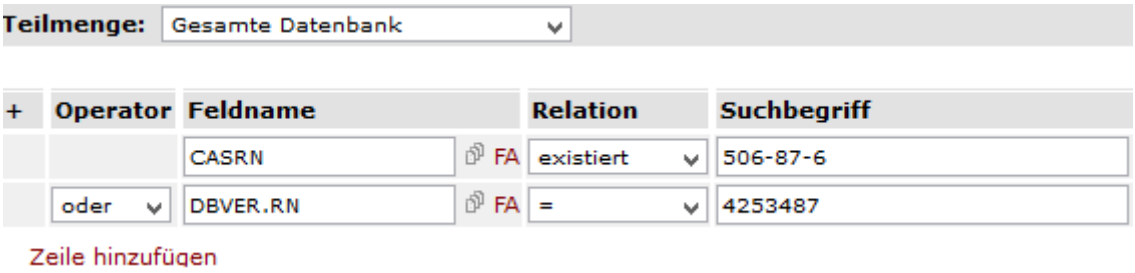
4.6 Datenmanipulationen

4.6.1 Expertensuche

Im Unterschied zu einer einfachen Abfrage, wo die Nutzer nur Namen oder (und) Nummern des gesuchten Stoffes in die Suchmaske eingeben können, benötigen Fachexperten die Möglichkeit beliebige selbst ausgesuchte Felder nach einem oder mehreren eingegebenen Werten zu durchsuchen. Es soll möglich sein, verschiedene Verknüpfungen zwischen den Feldern festzulegen. Beispiel: Suche nach Stoffen, die einen bestimmten Namen haben und kein Feld, das ein Gefahrstoffrecht beinhaltet.

Oder suche nach Stoffen, die „CASRN“ = 506-87-6 und EG-Nummer = 4253487 haben. In der Abbildung 12 ist die aktuelle Suchmaske für die Experten-Suche dargestellt. Man kann beliebig viele Feldnamen selber eingeben oder in einem Feldauswahl-Baum eine Auswahl treffen, die Felder mit den Operatoren „oder“, „und“, “ohne“, “link“ verknüpfen und angeben, ob das Feld in einem Stoff existieren soll oder nicht. In der Abbildung 12 sind zwei Felder „CASRN“ und „DBVER.RN“ mit dem Operator „oder“ verknüpft. Die Gleichheitsbedingungen wie „=“, „>“, „<“ unter dem Begriff „Relation“ stehen für entsprechende Felder ebenfalls zu Verfügung Diese Funktionalität muss MongoDB gewährleisten können. Zur Optimierung der Suchmaske in Hinsicht auf Benutzerfreundlichkeit sollte man zu den Überschriften wie „Operator“ und „Relation“ Hover-Hilfetexte[41] einbauen.

Abbildung 12: Suchmaske für GSBL-Experten-Suche



Um Daten aus MongoDB abfragen zu können, nutzt man die Methoden *find()* oder *findOne()*. Die Methode *find()* wählt die gewünschten Dokumente aus und gibt einen Zeiger auf die gefundene Menge zurück. Mit Hilfe dieser Zeiger kann man über das Ergebnis iterieren. Die Methode *findOne()* gibt nur ein Dokument, das die Suchkriterien erfüllt, zurück. Die Methode wählt das erste Dokument aus der Collection aus, entsprechend der natürlichen Ordnung, nach der die Dokumente auf der Festplatte gespeichert sind. Falls der Operator „oder“ aus dem Aufklappmenü gewählt wurde, muss das Dokument mindestens eins von mehreren Suchkriterien erfüllen, um in die Ergebnismenge zu gelangen.

MongoDB bietet unter anderen Operatoren auch einen Operator „oder“ (Schreibweise: $or) an. Die Suchkriterien werden in MongoDB als JSON-Dokumente formuliert. Operatoren sind spezielle Felder in diesen Dokumenten, die mit einem $-Zeichen beginnen.

Das nachfolgende Beispiel zeigt, wie eine entsprechende Abfrage in der MongoDB-Shell aussieht. Wenn ein Schlüssel in einem JSON-Abfragedokument einen Dezimalpunkt zwischen in den Zeichenketten hat, sowie bei „NAME.NAME“, bedeutet der erste Teil der Zeichenkette einen Schlüssel in dem äußeren Dokument und der andere Teil einen Schlüssel in dem eingebetteten. Mit den GSBL-Begriffen formuliert, steht der erste Teil für das Merkmal „NAME“, der zweite Teil für das Feld „NAME“. Als Ergebnis werden die Dokumente geliefert, in denen „NAME.NAME“ oder "RNAME.RNAME" "Silicontetraacetat" ist oder beide Felder den gleichen Wert haben.

[41] Hover-Hilfetexte sind erklärende Texte, die erscheinen sobald sich der Mauszeiger über einem zu erklärenden Begriff befindet.

db.gsbl.find ({$or:[{"NAME.NAME":"Silicontetraacetat"}, {"RNAME.RNAME": "Silicontetraacetat"}]}

Würde an der Stelle des $or-Operators der $and-Operator stehen, bedeutete dies, dass zwei Bedingungen erfüllt werden müssen: „NAME.NAME“ gleich "Silicontetraacetat" und "RNAME.RNAME" gleich "Silicontetraacetat".

4.6.2 Zugriff auf die Daten mit dem JAVA-Treiber

MongoDB verfügt über eine große Entwicklergemeinschaft und bietet eine Programmierschnittstelle, die viele Programmiersprachen (PHP, RUBI, JAVA) bedient. Der JAVA-Treiber bietet im Paket com.mongo die zentralen Klassen, solche wie DB, DBCollection, BasicDBObject, die auf die Objekte der Datenbank zugreifen. Anhand eines JAVA-Programms wird hier gezeigt, wie die Methode *find()* auf die Collection „Stoffe“ angewendet und ein Cursor für die Ergebnismenge eingesetzt wird. Zuerst baut man die Verbindung auf:

MongoClient mongo= new MongoClient(“localhost”, 27017);

Dann wird die Datenbank „gsbl“ und Collection „stoffe“ gewählt oder angelegt:

DB db= mongo.getDB(“gsbl”);
DBCollection collection=db.getCollection(“stoffe”);

Um Dokumente zu erzeugen, muss man mit der Klasse “BasicDBObject” arbeiten. Die Klasse stellt im Wesentlichen eine geordnete HashMap [42] dar. Um verschiedene Anfragekriterien zu verknüpfen, wird das Objekt der Klasse BasicDBList erzeugt. Mit der Methode *append()* fügt man ein Schlüssel-Wert-Paar zu der Instanz der Klasse “BasicDBObject” hinzu. Die Methode *add()* füllt ein „BasicDBList“-Objekt mit den erzeugten Anfragedokumenten. Anschließend fügt man zu einem BasicDBObject „searchQuery“ das „BasicDBList“-Objekt und den Operator $or hinzu.

BasicDBObject searchQuery=new BasicDBObject();
BasicDBList or=new BasicDBList();

BasicDBObject clause1=new BasicDBObject();
BasicDBObject clause2=new BasicDBObject();

Clausel1.append(“NAME.NAME”, "Silicontetraacetat");
clausel2.append(“RNAME.RNAME”, "Silicontetraacetat");
or.add(clause1);
or.add(clause2);
searchQuery=searchQuery.append(„$or“,or);

DBCursor cursor=collection.find(searchQuery);
Die Methode *hasNext()* holt die Ergebnismenge von der Festplatte und iteriert über sie.

[42] Eine HashMap dient zum Speichern von Key-Value (Schlüssel-Wert) Paaren.

```
while(cursor.hasNext())
{
        BasicDBObject obj=(BasicDBObject) cursor.next();
}
```

Falls man die Ergebnismenge ausgeben will, filtert man Schritt für Schritt die Merkmals-, Feldnamen und deren Werte aus.

```
Set<String> keys=obj.keySet(); // Alle Merkmalsnamen holen

for (String k:keys)
        {
        System.out.println(k);  // einen Merkmalsnamen schreiben
        BasicDBList value=(BasicDBList) obj.get(k); // einen Merkmalswert holen
        Set<String> keys_array=value.keySet(); // Feldnamen holen
                                        .
                                        .
                                        .
          }
```

Die Abfragen und Datenbankzugriffe mit dem JAVA-Treiber können viel Quellcode erzeugen. In JAVA gibt es Frameworks, solche wie Morphia, Spring Data MongoDB und Jongo, die das zu optimieren versuchen. Die Frameworks realisieren die Objekt-Dokument-Mapping, was die Abbildung von JAVA-Objekten auf die Dokumente in MongoDB und umgekehrt bedeutet. Nur Jongo übernimmt die Mongo-Shell-ähnliche Manipulationssprache mit der Verwendung von Strings. Um Jongo mit dem JAVA-Treiber zu verbinden muss man die Jongo-Bibliothek herunterladen und als JAR-Dateien in den Projektpfad einbinden.

Nachfolgend wird ein kleines Programm mit der Verwendung von Jongo vorgestellt. Um die Daten zu manipulieren, definiert man eine Klasse „Friend“, in der wir private Attribute entsprechend den gewünschten Merkmalen definieren. Die get()-Methoden gewährleisten den Zugriff auf die Attribute. Da die Merkmale Arrays von Dokumenten sind, wird für die Attribute GSBL und RNAME der Datentyp ArrayList gewählt.

```
package jongo_fvs;
import org.bson.types.ObjectId;
import java.util.ArrayList;


public class Friend
{
        private ObjectId _id;
        private ArrayList<Gsbl> GSBL; //heißt genauso wie das Merkmal
        private ArrayList<Rname> RNAME; //heißt genauso wie das Merkmal


        public  ObjectId get_id()
```

```
        {
          return _id;
        }

        public  ArrayList<Gsbl>  get_merkmal()

        {
         return GSBL;
        }
        public  ArrayList<Rname>  get_merkmal_2()

        {
         return RNAME;
        }

        public Friend(){}


}
```

Mit den Methoden der Klasse “Rname” greifen wir auf die Felder vom Merkmal „RNAME“ zu.

```
package jongo_fvs;

public class Rname
 {

        Rname(){}

        private Object RNAME;
        private Object ERSTDAT;

        public Object get_rname()
        {
                return RNAME;
        }
        public Object get_erstdatum()
        {
                return ERSTDAT;
        }
}
```

Der nachfolgende Test baut die Verbindung zur Datenbank „gsbl“ und zu der Collection „stoffe“. Dann werden vom Stoff mit „GSBL.GSBLRN“=“4934“ die Felder „GSBL.GSBLRN“ und „RNAME.RNAME“ von der Festplatte geholt und das Feld „RNAME.RNAME“ ausgegeben. Das _id-Feld muss man extra in der Anfrage ausschließen, sonst wird es automatisch mitgeliefert.

```
public class Test
{
        public static void main(String argv[]) throws  UnknownHostException
```

```
        {
        DB gsbl = new MongoClient().getDB("gsbl");

        Jongo jongo = new Jongo(gsbl);

        MongoCollection stoffe = jongo.getCollection("stoffe");

        Iterable<Friend> coll= stoffe.find ("{\"GSBL.GSBLRN\":\"4934\"}").projection
        ("{\"GSBL.GSBLRN\":1,\"RNAME.RNAME\":1}").as(Friend.class);

                while(coll.iterator().hasNext())
                {
                Friend i=coll.iterator().next();

                ArrayList<Rname> m=i.get_merkmal_2();

                        for(int n=0;n<m.size();n++)
                        {

                        System.out.println(m.get(n).get_rname().toString().replace("[","").
                        replace("]",""));
                        }

                }

        }

}
```

4.7 Optimierung durch Indexe

Damit die Abfrage performanter funktioniert, muss man für das zu durchsuchende Feld einen Index[43] anlegen. Indizes in MongoDB werden als Binär-Baum[44] in einer vordefinierten Sortierreihenfolge abgelegt. MongoDB unterstützt verschiedene Arten von Indizes. Für unser Datenmodell passt ein Multikey-Index, da er automatisch beim Anlegen eines Indexes für Felder mit den Werten vom Typ Array gebildet wird. Mit dem nachfolgenden Befehl erstellt man einen Multikey-Index auf dem Feld „RNAME.RNAME“ und mit der Zahl „1“ gibt aufsteigende Sortierung des Indexes an.

```
db.gsbl.ensureIndex({"RNAME.RNAME":1})
```

Die angelegten Indexe lässt man mit dem Befehl getIndexes() anzeigen:

```
db.stoffe.getIndexes()
```

Darauf bekommt man das Ergebnis:

```
 [
```

[43] Index hilft, auf ein Feld in der Datenbank schnell zugreifen, ähnlich wie ein Inhaltsverzeichnis in einem Buch hilft ein Kapitel zu finden

[44] Binärbaum ist ein Graph in einer speziellen Form, wo ein Konten höchstens zwei Kinderknoten besitzt

```
        {
                "v" : 1,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "gsbl.stoffe"
        },
        {
                "v" : 1,
                "key" : {
                        "RNAME.RNAME" : 1
                },
                "name" : "RNAME.RNAME_1",
                "ns" : "gsbl.stoffe"
        }
]
```

Man sieht, dass es zwei Indizes existieren: mit dem Namen „ _id“ und mit dem Namen „RNAME.RNAME“. Die Punktnotation deutet darauf hin, dass es sich um ein Feld in einem eingebetteten Dokument oder in einem Array von Dokumenten handelt.

Der Vorteil von MongoDB liegt in der Möglichkeit die eingebetteten Felder zu indexieren. Der Multikey-Index gewährleistet, dass alle Elemente in Arrays indexiert werden. Das erlaubt nach einem oder mehreren Elementen im Array zu suchen. In einem Array von Dokumenten kann man ein bestimmtes Feld in diesen Dokumenten indexieren. Das wird in dem nachfolgenden Beispiel illustriert. Dort ist ein Dokument mit dem Schlüssel „RNAME“ dargestellt. Das äußere „RNAME“ hat als Wert ein Array von Dokumenten. In den Dokumenten gibt es ein inneres Feld „RNAME“, das unter Nutzung eines Indexes durchgesucht werden kann.

```
{"RNAME":
        [{"BEARB": ["kraemer"],
        "RNAME": ["Neodym(III)sulfat"],
        "TYP": ["systematischer Name"],
        "SPR": ["Deutsch"]
        }]
}
```

Die Anfrage nach „RNAME= Neodym(III)sulfat“ sieht dann so aus:

```
db.gsbl.find({"RNAME.RNAME":" Neodym(III)sulfat "})
```

In GSBL existieren 452 unterschiedliche Merkmale, die wiederum in mehrere Felder unterteilt sind. MongoDB akzeptiert 64 Indizes. Man kann also höchsten entweder 64 Felder oder 64 Merkmale indexieren. Wenn man das ganze Merkmal indexiert, übergibt man der Funktion *ensureIndex()* nur das Merkmal (hier: Merkmal „RNAME).

```
db.gsbl.ensureIndex({"RNAME ":1})
```

Wenn man dann mit der Funktion *insert()* ein neues Dokument in die Datenbank einfügt, wie in dem nachfolgenden Beispiel, muss man in der Anfrage den ganzen Merkmalswert angeben.

db.stoffe.insert({"RNAME":[{"RNAME":["Propensäureethylester"]}]})

db.stoffe.find({"RNAME":{"RNAME":["Propensäureethylester"]}})

Das bedeutet, dass man alle zu dem Merkmal zugeordneten Felder in der Abfrage angeben muss. Sonst wird kein Index benutzt.

Beim Indexieren in MongoDB_2.6 kann ein Problem wegen der Einschränkungen für die Größe der Indizes entstehen. Der Fehler wird so gemeldet:

Btree::insert: key too large to index, failing

Um diese Einschränkung aufzuheben, startet man MongoDB-Server mit dem nachfolgenden Parameter:

mongod --setParameter failIndexKeyTooLong=false

Sollte der ODER-Operator die Felder verbinden, muss man alle zu durchsuchenden Felder indexieren. Sonst wird kein Index benutzt. Beispiel: Suche nach den Stoffen, die einen RNAME.RNAME = Phenol,5-chloro-2-methyl – und einen RNAME.TYP = EINECS-Synonym haben.

Für die Performanz-Untersuchung ruft man gleichzeitig mit einer Anfrage die EXPLAIN()-Funktion auf. Die gibt einige wichtige Metadaten über Ausführung der Query aus. Unter anderem wird die Nutzung der Indizes angezeigt.

Wenn zwei Felder indexiert sind, RNAME.RNAME und RNAME.TYP, und die Abfrage nach RNAME.RNAME= Phenol,5-chloro-2-methyl – oder RNAME.TYP= EINECS-Synonym ausgeführt wird, sieht man aus der Ausgabe der EXPLAIN()-Funktion im Feld „cursor“, dass zwei Indizes benutzt wurden: Btree RNAME.TYP und Btree RNAME.RNAME (Abbildung 13). Außerdem zeigt die Ausgabe, dass es der Multikey-Index benutzt und 18190 Stoffe geliefert (n=18190) wurden. Das Feld „nscannedObjects“ beinhaltet die Anzahl der durchsuchten Dokumente und „nscanned“- die Anzahl der gescannten Index-Einträge. Die Performanz kann deutlich verbessert werden, wenn man für die anzuzeigenden Attribute auch einen Index anlegt. Ausgabe der EXPLAIN-Funktion hat das Feld „indexOnly“, dass in diesem Falle auf TRUE gesetzt wird.

Abbildung 13: Ausgabe der EXPLAIN-Funktion

Bei Abfragen funktioniert im Hintergrund ein Abfrageoptimierer, der sich den schnellsten Plan für die Ausführung einer Anfrage wählt. Auf der Abbildung 13 sieht man das Feld "nscannedObjectsAllPlans“, das die Anzahl gelesener Dokumente für alle Abfragepläne enthält. Das Feld „nscannedAllPlans“ zeigt die Anzahl der gelesenen Index-Einträge für alle Abfragepläne.

In der aktuellen Suchmaske für die Experten-Suche kann man auch den Operator „link“ auswählen. Das bedeutet, dass die zu durchsuchenden Felder zu einem Subset gehören. In diesem Fall, genauso wenn der Operator „und“ gewählt wurde, kann man den Operator $and von MongoDB nutzen. Die Anfrage würde dann so aussehen:

db.gsbl.find({$and:[{"NAME.NAME":"2-HIDROXY BENZOESAEURE"},{
"RNAME.RNAME": "salicylic acid"}

Wenn man die beiden Felder NAME.NAME und RNAME.RNAME indexiert, wählt der Abfrageoptimierer den Index, der für die schnellste Ausführung der Anfrage besser geeignet ist. Folglich wäre es möglich, nicht alle durchzusuchenden Felder zu

indexieren, sondern solche, die in der kleineren Anzahl von Dokumenten vorkommen. Die Suche ist dadurch effizienter, weil es die kleinste Anzahl der Indexeinträge und entsprechend kleinste Anzahl der gefundenen Dokumente auf das Vorhandensein der weiteren Felder durchgesucht werden.

Falls die zu durchsuchende Felder zu einem Dokument, Subdokument oder zu einem Subset gehören, gibt es in MongoDB eine Möglichkeit, die Abfrage performanter mit Hilfe des Operators $elemMatch durchzuführen. Dieser Operator findet Dokumente in einer Collection, die ein Array mit mindestens einem Element, das die Abfragekriterien erfüllt, enthalten.

Beispiel: Suche nach RNAME= Glycine, N,N-bis(carboxymethyl)- und TYP= CAS-Name, wobei der Typ auf genau diesen RNAME bezieht. Das bedeutet RNAME und TYP zu einem Subdokument in Array „RNAME“ gehören. Wenn die Subdokumente noch andere Attribute außer RNAME und TYP enthalten würden, wäre es für die Anfrage irrelevant.

```
{„RNAME“: [
        {
        "RNAME": "Glycine, N,N-bis(carboxymethyl)-",
        "TYP": "CAS-Name"},
        {
        "RNAME": "Trinatriumnitrilotriacetat",
        "TYP": “IUPAC-Name”}
        ]
}
```

Das oben aufgeführte Beispiel sieht unter Verwendung des $elemMatch-Operators in der MongoDB-Shell wie folgt aus:

```
db.stoffe.find ({"RNAME":{$elemMatch:{"TYP":"CAS-Name","RNAME":"Glycine, N,N-bis(carboxymethyl)-"}}}).explain()
```

Es reicht, wenn nur ein Feld aus dem Abfragedokument ein Teil des Indexes ist. Wenn man zwei Indexe hat, auf dem RNAME.RNAME und auf dem RNAME.TYP, wählt der Abfrageoptimierer den für die Abfrage günstigeren. Die Abbildung 14 verdeutlicht, dass für das obengenannte Beispiel der RNAME.RNAME-Index benutzt wurde.

Abbildung 14: Ausgabe der EXPLAIN-Funktion

```
"cursor" : "BtreeCursor RNAME.RNAME_1",
"isMultiKey" : true,
"n" : 1,
"nscannedObjects" : 2,
"nscanned" : 2,
"nscannedObjectsAllPlans" : 5,
"nscannedAllPlans" : 10,
"scanAndOrder" : false,
"indexOnly" : false,
"nYields" : 0,
"nChunkSkips" : 0,
"millis" : 0,
"indexBounds" : {
        "RNAME.RNAME" : [
                [
                        "Glycine, N,N-bis(carboxymethyl)-",
                        "Glycine, N,N-bis(carboxymethyl)-"
                ]
        ]
},
"server" : "Irina:27017",
"filterSet" : false
```

Ab der Version 2.6.3 von MongoDB sollte es laut dem Changelog zudem eine beschleunigende Index-Intersektion geben. Diese kann man selber nicht erzwingen, sie wird vom Abfrageoptimierer automatisch gewählt. Falls die Index-Intersektion benutzt wurde, steht im Feld “Cursor” in der Ausgabe der EXPLAIN-Funktion „Complex Plan“.

Die Suchmaske für Expertensuche (Abbildung 12) bietet die Möglichkeit unter dem Begriff Relation nicht nur die Gleichheitsbedingungen wie „=“, „>“, „<“, sondern auch die Suche nach einer Teilzeichenketten (Wildcard-Suche) zu wählen.

Beispiel: Finde die Stoffe, die mit „Car“ anfangen. MongoDB bietet in diesem Falle die Möglichkeit die regulären Ausdrücke[45] zu benutzen. Die Anfrage wird folgendermaßen formuliert:

db.stoffel.find({"RNAME.RNAME":/^Car/i})

Das Zeichen ^ zeigt an, dass der gesuchte Begriff mit der anschließenden Zeichenkette anfängt, und das Zeichen „i“ am Ende, dass die Groß- und Kleinschreibung nicht beachtet wird. Falls die Zeichenkette am Anfang des Begriffes steht und die Groß- und Kleinschreibung berücksichtigt wird, ist die Anfrage am effizientesten, weil sie unter der Nutzung von Indexen geschieht. Anderenfalls nutzt das System zwar den Index, muss aber trotzdem in vielen Fällen die ganze Menge von Dokumenten durchgehen. Die Abbildung 15 verdeutlicht den Fall, bei dem der gesuchte Stoff den RNAME die Zeichenkette „eth“ enthalten soll. Dabei wird die

[45] Regulärer Ausdruck ist eine Zeichenkette, die Mengen von Zeichenketten mit Hilfe bestimmter syntaktischer Regeln beschreibt.

Groß- und Kleinschreibung nicht beachtet. Im Feld „nscannedObjects“ steht 316473, was bedeutet, dass fast die ganze Menge von Stoffen durchgesucht wurde. Das Feld „nscanned“ steht für die durchgesuchten Indexeinträge. Gefunden wurden 125876 Stoffe. Zum Vergleich verdeutlich die Abbildung 16 den Unterschied im Falle, dass der RNAME mit der Zeichenkette „eth“ anfängt und das Groß- und Kleinschreiben beachtet wird. Hier werden 1967 Stoffe gefunden und 1967 Stoffe durchgesucht. Das Feld „nscanned“ zeigt 3430 Indexeinträge, was den großen Unterschied zu dem vorherigen Bild bedeutet.

Abbildung 15: Ausgabe der EXPLAIN-Funktion für die Wildcard-Suche

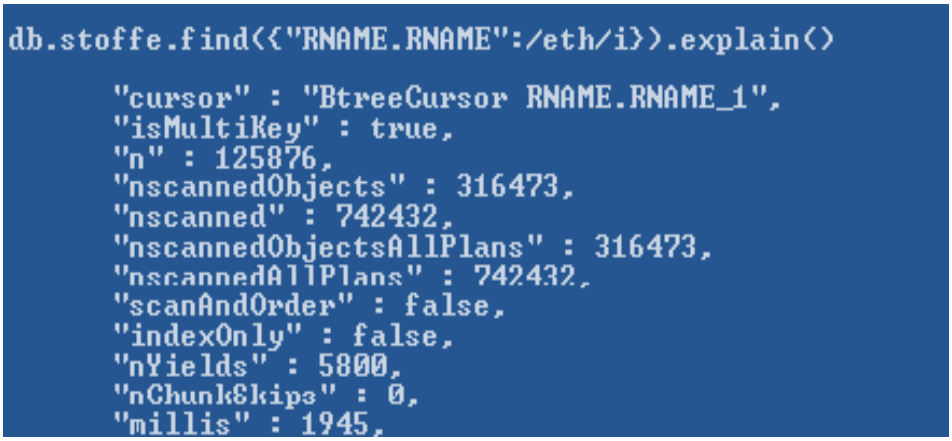
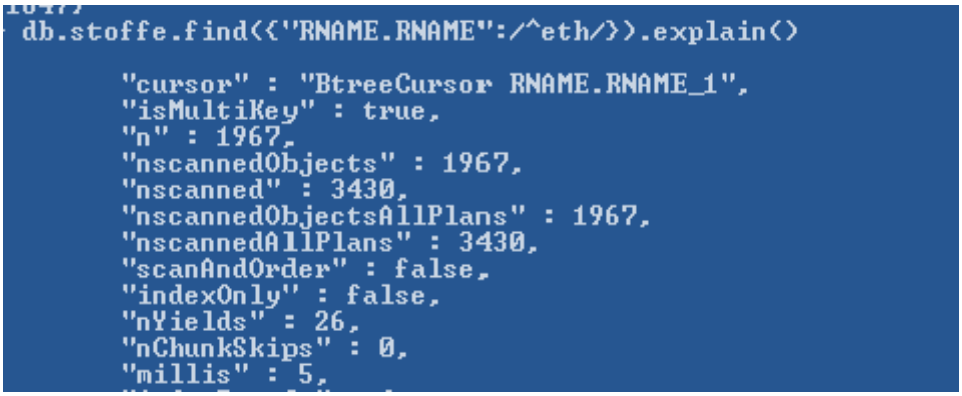


Abbildung 16: Ausgabe der EXPLAIN-Funktion für die Wildcard-Suche

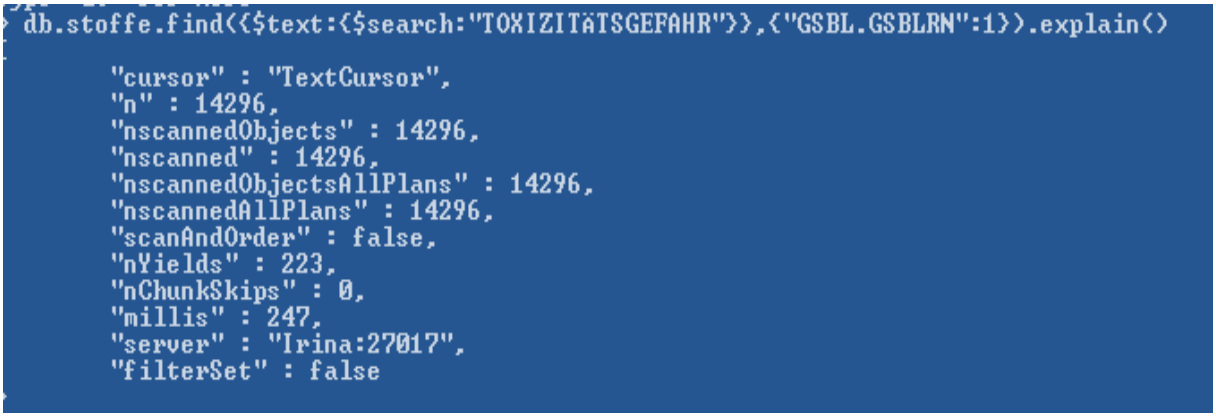


Ab der Version 2.6 ermöglicht MongoDB standardmäßig die Textsuche. Eine Collection kann höchsten einen Text-Index haben. Man kann aber einen Index über mehrere oder sogar alle Felder anlegen. Da ein Textindex viel Speicherplatz in Anspruch nimmt, ist es sinnvoll, nur die umfassenden Textfelder zu indexieren [$text, 2015]. Zum Beispiel die Felder mit den allgemeinen Gesundheitsgefahren (GFRXREA.GFRXREA) oder gefährlichen Reaktionen (GGALL.GGALL). Man legt einen Textindex über diese Felder mit dem folgenden Befehl an:

```
db.stoffe.ensureIndex({"GFRXREA.GFRXREA": "text", "GGALL.GGALL": "text"})
```

Angenommen man will Dokumente mit der Zeichenkette „Toxizitätsgefahr“ finden. Die Abbildung 17 zeigt den Aufruf der Methode find() mit der anschließenden Ausgabe der EXPLAIN-Funktion.

Abbildung 17: Ausgabe der EXPLAIN-Funktion für die Textsuche



4.8 Änderungen an Dokumenten

4.8.1 Änderungsmöglichkeiten

Für die Änderungsoperationen an den Daten braucht man zwei JSON-Dokumente: ein Suchdokument und ein Änderungsdokument. Das dritte Dokument ist optional und beinhaltet verschiedene Änderungsparameter. Ein oder mehrere Dokumente, die auf die Kriterien im Suchdokument passen, werden entsprechend den Anweisungen im Änderungsdokument verändert. Die Änderungsoperation ist in MongoDB auf der Ebene des einzelnen Dokumentes atomar, auch wenn man verschiedene Subdokumente innerhalb dieses Dokumentes ändert. Atomare Operation bedeutet, dass die Änderungen entweder komplett oder gar nicht ausgeführt sind. Die Änderungen über mehrere Dokumente sind nicht atomar, was die Durchführung komplexer Transaktionen, so wie es in den relationalen Datenbanken möglich ist, verhindert. Das wird aber durch einige Änderungsoptionen aufgewogen.
Für die Änderungen braucht man die UPDATE()-Methode. Die kann sowohl die ganzen Dokumenten als auch Teile von Dokumenten ersetzen. Standardmäßig verändert die UPDATE()-Methode ein einziges Dokument. Um mehrere Dokumente zu verändern, die den Anfragekriterien entsprechen, setzt man die Option *multi* auf *true*. Falls kein auf die Anfragekriterien passendes Dokument gefunden wurde, kann man das Dokument einfügen, indem man die Option *upsert* auf *true* setzt. Da MongoDB atomare Operationen nur auf der Ebene des einzelnen Dokumentes und nicht für mehrere Dokumente unterstützt, können die anderen Operationen in den gesamten Änderungsprozess eingreifen. Die Lösung hier bietet der $isolated-Operator, der sichert, dass die Änderungen an Dokumenten nur dann bei den Clients zu sehen sind, wenn alle Dokumente verändert werden. Ein Fehler während der Änderungsoperation macht allerdings die vorherigen Änderungen nicht rückgängig.
Die UPDATE()-Methode, sowie REMOVE()- und INSERT()-Methoden für das Löschen bzw. Einfügen der Daten, geben seit der Version 2.6 standardmäßig das WriteResult-Objekt zurück. Das heißt, der Treiber sendet den Befehl zur Datenbank und wartet auf die Antwort, ob die Operation erfolgreich war. Falls Fehler aufgetreten

sind, wird eine Meldung ausgegeben. Die meisten Änderungen, die im GSBL unternommen werden, sind das Einfügen bzw. Löschen eines neuen Feldes oder Merkmals, Änderungen der Feldinhalte sowie das Registrieren(Einfügen) eines neuen Stoffes. Um Stoffe zu registrieren braucht man die Methode INSERT().

4.8.2 Ändern und Hinzufügen der Merkmale und Felder

MongoDB bietet verschiedene Änderungsoperatoren. Die für den GSBL interessanten sind nachfolgend aufgelistet:
$set und $unset ändert bzw. löscht den Feldinhalt. Wenn das Feld nicht existiert, wird es angelegt. Der nachfolgende Befehl zeigt, wie man in einem Dokument mit der GSBLRN=1358 entweder ein neues Merkmal „RNAME“ anlegt oder den Wert des Merkmals ändert. Das Operator $set zeigt auf das Änderungsdokument mit den neuen Inhalten. Als Ergebnis bekommt man das WriteResult-Objekt, das den Status der Operation zurückgibt. Bei einem erfolgreichen Ergebnis enthält das WriteResult-Objekt die Anzahl der gefundenen, der eingefügten und der modifizierten Dokumente.

```
db.probe.update({"GSBL.GSBLRN":"1358"},{$set:{"RNAME":[{"RNAME":" 2-METHYLHEXANE "},{"RNAME":"HEXANE,2-METHYL-","Qualität":"GSBL"}]}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Die Stoff- bzw. Spezies-, Lieferanten und Zitatendaten wurden für MongoDB so modelliert, dass die Werte der Merkmale und Felder Arrays sind. Bei der Bearbeitung der Stoffdaten wird oft ein neues gleichnamiges Merkmal hinzugefügt (Beispiel: ein neuer RNAME). Da Merkmale in unserem Datenmodell Arrays von Dokumenten sind, wird ein neuer RNAME als ein neues Element des Arrays hinzugefügt. In dem Beispiel unten sieht man, dass das Merkmal „RNAME“, was dem Array „RNAME“ entspricht, nur ein Dokument mit den Feldern: BEARB, RNAME, TYP und SPR besitzt.

```
{"RNAME":
        [ {"BEARB": ["kraemer"],
        "RNAME": ["2-Nitro-1-aminobenzol"],
        "TYP": ["systematischer Name"],
        "SPR": ["Deutsch"]
        } ]
}
```

Um einen oder mehrere Werte zu dem Array hinzufügen, bietet MongoDB folgende Operatoren: $push, $addToSet. Der $addToSet-Operator fügt die Werte zu dem Array hinzu, nur wenn die Werte im Array noch nicht existieren. Das hilft Duplikate zu vermeiden. Der $push-Operator entgegen fügt die Werte zu dem Array hinzu, auch wenn sie schon existieren. In den nachfolgenden Beispielen wird die Anwendung vom $push-Operator verdeutlicht. An der Stelle vom $push-Operator kann $addToSet stehen. Zu dem oben abgebildeten Merkmal „RNAME“ fügt man mit Hilfe des $push-Operators ein neues Dokument mit den Feldern RNAME, TYP und BEARB hinzu. Nach dem Hinzufügen dieses Dokumentes zu dem Array enthält das Merkmal „RNAME“ nicht nur ein, sondern zwei Dokumente.

db.stoffe.update ({"GSBL.GSBLRN":"2"},{$push:{ "RNAME":{"RNAME":" 2-nitroaniline ","TYP": ["EINECS-Name "], BEARB": ["Franke"]}}})

Wenn danach nach diesem Dokument gesucht wird, sieht das Ergebnis wie unten gezeigt aus:


```
{"RNAME":
        [ {"BEARB": ["kraemer"],
        "RNAME": ["2-Nitro-1-aminobenzol"],
        "TYP": ["systematischer Name"],
        "SPR": ["Deutsch"]
        },
        {"RNAME":{"RNAME":" 2-nitroaniline ",
        "TYP": ["EINECS-Name "],
        "BEARB": ["Franke"]
        } ]
}
```


Sollten mehrere Werte an das Array angehängt werden, macht man das mit Hilfe des Modifikators: $each. Im Zusammenhang mit dem Operator $push bzw. $addToSet kann man damit mehrere Werte dem Array hinzufügen. Nachfolgend wird dargestellt, wie das Merkmal RNAME im Stoff mit der GSBLRN=2 zwei neue Dokumente mit den Feldern RNAME, TYP, BEARB bekommt. Diese Updates sind atomar, das heißt, die Operation ist erfolgreich nur wenn alle Werte hinzugefügt werden.

```
db.stoffe.update(

{"GSBL.GSBLRN":"2"},

        {

        $push:{ "RNAME":

                        { $each: [ {"RNAME":" o-(ortho)Nitro-anilin","TYP":
                        ["systematischer Name"], BEARB": ["Franke"] }, {"RNAME":" o-
                        (ortho)Nitro-anilin","TYP": ["systematischer Name"], BEARB":
                        ["kraemer"] } ]

                        }

                }

        }

)
```

Der Modifikator $position bestimmt, an welcher Position im Array die neue Werte hinzuzufügen sind.

4.8.3 Atomarität mit *Update if Current*

MongoDB unterstützt Updates innerhalb eines Arrays nur für das erste Auftreten des auf die Anfrage passenden Elementes. Zum Beispiel wir wollen den TYP auf „IUPAC-Name“ für die Felder „BEARB“ gleich „kraemer“ setzen. Aber Update wird nur für das erste Dokument im Array, wo BEARB „kraemer“ ist, durchgeführt. Wir wollen jedoch auch im zweiten Dokument den Typ ändern. Was ist zu tun, um die Atomarität des ganzen Updates zu sichern?

```
{"RNAME":
        [ {"BEARB": ["kraemer"],
        "RNAME": ["2-Nitro-1-aminobenzol"],
        "TYP": ["systematischer Name"],
        "SPR": ["Deutsch"]
        },
        {"RNAME": ["2-nitroaniline"],
        "TYP": ["EINECS-Name "],
        "BEARB": ["kraemer"]
        } ]
}
```

In diesem Falle wird der Mechanismus *Update if Current* verwendet, der den gleichzeitigen Zugriff auf die Daten steuert. *Update if Current* modifiziert Felder nur dann, wenn sie sich seit der Anfrage nicht geändert haben. Es gibt zwei Varianten, das durchzuführen. Das folgende Beispiel in der Mongo-Shell illustriert das oben dargestellte Update für die erste Variante. Man holt das ganze Dokument (in unserem Falle ist das ein Dokument mit der GSBLRN=2) und speichert dann in der Variable „rname“ das Merkmal „RNAME“.

```
var rname=db.pro.findOne({„GSBL.GSBLRN“:“2“})
var rname=rname.RNAME;
```

Dann weist man der Variable „neu“ ein neues Array zu.

```
var neu=[{"BEARB": ["kraemer"],”RNAME": ["2-Nitro-1-aminobenzol"],"TYP":[ IUPAC-Name "],"SPR": ["Deutsch"]},{"RNAME":["2-nitroaniline"],"TYP":[" IUPAC-Name "],"BEARB": ["kraemer"]}]
```

Im nächsten Schritt wird auf der Collection „probe“ ein Update durchgeführt, falls an dem Merkmal „RNAME“ seit der Anfrage keine Änderungen vorgenommen wurden.

```
db.probe.update({"RNAME":rname},{$set:{"RNAME":neu}})
```

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Mit der Methode find() erhält man das Dokument und sieht, dass die Feldinhalte für das Feld RNAME.TYP entsprechend den Update-Anweisungen geändert wurden.

```
 db.probe.find({„GSBL.GSBLRN“:“2“})
```

{"_id" : ObjectId("54e5e1a8a0dc77e8eb02496b"), "RNAME" : [{ "BEARB" : ["kraemer"], „RNAME": ["2-Nitro-1-aminobenzol"], "TYP" : [“IUPAC-Name"], "SPR" ["Deutsch"] }, {"RNAME" : ["2-nitroaniline"], "TYP" : [" IUPAC-Name "], "BEARB" : ["kraemer"] }] }

Die zweite Variante wäre, eine Versionsnummer in das Dokument einzufügen. Diese Nummer wird in der Anfrage eingeschlossen und bei jeder Änderung hochgezählt.

4.8.4 Änderungen mit dem Positionsoperator $.

Wie kann man die Änderungen an einem Element im Array vornehmen, wenn die genaue Position des Elements nicht bekannt ist. Dafür bietet MongoDB einen Positionsoperator $. Es wäre ein folgendes Szenario vorzustellen: man will ein Feld „NAME“ im Merkmal „NAME“ an der Stelle ändern, wo das Feld den Wert „Profluralin" hat. Der Operator $ findet die Position des zu ändernden Feldes. Wie nachfolgend gezeigt wird, nutzt man für die Änderung wie in den vorigen Beispielen den $set-Operator.

```
db.stoffe.update({"GSBL.GSBLRN":"4670","NAME.NAME":"Profluralin"},
{$set:{"NAME.$.NAME":"NURPROBE"}})
```

Wenn man genau dieses Beispiel mit Java durchführt, braucht man die Klasse BasicDBObject, um das Änderungsdokument zu definieren.

```
BasicDBObject  clause = new BasicDBObject("GSBL.GSBLRN","4670");
BasicDBObject clause1 = clause.append("NAME.NAME","profluralin");
BasicDBObject updateCommand = new BasicDBObject(“$set”, new BasicDBObject
(“NAME.$.NAME”, “NURPROBE”));
DBCollection collection;
collection.update (clause1,updateCommand);
```

4.8.5 Andere Möglichkeiten Operationen atomar zu machen

Wie es schon erwähnt wurde, unterstützt MongoDB die Atomarität einer Operation bezogen auf ein Dokument und keine Transaktionen. Für die gemeinsame Änderung von mehreren Dokumenten bietet die Datenbank außer *Update if Current* noch einen Mechanismus: *two-phase-commit*. Der *two-phase-commit* garantiert, wenn eine der Operationen fehlschlägt, muss die vorherige Operation auf den ehemaligen Zustand gebracht werden. Die Daten bleiben konsistent[46] und der einer Transaktion vorausgehender Zustand wird hergestellt. Im klassischen two-phase-commit-Protokol (2PC) ist der koordinierende Prozess darauf angewiesen, dass von allen Teilnehmern, die als Resource Manager bezeichnet werden, eine Rückmeldung folgt. In der ersten Phase erfolgen die Änderungen temporär und die Daten werden so gesichert, dass kein Verlust möglich ist. In der zweiten Phase schließen die Teilnehmer die Transaktionen ab und bestätigen dies der Koordinierungsstelle. Antwortet ein Resource Manager nach einer Zeitüberschreitung nicht, wird die Transaktion abgebrochen [Edlich, 2011]. Die Transaktion ist dann vollzogen, wenn alle Bestätigungen eingegangen sind.

[46] Konsistenz der Daten bedeutet, dass die Daten auch während der Verarbeitung und Übertragung korrekt bleiben

Die Idee des *two-phase-commit* in MongoDB läuft darauf hinaus, dass man ein zusätzliches Feld - pendingTransactions in das Dokument, das atomar geändert werden soll, einfügt und den Zustand der Transaktion in dem Dokument speichert. Die verschiedenen Zustände: initial, pending, commit oder rollback werden manuell gewechselt.

Auf das Konzept der Transaktionen ist der GSBL nicht angewiesen. Das befürwortet auch die Überführung des GSBL von den relationalen Systemen in eineNoSQL-System. Aber es sind einige Anwendungsfälle in GSBL, in denen die gemeinsame Änderung von Dokumenten sinnvoll wäre: man fügt neue Felder oder Feldinhalte in die Datenbank ein, die noch nicht existierten, und anschließend trägt diese Felder bzw. Feldinhalte in die umfassende Nachschlagetabelle ein. Eine andere Situation wäre: falls bestimmte Feldinhalte in einem Stoff geändert werden, müssen die Referenzen auf diesen Stoff in den anderen Stoffen gelöscht werden. Wenn die Änderungen an einem Dokument die Änderungen an vielen Dokumenten mit sich bringen, kann man in dem ersten Dokument ein zusätzliches Feld mit der aktuellen Zeit (currentTime) einfügen und nur falls alle Dokumente geändert werden, das zusätzliches Feld wieder löschen. Im Falle des Ausfalls des Systems kann man prüfen, ob das currentTime-Feld noch nicht gelöscht wird, und dann die Operation wiederholen.

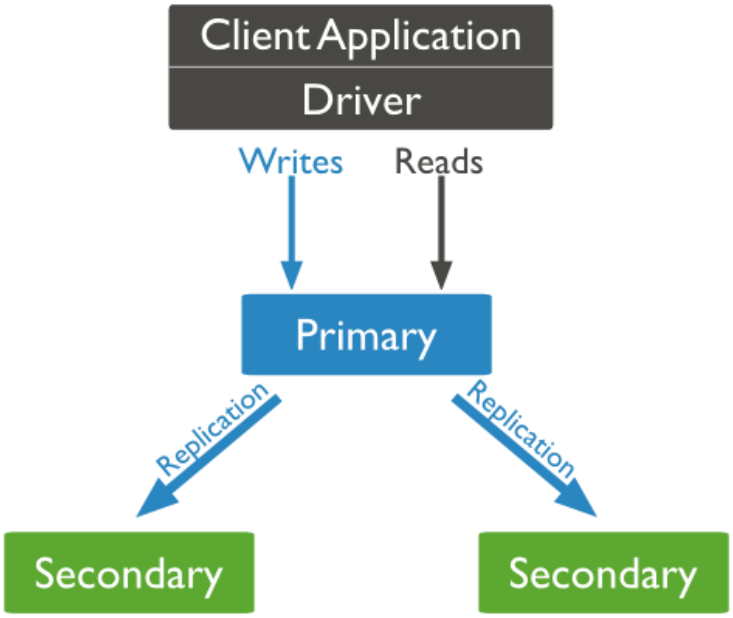
Eine andere sinnvolle Funktion, die MongoDB anbietet, ist *findAndModify*. Sie erlaubt es, atomar ein Dokument abzurufen und zu verändern. Damit vermeidet man konkurrierende Zugriffe, die zwischen dem Finden und Ändern der Daten stattfinden können. Falls die Option *upsert* auf *true* gesetzt wird und ein Dokument, falls es nicht existiert, eingefügt wird, muss das gesuchte Feld im Suchdokument einen Teil des eindeutigen Indexes sein. Anderenfalls können die konkurrierenden Zugriffe für das Entstehen der Duplikate sorgen.

4.9 Sicherheit der Daten

4.9.1 Ausfallsicherheit mit Replica Sets

Die Ausfallsicherheit wird durch sogenannte Replica Sets gewährleistet. Darunter versteht man eine Gruppe von Knoten, die Kopien voneinander sind. Innerhalb dieser Gruppe wird ein primärer Knoten gewählt, der sich aber ansonsten nicht von den anderen unterscheidet. Alle Schreiboperationen werden über diesen primären Knoten abgewickelt und von diesem automatisch an die anderen Knoten der Gruppe weitergereicht. Auf diese Weise werden die Daten redundant abgelegt [Volke]. Bei einem Ausfall der Primärknoten wird einer der Sekundärknoten zur neuen Primärinstanz gewählt. Es wird empfohlen, die ungerade Anzahl von Knoten zu haben, damit im Falle eines Ausfalls des Primärknotens die anderen Knoten eine Mehrheit für die Wahl bilden könnten. Sonst sollte man einen Arbiter installieren. Ein Arbiter ist ein Knoten, der nur an Wahlen des Primärknotens beteiligt ist. Der Primärknoten schreibt alle Schreiboperationen in den sogenannten oplog (operations log), was eine besondere Art von Collection ist, capped collection. Das ist eine Collection mit der begrenzten Anzahl von Einträgen. Nach einer bestimmten Anzahl werden die alten Einträge überschrieben. Sekundäre Knoten kopieren die Schreiboperationen und führen die aus.

Abbildung 18: Replica Sets http://docs.mongodb.org/manual/core/replication-introduction/



Die Datenkonsistenz kann dadurch gewährleistet werden, dass eine Schreibeoperation als erfolgreich gemeldet wird, wenn die Hälfte der Knoten die Änderungen übernommen hat. Man muss hier überlegen, was in diesem Falle wichtiger ist: die Konsistenz, wenn man die Bestätigung von der mehreren Knoten bekommen möchte oder die erhörte Performanz, wenn man keine Bestätigung verlangt. Nach der Voreinstellung in MongoDB werden sowohl schreibende als auch lesende Zugriffe nur auf dem Primärknoten erlaubt. Das stellt sicher, dass die Leseoperation die letzte Version des Dokumentes bekommt. Falls die größere Skalierbarkeit erreichz werden soll, muss man Leseoperationen auch für die sekundären Knoten zulassen.

Als Beispiel wird hier gezeigt, wie ein Replica Set mit drei Knoten zum Laufen gebracht wird. Zuerst werden drei Mongod-Instanzen gestartet. Die Replica Set wird rs1 genannt.

mongod --rest --replSet rs1 --dbpath /data/member1 --port 27001 --config /data /mongodb.config

mongod --rest --replSet rs1 --dbpath /data/member2 --port 27002 --config /data /mongodb.config

mongod --rest --replSet rs1 --dbpath /data/member3 --port 27003 --config /data /mongodb.config

Danach laufen die drei Instanzen zwar, aber sind noch nicht verbunden. Dafür definiert man bestimmte Konfigurationen und initialisiert mit denen Replica Sets.

cfg = { _id : "rs1", members : [{ _id : 0, host : "localhost:27001"},{ _id : 1, host : "localhost:27002"},
{ _id : 2, host : "localhost:27003"}]}

rs.initiate(cfg)

Nach der Initialisierung wird der Primär- und Sekundärknoten gewählt. Mit dem Befehl rs.status() kann man den Status der Replica Sets überprüfen. Auf der Abbildung 18 wird der Auszug aus dem Ergebnis in der Mongo-Shell dargestellt.

Abbildung 19: Replica Sets, Status

Mit Hilfe vom JAVA-Treiber baut man die Verbindung zu allen Knoten der Replica Sets folgendermaßen:

```
MongoClient mongoClient = new MongoClient(Arrays.asList(
                          new ServerAddress("localhost", 27001),
                          new ServerAddress("localhost", 27002),
                          new ServerAddress("localhost", 27003)));
```

Will man die Lesezugriffe auch für die Sekundärknoten zulassen, macht man das mit der Funktion setReadPreference().

```
mongoClient.setReadPreference (ReadPreference.secondaryPreferred())
```

Ausführliche Information zur ReadPreference findet man in der MongoDB-Dokumentation unter **http://docs.mongodb.org/manual/core/read-preference/.**

4.9.3 Datensicherung

MongoDB bietet Verfahren für das Erzeugen von Kopien im binären Format und Wiederherstellen der Daten mit den Werkzeugen mongodump und mongorestore an. Das Erstellen von Sicherheitskopien der Datenbank kann bei dem laufenden MongoDB-Server oder ohne den aktiven Server stattfinden. Das gilt sowohl für alle MongoDB-Datenbanken als auch für eine ausgewählte Datenbank, Collection oder einen Teil einer Collection. Falls der Hostnamen und die Portnummer nicht angegeben wurden, wie in dem nachfolgenden Befehl, wird mit dem Server auf dem Localhost und auf dem Port 27017 verbunden und die Sicherheitskopie mit dem Namen „dump“ in dem aktuellen Verzeichnis erstellt. Im Verzeichnis „gsbl“ erzeugt hier der Befehl eine binäre Datei „stoffe“ mit allen Stoffdaten und eine JSON-Datei mit den Metadaten, z.B. über existierende Indexe. Um die Benutzerdaten und Rollendefinitionen zu sichern, muss man den Zugang zu der Admin-Datenbank haben, da MongoDB dort solche Daten speichert.

mongodump –collection stoffe –db gsbl

Um die Sicherheitskopie auf einem nicht lokalen Server unter Angabe von Zugangsdaten zu erstellen, müssen die Optionen hinzufügt werden, nämlich port, host, username und password. Falls man den Befehl mongodump mehrmals ausführt, sollte man das Unterverzeichnis mit den Daten im Verzeichnis dump (wenn es vorhanden ist) umbenennen, sonst wird er überschrieben.

Im Falle der Replizierung der Daten mit Replica Sets ist es möglich mit der Option --oplog nicht nur alle Daten aus der Datenbank zu kopieren, sondern alle oplog-Einträge vom Anfang bis zu Ende des Sicherungsverfahrens.

Die Wiederherstellung der Daten von mongodump geschieht mit mongorestore. Standardmäßig sucht mongorestore nach den Daten in dem Verzeichnis „dump“. Wenn der mongod-Server nicht aktiv ist, muss der Pfad zur Datenbank angegeben werden.

mongorestore --dbpath /data

Bei dem aktiven Server ist die Angabe des Ports notwendig. Wenn eine Authentifizierung verlangt ist, muss man die Optionen –u(Username) und –p(password) angeben.

mongorestore --port 27017 --db gsbl -u Voronkina -p Irina /data/dump/gsbl

Wurde die Datensicherung mit der Option --oplog gemacht, kann man restore mit der Option –oplogReplay aufrufen, um die Point-in-Time-Kopien[47] der Schreiboperationen wiederherzustellen. Man muss überlegen, ob die Wiederherstellung mit der Option –objcheck: *mongorestore –objcheck* gemacht wird,

[47] Point-in-Time-Kopien sind Kopien der Daten, die zum Zeitpunkt der Momentaufnahme konsistent sind

mit der Ziel die Vollständigkeit der Daten zu überprüfen, oder mit der Option –drop *mongorestore –drop,* damit die Collection vor der Wiederherstellung gelöscht wird.

4.9.4 Andere Sicherheitsfunktionen

MongoDB bietet Authentifizierung auf System- und Datenbankebene und benutzerdefinierte Rollen. Um den Zugriff auf die Daten einzuschränken wird vom Benutzer seinen Namen und ein Passwort verlangt. Der MongoDB-Authentifizierungsmechanismus heißt „Challenge und Response“. Der Mechanismus basiert auf einer Challenge, einer zufälligen Bytefolge, die der Server an den Client mit einer Zufallszahl schickt. Der Client fügt sein Passwort hinzu und schickt einen daraus errechneten Hashwert[48] an den Server zurück. Dieser errechnet ebenfalls einen Hashwert aus denselben Daten und vergleicht es mit dem vom Client. Bei der Übereinstimmung wird die Anfrage ausgeführt.
Den Benutzern werden die Rollen zugeordnet, um Operationen, die sie durchführen dürfen, zu bestimmen. Die Benutzerdaten werden in der System-Collection „system.user“ gespeichert. Um einen neuen Benutzer mit seinen Rollen einzufügen, ruft man die Funktion createUser(), der die Benutzerdaten übergeben werden. Zum Beispiel wird ein neuer Benutzer für die Datenbank „admin“ mit dem Namen „IrinaAdmin“ und dem Passwort „password“ angelegt. Er bekommt einige administrative Privilegien, indem ihm die Rolle als „userAdminAnyDatabase” zugewiesen wird.
db.createUser({user:"IrinaAdmin",pwd:"password",roles:[{role:"userAdminAnyDatabase",db:"admin"}]}) [Enable Authentication, 2015]

Analog kann man andere Benutzer für alle Datenbanken definieren. Welche Privilegien welcher Rolle zugewiesen sind, kann man mit der Methode runCommand(), die einen Benutzernamen und showPrivileg gleich true übergeben bekommt.

db.runCommand({usersInfo:"IrinaAdmin",showPrivileges:true})

Per Voreinstellung verlangt MongoDB nicht nach der Authentifizierung. Diese Voreinstellung und die Tatsache, dass MongoDB per Default auf dem Port 27017 läuft, kann eine Sicherheitslücke für die Angriffe auf gespeicherte Daten bilden. Eine Forschung des Saarbrücker Kompetenzzentrums für IT-Sicherheit (CISPA) – hat bei einigen MongoDB-Datenbanken eine gravierende Sicherheitslücke entdeckt, die durch falsche oder fehlende Konfiguration zustande gekommen ist. Mehr darüber steht unter: http://cispa.saarland/mongodb/ und http://www.pcwelt.de/news/Studenten-finden-Millionen-ungesicherte-Kundendaten-Sicherheitsrisiko-9559037.html#sthash.nRfR8BCb.dpuf

Wenn man die Authentifizierung einstellen will, setzt man in der Konfigurationsdatei den Parameter auth auf true oder startet den Server mit der Option –auth. Danach muss man sich immer authentifizieren, um einem zugewiesene Operationen in der Datenbank durchzuführen. Zuerst meldet man sich als Administrator in der Mongo-

[48] Hashwert ist das Ergebnis einer Abbildung einer Eingabemenge (z.B. Passwort) auf eine Zielmenge(ein Hashwert). Die Rekonstruktion einer Eingabemenge aus dem Hashwert ist nicht möglich

Shell mit dem Befehl (für einen lokalen Server braucht man keine Angabe von der Portnummer):

mongo -u IrinaAdmin -p password --authenticationDatabase admin

Als Administrator kann man den Benutzern die Rollen zuweisen. Um in der Datenbank „gsbl“ die Suchoperationen durchzuführen, meldet man sich als Benutzer mit den entsprechenden Rechten, z.B. mit der “readWrite-Rolle“, an und als authenticationDatabase gibt gsbl an.

Die MongoDB Enterprise Lizenz erlaubt Kerberos-Authentifizierung[49]. Bei der Kerberos-Authentifizierung muss man statt des Benutzernamens das Kerberos-Identifizierungsmerkmal, Kerberos-Principal des Benutzers oder des Services, angeben werden. Kerberos-authentifizierte Verbindung zu einer Datenbank kann man mit dem folgenden Befehl aus der MongoDB-Shell heraus aufrufen:

db.auth({ mechanism: "GSSAPI", user: "Benutzername" })

Außerdem bietet MongoDB auch x509-Authentifizierung[50] mit SSL-Verschlüsselung[51] und Auditing[52]. LDAP-Proxy-Authentifizierung[53] gibt es nicht für Windows, sondern nur für Linux. Man braucht dafür MongoDB Enterprise Lizenz für Linux mit LDAP-Authentifizierung.

4.10 Export

4.10.1 Export im JSON- in CSV-Format

Das Exportieren der Daten geschieht mit dem Export Tool mongoexport, das sich in den Installationsdateien von MongoDB befindet. Mongoexport wird aber nicht für die Sicherung des ganzen Datenbestandes verwendet, weil er nicht zuverlässig die Datentypinformationen erfasst. Standardmäßig exportiert MongoDB die Daten folgendermaßen: für ein JSON-Dokument in der Datenbank ein JSON-Dokument in dem Exportfile. Angenommen, man will alle Stoffe aus der Collection „stoffe“ und der Datenbank „gsbl“ exportieren und nur das Gefahrstoffrecht (das Feld "EG1272_08.KPIK") anzeigen lassen. Die Daten werden in der JSON-Datei domain-bk gespeichert. Dafür nutzt man den folgenden mongoexport – Befehl mit den entspechenden Optionen:
mongoexport -d gsbl -c stoffe -f "EG1272_08.KPIK" -o domain-bk.json

Falls man die Dokumente haben will, in welchen das Feld "EG1272_08.KPIK" existiert, nutzt man dafür den Operator $exists.

[49] Kerberos ist ein verteilter Authentifizierungsdienst (Netzwerkprotokoll) für Computernetze (zum Beispiel für das Internet)
[50] X.509 ist ein Authentifizierungsverfahren mit Hilfe von digitaler Zertifikate auf der Basis von Infrastruktur der öffentlichen Schlüssel
[51] SSL-Protokoll stellt sicher, dass die Daten verschlüsselt durch das Netz gehen und dass sie vollständig und unverändert ans Ziel kommen
[52] Auditing erlaubt es, die Aktionen von mehreren Benutzern und Anwendungen in der Datenbank zu verfolgen
[53] LDAP ist die Abkürzung für das Lightweight Directory Access Protocol, das einen Verzeichnisdienst (Directory) unterstützt um Benutzer zu identifizieren

mongoexport -d gsbl -c stoffe -f "EG1272_08.KPIK" –q
"{'EG1272_08.KPIK':{$exists:true}}" -o domain-ck.json

Doch unerwartet exportiert MongoDB das ganze Feld EG1272_08 und nicht nur EG1272_08.KPIK (unten ist das Subdokument "EG1272_08" dargestellt). Warum geschieht das?

{"EG1272_08" : [{ "EGK" : "Acute Tox. 3 *", "EGH" : "H331" }, { "EGK" : "Acute Tox. 3 *", "EGH" : "H301" }, "KPIK" : ["GHS06", "GHS08"]] }

MongoDB exportiert im JSON-Format die Felder, die spezifiziert wurden, und ein_id-Feld. Falls ein Feld ein Feld in einem Subdokument ist, exportiert MongoDB das ganze Subdokument und nicht nur ein Feld. Im obigen Beispiel wird nicht nur das Feld EG1272_08.KPIK sondern das ganze Subdokument EG1272_08 exportiert.

Oft muss man aber die Anfragekriterien zu den Optionen hinzufügen. Zum Beispiel will man Felder "GSBL", "RNAME", "EG1272_08 " aus den Dokumenten exportieren, wo RNAME.RNAME=“Isoheptan“ ist.

mongoexport -d gsbl -c stoffe -f "GSBL","RNAME","EG1272_08.KPIK" --query
"{'RNAME.RNAME':'Isoheptan'}" -o domain-bk.json
Die Antwort in der Mongo-Shell bei dem erfolgreichen Export
connected to: 127.0.0.1
exported 1 records

Um nach Dokumenten zu suchen, in welchen das Feld "EG1272_08.KPIK" existiert und "EG1272_08.KPIK" ist nicht gleich "GHS07", benötigt man außer dem Operator $exists auch den Operator $ne:

mongoexport -d gsbl -c stoffe -f "EG1272_08.0.KPIK" -q "{'EG1272_08.KP
IK':{$exists:true},'EG1272_08.KPIK':{$ne:'GHS07'}}" -o domain-bk.json

Es ist möglich, die Daten als CSV-Datei zu exportieren. Im Gegensatz zum Export als JSON-Datei kann MongoDB die angegebenen Felder aus den Subdokumenten rausfiltern. Zum Beispiel, wenn man die Stoffe zu bekommen möchte, bei denen das Feld "EG1272_08.0.KPIK" nicht gleich „GHS06“ ist. Also, die Stoffe, die nach der Gefahrenklasse keine akute Toxizität haben.

mongoexport -d gsbl -c stoffe -f "EG1272_08.0.KPIK" -q
"{'EG1272_08.KPIK':{$ne:'GHS07'}}" -csv -o domain-ck.csv

In diesem Falle wird nur das Feld „EG1272_08.KPIK“, wenn es existiert, und nicht das ganze Subdokument in die CSV-Datei ausgegeben. Die CSV-Datei kann man danach in die EXCEL-Tabelle importieren.

Ein Stoff enthält meistens die Referenzen auf andere Stoffe (das Feld: “THES.RN“). Angenommen, man will die referenzierten Stoffe auch mitexportieren. Eine der Möglichkeiten wäre das mit dem Operator $in zu tun. Beispiel: es werden die Stoffe exportiert, bei denen „THES.RN“ mindestens einen der Werte hat, die in einem Array dargestellt sind. In unserem Falle sind es die Nummer: '118727', '118728', '118729'.

```
mongoexport --port 27017 --db gsbl --collection stoffe --query
"{ 'THES.RN':{$in:['118727','118728','118729']}}" -f
GSBL,RNAME,NAME,IUSABF.EIGNER_RN --csv --out
```

4.10.2 Konvertieren von JSON in SSF

Für den Import der Daten in die MongoDB wurde die Konvertierung aus SSF- ins JSON-Format mit einem JAVA-Programm unternommen. Wenn man die Daten als SSF-Datei exportieren will, muss man sie in umgekehrter Richtung umwandeln: JSON - SSF. Innerhalb dieser Arbeit wurde das Konvertierungsprogramm wieder in JAVA geschrieben. Für das Lesen von JSON-Dokumenten wurde die JAR-Datei „java-json“ heruntergeladen und in den JAVA-Klassenpfad eingebunden. Mithilfe der importierten Klassen org.json.JSONObject- und org.json.JSONArray liest man die Schlüssel und die Werte von den JSON-Dokumenten aus. Das äußere JSON-Dokument entspricht einer Zeile in der JSON-Datei. Beim Erzeugen eines neuen JSON-Objektes wird die gelesene Zeile übergeben. Falls die exportierten Dateien das _id-Feld enthalten, kann man es rausnehmen. Mit der Funktion getNames(), der man dieses Objekt übergibt, bekommt man alle Schlüssel aus dem Objekt und speichert sie in einem Array „merkmale“. Wenn ein Objekt einem Stoff entspricht, bedeuten die Schlüssel die Merkmalsnamen.

```
InputStreamReader reader=new InputStreamReader (new FileInputStream
(jsonFile),”UTF-8”);
BufferedReader jsonBuffer = new BufferedReader(reader);
String line=jsonBuffer.readLine();
JSONObject jsonObject=new JSONObject(line);
String [] merkmale= JSONObject.getNames(jsonObject);
```

Danach geht man in einer Schleife ein Array von Merkmalsnamen durch, die nacheinander der Variable str zuweist, und holt die Merkmalswerte dazu.

```
for (int i=0;i<s.length;i++)
{
String str=merkmale[i];
JSONArray a=(JSONArray) jsonObject.get(str);

<Weiterer Quelltext>
}
```

Analog bekommt man die Feldnamen, Feldinhalte aus den Subdokumenten und berücksichtigt dabei die Subsets.

4.10.3 Konvertieren JSON in XML

Für die Konvertierung von JSON- ins XML-Format wurden die JAR-Dateien „java-json“ und „json-simple“ heruntergeladen und in den JAVA-Klassenpfad eingebunden. Das nachfolgende Beispiel zeigt, wie ein Objekt der Klasse „Writer“ beim Erzeugen ein Objekt der Klasse File und die Zeichenkodierung UTF-8 bekommt. Mithilfe der importierten Klasse org.json.JSONObject wird ein neues JSON-Objekt erzeugt, dem eine gelesene Zeile übergeben wird.

```
File fileDir = new File(„C://…//test.xml);
Writer fw = new BufferedWriter (new OutputStreamWriter{new FileOutputStream
(fileDir),”UTF8”));
JSONObject jsonObject=new JSONObject(line);
```

Dann wird mit der Funktion toString() der Klasse org.json.XML, der das jsonObject übergeben wird, ein JSON-Objekt in das XML-Format umgeformt.

```
fw.write(XML.toString(jsonObject));
```

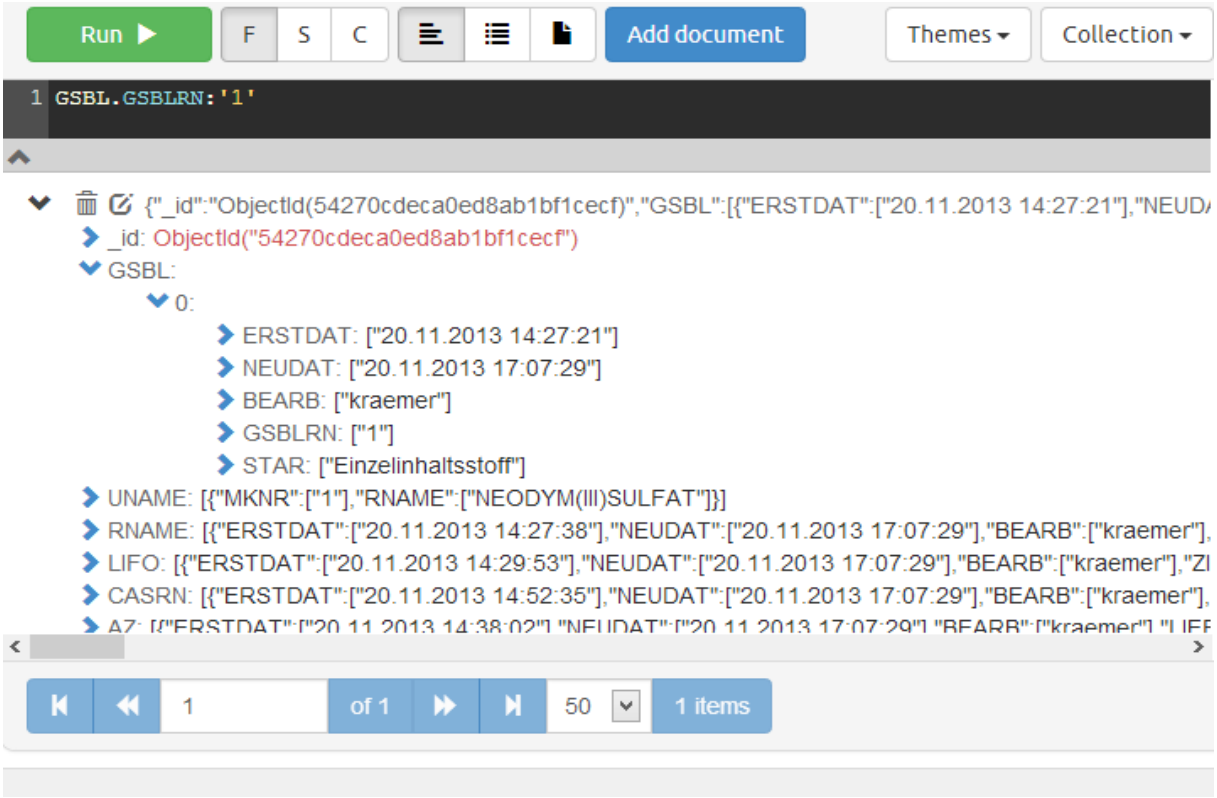
5 Ausblick

MongoDB kann die Erwartungen in Hinsicht auf den GSBL erfüllen. Ein flexibles Datenbankschema eignet sich im Besonderen zur Speicherung der komplexen Objekte in der Stoffdatenbank. Die Verschachtelung der Objekte (Unterteilung in Merkmale und Felder) entspricht am besten der dokumentorientierten Datenstruktur mit eingebetteten Dokumenten, die auch in Listen organisiert werden können. Die Suche und Änderungen an den eingebetteten Dokumenten, auch wenn sie in Listen abgelegt sind, können problemlos vorgenommen werden. Das Hinzufügen, Ändern und Entfernen von Feldern oder Merkmalen in der Datenbank im Gegensatz zu den relationalen Datenbanken ist von Seiten des Datenbanksystems immer möglich. Die atomare Änderung an einem Dokument ohne Datenbanksperre erhöht die Performanz im Vergleich zu relationalen Datenbanken deutlich. Trotz Fehlens des Transaktionskonzeptes und der Datenbanksperre kann man die zusammengesetzten Operationen über mehrere Dokumente sicher und konsistent durchführen. Schreiboperationen werden zuerst in einen Journal eingetragen. Danach werden die Änderungen in die Datenbank geschrieben. Bei dem Absturz der Datenbank werden die in Journal gebliebenen Operationen in der Datenbank ausgeführt. Damit wird die Dauerhaftigkeit der Daten gesichert. Das Journal kann man ausschalten, wenn man noch höhere Geschwindigkeit erreichen will. Um die Datenbanksicherheit zu gewährleisten, muss man die Replica Sets unbedingt aufsetzten. Das Journal hilft auch bei Replica Sets die abgestürzten Knoten wiederherzustellen. Falls die Performanz eines Datenbankserver nicht reicht, muss man den Einsatz von Sharding (Partitionieren der Datenbestände auf verschiedene Server, was horizontale Skalierung genannt) überlegen. Der Umfang des Datenvolumens im GSBL erlaubt es erstmal auf Sharding zu verzichten.
In dieser Arbeit wurde die umfangreiche Manipulationssprache von MongoDB dargestellt, mit der verschiedene Anwendungsfälle für GSBL problemlos gelöst werden können. Es wurde die Geschwindigkeit von Leseoperationen getestet und sehr gute Ergebnisse nachgewiesen. Eine der Stärken von MongoDB ist die Möglichkeit die Array-Werte zu indizieren. Bei dem Anwendungsfall „Expertensuche“ ist es zu empfehlen, den Einsatz der Indizes, einschließlich Textindex, für relevante Felder gut zu überlegen. Ohne Indizes erhöhen sich Laufzeiten. Da die Indizes vollständig in den Arbeitsspeicher passen müssen, ist es besser MongoDB auf einem

dedizierten Server[54] zu betreiben. Um die RAM-Beschränkung 32-Bit-Systeme zu umgehen, muss man MongoDB nur auf einem 64-Bit-System installieren.

Für eine einfache Oberfläche gibt es verschiedene Tools, davon sind MongoVUI und MMS (Mongo Management Studio) zu empfehlen. Die Abbildung 19 zeigt, wie unter Einsatz vom MMS der Stoff mit dem GSBLRN=1 gefunden und als Listenansicht dargestellt wird.

Abbildung 20: MMS, Listenansicht



Welche Besonderheiten von MongoDB muss man noch beachten? Wenn ein nachträgliches Hinzufügen von Merkmalen bzw. Feldern zu einer Vergrößerung des Dokumentes führt, kann das die Verschiebung des Dokumentes in einen freien Speicherbereich auslösen, was wiederum die Laufzeiten beeinträchtigen kann. MongoDB führt zu jeder Collection ein Attribut - Padding, das den zu reservierenden Speicher mit Padding auf der Basis von vorherigen Änderungen beschreibt.

MongoDB benutzt Memory-Mapped-Files, ein virtuelles Filesystem auf dem Dateisystem des Betriebssystems. Die Daten befinden sich in den Dateien, die Größe 2 GB erreichen können. Der Speicherplatz wird aber nicht wieder freigegeben. Um nach dem Löschen der Daten den Speicherplatz wieder freizugeben, wird der Server mit der Option --repair gestartet. Aber dann muss man berücksichtigen, dass alle Dateien kopiert werden müssen, also wird der doppelte Speicherplatz benötigt, und der Server nimmt keine Anfragen entgegen. Die andere Lösung ist in Replica

[54] Ein dedizierter Server ist ein Server, der für einen Netzwerkdienst oder mehrere dauerhaft betriebene Dienste vorgesehen ist, und nicht für alltägliche Aufgaben (Workstation) genutzt wird [Server, 2015].

Sets den Server zu stoppen, Daten zu löschen und den Knoten neu zu synchronisieren.

Um die Datenbanksicherheit zu garantieren und Authentifizierungs- bzw. Autorisierungsmechanismen einzuschalten muss die Datenbank richtig konfiguriert werden.

Mit seiner aktiven Entwicklergemeinschaft, guten Dokumentation, recht einfachen Einsatz- und Administrationsfähigkeiten kann MongoDB eine gute NoSQL-Alternative für GSBL darstellen.

Literaturverzeichnis

1. [GSBL, 2014] GSBL Koordinierungsstelle: Umweltbundesamt. GSBL-Gemeinsamer Stoffdatenpool Bund/Länder. In: http://www.gsbl.de/index.html. Stand: 2015. URL: http://www.gsbl.de/konzept.htm (Letzter Abruf am: 08.03.2015)
2. [GSBL-Handbuch, 2011] GSBL Koordinierungsstelle. Umweltbundesamt (2011): Gemeinsamer zentraler Stoffdatenpool des Bundes und Länder (GSBL). GSBL-Handbuch. Version: 3.022.0028. Potsdam: Ministerium für Umwelt, Gesundheit und Verbraucherschutz Brandenburg
3. [Datenbanken-verstehen, 2014] Datenbanken verstehen. Für Anfänger und Profis. In: URL: http://www.datenbanken-verstehen.de/datenbanken/datenbank-grundlagen/datenbankmodell/ (Letzter Abruf am: 08.03.2015)
4. [FH Köln, 2010] Datenbanken Online Lexikon. FH Köln, Campus Gummersbach. In: http://wikis.gm.fh-koeln.de/wiki_db/. Stand: 05.08.2010. URL: http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/Strukturierter-Datentyp (Letzter Abruf am: 08.03.2015)
5. [Türker, 2005] Can Türker, Gunther Saake: Objektrelationale Datenbanken. Ein Lehrbuch. 1. Auflage; Aufl. (2. November 2005), Heidelberg: dpunkt Verlag.
6. [PostgreSQL9, 2013] PostgreSQL9 SEPTEMBER 2013. Das freie objektrelationale Open Source Datenbanksystem. In: http://www.postgresql.de/index.whtml. URL: http://www.postgresql.de/index.whtml/pg (Letzter Abruf am: 08.03.2015)
7. [Klimt, 2013] Klimt, Wolfgang: Was Sie über SQL wissen sollten. In:. http://www.computerwoche.de/. Stand: 26.06.2013. URL: http://www.computerwoche.de/a/was-sie-ueber-nosql-wissen-sollten,2528753 (Letzter Abruf am: 08.03.2015)
8. [Key-Value, 2011] Eliteinformatiker. In: http://eliteinformatiker.de/. Stand: 01.06.2011. URL: http://eliteinformatiker.de/2011/06/01/nosql-key-value-datenbanken-memcachedb-project-voldemort-redis/ (Letzter Abruf am: 08.03.2015)
9. [Redis, 2014] Key-Value-Datenbanken: Redis. In: http://www.heise.de/. URL: http://www.heise.de/open/artikel/NoSQL-im-Ueberblick-1012483.html?artikelseite=2 (Letzter Abruf am: 08.03.2015)
10. [Edlich, 2011] Edlich, Stefan (2011): NoSQL. Einstieg in die Welt nicht relationaler Web 2.0 Datenbanken. 2. Auflage, München: Karl Hans Verlag
11. [Alvermann, 2011] Alvermann, Markus (01.2011): Alles ist möglich. Einführung in MongoDB. In: http://www.javaspektrum.de. URL: http://www.iks-gmbh.com/assets/downloads/Einfuehrung-in-MongoDB-iks.pdf (Letzter Abruf am: 08.03.2015)

12. [Volke] Volke, Sebastian: MongoDB. Seminararbeit im Modul NoSQL-Datenbanken. URL: http://dbs.uni- leipzig.de/file/seminar_1112_volke_ ausarbeitung.pdf (Letzter Abruf am: 08.03.2015)
13. [[$text, 2015] $text. In: http://docs.mongodb.org/manual/. URL: http://docs.mongodb.org/manual/reference/operator/query/text/ (Letzter Abruf am: 08.03.2015)
14. [Enable Authentication, 2015] Enable Authentication after Creating the User Administration. In: http://docs.mongodb.org/manual/. URL: http://docs.mongodb.org/manual/tutorial/enable-authentication-without-bypass/ (Letzter Abruf am: 08.03.2015)
15. [Wiki, 2015] Datenbank. In: http://de.wikipedia.org/. Stand: 05.03.2015. URL: http://de.wikipedia.org/wiki/Datenbank (Letzter Abruf am: 08.03.2015)
16. [FH Köln, 2012] Datenbanken Online Lexikon. FH Köln, Campus Gummersbach: NoSQL-Datenmodelle und –Systeme. In: http://wikis.gm.fh-koeln.de/wiki_db/. Stand: 10.11.2012. URL: http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/NoSQL (Letzter Abruf am: 08.03.2015)
17. [Thesaurus, 2015] Thesaurus. In: http://de.wikipedia.org/. Stand: 09.02.2015. URL: http://de.wikipedia.org/wiki/Thesaurus (Letzter Abruf am: 08.03.2015)
18. [Gasteiger, 2003] Gasteiger, Johann (18.12.2003): Rosdal. In: http://www2.chemie.uni-erlangen.de/ . Stand: 18.12.2003. URL: http://www2.chemie.uni-erlangen.de/projects/vsc/chemoinformatik/erlangen/struktur_rep/linearnotation/rosdal.html (Letzter Abruf am: 08.03.2015)
19. [Stoffart, 2014] Stoffart. In: http://www.gefahrstoff-info.de/. Stand: 03.2014. URL: http://www.gefahrstoff- info.de/ igs/oberfl/gdl/ hilfe/ merkmale/de_m191.html (Letzter Abruf am: 08.03.2015)
20. [Saake, 2010] Gunter, Saake (2010): Datenbanken. Konzepte und Sprachen. 4. Auflage. Heldelberg: mitp Verlag.
21. [Klasse, 2014] Klasse (Objektorientierung). In: http://de.wikipedia.org/. Stand: 24.02.20125. URL: http://de.wikipedia.org/wiki/Klasse_%28Objektorientierung%29 (Letzter Abruf am: 08.03.2015)
22. [Datentyp, 2014] Datentyp. In: http://de.wikipedia.org/. Stand: 14.02.2015. URL: http://de.wikipedia.org/wiki/Datentyp#Ordinale_Datentypen,2014 (Letzter Abruf am: 08.03.2015)
23. [Merschmann, 2008] Merschmann. Helmut (2008): Semantic Web: Das Internet soll klüger werden. In: http://www.spiegel.de/. Stand: 2008. URL: http://www.spiegel.de/netzwelt/web/semantic-web-das-internet-soll-klueger-werden-a-561831.html (Letzter Abruf am: 08.03.2015)
24. [Linked Open Data, 2014] Warum eigentlich Linked Open Data? In: http://www.linked-open-data.de/. URL: http://www.linked-open-data.de/ (Letzter Abruf am: 08.03.2015)

25. [E-Commerce, 2013] E-Commerce. In: www.dasWirtschaftslexikon.com. Stand: 2013. URL: http://www.daswirtschaftslexikon.com/e/e-commerce/e-commerce.htm (Letzter Abruf am: 08.03.2015)
26. [Server, 2015] Server. In: http://de.wikipedia.org/. Stand: 01.03.2015. URL: http://de.wikipedia.org/wiki/Server (Letzter Abruf am: 08.03.2015)