HAW HAMBURG

MASTERTHESIS
Laura Baensch

Konzeption und Realisierung einer Datenbankmigration zwischen relationalen und dokumentenorientierten Datenbanksystemen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Laura Baensch

Konzeption und Realisierung einer Datenbankmigration zwischen relationalen und dokumentenorientierten Datenbanksystemen

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr. Ulrike Steffens

Eingereicht am: 07. Oktober 2020

Laura Baensch

Thema der Arbeit

Konzeption und Realisierung einer Datenbankmigration zwischen relationalen und dokumentenorientierten Datenbanksystemen

Stichworte

Datenbankmigration, relationale Datenbanken, NoSQL, dokumentenorientierte Datenbanken, PostgreSQL, MongoDB, ArangoDB

Kurzzusammenfassung

Mit der Entstehung und Verbreitung der NoSQL Datenbanken, sind Datenbankkonzepte mit speziellen Kernkompetenzen etabliert worden. Diese bieten für bestimmte Anwendungsfälle eine angepasste Lösung, welche eine relationale Datenbank nicht effizient behandeln kann. Damit einher geht das Problem zwischen zwei andersartigen Datenbankmodellen zu migrieren. Nicht nur die Daten benötigen in einer NoSQL Datenbanken eine andere Struktur, sondern auch die verwendeten Funktionalitäten können variieren. Die Masterthesis analysiert zu dieser Problematik bereits vorhandene Konzepte zur Migration von relationalen zu dokumentenorientierten Datenbanken. Basierend darauf wird ein eigenes Konzept zur Migration und der damit einhergehenden Umwandlung der Struktur, entwickelt. Ebenfalls ausgewertet werden die Funktionalitäten anhand von konkreten Beispieldatenbanken. Diese Analyse und Auswertung bezieht sich nicht nur auf die Migration von relationalen zu dokumentenorientierten Datenbanken, sondern ebenfalls auf die Migration zwischen dokumentenorientierten. Um abschließend das Konzept auszuwerten wird ein Tool, die Rel(Dok)^2 entwickelt, welche mittels Beispieldatensätzen exemplarisch die korrekte Funktion des Tools und damit die des Konzeptes evaluiert.

Laura Baensch

Title of Thesis

Conception and realization of a database migration of relational and document-oriented databases

Keywords

Database migration, relational databases, NoSQL, document-oriented databases, PostgreSQL, MongoDB, ArangoDB

Abstract

With the emergence and distribution of NoSQL databases, database concepts with special core competences have been established. These offer for certain use cases adapted solutions, which a relational database can not handle efficiently. This is accompanied by the problem to migrate between two different database models. Not only the data needs a different structure in a NoSQL database, even the used functionalities can vary. The master thesis analyses to the complex of problems existing concepts for the migration from relational to document-oriented databases. Based on this, an own concept for the migration and the accompanied transformation of the structure will be developed. As well evaluated are the the functionalities on the basis of concrete example databases. This analysis and evaluation not only refers to the migration from relational to document-oriented databases, but also to the migration between document-oriented databases. To finally evaluate the concept, a tool, Rel(Dok)^2, will be developed, which evaluates with example data sets the correct function of the tool and thus of the concept.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

DAG Directed Acyclic Graph.

ER-Modell Entity-Relationship-Modell.

JS JavaScript.

KNN Künstliches Neuronales Netz.

AQL ArangoDB Query Language.

BSON Binary JSON.

DCL Data Control Language.

DDL Data Definition Language.

DML Data Manipulation Language.

DODS Document-Oriented Data Schema.

GUI Graphical User Interface.

ISO International Organization for Standardization.

JSON JavaScript Object Notation.

NoSQL Not only SQL.

SP Schema Paths.

SQL Structured Query Language.

TLS Table-Like-Structures.

UI User Interface.

1 Einführung

Dieses Kapitel leitet die Thematik zunächst mit einer Motivation ein, weshalb die Datenbankmigration als Thema relevant ist (vgl. Kapitel 1.1). In Kapitel 1.2 wird kompakt erläutert, mit welchen Problematiken sich die Masterthesis auseinandersetzt. Entsprechend dazu wird eine Forschungsfrage formuliert. Abgeschlossen wird dieses Kapitel mit dem weiteren Aufbau der Abschlussarbeit (vgl. Kapitel 1.3).

1.1 Motivation

Eine Datenbankmigration zwischen gleichen oder verschiedenen Datenbankmodellen kann aus einer Reihe von Gründen sinnvoll sein. Z.B. können durch den Wechsel von einer kommerziellen oder proprietären Datenbank zu einer Open Source Datenbank Kosten eingespart werden [29]. Oder eine Änderung der Lizenzierung wie es bei den NoSQL Datenbanken MongoDB[1] [41, 40] und Redis[2] [53, 52] vorgenommen wurde, erfordert einen Wechsel der Datenbank, da andernfalls neue Kosten entstehen würden. Ein weiterer möglicher Grund kann sein, dass durch die Verwendung von NoSQL Datenbanken deren Vorteile genutzt werden können, wie z.B. eine höhere Skalierbarkeit. Durch die Vielfalt der NoSQL Datenbankmodelle [43] kann die Verwaltung des zugrundeliegenden Datensatz in einem anderen Modell optimaler erfolgen z.B. durch den Wechsel von einer relationalen zu einer Graphdatenbank.

Damit die Vorteile der neuen Datenbank genutzt werden können, muss eine Migration von der Quell- zur Zieldatenbank erfolgen. Ändert sich dadurch das Datenbankmodell und damit auch die Struktur, wie die Daten hinterlegt sind, kann die Migration zu einem komplexen Prozess werden. Im Normalfall sollte nicht nur eine ausschließliche Migration der Daten erfolgen, sondern sollten ebenfalls alle genutzten Funktionalitäten übertragen werden, falls diese in der neuen Datenbank realisierbar sind.

[1] https://www.mongodb.com/
[2] https://redis.io/

Grundsätzlich ergeben sich vier verschiedene Migrationsszenarien. Zum einen die Migration zwischen relationalen Datenbanken. Hierbei muss keine große Umwandlung der Daten durchgeführt werden, sondern es ist auf die unterschiedlichen, unterstützten Datentypen und Funktionen zu achten. Eine solche Migration ist bereits mittels Tools gut lösbar, beispielsweise kann dem inoffiziellen Wiki von PostgreSQL [50] eine Liste von Tools, die eine Migration vereinfachen, entnommen werden. Das nächste Migrationsszenario erfolgt zwischen zwei NoSQL Datenbanken des gleichen Modells. Hierbei gilt das gleiche wie bei den relationalen Datenbanken, es sind die Unterschiede der Funktionen und Datentypen zu beachten. Alternativ kann zwischen NoSQL Datenbanken mit unterschiedlichen Modellen migriert werden. Da es sich hierbei um schemafreie Datenbanken handelt, können die zugrundeliegenden Daten sehr unterschiedlich ausfallen. Diese Art der Migration gestaltet sich schwierig, da die Daten nicht zwingend ein Schema aufweisen, für die Zieldatenbank jedoch in eine andere Struktur umgewandelt werden müssen. Zwar ist es für konkrete Szenarien möglich, zu migrieren, aber da es sich von Fall zu Fall unterscheiden kann, wird ein allgemeingültiger Migrationsprozess erschwert. Das letzte Migrationsszenario ist zwischen relationalen und NoSQL Datenbanken. Der Prozess lässt sich gut realisieren, wenn von relationalen, bei denen die Daten stark strukturiert vorliegen, zu NoSQL Datenbanken migriert wird. Andersherum müssen die schemafreien Daten in eine starke Struktur komprimiert werden. Dieses Szenario lässt sich für konkrete Anwendungsfälle realisieren, jedoch nicht mit einem generalisierten Verfahren, welches sich auf andere Fälle übertragen lässt. [29]

Auch wenn es für die Migration bei NoSQL Datenbanken nicht die Vielfalt an Tools und Dienstleistern gibt wie bei der Migration zwischen relationalen, bieten vereinzelte Datenbanken eigene Tools an. Diese können die Migration der Daten vereinfachen, in dem sie die Daten aus relationalen Modellen an das datenbankeigene anpassen, wie z.B. OrientDB[3] [48] oder RavenDB[4] [12]. Für Migrationen zwischen verschiedenen Datenbankmodellen im Bereich der Cloud-Services, gibt es Partnerunternehmen, welche als kostenpflichtige Dienstleistung Migrationen ohne Datenverlust und Downtime anbieten, wie z.B. striim[5] [59]. Eine Abhängigkeit entsteht dann dadurch, dass nur eine Auswahl an Datenbanken unterstützt wird [58].

Insgesamt fehlt eine Art Komplettlösung, welche die verschiedenen Aspekte einer Migration berücksichtigt. Neben der ausschließlichen Übertragung der Daten von einem

[3] https://orientdb.com/
[4] https://ravendb.net/
[5] https://www.striim.com/

Datenbanksystem zu einem anderen, ist die Übernahme weiterer Funktionalitäten wünschenswert. D.h. es sollten sämtliche Funktionalitäten wie z.B. Stored Procedures, Indizes, Anfragen etc. mit einbezogen werden. Tools, welche eine Migration in dem Umfang ermöglichen, gibt es nur für relationale Datenbanken, wie z.B. das AWS Schema Conversion Tool[6] [6]. Werden nur einzelne Aspekte bei dem Einsatz eines Migrationstools beachtet, so bleibt der Arbeitsaufwand bei dem Anwender groß. Im Idealfall gäbe es nicht nur für relationale, sondern auch für NoSQL Datenbanken ein solches Tool, welches das Mitwirken des Anwenders auf ein Minimum beschränkt.

1.2 Forschungsfrage

Für die Masterthesis soll ein Verfahren entwickelt werden, welches die Migration von relationalen zu dokumentenorientierten Datenbanken ermöglicht, sowie zwischen verschiedenen dokumentenorientierten Datenbanken. Die Auswahl des Datenbankmodells basiert auf vielfältigen Gründen. Obwohl NoSQL Datenbanken mittlerweile etabliert sind, stehen die relationalen in der Statistik an der Spitze, wie z.B. von der Website DB-Engines [10] oder Statista [57] entnommen werden kann. Unter den 10 populärsten befinden sich auch Vertreter der Schlüssel-Wert- und dokumentenorientierten Datenbanken. Schlüssel-Wert-Datenbanken wurden nicht ausgewählt, da diese nur eine triviale Struktur aufweisen und der Migrationsprozess sehr direkt durchgeführt werden kann (vgl. Kapitel 2.2.1 und 2.5.1). Mehr Gestaltungsmöglichkeiten bieten die dokumentenorientierten Datenbanken. Ein weiterer Grund hängt mit der Literatur zusammen. Diese hat sich bei den Migrationsverfahren von relationalen zu NoSQL Datenbanken auf die dokumentenorientierten fokussiert. Zwar werden für alle anderen Modelle ebenfalls Ansätze geboten, jedoch nicht in der Anzahl, wie es bei den dokumentenorientierten der Fall ist. Außerdem kann mit dem Wechsel von relationalen zu dokumentenorientierten Datenbanken eine Performanceverbesserung erzielt werden. Die Literatur stellt dazu zahlreiche Experimente vor, bei denen dies nachgewiesen werden konnte. Exemplarisch wird zumeist die MongoDB als Vertreter für die dokumentenorientierte Datenbank ausgesucht. Bei den relationalen Datenbanken wurden unterschiedliche Repräsentanten gewählt, wie z.B. Microsoft SQL Server[7] [47], PostgreSQL[8] [24] oder MySQL[9] [20]. [29]

[6] https://aws.amazon.com/de/dms/schema-conversion-tool/
[7] https://www.microsoft.com/de-de/sql-server/sql-server-2019
[8] https://www.postgresql.org/
[9] https://www.mysql.com/de/

Wie bereits in Kapitel 1.1 angemerkt, sind Tools zu einzelnen Themen der Migration oder kostenpflichtige Dienstleister vorhanden. Dabei handelt es sich meist um eine Migration der Daten ohne Berücksichtigung der Funktionalitäten. In der Masterthesis soll ein Konzept erarbeitet werden, welches sich aus zwei Teilen zusammensetzt. Zum einen soll eine Migration der Daten und Funktionalitäten von relationalen zu dokumentenorientierten Datenbanken möglich sein, zum anderen zwischen dokumentenorientierten Datenbanken. Abhängig vom Datenbankmodell der Quelle bzw. dem Ziel, ist eine Anpassung der Struktur, in der die Daten hinterlegt sind, notwendig. Des Weiteren sollen für den Migrationsprozess verschiedene Optimierungsziele zur Auswahl stehen, wie z.B. Platz- oder Zeitoptimierung. Abschließend sollte evaluiert werden, ob Abfragen auf die Quell- und Zieldatenbank die gleichen Ergebnisse liefern und der Datensatz durch den Prozess nicht in seiner Funktion verändert wurde.

Daraus ergibt sich für die Masterthesis folgende Forschungsfrage:
Wie kann eine Datenbankmigration von relational zu dokumentenorientiert, bzw. zwischen dokumentenorientierten Datenbanken erfolgen, wenn nicht nur die Daten, sondern auch die Funktionalitäten berücksichtigt werden?

1.3 Aufbau der Masterthesis

Die Abschlussarbeit gliedert sich im Weiteren wie folgt. Das Kapitel 2 setzt sich mit den Grundlagen, welche für die Thematik relevant sind, auseinander. Darunter fallen die relationalen (vgl. Kapitel 2.1) und NoSQL Datenbanken (vgl. Kapitel 2.2). Durch die verschiedenen Datenbankmodelle werden abweichende Konsistenzmodelle unterstützt (vgl. Kapitel 2.3). Abschließend befassen sich die Grundlagen mit der Datenbankmigration (vgl. Kapitel 2.4) und den verwandten wissenschaftlichen Arbeiten (vgl. Kapitel 2.5), welche die Migration von relationalen zu NoSQL Datenbanken behandeln.

In Kapitel 3 wird ein eigener Ansatz für die Migration von relationalen zu dokumentenorientierten Datenbanken entwickelt. Dazu wird in Kapitel 3.1 systematisch die Konzeptentwicklung begonnen, um basierend darauf, das eigene Konzept vorzustellen (vgl. Kapitel 3.2). Welche Funktionalitäten bei der Migration berücksichtigt werden können, wird in Kapitel 3.3 diskutiert. Abgeschlossen wird die Thematik mit einer knappen Zusammenfassung über die Komponenten, welche migriert werden (vgl. Kapitel 3.4).

Der Migrationsprozess zwischen dokumentenorientierten Datenbanken wird in Kapitel 4 erläutert. Neben den Daten (vgl. Kapitel 4.1) werden die Funktionalitäten ausgewertet (vgl. Kapitel 4.2). In Kapitel 4.3 wird kompakt zusammengefasst, welche Elemente bei der Migration umgesetzt werden können.

Kapitel 5 stellt das Migrationstool Rel(Dok)^2 vor, welches zur Verifizierung der in den vorherigen Kapiteln entwickelten Ansätze dient. Dazu werden die Anforderungen an das Tool definiert (vgl. Kapitel 5.1), der Entwurf vorgestellt (vgl. Kapitel 5.2) und auf die Implementierung eingegangen (vgl. Kapitel 5.3). Ebenfalls werden die Beispieldatensätze vorgestellt (vgl. Kapitel 5.4) und in Kapitel 5.5 die Ergebnisse ausgewertet.

Abgeschlossen wird die Masterthesis in Kapitel 6 mit dem Fazit und einem Ausblick.

2 Grundlagen

Dieses Kapitel befasst sich mit den Grundlagen zu relationalen und NoSQL Datenbanken, sowie mit einer Diskussion von relevanten wissenschaftlichen Arbeiten. Die Grundlagen gliedern sich in eine Einführung zu den relationalen (vgl. Kapitel 2.1) und zu den NoSQL Datenbanken (vgl. Kapitel 2.2). Des Weiteren wird auf die verschiedenen Konsistenzmodelle (vgl. Kapitel 2.3) eingegangen und der grundlegende Prozess einer Datenbankmigration vorgestellt (vgl. Kapitel 2.4). Abgeschlossen wird das Kapitel mit einer ausführlichen Diskussion zu den relevanten wissenschaftlichen Arbeiten im Bereich der Migration von relationalen zu NoSQL Datenbanken (vgl. Kapitel 2.5).

2.1 Relationale Datenbanken

Eine relationale Datenbank [36] verwaltet Informationen in Tabellen, welche aus Spalten und Zeilen bestehen. Spalten weisen einen Merkmals- oder Attributsnamen auf. Jede Zeile, welche auch als Tupel bezeichnet wird, bildet einen Datensatz ab. Ein Eintrag in einer Tabelle stellt entsprechend des Wertebereichs der Spalte einen bestimmten Datenwert dar. Jede Tabelle weist einen Identifikations- bzw. Primärschlüssel auf, welcher aus einem Attribut oder einer Kombination von Attributen besteht und dadurch eine eindeutige Identifikation erlaubt. Dieser Schlüssel muss minimal sein, d.h. es werden nicht mehr Attribute verwendet, als für die Einzigartigkeit nötig. Ebenso müssen Tabellenname, sowie jeder Attributsname der Tabelle eindeutig sein. Beziehungen können in der relationalen Datenbank mit Fremdschlüsseln abgebildet werden, welche als Attribut einer Tabelle auf eine andere verweisen. Innerhalb einer Tabelle liegt keine Ordnung der Spalten und Zeilen vor, sie ist beliebig und weist keine tiefere Bedeutung vor. [36]

Beispielhaft kann die Darstellung der Tabelle „Vorlesung“ aus der Tabelle 2.1 entnommen werden. „vorlesung_id“ ist der Primärschlüssel, über den jeder Datensatz eindeutig identifiziert werden kann. „professor_id“ ist ein Fremdschlüssel, welcher auf einen Datensatz

der Tabelle „Professor“ verweist. Exemplarisch ist „Grundseminar“ ein Datenwert, welcher der Spalte „titel“ angehört und der Zeile bzw. dem Datensatz mit der eindeutigen „vorlesung_id“ „msc1“ angehört.

Tabelle 2.1: Darstellung der Tabelle Vorlesung aus dem relationalen Modell

vorlesung_id	professor_id	titel	hörsaal
msc1	101	Grundseminar	H1
msc2	102	TTI	H2

Relationale Datenbanken können durch eine Reihe von Eigenschaften definiert werden, welche Meier und Kaufmann [36] aufstellen. Diese Art von Datenbanken gehört dem relationalen Modell an und erfasst die Daten in Tabellen, welche über Beziehungen mit anderen verknüpft werden können, um Abhängigkeiten darzustellen. Das Schema enthält Definitionen zu Identifikationsschlüsseln, Regeln für die Integrität, sowie zu Tabellen und deren Attributen. Über eine Sprache (vgl. Kapitel 2.1.2) können die Daten manipuliert, ausgewählt oder definiert werden. Die Architektur gewährleistet eine Datenunabhängigkeit, dadurch können Änderungen an der Datenbank erfolgen, ohne dass Anwendungsprogramme entsprechend angepasst werden müssen. Bei relationalen Datenbanken können gleichzeitig Abfragen oder Bearbeitungen an dem System durch mehrere Benutzer erfolgen, d.h. es wird ein Mehrbenutzerbetrieb unterstützt. Datenintegrität wird durch Hilfsmittel der Datenbank erreicht und es sind Mechanismen für Datenschutz und -sicherheit vorhanden. [36]

Im Folgenden wird auf das ER-Modell eingegangen (vgl. Kapitel 2.1.1), welches eine graphische Darstellung der relationalen Datenbank bietet und die Sprache SQL mit ihren Elementen erläutert (vgl. Kapitel 2.1.2).

2.1.1 ER-Modell

Das Entity-Relationship-Modell (ER-Modell) [62, 36] ist eine graphische Darstellung der Tabellen- und Beziehungsstruktur einer relationalen Datenbank. Es besteht aus Entitäten und Relationen, sowie Kardinalitäten, die die Wertigkeit der Beziehung beschreiben. Die Entität ist eine Einheit, welche mittels Schlüssel eindeutig bestimmbar ist und durch Eigenschaften, den Attributen beschrieben wird. Entitäten können in Beziehung

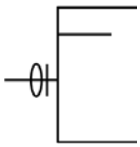
zueinander stehen, die durch drei Kategorien beschrieben werden, der 1:1-, 1:n- und n:m-Beziehung. Diese Bezeichnung ergibt sich aus den möglichen Kardinalitäten, welche aus einer Mindest- und Höchstanzahl zusammensetzt wird. Tabelle 2.2 verzeichnet alle möglichen Kombinationen, die mit diesen drei Kategorien gebildet werden können. Es ergeben sich vier Bereiche. Oben links befindet sich die 1:1-Beziehung, die 1:n-Beziehung oben rechts und unten links, sowie die n:m-Beziehung unten rechts. Die 1:1-Beziehung steht für mindestens 0 und maximal 1 (Typ c in der Tabelle) oder für mindestens 1 und maximal 1 (Typ 1 in der Tabelle) Beziehung zu anderen Entitäten. Das n bzw. m der 1:n- und n:m-Beziehung steht für beliebig viele und erweitert die beiden bisherigen Typen. In Kombination mit Typ 1 ergibt sich der Typ m, der für mindestens 1 und maximal viele steht. In Verbindung mit Typ c entsteht der Typ mc, welcher mindestens 0 und maximal viele bedeutet. Die Verwendung von Typ m oder Typ mc stellt die 1:n-Beziehung dar, werden nur die beiden verwendet, wird die n:m-Beziehung beschrieben. [62, 36]

Tabelle 2.2: Kombination der Beziehungstypen (stark angelehnt an Quelle [36])

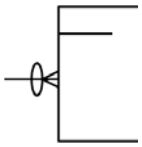
Typ	1	c	m	mc
1	(1,1)	(1,c)	(1,m)	(1,mc)
c	(c,1)	(c,c)	(c,m)	(c,mc)
m	(m,1)	(m,c)	(m,m)	(m,mc)
mc	(mc,1)	(mc,c)	(mc,m)	(mc,mc)

Eine n:m-Beziehung kann im relationalen Modell nicht direkt realisiert werden und benötigt die Verwendung einer Hilfstabelle, welche alle im Datensatz existierenden Schlüsselkombinationen der beiden Tabellen enthält. Dadurch werden aus der n:m-Beziehung zwei 1:n-Beziehungen zu der Hilfstabelle. Dargestellt werden können die Beziehungen in einem ER-Modell in der Krähenfußnotation, welche ebenfalls symbolisch die Minimal- und Maximal-Werte repräsentieren (vgl. Abbildung 2.1). [62, 36]

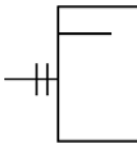
In Abbildung 2.2 befindet sich das ER-Modell einer stark vereinfachten Universität. Die Tabelle „student_vorlesung“ realisiert eine n:m-Beziehung, indem als Fremdschlüssel die Primärschlüssel der Tabellen „student“ und „vorlesung“ hinterlegt werden.



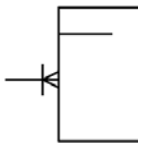
(a) Mindestens 0, maximal 1



(c) Mindestens 0, maximal viele



(b) Mindestens 1, maximal 1



(d) Mindestens 1, maximal viele

Abbildung 2.1: Krähenfußnotation der Beziehungen im ER-Modell

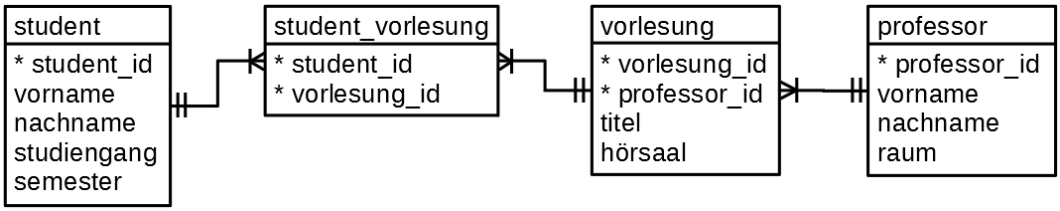


Abbildung 2.2: ER-Modell für die HAW

2.1.2 SQL als Sprache

Die Sprache Structured Query Language (SQL) lässt sich in drei Kategorien unterteilen, der Data Manipulation Language (DML) [62], der Data Definition Language (DDL) [62] und der Data Control Language (DCL) [16]. Jede dieser Kategorien besitzt typische Befehle, welche der Tabelle 2.3 entnommen werden können und diese erfüllen einen spezifischen Aufgabenbereich. Manipulationen der Daten erfolgen über die DML, damit können Daten eingegeben, geändert, gelöscht oder abgefragt werden. Des Weiteren erfolgt darüber die Definition und die Steuerung von Transaktionen. Die grundlegende Struktur der Datenbank, wie z.B. Domänen, Views, Relationen oder Schemata, wird mit der DDL beschrieben. Elemente zum Schutz der Daten in Datenbanken werden mit der DCL eingerichtet, darüber können Berechtigungen den Benutzern bzw. Rollen zugewiesen oder entzogen werden. [62, 16]

Tabelle 2.3: Typische Befehle für die Sprachen bei SQL (Entnommen aus Quelle [16])

DML	DDL	DCL
INSERT	CREATE	GRANT
UPDATE	ALTER	DENY
DELETE	DROP	
SELECT		

2.2 NoSQL Datenbanken

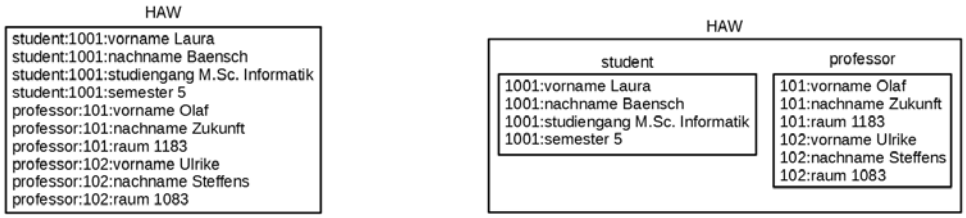
Unter NoSQL Datenbanken [36] werden jene verstanden, denen kein relationales Modell zugrunde liegt, dies bedeutet Not only SQL (NoSQL). Daraus folgt, dass andere Datenbanksprachen als SQL und keine Tabellen zur Speicherung der Daten verwendet werden. Des Weiteren sind die Datenbanken schemafrei. Darüber hinaus erfüllt das Datenbanksystem mindestens drei der fünf V's, die in der Big Data definiert werden, diese sind Volume, Variety und Velocity. NoSQL Datenbanken unterstützen die Datenreplikation und horizontale Skalierung. Das Konsistenzmodell (vgl. Kapitel 2.3) ist zumeist ein anderes als ACID und unterliegt oft dem CAP-Theorem, falls der Fokus auf Ausfalltoleranz oder Verfügbarkeit liegt. [36]

Im Folgenden wird ein Überblick über die vier Hauptmodelle von NoSQL Datenbanken gegeben, zu denen die Schlüssel-Wert- (vgl. Kapitel 2.2.1), dokumentenorientierten (vgl. Kapitel 2.2.2), Spaltenfamilien (vgl. Kapitel 2.2.3) und Graphdatenbanken (vgl. Kapitel 2.2.4) gehören. Hierbei handelt es sich um die Hauptmodelle bei NoSQL, ein weiteres Konzept sind die Multi-Modelle, bei denen in einem Datenbanksystem mehrere Arten kombiniert werden und somit dem Anwender über ein System verschiedene NoSQL Arten zur Verfügung stehen [26]. Beispielsweise bietet die OrientDB vier Modelle an und ist eine Graph-, dokumentenorientierte, Schlüssel-Wert- und Objektdatenbank. Dadurch kombiniert sie verschiedene Features in einem Datenbanksystem [46].

Zur Veranschaulichung der Darstellung der Daten in den jeweiligen NoSQL Datenbanken, wird das Beispielszenario des ER-Modell 2.2 der relationalen Datenbank verwendet. Nicht berücksichtigt wird dabei, ob generell ein Datensatz dieser Art für das jeweilige Datenbankmodell geeignet ist. Um auf einen Datensatz effizient zuzugreifen und verwalten zu können, muss abgewägt werden, welches Datenbankmodell für das Anforderungsprofil am besten geeignet ist.

2.2.1 Schlüssel-Wert-Datenbanken

Schlüssel-Wert-Datenbanken [36, 60, 9] verwalten ihre Daten, dem Namen entsprechend, in Schlüssel-Wert-Paaren und bilden damit die trivialste Art der NoSQL Datenbanken. Jeder Wert in der Datenbank besitzt einen Schlüssel, über diesen kann er identifiziert werden. Eine weitere Strukturierung der Daten ist bei diesem Datenbankmodell nicht vorgesehen, weshalb der Name des Schlüssels für eine eindeutige Identifizierung verwendet wird. Durch Sonderzeichen, wie z.B. Punkt, Doppelpunkt oder Schrägstrich kann der Name gegliedert und mehrere Informationen können übersichtlich hinterlegt werden. [36, 60, 9]
Die Kerneigenschaft einer Schlüssel-Wert-Datenbank ist, dass diese eine Menge von Schlüsseln, die identifizierenden Datenobjekte, verwaltet. Jeder Schlüssel ist genau einem Wert, dem deskriptiven Datenobjekt, zugeordnet. Durch diese eindeutige Identifikationsmöglichkeit, kann über den Schlüssel der Wert abgefragt werden. [36]
Essenziell für das effektive Auffinden von Daten ist das Erstellen von aussagekräftigen Schlüsseln. Eine, wie bei anderen Datenbanken übliche sequenzielle Nummerierung der Einträge, ist bei einer Schlüssel-Wert-Datenbank nicht funktional. Der Schlüssel ist die einzige Möglichkeit zur Definition, Strukturierung und Wiederfindung der zugehörigen Werte. [9]
Im Unterschied zur relationalen Datenbank wird keine Struktur wie z.B. die Organisation der Daten in Tabellen unterstützt. Ebenfalls wird keine komplexe Abfragesprache wie SQL geboten. Manche Schlüssel-Wert-Datenbanken bieten Namensräume an. Darüber können die Schlüssel thematisch gegliedert und in Kombination mit der Namensgebung der Schlüssel verwendet werden. Dadurch, dass die einzige Strukturierung über Schlüssel erfolgt, ergeben sich Namen, welche die relationale Struktur darstellen, wie z.B. Tabellenname:Primärschlüssel:Spaltenname. [60]
Die folgende Abbildung 2.3 zeigt einen Ausschnitt der Daten aus dem Universitätsmodell der HAW für eine Schlüssel-Wert-Datenbank. Abbildung 2.3a arbeitet ohne Namensräume, damit ist es notwendig den Tabellennamen miteinzubeziehen. Abbildung 2.3b verwendet Namensräume entsprechend des Tabellennamens, womit die explizite Nennung der Tabellen im Schlüssel umgangen wird.



(a) Ohne Namensraum (b) Mit Namensraum

Abbildung 2.3: Schlüssel-Wert-Datenbank

2.2.2 Dokumentenorientierte Datenbanken

Dokumentenorientierte Datenbanken [36, 60] erweitern das Schlüssel-Wert-Modell um eine stärkere Strukturierung der Daten. Diese werden in Dokumenten, welche aus Attribut-Wert-Paaren aufgebaut sind, hinterlegt. Des Weiteren sind sie schemafrei, d.h. es ist nicht notwendig für das Dokument ein Schema zu definieren. Jedes Dokument kann eigene Attribute besitzen. Ebenso ist eine Verschachtelung von Dokumenten möglich, sodass ein Dokument in einem Dokument gespeichert wird. Mittels Abfragen kann auf die kompletten Dokumente oder die verschiedenen Attribut-Wert-Paare zugegriffen werden. [36, 60] Abbildung 2.4 stellt die Kollektion HAW dar, in der der Beispieldatensatz des ER-Modell verwendet wurde (vgl. Abbildung 2.2). In Kollektionen können unterschiedliche Dokumente liegen. Hierbei wurde in dem Dokument zum Studenten per Referenz auf die Vorlesungen verwiesen. Verschachtelt in der Vorlesung liegen die Informationen zu den Professoren.

2.2.3 Spaltenfamilien-Datenbanken

Eine stärkere Strukturierung des Schlüssel-Wert-Schemas bieten die Spaltenfamilien-Datenbanken [36, 60, 9], welche die Daten in mehrdimensionalen Schlüsselräumen hinterlegen. Das Konzept basiert auf der Annahme, dass für einen Lesezugriff zumeist eine bestimmte Menge von Spalten benötigt wird, jedoch nicht alle. Somit wird eine Gruppierung von Spalten, die häufig zusammen benötigt werden, in Spaltenfamilien ermöglicht. Dadurch werden die Daten spaltenweise und nicht zeilenweise gespeichert, um den Zugriff effizienter zu machen. Dies führt dazu, dass unterschiedliche Zeilen verschiedene Spalten enthalten. [36, 60, 9]
Eine Spaltenfamilien-Datenbank zeichnet sich dadurch aus, dass die Daten in einem

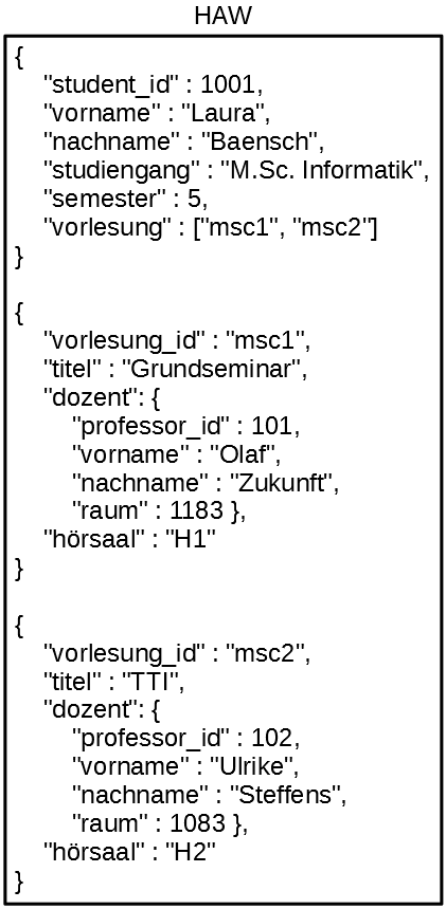


Abbildung 2.4: Dokumentenorientierte Datenbank

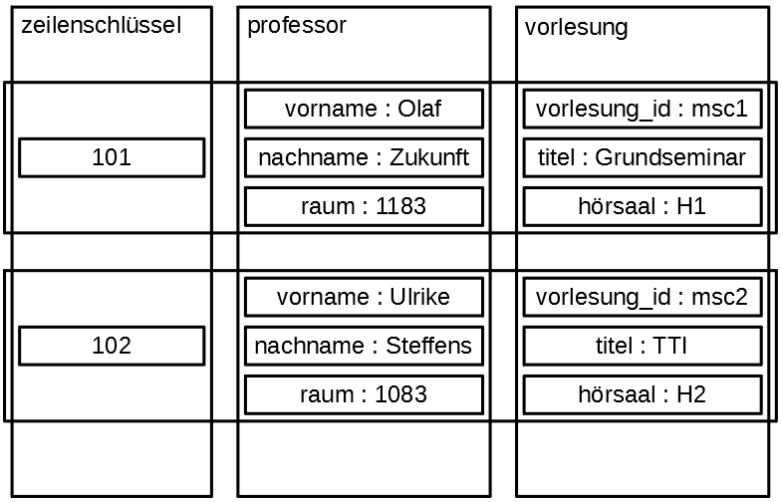


Abbildung 2.5: Spaltenfamilien Datenbank

mehrdimensionalen Schlüsselraum vorliegen und durch Zeilenschlüssel identifiziert werden, Spalten über Spaltenschlüssel. Die Spaltenfamilien definieren das Schema dieses Datenbankmodells und zeigen auf, welche Spalten in einem Zusammenhang stehen. In fragmentierten, sowie verteilten Architekturen ist aus Optimierungsgründen bezüglich der Antwortzeit, die Speicherung von kompletten Spaltenfamilien an einem Ort sinnig. [36]
Grundlegend ähneln Spaltenfamilien-Datenbanken den relationalen, da in beiden das Konzept von Spalten und Zeilen verwendet wird. Bei relationalen Datenbanken ist jedoch bei Daten, welche über verschiedene Tabellen verteilt liegen, ein Join nötig, um diese zu erhalten. Durch die Verwendung von Spaltenfamilien kann auf Spalten aus verschiedenen Tabellen mittels einen Aufrufs zugegriffen werden, da diese in einer Zeile verwaltet werden. [60]
Die Abbildung 2.5 zeigt die Tabellen „Vorlesung“ und „Professor“ des ER-Modells als Spaltenfamilien-Datenbank. Dadurch kann durch einen Aufruf auf die Informationen der Vorlesung und des entsprechenden Professors zugegriffen werden.

2.2.4 Graphdatenbanken

Die Graphdatenbank [36, 9, 60] bietet einen konträren Ansatz zur Modellierung der Daten, welcher die Beziehungen zwischen diesen veranschaulicht. Im Gegensatz zu den anderen Arten von NoSQL Datenbanken, befinden sich nicht die Informationen des einzelnen

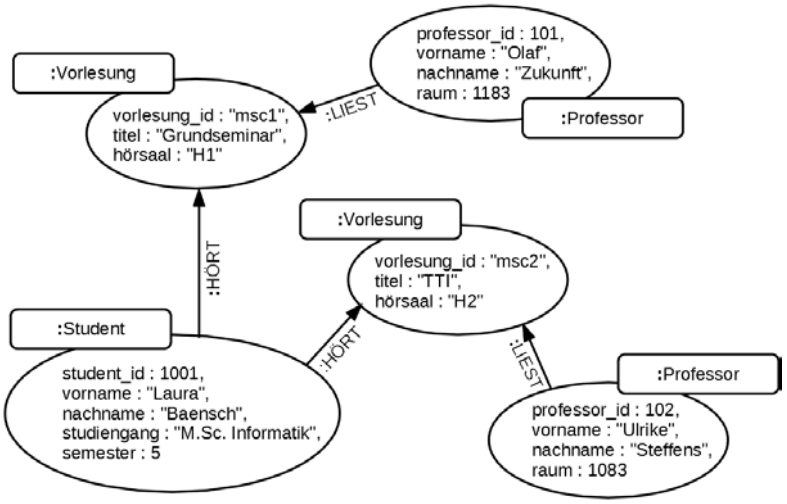


Abbildung 2.6: Graphdatenbank

Datensatzes im Fokus, sondern die Beziehungen zwischen den Datensätzen. Aufgebaut ist das Datenmodell in einer Graphenstruktur aus Knoten und Kanten, welche durch Attribute beschrieben werden. Knoten erhalten zusätzlich einen Bezeichner. Geeignet ist diese Form der Repräsentation von Daten, für Datensätze, die auf einer starken Vernetzung basieren. Damit bietet dieser Datenbanktyp eine Lösung für Datensätze, welche sich im relationalen Modell äußerst ineffizient lösen lassen, da komplexe Beziehungen mittels Join über verschiedene Tabellen ermittelt werden müssten. [36, 60, 9]
Graphdatenbanken liegt eine Graphenstruktur zugrunde, dementsprechend erfolgen Veränderungen am Datensatz über Graph-Transformationen oder Operationen. Diese sind im Bereich der Graphentheorie aus der Mathematik angesiedelt [9] und beziehen sich auf die Eigenschaften von Graphen, wie z.B. die Ermittlung von Pfaden im Graphen oder Nachbarknoten [36]. Damit die Konsistenz, im Bezug auf die Graphenstruktur, gewährleistet werden kann, sind Integritätsbedingungen überprüfbar [36].
Abbildung 2.6 zeigt exemplarisch, wie die Daten in einer Graphdatenbank repräsentiert werden können. Die Knoten sind von unterschiedlichen Typen, dem Student, dem Professor oder der Vorlesung. Die Beziehungen werden über Pfeile dargestellt.

2.3 Konsistenzmodelle

In diesem Kapitel werden die verschiedenen Konsistenzmodelle vorgestellt. Bei ACID (vgl. Kapitel 2.3.1) und BASE (vgl. Kapitel 2.3.2) handelt es sich um zwei gegensätzliche

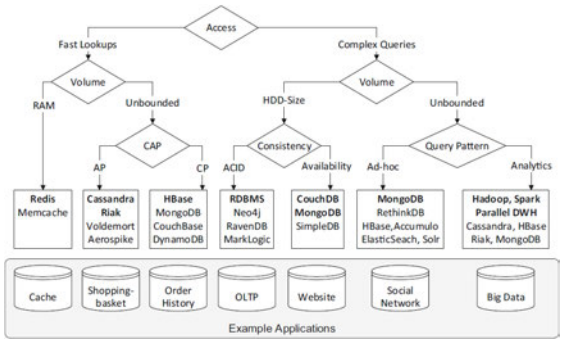


Abbildung 2.7: NoSQL Entscheidungshilfe (Entnommen aus Quelle: [17])

Modelle zur Konsistenz. Das CAP-Theorem (vgl. Kapitel 2.3.3) geht darauf ein, dass bei verteilten Systemen nur zwei von drei Zielen garantiert werden können.

Wie Kapitel 2.2 zu entnehmen ist, sind die NoSQL Datenbanken sehr unterschiedlich. Neben diesen Charakteristiken kann auch das unterstützte Konsistenzmodell ausschlaggebend sein, welche NoSQL Datenbank zu bevorzugen ist. Abbildung 2.7 zeigt eine Mögliche Auswahl der Datenbank auf, nachdem unter anderem bei dem Konsistenzmodell unterschieden wurde. Dazu werden exemplarisch Datenbanken, sowie mögliche Anwendungsbereiche genannt.

2.3.1 ACID

ACID [21, 36, 9] ist ein Akronym für die folgenden Begriffe: Atomicity (dt. Atomarität), Consistency (dt. Konsistenz), Isolation (dt. Isolation) und Durability (dt. Dauerhaftigkeit). Atomarität bedeutet, dass die Transaktion, welche mögliche Zwischenzustände haben kann, entweder vollständig oder gar nicht ausgeführt wird. D.h. eine Transaktion wird nur durchgeführt, wenn alle Zwischenschritte erfolgreich sind. Andersherum schlägt die komplette Transaktion fehl, sobald einer der Zwischenschritte nicht durchgeführt werden kann. Befindet sich die Datenbank vor und nach einer Transaktion in einem konsistenten Zustand, so ist diese insgesamt konsistent und die Daten garantiert widerspruchsfrei und alle definierten Protokolle oder Regeln wurden eingehalten. Eine Verletzung der Konsistenzbedingungen ist während einer Transaktion legitim, solange dies nach Abschluss nicht bestehen bleibt und wieder ein konsistenter Zustand erreicht wird. Die Isolation stellt sicher, dass keine Seiteneffekte bei parallel ablaufenden Transaktionen entstehen und diese somit voneinander isoliert sind. Dementsprechend kann auf Transaktionen, die noch nicht abgeschlossen sind oder sich in einem Zwischenzustand befinden, nicht

zugegriffen werden. Sind Datenbankzustände nur durch Transaktionen veränderbar und andernfalls bis dahin gültig, so liegt die Dauerhaftigkeit vor. Für unerwartete Störungen, wie Fehler, Abstürze oder andere Systemausfälle, bedingt die Dauerhaftigkeit, dass Transaktionen nur korrekt erfolgen. Diesem Modell gehören typischerweise relationale Datenbanken an, es gibt jedoch auch NoSQL Datenbanken, die dies erfüllen. [21, 36, 9]

2.3.2 BASE

Bei verteilten Systemen, welche z.B. ihren Schwerpunkt auf Ausfalltoleranz oder Verfügbarkeit legen, kann die Konsistenz nicht immer erfüllt werden. Entsprechend dazu gibt es das Konsistenzmodell BASE [51, 36, 9], ein Akronym von Basically available (dt. meist verfügbar), Soft State (dt. weicher Zustand) und Eventual consistency (dt. eventuell konsistent). Dies bedeutet, dass das System meistens verfügbar ist. Werden die Daten jedoch verändert oder befinden sie sich in einem inkonsistenten Zustand, kann eine Abfrage dieser Daten fehlschlagen. Das System befindet sich in einem weichen Zustand und dieser kann sich im Verlaufe der Zeit ändern, sodass die Konsistenz nicht immer erhalten werden kann und somit nur eventuell konsistent ist. Oftmals wird dieses Modell von NoSQL Datenbanken verwendet. [51, 36, 9]

2.3.3 CAP-Theorem

Das CAP-Theorem [8, 36, 9] ist ein Akronym für Consistency (dt. Konsistenz), Availability (dt. Verfügbarkeit) und Partition Tolerance (dt. Ausfalltoleranz). Dies basiert auf der Annahme, dass bei verteilten Systemen lediglich zwei von den drei Eigenschaften zutreffen können und somit die Kombinationen CA, CP oder AP möglich sind. Bei CAP bedeutet Konsistenz, dass alle lesenden Transaktionen bei einer Änderung der Daten den aktuellen Zustand erhalten. Verfügbarkeit steht dafür, dass laufende Anwendungen stetig in betrieb sind, auf Anfrage zur Verfügung stehen und die Antwortzeiten sich in einem akzeptablen Maß befinden. Entsteht keine Beeinträchtigung des Gesamtsystems, wenn ein Knoten im Rechnernetzwerk ausfällt, so liegt eine Ausfalltoleranz vor. Dementsprechend ist das Hinzufügen oder Entfernen von Knoten während des Betriebs möglich, ohne diesen unterbrechen zu müssen. Dieses Modell findet meist bei NoSQL Datenbanken Anwendung. [8, 36, 9]

Der Autor des CAP-Theorems merkt 12 Jahre später an, dass die These, dass lediglich zwei der drei Eigenschaften zutreffen können, missverständlich ist [7]. Dabei bedeutet diese Beschränkung der Eigenschaften nicht, dass die jeweiligen ausgewählten Eigenschaften komplett erfüllt werden, sondern dass es Zwischenschritte gibt und sie eher als eine Prozentzahl zwischen 0 und 100 betrachtet werden sollten. Ein weiterer Punkt ist, dass die Entscheidung zumeist zwischen der Erfüllung von A oder C zu treffen ist. Ein System kann in verschiedenen Subsystemen oder Anfragen individuell entscheiden, ob A oder C zu erfüllen ist. [7]

2.4 Datenbankmigration

Ziel einer Datenbankmigration ist es, die komplette Datenbank von einem Datenbankmanagementsystem in ein anderes umzuziehen. Wijaya et al. [63] stellt ein Framework vor, welches grundlegende Regeln für eine Migration behandelt und eine strukturierte Vorgehensweise aufzeigt. Dazu werden acht Schritte für den Migrationsprozess definiert. Im ersten Schritt soll analysiert werden, welche NoSQL Datenbank für den gegebenen Datensatz am besten geeignet ist, bezüglich der Architektur und Funktionalität. Der zweite Schritt befasst sich mit der Vorbereitung zur Migration. Dazu sollte erfasst werden, welche Daten und Funktionalitäten in das neue Modell übernommen werden können, welche Tools die Migration erleichtern oder durchführen sollen. Gegebenenfalls müssen eigene Programme/Tools erstellt werden, welche den Migrationsprozess unterstützen. Diese Vorbereitungsphase sollte gut geplant werden. Anschließend kann die Extraktion der Daten aus der Quelldatenbank erfolgen, im Idealfall mit so wenig Beeinträchtigungen des laufenden Betriebs wie möglich. Um die Daten in der NoSQL Datenbank sinnvoll zu hinterlegen, muss eine Transformation dieser erfolgen. Zur Absicherung kann dieser Schritt mit Validationsmethoden überprüft werden. Anschließend kann der umgeformte Datensatz in die Zieldatenbank eingelesen werden. Die Daten in der neuen Datenbank sollten validiert werden, um sicherzustellen, dass die Migration erfolgreich, sowie fehlerfrei erfolgt ist. Dies dient der Überprüfung der Integrität. Abschließend können Tests durchgeführt werden, welche sicherstellen sollen, dass die Datenbank die gleichen Ergebnisse liefert wie zuvor. Ebenso ist eine möglichst genaue und lückenlose Dokumentation der Migration sinnvoll. [63]
Wijaya et al. zeigen mit dem Framework einen grundlegenden Arbeitsablauf auf, welcher der generellen Praxis einer strukturierten Arbeitsweise folgt. Soll eine Migration

erfolgreich ablaufen, so ist eine detaillierte Planung des Ablaufes unablässig, denn unstrukturierte Arbeitsweisen können nicht nur den Prozess verlangsamen, sondern ihn auch fehlerhaft erfolgen oder im schlechtesten Fall fehlschlagen lassen.

2.5 Verwandte wissenschaftliche Arbeiten

Die im Folgenden vorgestellten wissenschaftlichen Arbeiten befassen sich mit der Migration von relationalen zu NoSQL Datenbanken. Je nach Art der Zieldatenbank unterscheidet sich das Verfahren, wie die Tabellenstruktur an die der Zieldatenbank angepasst wird. Das Kapitel gliedert sich nach den NoSQL Arten, Schlüssel-Wert- (vgl. Kapitel 2.5.1), Graph- (vgl. Kapitel 2.5.2), Spaltenfamilien (vgl. Kapitel 2.5.3) und dokumentenorientierte Datenbanken (vgl. Kapitel 2.5.4). Abgeschlossen wird diese Thematik mit einem Vergleich der verschiedenen NoSQL Datenbanken (vgl. Kapitel 2.5.5) bezüglich der Kerninformationen zur Migration.

2.5.1 Schlüssel-Wert-Datenbanken

Aufgrund der simplen Struktur von Schlüssel-Wert-Datenbanken wird bei der Migration kein großer Handlungsraum geboten. Alle Daten werden als Schlüssel-Wert-Paare hinterlegt, um effizient auf die Informationen zugreifen zu können. Dies wird durch das Fehlen von Struktur erzielt. Der Migrationsprozess ist hierbei weniger eine Strukturumwandlung, sondern ein Definieren von Schlüsseln, welche es effizient erlauben, die Daten wiederzufinden.

El Alami et al. [14] benötigen die Metainformationen aus der relationalen Datenbank für ihren Ansatz. Darunter fallen die Informationen zu Tabellen, Spalten und Schlüssel. Zieldatenbank ist Redis, bei der jeweils pro Schlüssel ein Datensatz einer Tabelle hinterlegt wird. [14]
Dieses Verfahren migriert die Daten direkt, ohne den Datensatz in verschiedene Schlüssel zu teilen, wie es in Kapitel 2.2.1 in den Grundlagen erläutert wurde. Je nachdem welche Daten bei einem Zugriff gewünscht sind, kann dies vor- oder nachteilig sein.

2.5.2 Graphdatenbanken

Graphdatenbanken bieten einen konträren Ansatz zur Verwaltung von Daten. Hierbei werden die Tabellen und Beziehungen in einen Graphen aus Knoten und Kanten umgewandelt. Allgemein betrachtet werden für diesen Prozess die Zeilen aus einer Tabelle in einen Knoten, die Spalten in Eigenschaften und die Beziehungen zwischen Tabellen in Relationen dargestellt. Um auf die Daten effizienter zugreifen zu können, kann diese grobe Darstellung der Umwandlung angepasst werden. Im Folgenden werden vier Konzepte vorgestellt.

Virgilio et al. [11] stellen ein Konzept vor, bei dem berücksichtigt wird, dass Informationen aus verschiedenen Knoten häufig zusammen abgefragt werden können. Damit nicht zu viele Daten in einem Knoten fusioniert werden, wird dies limitiert. Für diesen Prozess werden Schema Paths (SP) gebildet, welche alle möglichen Pfade durch die Tabellen abbilden, gestartet von einer Quelle bis zu einer Senke. Für jeden SP wird zwischen fünf Fällen unterschieden und basierend darauf ermittelt, wie der abschließende Graph aussieht. Die Fälle berücksichtigen in welcher Position im SP gerade gearbeitet wird und untersuchen den Zustand der Vorgänger und Nachfolger. Basierend auf den Informationen wird entschieden, ob ein Knoten, eine Eigenschaft oder eine Kante hinzugefügt wird. Des Weiteren bietet der Ansatz einen Umgang mit Join-Abfragen aus SQL. Über die Verwendung der SP kann auf den kürzesten Pfad auf die Informationen des Join zugegriffen werden. [11]
Dieser Ansatz versucht zu berücksichtigen, dass Abfragen Informationen aus verschiedenen Knoten benötigen. Damit dies effizient erfolgen kann, werden Informationen zusammengefasst. Dabei wird festgelegt, ab wann keine Informationen mehr zusammengefasst werden. Wie von den Autoren angemerkt ist die Berücksichtigung von realen Abfragen für das Zusammenfassen zukünftig geplant [11].

Singh et al. [55] bietet nach eigener Aussage einen besseren Ansatz als der von Virgilio et al. [11]. Dieser soll Entitäten, welche häufig bei Abfragen zusammen auftreten, im Graphen nah beieinander legen, um die Abfragezeit zu optimieren. Hier wird ebenfalls wie bei Virgilio et al. [11] zwischen den fünf Fällen unterschieden und sie sind in ihrer Definition identisch. Es ist daher nicht ersichtlich, inwiefern sich der Ansatz von Virgilio unterscheidet. [55]
Obwohl die Autoren zu Beginn anmerken, dass ihr Verfahren besser ist, als das von Virgilio [11], wurde keinerlei Evaluierung durchgeführt, ob dies auch den Tatsachen entspricht. Es wurden lediglich exemplarisch Abfragen bei der relationalen und Graphdatenbank

durchgeführt, bei denen die Graphdatenbank besser performt. Verifiziert werden konnte in der Arbeit von Singh et al. [55] jedoch nicht, dass der gewählte Ansatz im Bezug auf das bestehende Konzept bessere Ergebnisse erzielt.

Orel et al. [45] fasst den Migrationsprozess für Graphdatenbanken in mehreren Aspekten zusammen, welche kurz erläutert werden. Da es sich um kein relationales Modell handelt, muss der Graph weder azyklisch, noch zusammenhängend sein. Üblicherweise bildet die Zeile einer Tabelle einen Knoten ab. Enthält dieser Fremdschlüssel, so werden diese Kanten. Ist die Zeile aus einer n:m-Beziehung, so wird diese nicht als Knoten, sondern über Kanten dargestellt. Jeder Knoten, sowie jede Kante erhält ein Label, um eindeutig festzulegen, welchem Typ sie gehören. Bei Knoten ist dies zumeist der Tabellenname, ebenfalls bei Kanten, die eine n:m-Tabelle abbilden. Bei Kanten von Fremdschlüsseln, wird dieser als Label verwendet. Die Eigenschaften, welche Knoten und Kanten erhalten können, sind die Attribute aus der relationalen Tabelle. Alle Knoten und Kanten, welche eine Zeile aus der relationalen Datenbank darstellen, erhalten eine eindeutige ID, den Primärschlüssel. Handelt es sich um einen zusammengesetzten Schlüssel, so wird dieser zu einem eindeutigen zusammengefasst. Gibt es mehr als einen Fremdschlüssel zwischen zwei Tabellen, d.h. liegen parallele Beziehungen vor, so werden zwei Kanten mit zwei Labeln hinzugefügt. [45]
Die Autoren stellen damit ein Regelwerk vor, mit dem schematisch die Tabellenstruktur in eine Graphenstruktur aufgelöst wird.

Unal et al. [61] verwenden bei ihrem Migrationsprozess externe Tools zum Erfassen des Schemas der relationalen Datenbank. Die Migration zur Graphdatenbank beruht auf den dadurch gewonnenen Informationen. Die Umwandlung des Schemas basiert auf Transformationsregeln, die festlegen, wie das Schema in Knoten, Eigenschaften und Beziehungen umstrukturiert werden kann. Dazu bildet jede Zeile einer Tabelle einen eigenen Knoten, das Label wird durch den Tabellennamen festgelegt und Spalten werden zu Eigenschaften des Knotens. Technische Primärschlüssel, also jene, die automatisiert angelegt wurden, können entfernt werden. Alle anders gebildeten Primärschlüssel sollten erhalten und gegebenenfalls mit einem Unique Constraint belegt werden. Eigenschaften, welche häufig in Abfragen auftreten, werden indiziert. Über Fremdschlüssel werden die Beziehungen realisiert und können anschließend gelöscht werden. Daten mit Default Werten werden entfernt, ebenso ist bei denormalisierten Daten abzuwägen, diese in speziellen Knoten zu verwalten. Zwischentabellen, welche bei der Realisierung von n:m-Beziehungen entstehen, werden in der Graphdatenbank in Beziehungen dargestellt. Beinhaltet solch eine Tabelle Attribute, so werden diese Eigenschaften der Beziehung. [61]

Dieser Ansatz verwendet bei der Migration die Funktionalität der Indizes, welche auf häufigen Abfragen beruhen. Nicht erkenntlich ist jedoch, wie die Auswertung darüber erfolgen soll, welche Abfragen häufig sind. Es wird bei diesem Ansatz berücksichtigt, dass die SQL Tabelle nicht zwingend in der normalisierten Form vorliegt und Redundanzen vorhanden sind. Hier wird ebenfalls nicht erläutert, wie der Prozess der Erkennung der Duplikate erfolgen soll. Anzumerken ist, dass das Löschen der Primärschlüssel und das erneute Anlegen mit Unique Constraints bei Neo4j ein Prozess ist, der eine beschleunigte Performance bietet [44].

2.5.3 Spaltenfamilien Datenbanken

Migrationen von relationalen zu Spaltenfamilien Datenbanken fokussieren sich bei der Umwandlung auf die Verschachtelung von Daten. Dabei werden verschiedene Tabellen aus der relationalen Datenbank in entsprechenden Spaltenfamilien verwaltet. Die im Folgenden vorgestellten Ansätze besitzen verschiedene Schwerpunkte bei der Migration.

In einer Fallstudie erläutert Li [32] mittels dreier Regeln einen Prozess, über den die Migration von einer relationalen Datenbank zur HBase erfolgen kann. Die erste Regel befasst sich mit der Zusammengehörigkeit von Daten. Korrelierende Daten aus verschiedenen Tabellen sollten in einer Spaltenfamilie hinterlegt werden. Unter der Korrelation von Daten werden bei diesem Konzept jene verstanden, auf die zusammen zugegriffen wird (z.B. lesend oder schreibend). Dies dient der ausschließlichen Zusammenfassung von Daten, welche bei einem Aufruf vereint benötigt werden. Die zweite Regel befasst sich mit dem Umgang von Beziehungen zwischen Tabellen. Werden Informationen der referenzierten Tabelle benötigt, so wird der Fremdschlüssel auf beiden Seiten hinterlegt. Dazu wird je nach Kategorie der Beziehung (vgl. Kapitel 2.1.1) unterschiedlich verfahren. Beziehungen, welche auf einer Seite eine n-Beziehung aufweisen, werden in einer neuen Spaltenfamilie verwaltet. Dazu wird die Spaltenfamilie bei einer 1:n-Beziehung auf der 1-Seite angelegt, bei n:m-Beziehungen auf beiden Seiten. Die dritte Regel dient dem Abbau der Fremdschlüssel und dem Zusammenfassen der Tabellen. Dazu muss die Haupttabelle ermittelt werden, welche eine Tabelle ist, die unabhängig von anderen ist. Die eingebundene Tabelle ist jene, die über Fremdschlüssel referenziert wird und deren Daten von der Tabelle abhängig sind, die auf sie referenziert. [32]
Da es sich hierbei um eine Fallstudie handelt, wurde das Verfahren exemplarisch an einem Beispiel vollzogen und bietet keinen automatisierten Ansatz. Da das Beispiel mit einem simplen Datensatz arbeitet, wird auf die einfache Verschachtelung eingegangen, jedoch

nicht auf die multiple, da der Beispieldatensatz nicht die nötige Größe und Komplexität aufweist.

Der Umgang mit der Verschachtelung, ebenfalls der multiplen, wird in der Arbeit von Zhao et al. [67] erläutert. Dazu werden drei verschiedene Szenarien betrachtet, der Migration einer einzelnen Tabelle, einer einfachen Verschachtelung und der multiplen Verschachtelung. Bei einer einzelnen Tabelle wird der Primärschlüssel der relationalen Datenbank als der Zeilenschlüssel übernommen, die restlichen Spalten werden in einer Spaltenfamilie aus der Kombination des Bezeichners der Tabelle und der Spalte übertragen. Bei der einfachen Verschachtelung entstehen zwei Spaltenfamilien für die zwei zu migrierenden Tabellen aus dem relationalen Modell. Der Zeilenschlüssel ist wie bei der Migration einer einzelnen Tabelle, der Primärschlüssel einer der beiden Tabellen. Mit den restlichen Spalten und den Bezeichnern dieser wird gleich verfahren wie bei der Migration einer einzelnen Tabelle. Die Migration mit einer multiplen Verschachtelung erfolgt über Tabellen aus der relationalen Datenbank und einer Tabelle der Spaltenfamiliendatenbank. Mit den Tabellen aus der relationalen Datenbank wird gleich verfahren, wie bei der einfachen Verschachtelung. Die multiple Verschachtelung, welche sich bei der Migration mit einer Tabelle aus der Spaltenfamiliendatenbank ergeben würde, wird aufgelöst, indem die Spaltenfamilien umbenannt und zu einer einzigen zusammengefasst werden. Dazu wird der Tabellenname allen Spaltenfamilien hinzugefügt, welche nicht den Zeilenschlüssel gestellt haben. [67]
Neben der Migration der Daten befassen sich Zhao et al. [67] ebenfalls mit der Anpassung der Abfragen. Drei Faktoren sind dabei wichtig: es muss die korrekte HBase Tabelle verwendet werden und die entsprechenden Spaltenfamilien, sowie Spalten gefunden werden. Dazu wird aus der relationalen Datenbank ein Abfragegraph erstellt, bei dem die Tabellen als Knoten und die Beziehungen als Kanten dargestellt werden. Zu jedem Knoten wird das Level, in welchem sich dieser befindet und der Elternknoten hinterlegt. Die Namensgebung wurde schematisch festgelegt, bestehend aus dem Tabellen- und Spaltennamen, getrennt mit einem Doppelpunkt. Liegt eine multiple Verschachtelung vor, so werden die Namensteile durch einen Punkt verknüpft. Z.B. in Tabelle A liegt Tabelle B verschachtelt vor, so setzt sich der Name wie folgt zusammen: TabellennameA:TabellennameB.SpaltenName. Liegt keine Verschachtelung vor, so ist das Schema TabellenName:SpaltenName. [67]
Dieser Ansatz nimmt bei der Migration Änderungen an der Struktur der Daten vor und bietet einen Lösungsansatz für den Umgang mit Abfragen, welche ebenfalls auf die Änderungen angepasst werden müssen. Gelöst wird das Problem der multiplen Verschach-

telung über das Abflachen in eine einfache Verschachtelung, durch die Anpassung der Bezeichner.

Lee et al. [31] stellen ein Verfahren vor, welches die Daten eigenständig aus dem relationalen Schema denormalisiert und in die NoSQL Datenbank HBase migriert. Die Vorgehensweise orientiert sich an dem DDI Prinzip, welches für Denormalization, Duplication und Intelligent Keys steht. Durch die Denormalisierung werden bewusst Redundanzen hinzugefügt und es entstehen Duplikationen der Daten. Dadurch wird eine Zusammenfassung der Daten aus verschiedenen Tabellen in einer neuen, eigenständigen Tabelle ermöglicht, deren eindeutige Identifikation über den intelligenten Schlüssel erfolgt. Mittels dieses Prinzips werden Abfragen über verschiedene Tabellen verhindert und durch eine einzelne Abfrage kann auf alle Informationen zugegriffen werden.
Das Konzept zur Migration basiert auf einer speziellen Tabelle in MySQL, welche alle Schemainformationen beinhaltet und damit kann auf die Primär- und Fremdschlüssel jeder Tabelle zugegriffen werden. Aus dieser Tabelle wird zu Beginn des Migrationsprozesses eine ausgewählt und eine Linked-List angelegt. Diese dient primär dazu, die Verkettung von Beziehungen abzubilden. Die Bezeichner einer Spalte sollten ab diesem Schritt mit dem Tabellennamen verknüpft werden, denn Spalten aus verschiedenen Tabellen können den gleichen Bezeichner haben. Somit ist dennoch eine eindeutige Bestimmung gewährt. In der Linked-List wird der Primärschlüssel der aktuellen Tabelle hinterlegt und untersucht, ob Fremdschlüssel vorhanden sind. Ist dies der Fall, wird der Fremdschlüssel ebenfalls hinterlegt. Dieser Prozess wird solange wiederholt, bis keine Fremdschlüssel mehr vorhanden sind und die Linked-List dementsprechend die Verkettung der Tabellen ausgehend von der Starttabelle abbildet. Sind alle Tabellen untersucht worden, wird der Zeilenschlüssel aus den gespeicherten Primär- bzw. Fremdschlüsseln generiert. [31]
Bei diesem Vorgehen werden durch die Erfassung der Verkettung der Daten ausnahmslos alle Tabellen und deren Spalten in einer Spaltenfamilie zusammengefasst. Je nachdem wie der zugrundeliegende Datensatz über Beziehungen verknüpft ist, kann dies zu hohen Redundanzen führen. Daraus resultierend steigt der Speicherplatzbedarf. Wie von den Autoren angemerkt, ist das Teilen der Spalten in verschiedene Spaltenfamilien ein zukünftiges Thema [31].

2.5.4 Dokumentenorientierte Datenbanken

Die komplexen Beziehungen zwischen den Tabellen aus dem relationalen Modell werden bei dokumentenorientierten Datenbanken durch die Verschachtelung (Embedding)

oder der Referenz aufgelöst. Bei der Verschachtelung werden verschiedene Tabellen oder Dokumente zusammengefasst und in einem Dokument verwaltet. Die Referenz ähnelt den Beziehungen aus dem relationalen Modell und wird in einem Dokument hinterlegt. Sie verweist dementsprechend auf ein anderes Dokument und die Informationen liegen verteilt in zwei Dokumenten vor. Die Herausforderung besteht darin, beide Konzepte ausgewogen und an die Anforderungen der Datenbank angepasst, zu verwenden. Beide Möglichkeiten bieten verschiedene Vorteile. Die Verschachtelung ermöglicht es, mittels eines Aufrufs, auf sämtliche Informationen zuzugreifen, jedoch entstehen dadurch Redundanzen, welche mehr Speicherplatz benötigen. Bei einer Referenz kann verhindert werden, dass Dokumente zu komplex oder verschachtelt werden und dadurch der Zugriff auf einzelne Informationen erschwert wird. Erfolgt eine Abfrage über mehrere Referenzen, so verschlechtert sich die Antwortzeit. Insgesamt ist die Verwendung von Referenzen platzsparend, da keine Redundanzen gebildet werden. Welche der beiden Möglichkeiten zu bevorzugen ist, hängt von der Struktur des Datensatzes ab und welche Abfragen typischerweise auf dieser erfolgen. Im Folgenden werden verschiedene Herangehensweisen an diese Problematik vorgestellt.

Zhao et al. [65] haben die Konzepte des relationalen Modells in der dokumentenorientierten Datenbank MongoDB nachgewiesen und somit die theoretische Grundlage für eine Migration vom relationalen Modell zum dokumentenorientierten gelegt. Die relationale Algebra umfasst sechs primitive Operatoren, weitere können mittels Ableitungen von diesen ausgedrückt werden. Dies ist ebenfalls bei der MongoDB möglich. Eine weitere Komponente, die untersucht wurde, sind die Abfragemöglichkeiten mittels der relationalen Algebra unter der Verwendung von vier Operatoren. Für die MongoDB wurden diese Operatoren unter den Verhältnissen des Single Sharding und des Multiple Sharding untersucht. Bei Sharding handelt es sich um ein Konzept, welches nicht nur bei der MongoDB zu finden ist, sondern bei verschiedensten NoSQL Datenbanken, bei denen eine Kollektion in mehrere Shards unterteilt wird, um eine Lastenverteilung zu erreichen. [65] Die weiterführende Arbeit [67] befasst sich mit dem Umgang des Join-Operators bzw. dem Wegfallen dessen in dokumentenorientierten Datenbanken. Dazu wird eine verschachtelte Struktur gebildet, welche effizientere Abfragen auf den Daten zulässt. Verifiziert wurde dies mittels vier SQL Abfragen, welche eine steigende Anzahl von Join-Operationen enthalten. D.h. die erste Abfrage bezieht sich auf eine Tabelle, die vierte Abfrage bildet einen Join über vier Tabellen. Die dokumentenorientierte Datenbank erwies sich mit der Verschachtelung performanter als die relationale mit den Join-Operationen über die Tabellen. [67]

Die beiden Paper legen die theoretische Grundlage für die Migration von relationalen zu dokumentenorientierten Datenbanken. Zum einen wurde exemplarisch an der MongoDB nachgewiesen, dass diese fähig ist, die Daten und Abfragen wie eine relationale Datenbank zu handhaben. Zum anderen kann eine Verbesserung der Performance erzielt werden, indem in der dokumentenorientierten Datenbank die Join-Operationen entfallen, da eine verschachtelte Struktur vorliegt. Dadurch verringert sich die Abfragezeit, jedoch steigt durch die entstandenen Redundanzen bei der Verschachtelung der Platzbedarf. [29]

El Alami et al. [13] stellen einen Ansatz vor, bei dem die Daten aus der relationalen Datenbank analysiert und kategorisiert werden, um dann in einem Metamodell hinterlegt zu werden. Dazu wird zu jeder Tabelle erfasst, welche Spalten, Attribute und Schlüssel diese besitzt, sowie die Relationen mit den Beziehungstypen. Insgesamt ergibt sich daraus die grundlegende Struktur der zugrundeliegenden relationalen Datenbank. [13]
Hier wird nur auf die Kernproblematiken eingegangen, dem Extrahieren der Metainformationen und der Umgang mit den Beziehungen. Es wird ein Konzept vorgestellt, welches die notwendigsten Metadaten aus der relationalen Datenbank extrahiert und zwischenspeichert, sodass darauf basierend eine neue Struktur gebildet werden kann. [29]

Liang et al. [33] definieren ein Zwischenmodell, in dem die Daten aus einer relationalen Datenbank in Objekten hinterlegt werden. Von diesen Objekten ausgehend, kann eine Weiterverarbeitung für verschiedene NoSQL Datenbanken stattfinden. Eine Entität aus dem relationalen Modell entspricht einem Objekt. Zusätzlich dazu werden Informationen zu Eigenschaften, Beziehungen und Daten- und Abfrage-Features gespeichert. Die Features sorgen für den Erhalt der Integrität beim Migrationsprozess. Relevante Informationen zu der Größe des Datensatzes oder andere wichtige Informationen zu den Daten werden von den Daten-Features verwaltet. Die Abfrage-Features enthalten häufige Abfragen und können darüber der Zieldatenbank zur Verfügung gestellt werden. Die Strategien befassen sich mit der Umwandlung der Datenstruktur an die Zieldatenbank. Für jeden Datenbanktyp wird eine eigene Strategie benötigt, damit eine an die Zieldatenbank angepasste Umstrukturierung erfolgen kann. [33]
Das Konzept des Zwischenmodells bietet die Option, eine Migration für möglichst viele NoSQL Datenbankmodelle bereitzustellen. Dazu muss jedoch für jedes Modell eine Strategie definiert werden, welche die Transformation der Daten übernimmt. In dieser Arbeit wird nicht auf die Realisierung in den Strategien eingegangen, obwohl exemplarisch die Umwandlung der Struktur für eine Spaltenfamilien und dokumentenorientierte Datenbank gegeben wird. Vorteilhaft ist, dass durch das Abfrage-Feature die Abfragen einen

Einfluss auf die Bildung der neuen Struktur haben können. Durch die Erweiterung der Metadaten um die beiden Features, kann die Struktur besser an die Anforderungen des Datensatzes angepasst werden.

Die MigDB[1] [34] ist ein Tool, welches ein Künstliches Neuronales Netz (KNN) für die Empfehlung von Referenz oder Verschachtelung verwendet. Intern entscheiden beim KNN bei n-Beziehungen grundsätzlich Größe und Komplexität des Datensatzes der Tabelle über Referenz oder Verschachtelung. Bei entsprechend großen/komplexen „n“ sind Referenzen zu präferieren. Erfolgt eine manuelle Anpassung durch den Anwender für die vorgeschlagene Struktur des Datensatzes, wird dies an das KNN weitergeleitet. Dadurch kann das KNN lernen und zukünftig angepasstere Entscheidungen empfehlen. Neben der Migration der Daten bietet die MigDB zwei Module für die Abfragen an. Zum einen kann das Modul SQL-Abfragen automatisiert in den MongoDB Dialekt übersetzen, zum anderen gibt es das Hilfemodul, welches bei der Erstellung von MongoDB Abfragen unterstützt. Dies ermöglicht auch Einsteigern, Abfragen zu formulieren. [34]
Für dokumentenorientierte Datenbanken sind grundsätzlich nur die zwei Entscheidungsmöglichkeiten vorhanden: Verschachtelung oder Referenz. Nicht nur die Größe oder Komplexität des Datensatzes beeinflusst, welche Entscheidung sinnvoller ist, sondern auch die Abfragen. Diese werden jedoch vom KNN nicht berücksichtigt und somit kann es nicht unterscheiden, mit welcher Motivation der Anwender Anpassungen vorgenommen hat. Z.B. werden für das KNN hauptsächlich Migrationen durchgeführt, bei dem Abfragen auf den kompletten Datensatz erfolgen, so wird die Verschachtelung immer der Referenz vorgezogen. Das KNN lernt, dass beliebig große Datensätze zu verschachteln sind. Wird das Tool anschließend für eine Migration verwendet, bei dem Referenzen benötigt werden, da andernfalls auf mehrfach verschachtelte Daten zugegriffen wird, wird das KNN dennoch Verschachtelung empfehlen. Das Lernen durch Benutzerentscheidungen ist ohne die Berücksichtigungen der Abfragen nicht sinnvoll, da sonst keine besseren Entscheidungen durch das KNN getroffen werden können.

Der Digibrowser von Karnitis et al. [25] ist die Realisierung von Migrationen zwischen relationalen und relationalen zu dokumentenorientierten Datenbanken. Die referenzierte, gleichnamige Website[2] zu diesem Tool existiert nicht mehr. Der Prozess für die Umwandlung der Struktur an die Zieldatenbank wird in drei Schritte gegliedert, dem physischen Level und dem ersten und zweiten logischen Level der Daten. In dem physischen Level werden Tabellen ähnliche Strukturen gebildet die Table-Like-Structures (TLS). Darunter

[1] https://github.com/migdbdev
[2] http://digibrowser.com/

fallen sowohl Tabellen, als auch Views. Views bestehen aus einer definierten Konstellation von Tabellenausschnitten und können somit Informationen aus verschiedenen Tabellen verknüpfen. Das physische Level erfasst die Hinterlegung der Daten in der relationalen Struktur, in typischerweise normalisierter Form. Das erste logische Level erfasst die Daten als eine logische Einheit. Das beschreibt den Zusammenhang der Daten, unabhängig davon, wie diese in der Datenbank hinterlegt sind. Z.B. kann dem Beispieldatensatz der Autoren mit dem Universitätsbeispiel entnommen werden, dass zu einem Studenten Informationen zu der Person und den Noten verteilt gespeichert werden. Physisch sind diese getrennt, logisch jedoch zusammenhängend, denn die Noten werden dem Studenten zugeordnet. Das zweite logische Level dient dazu, abschließend den Aufbau der Dokumente zu bestimmten und basiert auf den TLS Strukturen. Mittels Tiefensuche Algorithmus werden weitere TLS gefunden und fusioniert. Darüber wird eine Verschachtelung der Daten erzielt. Der vorgestellte Algorithmus ist noch nicht fähig, extreme Verschachtelungen zu verhindern. Abschließend muss der Anwender bestimmen, von welchen TLS ausgehend die Tiefensuche und somit die Verschachtelung beginnt und wie stark diese erfolgen soll. [25]
Dieser Ansatz ist im Vergleich zu den anderen sehr komplex und dient dazu, ein Verständnis für die Struktur der zugrundeliegenden Daten zu bekommen und dann mittels Verifizierung durch den Anwender ein Template für die dokumentenorientierte Datenbank zu erstellen. Hier steht nicht im Vordergrund, dem Anwender automatisiert einen möglichst guten Vorschlag für eine Migration zu stellen, sondern ein Verständnis aufzubauen und basierend darauf, hauptsächlich manuell die Struktur zu bestimmen. Ebenso wird die Entscheidung nicht abgenommen, welche Tabellen besser verschachtelt oder referenziert werden. Der Anwender bestimmt von welcher Tabelle ausgehend die Verschachtelung beginnen soll.

Hamouda et al. [22] migrieren von relationalen zu dokumentenorientierten Datenbanken, ausgehend vom ER-Modell. Dazu wird das Document-Oriented Data Schema (DODS) gebildet. Das DODS besteht aus drei Teilen, den Komponenten, den Features und dem Mapping zwischen ER-Modell und DODS Spezifikationen. Die DODS Komponenten umfassen das logische und physische Modell der Datenbank, dazu gehören z.B. die Beziehungen, Attribute oder eine Kollektion, bzw. ein Dokument, welches das Äquivalent zu den Tabellen und Einträgen einer relationalen Datenbank ist. Die Features des DODS dienen zur Beschreibung des Datenmodells, wie z.B. die verschiedenen Schlüssel, Beziehungen etc. Beim Mapping zwischen ER-Modell und DODS Spezifikationen werden die unter Komponenten und Features erfassten Informationen auf die entsprechende Realisierung

in der dokumentenorientierten Datenbank gemappt. Auch bei diesem Ansatz ist der Umgang mit den verschiedenen Beziehungstypen besonders relevant. n:m-Beziehungen werden in unterschiedlichen Kollektionen hinterlegt. Bei 1:n-Beziehungen erfolgt die Verschachtelung, solange das „n“ kleiner ist als ein paar hundert Dokumente. 1:1-Beziehungen werden verschachtelt, es sei denn, die Daten auf beiden Seiten sind sehr groß oder mehrere Beziehungen bestehen auf die Tabelle, dann wird eine Referenz verwendet. [22]
Durch die Wahl von Thresholds für die Verschachtelung und die Referenz ist keine komplexe Logik nötig, um zwischen den beiden Optionen zu entscheiden. Werden Abfragen aus verschiedenen Bereichen der neuen Dokumente gestellt, kann dies ein praktikabler Ansatz sein, da beide Methoden gemischt werden. [29] Jedoch kann diese Arbeit keine typischen Abfragen erkennen und basierend darauf entscheiden, welche Option besser ist.

NoSQLayer[3] [54] ist ein Ansatz zur Migration, welcher sich in zwei Module unterteilt, dem Data Migration Module und dem Data Mapping Module. Das Data Migration Module erfasst sämtliche Metadaten der relationalen Datenbank, welche für die Umwandlung der Struktur nötig sind. Dazu gehören neben Informationen zu den Tabellen und Beziehungen ebenfalls Indizes. Beziehungen werden durch dieses Modul mittels Referenzen dargestellt. Das Mapping Module erlaubt es weiterhin SQL Abfragen zu stellen, welches dann diese umwandelt und an die NoSQL Datenbank weiterleitet. [54]
Die Autoren dieses Papers scheinen nur mit Referenzen zu arbeiten, da explizit darauf hingewiesen wird, dass Referenzen die Beziehungen darstellen. Des Weiteren wird das Konzept der Verschachtelung in der gesamten Arbeit nicht erwähnt. Dementsprechend wird bei diesem Ansatz das relationale Modell in einer dokumentenorientierten Datenbank nachempfunden und das Potential der Verschachtelung nicht ausgeschöpft. [29]

Kuszera et al.[4] [30] stellt einen Ansatz vor, bei dem die Daten und Beziehungen aus der relationalen Datenbank in Directed Acyclic Graph (DAG) dargestellt werden. Entsprechend stehen die Tabellen für die Knoten und die Beziehungen für die Kanten. Jeder DAG entspricht einer Einheit in der NoSQL Datenbank und entlang der Kanten kann die Transformation nachvollzogen werden, wie die Tabellen zu verschachteln sind. Der DAG entscheidet, ob eine Verschachtelung oder eine Verschachtelung eines Arrays von JavaScript Object Notation (JSON)-Objekten erfolgen soll. Die Referenz, welche die Beziehungen der relationalen Struktur darstellt, wird bei diesem Algorithmus nicht berücksichtigt. Liegt beim Blatt es DAG die Beziehung „1“ vor, so erfolgt eine Verschachtelung,

[3] https://github.com/nosqlayer/code
[4] https://github.com/evandrokuszera/metamorfose

liegt „viele“ vor, so erfolgt die Verschachtelung eines Arrays von JSON-Objekten. Dies wird solange durchgeführt, bis alle DAG Einheiten eine gemeinsame Wurzel haben. [30]
Bei dieser Arbeit wird die Referenz nicht berücksichtigt und alle Tabellen der relationalen Datenbank in einem Dokument verwaltet. Je nach Komplexität des verwendeten Datensatzes ist dies eine valide Methode, jedoch wächst die Größe eines Dokuments mit jeder Tabelle an und eine Abfrage auf Informationen, welche tief verschachtelt sind, kann nicht effektiv erfolgen. [29]

Yassine et al. [64] verdeutlicht mit der Arbeit, wie die Modellierung der Daten in einer dokumentenorientierten Datenbank die Performance beeinflussen kann. Dazu wurden die Daten auf drei Arten in der MongoDB hinterlegt. Bei der Ersten wurde ein großes Dokument erstellt, welches alle Tabellen über Verschachtelung enthält. Bei der Zweiten wurde eine Mischung aus Verschachtelung und Referenz getroffen, welche nicht willkürlich erfolgte, sondern angepasst an die Daten. Bei dem Dritten wurden die Daten in der dokumentenorientierten Datenbank über Referenzen verbunden und somit die Struktur der relationalen Datenbank simuliert. [64]
Die Ergebnisse zeigen, dass eine dokumentenorientierte Datenbank mit Referenzen schlechter performt als eine relationale, welches sich darin begründet, dass für dieses Szenario relationale Datenbanken gedacht sind, nicht aber dokumentenorientierte. Zu große Dokumente können je nach Abfrage problematisch werden, wenn in tiefen Verschachtelungen Werte abgeglichen werden. Im Schnitt performt die Mischung von Verschachtelung und Referenz am besten, was den Erwartungen entspricht. [29]
Hier wurde ein Konzept verwendet, bei dem verschiedene Kollektionen gebildet wurden. Theoretisch ist es in dokumentenorientierten Datenbanken möglich, verschiedene Dokumente in einer Kollektion zu verwahren. Welche Variante zu präferieren ist, hängt vom individuellen Fall ab. Entstehen enorm große Kollektionen, so kann es sinnvoller sein, diese aufzuteilen. Sind es nur vereinzelt Dokumente, so kann es zu bevorzugen sein, dass diese in der gleichen Kollektion verwahrt werden.

2.5.5 Vergleich der Ansätze

Stark abstrahiert betrachtet, folgen alle Ansätze dem folgenden Schema, wobei Schritt 6 optional ist.

1. Erfassung der Metadaten aus der relationalen Datenbank
2. Analyse der Beziehungen zwischen den Tabellen

3. Anpassung der Struktur an die Zieldatenbank
4. Verifizierung der vorgeschlagenen Struktur durch den Anwender
5. Einlesen der neu strukturierten Daten in die Zieldatenbank
6. Evaluierung der Performance und/oder des Datensatzes auf Korrektheit in der neuen Datenbank

Bei allen Arbeiten wird nach einem ähnlichen Schema verfahren, um die Daten umzuwandeln. Zunächst werden aus der Quelldatenbank die relevanten Metadaten zu den Tabellen extrahiert. Anschließend werden diese verwendet, um die Beziehungen zwischen den Tabellen aufzulösen und die Daten dem Modell der Zieldatenbank entsprechend anzupassen. Eine abschließende Bewertung der vorgeschlagenen Struktur wird nicht von allen Ansätzen berücksichtigt bzw. ist nicht erkennbar, ob dies vorgesehen ist. Dabei handelt es sich jedoch um einen sinnvollen Schritt, da der Anwender andernfalls die vorgeschlagene Struktur akzeptieren muss, auch wenn diese noch optimiert werden könnte. Ist die neue Struktur festgelegt, können die Daten entsprechend in die Zieldatenbank eingelesen werden. Abgeschlossen werden kann der Prozess mit einer Evaluierung. Diese überprüft exemplarisch, ob bei Abfragen auf beide Datenbanken das gleiche Ergebnis erzielt wurde und somit die Funktion des Datensatzes durch die Migration nicht verändert wurde.

Die Tabelle 2.4 gibt eine Zusammenfassung der wichtigsten Eckdaten des Migrationsprozesses für alle diskutierten Arbeiten. Nicht alle Ansätze sind so weit fortgeschritten, dass ein Prototyp entstanden ist oder eine Evaluierung durchgeführt wurde. Grundsätzlich wurden immer die Daten migriert, sowie dies an einem Beispieldatensatz dies veranschaulicht. Dabei schwankt die Komplexität des Datensatzes stark. Kleinere Datensätze geben keinen Einblick darüber, wie performant der Ansatz bei komplexeren Beispielen ist. Besonders lässt sich dann nicht erkennen, ob das Konzept bei komplexen Beispielen, die Struktur gut an die Gegebenheiten der Datenbank anpasst. Funktionalitäten wurden nur bei wenigen Arbeiten berücksichtigt. Dazu zählen in diesem Fall die Indizes bei Rocha et al. und die Abfragen bei Liyanaarachchi et al. und Zhao et al. Indizes wurden von Unal et al. zwar verwendet, aber diese wurden nicht auf Basis der relationalen Datenbank entwickelt, sondern aus Performance Gründen beim Einlesen in die Zieldatenbank. Neo4j rät beim Importieren von Datensätzen an, diese mit Indexen und Unique Constraints anzulegen, um bei abgleichenden Aufrufen performanter auf die Daten zugreifen zu können [44]. Die meisten Arbeiten haben eine Evaluierung durchgeführt, um zu überprüfen, ob die Daten korrekt in die Zieldatenbank übertragen wurden und Abfragen weiterhin

das gewünschte Ergebnis erzielen. Dies kann nicht widerspiegeln, ob der komplette Datensatz korrekt übernommen wurde. Ein Konzept für eine Verifizierung der Daten nach der Migration bietet z.B. Goyal et al. [18]. Dies stellt jedoch eine eigene Problematik bei Migrationen. Zur Veranschaulichung der verwendeten Algorithmen wurde zumeist von den Autoren der Pseudocode gegeben oder gar ein Prototyp vorgestellt, mit dem exemplarisch das Verfahren in der Praxis erprobt wurde. Der eigene Ansatz für die Migration wird in den folgenden Kapiteln genauer erläutert.

Tabelle 2.4: Vergleich der Ansätze im Bezug auf die erfüllten Eigenschaften

Autor	Daten	Datensatzgröße	Funktionalitäten	Evaluation	Pseudocode	Prototyp	
Schlüssel-Wert-Datenbanken							
El Alami et al. [14]	x				x		
Graphdatenbanken							
Orel et al. [45]	x	7		x	x	x*	
Unal et al. [61]	x	5	x**	x		x*	
Virgilio et al. [11]	x	5		x	x	x	
Singh et al. [55]	x	24		x	x	x*	
Spaltenfamilien Datenbanken							
Li et al. [32]	x	3					
Zhao et al. [66]	x	3	x		x		
Lee et al. [31]	x	11/34		x		x*	
Dokumentenorientierte Datenbanken							
El Alami et al. [13]	x						
Hamouda et al. [22]	x	4				x*	
Karnitis et al. [25]	x	9			x	x	
Kuszera et al. [30]	x	7		x	x		
Liang et al. [33]	x	8					
Liyanaarachchi et al. [34]	x	6	x	x	x	x	
Rocha et al. [54]	x	2/8	x	x	x	x	
Yassine et al. [64]	x	5		x		x*	
Zhao et al. [67]	x			x	x	x*	
Baensch	x	16	x	x		x	

**Kein Verweis auf einen Prototypen. Dem Paper ist jedoch zu entnehmen, dass mit einer Art Prototyp gearbeitet und Ergebnisse erzielt wurden.*
***Indizes werden nicht direkt migriert, sondern im Migrationsprozess hinzugefügt, um die Performance im Zielsystem zu verbessern.*

3 Migration von relationalen zu dokumentenorientierten Datenbanken

Bei einer Migration von relationalen zu dokumentenorientierten Datenbanken, müssen verschiedene Aspekte berücksichtigt werden. Dazu wird zunächst eine Methodik entwickelt, mittels derer ein grober, schematischer Ablauf der Datenbankmigration gegeben wird (vgl. Kapitel 3.1), unter Berücksichtigung der in Kapitel 2.5 vorgestellten Verfahren. Im Anschluss wird in Kapitel 3.2 ein eigenes Konzept für die Migration von relationalen zu dokumentenorientierten Datenbanken gegeben. Die Auswertung der Migrierbarkeit von Funktionalitäten wird beispielhaft anhand einer konkreten Quell- bzw. Zieldatenbank in Kapitel 3.3 diskutiert.

3.1 Methodik der Konzeptentwicklung

Wie Kapitel 2.5 entnommen werden kann, gibt es unterschiedliche Herangehensweisen für die verschiedenen NoSQL Datenbanken. Diese befassen sich damit eine an das NoSQL Datenbankmodell angepasste Struktur zu entwickeln. Um basierend auf diesen Informationen einen eigenen Ansatz für die Migration zu entwickeln, können diese analysiert werden. Dazu ergeben sich drei Stufen, in denen die Zieldatenbank ansteigend konkretisiert wird. Die ersten Schritte einer Datenbankmigration erfordern keine Spezifizierung des NoSQL Datenbankmodells. Im Weiteren Verlauf muss das Modell festgelegt werden, bis schlussendlich beispielhaft an einer spezifischen Datenbank die letzten Schritte der Migration durchgeführt werden.

3.1.1 Migration von relational zu NoSQL

Bei der Migration von relationalen zu NoSQL Datenbanken muss, unabhängig von der Kategorie der NoSQL Datenbank, die relationale Struktur aufgelöst werden. D.h. es ist

eine Umstrukturierung der Tabellen und Beziehungen in ein anderes Schema erforderlich. Für diesen Prozess können die wichtigen Informationen zur Struktur über Tabellen und Schlüssel erhalten werden. Die Tabellen enthalten mit ihren Spalten die Informationen zu der Strukturierung innerhalb der Tabelle und geben Aufschluss auf deren Größe bzw. Komplexität. Die Primärschlüssel einer Tabelle können als Fremdschlüssel in einer anderen hinterlegt sein und somit die Beziehungen zwischen den Tabellen ermittelt werden. Problematisch wird es mit den Beziehungen, wenn keine Fremdschlüssel definiert werden und nur ein Spaltenbezeichner in zwei verschiedenen Tabellen gleich ist. Diese verstecken Beziehungen lassen sich nur schwerlich auffinden. Wird im Bezeichner mit einer Struktur der Art: „name_id“ gearbeitet, könnten diese dennoch identifiziert werden. Das ist jedoch nur möglich, wenn dieses Schema ausschließlich bei Beziehungen und kein anderes Schema verwendet wird. Andernfalls können diese Beziehungen nicht automatisiert gefunden werden.
Das Erfassen der Abhängigkeiten zwischen Tabellen kann vom Zieldatenbanktyp unabhängig erfolgen. Alternativ kann bereits bei der Erhaltung dieser Informationen auf das Endergebnis für die Zieldatenbank hingearbeitet werden. Des Weiteren kann eine Unterscheidung erfolgen, für welche verschiedenen NoSQL Datenbanken das Konzept verwendbar sein soll. Unabhängigere Konzepte verwenden Zwischenmodelle, in denen zu den Daten aus der relationalen Datenbank möglichst viele Informationen hinterlegt werden, damit bei der Umstrukturierung zum Zieldatenbanktypen alle Informationen vorhanden sind. Alternativ kann der Ansatz nur für ein NoSQL Modell zugänglich sein und somit direkt nach der Ermittlung der Struktur des relationalen Modells mit der Migration zum Zieldatenbanktypen begonnen werden.

Insgesamt teilen sich die Verfahren in zwei Gruppen. Die erste bildet jene, welche direkt aus dem relationalen Modell die Tabellen und Beziehungsstrukturen auslesen und in einem Format zwischenspeichern, welches für die Weiterverarbeitung zum Schema der Zieldatenbank geeignet ist. Dabei werden nur die essenziellen Informationen verwendet, nämliche die Tabellen und Beziehungen. Diese Methode praktizieren die folgenden Ansätze (vgl. Kapitel 2.5).

- Zu jeder Tabelle werden relevante Daten hinterlegt, so auch die Kardinalität der Beziehungen zu den Tabellen, sowie den Informationen zu den Nachbarn. [13]
- Linked-Lists dienen zur Speicherung der Primär-/Fremdschlüssel und stellen somit die Verkettung bzw. die Beziehungen zwischen den Tabellen dar. [31]
- Regeln legen den Ablauf der Strukturumwandlung fest. [32]

- Mittels physischer bzw. logischer Level wird die Struktur der relationalen Datenbank nachgebildet. [25]
- Jede Tabelle bildet einen DAG, welcher ausgehend von dieser Tabelle alle Beziehungsketten darstellt. Werden die DAG aufgelöst, erhält man die komplette Struktur der relationalen Datenbank. [30]
- Es wird ein DAG gebildet. Durch den Algorithmus werden alle Knoten des DAG durchlaufen und eine Liste mit den Fremdschlüsseln ausgeben, welche die Struktur der relationalen Datenbank aufzeigen. [67]
- Die Tabellenstruktur wird in einer JSON-Datei hinterlegt, davon ausgehend wird mittels Modulen die Kardinalität der Beziehungen ermittelt. [34]

Die zweite Gruppe bildet, basierend auf möglichst vielen Informationen aus dem relationalen Modell, ein Zwischenmodell. Dies ermöglicht eine Weiterverarbeitung der Informationen zu verschiedenen NoSQL Kategorien, da alle erfassbaren Informationen gebündelt zwischengespeichert werden. Dementsprechend bieten solche Verfahren den Vorteil, dass eine größere Flexibilität geschaffen wird, im Bezug darauf, welche Zieldatenbanken unterstützt werden. Beispiele für Verfahren, welche sich an einem Zwischenmodell orientieren, sind die folgenden (vgl. Kapitel 2.5).

- In einem Objekt, welches eine Entität aus der relationalen Datenbank darstellt, werden die Beziehungen hinterlegt. [33]
- Basierend auf dem ER-Modell der relationalen Datenbank, werden die DODS mit allen relevanten Informationen gebildet. [22]

Die Objekte bieten neben den Basisinformationen noch weitere, wie z.B. zu Abfragen an. In den DODS werden die Basisinformationen mit zusätzlich den Attributen der Tabelle hinterlegt und setzten keine bestimmte NoSQL Kategorie voraus. In definierten Strategien wird erst das Zieldatenbankmodell festgelegt und die Struktur daran angepasst.

3.1.2 Migration von relational zu dokumentenorientiert

Liegen die Daten zu den Tabellen und deren Beziehungen vor, so kann eine Umstrukturierung erfolgen, welche an die Zieldatenbank angepasst sein sollte. Da je nach NoSQL Kategorie die Daten intern anders verwaltet und dargestellt werden, ist eine sinnvolle

Anpassung der Struktur an die neue Datenbank erforderlich. Um die Varianz zu verdeutlichen, wird kurz ein Einblick zu alle NoSQL Kategorien gegeben. Schlüssel-Wert-Datenbanken speichern alle Daten in Schlüssel-Wert-Paaren und erfordern somit eine Bildung von angepassten Schlüsseln zu den Werten, um effizient auf die Daten zugreifen zu können. Graphdatenbanken hingegen arbeiten mit Graphen und unterteilen die Daten aus der relationalen Datenbank in Knoten bzw. Kanten und Attributen zu diesen. Die Beziehungen werden in diesem Fall über Kanten dargestellt. Spaltenfamilien Datenbanken verschachteln Informationen aus verschiedenen Tabellen in einer Tabelle, um schneller auf zusammengehörige Daten aus verschiedenen Tabellen zugreifen zu können. Ähnlich ist das Verfahren bei dokumentenorientierten Datenbanken, diese hinterlegen die Daten in Dokumenten, welche ebenfalls die Möglichkeit bieten Informationen aus verschiedenen Tabellen zu verschachteln oder über eine Referenz auf andere Dokumente verweisen.

Bei der Bildung der neuen Struktur ist eine Anpassung auf ein spezielles NoSQL Datenbankmodell notwendig. Dies ist für die vier Ansätze der Graphdatenbanken in Kapitel 2.5 erfolgt und somit nicht für andere NoSQL Modelle geeignet. Da der Fokus auf den dokumentenorientierten Datenbanken liegt, werden diese nicht berücksichtigt. Die folgenden zwei Ansätze (vgl. Kapitel 2.5) sind durch die Vorgabe der Struktur nur für dokumentenorientierte Datenbanken ausgelegt.

- Jede Tabelle wird in einem Dokument abgebildet. [54]
- Die neue Struktur der Dokumente wird durch die Autoren festgelegt. [64]

Unabhängig von diesen zwei Ansätzen muss für die Migration entschieden werden, welche Tabellen über eine Verschachtelung in ein Dokument zusammengefasst werden können und welche Tabellen mittels Referenz in eigenständigen Dokumenten verwaltet und über dieses Mittel verknüpft werden. Die Entscheidung kann von verschiedenen Faktoren abhängig sein. Ein Faktor sind die unterschiedlichen Beziehungsarten zwischen den Tabellen (vgl. Kapitel 2.1.1). Tendenziell ist eine Verschachtelung bei 1:1-Beziehungen sinnvoll, bei komplexeren wie der n:m-Beziehung eine Referenz und bei 1:n-Beziehungen eine Einzelfallentscheidung. Pauschal lässt sich damit nicht eine Regel definieren, denn bei der Entscheidung ist zusätzlich noch zu berücksichtigen, wie komplex die Tabelle ist, über die die Referenz oder die Verschachtelung erfolgen soll und wie komplex bereits das Dokument ist, zudem diese Informationen hinzugefügt werden sollen. Ebenso ist die Anzahl und Komplexität der ausgehenden Beziehungen von dieser Tabelle relevant. Um der Problematik entgegenzuwirken kann ein Threshold definiert werden, welcher festlegt ab

welcher Komplexität von der Verschachtelung abzuweichen ist und eine Referenz erfolgen soll.

Ein anderer Faktor können die Abfragen auf den Daten sein. Werden häufig Informationen aus verschiedenen Tabellen benötigt, kann mittels Verschachtelung eine Verbesserung der Performance erzielt werden. Dabei spielt die Komplexität ein untergeordnete Rolle, vorausgesetzt es sind keine Abfragen auf einfach oder mehrfach verschachtelte Daten nötig. In den Fällen ist die Referenz sinnvoll, wenn andernfalls Abfragen auf tief verschachtelte Informationen erfolgen.

3.1.3 Datenbank individuelle Anpassungen bei der Migration

Nicht alle Aspekte einer Datenbankmigration können allgemein und ohne konkreten Anwendungsfall analysiert werden. Darunter fallen die Funktionalitäten und die Einschätzung derer im Bezug auf die Möglichkeit diese bei einer Migration zu berücksichtigen und in der Zieldatenbank einzubinden. Funktionalitäten, welche basierend auf einer konkreten Quelldatenbank untersucht wurden, können ebenfalls in weiteren Quelldatenbanken, welche nicht berücksichtigt wurden, vorhanden und somit prinzipiell für diese ebenfalls migrierbar sein. Beispielsweise ist die Varianz der unterstützen Datentypen breit, führt jedoch nicht zwingend dazu, dass eine Migration aufgrund dessen nicht erfolgen kann. Ebenso gehören Indizes zu den typischen Funktionalitäten einer Datenbank und werden intern in dieser auf verschiedene Weise realisiert. So ist eine Übernahme der Indizes möglich, diese können jedoch intern anders verwaltet werden als zuvor, erfüllt aber den gleichen Zweck in der Datenbank, d.h. eine Verbesserung der Performance bei Abfragen auf diesen. Eine detailliertere, exemplarische Analyse der Funktionalitäten ist dem Kapitel 3.3 zu entnehmen.

3.2 Konzept der eigenen Vorgehensweise bei der Migration

Nachdem im vorherigen Kapitel methodisch der Migrationsvorgang analysiert wurde, wird in diesem der eigene Ansatz vorgestellt. Dazu werden zunächst die bestehenden Konzepte ausgewertet. Basierend darauf wird ein Verfahren entwickelt, welches die Beziehungen aus der relationalen Datenbank auswertet und die wichtigen Informationen dazu speichert. Basierend darauf kann die neue Struktur der Daten definiert werden. Ebenso wird festgelegt, in welchem Maß die Verschachtelung der Referenz zu bevorzugen

ist. Abgeschlossen wird das Kapitel mit einem Einblick in die Migration der Funktionalitäten, welche im Folgekapitel (vgl. Kapitel 3.3) ausführlich diskutiert werden.

3.2.1 Auswertung der bestehenden Ansätze

Jene Vorgehen, welche eine feste Struktur vorgeben oder Zwischenmodelle verwenden, sind nicht praktikabel. Da eine automatische Erstellung einer neuen Struktur für die Zieldatenbank gefunden werden soll, sind die Verfahren, welche eine Festlegung der Struktur durch einen Menschen erfordern nicht nutzbar. Des Weiteren ist die pauschale Verwendung von Referenzen und somit eine Nachbildung des relationalen Modells im dokumentenorientierten nicht effizient (vgl. [64]) und schöpft das Potential der NoSQL Datenbank nicht aus. Ein Zwischenmodell ist für eine Migration, welche nur für ein bestimmtes Modell von NoSQL Datenbanken ausgelegt ist, nicht notwendig, da keine Erweiterbarkeit mit anderen Kategorien gegeben sein muss.

Die Methoden, welche unabhängig vom NoSQL Datenbankmodell sind, unterscheiden sich insbesondere in der Komplexität, wie die Beziehungen ermittelt und hinterlegt werden. Methoden, wie das ER-Modell oder verschiedene Level, welche erst im späteren Verlauf des Verfahrens die Beziehungen erfassen, besitzen den Nachteil, dass diese Strukturen zunächst gebildet werden müssen und dies im Vergleich zu anderen Verfahren zeitaufwändig ist. Das Konzept mit dem ER-Modell wäre nur eine Option, falls das Modell bereits verfügbar ist und nicht erst gebildet werden muss. Alternativ kann dies auch direkter, wie beispielsweise über Linked-List oder Graphen erfolgen. Bei allen drei Verfahren wird von jeder Tabelle ausgehen ermittelt, mit welchen anderen Tabellen diese eine Beziehung besitzt und speichert dies. Basierend auf diesen Daten kann eine neue Struktur gebildet werden.

Da der eigene Ansatz nur für dokumentenorientierte Datenbanken angepasst sein soll, bietet sich ein Verfahren an, welches eine ähnliche, simpel gehaltene Struktur für die Speicherung der Beziehungen verwendet, wie die Linked-List.

3.2.2 Auswertung der Beziehungen im relationalen Modell

Für den eigenen Ansatz werden Objekte mit den relevanten Informationen zu der relationalen Datenbank erstellt. Für jede Tabelle wird ein Objekt angelegt und diese in

einer Liste gesammelt, welche dann von der dokumentenorientierten Datenbank ausgewertet werden können. Der Prozess dafür kann in folgende Schritte grob zusammengefasst werden:

1. Verbindung zur relationalen Quelldatenbank herstellen
2. Erhalte eine Liste mit allen Tabellennamen in der Datenbank und iteriere einmal über diese
3. Hinterlege die relevanten Informationen (Tabellenname, Spaltennamen, Primärschlüssel) zu der aktuellen Tabelle in einem Tabellenobjekt
4. Erfasse die Fremdschlüssel und hinterlege die Daten dazu in einem Fremdschlüsselobjekt (Fremd-/Primärschlüssel, Tabellennamen, Beziehungstyp)
5. Suche in anderen Tabellen nach Beziehungen, welche nicht durch Fremdschlüssel dargestellt werden und aktualisiere das Fremdschlüsselobjekt
6. Ergänze das Tabellenobjekt mit dem Fremdschlüsselobjekt
7. Speichere das Tabellenobjekt in einer Liste
8. Ist Schritt 2 abgeschlossen, iteriere über die Liste der Tabellenobjekte
9. Ergänze die fehlenden Beziehungen auf der n-Seite, welche nicht als Fremdschlüssel in der Tabelle vermerkt sind, in dem entsprechenden Tabellenobjekt

Um Beziehungen umfassend auswerten zu können sind Informationen zu den Primär- und Fremdschlüsseln notwendig, sowie die Namen der in Beziehung stehenden Tabellen. Ebenso ist die Kardinalität der Beziehung essenziell. Dies kann in einem gesonderten Objekt hinterlegt und den allgemeinen Informationen zur Tabelle hinzugefügt werden. Die Kardinalität der Beziehung wird nur zwischen „1“ und „n“ unterschieden. Dazu muss jedoch die Gegenrichtung der Beziehung ebenfalls ermittelt und dem jeweiligen Objekt hinzugefügt werden. Sind für eine Tabelle keine Fremdschlüssel hinterlegt, so sind keine Beziehungen vorhanden.

Einfach zu erfassen sind die Beziehungen, welche über Fremdschlüssel realisiert werden und können direkt aus den Informationen zu einer Tabelle ausgelesen werden. Sind Beziehungen ohne Fremdschlüssel vorhanden, so darf kein Name eines Primärschlüssels als

weiterer Spaltennamen ohne Beziehung verwendet werden, damit sie identifiziert werden können. Dieses Schema muss ausschließlich für Beziehungen verwendet werden, andernfalls werden Beziehungen vermerkt, welche keine sind oder können nicht als solche erkannt werden. Bei der Iteration über die Tabellen werden zunächst die Beziehungen über Fremdschlüssel zu den jeweiligen Tabellen vermerkt. Anschließend werden die Primärschlüssel mit den restlichen Spalten abgeglichen und bei einer Übereinstimmung eine Beziehung der Tabelle hinterlegt und die Kardinalität ermittelt.

Zu beachten ist, dass sich ohne ER-Modell nur der aktuelle Stand der Datenbank erfassen lässt, im Bezug darauf, um welchen Beziehungstypen es sich zwischen den Tabellen handelt. Dies muss aber nicht der generellen Planung entsprechen, so kann beispielsweise die manuelle Auswertung ergeben, dass eine 1:1-Beziehung vorliegt, dies jedoch prinzipiell eine 1:n-Beziehung ist, welche zum Zeitpunkt der Migration nur als eine 1:1-Beziehung vorliegt. Das Problem kann nur durch den Anwender gelöst werden, indem eine Methode zur Korrektur angeboten wird. Dies kann allgemein über die Schlussbewertung erfolgen, denn wurde eine Tabelle durch das Verfahren als eine 1:1-Beziehung erkannt und basierend auf der Annahme verschachtelt, kann der Anwender auswählen, dass diese Tabelle stattdessen referenziert werden soll.

Eine Optimierung kann bei der Entscheidung über Verschachtelung oder Referenz erfolgen. Um zu evaluieren, welche die bessere Option ist, können die Abfragen dienen. Müssten diese häufig über verschachtelte Dokumente laufen, so wäre ein vermehrter Einsatz von Referenzen möglich. Des Weiteren kann dem Anwender zur Auswahl gestellt werden, eine Struktur zu bilden, welche vermehrt referenziert (weniger Speicherplatzbedarf) oder verschachtelt (gute Performance, wenn mit einem Aufruf alle Daten erhalten werden sollen). Soll die Migration Platz-effizient sein, so werden nur 1:1-Beziehungen und 1:n-Beziehungen von geringer Komplexität verschachtelt, der Rest referenziert. Wird eine bessere Performance ausgewählt, so werden typische Abfragen auf den Datensatz analysiert und die Verschachtelung stark bevorzugt und nur bei Dokumenten die eine zu große Komplexität aufweisen würden, referenziert. Wird keine der beiden Optionen ausgewählt, wird statisch entschieden, welche Option erfolgen soll, indem bestimmte Thresholds definiert werden, ab welcher Komplexität vom Embedding zur Referenz gewechselt werden soll.
Der Basisansatz verschachtelt grundsätzlich 1:1-Beziehungen und 1:n-Beziehungen, wenn die Tabelle, in die verschachtelt werden soll, keine n-Beziehung aufweist. Dies hätte zur Folge, dass mehrere Dokumente in mehreren Tabellen/Dokumente verschachtelt werden und dies zu massiven Redundanzen führt und damit den Speicherplatz um ein vielfaches

erhöhen würde. In diesen Fällen werden Referenzen angelegt, um dennoch einen schnelleren Zugriff auf die Daten zu gewähren, aber nicht große Mengen an Redundanzen zu produzieren. n:m-Beziehungen benötigen keine gesonderte Betrachtung, da diese über zwei 1:n-Beziehungen realisiert werden und die Verschachtelung ausgeschlossen wird, sobald mehrere n-Beziehungen zu der Tabelle, in die verschachtelt werden soll, vorhanden sind. Abschließend wird die nach diesem Verfahren ermittelte Struktur dem Anwender vorgeschlagen und dieser kann gegebenenfalls Änderungen vornehmen und manuell zwischen Verschachtelung, Referenz oder überspringen entscheiden. Bei überspringen werden die Tabelleninhalte unverändert als Dokumente eingelesen.
Grundsätzlich werden für jede Tabelle, welche die Basis eines Dokuments bildet, eine eigene Kollektion erstellt. Dies basiert auf dem Grund, dass andernfalls bei Abfragen auf Primär-/Fremdschlüssel mehr Dokumente erhalten werden, als ursprünglich bei der relationalen Datenbank. Wird dies nicht mit unterschiedlichen Kollektionen gehandhabt, können durch eine Abfrage Dokumente erhalten werden, welche nicht gewünscht sind und die komplette Abfragelogik verändern.

Nach der Migration kann eine Überprüfung auf Korrektheit der Daten erfolgen. Diese wird durch eine Evaluierung von definierten Testabfragen auf der Quell- und Zieldatenbank durchgeführt. Damit kann nicht die Korrektheit des kompletten Datensatzes gesichert werden, kann aber exemplarisch testen, ob der Datensatz nach der Migration die erwarteten Ergebnisse liefert. Eine komplexe Verifizierung des Datensatzes stellt eine eigne Problematik bei der Migration von Datenbanken und kann im Rahmen dieser Masterthesis nicht berücksichtigt werden.

3.2.3 Migration der Funktionalitäten

Berücksichtigt wurden bis jetzt nur die Informationen, um die Daten zu migrieren. Eine Migration von Funktionalitäten kann im Anschluss erfolgen, da diese zumeist durch die Änderung der Struktur an die neue angepasst werden müssen. Hierbei kann der Anwender aus einem Pool von unterstützen Funktionalitäten jene auswählen, welche bei der Migration übertragen werden sollen. Die zur Verfügung stehenden Funktionalitäten sind von der Auswahl der unterstützten Quelldatenbanken abhängig, da ausgehend von diesen ein Verfahren für die Migration entwickelt wird. Das schließt nicht aus, dass weitere Quelldatenbanken ähnliche Funktionalitäten aufweisen und somit theoretisch ebenfalls migrierbar wären, dafür kann jedoch keine Gewährleistung gegeben werden. Das nächste

Kapitel (vgl. Kapitel 3.3) befasst sich mit der generellen Möglichkeit Funktionalitäten, anhand einer ausgewählten Quell- und Zieldatenbank, zu migrieren.

Bei den unterstützen Funktionalitäten handelt es sich um jene, die automatisiert migriert werden können. D.h. grundsätzlich ist kein Einwirken des Anwenders notwendig. Sind jedoch Teile der zu migrierenden Funktionalität nicht kompatibel, muss der Anwender entscheiden, ob diese in angepasster Version migriert oder übersprungen werden sollen. Alle weiteren Funktionalitäten, die der Anwender eventuell benötigt und nicht unterstützt werden, müssen eigenverantwortlich im Anschluss realisiert werden.

3.3 Auswertung der Funktionalitäten für die Migration

Eine Auswertung der Funktionalitäten kann nur für exemplarisch ausgesuchte Datenbanken erfolgen, da für diese keine Standards vorhanden sind. Ebenso variieren die Funktionalitäten stark zwischen verschiedenen Datenbanken, sodass eine Analyse dazu immer im Einzelfall erfolgen muss. Ausgewählt wurden Open Source Vertreter des jeweiligen Datenbankmodells. Die MongoDB ist eine populäre dokumentenorientierte Datenbank und belegt im Ranking von DB-Engines[1] für ihr Modell den ersten Platz. PostgreSQL belegt im Ranking[2] lediglich Platz vier und befindet sich somit hinter MySQL. Mit PostgreSQL wurden jedoch bereits im Grundprojekt Erfahrungen gesammelt, sodass sie für die Masterthesis ebenfalls ausgewählt wurde. Die Migration wird von einer relationalen zu einer dokumentenorientierten Datenbank durchgeführt. Deswegen ist nur eine Auswertung aller Funktionalitäten von PostgreSQL nötigt, sowie ein Mapping auf die entsprechenden MongoDB Funktionalitäten. Analysiert wurden dazu die Dokumentation von PostgreSQL [19] und MongoDB [42] und die folgenden Informationen beziehen sich auf diese Quellen.

3.3.1 Datentypen

Die Varianz an Datentypen ist bei PostgreSQL und MongoDB groß. PostgreSQL bietet insgesamt 42 an, MongoDB hingegen nur 18, da sie intern mit dem Binary JSON (BSON) Standard arbeitet. Unter den 18 Datentypen sind einige die hauptsächlich für internen Zwecken verwendet werden. Bei der Migration ist zu beachten, dass eine Vielzahl der

[1] https://db-engines.com/de/ranking/document+store
[2] https://db-engines.com/de/ranking/relational+dbms

Tabelle 3.1: Numerische Datentypen

PostgreSQL	MongoDB
smallint	
integer	integer
bigint	long
decimal	
numeric	
real	
double precision	double
	decimal128
smallserial	
serial	
bigserial	

Datentypen nicht übernommen werden können und lediglich als String migriert werden. Dementsprechend sind die Funktionen, welche auf diese Datentypen angeboten wurden, nicht übertragbar und müssten manuell nachgebildet werden.

Numerische Datentypen

PostgreSQL bietet Datentypen speziell zur Serialisierung an, d.h. diese Datentypen werden für die automatische Erstellung von einzigartigen Identifikatoren verwendet. MongoDB bietet dazu die ObjectID, welche ebenfalls automatisch erstellt werden kann und somit ein äquivalent zu diesem Teil bildet. Die MongoDB differenziert bei Ganzzahlen nur zwischen integer und long, dementsprechend kann ein smallint dargestellt werden. Bei PostgreSQL sind numeric und decimal äquivalent und erlauben es benutzerdefiniert die Präzision vor und nach dem Dezimalpunkt zu bestimmen. Bei real und double handelt es sich um Fließkomma Datentypen, welche durch die MongoDB mit deciaml128 erweitert werden. Bei der Genauigkeit kann die MongoDB nur zwischen double und decimal128 entscheiden. Da real für 6 Zeichen und double für 15 Zeichen Genauigkeit bietet, kann real mit double dargestellt werden. Für numeric und decimal ist der Datentyp decimal128 zu verwenden. Die Datentypen zur Serialisierung können ebenfalls mit integer und long abgebildet werden. Eine Übersicht gibt die Tabelle 3.1.

Tabelle 3.2: Zeichen Datentypen

PostgreSQL	MongoDB
character(n)	
character varying(n)	
text	
	String

Tabelle 3.3: Datum/Zeit Datentypen

PostgreSQL	MongoDB
date	date
time [(p)] [without time zone]	
time [(p)] with time zone	
timestamp [(p)] [without time zone]	
timestamp [(p)] with time zone	
	timestamp
interval [fields]	

Zeichen

Die MongoDB arbeitet nur mit Strings, um Zeichen darzustellen (vgl. Tabelle 3.2). Bei PostgreSQL wird feiner unterschieden, jedoch sind diese Datentypen allesamt nur als String darstellbar in der MongoDB.

Datum und Zeit

Die MongoDB besitzt für interne Zwecke den Datentyp timestamp, welcher die Zeit im UTC-Format hinterlegt. Dies entspricht nicht direkt den Datentypen der PostgreSQL, da diese unter anderem Zeitzonen erlauben und ebenfalls nur die Speicherung der Zeit ohne Datum ermöglichen. Das Datum wird hingegen von beiden Datenbanken unterstützt und kann mit Funktionen auf diesen übernommen werden (vgl. Tabelle 3.3). Die weiteren Datentypen von PostgreSQL werden entweder als Datum in der MongoDB abgebildet oder müssen als String übernommen und dann weiterverarbeitet werden.

Tabelle 3.4: Unterstützte Datentypen

PostgreSQL	MongoDB
json	
jsonb	
Array	Array
boolean	Boolean

Unterstützte Datentypen

Sowohl PostgreSQL, als auch MongoDB bieten den Boolean mit den Wahrheitswerten true/false an, sowie die Verwendung von Arrays (vgl. Tabelle 3.4). PostgreSQL unterstützt die Datenformate json, bzw. jsonb, welche generell eine Darstellung von Daten im JSON-Format ermöglichen. Da die MongoDB mit JSON arbeitet, bzw. diese im BSON-Format hinterlegt und ebenfalls Dokumente in Dokumenten speichern kann, ist grundsätzliche eine Übernahme dieser Daten realisierbar.

Nicht unterstütze Datentypen

Da die PostgreSQL deutlich mehr Datentypen besitzt, wird eine Reihe von diesen nicht unterstützt und dadurch sind ebenfalls mögliche Funktionen auf diese Datentypen per se nicht verfügbar. Dennoch können die Informationen unter alternativen Datentypen gespeichert werden. Ebenso bietet der BSON Standard Datentypen, welche nicht in der PostgreSQL verwendet werden. Dies ist jedoch nicht erheblich, da nur von relational zu dokumentenorientiert migriert wird. Wird bei einer Migration einer dieser Datentypen verwendet, so wird dieser als ein String hinterlegt. Werden diese Daten anschließend von der MongoDB ausgelesen, ist eine Weiterverarbeitung dieser notwendig, da der Datentyp geändert wurde. Dies ist manuell durch den Anwender zu erfolgen. Eine Übersicht dazu gibt die Tabelle 3.5.

3.3.2 Abfragen

Grundsätzlich können SQL-Abfragen mit dem MongoDB Connector for BI [39] in den MongoDB-Dialekt übersetzt werden. Generell dient das Tool dazu, zwischen der MongoDB und Business Intelligence Tools zu vermitteln, da diese nicht immer die Schemafrei-

Tabelle 3.5: Nicht unterstütze Datentypen

PostgreSQL	MongoDB
Geometrisch	
point	
line	
lseg	
box	
path	
polygon	
circle	
Netzwerk Adressen	
cidr	
inet	
macaddr	
macaddr8	
Textsuche	
tsquery	
tsvector	
Sonstige	
pg_lsn	
txid_snapshot	
money	
bytea	
bit(n)	
bit varying(n)	
uuid	
xml	
MongoDB spezifisch	
	min key
	max key
	JavaScript
	JavaScript (with scope)
	Regular Expression
	Null
	Binary Data
	ObjectId
	Object

heit der MongoDB Daten verarbeiten können. Erweist sich das Tool als praktikabel, um die SQL-Abfragen unkompliziert in den MongoDB-Dialekt zu überführen, so wäre eine Verwendung sinnvoll. Zu beachten ist, dass durch die Veränderung der Datenstruktur, die Abfragen nicht übernommen werden können, falls abgefragte Informationen im neuen Schema in einer verschachtelten Struktur liegen. Ebenso kann durch die Referenz direkter auf Daten zugegriffen werden und die Abfragen müssen dementsprechend angepasst werden.

Die Alternative zu dem BI-Connector kann ein eigener Übersetzer sein. Dazu müsste abgedeckt werden, welche Schlüsselwörter aus der SQL möglich wären und wie mit diesen dann im Bezug auf die MongoDB zu verfahren ist, unter der Berücksichtigung, dass das Schema sich im Vergleich zu der relationalen Datenbank verändert hat. PostgreSQL stellt dazu eine Liste mit allen SQL-Schlüsselwörtern zur Verfügung [49].

Neben den Abfragen kann in beiden Datenbanken eine Textsuche erfolgen. Diese ist z.B. mit LIKE oder regulären Ausdrücken möglich.

3.3.3 Views

Bei Views handelt es sich um Abfragen, welche im Datenbanksystem hinterlegt werden. Diese dienen im relationalen Modell vor allem dazu, Informationen aus verschiedenen Tabellen für häufige Abfragen zu bündeln. Generell unterstützen beide Datenbank diesen Mechanismus, jedoch ist bei der Migration zu beachten, dass eine View überflüssig werden kann, wenn die Informationen bereits in einem Dokument hinterlegt sind. Ist dies nicht der Fall, kann die View übernommen werden. Zu beachten ist dabei, dass sich das Schema geändert hat und die View dementsprechend angepasst werden muss. Ebenso ist für Views, wie Abfragen essenziell, dass die Schlüsselwörter richtig erkannt und in die Sprache der MongoDB übersetzt werden.

3.3.4 Indizes

Beide Datenbanken bieten den Mechanismus der Indizes an, jedoch gibt es bei den Typen des Indizes kein Überschneidungen. Das bedeutet, wenn ein Index übertragen wird, dann wird dieser intern anders arbeiten als zuvor. Werden die Indizes dazu verwendet, die Performance bei Abfragen zu steigern, so können diese dennoch migriert werden, mit dem Hinweis, dass ein anderer Index-Typ verwendet wurde. Beide Datenbanken erlauben

Tabelle 3.6: Index Eigenschaften

Eigenschaft	PostgreSQL	MongoDB
unique	x	x
partial	x	x
sparse		x
TTL		x

es über ein oder mehrere Felder/Spalten den Index anzulegen. Ebenso können Indizes gewisse Eigenschaften aufweisen, wobei MongoDB mehr anbietet (vgl. Tabelle 3.6). Beide Datenbanken können dem Index die Eigenschaft unique oder partial zuweisen. Partielle Indizes können jedoch nicht ausgewertet werden, da diese Abfragen als Parameter verwenden. Werden die Abfragen nicht komplett ausgewertet, kann der partielle Index nicht migriert werden. Alternativ kann ein Index ohne die Eigenschaft auf den Feldern realisiert werden.

3.3.5 Benutzer/Rollen

Zu den administrativen und auch sicherheitsrelevanten Themen bei Datenbanken gehören die Benutzer, denen Rollen mit Berechtigungen zugewiesen werden können. Beide Datenbanken unterstützen dieses Konzept, sodass bei einer Migration diese berücksichtigt werden können. PostgreSQL bezeichnet die Berechtigungen als Privilegien, MongoDB als Aktionen. Zu beachten bei der Vergabe von Rechten ist, dass durch die Verschachtelung unter Umständen auf mehr Informationen zugegriffen werden kann, als bei den Tabellen. Sowohl PostgreSQL, als auch MongoDB weisen ein breites Spektrum an Berechtigungen auf, jedoch ergibt sich nur eine kleine Schnittmenge, welche der Tabelle 3.7 entnommen werden kann. Des Weiteren unterscheiden sich die Bezeichner stark, sodass für die Datenbanken jeweils das äquivalent genannt wird. Werden Benutzer bzw. Rollen migriert, die mehr Berechtigungen aufweisen, als in der Tabelle 3.7 verzeichnet sind, so können diese entweder nur mit dem Teil der Rechte erstellt werden, welcher ein Äquivalent aufweist oder sie werden ausgelassen.

Tabelle 3.7: Privilegien/Aktionen für Rollen

PostgreSQL	MongoDB
SELECT	find
INSERT	insert
UPDATE	update
DELETE	remove
CREATE	createCollection

3.3.6 Weitere Funktionalitäten

Die folgenden Funktionalitäten werden zwar von der PostgreSQL unterstützt, können jedoch nicht in die MongoDB übertragen werden und sind somit nicht migrierbar.

Mittels benutzerdefinierten Funktionen/Prozeduren erlaubt PostgreSQL, unter Verwendung verschiedener Sprachen wie Tcl, Perl, Python oder pgSQL, eigene Funktionen/Prozeduren zu schreiben. Bei der MongoDB können generell Funktionen mit JavaScript erstellt werden, bietet damit aber keine überschneidende Sprache mit PostgreSQL an. Eine automatisierte Übersetzung der Funktionen/Prozeduren in eine andere Programmiersprache wäre Voraussetzung, das stellt jedoch einen zu komplexen Vorgang dar.

Ähnlich verhält es sich mit den Triggern/Event Triggern. Diese werden ebenfalls in den Sprachen wie bei den benutzerdefinierten Funktionen/Prozeduren geschrieben. Des Weiteren ermöglicht die MongoDB das Anlegen von Triggern nur über MongoDB Stitch. Stitch gehört zum Cloud-Service Atlas und ist nicht für lokale Versionen verfügbar ist.

Ein weiteres Konzept ist die Vererbung bei PostgreSQL. Die MongoDB ist schemafrei und daher ist der Mechanismus der Vererbung nicht vorhanden. Da die Tabellen ausgelesen werden ist es irrelevant, ob Teile der Tabelle vererbt oder manuell angelegt wurden.

Rules erlauben in PostgreSQL neben der Ausführung eines Befehls, weitere Kommandos auszuführen. Dieses Konzept wird von der MongoDB nicht unterstützt.

3.4 Zusammenfassung

Basierend auf der vorangegangenen Analyse bezüglich der Funktionalitäten und der Methodik des eigenen Konzepts, werden nur die folgende Komponenten migriert.

- Daten (stark eingeschränkt)
- Indizes (unterschiedliche Konzepte)
- Benutzer und Rollen (stark eingeschränkt)

Die Daten werden entsprechend dem vorgestellten Konzept aus Kapitel 3.2 migriert. Dabei ist zu beachten, dass die MongoDB signifikant weniger Datentypen unterstützt als PostgreSQL. Dadurch kommt es zu veränderten Datentypen. Werden anderweitige Funktionen auf diesen Datentypen ausgeführt oder diese extern bearbeitet, so ist durch den Anwender zu beachten, dass der Datentyp geändert wurde. Der BI-Connector ist nicht direkt ein Tool zum Übersetzen der Abfragen. In der Masterthesis wird auf die Auswertung der Abfragen verzichtete, da diese ein zu komplexes Feld darstellen und gesondert betrachtet werden müsste. Mit den Abfragen hängen ebenfalls die Views zusammen. Sind die Abfragen nicht direkt übersetzbar, können die Views ebenfalls nicht übertragen werden. Bei den Indizes wird nur das Konzept der verbesserten Performance bei Abfragen übernommen. Intern werden diese bei den unterschiedlichen Datenbanken unterschiedlich gehandhabt. Benutzer bzw. Rollen können nur im beschränkten Ausmaß migriert werden, da kaum Übereinstimmungen bei den Rechten vorhanden sind.

Grundsätzlich können diese Mechanismen manuell ergänzt und erweitert werden. Das unterliegt jedoch der kompletten Verantwortung des Anwenders und wird nicht in der Masterthesis berücksichtigt. Migriert werden nur die Konzepte, bei denen dies nahezu vollständig automatisiert erfolgen kann.

4 Migration zwischen dokumentenorientierten Datenbanken

In diesem Kapitel werden die verschiedenen Aspekte einer Migration zwischen dokumentenorientierten Datenbanken betrachtet. Sowohl die Daten (vgl. Kapitel 4.1), als auch die Funktionalitäten (vgl. Kapitel 4.2) werden berücksichtigt. Im Gegensatz zu der Migration von relational zu dokumentenorientiert, muss keine Strukturumwandlung der Daten erfolgen. Dadurch beschränkt sich der Prozess auf den Import der Daten und die Umsetzung der Funktionalitäten. Abgeschlossen wird diese Thematik in Kapitel 4.3, in dem zusammenfassend geschildert wird, welche Aspekte bei einer automatisierten Migration im Rahmen der Masterthesis berücksichtigt werden können.

4.1 Migration der Daten

Grundsätzlich gestaltet sich die Migration zwischen zwei Datenbanken des gleichen Modells simpler als bei unterschiedlichen. Dokumentenorientierte Datenbanken arbeiten zumeist mit Dateien im JSON-Format und bieten die Möglichkeit, diese direkt per Export/Import zwischen den Datenbanken zu verschieben. Der Vorteil bei Verwendung der Import Funktion besteht darin, dass der komplette Datensatz in einem Schritt in die Zieldatenbank verschoben werden kann. Unabhängig von Quell- bzw. Zieldatenbank gestalten sich die Schritte für die Migration der Daten wie folgt:

1. Verbindung zur Quelldatenbank herstellen
2. Export des Datensatzes
3. Gegebenenfalls: Anpassungen des Datensatzes an Zieldatenbank
4. Verbindung zur Zieldatenbank herstellen
5. Import des Datensatzes

Zu beachten ist, dass zwischen den verschiedenen Datenbanken mit unterschiedlichen Datentypen gearbeitet wird und diese vor dem Import entsprechend modifiziert werden müssen. Beispielsweise arbeitet die MongoDB mit ObjectIDs, um die einzelnen Dokumente eindeutig zu identifizieren. Wird der Datensatz exportiert, werden die ObjectIDs wie folgt dargestellt: “_id“:{“$oid“:“5e4e96dc96dc45d3393c9344“}. Dieser Datentyp, signalisiert durch „$oid“, ist generell in anderen dokumentenorientierten Datenbanken nicht vorhanden und muss vor dem Import umgewandelt werden. Es ergeben sich zwei Möglichkeiten das Problem zu behandeln. Zum einen können die ObjectIDs entfernt und bei einem Import der Daten von der Zieldatenbank eigene IDs angelegt werden. Das ist nur akzeptabel, wenn in keinem Dokument eine Referenz auf die ObjectID vorliegt, andernfalls gehen die Referenzen verloren oder müssen neu angelegt werden. Die andere Möglichkeit sieht vor, den Datentyp für alle Dokumente anzupassen, sodass die Referenzen intakt bleiben.

Des Weiteren ist zu beachten, dass sich die Formatierung der Export-Datei von dem im Import erwarteten Darstellungsformat unterscheiden kann. Gängige Formatierungen, wie sie z.B. von MongoDB oder ArangoDB beim Import akzeptiert werden, sind zum einen das zeilenweise Hinterlegen von Dokumenten. Jede Zeile in der JSON-Datei entspricht einem kompletten Dokument, welches im Folgenden beispielhaft an kurzen Datensätzen dargestellt wird.

{“vorlesung_id“ : “msc1“, “professor_id“ : 101, “titel“ : “Grundseminar“, “hörsaal“ : “H1“}
{“vorlesung_id“ : “msc2“, “professor_id“ : 102, “titel“ : “TTI“, “hörsaal“ : “H2“}
{“professor_id“ : 101, “vorname“ : “Olaf“, “nachname“ : “Zukunft“, “raum“ : 1183}
{“professor_id“ : 102, “vorname“ : “Ulrike“, “nachname“ : “Steffens“, “raum“ : 1083}
...

Zum anderen kann die Datei aus einem Array von Dokumenten, welche durch ein Komma separiert werden, bestehen. Um die Lesbarkeit zu erhöhen, kann jedes Dokument in einer Zeile hinterlegt werden, jedoch ist das für dieses Format nicht zwingend notwendig.

[{“vorlesung_id“ : “msc1“, “professor_id“ : 101, “titel“ : “Grundseminar“, “hörsaal“ : “H1“}, {“vorlesung_id“ : “msc2“, “professor_id“ : 102, “titel“ : “TTI“, “hörsaal“ : “H2“}, {“professor_id“ : 101, “vorname“ : “Olaf“, “nachname“ : “Zukunft“, “raum“ : 1183}, {“professor_id“ : 102, “vorname“ : “Ulrike“, “nachname“ : “Steffens“, “raum“ : 1083}, ...]

Für beide Formatierungen sind die Leerzeichen zwischen zwei Attributen oder den Schlüssel-Wert-Paaren keine erforderlichen Voraussetzungen und als optional zu betrachten.

Werden die vorgestellten Problematiken beim Import der Daten berücksichtigt und entsprechend behandelt, so kann dieser komplikationslos durchgeführt werden.

4.2 Auswertung der Funktionalitäten für die Migration

Neben der Migration von Daten ist auch eine Übernahme ausgewählter Funktionalitäten möglich. Obwohl es sich hierbei um Datenbanken des gleichen Modells handelt, können sich die unterstützten Funktionalitäten unterscheiden. Nicht nur die Art der Funktionalität, sondern auch die Umsetzung dieser ist variabel. Da keine allgemeine Definition von dokumentenorientierten Funktionalitäten vorhanden ist, wird in diesem Kapitel eine exemplarische Auswertung durchgeführt. Obgleich diese an konkreten Beispielen erfolgt, schließt das nicht aus, dass andere Zieldatenbanken ebenfalls diese Funktionalitäten besitzen und theoretisch migrierbar wären.

Die Auswahl der Datenbanken ist über DB-Engines[1] geschehen, welche ein Ranking für verschiedene Datenbankmodelle bietet. Berücksichtigt wurden nur Open Source Vertreter und keine reinen Cloud-Service Datenbanken. Die MongoDB belegt den ersten Platz und ist eine dokumentenorientierte Datenbank, welche für ihren Cloud-Service Atlas zusätzlich als Suchmaschine fungiert. Um einen möglichst großen Kontrast zu dieser zu bieten, wurde ArangoDB ausgewählt, welche eine Multi-Modell Datenbank ist. Diese belegt Platz 12 im Ranking und ist die erste, die den gewünschten Kriterien entspricht. ArangoDB ist eine Graph-, Schlüssel-Wert-, sowie dokumentenorientierte Datenbank und besitzt eine eigene Suchmaschine, die ArangoSearch. Sie verwendet die Abfragesprache AQL, welche an SQL angelehnt ist, aber nur DML Operationen anbietet.

Die Erhebung der Informationen zu den Funktionalitäten, ist über die jeweilige Dokumentation von MongoDB [42] und ArangoDB [1, 3] erfolgt und stellt die Grundlage für die folgende Auswertung.

4.2.1 Datentypen

Bei den dokumentenorientierten Datenbanken ist das JSON-Format verbreitet. Da dieses jedoch nur sechs Datentypen zur Verfügung stellt, wird es zumeist von den Datenbanken erweitert. MongoDB verwaltet intern die Daten mit dem BSON-Format, ArangoDB

[1] https://db-engines.com/de/ranking/document+store

erweitert JSON mit dem VelocyPack[23]. Wie Tabelle 4.1 entnommen werden kann, sind die meisten und auch gängigsten Datentypen in beiden Datenbanken vorhanden. Fehlende Übereinstimmungen bei Zahlen haben zur Folge, dass sich die Genauigkeit verändern kann. Wird ein solcher Datentyp verwendet und ist nicht in der Zieldatenbank vorhanden, kann dies auf zwei Arten gehandhabt werden. Ist die Genauigkeit kein ein entscheidendes Kriterium für die Auswahl des Datentyps gewesen, so kann ein anderer ebenfalls akzeptabel sein. Alternativ kann die Zahl in einen String umgewandelt werden. Dabei ist zu beachten, dass bei der Weiterverarbeitung nun ein String vorliegt und dieser erst wieder angepasst werden muss. Neben den Zahlen stellt MongoDB eine ObjectID, welche der eindeutigen Identifizierung dient, zur Verfügung. Bei Referenzen auf die ObjectID, kann diese als String migriert und somit die Referenzen intakt gehalten werden. Für die Datentypen Regular Expression und JavaScript, JavaScript (with scope) ist kein Äquivalent in ArangoDB vorhanden. Diese können zwar als String übertragen werden, zu beachten ist jedoch, dass diese dann nicht mehr ihre ursprüngliche Funktion übernehmen können. Der Small Integer und der Uint der ArangoDB können alternativ als Integer oder Long dargestellt werden. Alle weiteren, unterschiedlichen Datentypen werden für interne Zwecke verwendet und sollten daher nicht beim Export der Daten vorhanden sein. Dementsprechend werden diese nicht weiter berücksichtigt.

4.2.2 Abfragen

Bei den Abfragesprachen unterschieden sich MongoDB und ArangoDB stark. ArangoDB verwendet ArangoDB Query Language (AQL), eine an SQL angelehnte Sprache. MongoDB hingegen arbeitet mit Operator:Wert-Paaren, welche auch verschachtelt sein können. Eine Übersicht über die grundlegendsten Operatoren kann der Tabelle 4.2 entnommen werden. Durch die unterschiedlichen Sprachen variiert ebenfalls die Darstellung der Operatoren. Die Basisoperatoren stellen eine Grundlage an gemeinsamen Operatoren, ohne die kaum eine Formulierung einer Abfrage möglich wäre. Bereits im Hauptprojekt [28] wurde für fünf dokumentenorientierte Datenbanken eine Analyse bezüglich der Operatoren durchgeführt. Dort hat sich gezeigt, dass die in Tabelle 4.2 aufgelisteten Operatoren nicht nur für MongoDB und ArangoDB eine Basis bilden, sondern auch für weitere dokumentenorientierte Datenbanken [28]. Neben den spezifischen Operatoren werden auch vordefinierte Funktionen verwendet, um Abfragen zu definieren. Z.B. ist in Tabelle 4.2

[2] https://github.com/arangodb/velocypack
[3] https://github.com/arangodb/velocypack/blob/master/VelocyPack.md

Tabelle 4.1: Datentypen von MongoDB und ArangoDB

Datentyp	MongoDB	ArangoDB**
JSON		
Number	x	x
String	x	x
Boolean	x	x
Array	x	x
Object	x	x
Null	x	x
Double	x	x
Binary Data	x	x
Date	x	x
Integer	x	x
Small integer		x
Uint		x
ObjectID	x	
Regular Expression	x	
JavaScript	x	
JavaScript (with scope)	x	
Timestamp	x*	
Long	x	
Decimal128	x	
Min key	x*	x*
Max key	x*	x*
External VPack values		x*
BCD		x
Tagging		x*
Custom Types		x*

**Datentypen, welche in der Regel nur für interne Zwecke verwendet werden.*
***VelocyPack Datentypen ermittelt über die Spezifikation [5]*

für die ArangoDB die Funktion EXISTS() zuständig, um zu überprüfen, ob ein Feld existiert.

Die größte Problematik bei der Abfragesprache ist, dass NoSQL keinen Standard stellt und jede Datenbank eigenständig entscheiden kann, wie die Abfragen erfolgen. Somit ist unumgänglich, dass für jede Datenbank eigenständig analysiert werden muss, ob und wie viele Operatoren/Funktionen migrierbar sind. Da die Abfragesprache weit komplexer ist, als hier in der Tabelle 4.2 dargestellt, definiert sie eine eigenständige Problematik und kann nicht in der Masterthesis berücksichtigt werden.

Vergleicht man die Operatoren aus Tabelle 4.2, fehlt der MongoDB ein Äquivalent zu LIKE, ! ∼, NONE und der ArangoDB zu $nor. Diese lassen sich jedoch dennoch mit den Basisoperatoren ausdrücken. Bei LIKE handelt es sich um einen speziellen Fall eines regulären Ausdrucks. Ebenso kann mit $not die Verneinung eines Regex ausgedrückt werden. Darf kein Wert im Array enthalten sein, kann ebenfalls mit der Verneinung gearbeitet werden, das Gleiche gilt für das fehlende $nor der ArangoDB.

Die hier aufgeführten Basisoperatoren bilden nur das absolute Minimum, um überhaupt eine Abfrage formulieren zu können. Damit wird nicht ansatzweise das Spektrum der Abfragesprachen dargestellt. Die ArangoDB besitzt 250 Schlüsselwörter, welche Operatoren und Funktionen beschreiben und für die Migration individuell ausgewertet werden müssten. D.h. für jeden Operator bzw. Funktion muss in der zu übersetzenden Sprache überprüft werden, ob es einen äquivalenten Operator/Funktion gibt. Ebenso ist für die Migration der Abfragen nötig, zu erkennen, welche Schlüsselwörter nicht übersetzt werden können und diese dementsprechend herauszufiltern. Neben den durch die Datenbank definierten Funktionen, können benutzerdefinierte erstellt werden. Eine Diskussion dieser erfolgt in Kapitel 4.2.8. Die ArangoDB besitzt insgesamt 43 Operatoren aus drei Kategorien.

Operatoren:	27
Array Operatoren:	2
High Level Operatoren:	14

Tabelle 4.2: Basisoperatoren von MongoDB und ArangoDB

Operator	MongoDB	ArangoDB
Vergleichsoperatoren		
Gleichheit	$eq	
Ungleichheit	$ne	!
Kleiner	$lt	<
Kleiner gleich	$lte	<=
Größer	$gt	>
Größer gleich	$gte	>=
String entspricht Muster		LIKE
String entspricht nicht Muster		NOT LIKE
Wert entspricht Regex	$regex	=~
Wert entspricht nicht Regex		!~
Element Operatoren		
Feld existiert	$exists	EXIST()*
Datentyp des Feldes	$type	IS_<TYPE>()**
Array Operatoren		
Wert in Array	$in	IN
Wert nicht in Array	$nin	NOT IN
Alle Werte der Abfrage im Array	$all	ALL
Mind. ein Wert im Array entspricht der Abfrage	$elemMatch	ANY
Kein Wert im Array enthalten		NONE
Länge des Arrays	$size	LENGTH()
Logische Operatoren		
Und	$and	&&, AND
Oder	$or	||, OR
Nicht	$not	!, NOT
Nor	$nor	
Arithmetische Operatoren		
Addition	$add	+
Subtraktion	$subtract	-
Multiplikation	$multiply	*
Division	$divide	/
Modulo	$mod	%
Andere Operatoren		
Ternärer Operator	ja	ja
Range	$range	..

**Nur in Verbindung mit ArangoSearch verwendbar.*
***Jeder Datentyp kann abgefragt werden z.B. IS_ BOOL().*

Zusätzlich werden 207 Funktionen von der ArangoDB definiert, welche sich in zehn Kategorien gliedern.

ArangoSearch:	10
Array:	28
Datum:	24
Dokument/Objekt:	15
Volltext:	1
Geo:	16
Sonstige:	22
Numerisch:	36
String:	39
Type check & cast:	16

Die MongoDB unterteilt ihre Operatoren/Funktionen in sechs Kategorien und besitzt insgesamt 231 Schlüsselwörter.

Abfrage- und Projektionsoperatoren:	44
Update Operatoren:	22
Aggregation Pipeline Stages:	30
Aggregation:	124
Abfragemodifikatoren:	11

Um die Abfragen vollwertig auswerten zu können, müsste eine Auswertung aller insgesamt 481 Schlüsselwörter und den damit verbundenen Operatoren/Funktionen untersucht werden. Es wäre eine Beschränkung auf die Basisoperatoren denkbar, jedoch sind damit nur absolut primitive Abfragen möglich, welche nicht die Realität widerspiegeln. Mit den Basisoperatoren wird nur etwas mehr als ein Zehntel der möglichen Operatoren/Funktionen abgebildet. In Anbetracht dessen werden die Abfragen bei der Migration nicht berücksichtigt.

4.2.3 Views

Beide Datenbanken unterstützen das Konzept der Views, ArangoDB ermöglicht dies nur unter der Verwendung von ArangoSearch[4], der Volltext Suchmaschine. Für die Views ergeben sich die gleichen Einschränkungen wie im vorherigen Kapitel 4.2.2. Liegt ein Mapping für alle Operatoren/Funktionen vor, dann können die Views entsprechend ausgewertet und übersetzt werden, andernfalls stellt dies einen zu aufwendigen Vorgang dar.

[4] https://www.arangodb.com/docs/stable/arangosearch.html

Dieser Prozess muss manuell vom Anwender durchgeführt werden und ist kein Element der Masterthesis.

4.2.4 Map-Reduce

Sowohl MongoDB, als auch ArangoDB können mittels JavaScript (JS) Map-Reduce Operationen ausführen. Der Unterschied liegt darin, dass MongoDB eine Operation anbietet, welche als Map und Reduce eine JS Funktion erwartet. ArangoDB hingegen unterstützt nicht nativ Map-Reduce und muss komplett über JS formuliert werden. Die Problematik, die dabei entsteht, ist, dass beide Datenbanken prinzipiell die Funktionalität unterstützen, aber auf unterschiedliche Weise. Eine mögliche Lösung für die Migration von MongoDB zu ArangoDB wäre eine Art Schablone für Map-Reduce, in die die Map- und Reduce-Funktion aus der MongoDB eingelesen wird. Anders herum gestaltet es sich schwierig, da ArangoDB keinen Rahmen festlegt, wie Map-Reduce in JS definiert werden muss. Damit wird ein gewisser Handlungsspielraum für den Aufbau gegeben, welcher es erschwert, automatisiert die wichtigen Funktionen für die MongoDB zu identifizieren. Da auch hier verschiedene Operatoren und Funktionen für das Map-Reduce verwendet werden, ist eine Migration nur möglich, wenn für alle Operatoren/Funktionen ein äquivalent in Quell- und Zieldatenbank vorhanden ist. Aufgrund dessen können prinzipiell Map-Reduce Operationen migriert werden, jedoch werden sie in der Masterthesis nicht berücksichtigt, da kein Mapping für die Operatoren/Funktionen erstellt wird.

4.2.5 Indizes

Dokumentenorientierte Datenbanken unterstützen unter anderem das Konzept der Indizes, um eine Verbesserung der Performance bei Abfragen zu erzielen. Dazu wird zwischen unterschiedlichen Typen von Indizes unterschieden, welche für verschiedene Ansprüche geeignet sind. Ebenso können diese unterschiedliche Eigenschaften aufweisen, wie unique, sparse, partial oder TTL. Nicht jeder Index kann jede Eigenschaft aufweisen. Bei beiden Datenbanken können die Indizes auf ein oder mehrere Felder, sowie auf Arrays angelegt werden. Bei den Arrays ist zu beachten, dass MongoDB und ArangoDB diese unterschiedlich interpretieren. MongoDB indiziert jedes Element des Arrays, ArangoDB nur das komplette Array. Ebenso gibt es für spezielle Datentypen einen eigenen Index, wie für Geospatial-Daten oder Volltext suchen. Je nach Datenbank kann der Index auch anders definiert sein. Bei MongoDB ist der TTL Index eine Eigenschaft, bei ArangoDB

ein Index-Typ, welcher über ein Feld mit einem Datum erfolgt. Unabhängig davon dient der Index in beiden Fällen dazu, Dokumente nach bestimmter Zeit automatisiert zu entfernen. Da ArangoDB eine Multi-Modell Datenbank ist, gibt es Indizes, die nicht für das dokumentenorientierte Modell geeignet sind. Dies ist z.B. der Edge-Index, welcher speziell für das Graphenmodell ist und deswegen hier nicht berücksichtigt wird. Die jeweiligen Kombinationen von Index-Typ und Index-Eigenschaft können für die ArangoDB der Tabelle 4.3 und für die MongoDB der Tabelle 4.4 entnommen werden.

Tabelle 4.3: Indizes bei der ArangoDB

Typ	Unique	Sparse
Geospatial		x
Fulltext		x
TTL		x
Persistent	x	x
Skiplist*	x	x
Hash*	x	x

**Mit der RocksDB Storage Engine sind Hash bzw. Skiplist Indizes Persistente. Für die Kompatibilität mit älteren Versionen der ArangoDB werden diese jedoch noch unterstützt.*

Die Indizes Geospatial, Fulltext und TTL der ArangoDB sind immer per Default sparse und können nicht unique sein.

Tabelle 4.4: Indizes bei der MongoDB

Typ/Eigenschaft	Unique	Sparse	Partial	TTL
Geospatial	x	x	x	x
Fulltext	x	x	x	x
Hash		x	x	x

Bei der MongoDB ist zu beachten, dass ein Index nicht gleichzeitig partial und sparse sein kann. Die MongoDB besitzt als Geospatial Index den 2d Index, welcher veraltet ist und nur für ältere Versionen der MongoDB gedacht ist. Der geoHaystack Index ist ein spezialisierter Geoindex und zumeist ist die Verwendung des 2dsphere-Indexes angeraten. Für die Migration bedeutet dies, dass diese drei Typen als ein Geospatial-Index angesehen werden und keine weitere Untergliederung möglich ist. Ebenfalls ist zu berücksichtigten, dass der Geoindex nur auf ein Feld erfolgen kann, bei der ArangoDB kann dieser auf mehreren liegen. Bei der TTL Eigenschaft ist darauf zu achten, dass diese auf Feldern mit

einem Datum angelegt wird, andernfalls funktioniert sie nicht. Das gilt gleichermaßen für die ArangoDB. Neben den in Tabelle 4.4 vorgestellten Indizes, kann die MongoDB Indizes verstecken, welche dann nicht für Abfragen verwendet werden können. Dementsprechend werden diese bei der Migration nicht berücksichtigt. Gleiches gilt für die Wildcard Indizes. Diese erlauben mit Platzhaltern die Felder unspezifiziert zu lassen und somit unbekannte Felder zu indizieren.

Da bei der Migration der Daten keine grundlegende Veränderung der Struktur zu erwarten ist, können die Indizes aus der Quelldatenbank ausgelesen und in der Zieldatenbank erneut angelegt werden. Ebenso gibt es sowohl bei den Typen, als auch den Eigenschaften eine große Übereinstimmung, sodass diese nahezu verlustfrei migriert werden können. Besitzt ein Index die partial Eigenschaft, muss der Anwender entscheiden, ob dieser ohne die Eigenschaft migriert wird oder übersprungen werden soll.

4.2.6 Skalierung

NoSQL bietet Konzepte für die Skalierung und Verfügbarkeit an. Die horizontale Skalierung wird durch das Sharding erzielt. Dabei wird der Datensatz auf verschiedenen Servern in Shards aufgeteilt, welche eine Teilmenge der Daten enthalten. Die Replikation ermöglicht eine hohe Verfügbarkeit, indem redundante Datensätze vorhanden sind. Dieses Konzept sorgt für eine Fehlertoleranz. Oftmals werden die beiden Konzepte kombiniert verwendet und nicht einzeln, dadurch entstehen Sharded Clusters. Hierbei ist zu beachten, dass bei einer fehlerhaften Einrichtung von Shards oder Replikationsservern erhebliche Einbußen bei der Performance zu erwarten sind. Da beide Konzepte ohne Einschätzung durch den Anwender nicht korrekt angelegt werden können, wird dies für die automatisierte Migration nicht übernommen und muss manuell durch den Anwender ergänzt werden.

4.2.7 Benutzer und Rollen

Eine Komponente für die Sicherheit der Datenbank ist das Erstellen und Verwalten von Benutzern und Rollen. Das Konzept ist sowohl in der MongoDB, als auch der ArangoDB vertreten, jedoch in einem unterschiedlichen Umfang. Aufgrund dessen, dass die MongoDB ein breiteres Spektrum an Aktionen für die Rollen zur Verfügung stellt, wurde sich an der ArangoDB orientiert. Aktionen sind Kommandos, welche der Benutzer in einer

definierten Rolle ausführen kann. ArangoDB unterteilt die Aktionen in drei Kategorien, der Server-, der Datenbank- und der Kollektionsaktionen. Erhält der Benutzer über eine Ebene den Zugriff, so können alle Aktionen verwendet werden. Es können keine Aktionen individuell zugeteilt werden, dies ist nur in der MongoDB möglich. Um eine Aktion auszuführen, muss dem Benutzer eine entsprechende Zugriffsebene zugewiesen werden. MongoDB verzichtet auf Zugriffsebenen, ist der Rolle eine Aktion zugeteilt, so kann diese verwendet werden. Darüber hinaus bietet die MongoDB eine individuellere Anpassung der Aktionen für die Benutzer und Rollen.

Bei Serveraktionen kann die Zugriffsebene entweder „administrate“ oder „no access“ sein. D.h. der Benutzer benötigt als Ebene „administrate“, alternativ kann er keine Aktion verwenden. Die Serveraktionen können der Tabelle 4.5 entnommen werden. Die MongoDB bietet keine explizite Aktion an, welche das Erstellen von Datenbanken erlaubt. Wird in der MongoDB ein Dokument in einer Kollektion angelegt, so wird die darüber definierte Datenbank automatisch angelegt, falls sie noch nicht vorhanden ist. Somit gibt es zwar keine äquivalente Aktion, aber es entsteht auch keine Einschränkung, da es nicht nötig ist, für diese Aktion explizit Berechtigungen zu besitzen.

Tabelle 4.5: Serveraktionen (entnommen aus [4])

Serveraktion	Datenbankebene
~~create database~~	~~administrate~~
drop database	administrate
create user	administrate
update user	administrate
drop user	administrate
shutdown server*	administrate

**Das Kommando muss bei der MongoDB gegen die Admin-Datenbank ausgeführt werden.*

Datenbankaktionen brauchen auf der Datenbankebene einen Zugriff, der entweder „administrate“ oder „access“ ist. Auf der Kollektionsebene kann der Zugriff entweder „read/write“ oder „read only“ sein. Auf beiden Ebenen kann bei „no access“ nicht darauf zugegriffen werden. Eine Aktion kann nur ausgeführt werden, wenn die der Tabelle entsprechende Kombination gegeben ist oder mehr Rechte erteilt wurden. Die Datenbankaktionen können der Tabelle 4.6 entnommen werden. Die ArangoDB bietet unter anderem die Möglichkeit, die Eigenschaften von Kollektionen auszulesen und zu modifizieren. Bei der MongoDB kann eine Veränderung der Eigenschaften über die Aktion „collMod“ erzielt werden. Eigenschaften werden über die Auflistung von Kollektionen angezeigt.

Tabelle 4.6: Datenbankaktionen (entnommen aus [4])

Datenbankaktion	Datenbankebene	Kollektionsebene
create collection	administrate	read/write
list collections	access	read only
rename collection	administrate	read/write
modify collection properties	administrate	read/write
read properties	access	read only
drop collection	administrate	read/write
create index	administrate	read/write
drop index	administrate	read/write
see index definition	access	read only

Kollektionsaktionen können Änderungen an Dokumenten vornehmen. Auf Datenbankebene müssen „administrate“ oder „access“ Rechte zugewiesen sein, auf Kollektionsebene „read/write“ oder „read only“. Hier gilt ebenfalls bei „no access“ kann nicht darauf zugegriffen werden. In Tabelle 4.7 sind die Aktionen aufgelistet. Da MongoDB variierende Bezeichner verwendet, wurden diese hier ergänzt.

Tabelle 4.7: Kollektionsaktionen (angelehnt an [4])

Kollektionsaktion	Datenbankebene	Kollektionsebene	MongoDB
read document	administrate, access	read/write, read only	find
create document	administrate, access	read/write	insert
modify document	administrate, access	read/write	update
drop document	administrate, access	read/write	remove
truncate a collection	administrate, access	read/write	remove()

Da alle Aktionen von der ArangoDB in die MongoDB migrierbar sind, mit der Ausnahme, dass das Erstellen von Datenbanken nicht explizit erfolgt, können Benutzer und Rollen bei der Migration berücksichtigt werden. Zu beachten ist, dass die MongoDB Aktionen besitzt, welche in der ArangoDB nicht realisierbar sind. Wird eine dieser Aktionen genutzt, so kann diese nicht migriert werden und der Anwender muss manuell entscheiden, ob eine neue Rolle ohne diese erstellt oder die komplette Rolle nicht berücksichtigt wird. Werden Rollen von der ArangoDB zur MongoDB migriert, so können diese uneingeschränkt abgebildet werden.

4.2.8 Benutzerdefinierte Funktionen

Funktionen werden bei MongoDB in JS geschrieben und in der Kollektion system.js für die Wiederverwendung gespeichert. ArangoDB erweitert mittels Funktionen, welche ebenfalls in JS geschrieben werden, die Abfragesprache AQL. Beispielsweise können damit in AQL bestimmte Abfragen vereinfacht oder fehlende Funktionen ergänzt werden.

Obwohl beide Datenbanken über JS benutzerdefinierte Funktionen bereitstellen, können diese bei einer Migration nicht direkt übernommen werden. Sind die zu migrierenden Funktionen primitiv gehalten und unabhängig von der zugrundeliegenden Datenbank, ist prinzipiell eine Übernahme möglich. Eine Gewährleistung, dass diese in der Zieldatenbank korrekt arbeiten, kann allerdings nicht gegeben werden. Zur Verdeutlichung der Problematik werden zwei Beispiele vorgestellt, welche sich auf die Datentypen und die Verwendung von Operatoren in den Funktionen beziehen.
MongoDB und ArangoDB weisen eine große Schnittmenge bei den Datentypen auf, jedoch gibt es Unterschiede. Z.B. wird in der Dokumentation von ArangoDB eine einfache Funktion vorgestellt, welche einen übergebenen Preis mit 0,19 multipliziert und das Ergebnis zurück gibt. Wird die Funktion nun nach einer Migration mit dem Datentyp „number“ anstatt „double“ aufgerufen, so kann das Ergebnis eine andere Genauigkeit aufweisen. Ist Genauigkeit bei der Funktion ein entscheidendes Kriterium, dann muss sichergestellt werden, dass in Quell- und Zieldatenbank der gleiche Datentyp verwendet wird.
Ebenfalls können Funktionen in JS datenbankspezifische Operatoren verwenden. Z.B. kann eine benutzerdefinierte Funktion der MongoDB mittels des Operators „find()“ definieren, welche Datensätze gefunden werden sollen. ArangoDB arbeitet mit AQL, welche Abfragen mit unterschiedlichen Operatoren formuliert. Solche Funktionen sind bei einer Migration an die Syntax der Zieldatenbank anzupassen. Dazu ist eine manuelle Überprüfung aller Funktionen und gegebenenfalls eine Anpassung nötig, welche nicht automatisiert erfolgen kann.

Insgesamt ist für die Migration der benutzerdefinierten Funktionen eine manuelle Bearbeitung und Bewertung unablässig. Aufgrund dessen werden diese nicht berücksichtigt, da der Prozess nicht ohne erheblichen Aufwand oder automatisiert erfolgen kann.

4.3 Zusammenfassung der Migration zwischen dokumentenorientierten Datenbanken

In der Gesamtheit lassen sich im Rahmen der Masterthesis die Daten und folgende Funktionalitäten migrieren:

- Daten (fast uneingeschränkt)
- Indizes (fast uneingeschränkt)
- Benutzer und Rollen (einseitig stark eingeschränkt)

Die Daten können unter der Berücksichtigung von Datentypen und Import-Formatierung vollständig übernommen werden. Wird ein genutzter Datentyp in der Zieldatenbank nicht unterstützt, so muss dieser in einen anderen umgewandelt werden. Dabei kann sich die Funktionsweise ändern, welches beachtet werden muss. Indizes können fast komplett uneingeschränkt übernommen werden. Wird die Eigenschaft partial verwendet, so muss dies durch den Anwender entschieden werden. Alle weiteren Indizes können übertragen werden, auch wenn der zugrundeliegende Typ sich dadurch ändert. Benutzer und Rollen können bei der Migration von ArangoDB zu MongoDB uneingeschränkt migriert werden. Durch die Vielfalt der Aktionen von der MongoDB sind starke Einschränkungen zu akzeptieren. Nur die aufgelisteten Aktionen sind migrierbar.

Nicht berücksichtigt werden können die Abfragen und damit verbunden auch nicht die Views und Map-Reduce. Der Arbeitsaufwand und die Komplexität eines kompletten Mappings der Operatoren/Funktionen übersteigt den Rahmen der Masterthesis. Das Gleiche gilt für benutzerdefinierte Funktionen. Ebenso nicht beachtet werden die Themen der Skalierung, da diese fachkundig beurteilt werden müssen und nicht ohne Bewertung des Anwenders eingerichtet werden können. Die Skalierung kann in beiden Datenbanken erfolgen und sollte an die Gegebenheiten der Datenbank angepasst werden. Wie ein Cluster mit Replikation und Sharding effektiv erstellt werden kann, hängt vom Einzelfall ab. Dazu ist eine genauere Analyse nötig, welche im Rahmen der Masterthesis nicht automatisiert erfolgen kann.

Alle manuellen Ergänzungen unterliegen der Verantwortung des Anwenders und müssen außerhalb des Migrationsprozesses gesondert behandelt werden und finden keine weitere Berücksichtigung in der Masterthesis.

5 Evaluierung der eigenen Konzepte durch das Migrationstool Rel(Dok)^2

In diesem Kapitel wird das im Rahmen der Masterthesis erstellte Migrationstool Rel(Dok)^2 vorgestellt, welches die in den vorherigen Kapiteln ausführlich diskutierten Konzepte auf die Umsetzbarkeit in der Praxis testet. Zunächst werden die Anforderungen an das Tool festgelegt (vgl. Kapitel 5.1) und ein Entwurf vorgestellt (vgl. Kapitel 5.2). Anschließend wird in Kapitel 5.3 auf die Implementierung eingegangen. Um die korrekte Funktion des Tools zu überprüfen, werden Beispieldatensätze in Kapitel 5.4 definiert und die Ergebnisse bei der Benutzung des Tools erläutert (vgl. Kapitel 5.5).

5.1 Anforderungen

Für die Anforderungen an die Rel(Dok)^2 wurden die Software-Qualitätseigenschaften [56] des Standards ISO 25010 [23] als Maßstab gewählt. Dieser Standard ist der Nachfolger des ISO 9126 Standards. Insgesamt werden acht Kategorien definiert. Zu diesen wird im Folgenden auf die Rel(Dok)^2 bezogen, analysiert, welche Merkmale diese erfüllt. Da es sich bei der Rel(Dok)^2 um ein Tool für die Evaluierung der Konzepte handelt, werden einige Eigenschaften nicht berücksichtigt, die für ein Produkt notwendig wären.

5.1.1 Technische Anforderungen

Um die Rel(Dok)^2 verwenden zu können, sind folgende Datenbanken und Versionen auf dem auszuführenden System notwendig.

T-1 MongoDB 4.0.11

T-2 ArangoDB 3.6.4

T-3 PostgreSQL 12.3

T-4 Java 14

Werden ältere bzw. neuere Versionen verwendet, kann nicht garantiert werden, dass die Rel(Dok)^2 kompatibel mit diesen ist. Um dies sicherzustellen, wäre ein weitreichender Support des Tools nötig.

5.1.2 Funktionale Anforderungen

Die funktionalen Anforderungen spezifizieren, welche Aktionen durch den Benutzer möglich sein sollen und sind im folgenden aufgelistet.

F-1 Der Anwender kann in einer Settings-Datei die Pfade zu den Import/Export Tools der Datenbank, sowie Benutzernamen, Passwort, Host und Port hinterlegen.

F-2 Der hinterlegte Benutzer weist die nötigen Berechtigungen auf, welche für den Migrationsprozess erforderlich sind.

F-3 Der Anwender kann auswählen, zwischen welchen Datenbanken migriert wird.

F-4 Der Anwender kann auswählen, ob die eindeutige ID übernommen oder automatisch durch die Datenbank neu angelegt wird.

F-5 Der Anwender kann auswählen, ob die Indizes und/oder Benutzer bzw. Rollen migriert werden.

F-6 Bei der Auswahl der Migration von der MongoDB zur ArangoDB kann zusätzlich entschieden werden, ob partielle Indizes und/oder Teilmengen von Benutzern migriert werden.

F-7 Bei der Migration von der PostgreSQL zur MongoDB wird ein neues Fenster mit dem automatischen Vorschlag für das Schema geöffnet. Darüber kann der Anwender individuelle Anpassungen vornehmen, nach denen abschließend migriert wird.

5.1.3 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen werden in verschiedene Kategorien unterteilt. Jede dieser Kategorien besitzt unterschiedliche Qualitätsmerkmale welche, basierend auf der Definition des ISO 25010 Standards [23], kurz erläutert werden. Die Bezeichner der Merkmale sind der Quelle [56] entnommen. Anschließend wird dargelegt, welche Anforderungen sich daraus für den Entwurf und die Realisierung der Rel(Dok)^2 ergeben.

Funktionalität

Zur Funktionalität gehören die Vollständigkeit, Richtigkeit und Angemessenheit. Die Vollständigkeit erfasst, ob durch die Funktionen des Tools alle Aufgaben und Benutzerziele erfüllt werden. Die Richtigkeit legt die fest, ob die Ergebnisse korrekt sind und die nötige Präzision aufweisen. Die Angemessenheit erfasst, ob die Erfüllung von bestimmten Aufgaben durch die Funktion des Tools erleichtert werden.

Die Rel(Dok)^2 realisiert ausgewählte Funktionalitäten und erfüllt damit die Angemessenheit. Da nicht alle im Konzept erfassten und theoretisch umsetzbaren Funktionalitäten über das Tool migrierbar sind, ist die Vollständigkeit nicht gegeben. Dennoch soll die Richtigkeit sichergestellt sein und nachvollziehbar bleiben, ob die Migration korrekt durchgeführt wurde. Daraus ergeben sich die folgenden zwei Anforderungen.

N-1 Es ist nachvollziehbar, ob die Migration durch die Rel(Dok)^2 korrekt erfolgt ist.

N-2 Die Rel(Dok)^2 ist fähig, die ausgewählten Funktionalitäten und die Daten zu migrieren.

Performanz

Die Performanz bezieht sich auf das Zeitverhalten, den Ressourcengebrauch und die Kapazität. Das Zeitverhalten umfasst die Antworts- und Ausführungszeit, der Ressourcengebrauch legt die Menge und die Art der Ressourcen fest und die Kapazität bestimmt das maximale Limit.

Da die Rel(Dok)^2 kein fertiges Produkt ist, sondern begleitend zur Masterthesis entwickelt wurde, werden keine Anforderungen an diese Bereiche gestellt.

Kompatibilität

Die Kompatibilität definiert über die Koexistenz und die Interoperabilität, inwiefern das Tool neben anderen Programmen agieren kann und wie stark der Austausch von Informationen zwischen verschiedenen Systemen oder Komponenten ist.

Daraus ergibt sich die Anforderung, dass die Rel(Dok)^2 neben den laufenden Datenbanken ausgeführt werden können muss, damit das Tool Informationen erhalten und auswerten kann.

N-3 Die Rel(Dok)^2 kann neben anderen Programmen ausgeführt werden.

N-4 Die Rel(Dok)^2 steht im Informationsaustausch mit den Datenbanken, welche im Migrationsprozess beteiligt sind.

Benutzbarkeit

Die Benutzbarkeit legt Kriterien fest, welche die Anwenderfreundlichkeit genauer definieren. Dazu gehört die Angemessenheit, welche dem Anwender ermöglicht zu erfassen, ob das Tool für seine Zwecke geeignet ist. Die Erlernbarkeit definiert den Schwierigkeitsgrad für die Benutzung des Tools. Die Bedienbarkeit bestimmt, ob das Tool einfach zu verwenden ist. Die Fehlertoleranz spiegelt wider, wie anfällig das Tool für Benutzungsfehler ist. Die Ästhetik erfasst, wie ansprechend das Tool optisch gestaltet ist. Die Zugänglichkeit bestimmt, ob das Tool für unterschiedliche Personen zugänglich ist.

Damit die Erlernbarkeit und Bedienbarkeit gegeben sind, soll die Rel(Dok)^2 übersichtlich und schlicht gestaltet sein, jedoch keine speziellen Kriterien für die Ästhetik erfüllen. Ebenso sollte für den Anwender erkennbar sein, wofür das Tool geeignet ist, damit die Angemessenheit definiert werden kann. Um die Fehlertoleranz sicherzustellen, muss ein breites Spektrum an typischen Nutzungsfehlern erfasst und abgefangen werden. Dies ist für den Rahmen der Masterthesis zu aufwändig und wird nicht berücksichtigt. Insgesamt ergeben sich daraus die folgenden Anforderungen.

N-5 Der Anwender kann erkennen, ob die Rel(Dok)^2 für seine Zwecke geeignet ist.

N-6 Der Anwender kann die Verwendung der Rel(Dok)^2 einfach erlernen.

N-7 Die Rel(Dok)^2 soll übersichtlich und leicht zu bedienen sein.

Zuverlässigkeit

Die Zuverlässigkeit bestimmt, wie verlässlich Funktionen oder Komponenten des Tools arbeiten. Dazu gehört die Reife, welche die Zuverlässigkeit unter normalen Bedingungen bestimmt. Die Verfügbarkeit legt fest, ob der Zugang zum Tool gewährt ist, wenn es benötigt wird. Die Fehlertoleranz erfasst, wie gut mit Fehlern auf Software- oder Hardwarebasis umgegangen werden kann. Die Wiederherstellbarkeit legt fest, wie gut das Tool die Daten nach einem Fehler wiederherstellen kann und den ursprünglichen Zustand wieder erreicht.

Für die Rel(Dok)^2 werden dazu keine weiteren Anforderungen gestellt, da diese weder dazu gedacht ist, auf verschiedensten Systemen zu laufen, noch ein fertiges Produkt darstellt. Damit können die Kriterien nicht gewährleistest werden. Die Verfügbarkeit soll gegeben sein, denn zur Masterthesis gehört für Interessierte der Zugang zum Tool.

N-8 Die Rel(Dok)^2 soll für Interessierte zugänglich sein.

Sicherheit

Unter die Sicherheit fallen die Konzepte Vertraulichkeit, Integrität, Nachweisbarkeit, Verantwortlichkeit und Authentifizierbarkeit. Unter Vertraulichkeit wird erfasst, ob der Zugriff auf die Daten nur von Berechtigten erfolgt, die Authentifizierbarkeit bestimmt dazu, ob eine Person oder ein Programm auch jene sind, für die sie sich ausgeben. Integrität erfasst, ob unautorisierte Zugriffe oder Modifikationen vom Tool unterbunden werden. Die Nachweisbarkeit stellt sicher, dass Aktionen erfolgt sind und damit nicht dupliziert werden und die Verantwortlichkeit legt fest, inwiefern Aktionen eindeutig nach verfolgbar sind.

Diese Konzepte der Sicherheit sind kein Kriterium für die Masterthesis, da die Rel(Dok)^2 nur zur Testung des entwickelten Konzeptes dient und nicht offiziell als ein Migrationstool veröffentlicht wird.

Wartbarkeit

Die Wartbarkeit definiert Kriterien, welche festlegen, wie gut eine Wartung durchgeführt werden kann. Die Modularität bestimmt, ob das Programm in einzelne Module unterteilt ist und somit bei Veränderungen nur minimale Anpassungen bei anderen Modulen nötig

sind. Die Wiederverwendbarkeit bestimmt, ob die Komponenten des Tools in anderen Systemen oder Programmen verwendet werden können. Die Analysierbarkeit legt fest, inwiefern es möglich ist, im Tool Mängel oder Fehlerursachen aufzuspüren, sowie Auswirkungen von Änderungen zu bewerten. Die Modifizierbarkeit erfasst, ob das Tool modifiziert werden kann, ohne die Qualität des Tools zu reduzieren oder Fehler hinzuzufügen. Die Testbarkeit bestimmt die Möglichkeit, Testkriterien auf das Tool anzuwenden.

Da die Rel(Dok)^2 für die Masterthesis entwickelt wurde, ist eine Wiederverwendbarkeit kaum gegeben, es sei denn, das Thema der Thesis wird aufgegriffen. Es werden keine Mechanismen für die Analyse geboten und die Modifizierbarkeit wird ebenfalls nicht erfasst, da es sich um kein fertiges Produkt handelt. Die Modularität soll unterstützt werden, da mehrere Datenbanken an dem Migrationsprozess beteiligt sind und eine Erweiterung nicht grundlegend ausgeschlossen sein soll. Die Rel(Dok)^2 soll sowohl mit Softwaretests, als auch durch Beispielszenarien getestet werden. Insgesamt ergeben sich damit die folgenden Anforderungen.

N-9 Die Rel(Dok)^2 schließt die Erweiterbarkeit über zusätzliche Datenbanken nicht aus und unterstützt das Konzept der Modularität.

N-10 Der komplette Migrationsprozess der Rel(Dok)^2 wird über Anwendungsfälle getestet.

N-11 Über Softwaretests werden grundlegende Funktionen der Rel(Dok)^2 getestet.

Portierbarkeit

Die Portierbarkeit definiert die Anpassbarkeit, Installierbarkeit und Austauschbarkeit. Die Anpassbarkeit legt fest, inwiefern ein Tool in einer anderen Umgebung oder auf anderer Hardware, Software arbeiten kann. Die Installierbarkeit bestimmt, wie effizient der Prozess der Installation/Deinstallation ist und die Austauschbarkeit bezieht sich darauf, ob das Tool ein anderes Produkt für die gleiche Thematik ersetzen kann.

Da die Rel(Dok)^2 im Rahmen der Masterthesis entwickelt wurde, kann sie kein bestehendes Produkt ersetzen, sondern ist speziell für die Thesis ausgelegt. Dabei handelt es sich nicht um ein installiertes Programm, sondern ein ausführbares Tool. Ob die Rel(Dok)^2 auf anderer Hardware oder Software weiterhin korrekt arbeitet, wird nicht überprüft. Es wird in den technischen Anforderungen festgelegt, welche Software bzw. Versionen benötigt werden.

5.2 Entwurf

Dieses Kapitel befasst sich mit dem Entwurf des Tools Rel(Dok)^2 für die Migration zwischen den verschiedenen Datenbanken. Bevor das Tool gestartet wird, ist es notwendig, in der Settings-Datei die Angaben zu Benutzer, Passwort, Host, Port, sowie den Pfaden zu den datenbankeigenen Import/Export Tools zu geben. Nach dem Start der Rel(Dok)^2 kann der Anwender die Beispieldatensätze in die Datenbanken laden, welche in den folgenden Kapiteln verwendet werden (vgl. Abbildung 5.2).

Eine Übersicht über das Datenmodell kann der Abbildung 5.1 entnommen werden. In der Klasse „RelDok2“ befindet sich die Main-Methode und diese verwaltet den Ablauf der Migration. Die Migration der Daten von einer relationalen zu einer dokumentenorientierten Datenbank erfolgt über ein weiteres GUI-Element und einem entsprechenden SchemaConverter, der das relationale Schema umwandelt. Für die Migration der Daten zwischen den dokumentenorientierten Datenbanken ist der JSONConverter verantwortlich. Die jeweiligen Klassen zu den Datenbanken realisieren die Kommunikation zu diesen und erfassen die Informationen zu den Funktionalitäten. Das Package „information“ enthält jene Klassen, die die relevanten Informationen zu den Funktionalitäten speichern. Über die Rel(Dok)^2 werden die Informationen zu den Indizes, sowie Benutzer/Rollen erhalten und an die entsprechende Datenbank-Klasse weitergeleitet, welche diese dann umsetzen kann. Im folgenden werden die verschiedenen Komponenten des Modells genauer beleuchtet.

Die GUI der Rel(Dok)^2 kann der Abbildung 5.3 entnommen werden. Bei der Migration von der PostgreSQL zur MongoDB muss die Struktur der Daten verändert werden. Dazu gibt die Rel(Dok)^2 einen Vorschlag (vgl. Abbildung 5.4), welcher durch den Anwender angepasst werden kann.

Zwischen den dokumentenorientierten Datenbanken ist der Migrationsprozess einfacher gestaltet, da nicht das komplette Schema verändert werden muss. Der Ablauf eines Migrationsprozesses von der MongoDB zur ArangoDB kann dem Sequenzdiagramm 5.5 entnommen werden. Dies stellt den Ablauf einer Migration mit den Indizes und den Benutzern/Rollen dar. Zunächst werden die Daten aus der MongoDB exportiert. Diese müssen an die Gegebenheiten der ArangoDB angepasst werden. Anschließend können die Daten aus der JSON-Datei in die ArangoDB importiert werden. Die Indizes müssen zunächst in der MongoDB erfasst und ausgegeben werden, damit diese in die ArangoDB eingele-

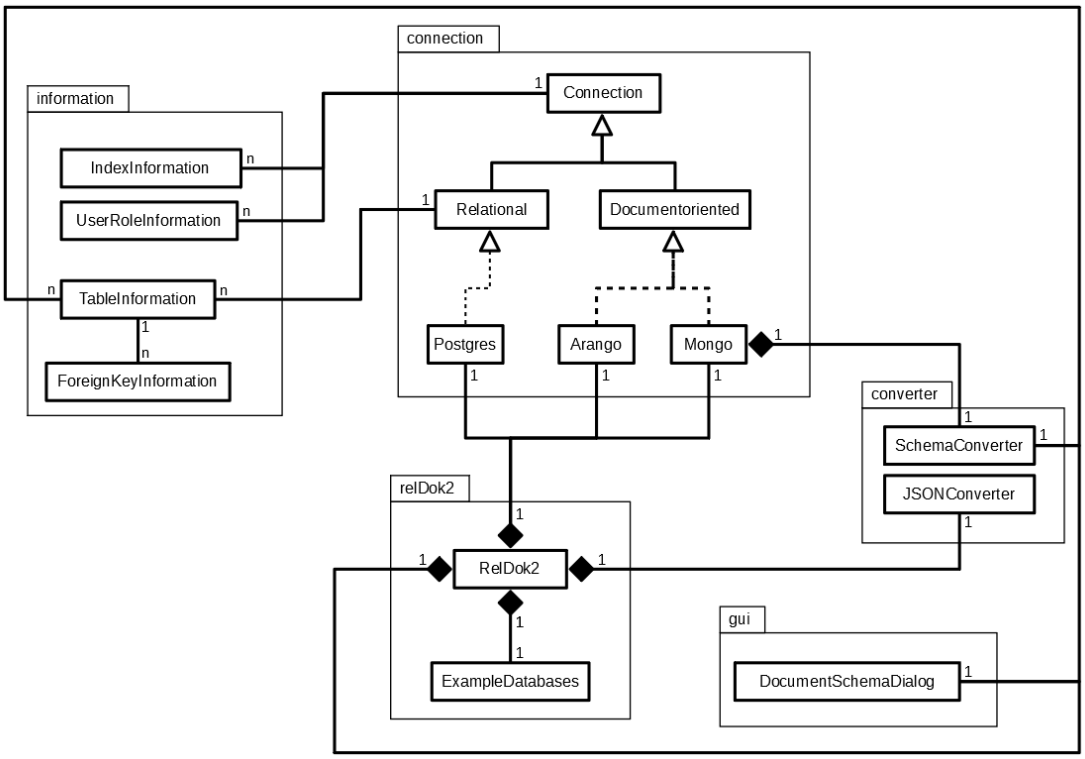


Abbildung 5.1: Erweitertes Klassendiagramm zur Rel(Dok)²

Abbildung 5.2: Option die Beispieldatensätze der Rel(Dok)² zu laden

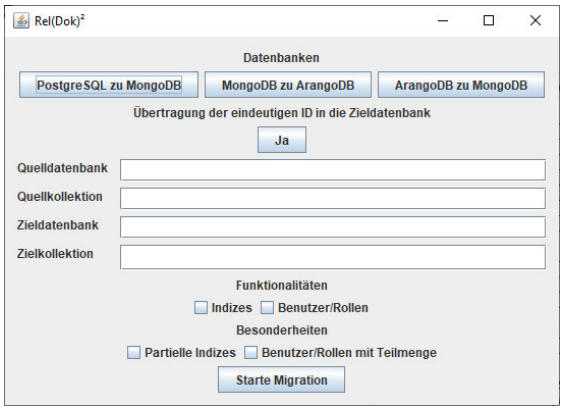


Abbildung 5.3: Benutzeroberfläche für die Rel(Dok)²

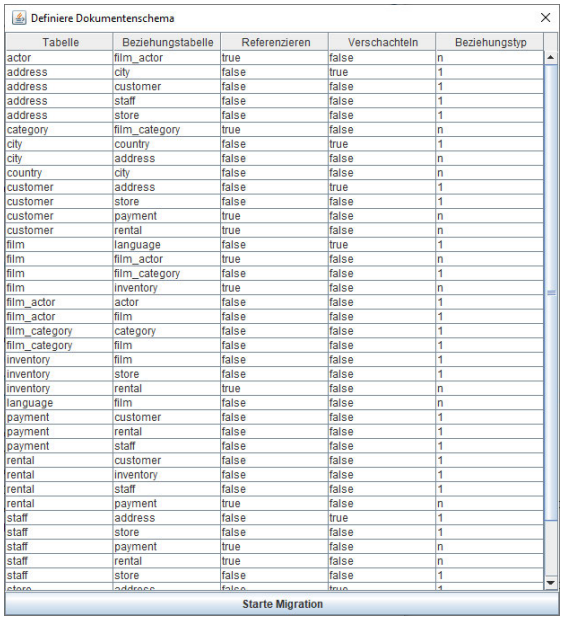


Abbildung 5.4: Strukturvorschlag für die dokumentenorientierte Datenbank

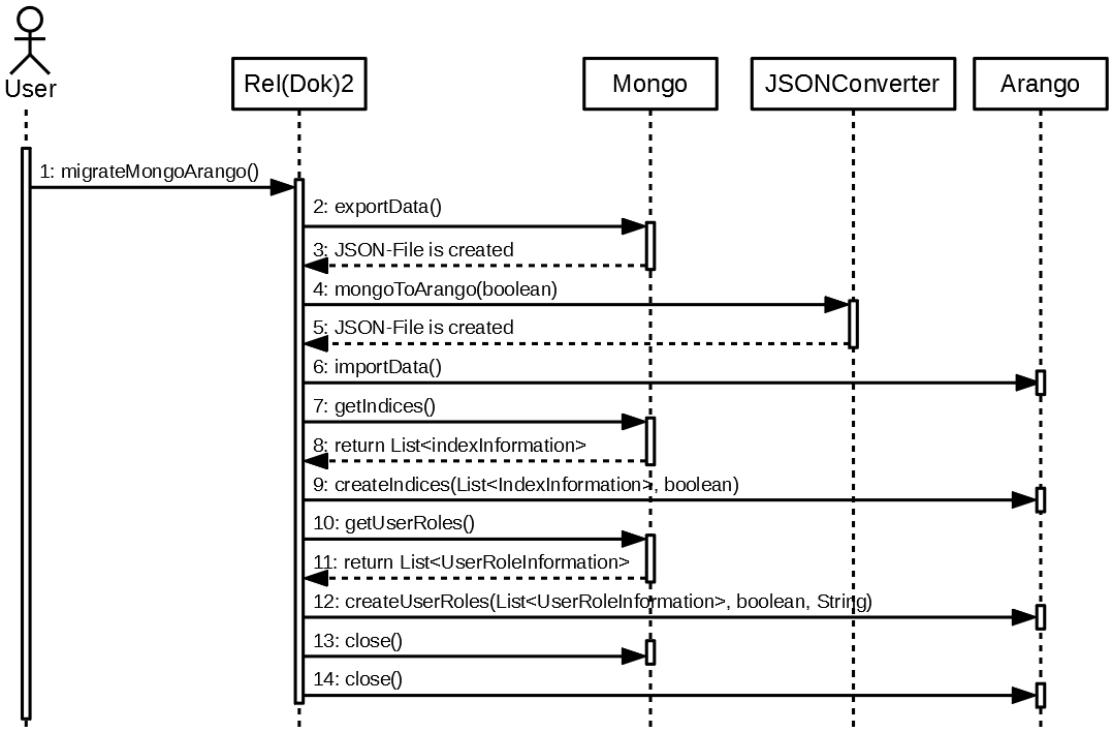


Abbildung 5.5: Sequenzdiagramm für die Migration von der MongoDB zur ArangoDB

sen und umgesetzt werden können. Der gleiche Prozess erfolgt für die Benutzer/Rollen. Abschließend werden die Verbindungen zu den Datenbanken geschlossen.

Bei der Migration von der relationalen zur dokumentenorientierten Datenbank, sind zusätzlich noch weitere Schritte nötig, um das neue Schema der Daten zu erstellen und dem Anwender zur Verifizierung zur Verfügung zu stellen. Dieser Prozess ist in dem Sequenzdiagramm 5.6 dargestellt. Zunächst werden die Tabellen in einem CSV-Format aus der Datenbank exportiert. Anschließend werden relevante Informationen zu den Tabellen und Beziehungen erfasst. Die Informationen zu den Beziehungen werden in einem zweiten Schritt vervollständigt, sodass für jede Tabelle die Beziehungen mit der Kardinalität hinterlegt sind. Anschließend kann basierend darauf, der Vorschlag des neuen Schemas für den Anwender erstellt werden. Wird in dem neuen Fenster das Schema akzeptiert, werden die Daten danach in die MongoDB eingelesen. Bei diesem Prozess wurden ebenfalls die Indizes, sowie Benutzer/Rollen migriert. Diese Schritte verlaufen analog zur Migration zwischen zwei dokumentenorientierten Datenbanken.

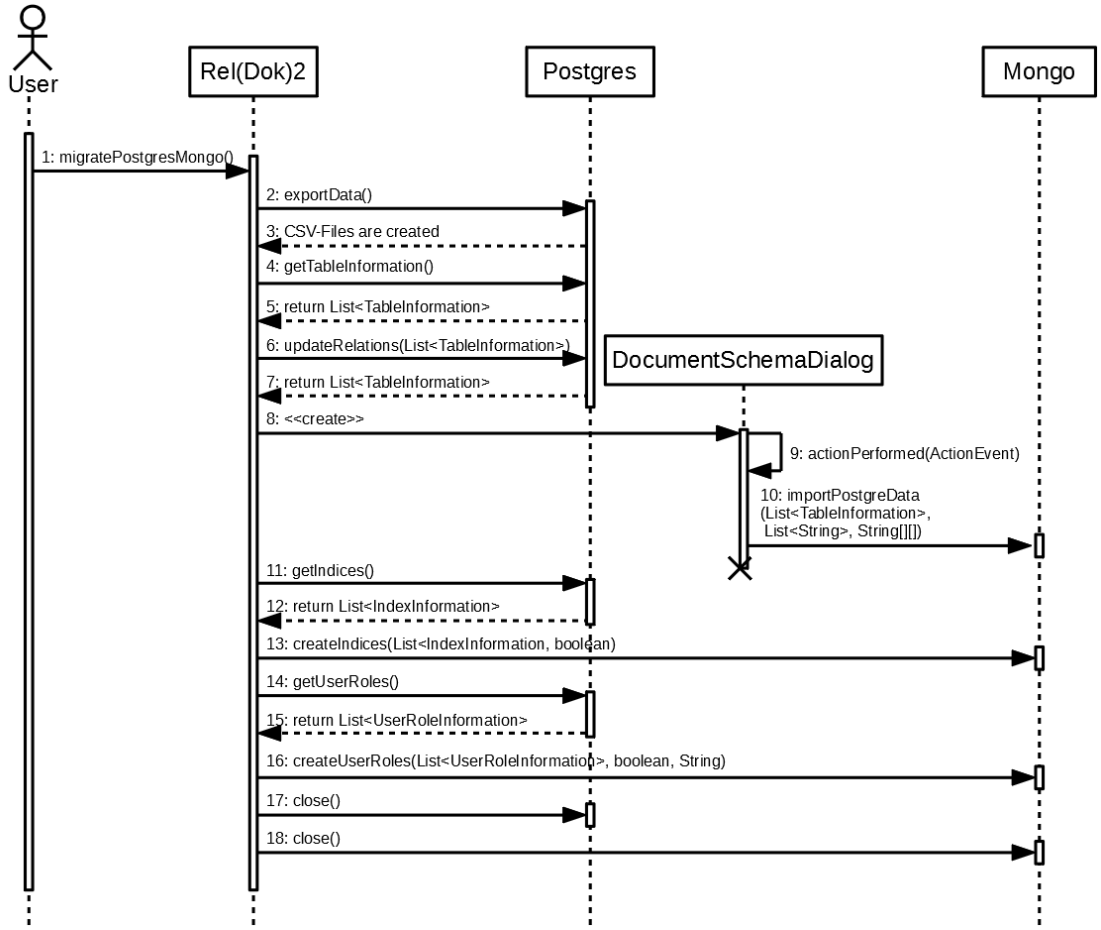


Abbildung 5.6: Sequenzdiagramm für die Migration von der PostgreSQL zur MongoDB

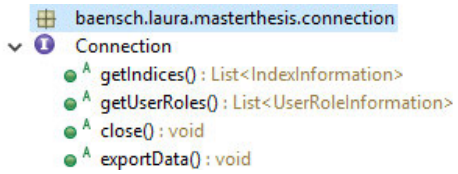


Abbildung 5.7: Interface Connection

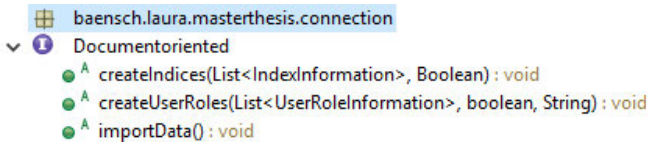


Abbildung 5.8: Interface Documentenoriented

Um eine Erweiterbarkeit nicht auszuschließen, wurden über Interfaces Methoden definiert, welche eine relationale bzw. dokumentenorientierte Datenbank implementieren müssen. Datenbankunabhängig müssen die Indizes, die Benutzer/Rollen, sowie die Daten exportierbar und die Verbindung geschlossen werden können, welches im Interface Connection spezifiziert wird (vgl. Abbildung 5.7).

Die dokumentenorientierten Datenbanken erweitern dies um den Import der Daten, sowie die Erstellung von Indizes und Benutzern/Rollen mit dem Interface Documentenoriented (vgl. Abbildung 5.8).

Die relationalen Datenbanken benötigen andere Methoden, da keine Daten eingelesen werden müssen, dies wird über das Interface Relational spezifiziert (vgl. Abbildung 5.9). Dafür ist eine Liste mit allen Tabellennamen notwendig, um die weiteren Schritte der Migration durchzuführen. Ebenso sind alle relevanten Informationen zu den Tabellen wichtig, wie z.B. zu den Spaltennamen, den Primär-/Fremdschlüssen usw. Um diese Informationen für die dokumentenorientierten Datenbanken auswertbar zu machen, werden die TableInformation (vgl. Abbildung 5.10) verwendet. Für die Verwaltung der Informationen zu den Beziehungen dienen die ForeignKeyInformationen (vgl. Abbildung 5.11). Des Weiteren müssen nach dem initialen Sammeln der Informationen, die Beziehungen

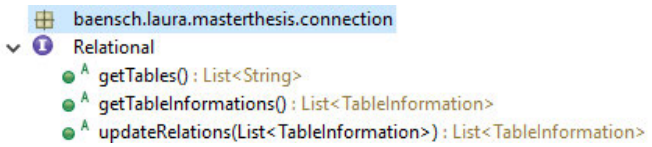


Abbildung 5.9: Interface Relational

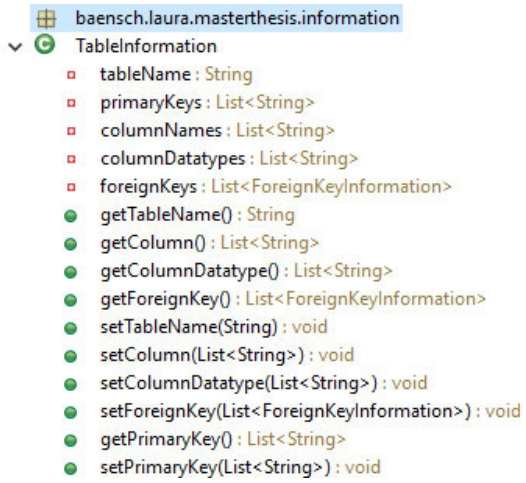


Abbildung 5.10: Klasse TableInformation

zwischen den Tabellen aktualisiert werden, damit diese in beide Richtungen für jede Tabelle hinterlegt sind. Dies ist für das Erstellen des neuen Schemas und der Darstellung aller möglichen Beziehungen in der GUI notwendig, dem DocumentSchemaDialog (vgl. Abbildung 5.12).

Insgesamt benötigt die Rel(Dok)² für jede Datenbank eine Klasse, welche die Verbindung zu dieser und die nötigen Methoden zur Migration der Daten und Funktionalitäten bietet. Diese implementieren die in den Interface definierten Methoden und ergänzen diese um interne Methoden, damit diese realisiert werden können. Detailliertere Informationen zu den jeweiligen Methoden können den Kommentaren im Quellcode entnommen werden. Im folgenden wird auf den Entwurf zur Migration der Daten, sowie den Funktionalitäten Indizes und Benutzer/Rollen eingegangen.

5.2.1 Migration der Daten

Bei der Migration der Daten ist der Vorgang stark von der Quell- und Zieldatenbank abhängig. Für relationale Datenbanken muss das Schema komplett neu erstellt werden, um effizient in der dokumentenorientierten Datenbank zu arbeiten. Bei der Migration zwischen verschiedenen dokumentenorientierten Datenbanken ist auf die unterschiedlichen Datentypen zu achten. Im folgenden werden beide Szenarien genauer erläutert.

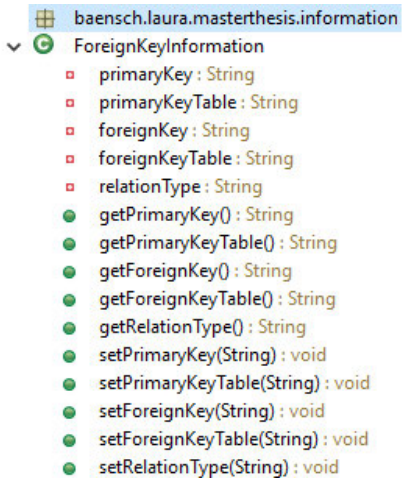


Abbildung 5.11: Klasse ForeignKeyInformation

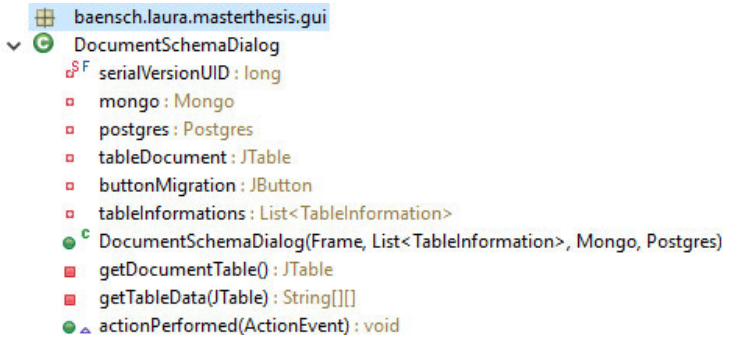


Abbildung 5.12: Klasse DocumentSchemaDialog

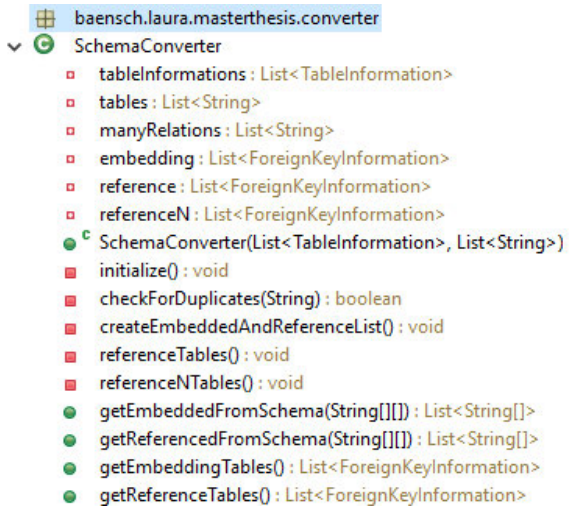


Abbildung 5.13: Klasse SchemaConverter

Relationale Datenbanken

Die Rel(Dok)² stellt dem Anwender automatisiert ein Schema für die Migration vor. Dies basiert auf dem Basiskonzept aus Kapitel 3.2. Die Umwandlung des Schemas erfolgt durch den SchemaConverter (vgl. Abbildung 5.13). Dem Anwender zugänglich gemacht wird es durch den DocumentSchemaDialog (vgl. Abbildung 5.12). Dazu werden die Beziehungen in der relationalen Datenbank erfasst und ausgewertet, welche die Kriterien für eine Verschachtelung und welche für eine Referenz erfüllen. Dieses Schema kann der Anwender der Rel(Dok)² der GUI entnehmen. Ebenfalls kann er darüber individuelle Anpassungen an dem vorgeschlagenen Schema vornehmen (vgl. Abbildung 5.4). Diese Tabelle weist alle wichtigen Informationen zu den in Beziehung stehenden Tabellen, sowie deren Kardinalität der Beziehung auf. Durch die Anpassung der Wahrheitswerte kann entschieden werden, welche Beziehung verschachtelt, referenziert oder ausgelassen werden soll.

Anschließend werden zunächst die zu referenzierenden und die zu überspringenden Tabellen in die dokumentenorientierte Datenbank übertragen. Im nächsten Schritt werden die zu verschachtelnden Tabellen in die bereits vorhandenen hinzugefügt. Tabellen, welche keine Veränderung benötigen, können direkt, von der CSV-Datei ausgehend, in die MongoDB importiert werden.

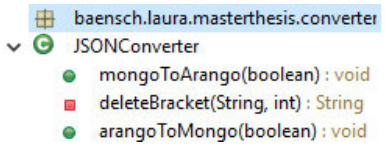


Abbildung 5.14: Klasse JSONConverter

Dokumentenorientierte Datenbanken

Bei der Migration der Daten ist es dem Anwender freigestellt, die _id bzw. den _key als primären, eindeutigen Schlüssel zu übernehmen. Obwohl es sich um Datenbanken des gleichen Modells handelt, müssen die exportierten Daten angepasst werden. Dies erfolgt durch den JSONConverter (vgl. Abbildung 5.14).

Für alle Datenbanken muss der eindeutige Schlüssel an den datenbankspezifischen Bezeichner angepasst werden, falls der Anwender diese Option ausgewählt hat. Für die Migration zur ArangoDB ist ebenfalls das Entfernen der Operatoren der MongoDB notwendig, da diese von anderen Datenbanken nicht interpretiert werden können. Bei der Entfernung der expliziten Datentypen ergeben sich programmatisch zwei Optionen. Zum einen kann die Verschachtelung rekursiv ausgewertet und Operatoren identifiziert und entfernt werden. Alternativ wird der Datensatz nach der typischen, einleitenden Sequenz ({“$) gesplittet. Dabei muss berücksichtigt werden, dass Operatoren geschachtelt vorliegen können und dementsprechend die schließenden Klammern ebenfalls entfernt werden müssen. Es wurde die zweite Alternative gewählt, denn das Dokument rekursiv auszuwerten ist besonders bei tiefen Verschachtelungen unpraktikabel. Durch das Splitten nach der Sequenz, kann der Datensatz unkompliziert verändert und wieder zusammen gesetzt werden.

Für die Migration zur MongoDB werden die Felder für die interne Verwaltung in der ArangoDB entfernt. Ebenso wird das Datum, automatisiert durch die Verwendung eines Regex, erkannt, wenn es das für die MongoDB typische Schema <YYYY-mm-dd THH:MM:ss.sssZ> aufweist. Zahlen, welche nicht als String hinterlegt sind, werden von der MongoDB intern erkannt und als Zahl dargestellt. Zahlen, welche als String vorliegen, werden nicht verändert, da nicht identifizierbar ist, ob dies beabsichtigt wurde, um eine bestimmte Genauigkeit sicherzustellen oder durch den Migrationsprozess entstanden ist.

Abbildung 5.15: Darstellung der VelocyPack Datentypen

Die ArangoDB bietet mit dem VelocyPack eine Möglichkeit, andere Datentypen einzulesen. Z.B. kann das Datum direkt über die Funktion „new Date()“ erstellt und muss nicht als String hinterlegt werden. Jedoch funktionieren darauf keine Indizes oder Abfragen, da die Daten unbestimmt in der Datenbank hinterlegt sind (vgl. Abbildung 5.15). Bei der ArangoDB wurde keine Lösung gefunden, bei der das Bytearray wieder in einen auslesbaren Zustand gebracht werden konnte, nachdem es über das VelocyPack eingelesen wurde. Dadurch, dass über das VelocyPack angelegte Datentypen nicht mehr weiterverwendbar sind, weder über die UI, noch mittels AQL, ist es nicht praktikabel, diese zu verwenden. AQL unterstützt nur die primitiven Datentypen wie JSON[1]. Über das VelocyPack können ebenfalls explizit die JSON-Datentypen angelegt werden. Da die ArangoDB diese jedoch automatisiert erkennen kann, ist es nicht notwendig, das VelocyPack zu verwenden. Insgesamt wird das VelocyPack für die Migration in die ArangoDB nicht genutzt, da es für diesen Zweck keine Vorteile bietet.

[1] https://www.arangodb.com/docs/stable/aql/fundamentals-data-types.html

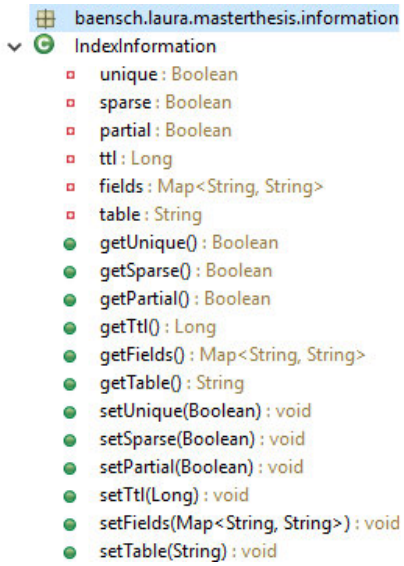


Abbildung 5.16: Klasse IndexInformation

5.2.2 Migration der Indizes

Um zwischen den verschiedenen Datenbanken die Informationen zu den Indizes auszutauschen, wurde die Klasse IndexInformation erstellt und verwendet (vgl. Abbildung 5.16). Dort sind alle relevanten Eigenschaften hinterlegt, welche in den drei Datenbanken migrierbar sind. Für relationale Datenbanken ist die Angabe der Tabelle zusätzlich nötig, bei dokumentenorientierten reicht die Verwendung von „fields“, welche ebenfalls für die Spalten der Tabellen verwendet wird. Die jeweiligen Datenbanken können eine Liste von IndexInformations erstellen, welche die gesammelten Informationen der zugrundeliegenden Indizes beinhaltet. Entsprechend werden diese wieder ausgewertet, um die Indizes anzulegen. Wird von der relationalen Datenbank zur dokumentenorientierten migriert, so wird intern eine Liste angelegt, welche die Pfade der verschachtelten Tabellen/Dokumente beinhaltet, damit der Index entsprechend in der Verschachtelung angelegt werden kann.

5.2.3 Migration der Benutzer/Rollen

Da die Übereinstimmung von Berechtigungen/Aktionen der verschiedenen Datenbanken relativ gering ist, wird bei der Migration erfasst, welche migrierbar sind und nur diese werden übertragen. Wird bei der Migration ausgewählt, dass bereits eine Teilmenge von

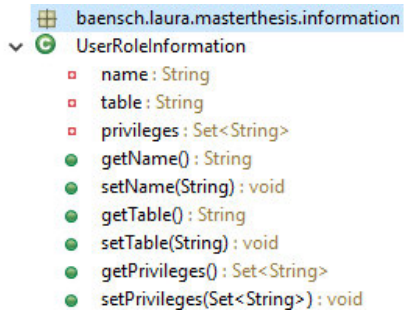


Abbildung 5.17: Klasse UserRoleInformation

Berechtigungen/Aktionen ausreicht, um in der Zieldatenbank erstellt zu werden, erhält der Benutzer mehr Zugriffsrechte als zuvor. Für relationale Datenbanken bedeutet dies, dass darüber dem Benutzer mehr Rechte zugewiesen werden können, falls Berechtigungen für die verschachtelten Dokumente nicht vorhanden sind. Bei der dokumentenorientierten ArangoDB kann dem Benutzer jeweils ein Set an Berechtigungen zugewiesen werden. Sind in der Quelldatenbank nicht alle nötigen Berechtigungen dazu vorhanden, kann über diese Option dennoch der Benutzer migriert werden. Generell werden nur jene Benutzer/Rollen migriert, welche mit der Quell-Kollektion oder -Datenbank verbunden sind. Alle weiteren Benutzer/Rollen werden nicht berücksichtigt. Der Austausch der Informationen zu diesen wird über die Klasse UserRoleInformation realisiert (vgl. Abbildung 5.17). Da bei der dokumentenorientierten Datenbank jeweils eine Kollektion migriert wird, sind neben der Angabe des Namens und den Berechtigungen keine weiteren Informationen nötig. Bei den relationalen Datenbanken wird die komplette Datenbank migriert, somit ist die Angabe der Tabelle erforderlich.

5.2.4 Auswertung der Anforderungen

Im Entwurf wurden alle funktionalen Anforderungen und darüber hinaus einige nichtfunktionale Anforderungen berücksichtigt. Die Rel(Dok)² kann Indizes und Benutzer/Rollen migrieren und erfüllt damit die Anforderung N-2. Durch die Verwendung von Interfaces werden Methoden definiert, welche eine implementierende Klasse realisieren muss und ermöglicht dadurch das Konzept der Modularität, welches in Anforderung N-9 definiert wurde. Anforderung N-4 wird mit der Rel(Dok)² als Vermittler zwischen den verschiedenen Datenbanken erfüllt. Bereits im Konzept wird erfasst, dass das Tool optisch einfach gestaltet sein soll, mit allen Auswahlmöglichkeiten in einem Fenster. Damit sind Anforderungen N-7 und N-6 gegeben, da der Anwender wenig Auswahlmöglichkeiten besitzt und

nur simple Angaben zu den Datenbanken geben muss. Damit ist der Migrationsprozess mit der Rel(Dok)^2 einfach zu erlernen.

5.3 Implementierung und Test

Die Implementierung ist analog zu dem im vorherigen Kapitel 5.2 vorgestellten Entwurf erfolgt. Der komplette Quellcode der Rel(Dok)^2 kann dem Anhang entnommen werden. Der Anhang zur Arbeit befindet sich auf CD und kann beim Erstgutachter eingesehen werden.

5.3.1 Besonderheiten bei der Implementierung

Abweichend vom Konzept müssen bei der ArangoDB die Berechtigungen explizit auf „none“ gesetzt werden, denn die implizite Verwendung durch den Defaultwert führt zu falschen Ergebnissen. Statt dem Defaultwert der entsprechenden Kollektion wird der Wert anderer Kollektionen in der Datenbank zurückgegeben. Das Problem konnte nur behoben werden, indem explizit für die jeweilige andere Beispieldatenbank die Berechtigungen auf „none“ gesetzt wurden. Wird nun ein anderer Datensatz verwendet, so wird durch die Rel(Dok)^2 der Defaultwert auf „none“ gesetzt. Es wird nicht explizit für alle Kollektionen die Berechtigung auf „none“ gesetzt. So können bei einer weiteren Migration die Rechte falsch ausgelesen und die Defaultwerte nicht erkannt werden.

5.3.2 Testung der Rel(Dok)^2

Um die grundlegenden Funktionen der Rel(Dok)^2 zu überprüfen, wurden JUnit Tests definiert. Diese liegen für den JSONConverter vor, welcher die exportierten Daten von der ArangoDB oder der MongoDB an die jeweilige Zieldatenbank anpasst. Dazu wurde die Option, die eindeutige ID aus der Quelldatenbank zu übernehmen, ebenfalls berücksichtigt. Des Weiteren wurden die Klassen Arango, Mongo und Postgres, welche für die Verbindung zu der Datenbank und der Umsetzung der Funktionalitäten verantwortlich sind, getestet. Diese Klassen implementieren die jeweiligen Interfaces und die dort definierten Methoden wurden über die Tests auf ihre korrekte Funktionsweise überprüft. Bei den dokumentenorientierten Datenbanken werden alle Index-Typen einzeln getestet,

ob diese korrekt erkannt und entsprechend umgesetzt werden, vorausgesetzt der Index-Typ wird von der Datenbank unterstützt. Ebenso müssen Benutzer/Rollen entsprechend ihrer Berechtigungen ausgewertet und angelegt werden. Bei relationalen Datenbanken muss getestet werden, ob alle Tabellen als eine Liste ausgegeben werden können und die gesammelten Informationen zu den Tabellen als eine Liste zurück gegeben werden. Datenbankmodell unabhängig wird getestet, ob die entsprechenden Listen zu den Indizes bzw. Benutzer/Rollen mit allen relevanten Informationen zu diesen korrekt erfasst und zurück gegeben werden. Alle JUnit Tests können erfolgreich abgeschlossen und durch den Quellcode rekonstruiert werden.

Der komplette Migrationsprozess wurde mit den Beispieldatensätzen getestet und die Auswertung dazu kann Kapitel 5.5 entnommen werden.

5.3.3 Auswertung der Anforderungen

Alle technischen Anforderungen bilden die Grundlage für die Implementierung des Tools, welche die Software und Versionen festlegen. Die bereits im Entwurf erfüllten Anforderungen (vgl. Kapitel 5.2.4) wurden bei der Implementierung ebenfalls umgesetzt. Dazu gehören alle funktionalen und die nicht-funktionalen Anforderungen N-2, N-4, N-6, N-7 und N-9. Darüber hinaus erfüllt die Implementierung weitere nicht-funktionale Anforderungen. Zur Rel(Dok)^2 wird begleitend eine ReadMe-Datei gestellt, welcher zu entnehmen ist, was das Tool bietet und wie es zu verwenden ist. Damit wird die Anforderung N-5 erfüllt und der Anwender kann erfassen, ob das Tool seinen Zwecken dient. Wie bereits im vorherigen Kapitel 5.3.2 diskutiert, sind Softwaretests zu den wichtigsten Funktionen der Rel(Dok)^2 vorhanden, welches Anforderung N-11 definiert. Anforderung N-10 wird mit der Testung durch Beispielszenarien in den folgenden zwei Kapiteln (vgl. Kapitel 5.4 und Kapitel 5.5) ausführlich behandelt. Damit ist ebenfalls Anforderung N-1 gegeben, da darüber rekonstruierbar nachvollzogen werden kann, ob die Migration erfolgreich war. Ebenso ist Anforderung N-8, welche festlegt, dass die Rel(Dok)^2 Interessierten zugänglich gemacht werden kann, sichergestellt, welches am Anfang des Kapitels bereits erwähnt wird (vgl Kapitel 5.3).

Weder im Entwurf noch in der Implementierung ist Anforderung N-3 festgelegt, welche besagt, dass die Rel(Dok)^2 neben anderen Programmen ausgeführt werden kann. Dazu sind keine expliziten Mechanismen umgesetzt worden, dennoch kann das Tool neben der Verwendung von Datenbanken oder anderen Programmen ausgeführt werden.

5.4 Beispieldatensätze

Um die Migration evaluieren zu können, sind Beispieldatensätze nötig, welche im folgenden vorgestellt werden. Für jede der dokumentenorientierten Datenbanken wird ein individueller, einfach gehaltener Datensatz erstellt. Dieser dient dazu, gezielt die migrierbaren Funktionalitäten zu überprüfen. Darunter zählen die Indizes, Benutzer bzw. Rollen, sowie die Übernahme der Datentypen. Außerdem wird ein komplexer, an die Realität angelehnter Datensatz hinzugefügt, welcher die realisierten Konzepte in der Rel(Dok)² auf eine allgemeine Nutzbarkeit testet.

5.4.1 PostgreSQL

Für PostgreSQL wurde ein realitätsnaher Datensatz ausgewählt und um eine unabhängige Tabelle erweitert. Dieser dient dazu, die automatische Erstellung des Schemas zu verifizieren. Für die Funktionalitäten wurden die bereits vorhandenen Indizes verwendet und der Datensatz um zwei Benutzer ergänzt, um das Konzept der Benutzer bzw. Rollen zu überprüfen.

Datensatz

Als realitätsnaher Datensatz wurde für die PostgreSQL vom PostgreSQL Tutorial[2] der Datensatz „dvdrental“ ausgewählt. Dieser besteht aus 15 Tabellen und 1:n- sowie n:m-Beziehungen und einer komplexeren Verflechtung der Tabellen untereinander. Neben den Fremdschlüsseln sind auch Verweise auf andere Tabellen ohne Fremdschlüssel erfolgt. Die Struktur kann dem ER-Modell in Abbildung 5.18 entnommen werden, welches auf der Ermittlung der Struktur von der Rel(Dok)² basiert.

Um zu überprüfen, ob alle Datentypen, welche nicht als String migriert werden, von der Rel(Dok)² identifiziert werden, wurde eine extra Tabelle „test“ im Datensatz angelegt, welche in keiner Beziehung zu den anderen Tabellen steht. Deswegen ist diese im ER-Modell nicht verzeichnet. Die Tabelle enthält 14 Spalten mit den Datentypen smallint, integer, bigint, numeric (decimal ist nicht vorhanden), real, double, smallserial, serial, bigserial, date, boolean, array von char, sowie json. Diese Datentypen sind prinzipiell in die MongoDB übertragbar.

[2] https://www.postgresqltutorial.com/postgresql-sample-database/

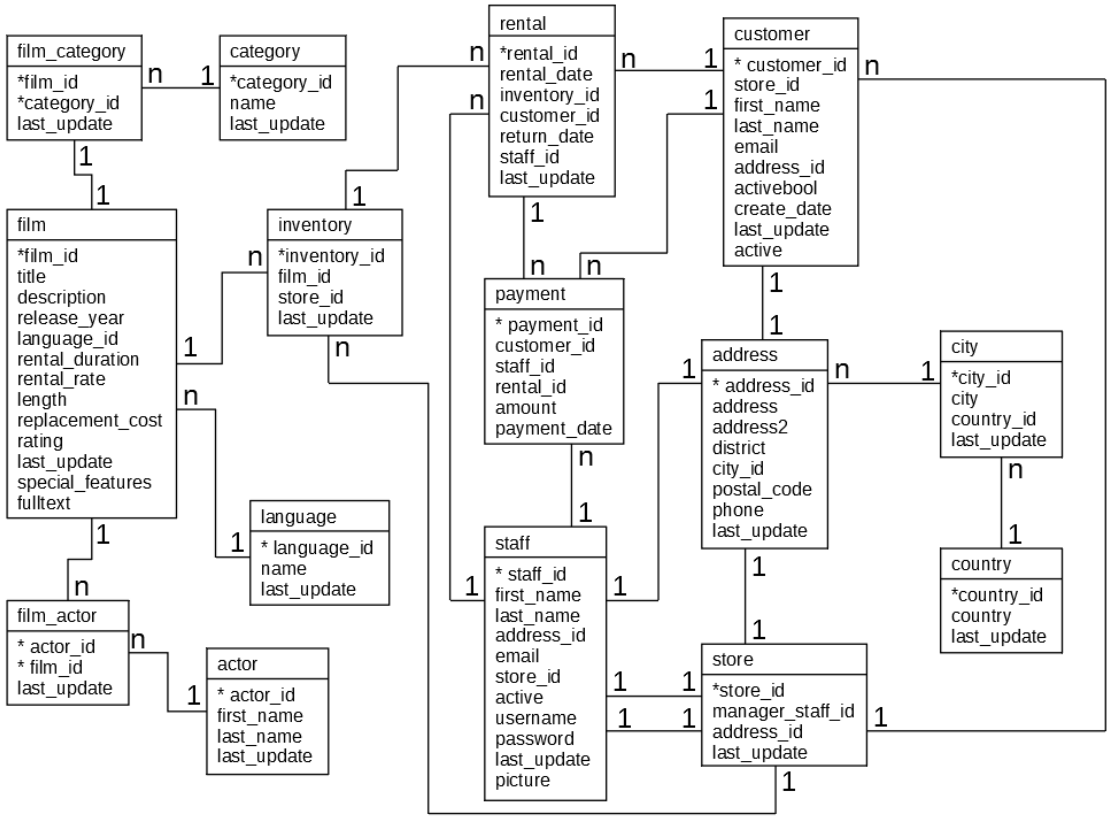


Abbildung 5.18: ER-Modell basierend auf den Informationen der Rel(Dok)²

Tabelle 5.1: Indizes in der PostgreSQL ohne Primärschlüssel-Indizes

Tabelle	Spalten	Eigenschaft
actor	last_name	
address	city_id	
city	country_id	
customer	address_id	
customer	store_id	
customer	last_name	
film	fulltext	text
film	language	
film	title	
film_actor	film_id	
inventory	store_id, film_id	
payment	customer_id	
payment	rental_id	
payment	staff_id	
rental	inventory_id	
rental	rental_date, inventory_id, customer_id	unique
store	manater_staff_id	unique

Indizes

Der Beispieldatensatz besitzt eine Reihe von Indizes, welche sich nicht nur auf Primär- und Fremdschlüssel beschränken. Darunter sind ein Text-Index und mehrere zusammengesetzte, welche in Tabelle 5.1 aufgelistet sind. Da die Index-Typen grundlegend unterschiedlich sind, ist es für die Migration irrelevant, welche verwendet wurden.

Benutzer/Rollen

Zu dem Beispieldatensatz wurden Benutzer bzw. Rollen zusätzlich angelegt. Die Zuweisung von Privilegien wurde so ausgewählt, dass bei einer Verschachtelung nicht für alle Tabellen die Berechtigungen vorhanden sind und der Benutzer nur unter Zustimmung migriert wird. Ebenso kann überprüft werden, ob eine Teilmenge der möglichen Rechte korrekt migriert wird, wenn diese Option ausgewählt wurde. Die Benutzer sind in Tabelle 5.2 verzeichnet.

Tabelle 5.2: Benutzer und Rollen in der PostgreSQL

Benutzer/Rolle	Privilegien	Tabellen
user1	update, delete	film, language
user2	insert, update, references	film, language, film_actor, film_category
postgres	alle	alle

5.4.2 MongoDB und ArangoDB

Für die MongoDB und die ArangoDB wird jeweils ein eigener, an die speziellen Gegebenheiten der Datenbank angepasster Beispieldatensatz erstellt. Ebenso gibt es einen individuellen, realitätsnahen Datensatz, da die MongoDB beim Import, Operatoren in einer JSON-Datei erkennen kann und diese in die entsprechenden Datentypen umwandelt. Dazu ist die ArangoDB nicht fähig. Die Funktionalitäten, Indizes und Benutzer bzw. Rollen werden im Beispieldatensatz so verwendet, dass diese ein breites Spektrum an möglichen Fällen abdecken.

Datensatz

Der simple Datensatz umfasst 10 Dokumente, von denen zwei ein Feld „date“ vorweisen, welches zur Überprüfung des TTL-Indexes dient. Arbeitet dieser korrekt und erkennt das Datum, sind im Beispieldatensatz lediglich 8 von den 10 Dokumenten vorhanden. Die ArangoDB erhält zusätzlich ein Dokument, welches mit dem VPack arbeitet und in dem alle Datentypen, außer den internen, spezifisch angelegt werden (vgl. Tabelle 4.1). Somit besitzt der Beispieldatensatz in der ArangoDB 9 von 11 Dokumenten. Im Folgenden ist beschrieben, wie der Beispieldatensatz aufgebaut ist:

{
“_id“:<String>, “_key“:<String>, *
“arrayField“:[<String>,<Integer>, ...],
“dateField“:<Long>, **
“date“:<String>, **
“dateBefore1970“:<Long>, **
“decimal128Field“:<String>, **
“doubleField“:<String>, **

“intField“:<Integer>,
“int32field“:<Integer>, **
“int64Field“:<Long>, **
“regexField“:{“pattern“:<String>,“options“:<String>}, **
“timestampField“:{“t“:<Integer>,“i“:<Integer>}, **
“location“:{“type“:<String>,“coordinates“:[<Double>, <Double>]}
}
* Die MongoDB verwendet “_id“, die ArangoDB “_key“.
** In der MongoDB sind diese Felder als String dargestellt und werden durch einen speziellen Datentyp-Operator in den Datentyp des Bezeichners gebracht. Die Repräsentation dieser Operatoren kann der MongoDB Extended JSON (v2)[3] entnommen werden.

Der realitätsnahe Datensatz wurde von Github entnommen. Für die MongoDB besitzt dieser Informationen zu 18.801 Unternehmen[4] und enthält die speziellen Datentyp-Operatoren. Für die ArangoDB besitzt dieser 43.991 Dokumente zu Flughäfen[5].

Indizes

Die ArangoDB arbeitet in der aktuellen Version mit der RocksDB Engine, welche nicht zwischen Hash-, Skiplist- und Persistenz-Indizes unterscheidet. Diese werden gesammelt als persistenter Index realisiert und nur für eine Kompatibilität versionsabwärts weiterhin unterstützt.

Für den simplen Datensatz wurden alle Index-Typen und -Kombinationen von Eigenschaften angelegt (vgl. Tabelle 5.3). Für die ArangoDB wird der partial Index ausgelassen, falls die Option der Migration von partiellen Indizes nicht ausgewählt wird. Dabei wird jedoch nur der Index, ohne die partielle Eigenschaft erstellt, da diese nicht unterstützt wird. Die MongoDB bietet keine große Varianz von Typen an, deswegen wird für die Indizes Skiplist und Persistent ein unspezifizierter Index-Typ angelegt, sowie für den partiellen.

Der realitätsnahe Datensatz für die MongoDB erhält einen Index auf ein Array, einen Text-Index, eine Kombination aus drei Feldern, sowie einen Index auf ein verschachteltes Feld. Die ArangoDB erhält auf den Flughafen Datensatz einen Geo-Index, welcher

[3] https://docs.mongodb.com/manual/reference/mongodb-extended-json/
[4] https://github.com/ozlerhakan/mongodb-json-files/blob/master/datasets/companies.json
[5] https://github.com/arangodb/example-datasets/tree/master/Airports

Tabelle 5.3: Indizes für den simplen Datensatz

Typ	Eigenschaft	Felder
Skiplist	unique	intField, int64field
Hash	sparse	decimal128Field
Persistent	unique, sparse	int32field
Fulltext	sparse	regexField
Geo	sparse	location
TTL	sparse	date
Index	partial	arrayField, Filter nach Strings

aus zwei Feldern besteht. Dieser ist für die MongoDB nicht migrierbar, da diese für Geo-Indexe auf genau einem Feld arbeitet. Des Weiteren wird ein Text-Index, sowie ein persistenter, mit einer Kombination aus drei Feldern, erstellt.

Benutzer und Rollen

Für die ArangoDB wurden insgesamt fünf Benutzer angelegt. Diese erhalten alle eine verschiedene Kombination der möglichen Berechtigungen (vgl. Kapitel 4.2.7) der drei Ebenen. Der Root-Benutzer wird direkt bei der Installation der Datenbank erstellt und erhält automatisch immer die höchst mögliche Berechtigung für alle drei Ebenen. Die vier weiteren Rollen erhalten in unterschiedlichen Kombinationen die Berechtigungen für die Datenbank- und Kollektionsebene (vgl. Tabelle 5.4).

Tabelle 5.4: Benutzerbeschreibung der ArangoDB

Benutzerbezeichner	Serverebene	Datenbankebene	Kollektionsebene
root	administrate	administrate	read/write
administrateRW	none	administrate	read/write
administrateRO	none	administrate	read only
accessRW	none	access	read/write
accessRO	none	access	read only

Bei der MongoDB ist das Berechtigungssystem komplexer und bietet individuellere Zuweisung von Rechten. Da hier der Anwender entscheiden kann, ob eine Migration der Benutzer bzw. Rollen bei einer Übereinstimmung von mindestens einer Aktion erfolgen soll,

werden die Rollenbeschreibungen zur MongoDB modifiziert. Es wurden für die MongoDB ebenfalls fünf Benutzer bzw. Rollen erstellt, welche äquivalent zu denen der ArangoDB sind. Um das Konzept der Zuweisung von Berechtigungen bereits bei mindestens einer Übereinstimmung von Aktionen zu überprüfen, wurde zu jeder Rolle eine Aktion hinzugefügt (vgl. Tabelle 5.5). Damit kann verifiziert werden, dass diese keine zusätzlichen Berechtigungen erhalten, wenn der Benutzer nicht ausgewählt hat, dass mindestens eine Aktion vorhanden sein muss. Andersherum erhalten diese Rollen je Ebene eine höhere Berechtigung, falls eine Teilmenge ausreicht.

Tabelle 5.5: Rollen-/Benutzerbeschreibung der MongoDB

Rollen-/Benutzerbezeichner	zusätzliche Aktionen	ArangoDB Äquivalent
administrator	-	root
databaseCollection	update user	administrateRW
databaseROCollection	create index	administrateRO
accessCollection	drop collection	accessRW
accessROCollection	insert	accessRO

Für den realitätsnahen Datensatz wurden jeweils zwei Benutzer mit Rollen angelegt. Die ArangoDB weist einen AirportsSuperUser (administrateRW) und einen AirportsUser (accessRO) auf, welche die Rechte entsprechend der Rolle aus Tabelle 5.4 besitzen. Die MongoDB erhält die Benutzer bzw. Rollen companiesSuperUser (actionsDatabaseCollection) und companiesUser (databaseROCollection) entsprechend Tabelle 5.5.

5.5 Ergebnisse

In diesem Kapitel werden die Ergebnisse mit den Beispieldatensätzen ausgewertet. Es wird untersucht, ob die Indizes, sowie Benutzer/Rollen korrekt übernommen und die Daten migriert wurden. Bei den Ergebnissen handelt es sich um eine exemplarische Auswertung, ob die Migration den Erwartungen entsprechend durchgeführt wurde. Dies stellt nicht sicher, dass in allen Anwendungsfällen die Migration korrekt erfolgen kann. Es dient einer Evaluierung, um eine Einschätzung zu geben, ob die Migration mit den Beispieldatensätzen korrekt arbeitet.

Alle Beispiele können mit der Rel(Dok)² nachvollzogen werden, wenn beim Start des Tools das Laden der Beispieldatensätze ausgewählt wird. Dort ist ebenfalls überprüfbar, ob

die im vorherigen vorgestellten Datensätze den Beschreibungen entsprechend vorhanden sind.

5.5.1 Migration von der PostgreSQL zur MongoDB

Im folgenden werden die Ergebnisse der Migration von der PostgreSQL zur MongoDB für die Daten, Indizes, sowie Benutzer bzw. Rollen ausgewertet. Verwendet wurde der realitätsnahe Datensatz „dvdrental"mit der zusätzlichen Tabelle „test“.

Daten

Die Migration der Daten erfolgt gemäß der Auswahl des Anwenders bezüglich der Verschachtelung und der Referenz. Insgesamt schlägt die Rel(Dok)² eine Verteilung von Referenz und Verschachtelung entsprechend der Tabelle 5.6 vor. Nach diesem Schema wurden die Dokumente in der MongoDB erstellt, sowohl Verschachtelung, als auch Referenz erfolgen über das Feld im Dokument, welches zuvor die ID oder IDs zu den entsprechenden Tabellen hinterlegt hatte. Beispielhaft kann ein Dokument der Tabelle „customer“ der Abbildung 5.19 entnommen werden. Die Migration des Schemas ist korrekt erfolgt, jedoch sind nicht alle Datentypen, welche über die CSV-Datei angelegt wurden, richtig erkannt worden (vgl. Abbildung 5.20). Um dieses Problem zu behandeln, wäre eine Auswertung der Datentypen und eine entsprechende Modifikation der Werte notwendig. Dies ist im aktuellen Zustand in der Rel(Dok)² nicht implementiert.

Tabelle 5.6: Schemavorschlag für die Migration von der Rel(Dok)²

Referenz	Verschachtelung	Überspringen
actor	address	test
categroy	city	payment
customer	country	film_actor
film	language	film_category
inventory		
rental		
staff		
store		

```
{
    "_id": {
        "$oid": "5f67c12dd4db6efbb4359acd"
    },
    "staff_id": 1,
    "first_name": "Mike",
    "last_name": "Hillyer",
    "address_id": {
        "address_id": 3,
        "address": "23 Workhaven Lane",
        "address2": "",
        "district": "Alberta",
        "city_id": {
            "city_id": 300,
            "city": "Lethbridge",
            "country_id": {
                "country_id": 20,
                "country": "Canada",
                "last_update": "2006-02-15 09:44:00"
            },
            "last_update": "2006-02-15 09:45:25"
        },
        "postal_code": "\"\"",
        "phone": "14033335568",
        "last_update": "2006-02-15 09:45:30"
    },
    "email": "Mike.Hillyer@sakilastaff.com",
    "store_id": 1,
    "active": "t",
    "username": "Mike",
    "password": "8cb2237d0679ca88db6464eac60da96345513964",
    "last_update": "2006-05-16 16:13:11.79328",
    "picture": "\\x89504e470d0a5a0a",
    "payment_id": [17505, 17508, 17504, 17517, 17518, 17519,
    "rental_id": [3, 6, 5, 9, 15, 17, 19, 23, 24, 26, 30, 2,
}
```


Abbildung 5.19: Beispieldokument der Tabelle „customer“

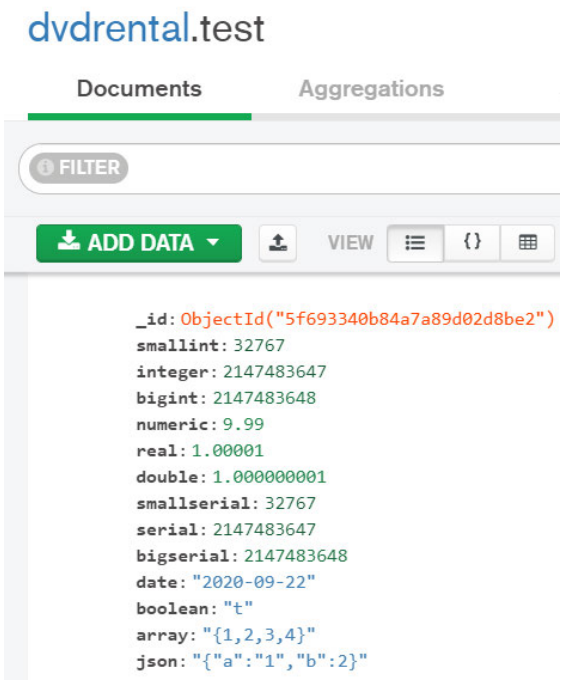


Abbildung 5.20: Beispieldokument der Tabelle „test“

Beispielabfragen

Da durch die Veränderung des Schemas bei der Migration sichergestellt werden muss, dass trotz Verschachtelung, die Daten korrekt übernommen wurden, werden die folgenden Abfragen spezifiziert.

- SELECT staff.staff_id FROM staff,address,city,country WHERE country.country_id 20 AND staff.address_id address.address_id AND address.city_id city.city_id AND city.country_id country.country_id;
- SELECT store.store_id FROM store,address,city,country WHERE country.country_id 8 AND store.address_id address.address_id AND address.city_id city.city_id AND city.country_id country.country_id;
- SELECT customer.customer_id FROM customer,address,city,country WHERE country.country_id 64 AND customer.address_id address.address_id AND address.city_id city.city_id AND city.country_id country.country_id;
- SELECT film.film_id FROM film JOIN film_category ON film_category.film_id film.film_id JOIN category ON category.category_id film_category.category_id WHERE category.category_id 6;
- SELECT film.film_id FROM film, inventory WHERE inventory.film_id film.film_id AND inventory.inventory_id 613;

Für die MongoDB müssen diese Abfragen entsprechend angepasst werden.

- staff: {“address_id.city_id.country_id.country_id“ : 20}
- store: {“address_id.city_id.country_id.country_id“ : 8}
- customer: {“address_id.city_id.country_id.country_id“: 64}
- category: {“category_id“ : 6}
- film: {“inventory_id“ : 613}

Werden die Abfragen in beiden Datenbanken durchgeführt, so wird in beiden Fällen der gleiche Datensatz angezeigt. Im Unterschied zur PostgreSQL weist der Datensatz der MongoDB weitere Informationen auf, da diese verschachtelt vorliegen.

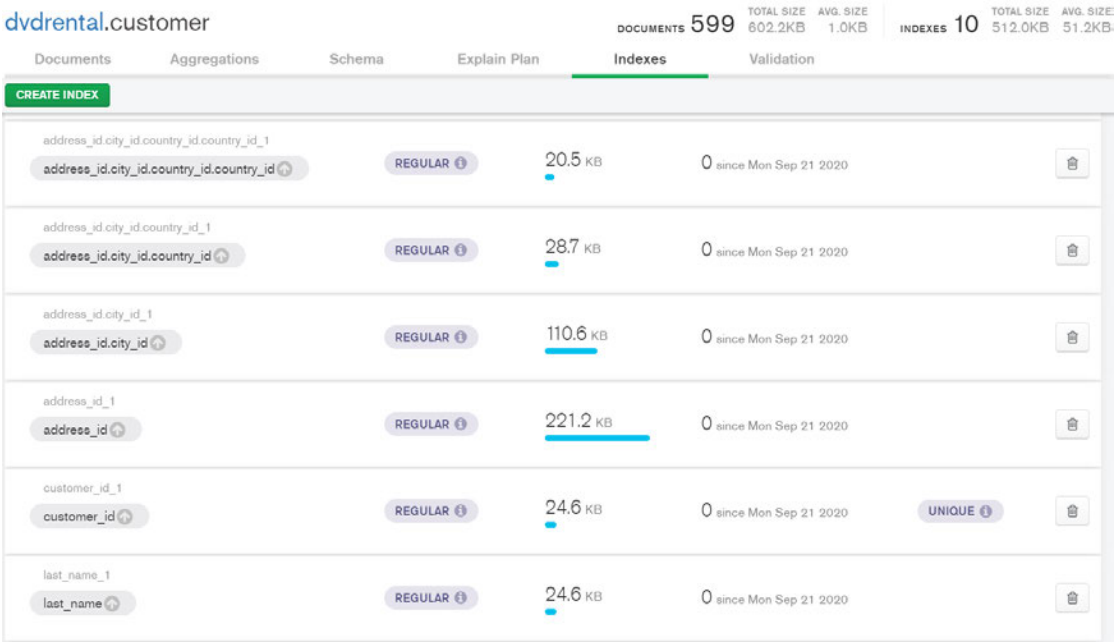


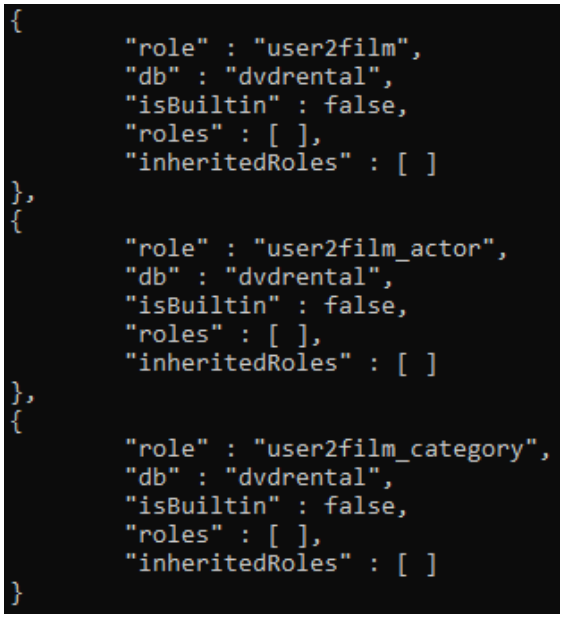
Abbildung 5.21: Auszug zu den Indizes in der Kollektion „customer“

Indizes

Die Indizes werden der Erwartung entsprechend übernommen. Tabellen, welche Indizes in der PostgreSQL besaßen, werden in die MongoDB übertragen, auch wenn diese verschachtelt vorliegen. In Abbildung 5.21 ist ein Auszug der Indizes der Kollektion „customer“ erkennbar. Hier sind Felder indiziert, welche direkt im Dokument oder bis zu dreifach verschachtelt sind. Analog ist dies für die restlichen Kollektionen erfolgt.

Benutzer/Rollen

Die Benutzer der PostgreSQL werden abhängig von der Auswahl des Anwenders migriert. Wird ausgewählt, dass eine Teilmenge bereits ausreicht, um den kompletten Benutzer zu migrieren, dann werden jene Benutzer/Rollen übertragen, welche durch das neue Schema mittels Verschachtelung oder Referenz auf mehr Daten Zugriff erhalten. Ist dies nicht gewünscht, werden nur die Benutzer migriert, welche die entsprechenden Berechtigungen für das komplette verschachtelte Dokument beinhalten. Abbildung 5.23 zeigt eine Migration mit Teilmenge, Abbildung 5.22 eine Migration, bei der nur jene Benutzer migriert wurden, welche die kompletten Berechtigungen für die Kollektion aufweisen.



```
{
        "role" : "user2film",
        "db" : "dvdrental",
        "isBuiltin" : false,
        "roles" : [ ],
        "inheritedRoles" : [ ]
},
{
        "role" : "user2film_actor",
        "db" : "dvdrental",
        "isBuiltin" : false,
        "roles" : [ ],
        "inheritedRoles" : [ ]
},
{
        "role" : "user2film_category",
        "db" : "dvdrental",
        "isBuiltin" : false,
        "roles" : [ ],
        "inheritedRoles" : [ ]
}
```


Abbildung 5.22: Zugehörige Rollen in der MongoDB zu dem Benutzer „user2“ ohne Teilmengen-Option

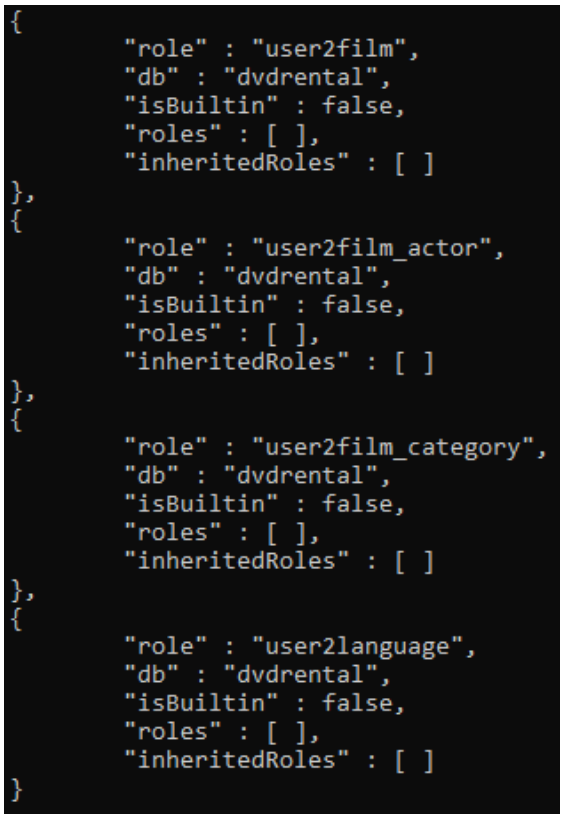
5.5.2 Migration von der MongoDB zur ArangoDB

Im folgenden werden die Ergebnisse bei der Migration von der MongoDB zur ArangoDB für die Daten, Indizes, sowie Benutzer und Rollen ausgewertet. Berücksichtigt wurde dafür der Beispieldatensatz und der realitätsnahe Datensatz „companies“.

Daten

Während des Migrationsprozesses werden alle expliziten Datentypen entfernt, wie der Abbildung 5.24 mit einem Dokument des Beispieldatensatzes entnommen werden kann. Ebenso ist die ObjectID der MongoDB als _key übernommen worden. Die von der ArangoDB automatisch angelegte _id setzt sich aus dem Namen der Kollektion und dem _key zusammen und identifiziert das Dokument innerhalb der Datenbank eindeutig. Diese _id ist nicht das Äquivalent der _id in der MongoDB, sondern dies ist der _key.

Wird bei der Migration die Auswahl getroffen, den eindeutigen Identifizierer zu übernehmen, liegen die ObjectIDs der MongoDB als _key in der ArangoDB vor. Alternativ werden diese entfernt und die ArangoDB legt eigene Keys an. Insgesamt ist die Migration



```
{
        "role" : "user2film",
        "db" : "dvdrental",
        "isBuiltin" : false,
        "roles" : [ ],
        "inheritedRoles" : [ ]
},
{
        "role" : "user2film_actor",
        "db" : "dvdrental",
        "isBuiltin" : false,
        "roles" : [ ],
        "inheritedRoles" : [ ]
},
{
        "role" : "user2film_category",
        "db" : "dvdrental",
        "isBuiltin" : false,
        "roles" : [ ],
        "inheritedRoles" : [ ]
},
{
        "role" : "user2language",
        "db" : "dvdrental",
        "isBuiltin" : false,
        "roles" : [ ],
        "inheritedRoles" : [ ]
}
```


Abbildung 5.23: Zugehörige Rollen in der MongoDB zu dem Benutzer „user2“ mit Teilmengen-Option

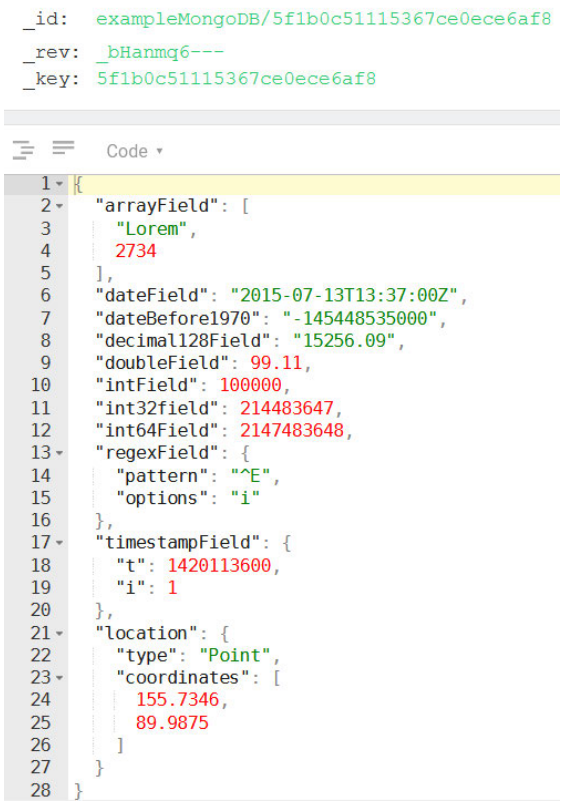


Abbildung 5.24: Detailansicht eines Dokuments aus dem Beispieldatensatz in der ArangoDB

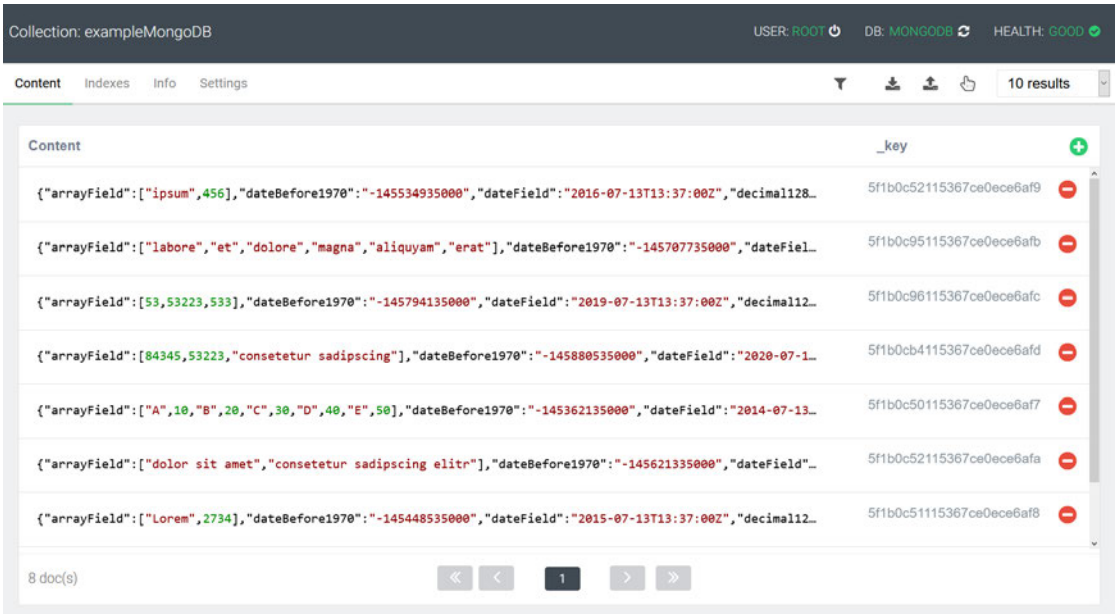


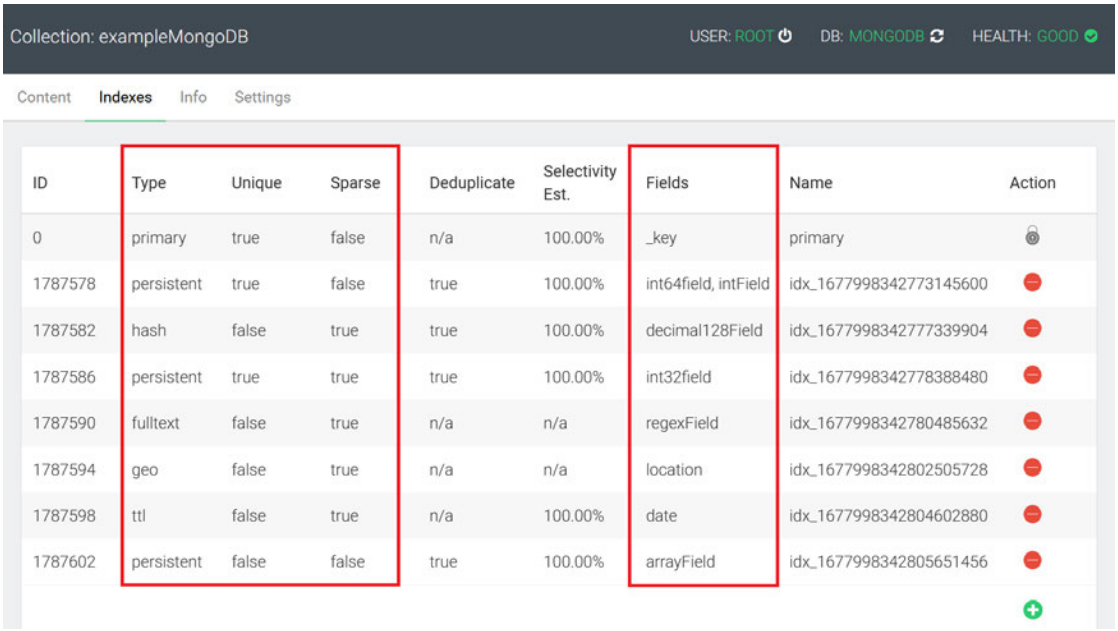
Abbildung 5.25: Dokumente in der ArangoDB

der Daten korrekt durchgeführt worden und alle 8 Dokumente des Beispieldatensatzes wurden migriert (vgl. Abbildung 5.25).

Bei der Migration des Datensatzes zu den Unternehmen wurden alle 18.801 Dokumente in die ArangoDB transferiert. Die im Datensatz verwendeten Operatoren für das Datum wurden durch die Rel(Dok)^2 erkannt und entsprechend entfernt, sodass diese als ein String dargestellt wurden.

Indizes

Die Index-Typen Hash und Skiplist werden nur versionsabwärts unterstützt. Diese werden stattdessen als persistenter Index realisiert. Bei der Migration des Hash-Index wird dieser in der ArangoDB dennoch als Hash-Index angelegt, da eine Kompatibilität noch gegeben ist. Wird bei der Rel(Dok)^2 die Migration von partiellen Indizes ausgewählt, so werden diese ohne die partielle Eigenschaft migriert. Insgesamt wurden die in der Abbildung 5.26 dargestellten Indizes übertragen, wobei die Option der Migration von partiellen Indizes ausgewählt wurde. Wird dies nicht gewünscht, würden alle Indizes, bis auf den persistenten Index auf dem Feld „arrayField“ migriert werden. Die in der Abbildung 5.26



ID	Type	Unique	Sparse	Deduplicate	Selectivity Est.	Fields	Name	Action
0	primary	true	false	n/a	100.00%	_key	primary	
1787578	persistent	true	false	true	100.00%	int64field, intField	idx_1677998342773145600	
1787582	hash	false	true	true	100.00%	decimal128Field	idx_1677998342777339904	
1787586	persistent	true	true	true	100.00%	int32field	idx_1677998342778388480	
1787590	fulltext	false	true	n/a	n/a	regexField	idx_1677998342780485632	
1787594	geo	false	true	n/a	n/a	location	idx_1677998342802505728	
1787598	ttl	false	true	n/a	100.00%	date	idx_1677998342804602880	
1787602	persistent	false	false	true	100.00%	arrayField	idx_1677998342805651456	

Abbildung 5.26: Indizes in der ArangoDB

dargestellten Indizes, entsprechen denen im Beispieldatensatz in Tabelle 5.3 vorgestellten Indizes, dementsprechend wurden diese korrekt migriert.

Benutzer/Rollen

Die MongoDB spezifiziert in der Rolle, für welche Kollektion die Berechtigungen gelten. Per Default erhalten die Benutzer in der ArangoDB keine Berechtigungen, sodass migrierte Rollen/Benutzer auf andere Kollektionen innerhalb der Datenbank keinen Zugriff besitzen. In Abbildung 5.27 sind exemplarisch die Berechtigungen des Benutzers „databaseCollection“ zu erkennen. Dieser ist dem Beispieldatensatz zugeordnet und besitzt die in der Tabelle 5.7 zugeordneten Berechtigungen. Für die Kollektion der Unternehmen ist der Default-Wert ausgewählt (welcher keine Berechtigungen darstellt) und ist grau unterlegt.

Wird zunächst nur der Beispieldatensatz migriert, so werden auch nur die Benutzer angelegt, welche Berechtigungen zu diesem besitzen (vgl. Abbildung 5.28). Die migrierten Benutzer sind rot umrandet, die anderen gehören dem Beispieldatensatz der ArangoDB an. Wird anschließend die Kollektion zu den Unternehmen migriert, werden die entsprechenden Benutzer hinzugefügt.

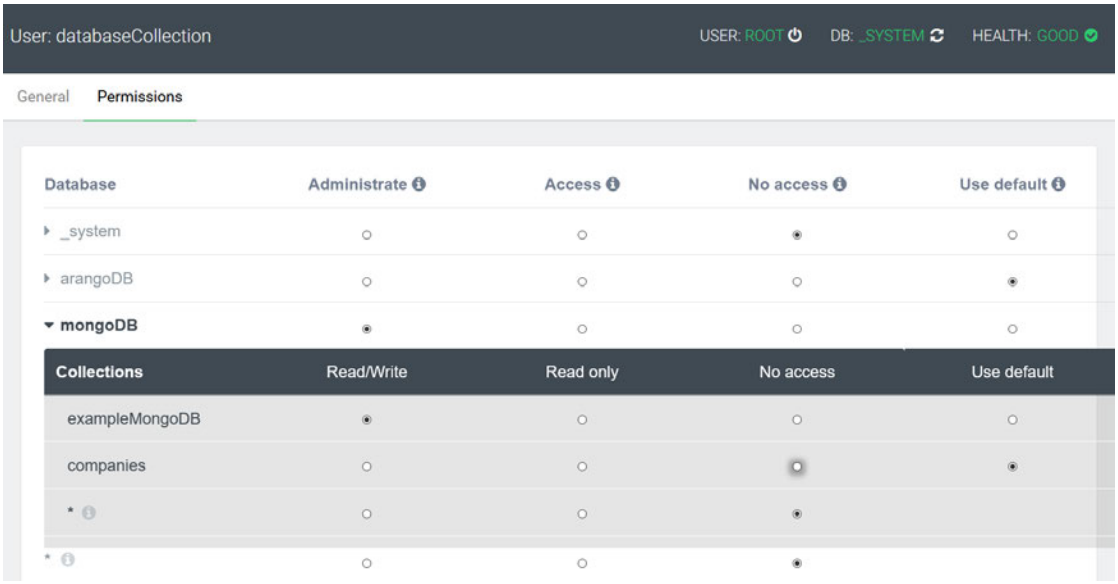


Abbildung 5.27: Berechtigungen des Benutzers „databaseCollection“ in der ArangoDB

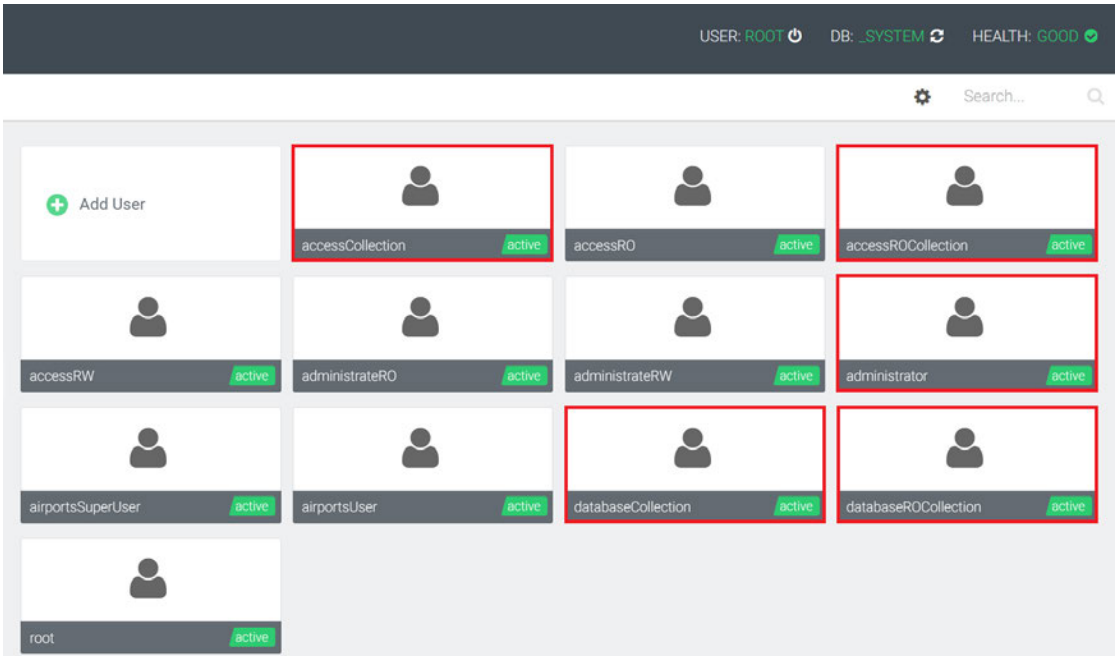


Abbildung 5.28: Übersicht aller Benutzer in der ArangoDB

Wird die Migration durchgeführt, ohne dass bereits bei einer Teilmenge die gesamte Zugriffsebene zugesprochen wird, ergeben sich die Rollen mit den Berechtigungen entsprechend der Tabelle 5.7. Vergleicht man diese mit der in Tabelle 5.4 angelegten Rollen in der ArangoDB, so ist erkenntlich, dass die Rollen korrekt migriert wurden. Bei der Migration besitzt die Rolle „databaseROCollection“ nur „access“ Berechtigungen bei der Datenbankebene, da die niedrigste Berechtigung vergeben wird und bei der Kombination kein Unterschied der Aktionen erkennbar ist.

Tabelle 5.7: Ergebnisse zur Migration der Benutzer/Rollen ohne Teilmenge

Rollenbezeichner	Serverebene	Datenbankebene	Kollektionsebene
administrator	administrate	administrate	read/write
databaseCollection	none	administrate	read/write
databaseROCollection	none	access	read only
accessCollection	none	access	read/write
accessROCollection	none	access	read only
companiesSuperUser	none	administrate	read/write
companiesUser	none	access	read only

Werden Teilmengen erlaubt, besitzen die Rollen mehr Berechtigungen als in der vorherigen Tabelle (siehe Tabelle 5.8). Hierbei war das Ziel jeweils die nächste Zugriffsebene in der Hierarchie zu erlangen: Serverebene > Datenbankebene > Kollektionsebene. Aufgrund dessen, dass bei einer Teilmenge jeweils die höheren Berechtigungen zugewiesen werden, erhalten die Benutzer statt „access“ „administrate“ in der Datenbankebene, da bei der Kollektionsberechtigung beide Zugriffsrechte die gleichen Aktionen erlauben.

Tabelle 5.8: Ergebnisse zur Migration der Benutzer/Rollen mit Teilmenge

Rollenbezeichner	Serverebene	Datenbankebene	Kollektionsebene
administrator	administrate	administrate	read/write
databaseCollection	administrate	administrate	read/write
databaseROCollection	none	administrate	read/write
accessCollection	none	administrate	read/write
accessROCollection	none	administrate	read/write
companiesSuperUser	administrate	administrate	read/write
companiesUser	none	administrate	read/write

Insgesamt ist die Migration in beiden Fällen den Erwartungen entsprechend durchgeführt und die Benutzer sind korrekt angelegt worden.

5.5.3 Migration von der ArangoDB zur MongoDB

Im folgenden werden die Ergebnisse der Migration der Daten, Indizes, sowie Benutzer und Rollen von der ArangoDB zur MongoDB ausgewertet. Berücksichtigt wurde dafür der Beispieldatensatz und der realitätsnahe Datensatz „airports“.

Daten

Wird bei der Migration die Übernahme des _key der ArangoDB ausgewählt, so wird dieser als _id in der MongoDB angelegt. Andernfalls generiert die MongoDB eigene ObjectIDs, um die Dokumente eindeutig zu identifizieren. In Abbildung 5.29 ist exemplarisch ein Dokument erkennbar, welches den _key als _id übernommen hat, welcher eine sechsstellige Nummer ist. Datum-Felder werden beim Migrationsprozess erkannt, wenn sie dem standardisierten Schema entsprechen (vgl. Kapitel 4.2.1). Ist eine Zahl größer als ein Integer darstellen kann, wird dieser als Long repräsentiert. Migriert wurden insgesamt 9 Dokumente, da in der ArangoDB neben dem Beispieldatensatz ein Dokument mit dem VPack angelegt wurde.

Der Datensatz des Flughafens wurde mit den 43.991 Dokumenten migriert, insgesamt sind sowohl beim Flughafen- als auch Beispieldatensatz die Dokumente korrekt migriert worden.

Indizes

In der ArangoDB sind für den Beispieldatensatz sechs Indizes hinterlegt, jeweils einer für jeden Typ (vgl. Tabelle 4.3). Obwohl Skiplist- und Hash-Indizes mit der aktuellen Version nicht mehr angelegt werden sollten, wurde hier aus Demonstrationszwecken auf alle Typen zurückgegriffen. Der Abbildung 5.30 können alle sechs Indizes entnommen werden. Da die MongoDB kein Äquivalent zu Skiplist- oder Persistent-Indizes bietet, sind diese als Index ohne speziellen Typ hinterlegt worden. Der TTL-Index ist bei der MongoDB kein Typ, sondern eine Eigenschaft.

```
{
    "_id": "012345",
    "arrayField": [84345, 53223, "consetetur sadipscing"],
    "dateField": {
        "$date": "2020-07-13T13:37:00.000Z"
    },
    "dateBefore1970": {
        "$numberLong": "-145880535000"
    },
    "decimal128Field": "22.57",
    "doubleField": 1000.54,
    "intField": 1,
    "int32field": 2147483647,
    "int64Field": {
        "$numberLong": "9223372036854775807"
    },
    "regexField": {
        "pattern": "^J",
        "options": "i"
    },
    "timestampField": {
        "t": 1577880000,
        "i": 1
    },
    "location": {
        "type": "Point",
        "coordinates": [-180, -90]
    }
}
```


Abbildung 5.29: Dokument in der MongoDB

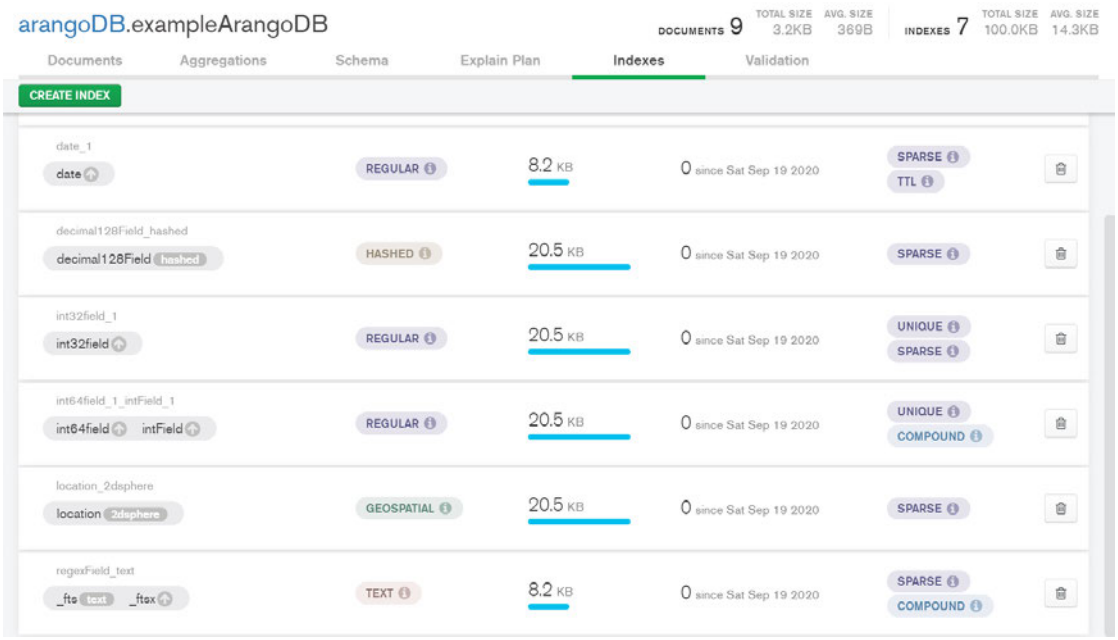


Abbildung 5.30: Indizes des migrierten Beispieldatensatzes in der MongoDB

Für den Datensatz des Flughafens wurden zwei der drei Indizes migriert, da die MongoDB Geo-Indizes nur auf einem Feld bilden kann, die ArangoDB hingegen auf mehreren. Damit wird nur der zusammengesetzte Index aus drei Feldern und der Text-Index migriert.

Die Indizes wurden für die Beispieldatensätze erwartungsgemäß angelegt.

Benutzer/Rollen

Vergleicht man die Tabellen aus Kapitel 4.2.7 zu den Berechtigungen für die verschiedenen Ebenen in der ArangoDB, so lassen sich die Aktionen entsprechend der Berechtigungen ableiten. Die Aktionen, welche den Benutzern bzw. Rollen zugewiesen wurden, können der Tabelle 5.9 entnommen werden. In Abbildung 5.31 sind exemplarisch drei Benutzer mit der zugewiesenen Rolle erkennbar. Die Rollen mit den Aktionen, welche exemplarisch für die Rolle „administrateRW“ in Abbildung 5.32 erkennbar ist, stimmen mit den Berechtigungen der ArangoDB überein. Damit wurden die Benutzer aus der ArangoDB korrekt in die MongoDB übertragen.

Tabelle 5.9: Ergebnisse zur Migration der Benutzer/Rollen

Benutzerbezeichner	Aktionen
root	collMod, createCollection, createIndex, createUser, dropCollection, dropDatabase, dropIndex, dropUser, find, insert, listCollections, listIndexes, remove, renameCollection, shutdown, update, updateUser
administrateRW	collMod, createCollection, createIndex, dropCollection, dropIndex, find, insert, listCollections, listIndexes, remove, renameCollection, update
administrateRO	find, listCollections, listIndexes
accessRW	find, insert, listCollections, listIndexes, remove, update
accessRO	find, listCollections, listIndexes
airportsSuperUser	collMod, createCollection, createIndex, dropCollection, dropIndex, find, insert, listCollections, listIndexes, remove, renameCollection, update
airportsUser	find, listCollections, listIndexes

Da in der ArangoDB Rollen und Benutzer nicht getrennt, sondern nur Benutzer angelegt werden, wurde in der MongoDB eine Rolle erstellt, welche den gleichen Bezeichner wie der Benutzer aufweist und die Aktionen verwaltet. In Abbildung 5.31 sind drei der

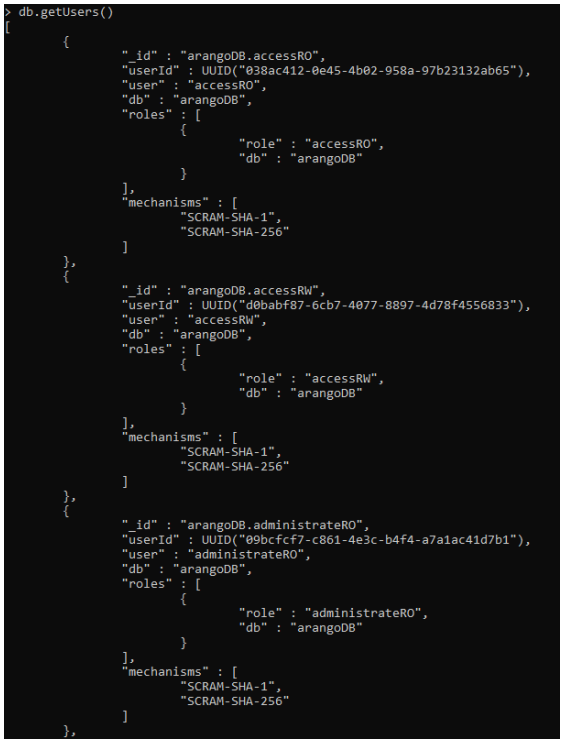


Abbildung 5.31: Exemplarische Darstellung von drei Benutzern in der MongoDB

sechs Benutzer exemplarisch abgebildet, welche eine gleichnamige Rolle aufweisen. Diese Rolle verwaltet die Aktionen, welche in Abbildung 5.32 exemplarisch für die Rolle „administrateRW“ erkennbar sind.

```
{
        "role" : "administrateRW",
        "db" : "arangoDB",
        "isBuiltin" : false,
        "roles" : [ ],
        "inheritedRoles" : [ ],
        "privileges" : [
                {
                        "resource" : {
                                "db" : "arangoDB",
                                "collection" : "exampleArangoDB"
                        },
                        "actions" : [
                                "collMod",
                                "createCollection",
                                "createIndex",
                                "dropCollection",
                                "dropIndex",
                                "find",
                                "insert",
                                "listCollections",
                                "listIndexes",
                                "remove",
                                "renameCollection",
                                "update"
                        ]
                }
        ],
```


Abbildung 5.32: Detailansicht der Rolle „administrateRW“ in der MongoDB

6 Fazit

Mit der Konzeption einer Migration von relationalen zu dokumentenorientierten, sowie zwischen dokumentenorientierten Datenbanken konnte gezeigt werden, dass die Daten übernommen, bzw. für relationale Datenbanken, diese entsprechend einem neuen Schema angepasst werden können. Mit einbezogen wurden nicht nur die Daten, sondern auch die Funktionalitäten. Diese wurden theoretisch ausgewertet, ob eine Migration in die Zieldatenbank möglich ist. Insgesamt betrachtet ist dies nur für wenige Funktionalitäten machbar und dies ebenfalls nur mit Einschränkungen. Für die Realisierung des Konzepts mit der Rel(Dok)^2 wurden schlussendlich nur zwei Funktionalitäten migriert, die Indizes, sowie die Benutzer bzw. Rollen. Die Indizes stellen ein Konzept, welches in relationalen, als auch in dokumentenorientierten Datenbanken gleichermaßen realisiert werden konnte. Im Vordergrund steht dabei die Möglichkeit, die Performance von Abfragen zu verbessern, nicht wie die Indizes intern arbeiten. Indizes können grundsätzlich auf den gleichen Feldern bzw. Tabellen wie in der Quelldatenbank übertragen werden. Bei den Benutzern bzw. Rollen beziehen sich die Einschränkungen darauf, dass die Datenbanken diese Funktionalität unterschiedlich interpretieren und damit nur für wenige Berechtigungen realisiert werden können. Grundsätzlich können jedoch datenbankunabhängig Benutzern bzw. Rollen Beschränkungen beim Zugriff auf die Daten gesetzt werden. Obwohl es sich bei der Migration zwischen zwei dokumentenorientierten Datenbanken um das gleiche Modell handelt, wurden die Funktionalitäten intern unterschiedlich umgesetzt. Im Bezug auf die Einschränkungen, die bei einer Migration hingenommen werden müssen, unterscheidet es sich kaum von der Migration zwischen zwei unterschiedlichen Modellen, wie es bei einer relationalen zu einer dokumentenorientierten Datenbank der Fall ist.

Ziel der Masterthesis war es, ein Konzept zur Migration von relationalen zu dokumentenorientierten, sowie zwischen dokumentenorientierten Datenbanken zu erstellen. Dabei wurden sowohl die Daten, als auch die Funktionalitäten berücksichtigt. Evaluiert wurde der Ansatz anhand des im Rahmen der Masterthesis entwickelten Tools Rel(Dok)^2. Basierend darauf kann die Forschungsfrage wie folgt beantwortet werden. Die Daten können

sowohl zwischen dokumentenorientierten als auch von relationalen zu dokumentenorientierten Datenbanken migriert werden, mit der Einschränkung, dass nicht alle Datentypen erhalten bleiben können. Bei den Funktionalitäten sind starke Einschränkungen hinzunehmen, da selbst bei der Migration zwischen dem gleichen Datenbankmodell, diese eine große Vielfalt aufweisen. Die Daten, die Benutzer/Rollen und die Indizes der Funktionalitäten können automatisiert migriert werden. Sowohl bei der Konzeption, als auch bei der Realisierung hat sich gezeigt, dass die Funktionalitäten sehr individuell für jede Datenbank ausfallen und nur eine kleine gemeinsame Schnittmenge bilden. Für einen Migrationsprozess bedeutet dies, dass bei den Funktionalitäten immer ein großer Teil der Migrationsarbeit beim Anwender bleibt, da für den konkreten Anwendungsfall entschieden werden muss, welche Funktionalitäten unabdingbar sind und wie diese in der Zieldatenbank realisiert werden können. Ebenso muss beachtet werden, dass sich die Datentypen verändert haben können, falls diese in der Zieldatenbank nicht vorhanden sind.

Konzeptionell sind neben den Indizes und Benutzern sowie Rollen ebenfalls die Abfragen migrierbar. Mit Rel(Dok)^2 wurde jedoch zusätzlich die Einschränkung vollzogen, dass Abfragen nicht berücksichtigt werden. Im Zusammenhang mit den Abfragen können ebenfalls die Views und das Map-Reduce nicht mit einbezogen werden. Für eine Migration derartiger Funktionalitäten ist ein Mapping der verschiedenen Abfragesprachen notwendig. Dazu müssen sämtliche Operatoren von Quell- und Zieldatenbank untersucht und analysiert werden, welches einen sehr komplexen Prozess darstellt und somit nicht in dieser Masterthesis behandelt wurde.

Verglichen mit den Ansätzen aus Tabelle 2.4, bietet das eigene Konzept einen Schwerpunkt auf der Auswertung der Funktionalitäten. Die Ansätze, welche die Funktionalitäten beachten, beziehen sich auf die Abfragen oder in sehr wenigen Fällen auf die Indizes. Abfragen wurden im Konzept der Masterthesis nicht berücksichtigt, dafür werden aber durch den in dieser Arbeit vorgestellten Ansatz die Indizes, sowie die Benutzer/Rollen ausgewertet. Eine Evaluierung ist, genauso wie für die meisten verwandten Arbeiten, erfolgt, um zu verdeutlichen, dass das Konzept auch in der Realisierung bestehen kann. Die Größe des Datensatzes ist so gewählt, dass dieser eine ausreichende Komplexität aufweist und ordnet sich im Vergleich zu den anderen Ansätzen im Bezug auf die Menge an Tabellen im oberen Feld an. Das im Rahmen der Masterthesis entstandene Tool Rel(Dok)^2 stellt einen Prototypen dar, jedoch wird dazu kein Pseudocode gegeben. Dies liegt daran, dass der komplette Quellcode einsehbar ist und im Entwurf die Komponenten nur erläuternd behandelt werden.

Insgesamt stellt die Migration von relationalen zu dokumentenorientierten Datenbanken ein weiteres Konzept zu den bestehenden Ansätzen dar. Eine Überprüfung, welche dieser Ansätze die beste Performance aufweist oder das optimale Verfahren für die Entscheidung von Referenz oder Verschachtelung bildet, konnte nicht durchgeführt werden. Dies wird erschwert durch das Problem, dass der Quellcode nur von wenigen Ansätzen vorhanden ist. Ebenso wurde die Rel(Dok)^2 unter bestimmten Versionen und Gegebenheiten der Datenbanken erstellt. Um weiterhin eine Funktion zu gewährleisten, sind anhaltende Updates unvermeidbar.

Bei der Migration von einer relationalen zu einer dokumentenorientierten Datenbank kann Rel(Dok)^2 nur begrenzt eine optimale Lösung für das neue Schema vorschlagen. Die für den Menschen erkennbaren logischen Strukturen in einem Datensatz, können nur bedingt automatisiert erkannt werden. Je nach Nutzung des Datensatzes kann die optimale Struktur stark variieren. Um eine gezielte Anpassung der Struktur in Bezug auf die Abfragen zu erstellen, ist eine Auswertung von häufig verwendeten Abfragen auf die Datenbank sinnvoll. Basierend darauf kann die Entscheidung getroffen werden, ob eine Referenz oder eine Verschachtelung zu bevorzugen ist.

Rel(Dok)^2 bietet im aktuellen Zustand nur eine Migration von PostgreSQL zu MongoDB, sowie zwischen MongoDB und ArangoDB an. Aufgrund dessen, dass der Migrationsprozess hauptsächlich komplett automatisch erfolgt, wurde auf Komponenten verzichtet, welche Einfluss durch den Anwender benötigen. Die Beurteilung des Schemas für die Migration von einer relationalen Datenbank stellt die Ausnahme und kann nicht automatisiert erfolgen. Ohne die Bewertung durch den Anwender kann nicht sichergestellt werden, dass das neue Schema korrekt an die Ansprüche der Zieldatenbank angepasst wird. Erweiterbar wäre das Tool sowohl im Hinblick auf die Anzahl der unterstützten Datenbanken, als auch darauf, dass dem Anwender optional mehr individuelle Anpassungen während des Migrationsprozesses zugesprochen werden. Z.B. könnten Indizes oder Benutzer bzw. Rollen selektiert und anschließend nur die getroffene Auswahl migriert werden. Damit würde dem Anwender eine filigrane Mitbestimmung während des Migrationsprozesses gewährt werden. Ebenso wurden keine verschiedenen Optimierungsstrategien für den automatischen Schemavorschlag der Rel(Dok)^2 realisiert, welche noch erfolgen könnten.

In weiterführenden Forschungen könnte sich mit den Abfragen bzw. den Sprachen dazu befasst werden. Diese besitzen große Differenzen sowohl beim Aufbau einer Abfrage an die Datenbank, als auch bei den Operatoren, welche zur Verfügung stehen. Diese

Faktoren erhöhen die Komplexität, die Sprachen aufeinander zu mappen. Dabei könnte besonders untersucht werden, inwiefern die Sprachen kompatibel sind, wenn diese eine ähnliche Sprachgrundlage besitzen. Mit der Auswahl der MongoDB wurde bewusst die Entscheidung getroffen, eine möglichst unterschiedliche Datenbank zu verwenden. Wie die ArangoDB zeigt, kann eine Abfragesprache für NoSQL Datenbanken sich grundlegend am Schema der SQL Abfragen orientieren. Basierend darauf, kann erforscht werden, ob bei einer Migration zwischen zwei ähnlicheren Datenbanken mehr Funktionalitäten übertragen werden können. Insgesamt bleibt die Auswertung von Abfragen bzw. Sprachen immer eine Einzelfallanalyse, da es für NoSQL Datenbanken keinen Standard bezüglich dieser gibt.

Literaturverzeichnis

[1] ArangoDB: ArangoDB v3.6.4 Documentation. – URL https://www.arangodb.com/docs/stable/. – (Zuletzt aufgerufen am: 06.10.2020)

[2] ArangoDB: Index basics. – URL https://www.arangodb.com/docs/stable/indexing-index-basics.html. – (Zuletzt aufgerufen am: 06.10.2020)

[3] ArangoDB: Introduction. – URL https://www.arangodb.com/docs/stable/aql/. – (Zuletzt aufgerufen am: 06.10.2020)

[4] ArangoDB: Managing Users. – URL https://www.arangodb.com/docs/stable/administration-managing-users.html. – (Zuletzt aufgerufen am: 06.10.2020)

[5] ArangoDB: VelocyPack (VPack). – URL https://github.com/arangodb/velocypack/blob/master/VelocyPack.md. – (Zuletzt aufgerufen am: 06.10.2020)

[6] aws: AWS Schema Conversion Tool. – URL https://aws.amazon.com/de/dms/schema-conversion-tool/. – (Zuletzt aufgerufen am: 06.10.2020)

[7] Brewer, Eric: CAP Twelve years Later: how the r̈ulesẗhave changed. In: Computer (2012), Nr. 2, S. 23–29

[8] Brewer, Eric A.: Towards robust distributed systems. In: PODC Bd. 7, 2000

[9] Celko, Joe: Joe Celko's Complete Guide To NoSQL. Elsevier, 2014

[10] DB-Engines: DB-Engines Ranking. – URL https://db-engines.com/de/ranking. – (Zuletzt aufgerufen am: 06.10.2020)

[11] De Virgilio, Roberto ; Maccioni, Antonio ; Torlone, Riccardo: Converting relational to graph databases. In: First International Workshop on Graph Data Management Experiences and Systems, 2013, S. 1–6

[12] Eini, Oren: Migrating data to RavenDB. – URL https://ravendb.net/articles/data-migration-from-cosmosdb-mongodb-sql-to-ravendb. – (Zuletzt aufgerufen am: 06.10.2020)

[13] El Alami, Alae ; Bahaj, Mohamed: Migration of a relational databases to NoSQL: The way forward. In: 2016 5th International Conference on Multimedia Computing and Systems (ICMCS) IEEE (Veranst.), 2016, S. 18–23

[14] El Alami, Alae ; Bahaj, Mohamed ; Khourdifi, Younes: Supply of a key value database redis in-memory by data from a relational database. In: 2018 19th IEEE Mediterranean Electrotechnical Conference (MELECON) IEEE (Veranst.), 2018, S. 46–51

[15] Flanagan, D.: JavaScript: das umfassende Referenzwerk ; [behandelt Ajax und DOM-Scripting]. O'Reilly, 2007. – ISBN 9783897214910

[16] Geisler, Frank: Datenbanken: grundlagen und design. mitp Verlags GmbH & Co. KG, 2014

[17] Gessert, Felix ; Wingerath, Wolfram ; Friedrich, Steffen ; Ritter, Norbert: NoSQL database systems: a survey and decision guidance. In: Computer Science-Research and Development 32 (2017), Nr. 3-4, S. 353–365

[18] Goyal, Akansha ; Swaminathan, Arun ; Pande, Rasika ; Attar, Vahida: Cross platform (RDBMS to NoSQL) database validation tool using bloom filter. In: 2016 International Conference on Recent Trends in Information Technology (ICRTIT) IEEE (Veranst.), 2016, S. 1–5

[19] Group, The PostgreSQL Global D.: PostgreSQL 12.2 Documentation. – URL https://www.postgresql.org/docs/12/index.html. – (Zuletzt aufgerufen am: 06.10.2020)

[20] Győrödi, Cornelia ; Győrödi, Robert ; Pecherle, George ; Olah, Andrada: A comparative study: MongoDB vs. MySQL. In: 2015 13th International Conference on Engineering of Modern Electric Systems (EMES) IEEE (Veranst.), 2015, S. 1–6

[21] Haerder, Theo ; Reuter, Andreas: Principles of transaction-oriented database recovery. In: ACM computing surveys (CSUR) 15 (1983), Nr. 4, S. 287–317

[22] Hamouda, Shady ; Zainol, Zurinahni: Document-Oriented Data Schema for Relational Database Migration to NoSQL. In: 2017 International Conference on Big Data Innovations and Applications (Innovate-Data) IEEE (Veranst.), 2017, S. 43–50

[23] ISO/IEC: ISO/IEC 25010:2011(en) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. – URL https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en. – (Zuletzt aufgerufen am: 06.10.2020)

[24] Jung, Min-Gyue ; Youn, Seon-A ; Bae, Jayon ; Choi, Yong-Lak: A study on data input and output performance comparison of MongoDB and PostgreSQL in the big data environment. In: 2015 8th International Conference on Database Theory and Application (DTA) IEEE (Veranst.), 2015, S. 14–17

[25] Karnitis, Girts ; Arnicans, Guntis: Migration of relational database to document-oriented database: Structure denormalization and data transformation. In: 2015 7th International Conference on Computational Intelligence, Communication Systems and Networks IEEE (Veranst.), 2015, S. 113–118

[26] Kleuker, Stephan: Grundkurs Datenbankentwicklung. Springer, 2006

[27] Kruse, R. ; Borgelt, C. ; Braune, C. ; Klawonn, F. ; Moewes, C. ; Steinbrecher, M.: Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze. Springer Fachmedien Wiesbaden, 2015 (Computational Intelligence). – ISBN 9783658109042

[28] Kuperjans, Laura: Analyse von Datenbankmigrationen zwischen relationalen und NoSQL Datenbanken. 2019

[29] Kuperjans, Laura: Migration zwischen relationalen und dokumentenorientierten Datenbanken. 2020

[30] Kuszera, Evandro M. ; Peres, Leticia M. ; Fabro, Marcos Didonet D.: Toward RDB to NoSQL: transforming data with metamorfose framework. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing ACM (Veranst.), 2019, S. 456–463

[31] Lee, Chao-Hsien ; Zheng, Yu-Lin: SQL-to-NoSQL schema denormalization and migration: a study on content management systems. In: 2015 IEEE International Conference on Systems, Man, and Cybernetics IEEE (Veranst.), 2015, S. 2022–2026

[32] Li, Chongxin: Transforming relational database into HBase: A case study. In: 2010 IEEE international conference on software engineering and service sciences IEEE (Veranst.), 2010, S. 683–687

[33] Liang, Dongzhao ; Lin, Yunzhen ; Ding, Guiguang: Mid-model design used in model transition and data migration between relational databases and NoSQL databases. In: 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity) IEEE (Veranst.), 2015, S. 866–869

[34] Liyanaarachchi, Gayan ; Kasun, Lakshan ; Nimesha, Malki ; Lahiru, Kanishka ; Karunasena, Anuradha: MigDB-relational to NoSQL mapper. In: 2016 IEEE International Conference on Information and Automation for Sustainability (ICIAfS) IEEE (Veranst.), 2016, S. 1–6

[35] Meier, A.: Werkzeuge der digitalen Wirtschaft: Big Data, NoSQL & Co.: Eine Einführung in relationale und nicht-relationale Datenbanken. Springer Fachmedien Wiesbaden, 2017 (essentials). – ISBN 9783658203375

[36] Meier, Andreas ; Kaufmann, Michael: NoSQL-Datenbanken. In: SQL-& NoSQL-Datenbanken. Springer, 2016, S. 221–240

[37] MongoDB: Indexes. – URL https://docs.mongodb.com/manual/indexes/. – (Zuletzt aufgerufen am: 06.10.2020)

[38] MongoDB: Map-Reduce. – URL https://docs.mongodb.com/manual/core/map-reduce/. – (Zuletzt aufgerufen am: 06.10.2020)

[39] MongoDB: MongoDB Connector for BI. – URL https://docs.mongodb.com/bi-connector/master/. – (Zuletzt aufgerufen am: 06.10.2020)

[40] MongoDB: Server Side Public License FAQ. – URL https://www.mongodb.com/licensing/server-side-public-license/faq. – (Zuletzt aufgerufen am: 06.10.2020)

[41] MongoDB: Server Side Public License (SSPL). – URL https://www.mongodb.com/licensing/server-side-public-license. – (Zuletzt aufgerufen am: 06.10.2020)

[42] MongoDB: The MongoDB 4.2 Manual. – URL https://docs.mongodb.com/manual/. – (Zuletzt aufgerufen am: 06.10.2020)

[43] Nayak, Ameya ; Poriya, Anil ; Poojary, Dikshay: Type of NOSQL databases and its comparison with relational databases. In: International Journal of Applied Information Systems 5 (2013), Nr. 4, S. 16–19

[44] NEO4J: 3.6. Import data. – URL https://neo4j.com/docs/getting-started/current/cypher-intro/load-csv/. – (Zuletzt aufgerufen am: 06.10.2020)

[45] OREL, Ognjen ; ZAKOŠEK, Slaven ; BARANOVIČ, Mirta: Property oriented relational-to-graph database conversion. In: automatika 57 (2016), Nr. 3, S. 836–845

[46] ORIENTDB: Multi-Model. – URL https://orientdb.org/docs//2.0/orientdb.wiki/Tutorial-Document-and-graph-model.html. – (Zuletzt aufgerufen am: 06.10.2020)

[47] PARKER, Zachary ; POE, Scott ; VRBSKY, Susan V.: Comparing nosql mongodb to an sql db. In: Proceedings of the 51st ACM Southeast Conference ACM (Veranst.), 2013, S. 5

[48] PONZI, Gabriele: OrientDB Teleporter – Making Migrations Easier (Part 1). – URL https://orientdb.org/integration/teleporter/orientdb-teleporter-making-migrations-easier. – (Zuletzt aufgerufen am: 06.10.2020)

[49] POSTGRESQL: Appendix C. SQL Key Words. – URL https://www.postgresql.org/docs/12/sql-keywords-appendix.html. – (Zuletzt aufgerufen am: 06.10.2020)

[50] POSTGRESQL WIKI: Converting from other Databases to PostgreSQL. – URL https://wiki.postgresql.org/wiki/Converting_from_other_Databases_to_PostgreSQL. – (Zuletzt aufgerufen am: 06.10.2020)

[51] PRITCHETT, Dan: Base: An acid alternative. In: Queue 6 (2008), Nr. 3, S. 48–55

[52] REDIS: Licenses. – URL https://redislabs.com/legal/licenses/. – (Zuletzt aufgerufen am: 06.10.2020)

[53] REDIS: REDIS SOURCE AVAILABLE LICENSE (RSAL) AGREEMENT. – URL https://redislabs.com/wp-content/uploads/2019/09/redis-source-available-license.pdf. – (Zuletzt aufgerufen am: 06.10.2020)

[54] ROCHA, Leonardo ; VALE, Fernando ; CIRILO, Elder ; BARBOSA, Dárlinton ; MOURÃO, Fernando: A framework for migrating relational datasets to NoSQL. In: Procedia Computer Science 51 (2015), S. 2593–2602

[55] Singh, Manpreet ; Kaur, Karamjit: Sql2neo: Moving health-care data from relational to graph databases. In: 2015 IEEE International Advance Computing Conference (IACC) IEEE (Veranst.), 2015, S. 721–725

[56] Spillner, A. ; Rossner, T. ; Winter, M. ; Linz, T.: Praxiswissen Softwaretest - Testmanagement: Aus- und Weiterbildung zum Certified Tester - Advanced Level nach ISTQB-Standard. dpunkt.verlag, 2014 (iSQI-Reihe). – ISBN 9783864915154

[57] statista: Ranking of the most popular database management systems worldwide, as of September 2019*. – URL https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/. – (Zuletzt aufgerufen am: 06.10.2020)

[58] striim: Data Sources and Targets. – URL https://www.striim.com/sources-and-targets/. – (Zuletzt aufgerufen am: 06.10.2020)

[59] striim: Stream to the Cloud. – URL https://www.striim.com/. – (Zuletzt aufgerufen am: 06.10.2020)

[60] Sullivan, Dan: NoSQL for mere mortals. Addison-Wesley Professional, 2015

[61] Unal, Yelda ; Oguztuzun, Halit: Migration of data from relational database to graph database. In: Proceedings of the 8th International Conference on Information Systems and Technologies, 2018, S. 1–5

[62] Unterstein, Michael ; Matthiessen, Günter: Relationale Datenbanken und SQL in Theorie und Praxis. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012

[63] Wijaya, Yansyah S. ; AkhmadArman, Arry: A Framework for Data Migration Between Different Datastore of NoSQL Database. In: 2018 International Conference on ICT for Smart Society (ICISS) IEEE (Veranst.), 2018, S. 1–6

[64] Yassine, Fatima ; Awad, Mamoun A.: Migrating from SQL to NOSQL Database: Practices and Analysis. In: 2018 International Conference on Innovations in Information Technology (IIT) IEEE (Veranst.), 2018, S. 58–62

[65] Zhao, Gansen ; Huang, Weichai ; Liang, Shunlin ; Tang, Yong: Modeling MongoDB with relational model. In: 2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies IEEE (Veranst.), 2013, S. 115–121

[66] Zhao, Gansen ; Li, Libo ; Li, Zijing ; Lin, Qiaoying: Multiple nested schema of HBase for migration from SQL. In: 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing IEEE (Veranst.), 2014, S. 338–343

[67] Zhao, Gansen ; Lin, Qiaoying ; Li, Libo ; Li, Zijing: Schema conversion model of SQL database to NoSQL. In: 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing IEEE (Veranst.), 2014, S. 355–362

A Anhang

A.1 Datenträger CD

Die, der Masterthesis beigelegten CD, enthält eine digitale Version der Masterthesis im PDF-Format und den Quellcode zur Rel(Dok)^2, sowie die Dateien zu den Beispieldatenbanken.

Glossar

Big Data Beschreibt große Datenmengen, welche neue Verfahren brauchen, um diese auszuwerten [35].

Directed Acyclic Graph Ein gerichteter Graph der keine Zyklen aufweist.

Entity-Relationship-Modell Graphische Darstellung der Beziehungs- und Tabellenstruktur einer relationalen Datenbank [62, 36].

Funktionalitäten Unter Funktionalitäten werden sämtliche unterstützte Funktionen der jeweiligen Datenbank verstanden. Dies können beispielsweise Indizes, Stored Procedures, Abfragen oder MapReduce Unterstützung sein [29].

JavaScript Clientseitige Programmiersprache, welche objektorientierte Konzepte unterstützt [15].

Künstliches Neuronales Netz Ein System zur Verarbeitung von Informationen, für das das Nervensystem/Gehirn von Lebewesen als Vorbild dient. Die Informationen passieren ein Netz, welches ein Graph mit gewichteten Kanten ist [27].

Map-Reduce Verfahren, um große Datenmengen verteilt und schnell auszuwerten [38].

partial Bei der partial Eigenschaft werden nur die Dokumente indiziert, welche den Kriterien der Filteroption entsprechen [2, 37].

sparse Sparse-Indizes erfolgen nur auf den Dokumenten, welche das zu indizierende Feld aufweisen [2, 37].

TTL Der TTL-Index dient zur automatischen Entfernung von Dokumenten auf Datum-Feldern [2, 37].

unique Indizes mit der Eigenschaft unique versichern, dass keine Duplikate auf dem Feld liegen [2, 37].

Variety Variety steht für die Vielfalt an Daten und Datenquellen [35].

Velocity Velocity steht für die Verarbeitungsgeschwindigkeit [35].

Volume Volume steht für die Datenmenge [35].

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Masterarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Konzeption und Realisierung einer Datenbankmigration zwischen relationalen und dokumentenorientierten Datenbanksystemen

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	[redacted]
Ort	Datum	Unterschrift im Original