Masterarbeit

Analyse und Evaluation von NoSQL-Datenbanken im Verhältnis zu relationalen Datenbanken anhand von Benchmarktests

Hochschule Anhalt
Anhalt University of Applied Sciences

Studiengang Informationsmanagement

vorgelegt von

Dennis Maximilian Hofmann

Mat. Nr. 4063386

Erstgutachter: Prof. Dr. Michael Worzyk
Zweitgutachter: Dr. Bernd Krause

Halle (Saale), den 18.09 .2017

ABSTRAKT

Die vorliegende Masterarbeit präsentiert einen Überblick über die beiden Datenbankmodelle: Relationale Datenbank und NoSQL-Datenbank. Hierzu werden zunächst deren Funktionsweisen sowie die verschiedenen Softwarelösungen expliziert.

Im Anschluss an den theoretischen Teil werden jeweils vier Benchmarktests durchgeführt, die die zwei Systeme miteinander vergleichen. Der verwendete Datensatz bezieht sich hierbei auf die Kriminalitätsstatistik der Stadt Chicago vom Jahr 2001 bis heute. Der Datensatz unterteilt sich in 22 Spalten und in über 5,5 Millionen Datensätze. Es wird bewusst ein Exempel aus der Realität ausgewählt, um mithilfe dessen die späteren Suchanfragen optimal auszuwerten.

Die verschiedenen Benchmarktests erfolgen unter identischen Hardware- sowie Softwarebedingungen, um so ein möglichst objektives Ergebnis herbeizuführen.

Anhand der grafischen Aufarbeitung der Testergebnisse wird sich eine Empfehlung für eines der beiden Datenbanksysteme versprochen. Die Aussage wird keine Allgemeingültigkeit besitzen, da hierzu die Projektzeit nicht ausreichend ist und auch die technischen Mittel im Sinne von Rechenzentren fehlen.

Ziel ist jedoch, unter Zuhilfenahme dieser Masterarbeit, ähnliche Datensätze maßgeblich zu klassifizieren, sodass die Selektion des effizienteren Systems fortan ohne vorherigen Benchmarktest stattfinden kann.

ABSTRACT

The master thesis on hand presents an overview of the following two database models: the relational database und the NoSQL database. Initially, it explains the functionality and the different software solutions.

In addition to the theoretical part, there will be an implementation of four benchmark tests which leads to comparing the two systems with one another. The utilized data set refers to crime statistics of Chicago from 2001 until today. This data set divides into 22 columns consisting of over 5.5 million data sets. The usage of an example set in reality is deliberate in order to ensure the ideal evaluation of the following search request.

The various benchmark tests are carried out under identical hardware and software conditions to enable an outcome as objective as possible.

Based on the graphic elaboration of the test results, the aim is to recommend one of the database systems. This proposition will not be universal because of the limited period of time and the lock of technical resources like data centers.

The main goal of the master thesis is the classification of similar data sets in order to help select the most efficient system without any previous benchmark tests.

EIDESSTATTLICHE ERKLÄRUNG

Hiermit versichere ich, die vorliegende Masterarbeit ohne Hilfsmittel Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die wörtlich oder inhaltlich aus den Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Halle (Saale), den 18.09.2017

....

Dennis Maximilian Hofmann

INHALTSVERZEICHNIS

ABSTRAKT I

ABSTRACT II

EIDESSTATTLICHE ERKLÄRUNG III

INHALTSVERZEICHNIS IV

ABBILDUNGSVERZEICHNIS VI

TABELLENVERZEICHNIS VIII

ABKÜRZUNGSVERZEICHNIS IX

1 VORWORT 1

2 AUFGABENSTELLUNG 2

3 NOSQL-DATENBANKEN 3

3.1 EINFÜHRUNG IN DIE NOSQL-DATENBANKEN 4
3.1.1 Vertikale/Horizontale Skalierung *5*
3.1.2 CAP-Theorem/Brewer Theorem *6*
3.1.2.1 Konsistenzmodell BASE 9
3.1.3 Map/Reduce *10*
3.1.4 Consistent-Hashing *12*
3.1.5 Vector Clocks *13*
3.2 NOSQL-DATENBANKEN 14
3.2.1 Key/Value-Stores *14*
3.2.1.1 Datenmodell und Funktionsweise 14
3.2.1.2 Beispielanwendung „Redis“ 16
3.2.2 Document Stores *18*
3.2.2.1 Einleitung und Funktionsweise 18
3.2.2.2 Beispielanwendung „CouchDB“ 20
3.2.3 Wide Column Stores *21*
3.2.3.1 Einleitung und Funktionsweise 21
3.2.3.2 Beispielanwendung „Cassandra“ 24
3.2.4 Graphdatenbanken *26*
3.2.4.1 Einleitung und Funktionsweise 27
3.2.4.2 Beispielanwendung „Neo4j“ 29
3.2.5 Data Stream Systems *31*
3.2.5.1 Data Stream 31
3.2.5.2 DBMS vs. DSMS 32
3.2.5.3 Anfragen 33
3.2.5.4 DSMS Beispielanwendung „Stream“ 34
3.2.6 Kritik an NoSQL-Datenbanken *37*
3.2.6.1 Unzureichender Support 37
3.2.6.2 Geringe Standardisierung 38
3.2.6.3 Fehlende Nachhaltigkeit 38
3.2.6.4 Mangelnde Entwicklungsreife 38
3.2.7 Zusammenfassung *39*

4 RELATIONALE DATENBANKEN 41

4.1 AUFBAU EINER RELATIONALEN DATENBANK 41
4.2 ENTITY-RELATIONSHIP-MODELL 43
4.3 ANFRAGENVERARBEITUNG 44
4.3.1 Anfrageverarbeitung mit SQL *47*

4.4 Speicherstrukturen von relationalen Datenbanken 49
4.5 Regulärer Ausdruck 51
4.6 Verteilte Datenhaltung 52
4.6.1 Verteilte Anfragenverarbeitung 55
4.6.2 Synchronisation 56
4.7 Datenbanksicherheit und Autorisierung 57
4.8 Kritik an relationalen Datenbanken 59
4.9 Einleitung in MySQL 60
4.9.1 phpMyAdmin 61
4.10 Zusammenfassung 61

5 BENCHMARKTEST 63
5.1 Installation und Konfiguration der Datenbank-Systeme 65
5.1.1 MySQL 65
5.1.2 NoSQL – Apache Cassandra 68
5.2 Testdatensatz 73
5.3 Verwendete Hardware 75
5.4 Durchführung des Benchmarktests 76
5.4.1 Test 1 Select All – Abfragen aller Daten 76
5.4.1.1 MySQL 76
5.4.1.2 NoSQL 78
5.4.1.3 Auswertung 80
5.4.2 Test 2 Equal – Abfrage auf Gleichheit 81
5.4.2.1 MySQL 81
5.4.2.2 NoSQL 83
5.4.2.3 Auswertung 85
5.4.3 Test 3 Small-Range – Kleinere Bereichsabfrage 86
5.4.3.1 MySQL 86
5.4.3.2 NoSQL 88
5.4.3.3 Auswertung 90
5.4.4 Test 4 Large-Range – Größere Bereichsabfrage 91
5.4.4.1 MySQL 91
5.4.4.2 NoSQL 93
5.4.4.3 Auswertung 95
5.5 Analyse und Bewertung der Ergebnisse 96
5.6 Probleme und Messfehler 98
5.7 Zusammenfassung 100

6 FAZIT 102

7 ZUSAMMENFASSUNG UND AUSBLICK 104

8 LITERATURVERZEICHNIS 107

ABBILDUNGSVERZEICHNIS

Abbildung 1: Bekannte Vertreter von RDBMS und NoSQL([27], S. 6) 3

Abbildung 2: CAP-Theorem/Eric Brewer ([27], S. 33) 7

Abbildung 3: CAP-Theorem Verfügbarkeit und Ausfalltoleranz 9

Abbildung 4: Map/Reduce Beispielgrafik [14] 11

Abbildung 5: Map/Reduce Erweiterung ([8], S. 10) 12

Abbildung 6: Key/Value-Store Datenmodell mit Entität Person 15

Abbildung 7: Operationen von Key/Value-Datenbanken 16

Abbildung 8: Start des Redis Servers 17

Abbildung 9: Redis Sets Beispielanwendung mit HS Anhalt 17

Abbildung 10: XML und JSON im direkten Vergleich 18

Abbildung 11: Beispielhafter Document Store mit Entität Person.............................. 20

Abbildung 12: Wide Column Store Datenmodell mit Entität Person 22

Abbildung 13: Formulierung von Restriktions- und Projektionsbedingungen 24

Abbildung 14: Datenmodell „Keyspace Kunde“... 26

Abbildung 15: Verschiedene Graphmodelle 28

Abbildung 16: Beispielgraph mit Beziehungen zwischen den Studenten der Hochschule Anhalt 30

Abbildung 17: Unterschiede zwischen DBMS und DSMS[29] 33

Abbildung 18: Beziehung von Datentypen und Operationen....................................... 34

Abbildung 19: Aufbau einer relationalen Datenbank.. 42

Abbildung 20: Entity-Relationship-Modell... 44

Abbildung 21: Kartesisches Produkt (Kreuzprodukt) 45

Abbildung 22: Union (Vereinigung) 45

Abbildung 23: Differenz.. 46

Abbildung 24: Projektion 46

Abbildung 25: Restriktion 47

Abbildung 26: Anfrageverarbeitung einer replizierten Master-Slave Architektur......... 53

Abbildung 27: Horizontale und vertikale Fragmentierung im direkten Vergleich 54
Abbildung 28: Vierstufiger Verarbeitungsprozess einer Leseanfrage 56
Abbildung 29: Konfiguration des Datenbank-Servers 66
Abbildung 30: Überprüfung der Server-Verbindung 66
Abbildung 31: Erfolgreicher Import des Datensatzes 68
Abbildung 32: Start des CQL-Clients 69
Abbildung 33: Import der CSV-Datei Crimes... 71
Abbildung 34: EXPAND OFF 72
Abbildung 35: EXPAND ON 72
Abbildung 36: Auswertung der Laufzeitanalyse unter Cassandra 72
Abbildung 37: Auswertung der Abfrage SELECT ALL MySQL................................. 78
Abbildung 38: Auswertung der Abfrage SELECT ALL NoSQL 80
Abbildung 39: Gegenüberstellung von MySQL und NoSQL - SELECT ALL 81
Abbildung 40: Auswertung der Abfrage auf Gleichheit MySQL 83
Abbildung 41: Auswertung der Abfrage auf Gleichheit NoSQL.................................. 85
Abbildung 42: Gegenüberstellung von MySQL und NoSQL - Equal 86
Abbildung 43: Auswertung der Small Range Abfrage MySQL.................................... 88
Abbildung 44: Auswertung der Small Range Abfrage.. 90
Abbildung 45: Gegenüberstellung von MySQL und NoSQL – Small Range................ 91
Abbildung 46: Auswertung der Large Range Abfrage MySQL.................................... 93
Abbildung 47: Auswertung der Large Range Abfrage NoSQL 95
Abbildung 48: Gegenüberstellung von MySQL und NoSQL – Large Range................ 96

TABELLENVERZEICHNIS

Tabelle 1: Vertikale/Horizontale Skalierung([27], S. 377) 6

Tabelle 2: Beschreibung der einzelnen Columns des Testdatensatzes[26] 75

Tabelle 3: Ergebnisse der Abfrage SELECT ALL MySQL... 77

Tabelle 4: Ergebnisse der Abfrage SELECT ALL MySQL... 79

Tabelle 5: Ergebnisse der Abfrage auf Gleichheit MySQL 82

Tabelle 6: Ergebnisse der Abfrage auf Gleichheit NoSQL 84

Tabelle 7: Ergebnisse der Small Range Abfrage MySQL.. 87

Tabelle 8: Ergebnisse der Small Range Abfrage NoSQL 89

Tabelle 9: Ergebnisse der Large Range Abfrage MySQL.. 92

Tabelle 10: Ergebnisse der Large Range Abfrage NoSQL 94

ABKÜRZUNGSVERZEICHNIS

ACID	Atomicity, Consistency, Isolation, Durability
ACM	Association for Computing Machinery
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BASE	Basically, Available, Soft State, Eventually Consistent
BSON	Binary JSON
CAP	Consistency, Availability, Partition Tolerance
CPU	Central Processing Unit
CQL	Continuous Query Language
DAHP	DBMS-Active Human-Passiv
DBA	Datenbankadministrator
DBM	Database Manager
DBMS	Datenbank-Management-System
DCL	Data Control Language
DDL	Data Definition Language
DML	Data Manipulation Language
DSMS	Data Stream Management System
ER-Modell	Entity-Relationship-Modell
GDBMS	Generic DataBase Management System
GPL	General Public License
GUI	Graphical user interface
HADP	Human-Active DBMS-Passiv
HDD	Hard Disk Drive
ISAM	Index Sequential Access Method
JSON	Java Script Object Notation
KSV	Key/Value Store
LRDIMMS	Load Reduced Dual Inline Memory Module
PHP	Personal Homepage/ Hypertext Preprocessor
RAM	Random-Access-Memory
RDBMS	Relational Database Management System
RDIMMS	Registered Dual Inline Memory Module
REST	Representational State Transfer

SPARQL	Sparql Protokoll and RDF Query Language
SQL	Structured Programming Language
SSD	Solid-State-Drive
TTL	Time to live
UTF-8	Universal Character Set
VM	Virtuelle Maschine
XML	Extensible Markup Language

1 Vorwort

Die vorliegende Masterarbeit mit dem Titel „Analyse und Evaluation von NoSQL-Datenbanken im Verhältnis zu relationalen Datenbanken anhand von Benchmarktests“ beschäftigt sich mit verschiedenen Datenbanksystemen. Die Idee zu diesem Thema entwickelte sich während der Zeit meines Studiums an der Hochschule Anhalt.

Gemeinsam mit Herrn Prof. Dr. Michael Worzyk wurde das Thema definiert und die dazugehörige Aufgabenstellung formuliert. Nach langer Einarbeitungszeit in das Thema konnten im anschließenden Benchmarktest hingegen aller Erwartungen erstaunliche Ergebnisse ausgemacht werden. Während dieser Zeit standen mir die beiden Dozenten Herr Prof. Dr. Michael Worzyk und Herr Dr. Bernd Krause mit Rat und Tat zur Seite, denen ich recht herzlich danken möchte.

Des Weiteren bedanke ich mich bei Lisa Müller, die viel Zeit in die Korrektur meiner Arbeit investiert hat sowie bei meinen Eltern, die mir dieses Studium ermöglicht haben und auf deren Unterstützung ich immer zählen kann. Ein letzter Dank geht an meine Freunde, die mich während dieser Zeit unterstützten und begleiteten.

2 Aufgabenstellung

Vor dem Hintergrund steigender Datenmengen sowie dem ständig wachsenden Datenaustausch zwischen einzelnen Datenbanken wird die Forderung nach innovativen Ansätzen in der Welt der Datenbanken größer. Seit den 2000er Jahren haben sich NoSQL-Systeme etabliert und Alternativen für die Verwaltung von Big Data Anwendungen aufgezeigt. Jedoch sind die schon länger existierenden relationalen Datenbanken noch durchaus konkurrenzfähig und finden bis heute zahlreiche Anwendungsfelder.

Das erste Kapitel der Arbeit widmet sich den theoretischen Grundlagen von NoSQL-Systemen zum besseren Verständnis der differenten Funktionsweisen.

In gleicher Weise gilt es die Theorie der relationalen Datenbanken auszuarbeiten. Dies geschieht jedoch nicht in aller Tiefe, da es hierbei keine große Datenbankvielfalt mit grundlegend abweichenden Funktionen gibt.

Anhand der vorangegangenen Theorie verspricht man sich, die spätere Analyse des Benchmarktests exakter durchführen zu können und mittels dieser, mögliche Probleme aufzudecken.

Der praktische Teil umfasst das Aufsetzen zweier Datenbanksysteme, die nachfolgend anhand eines Testdatensatzes im Größenbereich von 2 Millionen bis zu 10 Millionen Daten einem Benchmarktest unterzogen werden. Zum Test wird identische Hardware verwendet, um möglichst faire Bedingungen zu schaffen. Die Ergebnisse werden im Anschluss in Form von Diagrammen grafisch aufgearbeitet und miteinander verglichen. Es werden deutliche Leistungsunterschiede zwischen den beiden Systemen erwartet, deren Entstehung genauer analysiert wird. Anschließend wird eine Empfehlung für eines der getesteten Datenbanksysteme ausgesprochen, sofern die Ergebnisse dies zulassen.

3 NoSQL-Datenbanken

NoSQL-Systeme erfreuen sich seit einigen Jahren einer immer größeren Beliebtheit und haben insbesondere durch das Web 2.0 einen regelrechten Hype erfahren. Erste Überlegungen mit NoSQL gehen auf den US-amerikanischen Informatiker Ken Thompson zurück, der im Jahr 1979 eine Key/Value-Datenbank namens DBM entwickelte. In den 1980er verbreiteten sich Systeme wie Lotus Notes und Berkley DB, die besonders für kleine Datenmengen geeignet sind. Die 2000er Jahre brachten den Durchbruch für NoSQL-Systeme. Dank den Internetgiganten Google, Facebook, Yahoo, Amazon und Co. entstanden NoSQL-Systeme in rasantem Tempo, die in hohem Maße für große Datenmengen prädestinierter scheinen als relationale Datenbankmodelle. In den Jahren 2006-2009 kristallisierten sich die noch heute beliebten und wertvollsten NoSQL-Systeme wie CouchDB, Cassandra, Redis, MongoDB und Neo4j heraus. Diese Systeme wurden keineswegs entwickelt, um relationale Datenbanksysteme (RDBMS) zu verdrängen, sondern bieten durch „Not Only SQL" vor allem im Bereich Big Data große Vorteile, sei es in der Performance oder im Speichermanagement.

Die nachfolgende Abbildung präzisiert die Zuordnung zu relationalen Systemen und NoSQL-Datenbanken. Dabei wird sich auf die bekanntesten Vertreter beschränkt.

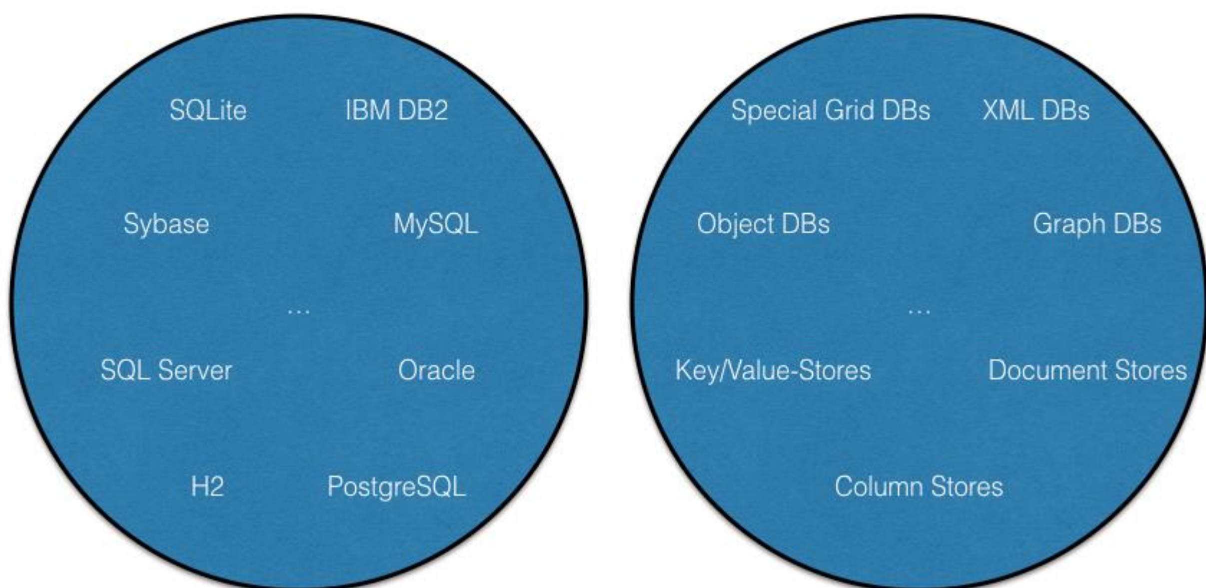


Abbildung 1: Bekannte Vertreter von RDBMS und NoSQL([27], S. 6)

3.1 Einführung in die NoSQL-Datenbanken

NoSQL beschreibt eine neue Generation, die gegenüber den relationalen Datenbanksystemen den heutigen Bedürfnissen des Web 2.0 angepasst ist. Nach Shashank Tiwani sind NoSQL-Datenbanken folgendermaßen definiert:

>> „Not Only SQL". „Whatever the literal meaning, NoSQL is used today an umbrella term for all databases and data stores that don´t follow the popular and well-established RDBMS principles and often relate to large datasets accessed and manipulated on a Web scale. This means NoSQL is not a single product or even a single technology. It represents a class of products and a collection of diverse, and sometimes related, concepts about data storage and manipulation." << ([37]Tiwani, 2011, S. 4)

Die zurückliegenden Jahre indizieren, dass NoSQL nach den Autoren Edlich, Friedland, Hampe und Brauer mindestens einen der nachfolgenden Punkte berücksichtigen muss:([27], S.2)

- *Kein relationales Datenmodell.* Besonders im Bereich von Graphdatenbanken zeigen relationale Modelle enorme Nachteile auf. Werden diese mittels Tabellen und *join*-Operationen gelöst, entstehen erhebliche Performanceprobleme, da ein Übertragen der Traversionen sehr aufwendig ist.

- *Systeme müssen auf horizontale Skalierbarkeit ausgerichtet sein.* Im Gegensatz zu relationalen Datenbanken lassen sich NoSQL-Datenbanken nicht nur in vertikale Richtungen skalieren, sondern können auch auf mehrere Rechnersysteme oder Rechenzentren aufgeteilt werden. Dem wird eine besondere Bedeutung zuteil, da vor allem durch die Globalisierung Rechenzentren von Google und Co. auf der ganzen Welt entstehen.

- *Das NoSQL-System muss frei zugänglich sein und eine Open Source Lizenz besitzen.* Mittlerweile gibt es zahlreiche Vertreter beispielsweise Amazon Simple DB, die für den Endverbraucher nicht unentgeltlich zur Verfügung stehen, sondern in Anspruch genommene Ressourcen in Rechnung stellen.

- *Das System muss frei von Schemen sein oder nur wenige Restriktionen aufweisen.* Damit wird auf ein vordefiniertes Schema verzichtet und demzufolge kann die Speicherung der Daten sehr flexibel gestaltet und auf die entsprechenden Anwendungen übertragen werden.

- *Das System muss eine einfache Datenreplikation unterstützen.* Daraus ergibt sich die logische Konsequenz der horizontalen Skalierbarkeit. Bei Systemen wie Couch DB oder Redis erfolgt eine Replikation der Daten mit dem Kommando „slave of 192.168.0.1 6379“ (IP und Port an entsprechenden Masterknoten anpassen).

- *Einfache API*

- *Meist BASE oder ein andersartiges Konsistenzmodell jedoch nicht ACID.* Das Verteilen von Datenbanksystemen hat zur Folge, dass nicht immer eine sofortige Konsistenz der Daten hergestellt wird. Im Gegensatz etwa zu Bankanwendungen ist dies bei Freundschaftsanfragen in Facebook nicht zwangsläufig erforderlich, da eine kurze Inkonsistenz der Daten nicht sicherheitskritisch ist.

Auch hier darf nicht von einer fehlerfreien Definition der NoSQL-Datenbanken ausgegangen werden, da Systeme wie MySQL oder PostgreSQL ebenfalls eine Open Source Lizenz aufweisen. Des Weiteren existieren mittlerweile mehr als 100 verschiedene Systeme, die nicht ausnahmslos nach dem Konsistenzmodell BASE verfahren, sondern gleichermaßen nach dem relationalen Modell ACID betrieben werden.

3.1.1 Vertikale/Horizontale Skalierung

Die exponentiell steigenden Anforderungen an die Datenverarbeitung von IT-Unternehmen führen dazu, dass vertikales Skalieren meist nicht mehr ausreichend und eine horizontale Skalierung unumgänglich ist.

Das vertikale Skalieren steigert dabei das Leistungspotenzial einzelner Server durch Aufrüsten alter CPU´s, dem Hinzufügen von Speicher, dem Austausch von RDIMMS gegen LRDIMMS sowie der Transformation von HDD gegen Enterprise SSD Festplatten.

Die horizontale Skalierung potenziert das Leistungspotenzial von modernen Rechenzentren durch die Verwendung mehrerer Server und erzielt damit deutlich höhere Datenverarbeitungsraten. [33]

Vertikale Skalierung	Horizontale Skalierung
Leistungssteigerung kostengünstig	Höhere Strom- und Kühlkosten
Einfache Leistungssteigerung (CPU, RAM)	Höhere Lizenzkosten

Niedriger Stromverbrauch	Hohe Netzwerkauslastung -> Hohe Kosten
Geringe Lizenzkosten	Hoher Platzbedarf
Niedrige Netzwerkbelastung	Vertikale Skalierung weiterhin möglich
Geringer Platzbedarf im Rechenzentrum	Hohe Leistungsrate trotz Standard Servern
Keine langfristige Lösung -> Server erreichen Leistungsgrenze	

Tabelle 1: Vertikale/Horizontale Skalierung([27], S. 377)

3.1.2 CAP-Theorem/Brewer Theorem

Mit Beginn des Internet-Booms wurde schnell klar, dass freie relationale Datenbanken wie PostgreSQL und MySQL bereits mit den Anforderungen von mittelgroßen Datenmengen an ihre Grenzen stoßen. Durch vertikales Skalieren, dem Einsatz leistungsfähiger Hardware, konnte die große Datenmenge und Vielfalt nicht mehr bewältigt werden. Auf dem ACM-Symposium (Association for Computing Machinery) im Jahr 2000 diskutierte Eric Brewer in seinem Referat über die horizontale Skalierung und zwar dem Einsatz mehrerer Server.

Das Eric Brewer Theorem/CAP-Theorem steht für die Vereinbarkeit von Konsistenz (Consistency), Verfügbarkeit (Availability) und Ausfalltoleranz (Partition Tolerance). Anhand des Theorems zeigt Brewer, dass es nicht möglich ist, alle drei Größen in einem verteilten System zur selben Zeit zu realisieren. Maximal zwei von drei Größen können zeitgleich durchgeführt werden, was dazu führt, dass in der Praxis in der hohe Konsistenz und Partitionstoleranz gefragt ist, die Verfügbarkeit leidet.

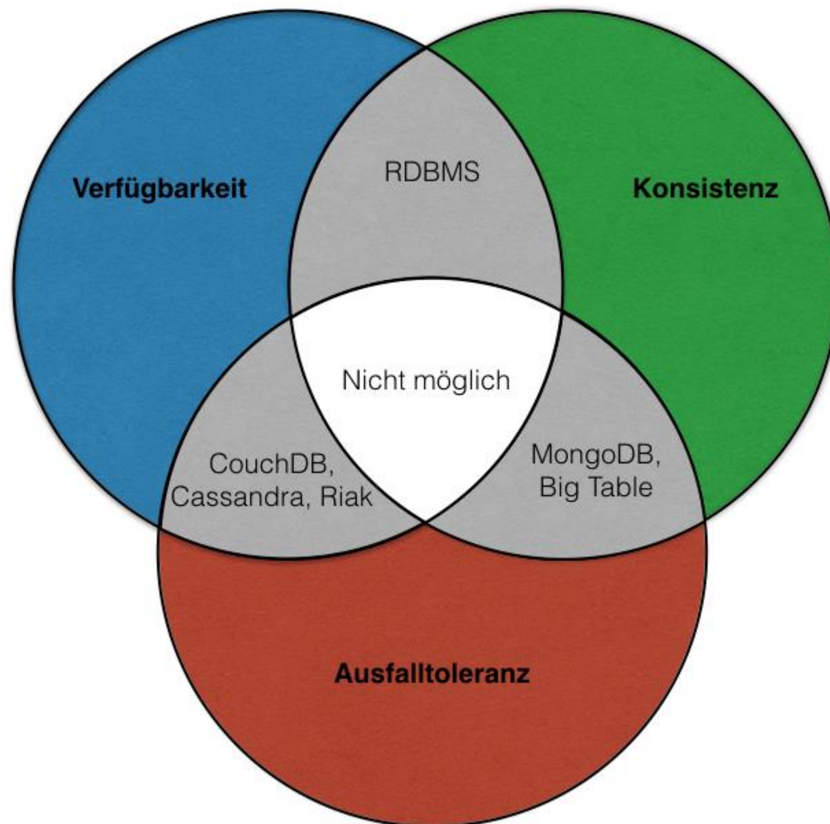


Abbildung 2: CAP-Theorem/Eric Brewer ([27], S. 33)

Konsistenz (Consistency): Bei einem konsistenten NoSQL-System sind komplexe Operationen, die aus vielen Teiloperationen bestehen, dann konsistent, wenn nach Abschluss sämtlicher Transaktionen alle Konten im System denselben Eintrag besitzen. Bei inkorrekten Daten reagiert die Datenbank mit einer Fehlermeldung sowie dem Zurückweisen der Daten, bis diese einen konsistenten Zustand erreicht haben. Ein Lesezugriff kann erst dann erfolgen, wenn alle Transaktionen auf dem replizierten Knoten abgeschlossen sind. Erfolgen Aktualisierungen in sehr großen verteilten Systemen, können längere Zugriffszeiten die Folge sein.([27], S.31 ff.)

Verfügbarkeit (Availability): Die Verfügbarkeit setzt in einem verteilten System akzeptable Antwortzeiten voraus. Dies ist besonders für Betreiber von E-Commerce Plattformen wie Amazon und Ebay essenziell, da die Frusttoleranz kaufwilliger Kunden bei Wartezeiten auf Anfrageergebnisse deutlich niedriger als bei Mitarbeitern ist. Die Verfügbarkeit ist ein zentraler Bestandteil, da zahlreiche Studien zeigen, dass Auftraggeber aufgrund höherer Antwortzeiten oftmals umgehend den Anbieter wechseln.

Ausfalltoleranz (Partition Tolerance): Die Ausfalltoleranz bedeutet, dass der Zugriff trotz geplanter (Wartung) oder ungeplanter (technischer Defekt) Serverausfälle weiterhin gewährleistet wird. Die Anzahl der ausgefallenen Server muss dabei jedoch stets klei-

ner als N/2 sein, um eine uneingeschränkte Funktionsfähigkeit des Systems zu garantieren. In großen Rechenzentren ist es in höchstem Maße wichtig, dass Ausfälle vom System abgefangen werden, um die Erreichbarkeit zu jeder Zeit sicherzustellen.

Konsistenz & Ausfalltoleranz

Bei diesem Modell steht die Konsistenz sowie die Ausfalltoleranz der Daten an oberster Stelle und findet dementsprechend im Bereich von Banken besondere Anwendung. Dabei muss die korrekte Buchung von Geldbeträgen auch bei Störungen im Datenverkehr gewährleistet sein. Die Verfügbarkeit am Bankautomaten ist sekundär. Demzufolge werden geplante Transaktionen des Kunden bei Netzwerkstörungen nicht durchgeführt, sodass ein optimaler Schutz vor Fehlbuchungen besteht.

Konsistenz & Verfügbarkeit

Konsistenz und Verfügbarkeit sind meist bei relationalen Datenbanken zu finden, wodurch die Ausfalltoleranz marginal ist. Diese Systeme favorisieren leistungsstarke Netzwerke sowie Server, die die Portierung der Daten auf mehrere Rechner überflüssig macht.

Verfügbarkeit & Ausfalltoleranz

In Anwendungsfällen, in denen die Verfügbarkeit sowie die Ausfalltoleranz im Vordergrund stehen, ist nach dem Theorem von Brewer eine konsistente Datenhaltung unmöglich. In Abbildung 3 wird ein verteiltes System aus den Punkten P1 und P2 angenommen, das beispielsweise Kommentare oder Posts von Nutzern speichert. Dabei können einzelne Kommentare seitens des Nutzers bearbeitet, kommentiert oder gelöscht werden. Im Modell wird P1 die Daten ändern und P2 die modifizierten Daten auslesen. Hierzu starten P1 und P2 mit demselben Datensatz D0. Nach der Abänderung durch P1 werden die Daten D1 durch den Synchronisationsmechanismus S auf den aktuellen Stand gebracht, sodass P2 die neuen Daten D1 von P1 auslesen kann. Bei einem Netzwerkfehler wird die Netzwerkverbindung zwischen P1 und P2 getrennt, sodass eine Synchronisation durch S nicht möglich ist. Werden von P1 weitere Änderungen vorgenommen und der Datensatz von D1 auf D2 aktualisiert, kann S die Daten nicht mehr synchronisieren, sodass P2 nach wie vor auf den Datenbestand D1 zugreift. Dies hat zur Folge, dass die Konsistenz der Daten nicht garantiert ist und somit keine Kommunikation zwischen P1 und P2 besteht, da P1 blockiert ist.

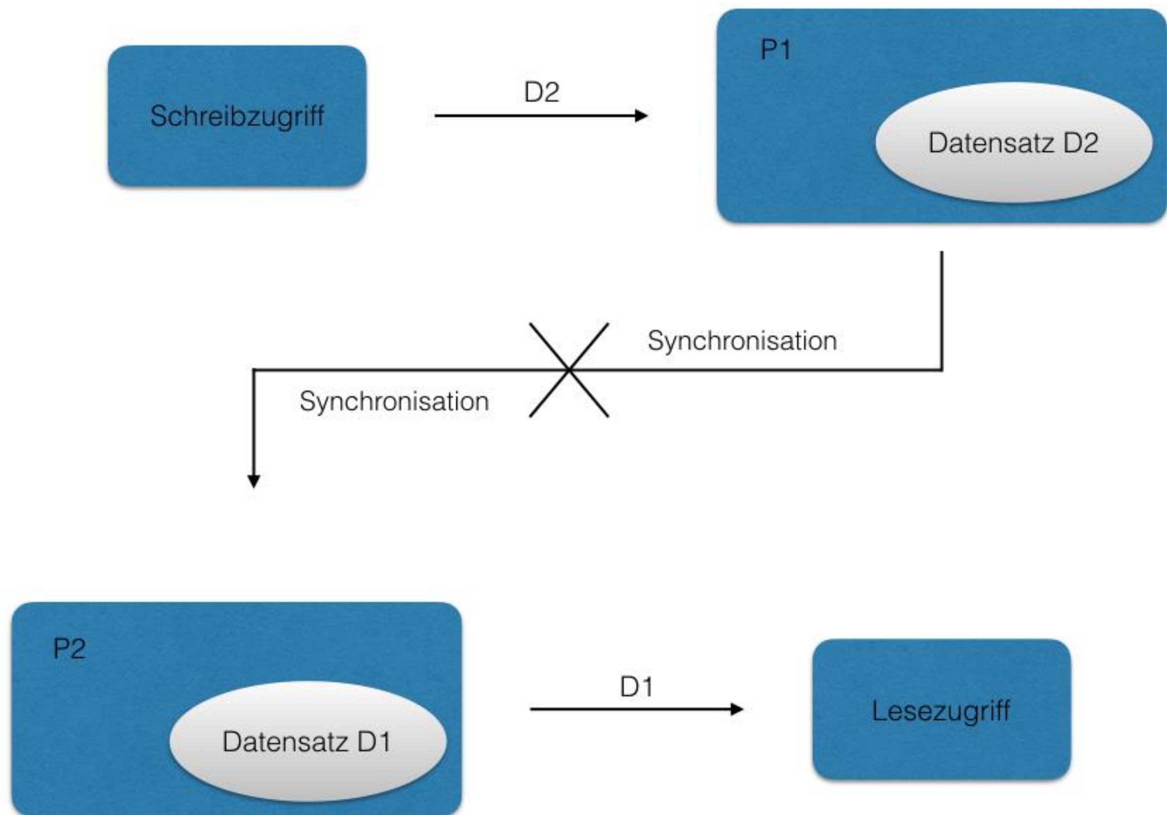


Abbildung 3: CAP-Theorem Verfügbarkeit und Ausfalltoleranz

3.1.2.1 Konsistenzmodell BASE

Das Konsistenzmodell BASE (Basically Availability, Soft State, Eventually Consistent) rekurriert die Probleme des CAP-Theorems und zeigt ein alternatives Konsistenzmodell. BASE betrachtet dabei die Konsistenz der Daten als sekundär und legt das Hauptaugenmerk auf die Verfügbarkeit. Dennoch wird die Konsistenz nicht außer Acht gelassen, sondern es wird angenommen, dass zu einem Zeitpunkt x die Daten permanent konsistent sind. Daraus resultiert, dass nicht unmittelbar nach jeder Transaktion eine Sicherstellung stattfindet, sodass Systeme mit einer Vielzahl an replizierten Daten dies erst nach einer großen Zeitspanne erreichen. Die maximale Dauer des Zeitfensters wird hierbei auf Basis von Faktoren wie der Zahl der replizierten Knoten, der durchschnittlichen Reaktionszeit sowie der durchschnittlichen Last des Systems bestimmt. ([27], S. 33 ff.)

Basically Available:

Der Ausfall einzelner Knoten verhindert nicht die Verfügbarkeit des Gesamtsystems, da von einer prinzipiellen Verfügbarkeit immer ausgegangen wird.

Soft State:

Die Modifikation der Daten erfolgt bis zu deren vollständigen Abschluss zunächst nur im Cache (flüchtiger Arbeitsspeicher) aus Gründen der Performance. Treten während dieser Phase Systemfehler oder Systemausfälle auf, können Änderungen verloren gehen.

Eventually Consistent:

Die Konsistenz der Daten tritt aufgrund der großen Anzahl an replizierten Knoten erst zu einem späteren Zeitpunkt ein.([36], S. 341 ff.)

Generell gelingt es auch dem Konsistenzmodell BASE nicht das Theorem von Eric Brewer außer Kraft zu setzen, sondern es bietet NoSQL-Datenbanken die Perspektive sich ein Stück weit vom Theorem zu lösen.

3.1.3 Map/Reduce

Map/Reduce ist das bekannteste Framework, das zur effizienten Verarbeitung großer Datenmengen 2004 von Google entwickelt wurde. Dabei kann die zu verarbeitende Datenmenge ausgehend von mehreren Terabytes bis hin zu vielen Petabytes umfassen. Die Grundidee liegt hierbei in der Parallelisierung von Prozessen, wodurch sich vor allem funktionale Sprachen definieren. Trotz Parallelisierung treten keine Seiteneffekte wie „deadlocks" oder „race conditions" auf. Jede Operation, die während der Map/Reduce-Phase ausgeführt wird, arbeitet auf einer Kopie des originalen Datensatzes, wodurch sich Operationen nicht gegenseitig behindern und im Fehlerfall auf den Originaldatensatz zurückgegriffen werden kann. Die Map-Funktion des Algorithmus wird als Erstes sukzessiv auf jedes Element der Eingabeliste angewendet und teilt die Liste in kleine Arbeitspakete (Schlüssel-Wert-Paare), die der Anzahl der verschiedenen Mapper im Cluster entspricht. Jedes Arbeitspaket kann zu Beginn unabhängig von einem Mapper bearbeitet werden. Nach erfolgreicher Bearbeitung werden die Daten zwischengespeichert. Im Anschluss daran werden alle Tupel mit gleichem Schlüssel zusammengefasst, sodass jeder Schlüssel nur einmal existiert und somit eindeutig ist.([27], S. 12 ff.) In der Reduce-Phase wird anschließend für jedes Zwischenergebnis ein Endergebnis berechnet, das folglich im Dateisystem abgespeichert wird. Die Reduce-Funktion endet, nachdem alle Ergebnisse berechnet sind. Abbildung 4 zeigt anhand eines fiktiven praktischen Beispiels die einzelnen Schritte von Map/Reduce.

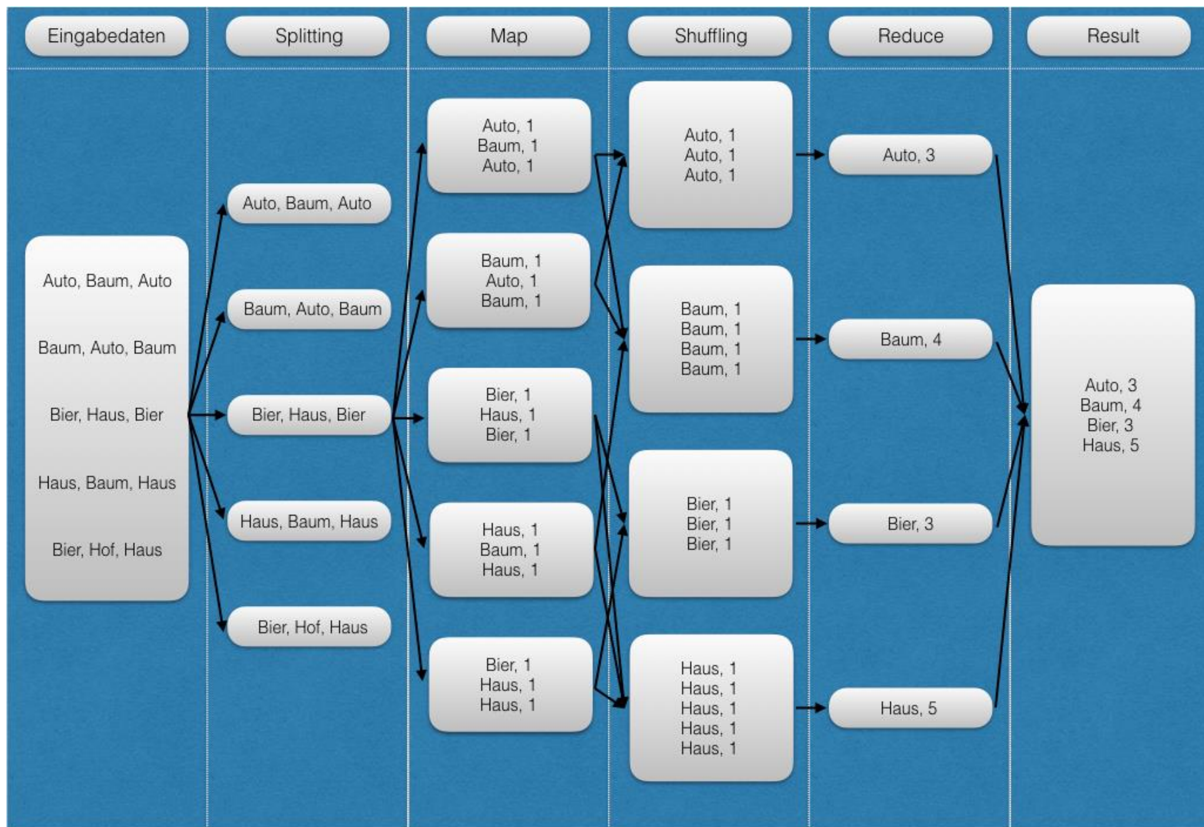


Abbildung 4: Map/Reduce Beispielgrafik [14]

Abbildung 5 zeigt ein erweitertes Modell von Map/Reduce mit Implementierung eines zusätzlichen Masterknotens. Dieser wird zunächst bestimmt und übernimmt die Koordination der restlichen Knoten (Mapper, Reducer) sowie die Datenverteilung. Die entsprechenden Map-Funktionen werden dabei auf die Mapper (Sklaven) aufgeteilt. Während der ersten Berechnungsphase werden diese in bestimmten Zeitintervallen vom Master überwacht, damit der aktuelle Berechnungsstatus sowie die fehlerfreie Funktion der Mapper sichergestellt ist. Des Weiteren wird vom Master die Koordination der Reduce-Knoten übernommen.

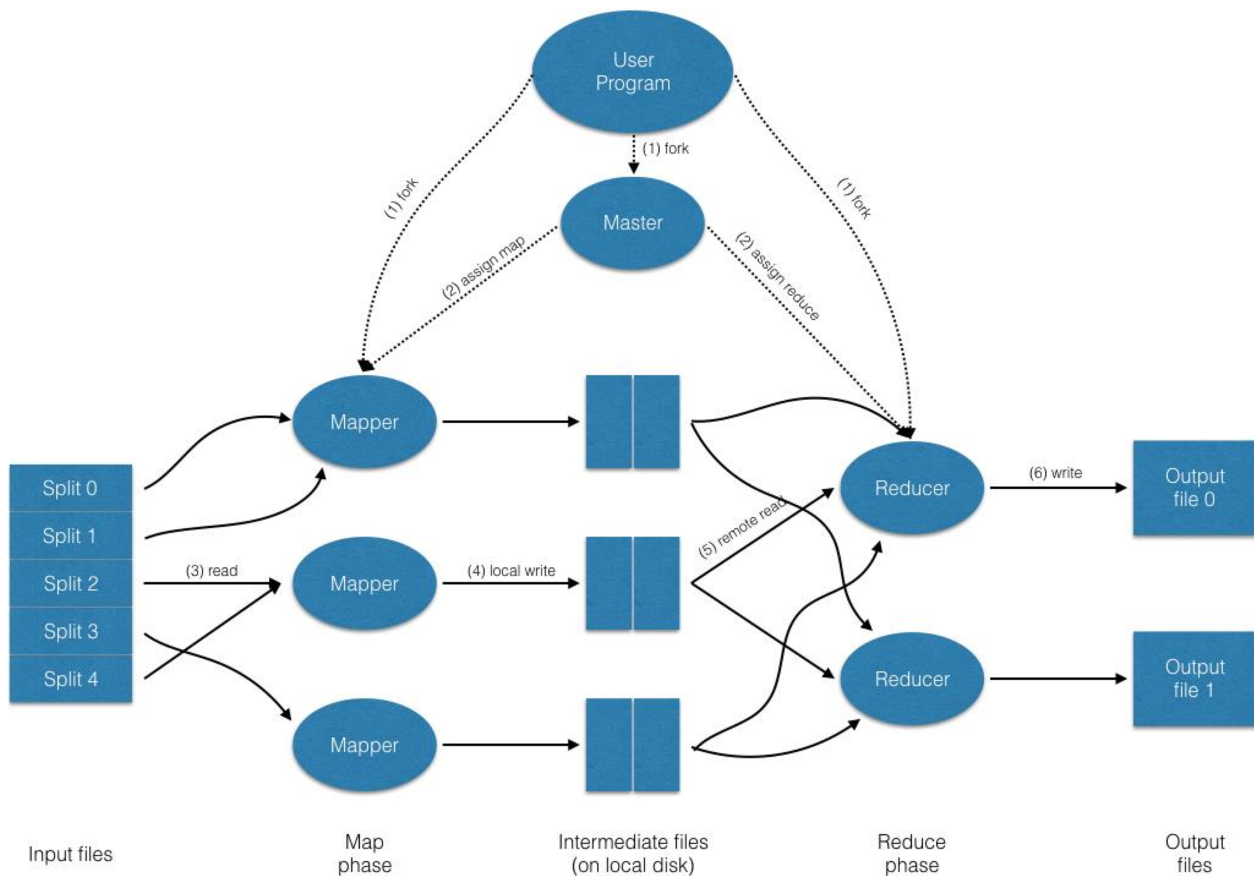


Abbildung 5: Map/Reduce Erweiterung ([8], S. 10)

Während der Berechnung treten bisweilen Hardware- oder Netzwerkfehler auf, wodurch Teilsysteme nicht erreichbar sind. Durch die periodische Überwachung aller Knoten werden jedoch fehlerhafte Rechner ausfindig gemacht und deren Jobs vom Master an neue funktionsfähige Rechner weitergegeben. Darüber hinaus kann es auch zu einer Fehlfunktion des Masters kommen. Um dessen Funktionsfähigkeit zu sichern, werden regelmäßige Backups (Checkpoints) durchgeführt. Zudem ist nach einem Ausfall die Zurücksetzung auf den zuletzt gespeicherten Checkpoint möglich. ([27], S. 14 ff.)

3.1.4 Consistent-Hashing

Konsistente Hash-Funktionen (engl. consistent hashing) dienen dazu aus einer Menge an Schlüsseln eine Speicheradresse oder einen Hash-Wert zu berechnen, an dessen Stelle der Datensatz geschrieben wird. Consistent-Hashing ist auf die klassische Hashtabelle zurückzuführen, bei der jedes Element den Ablageort (Eintrag) in der Hashtabelle bestimmt. Der Vorteil eines solchen Verfahrens liegt in der konstanten Laufzeit O(1) im Gegensatz

zu B-Bäumen. Besondere Anwendungen erfahren die Hashfunktionen in verteilten Netzwerken, da jeder Teilnehmer einen Bereich des Datensatzes speichert und verwaltet. In der Praxis werden dem Peer-to-Peer Netzwerk meist neue Knoten hinzugefügt oder alte entfernt, was dazu führt, dass bestehende Einträge an unzutreffender Stelle wiederzufinden sind. Ursache hierfür ist das „Vergrößern bzw. Verkleinern" im Netzwerk, da sich dadurch auch die Hashfunktion und somit die Hashtabelle ändert. Zur Vermeidung ständiger Neuberechnungen wird Consistent-Hashing eingesetzt. Wird ein neuer Teilnehmer in das Peer-to-Peer Netz integriert, so erfolgt die Anordnung gemäß seines Hash-Werts und alle Datensätze, die im Uhrzeigersinn vor dem neuen Server liegen, werden auf diesen kopiert. Demnach bleiben alle übrigen Datensätze davon unberührt.([9], S. 127 ff.)

Wird ein Server entfernt, werden die Datensätze des betroffenen Servers auf den im Uhrzeigersinn nachfolgenden Server kopiert, dadurch sind weitere Datensätze von dieser Maßnahme nicht betroffen. Verfügt ein Server über größere Kapazitäten als sonstige, so erhält dieser eine Vielzahl von Hashwerten, die der Anzahl und Leistungsfähigkeit des jeweiligen Servers entsprechen. Im Bereich von NoSQL gibt es mittlerweile zahlreiche Vertreter von Consistent-Hashing wie etwa Amazon Dynamo Paper.([22], S. 123) Das Verfahren hat auch großen Einfluss auf die Entwicklungen von NoSQL-Datenbanken wie beispielsweise Riak.

3.1.5 Vector Clocks

Datenbanksysteme werden mithilfe von Vector Clocks vor einem Serverausfall oder Netzwerkkonflikten zur Vermeidung inkonsistenter Datensätze geschützt. Das Konzept von Vector Clocks weist entsprechenden Datensätzen einen eindeutigen Zeitstempel zu. Durch das Vektorisieren wird sichergestellt, dass Kausalzusammenhänge weiterhin bestehen, auch bei einem Ausfall des Systems.

Jeder Prozess erhält einen inkrementierten Zähler zugewiesen, der beim Senden und Empfangen einer Nachricht erhöht wird. Zur Konfliktvermeidung muss der Prozess den Sender des Objekts sowie dessen Zeitstempel identifizieren.

Ein Beispiel für eine clientseitige Konfliktauflösung wird anhand des Warenkorbs von Amazon repräsentiert. Der Warenkorb muss dabei eine Vielzahl von Artikeln aufnehmen, ohne dass Schreiboperationen aufgrund von Netzwerkausfällen zurückgewiesen werden. Ebenso darf keinesfalls ein Datenverlust infolge eines serverseitigen Konflikts entstehen. Amazon Dynamo Paper ist daher auf eine minimale Schreibkonsistenz (w_1) konfiguriert, sodass nach jedem Schreibvorgang die Konsistenz der Daten generiert werden muss. ([27], S. 43 ff.) Mit diesem System verzeichnet Amazon in nur 0,06% aller Fälle Inkonsistenzen, die überwiegend durch konkurrierende Schreiboperationen auftreten. Dieses

Problem entsteht, sobald mehrere Familienmitglieder über einen Account an verschiedenen Rechnern Bestellungen bei Amazon tätigen.([13], S. 23)

3.2 NoSQL-Datenbanken

3.2.1 Key/Value-Stores

Key/Value-Stores (KVS) sind die einfachste Form von NoSQL-Datenbanken, da diese aufgrund ihres simplen Datenmodells und dem geringen Funktionsumfang unkompliziert zu verwenden sind. Die Systeme sind auf eine möglichst hohe Geschwindigkeit von einfachen Lese- und Schreiboperationen ausgelegt. Infolge ihrer minimalistischen Ausrichtung ist ihr Einsatzgebiet jedoch beschränkt.

Im nächsten Abschnitt wird die Funktionsweise von Key/Value-Datenbanken, insbesondere die von Redis präzisiert. Neben Redis zählen Amazon Dynamo, Project Voldemort und Riak zu den bekanntesten Repräsentanten. ([27], S. 151)

3.2.1.1 Datenmodell und Funktionsweise

Key/Value-Stores bestehen aufgrund ihrer Einfachheit exakt aus einem Schlüssel (Key) und einem Wert (Value). Der Wert des Datenobjekts wird über eine Schnittstelle in die Datenbank integriert. Dazu wird ein eindeutiger Schlüssel ausgegeben anhand dessen die Datenobjekte abrufbar sind. Existieren mehrere Schlüssel-Wert-Paare spricht man von sogenannten Buckets, die einer klassischen zweispaltigen relationalen Tabelle ähneln. Dabei enthält die erste Spalte die Primärschlüssel und in der zweiten finden sich die jeweiligen Attributwerte wider, siehe Abbildung 6. Das Bucket Person impliziert eine eindeutige Personenkennziffer sowie die zugehörigen Attributwerte.

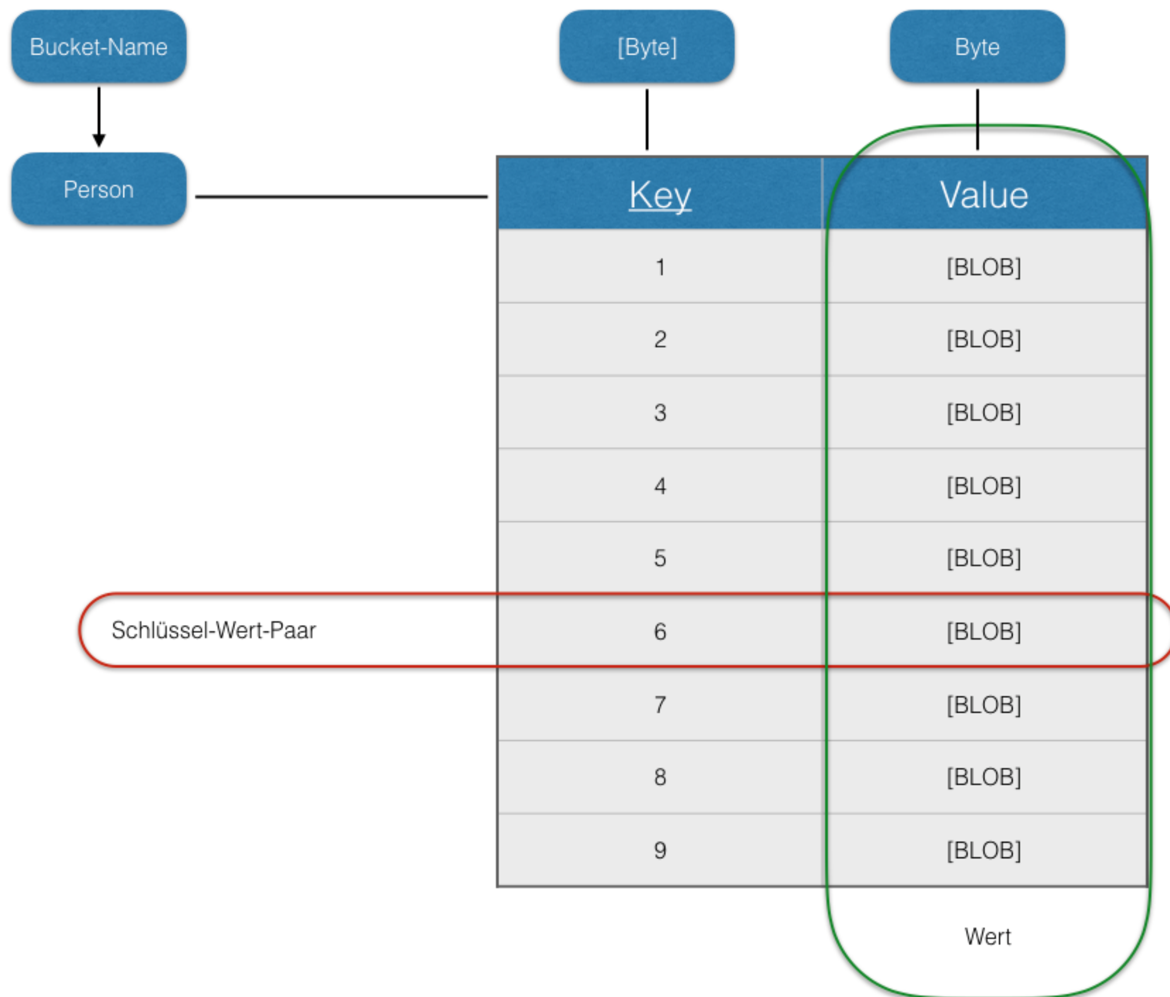


Abbildung 6: Key/Value-Store Datenmodell mit Entität Person

Die Speicherung von Keys und Values erfolgt in Byte-Arrays der Größe 0-255. Die Key/Value-Datenbank ermöglicht die Auswertung der Daten auf Anwendungsebene. Beim Lesen eines Key/Value-Paares werden die Paare in das entsprechende Datenformat überführt.

Key/Value-Datenbanken sind nicht befähigt, Metainformationen zu den verschiedenen Schlüssel-Wert-Paaren vorzuhalten. Infolgedessen ist die Speicherung der Namen von Primärschlüsseln oder von Attributen impraktikabel. Ungünstig wirkt sich dies vor allem auf die Primärschlüssel aus.([27], S. 152 ff.) Da die Eindeutigkeit nicht standardmäßig überprüft wird, resultiert daraus ein Datenverlust, da bereits existierende Schlüssel vom System überschrieben werden. Des Weiteren wird ein minimaler Funktionsumfang geboten, sodass Änderungsoperationen lediglich aus SET und DELETE bestehen. Der Operator SET zeichnet sich verantwortlich für das Abspeichern neuer Key/Value-Paare, wogegen der Operator DELETE zur Entfernung autorisiert ist. Eine Alternative, die vorhandenen Daten mithilfe eines Update-Befehls zu aktualisieren, besteht nicht. Bei einem

```
> SET    <Key> <Value>
> DEL    <Key>
> GET    <Key>
```

erforderlichen Update des Datenbestands wird zunächst mit DELETE entfernt und anschließend mit SET neu hinzugefügt. Das Auslesen erfolgt mit GET.

Key/Value-Stores offerieren die Option, Daten auf mehrere Server zu verteilen. Mithilfe des eindeutigen Schlüssels stellt das Aufteilen des Datensatzes kein Problem dar. Durch die Hash-Funktion wird die Zuständigkeit der einzelnen Server für den jeweiligen

Abbildung 7: Operationen von Key/Value-Datenbanken

Datensatz berechnet. Ebenso kann im Umkehrschluss anhand der Hash-Funktion der entsprechende Eintrag ausgelesen werden. Darüber hinaus werden Replikationen zur Ausfallsicherung erstellt. Die Datenbank Redis bringt hierzu das Master-Slave Prinzip zum Einsatz, bei dem jedem Masterknoten mehrere Slaves zugeordnet sind.

Key/Value-Datenbanken eignen sich insbesondere für sehr einfache Datenbanksysteme, die nicht die Prätention haben möglichst viele Elemente in Relation zueinander zu stellen, sondern in denen einseitige Beziehungen existieren. Aufgrund der Simplizität bietet das System sehr hohe Lese- und Schreiboperationen.

3.2.1.2 Beispielanwendung „Redis“

Redis zählt zu den namhaften Vertretern der Key/Value-Stores und wurde, wie das Gros der NoSQL-Systeme, Anfang 2009 von Salvatore Sanfillippo zunächst als Ein-Mann-Projekt kreiert und praktiziert. Redis ist eine In-Memory Datenbank, deren Einträge ausschließlich im Hauptspeicher verwaltet und gespeichert werden. Durch die Verwendung von In-Memory wird das System enorm beschleunigt. Bei einem Ausfall des Servers ergeben sich daraus negative Auswirkungen, die in einem möglichen Datenverlust enden. Redis bietet jedoch differente Mechanismen zum Schutz des Users vor Datenverlust. Dazu werden die Daten zyklisch auf die Festplatte übertragen oder durch „Append Only Mode“ zur Rekonstruktion in Logfiles transportiert.

Die Installation von Redis ist innerhalb kürzester Zeit realisierbar und bedarf nur einem Minimum an Befehlen.([27], S. 153 ff.) Unter Ubuntu erfolgt die Konfiguration mit *sudo apt-get install redis-server.* Alle erforderlichen Abhängigkeiten werden vom System unverzüglich aufgelöst, sodass manuelle Paketinstallationen überflüssig sind. Das Starten des Servers mit Ubuntu erfolgt durch *./redis-server.*

```
dennis@dennis-Aspire-3820T:/usr/bin$ ./redis-server
3847:C 17 Feb 11:58:36.530 # Warning: no config file specified, using the default config. In order to specify a config file use ./redis-server /path/to/redis.conf
3847:M 17 Feb 11:58:36.531 * Increased maximum number of open files to 10032 (it was originally set to 1024).

                Redis 3.0.6 (00000000/0) 64 bit

                Running in standalone mode
                Port: 6379
                PID: 3847

                      http://redis.io

3847:M 17 Feb 11:58:36.532 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
3847:M 17 Feb 11:58:36.532 # Server started, Redis version 3.0.6
3847:M 17 Feb 11:58:36.532 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
3847:M 17 Feb 11:58:36.532 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.
3847:M 17 Feb 11:58:36.532 * The server is now ready to accept connections on port 6379
```

Abbildung 8: Start des Redis Servers

Zunächst ist die Anlegung des Ports durch den Nutzer maßgeblich, um eine künftige Kommunikation mit den einzelnen Clients zu gewährleisten. Dazu wird im Anschluss der Client gestartet und auf denselben Port wie der Server konfiguriert. In der Folge ist es durchführbar, Daten mit den Befehlen SET und GET zu schreiben und auszulesen. Die untere Abbildung indiziert einen exemplarischen Anwendungsfall, der anhand Redis Sets konzipiert wurde. Redis Sets sind dazu dienlich, eine ungeordnete Sammlung von einzigartigen Strings zu erzeugen. Bei dieser Beispielanwendung benötigt das System permanent eine Laufzeit O(1) in puncto Testen, Hinzufügen oder Löschen.

```
dennis@dennis-Aspire-3820T:/usr/bin$ ./redis-cli
127.0.0.1:6379> SADD UNI "Hochschule Anhalt"
(integer) 1
127.0.0.1:6379> SADD UNI "Anhalt University of Applied Sciences"
(integer) 1
127.0.0.1:6379> SADD UNI "Bernburger Str. 55"
(integer) 1
127.0.0.1:6379> SADD UNI "06366 Koethen"
(integer) 1
127.0.0.1:6379> SADD UNI "Tel.: +49 (0) 3496 67 1000"
(integer) 1

127.0.0.1:6379> SMEMBERS UNI
1) "Anhalt University of Applied Sciences"
2) "Bernburger Str. 55"
3) "Hochschule Anhalt"
4) "06366 Koethen"
5) "Tel.: +49 (0) 3496 67 1000"
127.0.0.1:6379>
```

Abbildung 9: Redis Sets Beispielanwendung mit HS Anhalt

Redis verfährt nach dem Grundsatz „keep it simple“ und ist dementsprechend speziell für einfache Datenmodelle geeignet, die eine hohe Performance und Skalierbarkeit mit sich bringen.

3.2.2 Document Stores

Document Stores präsentieren eine Vielzahl an Perspektiven gegenüber Key/Value-Stores und konstituieren somit einen Kompromiss zwischen einfachen NoSQL-Systemen und relationalen Datenbanken. Document Stores punkten vor allem bei den semistrukturierten Daten, da diese besonders effizient verwaltet werden können. Analog der Key/Value-Stores kann diese Variante sehr einfach repliziert werden. Die renomiertesten Vertreter sind MongoDB und CouchDB.([27], S. 117)

3.2.2.1 Einleitung und Funktionsweise

Document Stores sind zur Verarbeitung von Daten in Dokumenten autorisiert. Hierbei handelt es sich jedoch weder um Word- noch um Textdokumente, sondern um strukturierte bzw. semistrukturierte Daten, die üblicherweise in Formaten wie Java Script Object Notation (JSON), Binary Script Object Notation (BSON) oder Extensible Markup Language (XML) vorliegen. Document Stores existieren seit 1984, als IBM mit deren Groupware System Lotus Notes große Erfolge feierte. Besonders zur Verarbeitung komplexer XML Dokumente ist diese Datenbank sehr qualifiziert. Das XML Format wurde in den vergan

genen Jahren jedoch zunehmend durch JSON ersetzt. Die Gründe hierfür sind vielfältig. Primär bei kleineren Anwendungen ist XML zu komplex und ineffizient. Durch den Verzicht von Auszeichnungselementen (Tags) bieten JSON und BSON eine bessere Übersicht und erleichtern das Bearbeiten seitens des Anwenders.

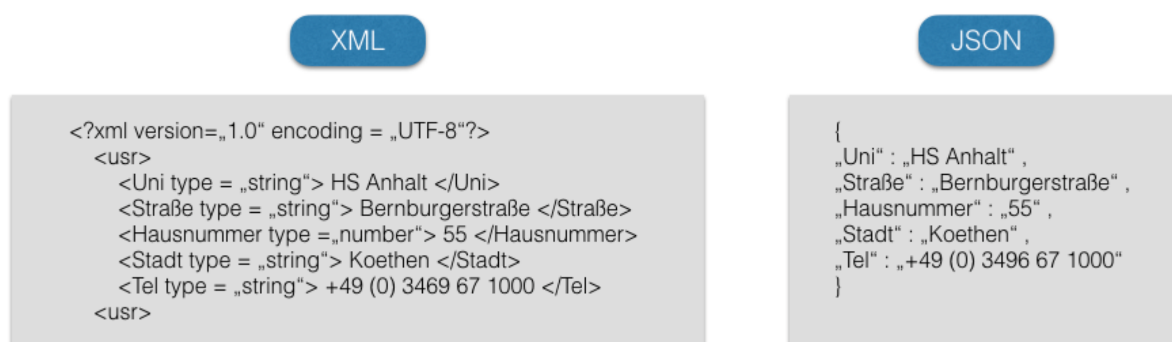


Abbildung 10: XML und JSON im direkten Vergleich

Document Stores reproduzieren Datensätze als eigenständige Dokumente. Dabei werden die Attribute sowie die Attributwerte mithilfe von Key/Value entsprechend abgelegt. Die zugewiesenen Keys erhalten einen eindeutigen Wert und identifizieren die abgelegten Dokumente präzise. Das JSON-Objekt kann eine Reihe von primitiven Datentypen als Schlüssel-Wert-Paar speichern, siehe Abbildung 10. Hierbei handelt es sich um String, Number, Boolean und Null. Mit den Datentypen Array und Objekt sind zudem Verschachtelungen beliebiger Tiefe denkbar. BSON stellt zusätzlich zwei weitere Datentypen „Date“ und „BLOB“ zur Verfügung. Abbildung 11 zeigt die Entität Person angewendet auf das Datenmodell eines Document Stores. Jede Person kann über ihre jeweilige ID evident identifiziert werden. Die Attribute und Attributwerte werden durch die entsprechenden Schlüssel-Wert-Paare repräsentiert. Alle Dokumente im Datenbankmodell CouchDB sind auf eine maximale Größe von 16 MB begrenzt. Wird diese Größe überschritten, kommt es zu einer Aufteilung auf mehrere Dokumente, ohne dabei deren kausale Abhängigkeit aufzulösen. Document Stores verwenden ähnliche Speicherstrukturen wie relationale Datenbanken, die Einträge persistent auf der Festplatte speichern. Die Dokumente werden zunächst im Hauptspeicher abgelegt, bevor diese auf den Sekundärspeicher gelangen, um schnelle Zugriffszeiten zu gewährleisten. Document Stores favorisieren als primären Zugriffspfad B*-Bäume.([27], S: 177 ff.)

Generell stellt dieses Modell der NoSQL-Datenbank einen Kompromiss zwischen den leicht gewichtigen Key/Value-Stores und den mächtigen relationalen Datenbanken dar. Trotz der schemafreien Daten wird die Option geboten in JSON-Dokumenten, Metainformationen wie Attributnamen und Datentypen zu speichern. Dadurch ist es realisierbar sämtliche Anfrageoperationen relationaler Datenbanken auszuführen. Darüber hinaus werden Dokumente nicht fragmentiert, sodass jedes Dokument atomar auf einen Server verteilt wird.

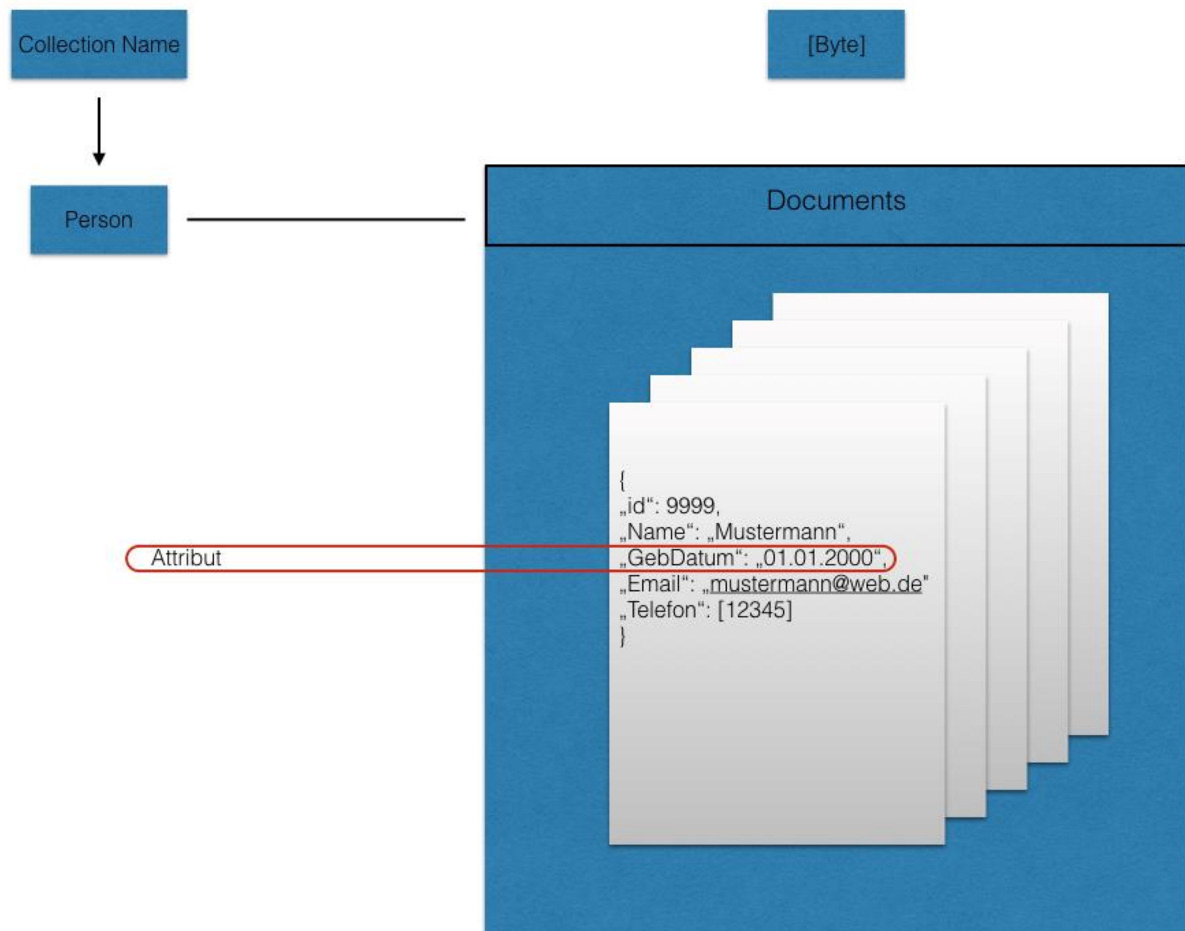


Abbildung 11: Beispielhafter Document Store mit Entität Person

3.2.2.2 Beispielanwendung „CouchDB“

CouchDB gehört zu den etablierten NoSQL-Systemen, deren Anspruch es ist, den Herausforderungen des Web 2.0 standzuhalten. Die Datenbank orientiert sich in hohem Maße an Googles Big Table und greift auf das Map/Reduce-Verfahren (siehe Kapitel 3.1.3) zurück. Die schemafreie und dokumentenorientierte Datenbank administriert deren Daten im JSON-Format. Dabei erfolgen alle Zugriffe über *http* mittels der Restful JSON-API. Die Daten werden in B-Bäumen abgelegt, wodurch jedes Dokument über eine Dokumenten-ID sowie eine Revisions-ID verfügt. Bei einer Modifikation der Daten wird der Zähler der Revisions-ID inkrementiert. Die punktuellen Eigenschaften eines JSON-Objekts werden durch ein Komma abgetrennt, sodass eine Analogie zu den Tupeln des Key/Value-Stores besteht.

Der Anfrageverarbeitung unter CouchDB stehen ähnlich den Key/Value-Stores die Operationen GET, PUT und DELETE zur Verfügung. Zur Integration neuer Kollektionen

muss die Operation PUT ausgeführt werden. Diese benötigt hierzu die URL des Dokuments sowie den JSON-Code.

put http://<couchDB>/<collection>/<id><json>

Eine weitere Parallele zu den Key/Value-Stores stellt die fehlende Updatefunktion dar. Bei einer erforderlichen Aktualisierung eines Dokuments muss dies zunächst aus der Datenbank gelesen, um anschließend auf Applikationsebene modifiziert zu werden. Durch einen erneuten Zugriff wird das überarbeitete Dokument gespeichert. Das chronologisch ältere Dokument wird in der Folge anhand der Revisions-ID als ungültig markiert. Die Realisierung des DELETE-Befehls erfolgt unter Zuhilfenahme der Dokumenten-ID sowie der Versionsnummer.

Mit der Methode GET können zuvor gespeicherte JSON-Dokumente aufs Neue ausgelesen werden.

get http://<couchDB>/<collection>/<id>

Neben den Basisoperatoren besteht zudem die Option, komplexere Anfragen zu formulieren. Dazu sind Design-Dokumente praktikabel, die mit verschiedenen Views Aggregationen bewerkstelligen, ohne dass die zugrunde liegenden Dokumente editiert werden.([27], S. 118 ff.)

3.2.3 Wide Column Stores

Column Family Stores wurden zur Verarbeitung großer Datenmengen in verteilten Systemen entwickelt und basieren auf dem Konzept von Googles Big Table. Bedeutende Vertreter sind Cassandra, HBase, Hypertable und Big Table. Speziell die Innovation von Big Table verhalf den Column Family Stores zum Durchbruch. Google hat dazu ein System entwickelt, das neben der effizienten Verarbeitung großer Datenmengen vorrangig lineare horizontale Skalierbarkeit, große Performance und permanente Verfügbarkeit impliziert. Bemerkenswert sind vor allem die vielfältigen Einsatzmöglichkeiten von Big Table. Sie finden ihre Anwendung in über sechzig Produkten von Google darunter Google Earth, Google Analytics, Google Mail und Google Index.([27], S. 7)

3.2.3.1 Einleitung und Funktionsweise

Column Family Stores verfolgen prinzipiell das Konzept von relationalen Datenbanken, ohne auf feste Schemen oder statische Daten zu beharren. Die Anfragesprache weist

dabei SQL ähnliche Strukturen auf, wodurch die Daten ausgelesen und manipuliert werden können. Die Datenhaltung erfolgt im Gegensatz zu relationalen Datenbanken zwar ebenfalls in Tabellen, jedoch werden die Einträge in Spalten und nicht in Zeilen abgelegt. Jeder Eintrag besteht aus einem Primärschlüssel (meist Name der Spalte), den Daten sowie einem timestamp (Zeitstempel). Attribute können zudem spaltenübergreifend zusammengefasst werden, woraus sich der Begriff der Column Family ableitet. In einer Column Family existiert keine logische Struktur, sodass diese beliebig viele Columns enthalten kann. Eine spätere Identifizierung erfolgt über die Keys.

Durch das Speicherverfahren in Byte Arrays werden neben primitiven Datentypen komplexe Objekte und Bilder in der Datenbank abgelegt. Byte Arrays können vom Datenbanksystem nicht interpretiert werden. Attributwerte müssen demzufolge aufseiten der Applikationslogik geparst werden, um deren Abhängigkeiten aufzulösen.

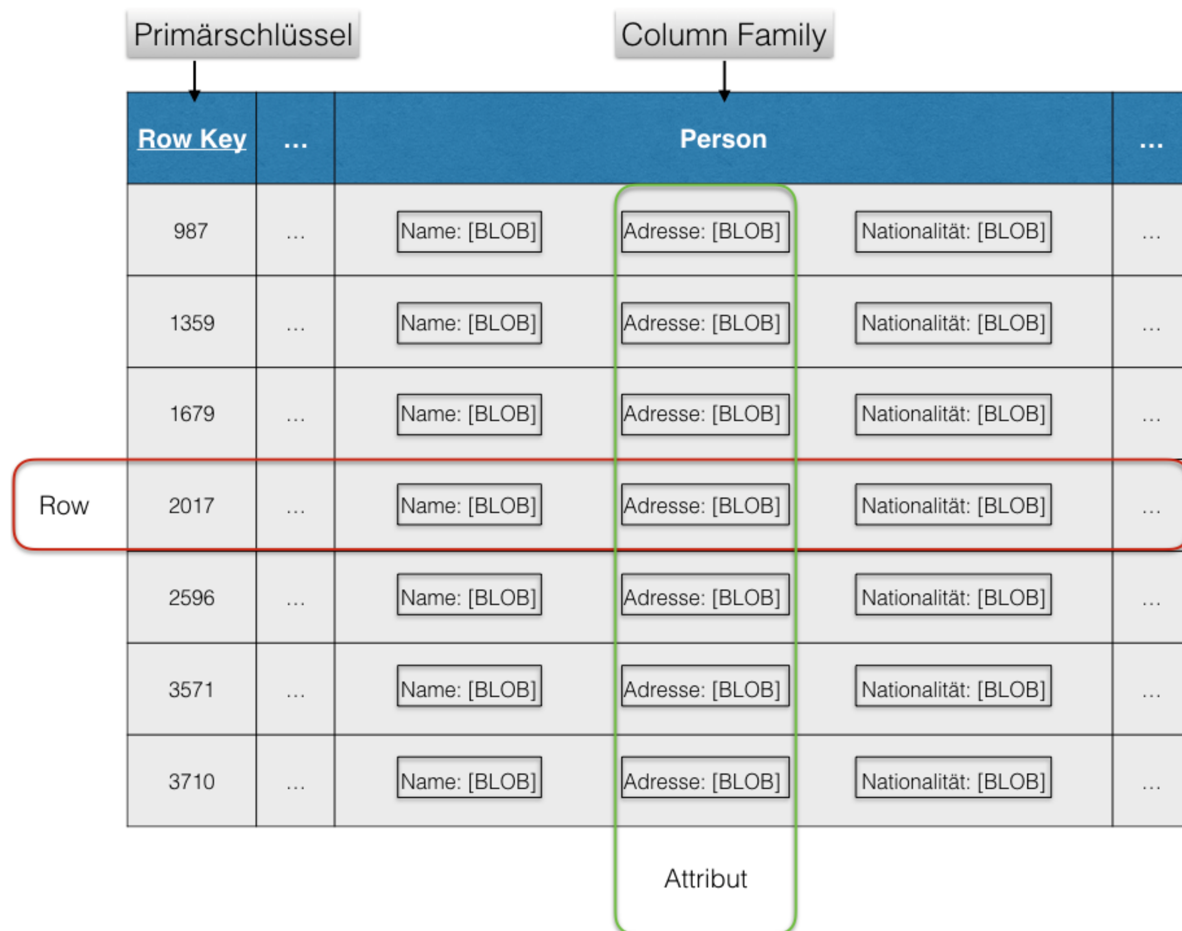


Abbildung 12: Wide Column Store Datenmodell mit Entität Person

Abbildung 12 zeigt exemplarisch ein Datenmodell mit Entität Person. Die ID vom Datentyp String stellt den Primärschlüssel dar. Jeder ID sind diverse Werte der Column Family zugeordnet. Diese bilden zusammen ein Schlüssel-Wert-Paar, das ein Attribut sowie dessen Wert enthält. Der hier abgebildete und limitierte Datensatz kann um bis zu zwei Milliarden Schlüssel-Wert-Paare erweitert werden.

Bei der Datenintegrität speichern weder Column Family Stores noch Key/Value-Stores weitere Metainformationen außer den Column Family Namen. Des Weiteren findet auch keine Prüfung der Primärschlüssel auf Eindeutigkeit statt, sodass durch Einfüge-Operationen möglicherweise bereits existierende Primärschlüssel überschrieben werden, was einen Datenverlust zur Folge hat. Die fehlende Eindeutigkeitsprüfung darf jedoch nicht pauschalisiert werden. So bildet Cassandra hierbei eine Ausnahme, da Schreiboperationen mithilfe eines Validators überprüft werden.([27], S. 7)

Prinzipiell erfolgt die Interaktion der Datenbank, wie auch von sonstigen NoSQL-Systemen bekannt, über die Kommandozeile mit den bereits vorgestellten Operationen PUT, GET, DELETE. Die unmittelbare Aktualisierung kann dagegen nicht bewerkstelligt werden, aufgrund dessen wird der alte Datensatz überschrieben bzw. ein neuer angelegt. Schreiboperationen finden grundsätzlich nach folgender Syntax statt:

put <Tabellen Name>, <Row Key>, <Column Family:Column>, <Wert>

Zusätzlich besteht die Alternative mehrere Datensätze pro Operation zu verarbeiten. Datensätze können zudem mit einem Zeitstempel (Time to Live – TTL) versehen und nach dem Ablaufdatum automatisch entfernt werden.

Lesezugriffe auf die Datenbank erfolgen mit GET und verfahren nach folgender Syntax:

get <Tabellen Name>, <Restriktion>, <Projektion>

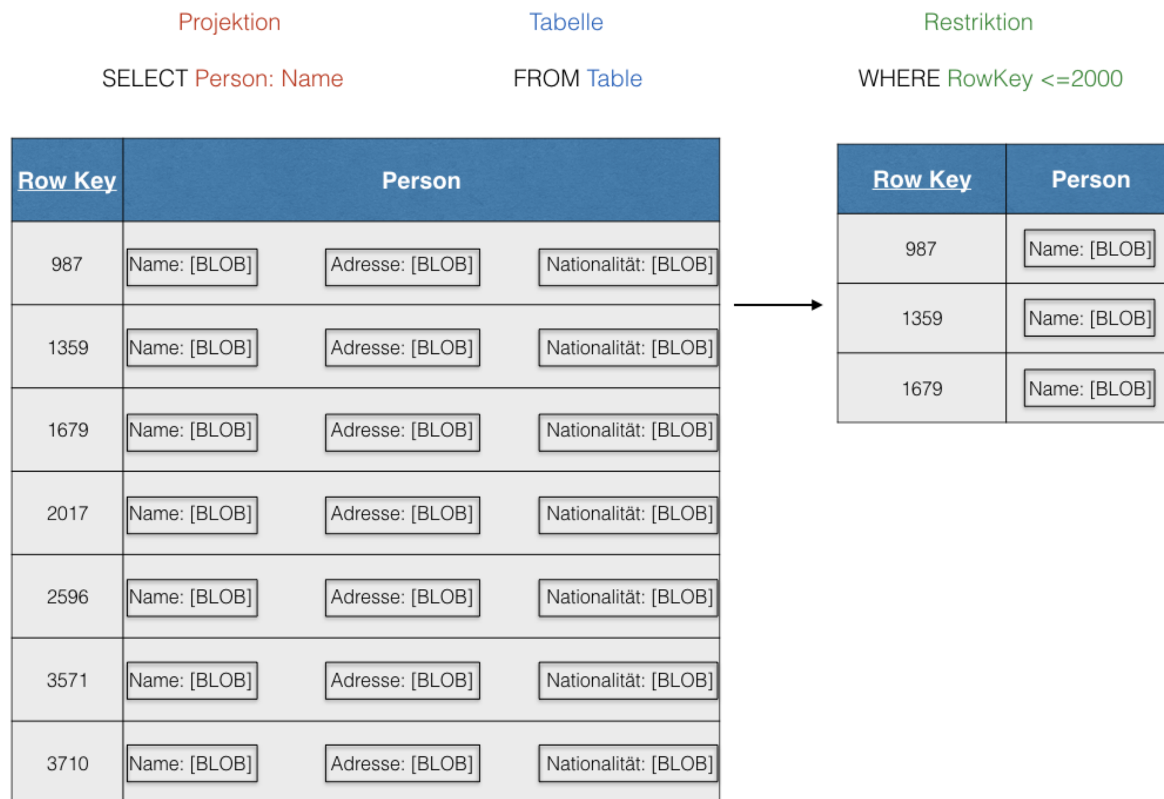


Abbildung 13: Formulierung von Restriktions- und Projektionsbedingungen

Bei einem Lesezugriff ist die Projektionsbedingung nicht obligatorisch, sondern es wird lediglich die Ergebnismenge der Attribute (Columns) weiter eingeschränkt. Restriktionsbedingungen müssen immer verwendet werden und bestehen aus regulären Ausdrücken. Die Anwendung ist ausschließlich auf den Primärschlüssel praktikabel und schließt Restriktionen auf Ergebnismengen aus.

3.2.3.2 Beispielanwendung „Cassandra“

Cassandra wurde von Facebook kreiert und ist seit 2008 unter der Open Source Lizenz frei zugänglich. Im Gegensatz zu HBase und Googles Big Table, ebenfalls den Column Family Stores zuzuordnen, wird ein hybrider Ansatz geboten. Obendrein enthält Cassandra Key/Value-Eigenschaften sowie eine flexible Schemaunterstützung, wodurch größtmögliche Flexibilität als auch Skalierbarkeit gewährt wird und dennoch Gemeinsamkeiten zu SQL-Systemen existieren.

Das Spezifikum von Cassandra ist durch die verteilte Architektur begründet, die es erlaubt Millionen von Datensätzen zu speichern und zu verwalten, die täglich in sozialen Netzwerken produziert werden. Erst durch die horizontale Skalierung ist die Grundlage

geschaffen, soziale Netzwerke mit unermesslich vielen Nutzern zu erstellen. Zusätzlich arbeitet Cassandra mit dem Konsistenzmodell BASE. Die Konsistenz der Daten ist somit nicht immer sichergestellt, was aber für die Mehrheit der Datensätze unerheblich ist. Der Fokus liegt auf der Verfügbarkeit sowie der Partitionierbarkeit des Systems. Für Anwender muss Cassandra bzw. Facebook vor allem jederzeit erreichbar und seitens sozialer Netzwerke partitionierbar sein, damit das Datenvolumen verarbeitet werden kann. Auf technischer Seite ist Cassandra mittels Java und Ruby + Gems implementiert. Gegenwärtig bietet das Datenbanksystem noch keine eigene GUI, sodass die gesamte Datenverwaltung ausschließlich per UNIX Terminal durchgeführt wird. Die elementare Konfiguration geschieht in der Datei *storage-conf.xml*. Weitere Konfigurationen, wie die Inbetriebnahme mehrerer Rechner, sind in der Datei *cassandra.in.sh* hinterlegt. Unter Cassandra werden keine Abfragen ausgeführt, sondern die Datenmenge wird mittels Map/Reduce reduziert, wodurch die Dezimierung des Gesamtdatenbestands eintritt. Die einfache Skalierbarkeit repräsentiert einen signifikanten Vorteil des Systems. Neue Knoten können in das Cluster mithilfe der Adresse eines beliebigen Knotens einsteigen (seed node). Das Bootstrap-Verfahren verteilt im Anschluss die Last aller Knoten optimal unter Zuhilfenahme eines statistischen Mittels. ([27], S 82 ff.)

Das Datenmodell Cassandra umfasst folgende Hauptkomponenten:

- Keyspace: Bildet die oberste Ebene der Datenstruktur. Theoretisch sind mehrere Keyspaces denkbar, jedoch ist deren Anwendung in der Praxis überaus selten.
- Column Family: Jeder Keyspace besteht lediglich aus einer Column Family. Die nachfolgende Abstraktionsebene bilden die Rows, die in beliebiger Anzahl gespeichert werden können. Rows setzen sich auf der untersten Abstraktionsebene aus Columns sowie den dazugehörigen Schlüsseln zusammen. Alle Daten werden als Bytes unendlicher Größe gespeichert und meist als UTF-8 oder 64-bit Integerwert interpretiert.
- Rows: Besitzen stets einen eindeutigen Schlüssel zur Identifikation von Daten der Columns und Super Columns.
- Super Columns: Setzen sich aus einfachen Columns zusammen mithilfe derer die Darstellung komplexerer Datentypen realisierbar ist.
- Columns: Sind Tupel, die meist einen Namen, einen Wert sowie einen Zeitstempel zum Inhalt haben und die unterste Ebene bilden.

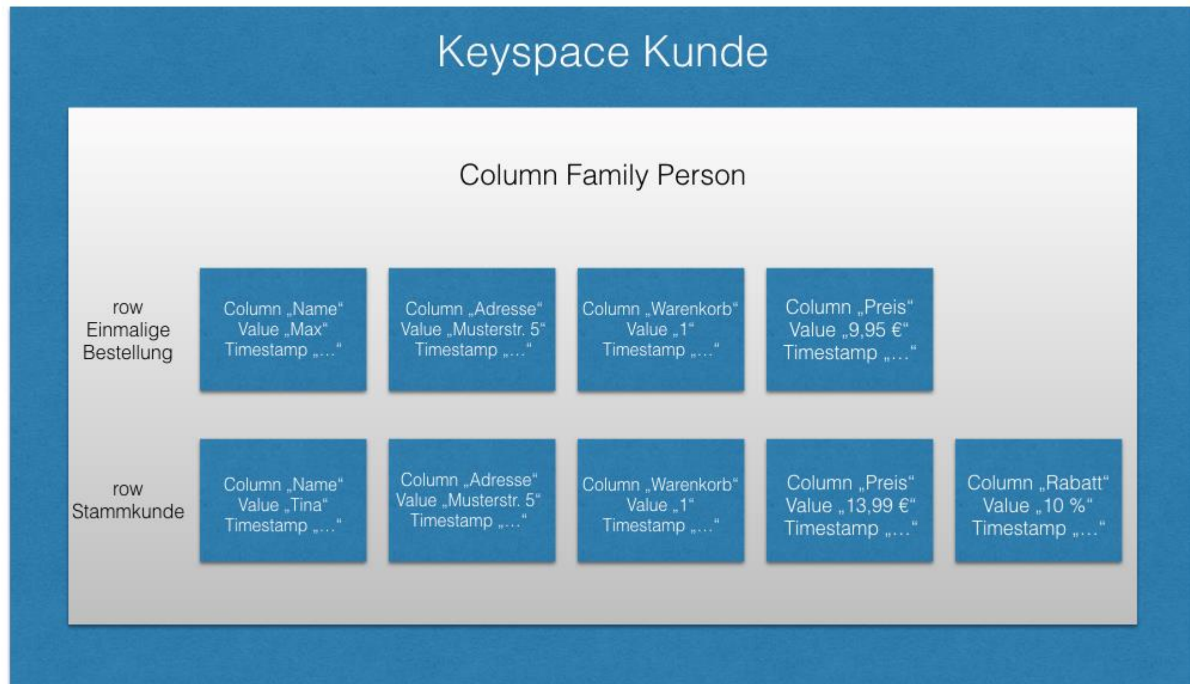


Abbildung 14: Datenmodell „Keyspace Kunde“

3.2.4 Graphdatenbanken

Graphdatenbanken verdanken ihren Durchbruch dem Boom an Smartphones sowie mobilen Computern und dem wachsenden Interesse an komplexen, datenintensiven und verteilten Rechenanwendungen. Konzipiert wurden diese Systeme vor dem graphentheoretischen Hintergrund, dadurch eignen sie sich insbesondere zur Durchführung von Topologieanwendungen auf den verbundenen Datensätzen.

Zur Erfüllung der hohen Qualitätsanforderungen reicht es schon lange nicht mehr einzelne Datensätze zu speichern, sondern die Verknüpfung der Daten untereinander ist ein Gebot, das mithilfe von Graphdatenbanken realisiert wird. Unter der Graphentheorie wird ein Gebiet der Mathematik verstanden, welches die Problematik mittels Knoten und Kanten in einem demonstrativen Prozedere darstellt. Die Ursprünge reichen bis ins 18. Jahrhundert zurück, als der Mathematiker Leonard Euler erstmals anhand der Graphentheorie bewies, dass keine Lösung für das „Königsberger Brückenproblem“ existiert. ([27], S. 8)

3.2.4.1 Einleitung und Funktionsweise

Die Graphen werden mittels Knoten und Kanten erstellt. Mathematisch betrachtet sind Knoten und Kanten als Tupel interpretiert: G = (V,E), wobei V ≠ ∅ eine endliche Menge von Knoten (V) und eine endliche Menge von Kanten (E) darstellt. Prinzipiell wird differenziert zwischen gerichteten Graphen z.B. Mutter-Kind-Beziehung und ungerichteten Graphen beispielsweise Freundschaften. Zusätzlich können Cluster in Graphen reproduziert sowie diverse Cluster miteinander verbunden werden.

Anhand der Clusterbildung werden die Graphen auf verschiedene Rechner distribuiert. Dabei ist es relevant, dass die einzelnen Cluster so wenig Verbindungen wie möglich zum nächsten Cluster aufweisen, damit der Datenaustausch über langsame Verbindungen vorzugsweise gering gehalten wird. Die Graphentheorie integriert verschiedene Arten von Graphen und klassifiziert wie folgt:

- **Einfache Graphen:** Einfache Graphen implizieren Knoten und Kanten. Dabei ist jeder Knoten mit einer Kante versehen, die ungerichtet, also bidirektional, transversierbar ist (Abbildung 15a).
- **Gerichtete Graphen:** Gerichtete Graphen besitzen das Spezifikum, dass die Kanten nur unidirektional traversiert werden können, siehe Abbildung 15b. Im Modell existiert ein Weg von den Knoten <A, B, F>, jedoch nicht in die entgegengesetzte Richtung. Jeder gerichtete Graph hat somit immer einen definierten Anfangs- und Endknoten.
- **Gewichtete Graphen:** Gewichtete Graphen erhalten an ihrer Kante Zahlenwerte, die abhängig vom verwendeten Algorithmus Präferenzen für die Transversierung dieser Kante angeben. Abbildung 15c indiziert, dass der Algorithmus für die Transversierung des Graphen von Knoten C nach Knoten F den Pfad C,A,B,D,F gegenüber dem Pfad C,E,B,D,F favorisiert, da die Summe der Kantenwerte der ersten Applikation 2,6 und der zweiten 2,8 beträgt. Der Algorithmus verwendet immer den Pfad mit der niedrigsten Kantensumme.
- **Multigraphen:** Multigraphen verfügen über zwei Knoten und mehrere Kanten, wodurch unterschiedlichste Beziehungen innerhalb sozialer Netzwerke abgebildet werden. So repräsentiert beispielsweise im sozialen Netzwerk eine Kante „ist befreundet mit“ und eine andere Kante „gefällt der Beitrag der Person“ (Abbildung 15d).

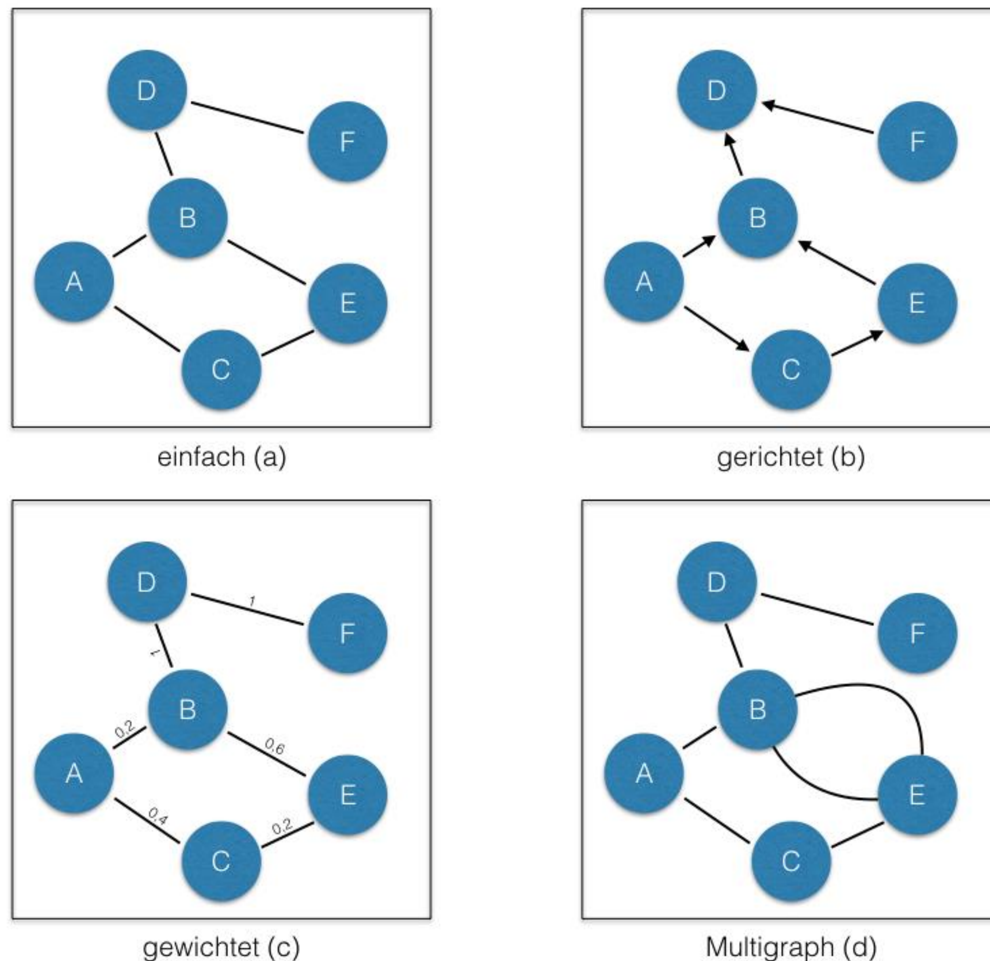


Abbildung 15: Verschiedene Graphmodelle

Zur effizienten Suche innerhalb der Graphdatenbanken existieren verschiedene Suchalgorithmen, die für differente Szenarien notwendig sind.

Die Breitensuche beschreibt einen Suchalgorithmus, der durch die rekursive Breitensuche nach den direkten Nachbarn des Ausgangsknotens erfolgt. Ist ein direkter Nachbar disponibel, bildet dieser den aktuellen Ausgangsknoten und die Suche beginnt aufs Neue. Aufgrund des enormen Leistungsbedarfs wird ein Speicheraufwand von O (|V| + |E|) aufgewiesen, der insbesondere für flache Strukturen mit geringer Tiefe geeignet ist. ([27], S. 209)

Die beschränkte Tiefensuche folgt rekursiv der nächstgelegenen Kante des Ausgangsknotens bis zum Erreichen des Endknotens, wobei das Ergebnis der Tiefensuche bei gleichem Speicherverbrauch O (|V| + |E|) wie bei der Breitensuche nur dann optimal ist, solange nur ein Pfad zum Endknoten existiert. Mehrere mögliche Ergebnisse werden nicht erfasst, wodurch das Resultat bei verschiedenen Lösungsmöglichkeiten eingeschränkt ist. Lediglich die Speicheranforderung bei nur einem möglichen Ergebnis ist maximal.

Zur Erreichung des bestmöglichen Resultats mithilfe der Suchalgorithmen bedarf es der Kombination Breiten- sowie beschränkter Tiefensuche. Die iterative Tiefensuche verbindet beide Suchalgorithmen. Dazu findet die Tiefensuche t=1 Anwendung, die diesen Wert nach jeder Iteration um 1 erhöht. Damit steigert sich die Laufzeit mit jeder neuen Dimension in die Tiefe und weist Analogien zur Breitensuche auf.

Graphdatenbanken zeigen enorme Möglichkeiten, vorwiegend unstrukturierte oder auch semistrukturierte Daten zu speichern. Im Vergleich zu konventionellen Datenbanken sind die Kosten zur Verknüpfung von Daten minimaler, da zum einen unverzüglich nach Datenimport deren Bezug zueinander definiert wird und zum anderen die räumliche Relation aufgrund der grafischen Darstellung gegeben ist. Die räumliche Koordination der Daten bewirkt deutlich schnellere Effekte der Suchalgorithmen. Alles in allem eröffnen Graphdatenbanken Perspektiven, jedoch können sie bisher die konventionellen Systeme nicht ersetzen, da nur verhältnismäßig kleine Datenbanken händelbar sind. Zudem optimieren konventionelle Datenbanken den mehrmaligen Zugriff entgegen den Graphdatenbanken, die nur einen einzigen Zugriff erlauben.

3.2.4.2 Beispielanwendung „Neo4j“

Neo4j gehört zu den renommierten Graphdatenbanksystemen und ist ein gängiges NoSQL-System mit Open Source Lizenz, das Verfahren auf zwei unterschiedliche Weisen praktiziert. Die Embedded-Variante wird innerhalb der Java-Applikation genutzt und vollständig integriert, sodass Neo4j unter derselben Prozess-ID wie die Applikation läuft, aufgrund dessen können sehr hohe Verarbeitungsgeschwindigkeiten erreicht werden. Die Konfiguration als eigenständiger Client-Server-Prozess ist ebenfalls ausführbar. Hierfür ist ein client-seitiger Zugriff über die REST (Representational State Transfer) Schnittstelle notwendig. Zudem wird Neo4j, ebenso wie andere NoSQL-Systeme auch, als zentrales GDBMS oder in der verteilten Konfiguration betrieben. Dabei erfolgt die gesamte Speicherung der Daten dekzentriert, wodurch ein Hauptspeicher zentrierter Betrieb im Rahmen des Möglichen ist.

Neo4j folgt dem Property-Graph-Modell und kann dadurch Knoten gruppieren. Ein Knoten kann verschiedenen Gruppen, einer einzigen Gruppe oder auch keiner Gruppe zugeordnet sein. Die Beziehung einzelner Knoten untereinander erfolgt mithilfe gerichteter Kanten. Dadurch ist immer ein Start- und Endknoten gegeben, der ebenso Schleifen beinhalten kann.

Es werden vier verschiedene Zugriffsmechanismen zur Verfügung gestellt: Core API, Traversal Framework sowie Cypher und Gremlin mit unterschiedlicher Mächtigkeit. Diese können im Embedded Betrieb oder in der Client-Server-Konfiguration betrieben werden. In der Folge wird der Fokus auf die Anfragesprache Cypher gelegt, die häufig als

primäre Anfragesprache bezeichnet wird. Syntaktisch zeigt Cypher viele Analogien zu SQL und SPARQL. Kernfunktion der Sprache ist das Beschreiben von Mustergraphen zur Informationsextraktion.

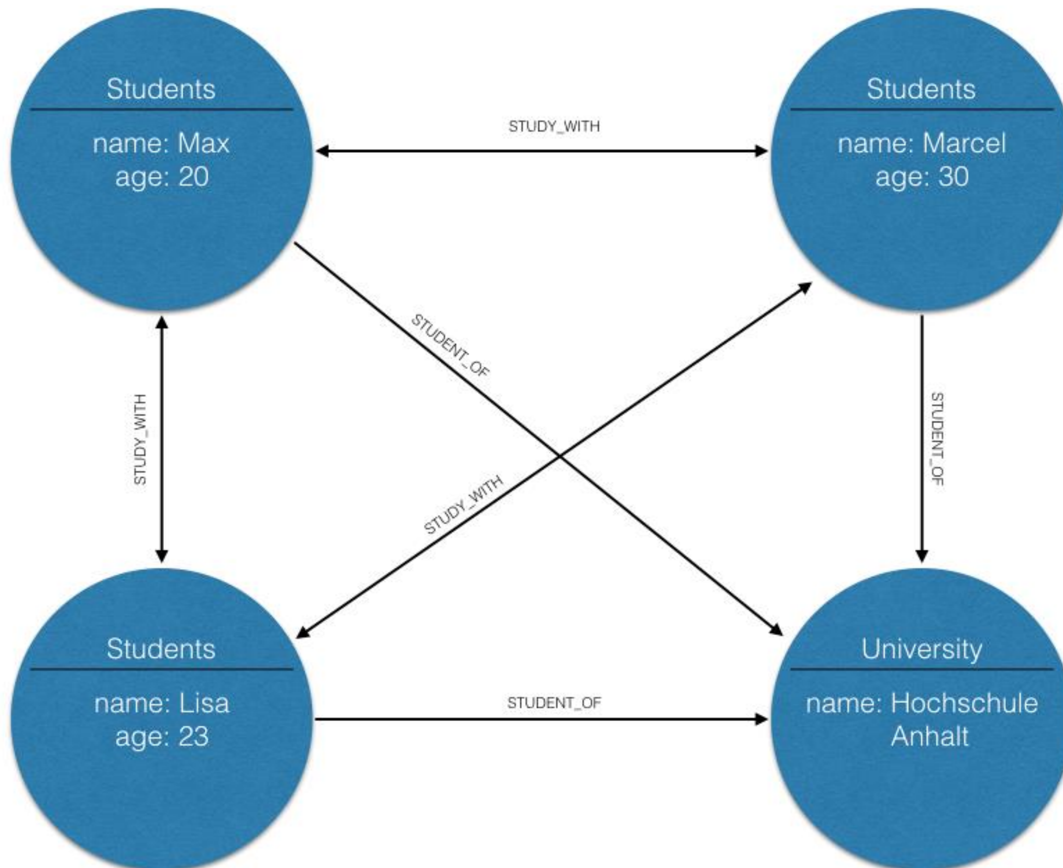


Abbildung 16: Beispielgraph mit Beziehungen zwischen den Studenten der Hochschule Anhalt

Anhand Abbildung 16 wird ein Beispielgraph mit Beziehungen der Studenten untereinander sowie zur Hochschule Anhalt aufgezeigt. Die Knoten und Kanten besitzen hierzu verschiedene Attribute und Label, die diese genauer charakterisieren. Mit Unterstützung von CREATE werden neue Knoten sowie Kanten erzeugt. SET fügt den erzeugten Knoten und Kanten Eigenschaften hinzu, die mit der DELETE-Klausel wieder entfernt werden können.

Mit

CREATE (n: Students {name: “Max“, age 20})

CREATE (n: Students {name: “Marcel”, age 30})

CREATE (n: Students {name: “Lisa”, age 23})

CREATE (n: University {name: “Hochschule Anhalt”})

wird ein neuer Knoten angelegt, der das Label „Students“ enthält und als Eigenschaft, Name und Alter der Studenten speichert. Ebenso wird ein Knoten mit University erzeugt, deren Name „Hochschule Anhalt“ ist. Im Anschluss können Kanten zwischen den einzelnen Knoten mit folgender Syntax errichtet werden:

```
MATCH a: Students, b: Students, c: Students
WHERE a.name =”Max” AND b.name =”Marcel” AND c.name =”Lisa”
CREATE a<- [r:STUDY_WITH] ->b
CREATE b<- [r:STUDY_WITH] ->c
CREATE c<- [r:STUDY_WITH] ->a
CREATE a- [r:STUDENT_OF] ->d
CREATE b- [r:STUDENT_OF] ->d
CREATE c- [r:STUDENT_OF] ->d
Return r
```

Die Match-Funktion ist obligatorisch, womit sichergestellt ist, dass die Knoten im Graphen vorhanden sind. Anhand der ebenfalls erforderlichen WHERE-Klausel wird eine Filterung der Ergebnistupel mittels Prädikaten durchgeführt. Unter Verwendung von CREATE werden Kanten zwischen den verschiedenen Knoten angelegt. Im Beispiel studieren die drei Personen gemeinsam an der Hochschule Anhalt. Die Richtung der Kanten wird mithilfe von <- und -> angegeben.

In Abbildung 16 handelt es sich bei jeder Person um einen Studenten der Hochschule Anhalt, sodass die Kanten unter den Studenten bidirektional angelegt wurden.([27], S. 290 ff.)

3.2.5 Data Stream Systems

3.2.5.1 Data Stream

Datenstrom (engl. Data Stream) ist ein kontinuierlich unendlicher Strom von Daten, der aus einer potenziell unendlichen Folge von Datensätzen besteht. Relevant ist hierbei, dass weder Größe noch Ankunftszeit der eingehenden Datensätze vorhergesagt oder beeinflusst werden kann.

Aufgrund dessen ist eine Speicherung vor der Verarbeitung, wie etwa bei klassischen Datenbanksystemen üblich, nicht realisierbar. Somit ist auch der Zugriff auf Datensätze nach beliebiger Zeit nicht durchführbar, da diese nach Bearbeitung durch das DSMS sofort verworfen werden, um unnötigen Datenmüll zu vermeiden. Eine Speicherung ist in-

folge der Vielzahl ohnehin nicht möglich. Im Normalfall wird lediglich ein geringer Datensatz gespeichert, beispielsweise Durchschnittswerte sowie Start- und Endpunkt, um spätere Anfragen und Analysen zu ermöglichen. Ergo ist eine Analyse und Verarbeitung in Echtzeit zwingend erforderlich. Datenströme von Patienten im Krankenhaus manifestieren, dass jede kleinste Verzögerung der Datenströme im Extremfall über Leben und Tod des Patienten entscheiden.

Datenströme werden generell in drei Hauptkategorien unterteilt: Punktströme, Tupelströme und XML-Ströme. Punkt- und Tupelströme besitzen keine Struktur und liegen nicht in geschachtelter Form vor. Des Weiteren weisen sie immer dieselbe Länge auf. Im Gegensatz dazu stehen XML-Datenströme. Diese sind strukturiert und können beliebig tief verschachtelt sein. Zudem bestehen sie aus unterschiedlicher Länge, wodurch eine Auswertung der Ströme sehr komplex ist. Trotz der Komplexität gilt XML als Standardformat zum Datenaustausch im Internet.([10], S, 1-6)

3.2.5.2 DBMS vs. DSMS

In klassischen Datenbankmanagementsystemen werden alle Daten dauerhaft in Relationen gespeichert. Dabei wird angenommen, dass die Datensammlung korrekt und vollständig ist, sodass spätere grundlegende Eingriffe nicht erforderlich werden. Die Daten werden indiziert und somit für einen späteren Zugriff aufbereitet. Bei dieserart DBMS wird eine Anfrage durch einen aktiven Nutzer gestellt, der die erforderlichen Daten mithilfe einer sogenannten Pull-Kommunikation abruft. Nachdem die Anfrage bearbeitet wurde, erhält der Nutzer ein exaktes Ergebnis, das zum Zeitpunkt der Anfrage einer vollständigen Auswertung des Datenbestands entspricht. Hierfür ist kein Echtzeitsystem erforderlich, da der Datenbestand unverändert bleibt und zu jedem beliebigen Zeitpunkt erneut abgerufen werden kann. Generell ist ein klassisches DBMS als passives System anzusehen, das von Nutzern durch aktive Anfragen verwendet wird (HADP Human-Active, DBMS-Passive). Bei Datenströmen ist lediglich ein einmaliger sequentieller Zugriff auf die Daten möglich, da diese aufgrund des hohen Speicherbedarfs meist nur im Hauptspeicher gehalten werden. Zu diesem Zweck werden Einpass-Algorithmen verwendet, die mit einem einzigen Durchlauf über die Daten auskommen. Bei diesem Modus ist kein aktiver Nutzer erforderlich, der kontinuierlich Anfragen an das System stellt. Alle Anfragen werden vom System selbst indiziert und automatisch an den Nutzer übermittelt. Man spricht daher von einer Push-Kommunikation. Bei sehr hohem Datenaufkommen ist eine vollständige Bearbeitung in Echtzeit nicht immer möglich, sodass Tupel verworfen werden und nicht in die Auswertung einfließen. Die Ergebnisse büßen dadurch an Genauigkeit ein. Für den Nutzer ist die exakte Auswertung der Daten jedoch marginal, da dieser meist nur über „Ausreißer“ oder ungewöhnliche Werte (Bsp. Sensorwerte) informiert

werden möchte. DBMS-Active und Human-Passiv (DAHP) sind dazu autorisiert.([23], S. 46)

	Database Management System (DBMS)	Data Stream Management System (DSMS)
Daten	Persistente Daten	Transiente Daten
Anfragen	Transiente Anfragen	Persistente Anfragen
Anfrageform	Einmalige Anfragen	Kontinuierliche Anfragen
Zeitanforderung	Keine Zeitanforderung	Echtzeitanforderung
Ergebnisse	Exakte Daten	Ungenaue Daten
Indizierung	Datenindizierung	Anfragenindizierung
Aktualisierung der Daten	Sehr niedrige Update-Rate	Sehr hohe Update-Rate
Speicher	(Theoretisch) Unendlicher Sekundärspeicher	Beschränkter Hauptspeicher

Abbildung 17: Unterschiede zwischen DBMS und DSMS[29]

3.2.5.3 Anfragen

Anfragen an Datenbanksysteme bzw. Datenstromsysteme können völlig unterschiedlicher Natur sein.

Einmalige Anfragen implizieren dabei die einfachste Form. Diese werden ein einziges Mal an das System gestellt und dokumentieren eine Momentaufnahme zum Zeitpunkt der Anfragestellung, insbesondere bei DBMS-Systemen üblich.

Kontinuierliche Anfragen beschreiben Anfragen, die automatisch vom System generiert werden, sobald neue Daten einfließen. Die Anfragen betrachten einen Zeitraum x und geben alle Ergebnisse des Datenstroms wider, die kontemporär abgearbeitet werden.

Vordefinierte Anfragen sind Anfragen, die gestellt werden bevor eine Verarbeitung stattgefunden hat. Dabei können diese in Form von einmaligen oder kontinuierlichen Anfragen vorliegen.

Ad-hoc Anfragen können jederzeit an das System bzw. den Datenstrom gestellt werden. Meist sind diese als einfache vordefinierte Anfragen zu verstehen, bei denen das Ergebnis zuerst noch vom Datenstrom bearbeitet wird. Auch Ad-hoc Anfragen an Daten aus der Vergangenheit sind ausführbar, sofern diese gespeichert wurden. In der Regel werden jedoch alle Daten des Datenstroms sofort gelöscht. Es besteht dennoch die Möglichkeit Daten zu speichern, wobei gilt: Desto mehr Daten desto detaillierter die spätere Ad-hoc Anfrage.

Sliding Windows bietet die Alternative, angenäherte Ergebnisse von Datenströmen zu erstellen. Dabei wird ein Ausschnitt des Datenstroms in Form eines Fensters betrachtet, das eine beliebige Größe aufweist. Wird das Fenster über den Datenstrom verschoben, wird ein neuer Ausschnitt sichtbar, der bei gleichgroßem Fenster dieselbe Anzahl an Daten liefert. Ein besonderes Spezifikum von Sliding Windows ist deren Determinismus, wodurch zukünftige Ergebnisse durch Vorbedingungen eindeutig festgelegt werden.([38], S. 1-7)

3.2.5.4 DSMS Beispielanwendung „Stream“

DSMS stellt die Option bereit, Datenströme zu überwachen und diese mit SQL ähnlichen Abfragen zu formulieren. Zur Überwachung der Tupelstromsysteme haben sich die beiden Systeme AURORA der Universitäten Brandeis und Drown sowie das MIT und STREAM der Universität Stanfort herauskristallisiert. Im Folgenden wird STREAM konkretisiert, da dessen CQL (continuous query language) klassischen SQL-Anfragen ähnelt.

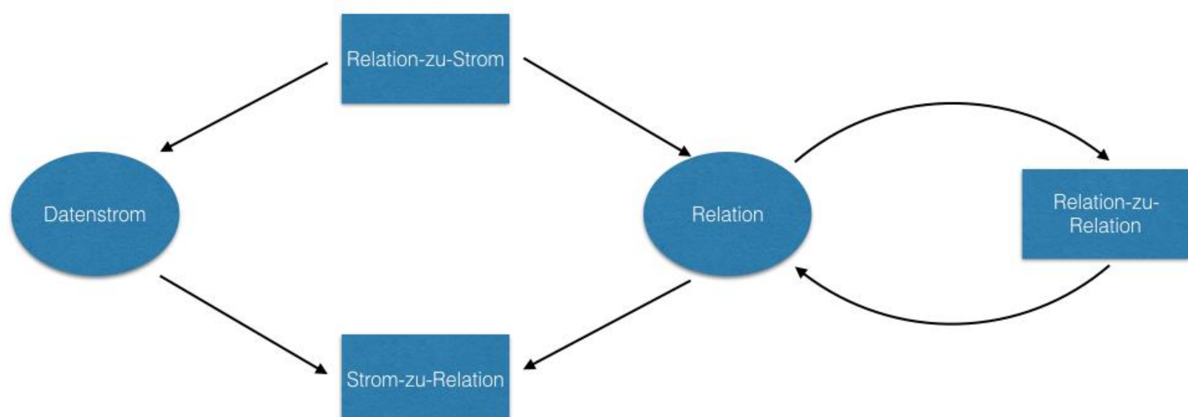


Abbildung 18: Beziehung von Datentypen und Operationen

STREAM steht für **ST**anfort st**RE**m dat**A** **M**anager und impliziert ein DSMS, das zur Bearbeitung kontinuierlicher Anfragen über kontinuierliche Datenströme eingesetzt

wird. Abbildung 18 skizziert die beiden Arten bestehend aus Datenströmen und Relationen. Die Datentypen können in Form von vier unterschiedliche Operationen angewendet werden.

Datenstrom: Der Datenstrom beschreibt eine unendliche kontinuierliche Folge von Wertpaaren, die jeweils aus Datentupeln und einem Zeitstempel bestehen. Anhand des Zeitstempels ist es möglich die Ankunftszeit der Datentupel zu bestimmen und somit deren Reihenfolge.

Relation: Die Relation beschreibt die Abhängigkeit der verschiedenen Datentupel zueinander. Dabei ist die Determiniertheit umso größer, desto geringer deren zeitliche Differenz ist.

Relation-zu-Relation-Operator: Der Relation-zu-Relation-Operator nimmt eine oder mehrere Relationen als Eingangsparameter an und erstellt mittels dessen eine Relation als Ausgangsparameter. Die Operationen entsprechen exakt den Operationen von DBMS.

Strom-zu-Relation-Operator: Der Strom-zu-Relation-Operator nimmt einen oder mehrere Datenströme als Eingangsparameter an und generiert daraus eine Relation als Ausgangsparameter. Die verschiedenen Datenströme können mit drei unterschiedlichen Typen von gleitenden Fenstern (sliding windows) bearbeitet werden. Zeitbasierte Fenster definieren sich über den Zeitparameter T und legen fest, welche Wertpaare im Zeitraum T betrachtet werden. Des Weiteren besteht die Alternative die Sonderfälle „Now“ und „Unbounded“ zu definieren. „Unbounded“ bezieht sich dabei auf alle Wertpaare, die einen gültigen Zeitstempel aufweisen und bis zum momentanen Zeitpunkt eingetroffen sind. „Now“ legt fest, dass nur die Wertpaare verwendet werden, die einen aktuell gültigen Zeitstempel besitzen.([38], S. 4)

Die folgenden Beispiele zeigen mögliche Definitionen:

- PKWGeschwindigkeit [10 Minutes] gibt die Geschwindigkeit aller PKWs an, die in den letzten 10 Minuten registriert wurden.
- PKWGeschwindigkeit [Now] definiert ein Fenster, bei dem alle Geschwindigkeiten der PKWs registriert werden, die aktuell gemessen werden.
- PKWGeschwindigkeit [Range Unbounded] definiert ein Fenster aller je erfassten Geschwindigkeiten.

Anzahlbasierte Fenster werden über den Parameter N definiert, der die Anzahl der Wertpaare zum jeweiligen Fenster angibt. N kann auch mit einem konkreten Wert definiert werden oder den Sonderfall „Unbounded“ aufweisen. Beim Parameter N handelt es sich stets um einen positiven Wert.

Die folgenden Beispiele zeigen mögliche Definitionen von N:

- PKWGeschwindigkeit [Rows 100] definiert ein Fenster, dessen Größe die letzten 100 Zeilen des Datenstroms enthält.
- PKWGeschwindigkeit [Rows Unbounded] definiert ein Fenster, das alle bis dahin aufgetretenen Geschwindigkeiten enthält.

Relation-zu-Strom-Operator: Der Relation-zu-Strom-Operator nimmt eine oder mehrere Relationen als Eingangsparameter an und generiert zu diesem Vorgang einen Strom von Ausgangsparametern. Die Anfragesprache CQL verwendet folgende Operationen: Istream erzeugt einen *input stream,* bei dem alle Wertpaare zu einer Relation hinzugefügt werden (R(t) \ R(t-1)). Beim Hinzufügen der Wertpaare ist der Zeitstempel des jeweiligen Paares für die Reihenfolge verantwortlich.

Istream

```
SELECT Istream (*)
From PKWGeschwindigkeit [Range Unbounded]
Where km/h > 50
```

(Gibt alle Werte zurück, die als Prädikat einen Wert größer 50 besitzen)

Dstream erzeugt einen *delete stream,* bei dem alle Wertpaare aus einer Relation des alten Zeitraums entfernt werden (R(t-1) \ R(t)).

Dstream

```
SELECT Dstream (*)
From PKWGeschwindigkeit [Range 10 Minutes]
```

(Es werden alle Elemente mit aktuell gültigem Zeitstempel gelöscht und ausgegeben).

Rstream erzeugt einen *relation stream,* der zu einem Zeitpunkt alle in einer Relation befindlichen Wertpaare enthält (R(t)).

Rstream

```
SELECT Rstream (*)
From PKWGeschwindigkeit [Now]
Where km/h > 50
```

(Es wird nur das Element mit einem aktuell (Now) gültigen Zeitstempel ausgegeben

3.2.6 Kritik an NoSQL-Datenbanken

NoSQL-Systeme entsprechen den Herausforderungen der heutigen Zeit und bieten vielfältige Möglichkeiten, Big Data auf komfortable Art und Weise zu lösen. Dennoch weisen sie neben Defiziten auch positive Entwicklungen auf, wie die nachfolgende Analyse dokumentiert.[22]

3.2.6.1 Unzureichender Support

Der zu geringe bzw. nicht ausreichende Support gegenüber relationalen Systemen charakterisiert einen entscheidenden Nachteil. Mit dem Open Source Projekt NoSQL fehlt es generell an Unternehmen, die sich den Supportanfragen annehmen und diese bearbeiten. Die Marktlücke im Bereich NoSQL, kommerziellen Support für Open Source Software anzubieten, wird sukzessiv durch Unternehmen wie Datastax[1], Cloudera[2], Teradata[3], Pivotal[4] und Mongo DB Inc. geschlossen. Den Unternehmen fällt es einerseits schwer, sich dem ständigen Wandel neuer Ideen von Open Source Entwicklern anzupassen und zum anderen können sie keinen vergleichbar umfangreichen Support wie die Big-Player Oracle, Microsoft oder IBM offerieren[22].

[1] http://www.datastax.com
[2] http://www.cloudera.com
[3] http://www.teradata.com
[4] http://www.pivotal.com

3.2.6.2 Geringe Standardisierung

Relationale Datenbanken agieren nach festen Strukturen, die weitestgehend dieselbe Syntax aufweisen und somit grundsätzlich immer nach der standardisierten Anfragesprache SQL verfahren. Zudem werden einfache Transfers zwischen den Systemen ermöglicht, da die Strukturen von Oracle-Datenbanken ebenso bei Microsoft anwendbar sind. Im Gegensatz zu NoSQL-Datenbanken wird signifikant, dass es sich hier um eine Sammlung von sehr heterogenen Systemen handelt. Die bekannten Vertreter wie Key/Value, Wide Column Stores oder Graphdatenbanken unterscheiden sich erheblich in ihrer Funktionsweise, wodurch ein Wechsel zwischen einzelnen NoSQL-Systemen aufgrund fehlender Standardisierung deutlich diffiziler ist als bei relationalen Datenbanken.[22]

3.2.6.3 Fehlende Nachhaltigkeit

NoSQL-Systeme erfahren in den 2000er Jahren einen unglaublichen Hype, da es zunächst so scheint als sei NoSQL die Lösung aller Probleme des Web 2.0. Durch die Open Source Lizenz entstehen zahlreiche NoSQL-Systeme, die meist nur für einen speziellen Anwendungsfall geeignet sind. Diese Systeme erweisen sich jedoch als kurzlebig. Einhergehend mit dem Support stellt die Nachhaltigkeit ein wichtiges Kriterium dar. Aufgrund der Open Source Lizenz zeichnet sich kein Big-Player für das System verantwortlich, das auf Erfolg und Fortbestand angewiesen ist und somit eine perfekte portabilität auf andere Systeme in Zukunft garantiert.[22]

3.2.6.4 Mangelnde Entwicklungsreife

NoSQL leidet seit Jahren unter dem Klischee immer noch nicht den Prototypstatus verlassen zu haben. Dies ist für die meist kleineren Systeme auch zutreffend, da sie sich einer nur geringen Beliebtheit erfreuen und Entwicklungen nicht vorangetrieben wurden. Die etablierten NoSQL-Systeme haben sich deutlich über den Prototypstatus hinaus optimiert und bieten leistungsstarke Systeme mit geringen Ausfällen. Natürlich ist das Argument der relationalen Datenbanken nicht von der Hand zu weisen, dass diese schon über Jahrzehnte existieren und stetig weiterentwickelt werden. Oracle schrieb zu dieser Thematik ein Whitepaper, in dem es heißt:[22]

„Oracle has been developing database-centric software for thirty years. Building any database is hard, to build one as good as Oracle´s takes a long, long time. […] Go for the

tried and true path. Don´t be risking your data on NoSQL databases". – ([7] Alan Downing, 2011)

3.2.7 Zusammenfassung

Der erste Teil dieser Arbeit beschreibt diverse NoSQL-Systeme anhand von Modellen und präsentiert ein erstes Resümee über Stärken und Schwächen einzelner Datenbanken. Dabei werden nur die signifikanten Merkmale einzelner Systeme aufgegriffen, da weitere Ausführungen den Rahmen dieser Masterarbeit übersteigen würden. Angesichts anhaltender Diskussionen über NoSQL-Datenbanken im Kontext zu Big Data Anwendungen stehen vor allem der Umgang mit hoher Datenvielfalt, die Verarbeitungsgeschwindigkeit von Anfragen sowie die Skalierbarkeit der Datenbanksysteme im Fokus.

Insbesondere Key/Value-Stores zeichnen sich durch ihre sehr hohe Verarbeitungsgeschwindigkeit aus und sind dabei frei von Schemarestriktionen. Des Weiteren sind sie einfach zu handhaben und können innerhalb kürzester Zeit angewendet werden. Bedingt durch die minimalistische Ausrichtung und die geringe Anfragemächtigkeit sind sie jedoch nur teilweise für Big Data Anwendungen geeignet. Der Performancevorteil beim Lesen und Schreiben der Daten wird durch das Arbeiten mithilfe des Sekundärspeichers eingebüßt, wodurch die Priorität gegenüber anderen NoSQL-Systemen deutlich dezimiert wird.

Document Stores kommen speziell für die Verwaltung von persistent heterogenen Datensätzen infrage. Ebenfalls besteht die Möglichkeit homogene Datensätze zu verarbeiten, indessen nimmt die Performance des Systems darunter Schaden. Die Softwarelösungen MongoDB und CouchDB sind problemlos zu skalieren, jedoch nicht immer ohne Add-Ons von Drittanbietern.

Column Family Stores sind zugeschnitten auf große Datensätze und reduzieren die Datenmenge nach dem Map/Reduce-Verfahren infolge von Anfragen. Neben der Verarbeitung von gigantischen Datenmengen ist vor allem auch die performante Verarbeitung von persistenten Schreiboperationen eine der großen Vorzüge. Der benötigte Aufwand zum Skalieren eines Column Family Stores ist vom jeweiligen Datenbanksystem selbst abhängig.

Graphdatenbanken sind keineswegs mehr ein Randgebiet der Forschung, sondern haben sich zu einem Datenmodell herauskristallisiert, das besonders durch die grafische Aufarbeitung einfach zu praktizieren ist. Informationen können mithilfe des Property-Graphen auf agile Art und Weise verwaltet werden. Alles in allem sind Graphdatenbanken weitestgehend auf einmalige Abfragen ausgelegt und nicht auf die kontinuierliche Verarbeitung von wohldefinierten Transaktionen.

Data Stream Systems eignen sich in hohem Maße für kontinuierliche Datenströme, die beispielsweise von Sensoren oder Messstationen erfasst werden. Dieses Fachgebiet erfreut sich in den letzten Jahren einer immer größeren Beliebtheit im Kontext von Big Data Anwendungen. Bei der Verarbeitung von konstanten Datenströmen ist es meist überflüssig alle erzeugten Daten zu erfassen und zu verarbeiten, sondern Messfehler werden in Form von ungenauen oder fehlenden Werten beziehungsweise ganzen Ausfällen von Sensoren protokolliert.

4 Relationale Datenbanken

Ted Codd, britischer Mathematiker und Datenbanktheoretiker von IBM Research, entwickelte im Jahr 1970 das relationale Datenbankmodell. Aufgrund der mathematischen Simplizität verbunden mit mathematischen Relationen fand das Modell schnell die Aufmerksamkeit der Branche. Dank Innovationen sowie dem kommerziellen Vertieb mittels Oracle, MySQL, Microsoft SQL Server, PostgreSQL und DB2 gehören relationale Datenbanken bis heute zu den meist verbreiteten Systemen.

Im Zitat von F.D. Rolland heißt es dazu:

„Relationale Datenbanken dominieren zur Zeit den Markt. Sie stellen bemerkenswert einfache Mittel zur Darstellung und Manipulation von Daten zur Verfügung" ([31]Rolland, 2003, S. 51).

Anknüpfend daran werden relationale Datenbanken vor dem Hintergrund der Big Data Anwendungen genauer betrachtet. Vor allem das Datenmodell, die Anfragesprache SQL sowie die technischen Eigenschaften werden diskutiert. Ein besonderer Fokus liegt zudem auf MySQL, ein weit verbreitetes relationales Datenbanksystem, das aufgrund seiner GPL (General Public License) frei zugänglich ist.

4.1 Aufbau einer relationalen Datenbank

Das folgende Kapitel präzisiert den Aufbau einer relationalen Datenbank und definiert grundlegende Begrifflichkeiten.

Das Speichern und Verwalten der Daten erfolgt in einer einfachen zweidimensionalen Tabelle, auch Relation genannt. Dabei enthält eine Relation immer den Namen sowie die Bezeichnung der jeweiligen Spalte. Spalten werden ebenso als Attribute beschrieben und besitzen einen Wertebereich. Die Einträge einer Relation ergeben sich aus deren Zeilen. Diese sind geordnete Listen, die auch als Tupel bezeichnet werden. Abbildung 19 zeigt die Relation *Studenten*, die im Beispiel aus den vier Attributen: Mat. Nr., Name, Vorname sowie deren Ausgaben besteht. Hierbei charakterisiert die Zeile: Mat. Nr.: 5132, Name: Müller, Vorname: Jngo sowie die Ausgaben in Höhe von 120€ die Tupel.

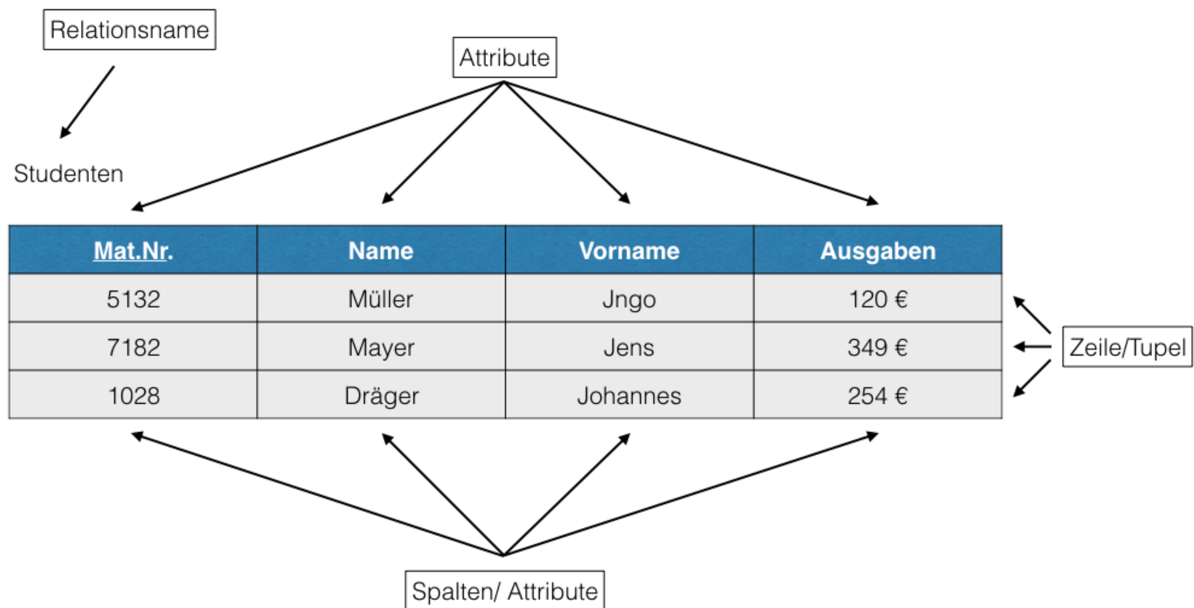


Abbildung 19: Aufbau einer relationalen Datenbank

Es werden drei unterschiedliche Schlüssel zur Identifikation des Datensatzes verwendet. Der Kardinalschlüssel besteht aus einem oder mehreren Attributen und beschreibt die Einzigartigkeit des Tupels. Existieren in einem Tupel mehrere Kardinalschlüssel, wird einer als Primärschlüssel ausgezeichnet. In einer Relation ist immer nur ein Primärschlüssel definiert, um eine eindeutige Identifikation sicherzustellen.([15], S. 57) In der Abbildung oben wird der Primärschlüssel durch die Mat. Nr. des Studenten repräsentiert. Die Entitätsintegrität legt zudem fest, dass es in einer Relation keinesfalls mehrere Tupel mit demselben Primärschlüssel gibt. Des Weiteren darf der Wert des Primärschlüsselattributs niemals NULL annehmen.([31], S. 58)

Fremdschlüssel beschreiben Attribute, die in mehr als nur einer Tabelle vorkommen und dabei den Primärschlüssel darstellen. Außerdem dienen sie dazu, die Daten verschiedener Tabellen in Beziehung zueinander zu setzen und zu verknüpfen. Die Regel der referentiellen Integrität besagt, dass jeder Wert in einem Fremdschlüsselattribut ebenfalls in der Relation vorkommen muss, in der dieses Attribut als Primärschlüssel auftritt.([15], S. 57 ff.) Fremdschlüsselbeziehungen können jedoch auch Probleme mit sich bringen, wenn beispielsweise ein Tupel aus einer Relation gelöscht wird, auf die sich der Fremdschlüssel bezieht. Folgende Lösungen sind denkbar: Zum einen sollte der referenzierte Schlüssel nicht geändert oder gar gelöscht werden und zum anderen besteht die Option, auch Änderungen unverzüglich auf die abhängigen Fremdschlüssel zu übertragen oder aber auf NULL zu setzen.([21], S 43)

4.2 Entity-Relationship-Modell

Das Entity-Relationship-Modell (ER-Modell) ist eine gängige Notationsform, die als Entwurf zur Planung einer Datenbankstruktur von Peter Chen 1976 entwickelt wurde. ER-Modelle transformieren hierzu die Zusammenhänge der realen Welt auf eine abstrakte Ebene und legen dar, wie inhaltliche Bezüge in den Tabellen der Datenbank zu modellieren sind. Darüber hinaus werden die Anforderungen an die spätere Datenbank seitens des Auftraggebers akribisch kommuniziert und analysiert. Das ER-Modell besteht im Wesentlichen aus Entitäten und Beziehungen. Die Entität beschreibt dabei ein Objekt aus der Realität (z.B. einen Gegenstand, eine Person oder ein Ereignis), das als Tupel bezeichnet wird. Alle Entitäten in einem ER-Modell werden durch Rechtecke kenntlich gemacht. Die Beziehung der Entitäten zueinander wird durch verschiedene Beziehungstypen verdeutlicht, auch Konnektivität genannt. Im Modell werden Relationen in Form von Rauten dargestellt. Sowohl Entitäten als auch Beziehungen verfügen über eine unendliche Anzahl von Attributen, die anhand von Ovalen symbolisiert werden.([30], S. 57)

Abbildung 20 umreißt ausschnittsweise das Entity-Relationship-Modell der Hochschule Anhalt. Hierfür werden drei Entitäten: *Dozent*, *Kurs* und *Student* angelegt, deren Attribute näher beschrieben werden. Die entsprechenden Primärschlüsselbeziehungen sind durch das Unterstreichen des jeweiligen Attributs kenntlich gemacht. Die Beziehung zwischen den einzelnen Entitäten wird durch die Beziehungstypen *hält Vorlesung* und *ist Teilnehmer* dargestellt. Dabei wird jede Beziehung mithilfe von Kardinalitäten exakt spezifiziert. Es erfolgt eine Differenzierung zwischen folgenden drei Arten:

1:1 Beziehung: Hierbei wird einer Entität E_1 maximal eine Entität E_2 zugeordnet. Ebenso gilt das E_2 maximal eine Beziehung zu E_1 besitzt. Es existiert eine 1:1 Beziehung zwischen Dozent und Data Mining, da ein Dozent höchstens eine Vorlesung zugleich halten kann und der Kurs Data Mining ebenso nur von einem Dozenten unterrichtet wird.

1:N Beziehung: Besteht zwischen dem Kurs Data Mining und den Studenten. Dabei werden einer Entität E_1 beliebig viele Entitäten E_2 zugeordnet. Gleichzeitig weist die Entität E_2 zur Entität E_1 im Höchstfall eine Beziehung auf. Dadurch kann eine beliebige Anzahl von Studenten den Kurs Data Mining besuchen, siehe Abbildung 20.

M:N Beziehung: Ordnet einer Entität E_1 beliebig viele Entitäten E_2 zu, überdies kann Entität E_2 ebenso willkürlich viele Beziehungen zu E_1 besitzen. Eine derartige Beziehung kann beispielsweise zwischen Studenten und deren Adresse bestehen, da viele Studenten aufgrund des Studiums einen Zweit-oder Drittwohnsitz haben. Zusätzlich werden starke und schwache Entitätstypen unterschieden. Die starken Entitätstypen besitzen einen Primärschlüssel und sind von den übrigen Entitäten losgelöst. Schwache Entitäten sind dagegen auf eine starke Entität mit Primärschlüssel angewiesen, da sie selbst keinen besitzen. Beispielhaft kann hierzu die Beziehung zwischen Eltern und ihren Kindern angeführt werden. Da Kinder meist über die Eltern mitversichert sind, besitzen sie keine eigene

Versicherungsnummer. Zur Identifikation des Kindes ist es notwendig, dass eine M:N Beziehung zwischen Eltern und Kind existiert und die Entität Eltern eine Versicherungsnummer besitzt, die gleichzeitig den Primärschlüssel bildet.([30], S. 69)

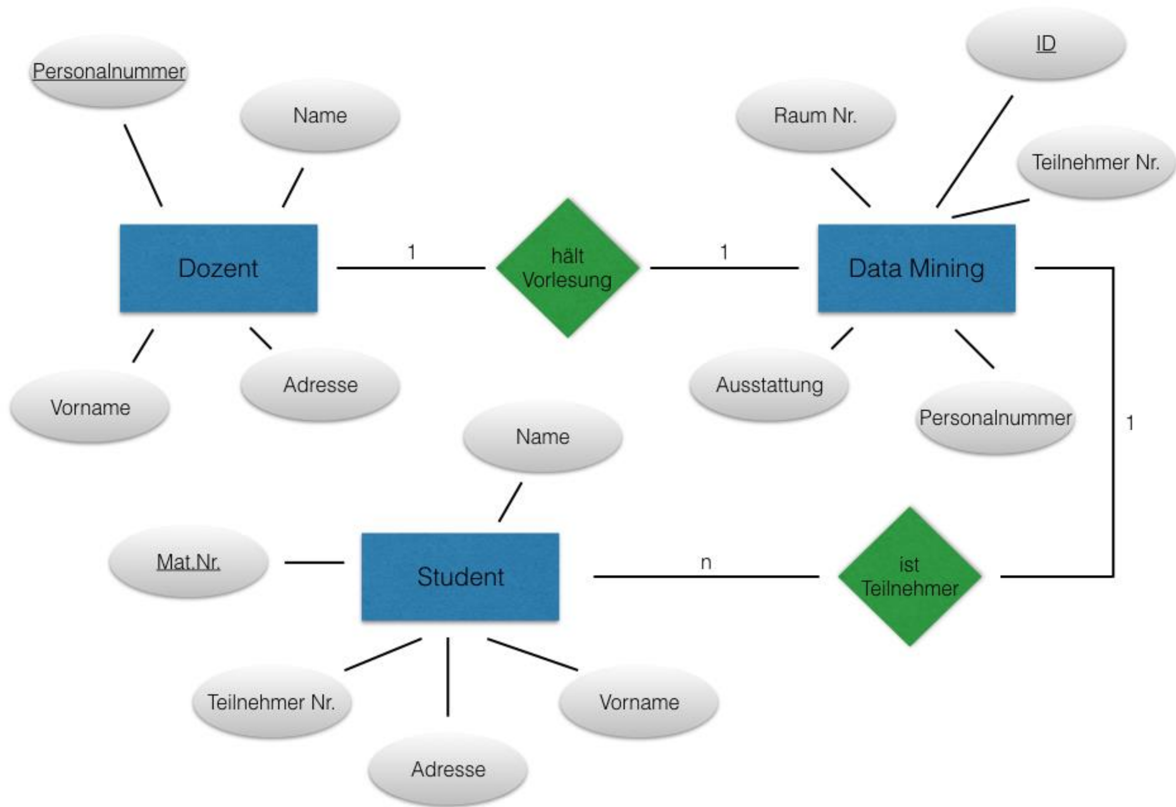


Abbildung 20: Entity-Relationship-Modell

4.3 Anfragenverarbeitung

Die relationale Algebra bietet eine Vielzahl von unterschiedlichen Anfrage- sowie Änderungsoperationen. Anfrageoperationen stellen das Fundament der relationalen Datenbank dar mit deren Hilfe Anfragen gestellt und Informationen selektiert werden können. Die verwendeten relationalen Operationen können beliebig tief geschachtelt werden. Änderungsoperationen dienen dazu, den existierenden Datenbestand zu modifizieren. Zu diesem Zweck werden neue Tupel in den Datenbestand eingefügt (INSERT), aktualisiert (UPDATE) oder gelöscht (DELETE). Konträr zu den Anfrageoperationen können Änderungsoperationen nur auf die jeweilige Relation angewendet werden.([30], S. 299)

Die relationale Algebra verfügt über sechs Basisoperationen, die zum einen aus Operationen der Mengenlehre (kartesisches Produkt, Vereinigung und Differenz) und zum anderen aus speziellen Operationen (Restriktion und Projektion) bestehen. Alle Basisoperationen sind dabei einzigartig und können nicht durch andere Operationen simuliert werden.

Kartesisches Produkt (Kreuzprodukt)

Es existieren zwei Relationen R_1 und R_2 mit den Attributen A_1 , … , A_n sowie B_1 , … , B_m. Die Tupel werden aus den Attributen folgendermaßen dargestellt: (a_{i1} , … , a_{in}) und (b_{j1} , … , b_{jm}). Daraus ergibt sich für das kartesische Produkt:

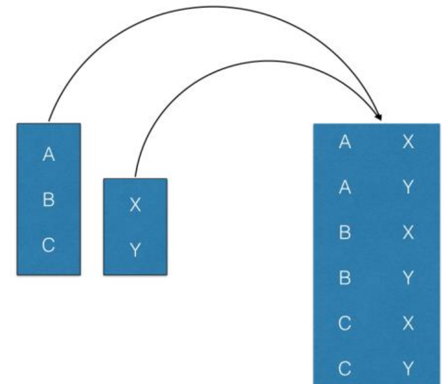


Abbildung 21: Kartesisches Produkt (Kreuzprodukt)

$$R_1 \times R_2 = \left(a_{i1}, \dots, a_{in}, b_{j1}, \dots, b_{jm}\right) \quad (a_{i1}, \dots, a_{in}) \in R_1 \; und \; \left(b_{j1}, \dots, b_{jm}\right) \in R_2$$

für alle Tupel

$$(1 \leq i \leq |R_1| , 1 \leq j \leq |R_2|)$$

Union (Vereinigung)

Zwei Relationen R_1 und R_2 werden mithilfe des Union Operators zusammengefügt. Der Operator ist hierbei kommutativ anwendbar, d.h. die Abfragereihenfolge der Eingangsparameter ist irrelevant.

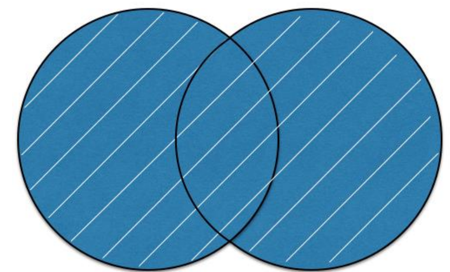


Abbildung 22: Union (Vereinigung)

$$R_1 \cup R_2 = \{t | t \in R_1 \; oder \; t \in R_2\}$$

Differenz

Für die Relationen R_1 und R_2, deren Eingangsmengen vereinigungskonform sind, wird die Differenz der beiden Mengen gebildet.

Dabei gilt:

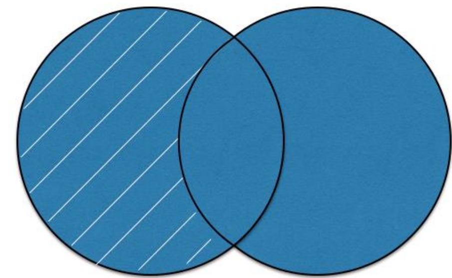


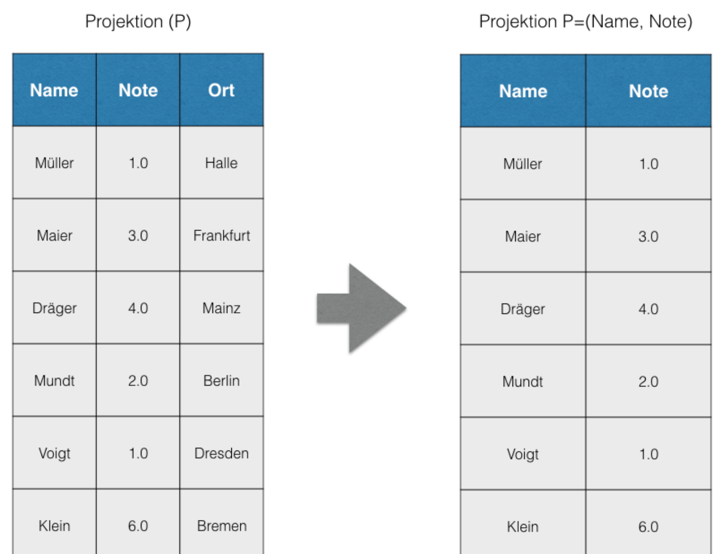
Abbildung 23: Differenz

$$(R_1 - R_2) = \{(a_1, \dots, a_n) | (a_1, \dots, a_n) \in R_1 \; und \; (a_1, \dots, a_n) \notin R_2\}$$

Projektion

Die Projektion ist eine weitere Grundoperation zur Selektion einzelner Spalten innerhalb der Relation. Alle doppelten Tupel werden dabei unterdrückt, sodass bei zwei gleichen Attributen nur ein Tupel an die Ergebnismenge zurückgeliefert wird. Unter Zuhilfenahme des Operatorsymbols π wird die Projektion dargestellt.

Abbildung 24 zeigt die Auflistung von Attributen der Relation R_1:



Projektion (P)

Name	Note	Ort
Müller	1.0	Halle
Maier	3.0	Frankfurt
Dräger	4.0	Mainz
Mundt	2.0	Berlin
Voigt	1.0	Dresden
Klein	6.0	Bremen

Projektion P=(Name, Note)

Name	Note
Müller	1.0
Maier	3.0
Dräger	4.0
Mundt	2.0
Voigt	1.0
Klein	6.0

Abbildung 24: Projektion

$$R_2 = \pi < a_1, \dots, a_n > (R_1)$$

Restriktion (Selektion)

Die Restriktion bildet das Pendant zur Projektion. Dementsprechend werden hierbei nicht die Spalten selektiert, sondern die Zeilen. Das Ergebnis stellt die neue Relation dar.

Die Restriktion wird durch σ symbolisiert und benötigt zusätzlich eine Selektionsbedingung. Diese wird durch den boolschen Ausdruck formuliert, der aus den arithmetischen Vergleichsoperatoren sowie den logischen Operatoren besteht.

Demzufolge gilt:

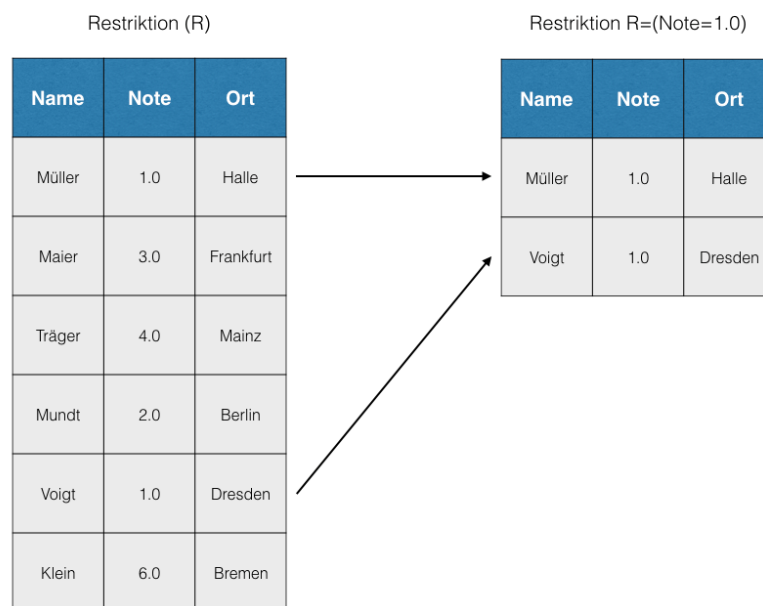


Abbildung 25: Restriktion

$$R_2 = \sigma < Selektionsbedingung > (R_1)$$

Umbenennen

Eine weitere wichtige Grundoperation beinhaltet das Umbenennen. Dies ist unverzichtbar, um unter den Attributen eine Namensgleichheit für die Mengenoperationen (Vereinigung, Differenz, Durchschnitt usw.) herzustellen. Des Weiteren ist eine Namensgleichheit oder auch Namensungleichheit für die Divisionsoperatoren zu generieren.

Damit gilt:

$$R_2(\, b_1 \,, \dots, b_n) \rightarrow R_1(a_1 \,, \dots, a_n)$$

4.3.1 Anfrageverarbeitung mit SQL

Structured Query Language (SQL) ist eine fundamentale Komponente der relationalen Datenbanken. Mithilfe dieser „Standardsprache“ ist die Kommunikation mit der Datenbank überhaupt erst möglich. Das DBMS erhält SQL-Anweisungen, die dazu führen bestimmte Informationen in einer Datenbank zu suchen oder diese anhand einer definierten Struktur zusammenzustellen und aufzubereiten. Dies geschieht unmittelbar, im Gegensatz zu NoSQL-Systemen, deren Einträge zunächst gelöscht und erneut angelegt werden. SQL-Befehle können neben den klassischen Informationsabfragen ebenso Daten des Datenbestands verändern, einfügen oder löschen mittels der sogenannten Data Manipu-

lation Language (DML). Ergänzend zur DML besteht die Option mithilfe der Data Definition Language (DDL) das Schema der Datenbank zu definieren und zu verändern. Die Verwaltung der Nutzerrechte geschieht über die Data Control Language (DCL). Aufgrund der enormen Mächtigkeit sowie der Befehlsvielfalt wird in dieser Masterarbeit nur das Grundkonzept von SQL-Anfragen konkretisiert, ohne Sonderfälle oder komplexe Abfragemodelle einzubeziehen.

Bei SQL handelt es sich um eine deskriptive Sprache, die im Gegensatz zu prozeduralen Sprachen keinen Lösungsweg vorgibt, sondern sich lediglich auf das erwartete Ergebnis konzentriert. Die Grundstruktur einer SQL-Anweisung setzt sich stets aus den drei Klauseln SELECT, FROM, WHERE zusammen und impliziert folgendes Format:

```
SELECT   <Attributliste>
FROM     <Tabellenliste>
WHERE    <Bedingung>
```

Die SELECT-Anweisung greift dabei auf eine Liste mit Attributnamen zu, deren Werte mit der Ausführung der Anfrage erzeugt werden und erlangt dabei Zugang auf einen einzigen oder eine Vielzahl von Attributnamen. FROM wird immer auf einer Tabellenliste zum Einsatz gebracht, die eine Namensliste der Relation enthält. Die WHERE-Bedingung wird mithilfe eines boolschen Ausdrucks gebildet, der die mit der Anfrage zu bearbeitenden Tupel qualifiziert.([30], S. 278)

Generell kann eine SQL-Anweisung aus bis zu sechs Klauseln bestehen. Es sind jedoch lediglich die ersten beiden: SELECT und FROM zwingend erforderlich. Mit weiteren obligatorischen Klauseln wie GROUP BY (spezifiziert Gruppenattribute), HAVING (Bedingung für auszuwählende Gruppen) und ORDER BY (Ordnung des Resultats) können Anfragen verfeinert werden. Die Auswertung der SQL-Anfrage erfolgt konzeptuell, respektive wird zunächst die FROM-Klausel ausgewertet, um die betroffenen Tabellen zu identifizieren. Im Anschluss folgt die WHERE-Klausel, die die Bedingung enthält. Nach der Auswertung der WHERE-Klausel und die Bedingung den Wert true (wahr) zurückgibt, werden die Werte der SELECT-Klausel spezifizierenden Attribute dieser Tupel-Kombination in das Resultat der Abfrage aufgenommen.

SQL-Anfragen bieten zahlreiche Formulierungsmöglichkeiten für die gleiche Anfrage, sodass der Nutzer die adäquateste Lösung anwenden kann. Daraus resultieren auch einige Nachteile. Aufgrund der Vielfältigkeit kann es geschehen, dass Nutzer keine optimalen Anfragen stellen oder deren Anfragen ineffizient sind.([30], S. 190)

4.4 Speicherstrukturen von relationalen Datenbanken

Die Speicherstrukturen von relationalen Datenbanken erfolgen standardmäßig auf Magnetplatten (Hard Disk Drive – HDD), das geläufigste Speichermedium bis in die frühen 2000er Jahre, zudem extrem kostengünstig. Die aktuellen Solid State Disks (SSD) gewinnen zunehmend an Einfluss, da diese aufgrund ihrer Geschwindigkeit und Performance den HDD Platten deutlich überlegen sind. Jedoch finden sie wegen ihrer geringen Speicherkapazität und hoher Kosten nur äußerst selten Anwendung. Die maßgeblichsten und renommiertesten Vertreter von Speicherstrukturen: Heap, Hash-Funktion, ISAM und B*-Baum werden anschließend näher ausgeführt und demonstrieren die physische Verteilung einzelner Datensätze auf einzelne Blöcke (4096 Bytes).([30], S. 389)

Heap

Die Heap Struktur (Haufen) stellt die einfachste Form der Datenspeicherung dar, die auch bei den Programmiersprachen C, C++, C# usw. zum Einsatz kommt. Dabei werden die Daten in Blöcken nach ihrem Erscheinungsdatum auf dem Sekundärspeicher abgelegt und als lineare Liste miteinander verkettet. Diese Speicherstruktur ist besonders effizient bei der Ablegung neuer Daten, da diese auf den obersten freien Block geschrieben werden. Diese Effizienz geht jedoch beim Aufruf von Datensätzen aus dem Heap verloren, da die Speicherblöcke anhand der linear verketteten Liste Block für Block abgesucht werden. Daraus ergibt sich, dass bei n Blöcken somit im Durchschnitt immer die Hälfte (n/2) im Hauptspeicher gelesen werden muss, bis der entsprechende Datensatz gefunden wird. Das Löschen eines Datensatz bewirkt, dass auch der Speicherplatz nicht mehr zur Verfügung steht, sodass die Reorganisation zu gegebener Zeit unverzichtbar wird. Heap eignet sich speziell für kleinere Datenbanken zum konstruktiven und schnellen Arbeiten.([28], S. 46)

Hash-Funktion

Die Hash-Funktion repräsentiert eine alternative Speicherstruktur, die mithilfe einer mathematischen Funktion den Hash-Wert ermittelt und den Datensatz gemäß dieses Wertes an der entsprechenden Adresse hinterlegt. Oftmalig wird zum Errechnen des Hash-Werts die mathematische Funktion Modulo n verwendet. Anhand des entstehenden Rests aus dieser Funktion kann ein Speicherblock auf dem Sekundärspeicher ermittelt werden.([28], S. 48)

$$H(a) := a \text{ MOD } n$$

Ist der ermittelte Speicherplatz bereits belegt, werden die Daten sequentiell in einen speziellen Überlaufbereich geschrieben. Alternativ kann nach einem freien Speicherblock gesucht werden, um den Datensatz einzufügen.

Die Hash-Speicherstruktur offeriert ein sehr effizientes Verfahren, Datensätze im Speicher abzulegen, da diese anhand einer einfachen mathematischen Funktion die Suche nach der Adresse des Datensatzes berechnet. Zusätzlich sind die Löschoperationen sehr hilfreich. Ein Defizit ist jedoch, dass Bereichsabfragen nicht durchführbar sind, da lediglich eine Adresse des Speicherblocks ermittelt werden kann.

ISAM

Die ISAM-Speicherstruktur (Index Sequential Access Method) ähnelt der B*-Baum Struktur. Dabei werden die Daten zunächst indiziert und in den entsprechenden Blöcken auf der Festplatte abgelegt. Die Schlüssel werden dazu in den einzelnen Blöcken ihrer Größe nach aufsteigend sortiert und stellen somit die logische Verbindung zum Index her. Das Einfügen neuer Werte erfolgt mithilfe des Index. Dort wird als Erstes die Blockadresse für den Datensatz bestimmt und dieser anschließend, gemäß seines Werts, in die sortierte Liste eingefügt. Ist es nicht möglich einen neuen Datensatz einzufügen, da der entsprechende Block bereits belegt ist, erfolgt die Speicherung in einem sogenannten Überlaufblock. Die Suche innerhalb der ISAM-Speicherstruktur ist sehr leistungsfähig, jedoch nur bei wenigen Überlaufblöcken. In diesem Szenario findet eine Leseoperation auf den Index statt, um die entsprechende Adresse des Datensatzes zu ermitteln. Im Anschluss ist eine weitere Leseoperation im Block geboten, um den Datensatz auszulesen.

Aufgefundene Werte können gelöscht werden, wodurch die Freigabe des Speicherplatzes wieder eintritt. Eine Neustrukturierung der Indexseite ist unverzichtbar, falls es sich um den ersten gelöschten Wert handelt.([30], S. 97)

B*-Baum

B*-Bäume stellen eine Erweiterung des B-Baums dar, unterscheiden sich jedoch dahingehend, dass diese mindestens zu 2/3 aufgefüllt sein müssen. Betrachtet man den Wurzelknoten, so weisen dessen linke Blätter kleinere und dessen rechte Blätter größere oder gleich große Schlüsselwerte hinsichtlich des Wurzelknotens auf. Alle Bäume, seien es B-Bäume, B+-Bäume oder B*-Bäume, müssen stets ausbalanciert sein, sodass sich für alle Datensätze ein gleichbleibender Suchaufwand ergibt. Der Wurzelknoten enthält prinzipiell ein Element und die Blätter müssen immer bis zur Hälfte gefüllt sein. Das Schreiben und Lesen von Werten erfolgt nach demselben Muster analog der Index Sequential Access Method. Die Integration des Werts in die sortierte Liste entspricht dabei der Reihenfolge.([28], S. 52 ff.)

4.5 Regulärer Ausdruck

Reguläre Ausdrücke spielen, insbesondere bei relationalen Datenbanken, zur Definition eines festen Schemas eine entscheidende Rolle. NoSQL-Datenbanken hingegen sind schemafrei und somit nur bedingt auf reguläre Ausdrücke angewiesen.

Der reguläre Ausdruck ist eine Zeichenkette, die mit syntaktischen Regeln, Mengen und Untermengen von differenten Zeichenketten beschreibt. In der Praxis werden diese Ausdrücke meist zur Überprüfung von bestimmten Zeichenmustern verwendet. Beispielsweise findet eine Überprüfung von Formularen (Bsp.: Benutzernamen, Adressdaten, Passwörter und E-Mail-Adressen) häufig im Web Anwendung.

Ein regulärer Ausdruck einer E-Mail-Adresse kann sich folgendermaßen zusammensetzen:

^([a-zA-Z0-9_+-\/\.]+)@((([a-zA-Z0-9\-]+)).([a-zA-Z]{2,4}|[0-9]{0,max})(\]?)$

Das Element vor dem @-Zeichen lässt alle möglichen Buchstaben von a-z zu, wobei die Groß- oder Kleinschreibung bedeutungslos ist. Des Weiteren können auch die Zahlen 0-9 + - _ \/ verwendet werden. Das Symbol + nach Abschluss des ersten regulären Ausdrucks gibt an, dass dieser zumindest ein einziges Mal existieren muss, aber auch mehrfach verwendet werden kann. Nach dem @-Zeichen, dass sich zwingend nach dem ersten Teilbereich des regulären Ausdrucks anschließen muss, sind wiederum alle angegebenen Zeichen anwendbar. Im Anschluss daran folgt der Punkt, der in den meisten Fällen den Providernamen vom Ländercode abgrenzt. Nach dem Punktsymbol ist erneut die Groß- oder Kleinschreibung applikabel. Die Zeichenkette muss aus mindestens zwei, aber höchstens vier Zeichen bestehen. Gleichermaßen kann an den Ländercode eine natürliche Zahl angehängt werden, die eine Länge von 0-max aufweist. Die Inanspruchnahme dieses Ausdrucks ist nicht zwangsläufig erforderlich, was anhand des Fragezeichens im regulären Ausdruck symbolisiert wird. ([28], S. 73)

Die Abfrage einer gültigen E-Mail-Adresse wird unter MySQL mithilfe folgender Syntax ausgeführt:

```
mysql>  SELECT Emailadresse
        FROM Person
        WHERE Emailadresse RLIKE
        „^([a-zA-Z0-9_+-\/\.]+)@((([a-zA-Z0-9\-]+)).([a-zA-Z]{2,4}|[0-
        9]{0,max})(\]?)$“
        LIMIT 5;
```

Daraus resultiert folgendes beispielhaftes Ergebnis:

```
+---------------------------------------------+
| Emailadresse                                |
+---------------------------------------------+
| 3.Mustermann@yahoo.de                       |
| 98301@aol.com                               |
| 2357.Mustermann@web.de                      |
| Mustermann@info.de                          |
|Mustermann.Musterfrau@gmx.de                 |
+---------------------------------------------+
```

4.6 Verteilte Datenhaltung

Höhere Anforderungen sowie Datenmengen durch soziale Netzwerke und Online-Shops lassen die Stimmen nach schnellen Lese- und Schreibzugriffen auf die Datenbank lauter werden. Besonders NoSQL-Systeme haben sich diese Marktlücke zu Nutze gemacht und bieten Alternativen, wo relationale Datenbanken in den letzten Jahren an ihre Grenzen gestoßen sind. Um wettbewerbsfähig zu bleiben, müssen auch relationale Datenbanken die horizontale Skalierung siehe Kapitel 3.1.1 zum Einsatz bringen, ohne dabei Leistungseinbußen zu verzeichnen.

Unter Verwendung von Replikationen ist eine einfache Verteilung der Daten auf mehrere Rechner durchführbar. Durch jahrelange Praxis wurde evident, dass die redundante Speicherung der Daten nicht nur vor Ausfallsicherheit schützt, sondern auch die Kommunikationskosten senkt. Anfragen können sowohl an geografisch günstige Replikate verwiesen und gleichzeitig auf leistungsstarke Systeme verteilt werden, die enorme Geschwindigkeitsvorzüge mit sich bringen. ([1], S. 448])

Abbildung 26 veranschaulicht das Master-Slave Prinzip mit der gleichmäßigen Aufteilung der replizierten Daten auf die Server (Slave 1, Slave 2 und Master). Erfolgt seitens des Nutzers oder des Systems ein Lesezugriff, so kann dieser gleichermaßen auf die vorhandenen Server verteilt werden, umso eine Überlastung eines einzelnen Servers auszuschließen. Konträr zu einem einzelnen Server gestaltet sich der Schreibzugriff auf die Datenbank deutlich komplexer. Bei einem zeitgleichen Schreibzugriff auf alle Server kann die Konsistenz nicht mehr hergestellt und möglicherweise könnten veraltete Daten übernommen werden. Zur Konfliktvermeidung findet die Kommunikation ausschließlich mit dem Masterknoten statt. Die nachfolgende Synchronisation mit dem Slave-Knoten kann

dabei synchron mit sofortiger Konsistenzherstellung oder asynchron erfolgen, was bei geografisch verteilten Systemen meist die schnellere Verarbeitung von Schreibzugriffen darstellt.

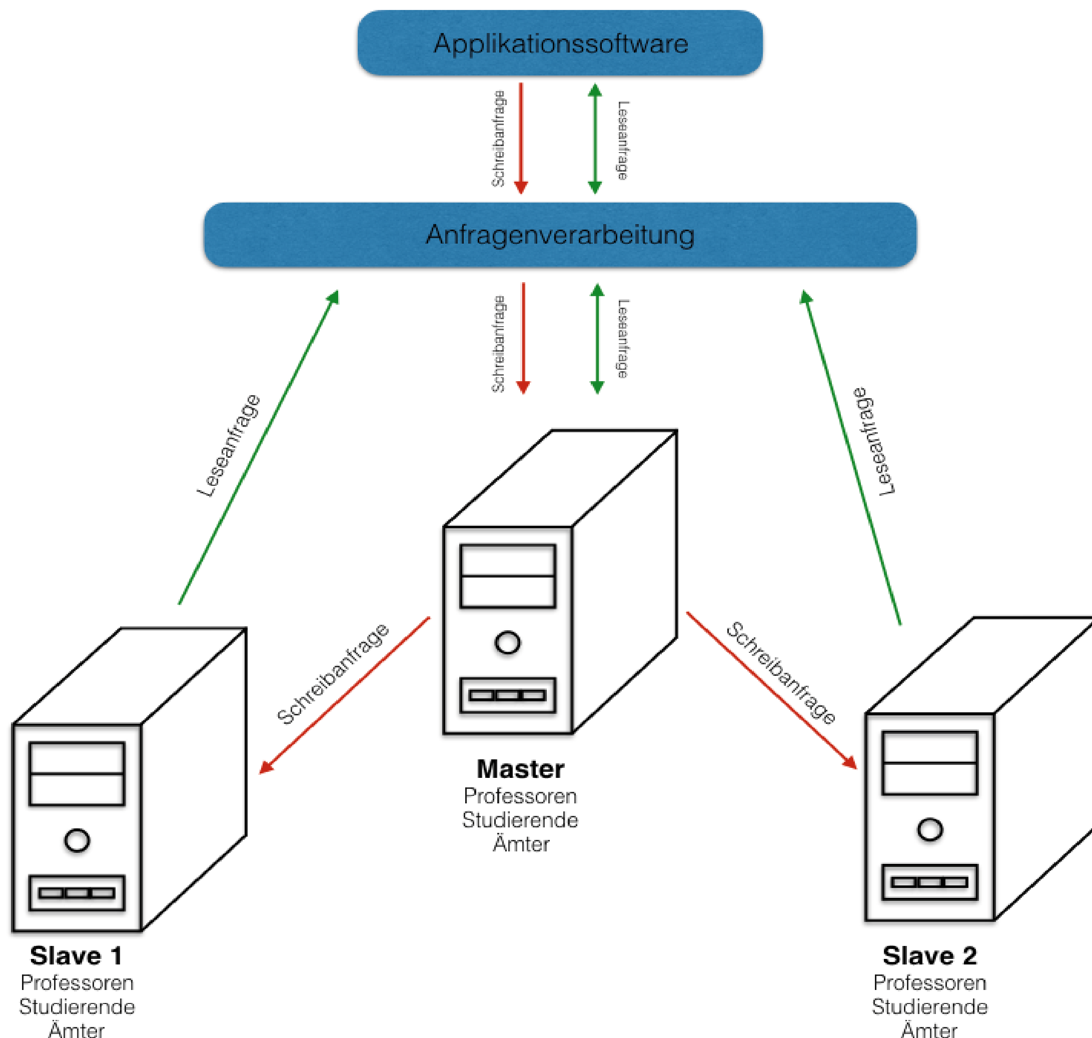


Abbildung 26: Anfrageverarbeitung einer replizierten Master-Slave Architektur

Die Replikation der Daten hat nicht nur positive Aspekte zur Folge. Insbesondere wirken sich der erhöhte Speicherplatz für die Vielzahl an Replikationen und die steigende Systemkomplexität ungünstig aus. Replikationen eignen sich zunächst sehr gut für Leseanfragen, jedoch weisen sie aufgrund des Master-Slave Prinzips erhebliche Defizite auf, sobald es zu zahlreichen Schreiboperationen kommt. Mithilfe des Top-Down Ansatzes ([20], S 77 ff.) kann diese Problematik gelöst werden, der ähnliche Strukturen und Funktionsweisen wie etwa Map/Reduce aufzeigt. Hierbei wird die zu schreibende Datenmenge zerkleinert und logisch aufgeteilt, sodass die Modifizierung vom Masterknoten und von den einzelnen Schreiboperationen der entsprechenden Server bearbeitet wird.

Zur Gewährleistung eines reibungslosen Ablaufs muss zunächst das Server-Cluster gemäß der Entitäten des Datenmodells partitioniert werden. Obige Abbildung demonstriert folgende Partitionierung der Hochschule Anhalt: Studenten (Mat. Nr., Adresse, Leistungen), Dozent (Perso. Nr., Fachbereich, Sprechzeiten) und Verwaltung (Prüfungsamt, Öffnungszeiten, Raumplan). Gleichwohl kann die Partitionierung einzelner Relationen dazu führen, dass infolgedessen die Kapazitäten der Server überschritten werden. Abhilfe schafft die horizontale sowie vertikale Fragmentierung. Die horizontale Fragmentierung unterteilt die Relation zeilenweise (horizontal) in strukturell gleiche Fragmente, um diese dann auf verschiedene Server zu distribuieren. Ebenso ist dies auch vertikal praktikabel. Hierzu wird eine Relation spaltenweise (vertikal) zerlegt, wobei die Unterteilung jederzeit vollständig rekonstruiert werden kann. Relevant ist hierbei, dass jedes der einzelnen Fragmente stets den entsprechenden Primärschlüssel beinhaltet.([20], S. 138,147) Abbildung 27 zeigt die horizontale bzw. vertikale Fragmentierung am Beispieldatensatz der Hochschule Anhalt.

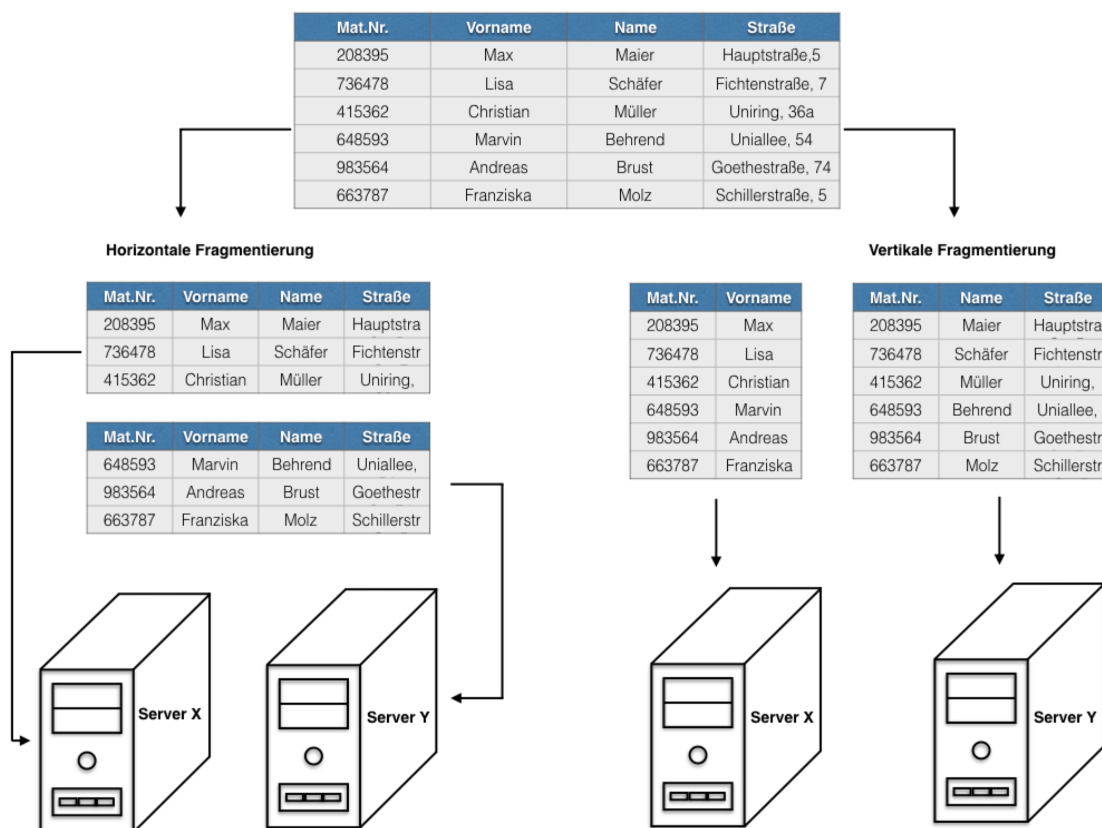


Abbildung 27: Horizontale und vertikale Fragmentierung im direkten Vergleich

4.6.1 Verteilte Anfragenverarbeitung

Die verteilte Anfragenverarbeitung gestaltet sich durch die Fragmentierung der Datenmenge von einem zentralen Rechner auf viele kleinere lokale Rechner recht diffizil. Die zentrale Anfragestellung an ein Datenbanksystem ist nicht mehr gegeben, daher ist von Anbeginn eine Aufteilung in mehrere lokale Anfragen unverzichtbar. Zur Kostenminimierung müssen die Anfragen im Optimalfall parallel auf den entsprechenden Rechnern ausgeführt und abgearbeitet werden. Beim Zurückgeben müssen die lokalen Rechner ihre lokalen Ergebnisse ähnlich wie bei Map/Reduce schließlich zu einem globalen Ergebnis zusammenführen und an die Anwendungssoftware ausgeben. Abbildung 28 präsentiert die Hochschule Anhalt sowie deren Standorte durch drei lokale Server. In der Abfrage wird recherchiert, welche Professoren älter als 50 Jahre sind und am gleichen Hochschulstandort lehren.([20], S. 117 ff.)

Deutlich komplexer gestaltet sich eine lokale Abfrage, wenn diese von zusätzlichen Abfragen abhängig ist. Beim Einsatzgebiet von Verbundoperationen ist es deshalb unumgänglich, dass die Abfrage sequentiell erfolgt, damit Ergebnis A vor Ergebnis B zurückgeliefert wird, da B von A abhängig ist. Trotz der Option Verbundoperationen zu verwenden, sollten diese wegen ihrer hohen Komplexität vermieden werden. Generell werden verteilte Anfragen fast ausnahmslos vor dem Anwender verborgen, sodass eine Regulierung seitens der Datenbank stattfindet. Dennoch gilt, dass Anfragen an verteilte Datenbanksysteme grundsätzlich zeitintensiver sind als bei nicht verteilten Systemen.

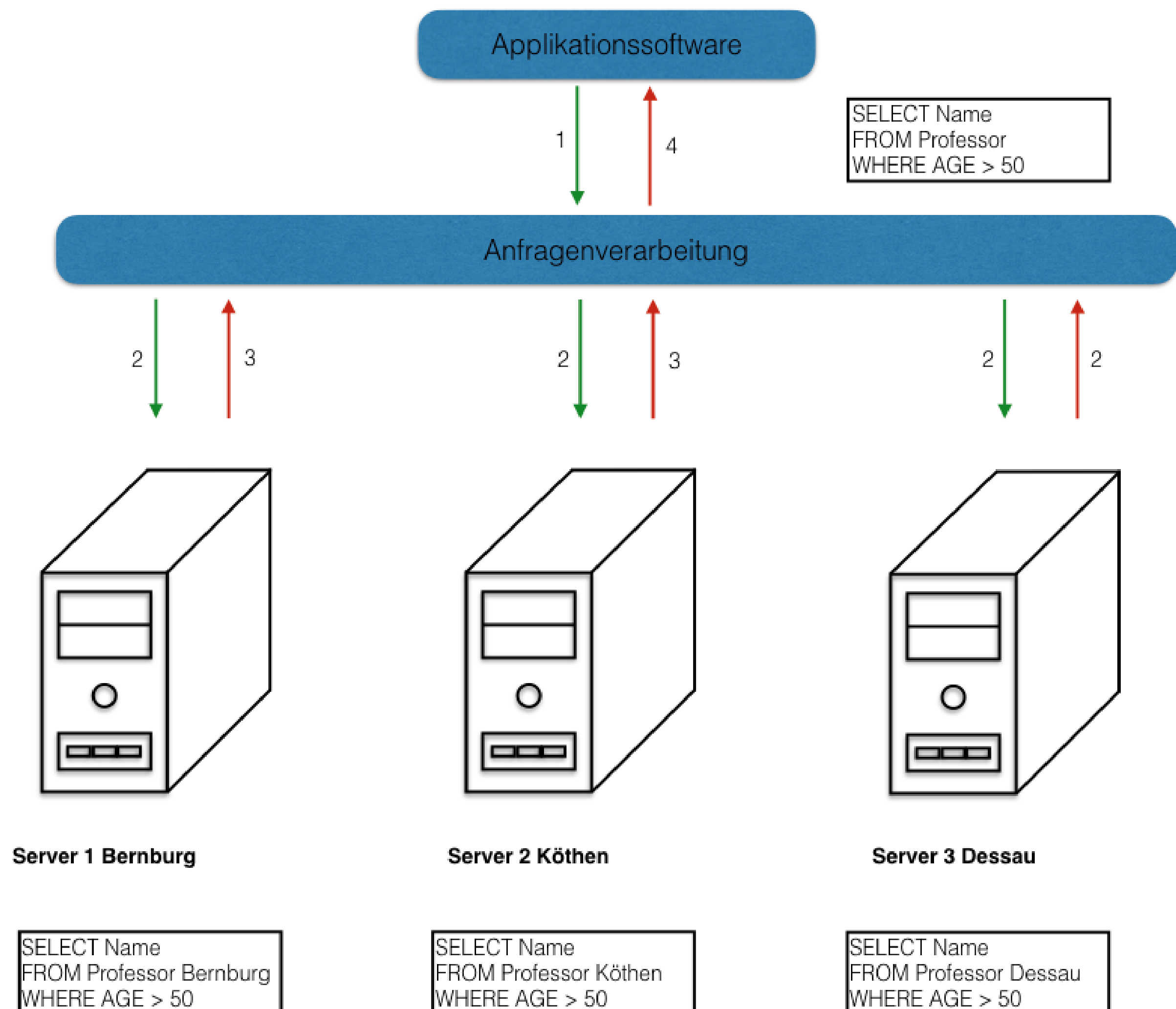


Abbildung 28: Vierstufiger Verarbeitungsprozess einer Leseanfrage

4.6.2 Synchronisation

Die Synchronisation in verteilten Datenbanksystemen hat wesentlich zum Erfolg des Mehrbenutzerbetriebs beigetragen. Bei Fehlern in der Synchronisation der Datensätze können eine Reihe von unerwünschten Phänomenen entstehen. Häufig sind dabei Anomalien zu verzeichnen, wie etwa verloren gegangene Änderungen „lost updates“, Lesen „schmutziger“ Änderungen „dirty read“, inkonsistente Analyse „non-repeatable read“ sowie Phantome.

Als „lost updates“ werden Änderungen bezeichnet, die durch eine zweite oder dritte Transaktion fälschlicherweise überschrieben werden. Unter „dirty read“ versteht man Transaktionen, die nicht vollständig bearbeitet wurden, sodass ein möglicher Lesezugriff auf einen veralteten oder fehlerhaften Datensatz erfolgt. Die inkonsistente Analyse sowie

das Phantom-Problem beschreiben parallel ausgeführte Änderungen, sodass durch einen Lesezugriff unterschiedliche Datenbankzustände sichtbar sind.

Generell wird zwischen der zentralen sowie der verteilten Synchronisation differenziert, wobei die Funktionsweise der zentralen Synchronisation identisch mit nicht verteilten Datenbanksystemen ist. Dabei wird jede Transaktion von einem Masterknoten auf deren Slaves synchronisiert, ohne dass es zu den zuvor beschriebenen Anomalien kommt. Dieses Verfahren bedeutet einen erhöhten Kommunikationsaufwand einhergehend mit deutlichen Performance-Einbußen. Aufgrund dessen muss eine verteilte Synchronisation in einem verteilten Datenbanksystem stattfinden, sodass eine Transaktion auf dem entsprechenden Rechner lokal ausgeführt wird. Dies wird mithilfe des Commit-Protokolls zur Vermeidung unterschiedlichster Anomalien koordiniert ([29], S.136). Auf die Konkretisierung diverser Verfahren zur Behandlung von Deadlocks sowie Zeitmarkenverfahren wird verzichtet, da diese für den anschließenden Benchmarktest unbedeutend sind.

4.7 Datenbanksicherheit und Autorisierung

Die enorm wachsende Informationstechnologie verbunden mit einem kostengünstigen Anschaffungspreis elektronischer Geräte lässt in zunehmenden Maß kritische Stimmen nach mehr Datensicherheit, insbesondere bei sensiblen Daten, laut werden. Hierbei liegt beispielsweise der Fokus auf Patientendaten, die streng vertraulich sind und gesetzlich geschützt werden müssen vor dem Zugriff durch unberechtigte Personen. Ebenso gilt dies für Informationen auf Regierungs- und Firmenebene, die nicht öffentlich zugänglich sein dürfen.

Die Datenbanksicherheit wird mit verschiedensten Mechanismen garantiert. Hierbei kommt dem Datenbankadministrator (DBA) die zentrale Autorität für die Verwaltung des Datenbanksystems zu. Der DBA hat den Auftrag Privilegien an Nutzer zu vergeben sowie die Einstufung von Nutzern und Daten in Übereinstimmung mit den Sicherheitsvorschriften der Organisation zu verwalten.

Der DBA oder auch Superuser-Account beinhaltet Berechtigungen, die anderen Datenbank-Accounts und –nutzern nicht zur Verfügung stehen, wodurch dieser zur Implementierung folgender Aktionen privilegiert ist: ([30], S. 469)

- *Anlegen von Accounts:* Hiermit erhalten Nutzer oder Nutzergruppen einen Zugang zum DBMS, der mithilfe eines Benutzernamens sowie eines Passworts geschützt ist.
- *Vergabe von Privilegien:* Mithilfe von Privilegien werden verschiedene Sonderfunktionen für Nutzer freigegeben.

- *Widerruf von Privilegien:* Optional kann die Vergabe der zuvor gestalteten Privilegien widerrufen werden.
- *Zuweisung von Sicherheitsstufen:* Anhand derer werden Zuweisungen von Benutzer-Accounts auf der entsprechenden Sicherheitsstufe angelegt.

Die einfachste Form der Zugriffskontrolle erfolgt unter Verwendung von Benutzernamen sowie Passwörtern, die zu Beginn eines Datenbankzugriffs die Legitimation des Nutzers aufweisen müssen. Dabei wird die Gültigkeit der Daten überprüft, bevor der Zugang bewilligt wird. Die Verwaltung der Login-Daten übernimmt das Datenbanksystem, indem in einer Tabelle sowohl Benutzernamen als auch Passwörter für die verschiedenen Nutzer angelegt und verwaltet werden. Unter Verwendung von Log-Files können zudem im Falle von inkorrekten Aktualisierungsoperationen oder unzulässigen Manipulationen die entsprechenden Nutzer durch ihre Login-Daten ermittelt werden. Neben den Login-Informationen werden verschiedene Privilegien zur Verfügung gestellt.([30], S. 471) Dies bedeutet für Account-Eigentümer jedoch nicht uneingeschränkt zu allen angebotenen Funktionen Berechtigung zu haben. Prinzipiell werden zwei Arten von Ebenen zur Nutzung unterschieden:

- *Account Ebene:* Bildet die oberste Ebene. Hier werden seitens des DBA spezielle Privilegien für jeden Account definiert, die unabhängig von den Relationen innerhalb der Datenbank sind.
- *Relationsebene:* Stellt die unterste Ebene dar. Hier wird mittels des DBA festgelegt, welche Relationen von den verschiedenen Accounts gesichtet und bearbeitet werden dürfen.

Die Privilegien auf Account-Ebene beschreiben die Rechte einzelner Accounts. Dies betrifft Funktionen wie CREATE SCHEMA oder CREATE TABLE, um Schemata oder Basisoperationen zu erstellen, aber auch das Hinzufügen (INSERT) oder Löschen (DELETE) von Attributen aus der Relation.

Auf Relationsebene werden die einzelnen Privilegien für Relationen und Attribute in der Tabelle fixiert. Dadurch ist es möglich verschiedene Nutzer von Teilen der Datenbank auszuschließen, beispielsweise, wenn dies für deren Arbeit irrelevant ist. Die Vergabe von Privilegien erfolgt seitens des DBA mithilfe des GRANT-Befehls. Es wird beispielhaft angenommen das Datenbanksystem besteht aus vier Accounts X_1, X_2, X_3, X_4, wobei X_1 den DBA symbolisiert. Verwendet X_1 den GRANT-Befehl, um X_2 Rechte an den Basisoperationen zu übertragen, geschieht dies folgendermaßen:

GRANT CREATE TAB TO X_2

Ebenso kann X_1 den Funktionsumfang von X_2 reduzieren und nur ausgewählte Operationen tolerieren.

GRANT INSERT, DELETE ON MUSTERTABELLE TO X_2

Besteht für X_2 keine Legitimation mehr an der Datenbank bzw. der Mitarbeiter scheidet aus dem Unternehmen aus, werden dessen Rechte durch X_1 unter Zuhilfenahme der REVOKE-Funktion entzogen.

REVOKE INSERT ON MUSTERTABELLE FROM X_2

Bei der Applikation des GRANT-Befehls von X_1 auf X_2 wird X_2 gleichfalls das Recht zum Transfer der Privilegien an andere Accounts zugeteilt. Anhand der Ganzzahl i wird dies unterbunden. X_1 verhindert mit Angabe von i=0, dass X_2 die Rechte weitergeben kann. In gleichem Maße definiert X_1 mittels i, wie viele Übertragungen von X_2 an andere Accounts erfolgen können.

Die vorgestellten Methoden zur Datensicherheit von Datenbanken zeigen grundlegende Techniken, die unkompliziert und mit wenig Zeitaufwand umgesetzt werden können. Es wird auf die Vorstellung weiterer Aspekte verzichtet, da dies an dieser Stelle zu umfangreich wäre.([30], S. 476)

4.8 Kritik an relationalen Datenbanken

Relationale Datenbanken können ihre Position gegenüber den NoSQL-Systemen erfolgreich durchsetzen. Ein Spezifikum ist die plattformübergreifende Anfragesprache SQL. Bei der Einarbeitung in die Systemlandschaft von Berkeley DB oder von Oracle kann die Funktionsweise wie auch die Anfragesprache auf differenten relationalen Datenbanksystemen sei es DB2 von IBM oder Microsoft Access angewendet werden. Dennoch existieren durchaus auch Defizite. Die Segmentierung von Objekten auf viele inhomogene Relationen stellt ein bedeutsames Negativum dar. Anwendungsobjekte beinhalten meist eine sehr komplexe Struktur, da sie selbst aus mehreren Objekten bzw. Listen bestehen. Generell werden Tupelmengen von unterschiedlicher Größe gespeichert, daraus resul-

tiert, dass die multiplen Anwendungsobjekte mithilfe zahlreicher Operationen der verschiedensten Relationen reproduziert werden müssen. Darauf basieren sehr komplexe sowie unübersichtliche Anfragen, die den Zugriff auf ein Objekt ineffizient gestalten und hinsichtlich des Entwicklungsaufwands deutlich aufwendiger machen. Faktisch ist auch die Eindeutigkeit der Primär- sowie Fremdschlüssel kritisch zu sehen. Eindeutige Attribute bzw. Werte werden nur bedingt vorgehalten, sodass künstliche Schlüssel erzeugt werden müssen. Dies hat zur Folge, dass kein Bezug zur Relation besteht und lediglich die Funktion einer Verwaltungsinformation übernommen wird. Die vielfältige Struktur der Datenbank bedarf eines größeren Speicherplatzes, wodurch das System, wenn auch nicht merklich, ineffizient arbeitet. Zudem entstehen vor allem dann Probleme, wenn es die horizontale Skalierung der Systeme betrifft. Die Systeme eignen sich für diese Art der Skalierung nur bedingt, da sie keine Schemafreiheit besitzen und nur beschränkt mit Replikationen arbeiten. Oracle bietet mittlerweile verschiedene Lösungen wie beispielsweise objektorientierte Datenbanken oder In-Memory Datenbanken zur Bewältigung des stetig steigenden Datenaufkommens an.([20], S. 123)

4.9 Einleitung in MySQL

MySQL ist ein relationales Datenbanksystem der Oracle Corporation, das 1994 unter dem Namen mSQL gegründet wurde. Im Fokus stand dabei die Verarbeitung großer Datenmengen sowie hoher Performance. Zusätzlich wurde MySQL in C sowie C++ entwickelt und kann somit auf allen gängigen Linux- und Windowssystemen ausgeführt werden. Neben verschiedener API´s und Bibliotheken ist es zudem zur Verwendung von Kernel-Threads Multithread fähig. Dies erlaubt es nicht nur auf einem Kern zu arbeiten, sondern sich den heutigen Multicore Architekturen anzupassen. Infolgedessen wurden auch die maximalen Speichergrößen von ursprünglich 4 Gbyte in Version 3.22 auf 65.536 Terabyte in Version 3.23 erhöht, um den steigenden Bedürfnissen zu genügen. Das Feature von MySQL stellt der Webservice unter Zuhilfenahme der Skriptsprache PHP (Personal Home Page/ Hypertext Preprocessor) dar. Mithilfe von PHP werden dynamische Webseiten erzeugt, die syntaktisch an die Programmiersprache C angelehnt sind. Trotz der Übernahme von Oracle im Januar 2010 wird MySQL weiterhin unter der General Public License (GPL) verbreitet. Besonders attraktiv ist dieses Lizenzmodell für Entwickler, da der individuelle Gebrauch der Software keinerlei kommerziellen Bedingungen unterliegt.

4.9.1 phpMyAdmin

PhpMyAdmin ist eine Weboberfläche, die für den grafischen Zugriff auf MySQL-Datenbanken konzipiert wurde. Neben der grafischen Aufbereitung werden zahlreiche weitere Funktionen wie beispielsweise Informationen über Tabellenformate und Zeichensätze offeriert. Mit der aktuellen Version von phpMyAdmin können zudem über die GUI (Graphical User Interface) neue Tabellen sowie Datensätze induziert werden. Dennoch muss die Datenbank auch in ihrer neusten Version händich über die Administrationsoberfläche generiert werden. Neben dem Erstellen, Bearbeiten und Löschen von Tabellen und Datensätzen können über phpMyAdmin SQL-Queries erzeugt werden, die an den Datenbankserver übertragen werden. Die Ausgabe einer Anfrage mit korrekter Syntax wird in der GUI der Weboberfläche generiert. Darüber hinaus bietet phpMyAdmin zahlreiche Funktionen in Verbindung mit MySQL bzw. MariaDB und ist insbesondere als Tool an deren Seite von enormer Bedeutung.[25]

4.10 Zusammenfassung

Relationale Datenbanken sind seit Jahren der Platzhirsch unter den Datenbanksystemen und zählen bis dato zu den am häufigsten verwendeten kommerziellen Systemen. Durch die Big Data Bewegung wurde eine neue Datenbankgeneration wie NoSQL hervorgebracht, die jedoch mit dem voluminösen Erfahrungspotenzial der relationalen Datenbanken nicht mithalten kann. Das Entity-Relationship-Modell stellt für die Applikation von relationalen Datenbanken eine große Hilfe dar, da hiermit die Kausalität der realen Welt auf eine abstrakte Ebene zur Modellierung inhaltlicher Bezüge in den Tabellen überführt wird. Trotz des Abstraktionsniveaus sind die Zusammenhänge durch verschiedene Geometrien grafisch aufbereitet, sodass diese leicht verständlich sind. Das Fundament bilden die Änderungs- und Anfrageoperationen, die es arrangieren den existierenden Datenbestand zu modifizieren. Die Anfrageoperationen instruieren zunächst die Grundlage zum Bearbeiten neuer Tupel in der Datenbank. Die Änderungsoperationen basieren auf den Operationen der Mengenlehre und sind dabei exklusiv, sodass diese nicht durch andere Operationen kommutiert werden können.

Die Anfragesprache (SQL) ist ein mächtiges Instrument dieses Systems und hat maßgeblich zur Etablierung als Standardlösung beigetragen. Mithilfe dieser können innerhalb der Datenbank Informationen gesucht, aufbereitet und zusammengestellt werden. Durch die Deskriptivität der Sprache wird kein konkreter Lösungsweg definiert, sondern das erwartete Ergebnis beschrieben. Somit können verschiedene SQL-Anfragen zur selben Lösung führen, sodass der Nutzer das für ihn ideale Ergebnis selbst bestimmen kann.

Die verschiedenen Speicherstrukturen sind für das Erstellen einer Datenbank sowie den späteren Zugriff darauf entscheidend. Das Arbeiten mit der Speicherstruktur Heap ist zudem einfach zu erstellen, hat jedoch den Nachteil, dass bei größeren Datenmengen immer n/2 Blöcke durchsucht werden müssen, bis die Information eruiert wird. Andere Speicherstrukturen wie ISAM oder B*-Bäume sind deutlich komplexer und zeitintensiver bei der Erstellung, jedoch erfolgt die spätere Informationssuche bei dieser Speicherstruktur deutlich schneller.

Die verteilte Datenhaltung mithilfe von Replikationen, Fragmentierung, Allokation sowie verteilter Transaktionsverwaltung führt aufgrund des sehr großen und mächtigen Funktionsumfangs oft zu Komplikationen, da viele dieser Systeme eine verteilte Datenhaltung nur bedingt ermöglichen. Zusätzlich bedarf es zur Replizierung eines relationalen Datenbanksystems an Know How. Konträr hierzu sind beispielsweise NoSQL-Systeme, die auf die verteilte Datenhaltung spezialisiert sind und teilweise binnen kürzester Zeit ohne Fachkompetenz repliziert werden können.

MySQL bietet bezüglich der Datenverteilung einige Lösungsverfahren, so können beispielsweise mehrere Replikate asynchron aktualisiert werden. Dennoch bleibt es auch hier Aufgabe der Applikationslogik Leseanfragen zu verteilen, da dies von MySQL nicht übernommen wird. Neben der Kritik an fehlender Replikation von Daten ist die plattformübergreifende Verwendung positiv anzumerken, da Oracle wie auch Microsoft bei der Anfragesprache und dem Aufbau einer solchen Datenbank immer nach demselben Schema verfahren.

5 Benchmarktest

Der praktisch orientierte Teil der Masterarbeit demonstriert anhand von Benchmarktests die fundamentale Divergenz der Datenbanksysteme NoSQL und relationaler Datenbanken.

Der Begriff Benchmarking ist dabei in den unterschiedlichsten Wissenschaftsbereichen als genormtes Mess- und Bewertungsverfahren klar definiert, wohingegen beim Begriff Benchmark eine Dissonanz in der Wissenschaft herrscht. Benchmark entspricht in der Geografie bzw. in der Wirtschaftswissenschaft einem Referenzwert an dem Vergleichsmessungen durchgeführt werden. In der Informatik hingegen und besonders bei Datenbanken beschreibt Benchmark das Programm, das die Messung unter Einfluss einer generierten Last durchführt [5]. Benchmarks mit Bezug zu Datenbanken bestehen aus diversen Anfrage- und Änderungsoperationen sowie der Anfragesprache SQL mithilfe derer Messungen zwischen verschiedenen Systemen und deren Leistungsfähigkeit bewertet werden können. In der Folge wird ausschließlich der Begriff Benchmark sowie das entsprechende Suffix verwendet.

Auf unterschiedlichen Abstraktionsebenen können somit Benchmarktests erfolgen. Für die zugrundeliegende wissenschaftliche Arbeit ist jedoch nur der Applikations-Benchmarktest relevant, da Kernelbenchmarktests und synthetische Benchmarktests sich überwiegend mit der konventionellen Rechnerarchitektur sowie deren Kapazitäten beschäftigen. Im durchzuführenden Applikations-Benchmarktest werden die differenten Datenbankmodelle getestet, die auf ein und demselben Rechner ausgeführt werden, da folglich die Rechenkapazität ohnehin konstant bleibt.

Ein Benchmarktest hat prinzipiell die Evaluierung multipler Abfragen sowie Operationen von Datenbanken zum Ziel, um Schwachstellen zu dokumentieren und daraus optimierende Prozesse einzuleiten. Zur besseren Analyse werden die verschiedenen Softwarelösungen zum Erfassen der Messwerte nutzbringend innerhalb eines Benchmarktests eingesetzt. Dieser muss jedoch drei von vier der nachfolgend aufgelisteten Kriterien erfüllen, um verwertbar zu sein.

Portabilität: Zur realitätsnahen Durchführung muss eine identische Applikation für NoSQL-Systeme wie auch für relationale Datenbanken verwendet werden. Dies stellt eine Obstruktion dar, da zurzeit noch keine Applikation zur Testung beider Systeme existiert. Die Applikationen sind dabei immer auf ein

bestimmtes Datenbanksystem spezialisiert. Für den nachfolgenden Test sind die Abweichungen aufgrund des relativ kleinen Datensatzes sehr gering.

Repräsentatitvität: Aufgrund derer soll ein möglichst aussagekräftiges Ergebnis ermittelt werden. Dies soll mit dem vorhanden Datensatz sowie der verwendetetn Hardware realisiert werden.

Skalierbarkeit: Prinzipiell müssen die Tests auch für neue Rechnergenerationen und neue Testdatensätze geeignet sein. Da es sich in dieser Arbeit um eine Momentaufnahme unter aktuellen Gegebenheiten handelt, spielt die Skalierbarkeit auf zukünftige Systeme eine nur untergeordnete Rolle.

Systemverständnis: Anhand der Analyse sollen vor allem Leistungsunterschiede aufgedeckt und Optimierungen mithilfe der ersten Tests vorgenommen werden.

Die genannten Benchmark-Klassen dienen als Richtlinie und werden durch eine entsprechende Selektion des Benchmarktools den Anforderungen gerecht.

Die Portabilität des Tools wird durch die Programmierung erreicht, die möglichst wenig systemabhängige Anteile verwendet. Um dies zu erreichen, muss das Tool seitens des Nutzers implementiert werden. Im nachfolgenden Benchmarktest wird auf eine Open Source Softwarelösung von MySQL gesetzt, da eine Implementierung über den Rahmen dieser Masterarbeit hinausgehen würde. Neben der Portabilität gilt es auch bei der Skalierbarkeit diverses zu beachten. Die Größe der Datenmenge ist nachfolgend nicht veränderbar und muss mit der in der Testumgebung verwendeten Hardware verarbeitbar sein. Oftmals ist ein besseres Systemverständnis gegeben, sodass einzelne Parameter variiert werden können, um dadurch die Abhängigkeit der Leistung von diesen Lastparametern zu eruieren.

Die Repräsentativität der Ergebnisse ist besonders bei Kernel- und Applikationsbenchmarks als äußerst realistisch einzuordnen, da diese sich auf die verwendete Applikation bezieht und dort entsprechende Leistungsmessungen durchführt.([39], S. 37)

Ein Benchmarktest sollte grundsätzlich mehrmalig durchgeführt werden, um aussagekräftige Ergebnisse zu generieren. Oftmals werden bei einmaliger Verwendung fehlerhafte Ergebnisse geliefert, wegen unzureichender Kalibrierung oder gegenwärtiger Auslastung des Systems.

Generell sind Benchmarktests in jedem Bereich der Informatik rational, da mit deren Hilfe Defizite identifiziert und ausgemerzt werden können. Aufgrund hoher Kosten sowie dem Bedarf entsprechender Entwicklungszeit finden sie nicht immer Anwendung.

5.1 Installation und Konfiguration der Datenbank-Systeme

Die Installation und Konfiguration indiziert nachfolgend die Bezugsquellen der Systeme und präsentiert einen Einblick in deren Konfiguration. Es werden vor allem die notwendigen Parameter zur Vorbereitung des späteren Benchmarktests konkretisiert. Die Architektur sowie deren Funktionsvielfalt wird vernachlässigt, da diese aus den vorangegangenen Kapiteln zu entnehmen ist.

Im Anschluss werden die wichtigsten Konfigurationsschritte beginnend beim Download der Datenbank, über die erforderlichen Einstellungen der Parameter bis hin zum Import des Datensatzes für den späteren Benchmarktest dargelegt.

5.1.1 MySQL

Anhand von MySQL wird demonstriert, welche Einstellungen zur effizienten Durchführung des Benchmarktests unverzichtbar sind.

MySQL ist für den nicht kommerziellen Anwender in der Standardversion kostenlos unter MySQL.com[5] verfügbar. Neben dem MySQL-Community Server wird zudem die MySQL-Workbench installiert, da diese mithilfe der GUI eine aufwendige Konfiguration über das Terminal erspart. Beide Anwendungen werden unter der GPL-Lizenz (Generell Public License) vertrieben und können somit verwendet, geändert oder kopiert werden, solange die Lizenzrechte bei Software-Änderungen, -Erweiterungen oder bei der Verwendung von Softwareteilen erhalten bleiben.

Nach Download sowie Installation des MySQL-Servers und der MySQL-Workbench unter Mac OS Sierra wird mit der Konfiguration begonnen. Hierzu wird in der Workbench zunächst der MySQL-Server auf die Bedürfnisse des Nutzers angepasst.

[5] https://dev.mysql.com/downloads/

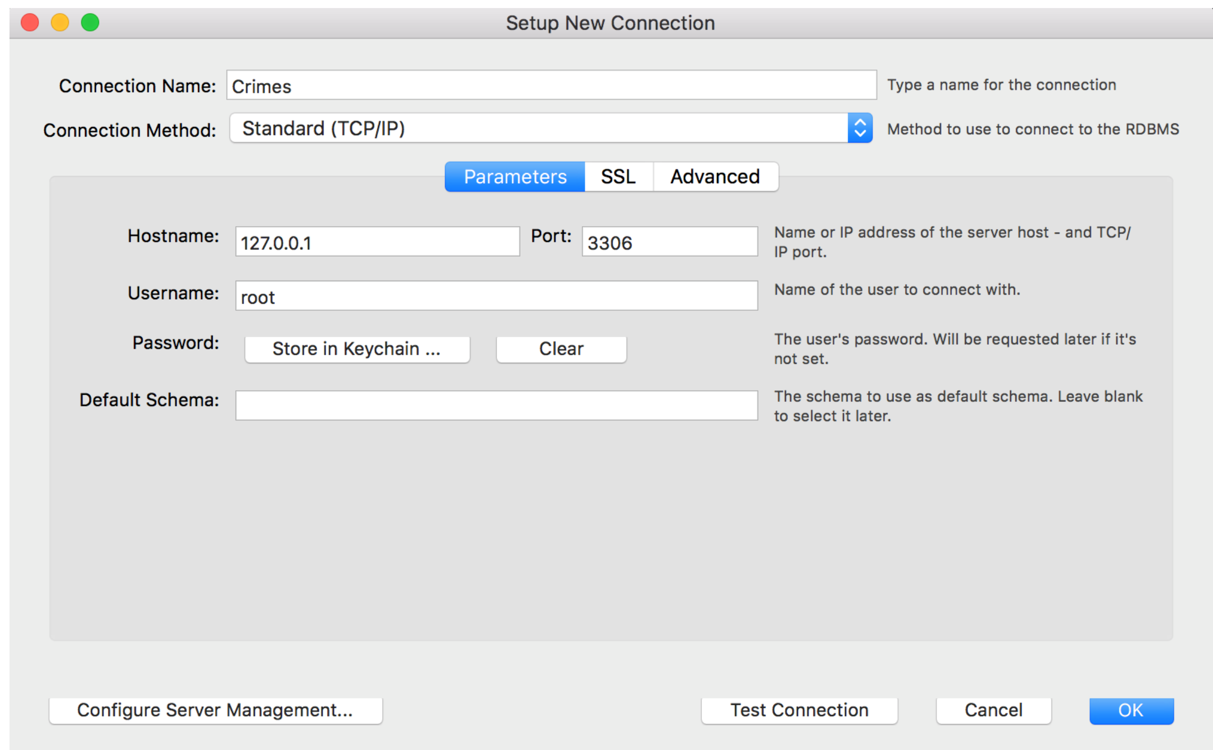


Abbildung 29: Konfiguration des Datenbank-Servers

Der Server erhält den Namen des späteren Datensatzes „Crimes“ s. Abbildung 29. Des Weiteren ist die Erreichbarkeit des Server unter der IP-Adresse 127.0.0.1 (localhost) und dem Port 3306 gegeben. Der Benutzername wird hierbei aus der Standardkonfiguration mit „root“ übernommen und das Password auf „0000“ gesetzt, da diese Eingabe zur korrekten Konfiguration des Servers unbedingt erforderlich ist.

Nach erfolgreicher Überprüfung der Server-Verbindung ist dessen Konfiguration abgeschlossen, sodass die MySQL-Datenbank nun verwendet werden kann.

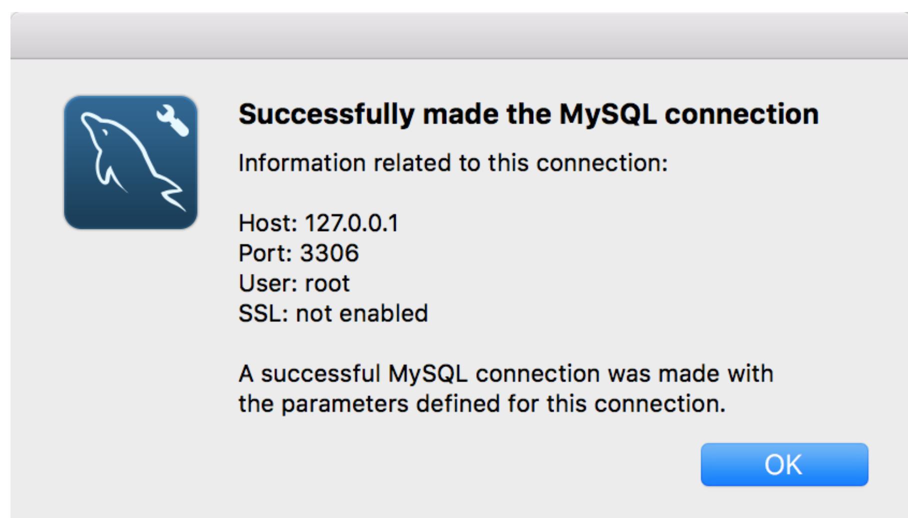


Abbildung 30: Überprüfung der Server-Verbindung

Als nächstes wird eine neue Datenbank mit dem Namen „Test12“ implementiert. Diese enthält die Tabelle mit dem Namen „crimes_-_2001_to_present“ sowie den Colums und den zugehörigen Datentypen des Testdatensatzes.

```
CREATE TABLE `crimes_-_2001_to_present` (
`ID` int (11) DEFAULT NULL,
`Case Number` text,
`Date` text,
`Block` text,
`IUCR` text,
`Primary Type` text,
`Description` text,
`Location Description` text,
`Arrest` text,
`Domestic` text,
`Beat` text,
`District` text,
`Ward` int (11) DEFAULT NULL,
`Community Area` int (11) DEFAULT NULL,
`FBI Code` text,
`X Coordinate` int (11) DEFAULT NULL,
`Y Coordinate` int (11) DEFAULT NULL,
`Year` int (11) DEFAULT NULL,
`Updated On` text,
`Latitude` double DEFAULT NULL,
`Longitude` double DEFAULT NULL,
`Location` text
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Mithilfe der “Table Data Import Wizard” Funktion wird der entsprechende Datensatz in die zuvor erzeugte Tabelle integriert. Zum Importieren des CSV (Comma-separated values) Datensatzes sind jedoch vorab Anpassungen notwendig. Als Line Seperator wird CR (Carriage return) gewählt, der angibt an welcher Stelle eine Textzeile übergeben wird. Als Field Seperator kann bei einer CSV-Datei nur das Komma als Trennzeichen gewählt werden, da andernfalls die Fehlermeldung „Unhandeld exception: list index out of

range" erscheint. Da MySQL keine Kodierung im ASCII-Format zulässt, wird alternativ das UTF-8 Format verwendet, welches im Bereich der ersten 128 Zeichen exakt dem ASCII-Format entspricht. Im Anschluss daran erfolgt automatisch der Datenimport der CSV-Datei. Dafür werden alles in allem 41242.240s benötigt, was annähernd 11,456 Stunden in Anspruch nimmt. Insgesamt werden 5667450 Datensätze importiert (siehe Abbildung 31).

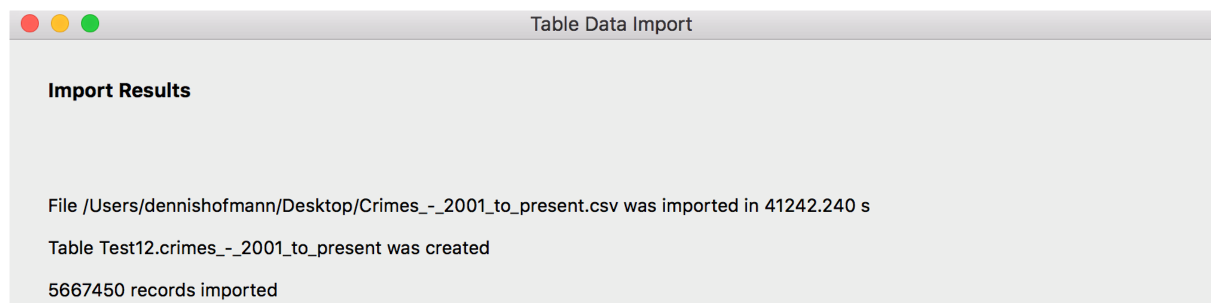


Abbildung 31: Erfolgreicher Import des Datensatzes

5.1.2 NoSQL – Apache Cassandra

Seitens der NoSQL-Systeme wird sich für Cassandra entschieden, das ein einfaches verteiltes Datenbankmanagementsystem ist und insbesondere sehr große Datenmengen verarbeiten kann sowie auf hohe Skalierbarkeit und Ausfallsicherheit ausgelegt ist.

Ebenso wie MySQL ist auch die Software von Apache Cassandra[6] für den nicht

kommerziellen Betrieb in der Standardversion kostenlos unter der GPL-Lizenz verfügbar.

Die Standardversion von Apache verfügt über keine grafische Benutzeroberfläche, wodurch die nachfolgende Installation und Konfiguration ausschließlich im Terminal von MacOS stattfindet. Der Download im Terminal von MacOS wird mit: *curl –o http://www.apache.org/dsc.tar.gz* gestartet. Im Anschluss erfolgt das Entpacken der tape archiver (tar.) Datei mit: *tar –xzf dsc-cassandra-1.2.2-bin.tar.gz*. Cassandra wird im nächsten Schritt gestartet, nachdem in den Ordner *bin* navigiert wurde. Hier wird die Datei mit Administratorrechten ausgeführt: *sudo ./cassandra –R*. Mit der Endung *–R* erhält die Datenbank Zugriff auf das gesamte System und damit auch auf alle Dateien und Einstellungen sämtlicher Benutzer.[6]

In einem neuen Terminal wird anschließend der Client gestartet, über den im weiteren Verlauf alle Konfigurationen an der Datenbank vorgenommen werden. Dieser wird

[6] www.cassandra.apache.org

ebenfalls wie der Server mit *./cqlsh* gestartet. Danach verbindet sich dieser automatisch mit dem Server, welcher auf den localhost 127.0.0.1:9042 ausgeführt wird.

```
Last login: Tue Jul 25 13:45:32 on ttys001
Denniss-MacBook-Pro:~ dennishofmann$ ls
Applications    Downloads       Music           dsc.tar.gz
Desktop         Library         Pictures        dwhelper
Documents       Movies          Public
Denniss-MacBook-Pro:~ dennishofmann$ cd Downloads/apache-cassandra-3.10\ 2/bin/
Denniss-MacBook-Pro:bin dennishofmann$ ls
cassandra               debug-cql.bat           sstableupgrade.bat
cassandra.bat           nodetool                sstableutil
cassandra.in.bat        nodetool.bat            sstableutil.bat
cassandra.in.sh         source-conf.ps1         sstableverify
cassandra.ps1           sstableloader           sstableverify.bat
cqlsh                   sstableloader.bat       stop-server
cqlsh.bat               sstablescrub            stop-server.bat
cqlsh.py                sstablescrub.bat        stop-server.ps1
debug-cql               sstableupgrade
Denniss-MacBook-Pro:bin dennishofmann$ ./cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.10 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>
```

Abbildung 32: Start des CQL-Clients

Der Datei-Import der Kriminalitätsstatistik der Stadt Chicago ist zunächst mit zahlreichen Konfigurationen verbunden, da die CSV-Datei im Gegensatz zu relationalen Datenbanken nicht einfach eingefügt werden kann. Als Erstes muss der Header der CSV-Datei geringfügig modifiziert werden, da Cassandra einen Spaltennamen bestehend aus zwei Wörtern nicht übersetzen kann und hierzu eine entsprechende Fehlermeldung zurückliefert. Mithilfe eines Unterstrichs ist das Leerzeichen im Header verschwunden, sodass der Import des Datensatzes beginnen kann.

Das Anlegen der Datenbank erfolgt in den Grundzügen wie unter MySQL. Dazu wird zunächst ein Keyspace mit *CREATE* angelegt. Des Weiteren wird als *Strategy_class* die sogenannte *`SimpleStrategy´* verwendet. Diese ist für ein zentrales Datenzentrum konzipiert, um einen deterministischen Knoten anzulegen. Alle weiteren Knoten werden anschließend im Uhrzeigersinn hinzugefügt. Darüber hinaus besteht die Option als *Strategy_class* die *`Network Topology´* zu wählen, wodurch mehrere Datenzentren verteilt installiert werden. Jedoch ist diese Funktion für den Benchmarktest irrelevant, da der Datensatz mit 5,6 Millionen Einträgen verhältnismäßig klein und eine Verteilung nicht notwendig macht.

Beim Anlegen des Keyspace wird der Replikationsfaktor bestimmt. Mit dem Befehl *stragtey_options : replication_factor = `1´* wird der Parameter gesetzt. Dies ist jedoch bei der Verteilung auf mehrere Datenzentren nicht möglich, da der Replikationsfaktor hierbei nicht global, sondern lokal bestimmt werden muss.

Mit *use crimes* wird der zuvor angelegte Keyspace aufgerufen. Im Anschluss daran wird die ColumnFamily erstellt. Dabei ist es erforderlich jeder Spalte den richtigen Datentypen zuzuordnen, um Konflikte bei späteren Abfragen zu vermeiden.

```
CREATE ColumnFamily crimes
    (ID int PRIMARY KEY,
    Case_Number varchar,
    Date varchar,
    Block varchar,
    ICUR varchar,
    Primary_Type varchar,
    Description varchar,
    Location_Description varchar,
    Arrest varchar,
    Domestic varchar,
    Beat varchar,
    District varchar,
    Ward int,
    Comunity_Area int,
    FBI_Code varchar,
    X_Coordinate float,
    Y_Coordinate float,
    year int,
    Updated_On varchar,
    Latitude float,
    Logitute float,
    Location varchar,
);
```

Nach der Definition der Datentypen wird der Pfad der CSV-Datei unter CQL angegeben:

```
copy crimes     (ID, Case_Number, Date, Block, ICUR, Primary_Type, Description,
                 Location_Description, Arrest, Domestic, Beat, District, Ward,
                 Community_Area, FBI_Code, X_Coordinate, Y_Coordinate, Year,
                Updated_On, Latitude, Longitute, Location)
```

from `Crimes_-_2001_to_present.csv
with HEADER = TRUE;

Der Import des Datensatzes benötigt lediglich 11 Minuten und 44 Sekunden. Damit wird ein deutlich besseres Ergebnis im Gegensatz zu relationalen Datenbanken mit 11 Stunden erzielt. Ebenso ist auch beim AVG-Wert eine signifikante Steigerung zu verzeichnen, die bei Cassandra 9718 rows/s und bei MySQL gerade einmal 107 rows/s beträgt.

```
cqlsh> create ColumnFamily crimes
cqlsh> use crimes
   ...
cqlsh>
cqlsh> desc keyspaces;

system_auth    crimes  dev                 test
system_schema  system  system_distributed  system_traces

cqlsh> use crimes;
cqlsh:crimes> create ColumnFamily crimes
          ... (ID int PRIMARY KEY,
          ... Case_Number varchar,
          ... Date varchar,
          ... Block varchar,
          ... ICUR varchar,
          ... Primary_Type varchar,
          ... Description varchar,
          ... Location_Description varchar,
          ... Arrest varchar,
          ... Domestic varchar,
          ... Beat varchar,
          ... District varchar,
          ... Ward int,
          ... Community_Area int,
          ... FBI_Code varchar,
          ... X_Coordinate float,
          ... Y_Coordinate float,
          ... Year int,
          ... Updated_On varchar,
          ... Latitude float,
          ... Longitute float,
          ... Location varchar
          ... ) ;
cqlsh:crimes> copy crimes (ID, Case_Number, Date, Block, ICUR, Primary_Type, Description, Location_Description, Arrest, Domestic,
e, Year, Updated_On, Latitude, Longitute, Location) from 'Crimes_-_2001_to_present.csv' with HEADER = TRUE;
Using 7 child processes

Starting copy of crimes.crimes with columns [id, case_number, date, block, icur, primary_type, description, location_description,
inate, y_coordinate, year, updated_on, latitude, longitute, location].
rocessed: 1005000 rows; Rate:   15463 rows/s; Avg. rate:    9718 rows/s
```

Abbildung 33: Import der CSV-Datei Crimes

Die Anfragesprache CQL basiert auf den Grundzügen von SQL und bietet zudem enorme Konfigurationsmöglichkeiten. Da Nutzer oftmals nur mit SQL vertraut sind, kann es bei ersten Anfragen zu Fehlermeldungen bzw. unbekannten Reaktionen der Datenbank kommen. Standardmäßig werden maximal 100 Einträge von CQL ausgegeben, ganz gleich um welche Anfrage es sich handelt. Mit der Funktion *PAGING OFF* wird die Beschränkung von 100 Einträgen außer Kraft gesetzt.

Eine weitere wichtige Funktion zur besseren Übersicht der Einträge ist *EXPAND ON*. Damit werden die Einträge der Datenbank vertikal ausgegeben, wodurch die Übersichtlichkeit erheblich verbessert wird, siehe Abbildung 34/35.

```
 id       | arrest | beat | block                          | case_number | comm
de | location                    | location_description           | longitute |
---------+--------+------+--------------------------------+-------------+-----
---+-----------------------------+--------------------------------+-----------+--
  9808094 |   true | 1223 |          023XX W MADISON ST |    HX457181 |
18 |  (41.881183104, -87.68498211) |          GROCERY FOOD STORE | -87.68498 |
  1792034 |  false | 1723 |          035XX W MONTROSE AV |     G613045 |
12 |  (41.96120509, -87.715905512) |                  SIDEWALK |  -87.7159 |
  3819940 |   true | 1822 |          012XX N LARRABEE ST |    HL187780 |
45 | (41.904493689, -87.643246928) | CHA HALLWAY/STAIRWELL/ELEVATOR | -87.64325 |
  5709208 |  false | 1434 |            025XX W MOFFAT ST |    HN510498 |
83 | (41.914824157, -87.690740714) |                 APARTMENT | -87.69074 |
  6997618 |   true | 1824 |        010XX N LAKE SHORE DR |    HR389467 |
87 | (41.900870422, -87.624184899) |             PARK PROPERTY | -87.62418 |
  2301876 |  false | 0911 |          052XX S ROCKWELL ST |    HH584718 |
45 | (41.798455397, -87.689047661) |                 RESIDENCE | -87.68905 |
  8653411 |  false | 0423 |        086XX S COMMERCIAL AVE |    HV329091 |
28 | (41.738280662, -87.551447254) |  GOVERNMENT BUILDING/PROPERTY | -87.55145 |
  3472067 |  false | 2324 |             010XX W DAKIN ST |    HK542545 |
35 | (41.953499811, -87.655192221) |                 RESIDENCE | -87.65519 |
  7397597 |   true | 0913 |           032XX S LEAVITT ST |    HS199665 |
91 | (41.834907625, -87.680301045) |                  SIDEWALK |  -87.6803 |
  6470970 |  false | 1014 |      015XX S SPRINGFIELD AVE |    HP549685 |
99 | (41.859983911, -87.722555478) |    RESIDENCE PORCH/HALLWAY | -87.72256 |
  9587017 |   true | 1522 |          002XX S LARAMIE AVE |    HX236935 |
21 |  (41.87720717, -87.754950515) |                    STREET | -87.75495 |
  1416569 |  false | 0923 |           028XX S EMERALD AV |     G136434 |
02 | (41.842017799, -87.645015951) |                 RESIDENCE | -87.64502 |
  1817764 |  false | 0124 |          002XX E RANDOLPH ST |     G649937 |
63 | (41.884631782, -87.622249231) |  COMMERCIAL / BUSINESS OFFICE | -87.62225 |
  7689629 |   true | 0532 |         119XX S MICHIGAN AVE |    HS495788 |
51 | (41.677514907, -87.620496263) |                     ALLEY |  -87.6205 |
  4459729 |  false | 1432 |           026XX N ELSTON AVE |    HL758862 |
74 | (41.929743818, -87.684273777) |          DEPARTMENT STORE | -87.68427 |
  5000310 |  false | 0511 |           100XX S RHODES AVE |    HM560028 |
38 | (41.712383472, -87.610251629) |                 RESIDENCE | -87.61025 |
  7948129 |  false | 1822 |            015XX N DAYTON ST |    HT180338 |
88 | (41.908880377, -87.649445039) |                    STREET | -87.64944 |
  3670003 |  false | 0433 |        129XX S MARQUETTE AVE |    HK769813 |
42 |  (41.66041693, -87.556847248) |                     ALLEY | -87.55685 |
  9198373 |  false | 2013 |         058XX N GLENWOOD AVE |    HW343845 |
34 | (41.987338819, -87.665062403) |  GOVERNMENT BUILDING/PROPERTY | -87.66506 |
  5474589 |   true | 2524 |            024XX N HAMLIN AVE |    HN299720 |
```

Abbildung 34: EXPAND OFF

```
@ Row 98
----------------------+--------------------------------
 id                   | 2543313
 arrest               | true
 beat                 | 2122
 block                | 004XX E 34TH ST
 case_number          | HJ124177
 community_area       | 35
 date                 | 01/13/2003 06:00:00 PM
 description          | ARMED: HANDGUN
 district             | 002
 domestic             | false
 fbi_code             | 03
 icur                 | 031A
 latitude             | 41.83314
 location             | (41.833139425, -87.615319213)
 location_description | GROCERY FOOD STORE
 longitute            | -87.61532
 primary_type         | ROBBERY
 updated_on           | 04/15/2016 08:55:02 AM
 ward                 | 4
 x_coordinate         | 1.1799e+06
 y_coordinate         | 1.8826e+06
 year                 | 2003

@ Row 99
----------------------+--------------------------------
 id                   | 2525452
 arrest               | true
 beat                 | 0321
 block                | 014XX E MARQUETTE RD
 case_number          | HJ103262
 community_area       | 42
 date                 | 01/02/2003 04:39:00 PM
 description          | SIMPLE
 district             | 003
 domestic             | false
 fbi_code             | 08A
 icur                 | 0560
 latitude             | 41.77514
 location             | (41.775138291, -87.590160894)
 location_description | OTHER
 longitute            | -87.59016
 primary_type         | ASSAULT
 updated_on           | 04/15/2016 08:55:02 AM
 ward                 | 20
```

Abbildung 35: EXPAND ON

Zur Bestimmung der Laufzeiten des späteren Benchmarktests gibt es verschiedene Optionen, die jedoch nicht alle für das Testvorhaben zielführend sind. Zwei Analysetools sind dabei genauer zu betrachten. Zum einen das Opscenter von Datastax[6], das besonders mit seiner grafischen Oberfläche punktet und vielfältige Konfigurationen am Cluster zulässt. Jedoch mit dem entscheidenden Nachteil, dass es keine Analysen einzelner Abfragen ausgeben, sondern lediglich eine Mittelwertberechnung aus den gestellten Anfragen erfolgt.

Zum anderen bietet CQL ein eigenes Tool zur Analyse der read/write Anfragen. Dieses kann unkompliziert durch die Funktion *TRACING ON* in Betrieb gesetzt werden. Bei Aktivierung des Tools erhält der Nutzer nach jeder Abfrage eine tabellarische Übersicht über die Laufzeiten von CQL.

```
Tracing session: 93587a00-778e-11e7-a2a0-ad6afef9a836

 activity                                                                                                       | timestamp                  | source    |
----------------------------------------------------------------------------------------------------------------+----------------------------+-----------+
                                                                                              Execute CQL3 query | 2017-08-02 16:26:29.920000 | 127.0.0.1 |
                                           Parsing select * from crimes LIMIT 60; [Native-Transport-Requests-1] | 2017-08-02 16:26:29.920000 | 127.0.0.1 |
                                                               Preparing statement [Native-Transport-Requests-1] | 2017-08-02 16:26:29.921000 | 127.0.0.1 |
                                                          Computing ranges to query [Native-Transport-Requests-1] | 2017-08-02 16:26:29.921000 | 127.0.0.1 |
 Submitting range requests on 257 ranges with a concurrency of 1 (29717.486 rows per range expected) [Native-Transport-Requests-1] | 2017-08-02 16:26:29.922000 | 127.0.0.1 |
                                                 Submitted 1 concurrent range requests [Native-Transport-Requests-1] | 2017-08-02 16:26:29.924000 | 127.0.0.1 |
                Executing seq scan across 5 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [ReadStage-2] | 2017-08-02 16:26:29.924000 | 127.0.0.1 |
                                                              Read 60 live and 18 tombstone cells [ReadStage-2] | 2017-08-02 16:26:29.934000 | 127.0.0.1 |
                                                                                              Request complete | 2017-08-02 16:26:29.936073 | 127.0.0.1 |
```

Abbildung 36: Auswertung der Laufzeitanalyse unter Cassandra

5.2 Testdatensatz

Benchmarktests werden mithilfe sogenannter Testdatensätze durchgeführt, die den Ansprüchen an Qualität und Quantität genügen müssen. Des Weiteren soll der Datensatz kein konstruiertes Konstrukt aus redundanten Datensätzen darstellen, sondern Werte der Realität enthalten, damit die Bewertung und Analyse der Testergebnisse objektiv betrachtet werden kann. Der Fokus liegt hierbei auf der Quantität der Daten, die weder zu gering noch zu umfangreich sein dürfen, um auf Standard Computern getestet zu werden. Zudem müssen die Ergebnisse im messbaren Bereich liegen.

Viele Open Data Portale wie das bpb (Bundesamt für politische Bildung), das GOVDATA (Datenprotal für Deutschland) und das Open Data Portal der Deutschen Bahn stellen zahlreiche Datensätze zur Verfügung, die jedoch den Anforderungen an den durchzuführenden Benchmarktest nicht gerecht werden und teilweise strengen Datenschutzbestimmungen unterliegen. Eine Ausweitung der Suche auf den amerikanischen Raum führt zur Webseite DATA.GOV. Hier findet sich eine Vielzahl an Datensätzen, die aufgrund ihrer Eigenschaften prädestiniert scheinen. Es wird sich für den Datensatz „Crimes – 2001 to present“ entschieden, der mit 5,66 Millionen Datensätzen in 22 Spalten unterteilt ist und sich für den späteren Benchmarktest gut eignet und zudem interessante Informationen über die Kriminalitätsrate der Stadt Chicago liefert. Der Datensatz wird vom Chicago Police Department täglich aktualisiert und erweitert, sodass im Testdatensatz alle Daten von 2001 bis zum 5. Juni 2017, ausgenommen Mordfälle unter Achtung der Persönlichkeitsrechte der Straftäter, vorhanden sind. Neben dem CSV-File sind auch andere Formate auf DATA.GOV wie etwa JSON oder XML verfügbar. Die URL des Chicago Police Departments[7] zeigt eine detaillierte Beschreibung der einzelnen Spalten und Zeilen des Datensatzes sowie des Datentyps. Zur Übersicht und zum besseren Verständnis des Tests wird eine Kurzform anhand der Tabelle dargestellt.[26]

Zeilenname	Beschreibung	Datentyp
ID	Primärschlüssel und eindeutige Identifikationsnummer jedes Ereignisses	Number (int)
Case Number	Einzigartige Ereignisnummer mit Bezug zum zuständigen Police Department	Text
Date	Datum und Zeit des Ereignisses	Varchar
Block	Adresse des Ereignisses	Text

[7] http://www.data.cityofchicago.org

ICUR	Illinois Uniform Crime Reporting Code. Interner Code zum Aggregieren von Ereignissen	Text
Primary Type	Kategorisierung der Ereignisse	Text
Description	Exakte Beschreibung des Ereignisses	Text
Location Description	Kategorisiert den Ort des Ereignisses	Text
Arrest	Festnahme des Täters ist erfolgt	Text
Domestic	Das Ereignis fand auf öffentlichem oder privatem Boden statt	Text
Beat	Beschreibt den zuständigen Polizeisektor sowie den zugehörigen Dienstwagen	Text
District	Zuständiger Polizeidistrikt	Text
Ward	Stadtteil des Ereignisses	Number (int)
Community Area	Definiert die Zuständigkeiten auf Gebietsgrenzen	Number (int)
FBI Code	Klassifizierung der Ereignisse anhand eines Codes innerhalb der FBI Datenbank	Text
X Coordinate	x Koordinate innerhalb der State Plane Illinois East NAD 1983 Projektion	Number (int)
Y Coordinate	y Koordinate innerhalb der State Plane Illinois East NAD 1983 Projektion	Number (int)
Year	Jahr des Ereignisses	Number (int)

Updated On	Datum und Zeit der letzten Aktualisierung des Ereignisses	Text
Latitude	Breitengrad	Number (double)
Longitute	Längengrad	Number (double)
Location	Geografische Einheit der Längen- & Breitengrade	Text

Tabelle 2: Beschreibung der einzelnen Columns des Testdatensatzes[26]

5.3 Verwendete Hardware

Benchmarktests sind ein beliebtes Mittel zum Testen von Systemen, insbesondere von Datenbanken. Ein Test ist nicht nur von der verwendeten Software sowie des Testdatensatzes abhängig, sondern ebenso von der zum Einsatz gebrachten Hardware.

Diese ist in besonderem Maße für die Ergebnisse verantwortlich. Deren Leistungsfähigkeit ist jedoch sekundär, da es zunächst entscheidend ist, dass der Test unter gleichen Bedingungen stattfindet sprich die Hardware für beide Tests eine identische Konfiguration aufweist.

Neben dem Unix-Betriebssystem MacOS Sierra, welches sich besonders für NoSQL-Datenbanken eignet, wird folgende Hardware verwendet:

- Macbook Pro Retina 15 Zoll
- Prozessor 2,4 GHz Intel Core i7
- Arbeitsspeicher 8 GB 1600 MHz DDR3
- Grafikkarte NVIDIA GeForce Gt 650M 1024 MB
- Festplatte 256 GB SSD

Die durchzuführenden Tests werden jeweils nach einem Neustart des Systems und aus dem laufenden Betrieb heraus gestartet, um so nebenläufige Prozesse zu vermeiden, die sich negativ auf die Performance des jeweiligen Datenbanksystems auswirken könnten.

5.4 Durchführung des Benchmarktests

Die Durchführung erfolgt mit vier unterschiedlichen Testszenarien. Dabei werden die Zugriffszeiten zum Abfragen von Datensätzen aus der Datenbank gemessen und anschließend grafisch aufgearbeitet. Die vier Abfragen segmentieren sich folgendermaßen:

- Abfrage aller Datensätze der Datenbank
- Abfrage auf Gleichheit mehrerer Datensätze
- Abfrage eines kleinen Bereichs an Datensätzen
- Abfrage eines großen Bereichs an Datensätzen mit oberer sowie unterer Schranke

Nach Beendigung eines jeden Tests erfolgt die Auswertung der Ergebnisse von MySQL sowie von NoSQL. Es werden jedoch keine Analysen vorgenommen, die die Divergenzen genauer beschreiben. Dies geschieht in Kapitel 5.5.

5.4.1 Test 1 Select All – Abfragen aller Daten

5.4.1.1 MySQL

Mithilfe des SELECT ALL Befehls werden alle vorhandenen Datensätze der Datenbank ausgegeben, um somit die Dauer sämtlicher 5667450 Datensätze zu errechnen. Die SQL-Anweisung zur Ausgabe wird wie folgt konstruiert:

```
SELECT *
FROM `crimes_-_2001_to_present
```

Im oberen Beispiel ist die einfachste Form der Abfrage dargestellt, da diese weder WHERE-Bedingung noch Unterabfragen verwendet und mit dem Operator * alle Datensätze der entsprechenden Tabelle selektiert. Diese Abfrage wird insgesamt zehnmal wiederholt, damit ein repräsentativer Wert entsteht. Zusätzlich wird die Abfrage mehrmals nach dem Neustart des PC´s durchgeführt, um die Ablage des Datensatzes im Hauptspeicher sowie eine Abweichung der Ergebnisse zu vermeiden. Es können jedoch keine signifikanten Unterschiede festgestellt werden, was anhand der Tabelle dokumentiert wird.

SELECT ALL MySQL		
	Dauer	Neustart des PC´s/ Datenbank
1.)	31,153 sec.	Ja
2.)	30,761 sec.	Nein
3.)	31,747 sec.	Ja
4.)	32,144 sec.	Nein
5.)	31,030 sec.	Ja
6.)	30,969 sec.	Nein
7.)	30,899 sec.	Ja
8.)	31,495 sec.	Nein
9.)	31,292 sec.	Ja
10.)	30,747 sec.	Nein
AVG.	**31,2237 sec.**	

Tabelle 3: Ergebnisse der Abfrage SELECT ALL MySQL

Das nachfolgende Diagramm visualisiert die Tabelle und zeigt die durchschnittliche Dauer zum Auslesen aller Datensätze. Nach mehrfacher Testung können nur geringe Abweichungen des Durchschnittwerts im Zehntelbereich ausgemacht werden, sodass dieser Wert als repräsentativ betrachtet werden kann.

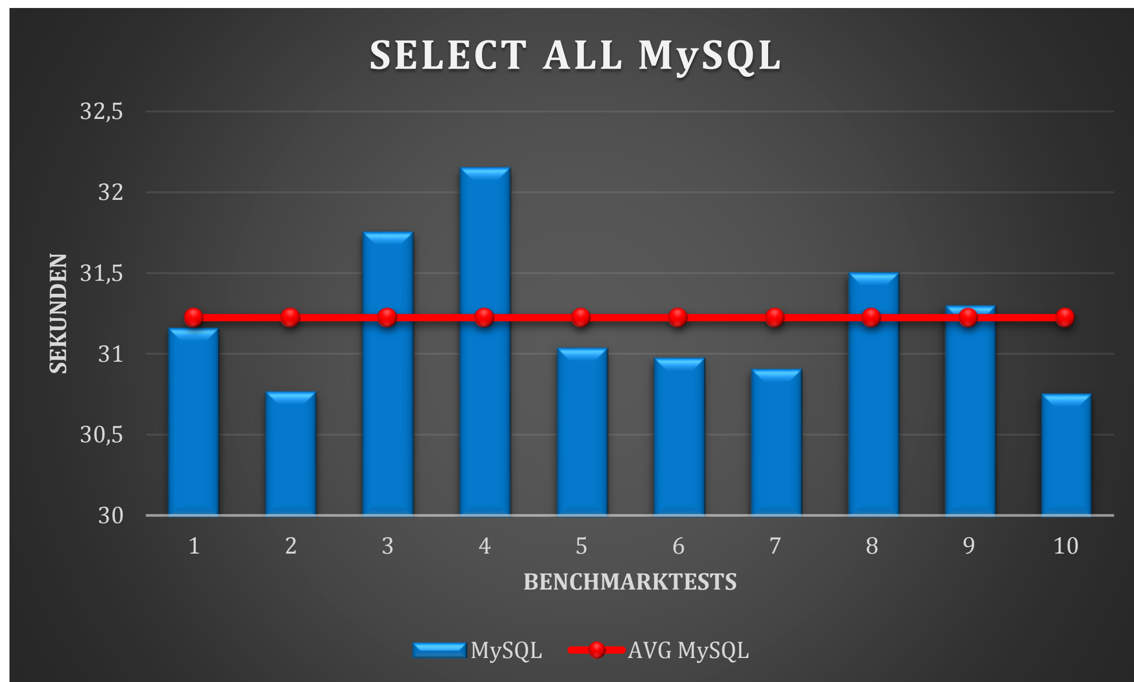


Abbildung 37: Auswertung der Abfrage SELECT ALL MySQL

5.4.1.2 NoSQL

Ebenso wird unter dem NoSQL-System Cassandra eine Abfrage über den gesamten Datensatz angelegt. Diese erfolgt unter der Abfragesprache CQL mit der SELECT-Anweisung in gleichem Maße wie bei relationalen Datenbanksystemen. Zur Auswahl aller Datensätze wird hierfür der Stern-Operator verwendet mit dessen Hilfe alle 5667450 Datensätze ausgegeben werden.

```
SELECT *
FROM crimes
LIMIT 6000000;
```

Im Gegensatz zu SQL ist es bei dieser Abfrage erforderlich als Erstes die Funktion TRACING mit dem Befehl TRACING ON zu aktivieren, damit im Anschluss an die Ausgabe eine Laufzeitanalyse seitens CQL erfolgt. Außerdem muss PAGING deaktiviert sein, da ansonsten nur maximal 100 Einträge zurückgeliefert werden. Mit der Anweisung LIMIT im CQL-Code wird die Datenausgabe auf 6 Millionen Einträge beschränkt. Dies ist geboten, da Cassandra eine interne Laufzeitanalyse vollzieht und die Ausgabe der Datensätze ansonsten abgebrochen wird.

SELECT ALL MySQL		
	1-Knoten	2-Knoten
1.)	176,59 sec.	141,96 sec.
2.)	159,48 sec.	123,21 sec.
3.)	161,35 sec.	121,68 sec.
4.)	162,59 sec.	205,51 sec.
5.)	145,93 sec.	162,37 sec.
6.)	166,28 sec.	153,81 sec.
7.)	160,55 sec.	160,33 sec.
8.)	155,73 sec.	185,83 sec.
9.)	173,76 sec.	161,41 sec.
10.)	130,86 sec.	163,73 sec.
AVG.	**159,31 sec.**	**157,98 sec.**

Tabelle 4: Ergebnisse der Abfrage SELECT ALL MySQL

Die Visualisierung im Diagramm verdeutlicht, dass die Verwendung von mehreren Knoten gegenüber einem Knoten nur minimal differiert. Eklatant ist jedoch, dass die Schwankungen deutlich größer werden je mehr Knoten zum Einsatz kommen. Trotz größerer Schwankungen sinkt der Mittelwert bei steigender Knotenanzahl etwas.

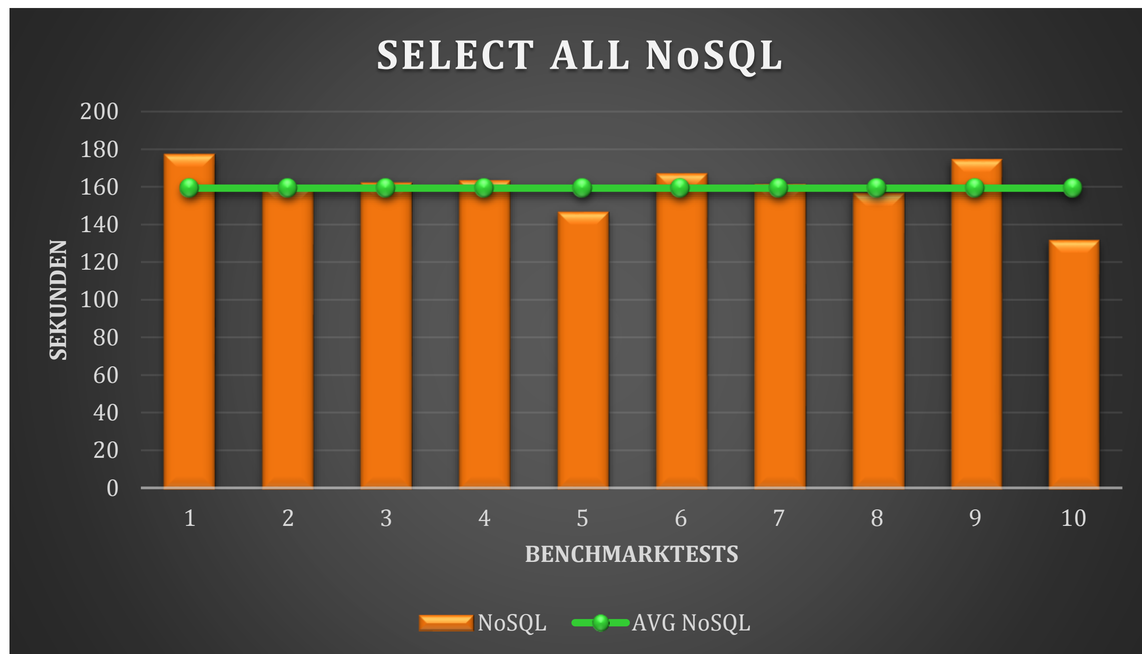


Abbildung 38: Auswertung der Abfrage SELECT ALL NoSQL

5.4.1.3 Auswertung

In Diagrammen werden die Ergebnisse der SELECT ALL Abfrage aufgearbeitet, die sowohl unter MySQL als auch unter NoSQL–Cassandra durchgeführt wurde. Es wird offenkundig, dass sich die Werte von MySQL auch nach mehrmaliger Wiederholung des Benchmarktests kaum verändern und somit eine konstante Linie bilden. Die Dauer der SELECT ALL Abfrage liegt im Bereich von 32 Sekunden. Die Laufzeiten weichen dabei nur sehr geringfügig vom Mittelwert mit gerade einmal 2-3 Sekunden ab.

Konträr dazu sind die Abfragewerte unter Cassandra um ein Vielfaches höher. Diese liegen im Bereich von 160 Sekunden für die Ausgabe des gesamten Datensatzes. Zudem sind große Schwankungen beim mehrfachen Durchführen des Tests zu verzeichnen. Dabei ergeben sich Abweichungen vom Mittelwert von minus 30 Sekunden bis plus 16 Sekunden.

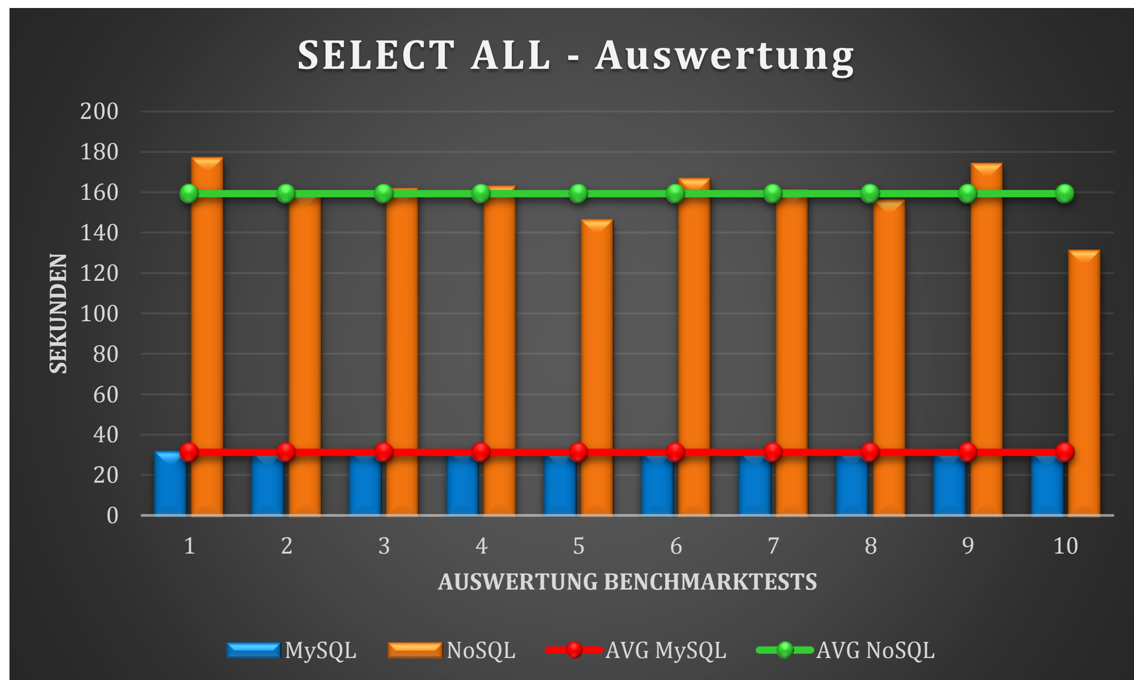


Abbildung 39: Gegenüberstellung von MySQL und NoSQL - SELECT ALL

5.4.2 Test 2 Equal – Abfrage auf Gleichheit

5.4.2.1 MySQL

Ein weiterer Benchmarktest zur Analyse der Bearbeitungszeiten wird mit der Abfrage auf Gleichheit erzeugt. Hierbei sollen alle Autodiebstähle im Jahr 2005 zurückgeliefert werden, die in Chicago registriert wurden. Zur Durchführung wird nachfolgendes SQL-Statement verwendet:

```
SET @n=0;
SELECT @n:= @n+1 AS `counter`, Date, `Primary Type`
FROM `crimes_-_2001_to_present`
WHERE Description = “AUTOMOBILE” AND year =”2005”
ORDER BY Date
```

Die SET-Anweisung dient zur Nummerierung der Ergebnisse, um stichprobenartig die korrekte Ausgabe der Datensätze zu überprüfen. Selektiert wird nach dem Datum sowie dem Primary Type (Kurzbeschreibung des Ergebnisses). In der WHERE-Klausel werden anschließend alle Datensätze ausgewählt, in deren Beschreibung sowohl AUTOMOBILE als auch das Jahr 2005 zu finden sind. Mit ORDER BY Date wird eine chronologisch nach

Datum sortierte Liste erzeugt und die Datensätze entsprechend geordnet. Dabei werden 16407 Datensätze als Rückgabewert generiert.

Abfrage auf Gleichheit MySQL		
	Dauer	Neustart des PC´s/ Datenbank
1.)	6,820 sec.	Ja
2.)	6,659 sec.	Nein
3.)	6,614 sec.	Ja
4.)	6,685 sec.	Nein
5.)	6,769 sec.	Ja
6.)	6,662 sec.	Nein
7.)	6,716 sec.	Ja
8.)	6,725 sec.	Nein
9.)	6,710 sec.	Ja
10.)	6,587 sec.	Nein
AVG.	**6,6947 sec.**	

Tabelle 5: Ergebnisse der Abfrage auf Gleichheit MySQL

Nachfolgendes Diagramm visualisiert die obige Tabelle und zeigt zudem den AVG-Wert. Auch hier ist keine Abweichung zwischen dem Neustart des PC´s/ Datenbank und der Abfrage aus der Datenbank zu identifizieren.

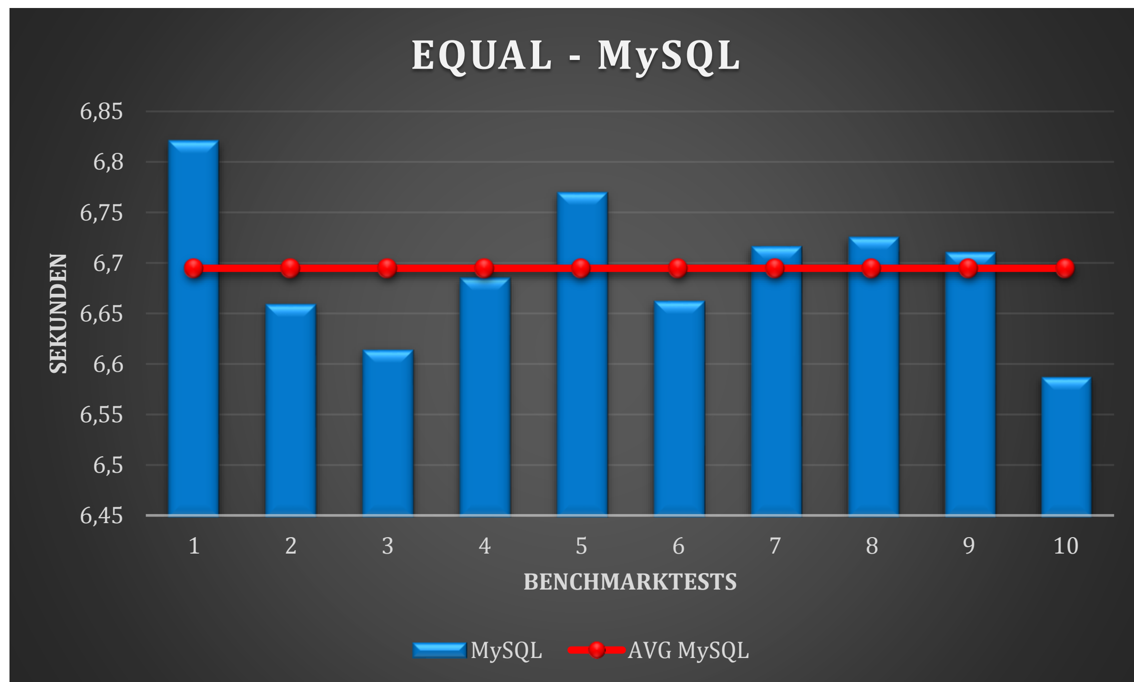


Abbildung 40: Auswertung der Abfrage auf Gleichheit MySQL

5.4.2.2 NoSQL

Die Abfrage aller Autodiebstähle im Jahr 2005 wird ebenso mithilfe der NoSQL-Datenbank Cassandra getestet. Dabei bildet die WHERE-Bedingung in CQL eine Differenzierung zur SQL-Abfrage.

```
SELECT Date, Primary_Type
FROM crimes
WHERE Description = `AUTOMOBILE´ AND year = `2005´
ORDER BY Date
LIMIT 200000
ALLOW FILTERING;
```

Ein besonderes Augenmerk liegt auf der Funktion ALLOW FILTERING, die zur Durchführung der Abfrage erforderlich ist, da ansonsten Cassandra folgende Fehlermeldung zurückgibt:

„Bad Request: Cannot execute this query as it might involve data filtering an this may haven unpredictable performance"

Damit wird ineffizienten Abfragen vorgebeugt. Es wird beispielsweise ausgeschlossen, dass Cassandra sämtliche 5,6 Millionen Datensätze vergeblich durchsucht, um lediglich einen passenden Datensatz zu finden.

Abfrage auf Gleichheit NoSQL		
	1-Knoten	2-Knoten
1.)	36,27 sec.	29,51 sec.
2.)	29,17 sec.	37,20 sec.
3.)	33,54 sec.	35,87 sec.
4.)	39,87 sec.	41,14 sec.
5.)	35,42 sec.	28,01 sec.
6.)	33,41 sec.	27,53 sec.
7.)	37,27 sec.	36,83 sec.
8.)	34,17 sec.	42,29 sec.
9.)	30,54 sec.	31,47 sec.
10.)	34,76 sec.	32,84 sec.
AVG.	**34,44 sec.**	**34,26 sec.**

Tabelle 6: Ergebnisse der Abfrage auf Gleichheit NoSQL

Das unten aufgeführte Diagramm demonstriert, dass die Verwendung von 2 Knoten zu größeren Schwankungen in der Laufzeit führt. Zudem wird dargelegt, dass sich auch hier bei steigender Knotenanzahl der Mittelwert geringfügig reduziert.

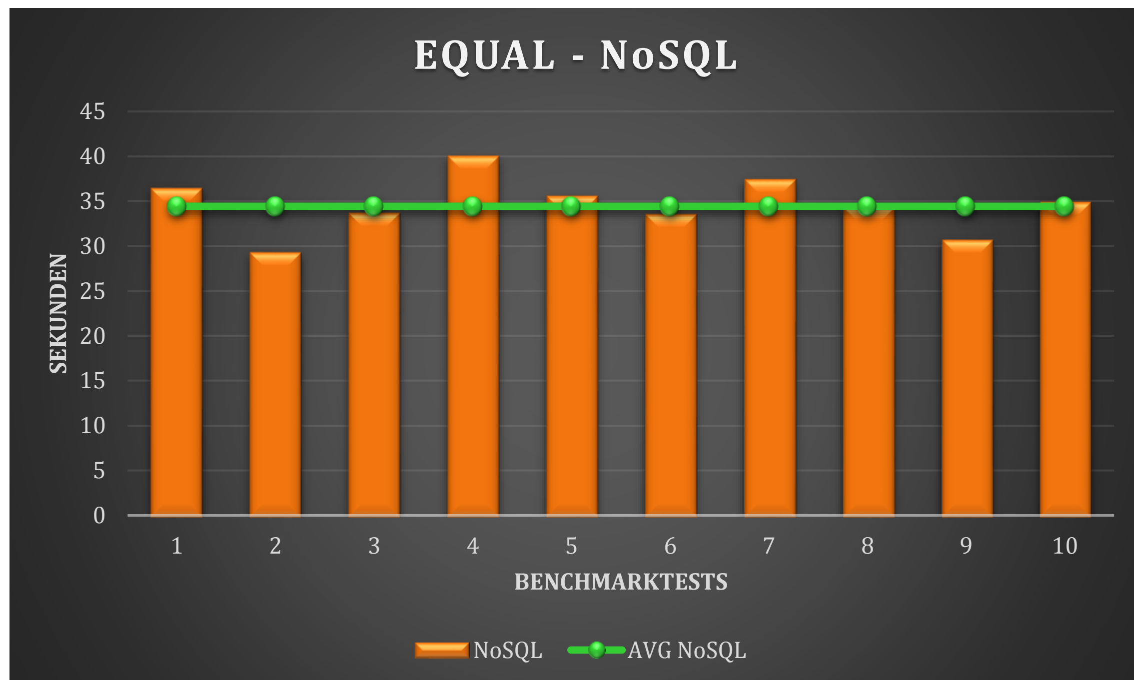


Abbildung 41: Auswertung der Abfrage auf Gleichheit NoSQL

5.4.2.3 Auswertung

Die folgende Darstellung veranschaulicht die Abfrage auf Gleichheit. Betrachtet man die Laufzeit unter MySQL, so beträgt diese im Durchschnitt 6,6 Sekunden. Die Schwankungen liegen dabei im Zehntel und in manchen Fällen sogar im Hundertstel-Sekundenbereich. Der Neustart des PC´s hat indes keine eklatanten Auswirkungen auf den Test.

Die NoSQL-Datenbank Cassandra weist auch beim zweiten Benchmarktest deutlich höhere Laufzeiten auf, wie auch schon im vorangegangenen Test. Dabei sind die Werte um das Fünffache größer, sodass die identische Abfrage im Durchschnitt 34,4 Sekunden benötigt. Der Durchschnittswert sinkt bei steigender Knotenanzahl leicht, jedoch steigen die Schwankungen der einzelnen Tests deutlich an.

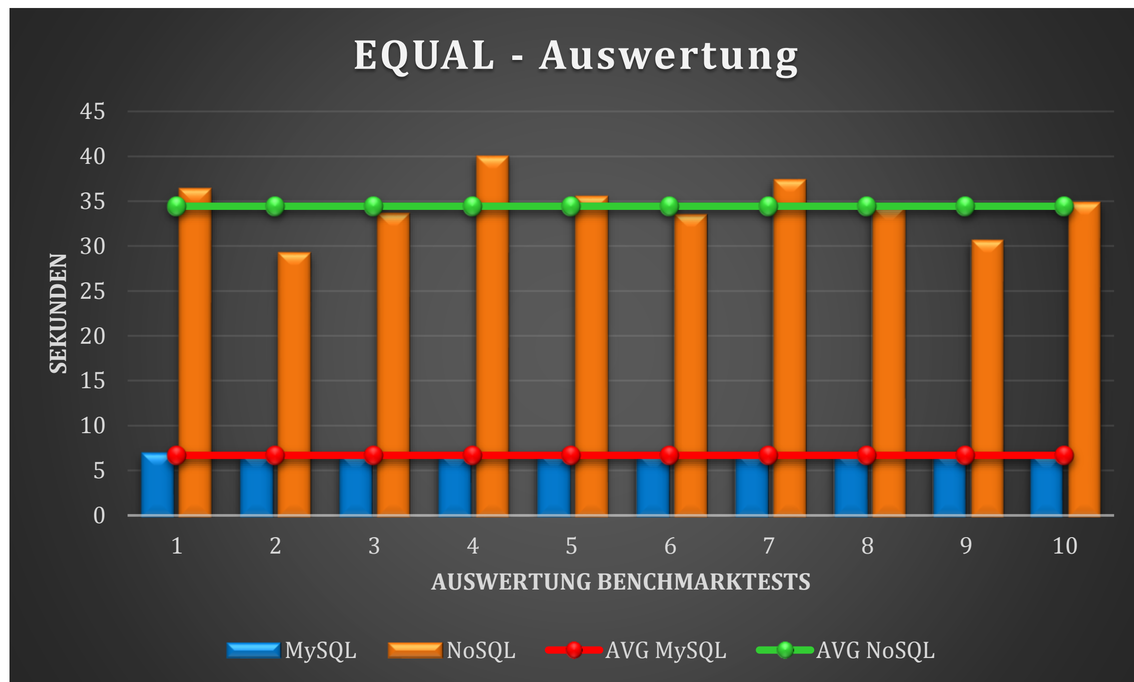


Abbildung 42: Gegenüberstellung von MySQL und NoSQL - Equal

5.4.3 Test 3 Small-Range – Kleinere Bereichsabfrage

5.4.3.1 MySQL

In dieser Abfrage wird ein sehr kleiner Bereich an Daten abgefragt, der jedoch nicht zwangsläufig weniger Zeit in Anspruch nimmt als größere Bereichsabfragen. Die SQL-Anfrage muss bei sämtlichen Bereichsabfragen ein Mal den kompletten Datensatz sichten, um ein Ergebnis zurückzuliefern. Die folgende SQL-Anfrage wird zum Test der Small Range verwendet:

```
SET @n=0;
SELECT @n:=@n+1 AS 'Ereignis', Date, Description
FROM `crimes_-_2001_to_present`
WHERE District >=30
```

Die Selektion erfolgt durch die Nummerierung, die Datumsangabe sowie dem entsprechenden Ereignis. Die WHERE-Bedingung besagt, dass alle Kriminalfälle in den Polizeidistrikten größer gleich 30 ausgegeben und 121 Datensätze erhalten werden. Aufgrund der Aufteilung der Polizeidistrikte scheint ein sehr gering ausfallender Datensatz mit 121 Einträgen als plausibel, da die drei Distrikte im Norden der Stadt zu finden sind, wo es kaum Gangs bzw. sonstige ethnische Gruppierungen gibt. Der Test wird zudem zur

Überprüfung der Kriminalitätsstatistiken sowie der Berichterstattung mit den Polizeidirektionen im Süden der Stadt erneut ausgeführt. Dabei werden in dem vorherrschenden Viertel der Gangs und Latinos über 2689400 Einträge zurückgeliefert.

Small Range MySQL		
	Dauer	Neustart des PC´s/ Datenbank
1.)	6,936 sec.	Ja
2.)	6,939 sec.	Nein
3.)	6,946 sec.	Ja
4.)	6,747 sec.	Nein
5.)	7,129 sec.	Ja
6.)	6,997 sec.	Nein
7.)	6,843 sec.	Ja
8.)	6,954 sec.	Nein
9.)	6,798 sec.	Ja
10.)	6,901 sec.	Nein
AVG.	**6,919 sec.**	

Tabelle 7: Ergebnisse der Small Range Abfrage MySQL

Das untere Diagramm dokumentiert, dass die Bearbeitungszeiten der Datenbank sehr nahe zusammenliegen und bis auf wenige Ausreißer kaum zu unterscheiden sind. Die Ausreißer, welche hier im Zehntel-Sekunden Bereich liegen, können auch auf nebenläufige Prozesse bzw. Interrupts des PC´s hindeuten, die nie ganz auszuschließen sind.

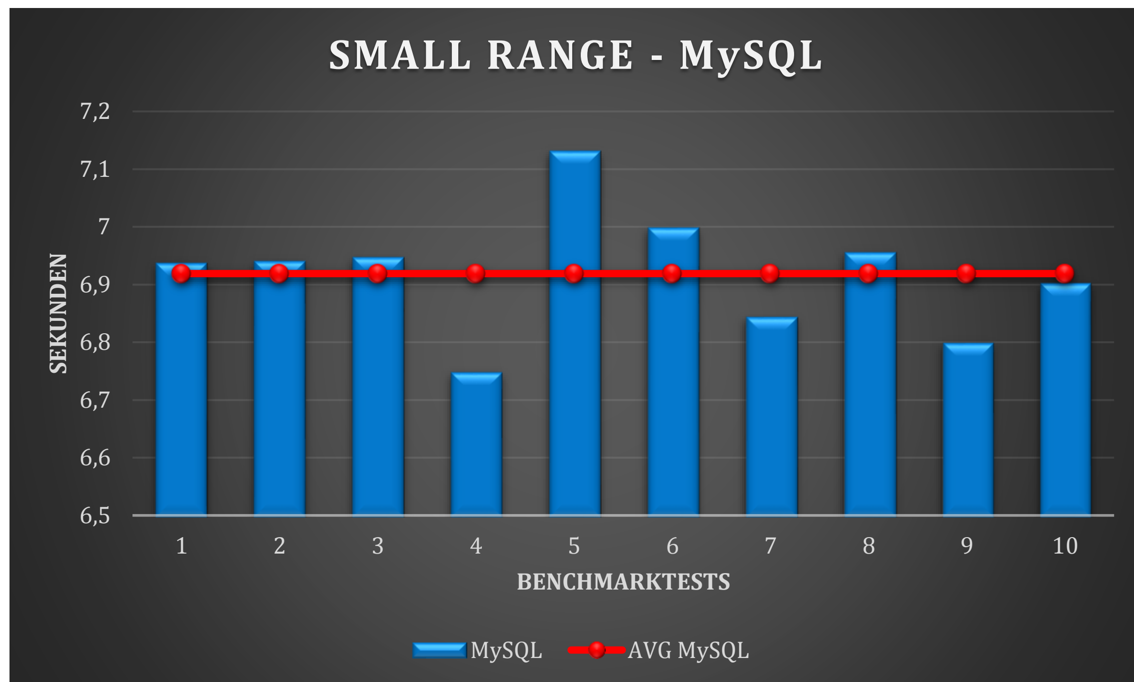


Abbildung 43: Auswertung der Small Range Abfrage MySQL

5.4.3.2 NoSQL

Der nachfolgende Benchmarktest formuliert die Small Range Abfrage an den Datensatz, die lediglich einen sehr kleinen Bereich an Daten unter Verwendung der CQL-Anfrage zurückliefern soll:

```
SELECT Date, Description
FROM crimes
WHERE District >= `30´
LIMIT 1000
ALLOW FILTERING;
```

Die Small Range Abfrage wird ebenfalls mit den bereits bekannten Funktionen LIMIT und ALLOW FILTERING durchgeführt. Des Weiteren finden mehrere Tests mit der Funktion EXPAND ON statt, um zu eruieren, ob diese Funktion sich in irgendeiner Weise auf die Laufzeiten auswirkt. Es werden jedoch ebenso wenig Unterschiede ausgemacht wie bei der Abfrage SELECT ALL.

Small Range NoSQL		
	1-Knoten	2-Knoten
1.)	20,13 sec.	21,23 sec.
2.)	24,57 sec.	25,76 sec.
3.)	19,12 sec.	20,33 sec.
4.)	21,31 sec.	18,75 sec.
5.)	20,98 sec.	19,26 sec.
6.)	19,76 sec.	17,37 sec.
7.)	23,54 sec.	21,20 sec.
8.)	22,71 sec.	19,77 sec.
9.)	20,91 sec.	20,23 sec.
10.)	21,20 sec.	16,71 sec.
AVG.	**21,42 sec.**	**20,06 sec.**

Tabelle 8: Ergebnisse der Small Range Abfrage NoSQL

Abbildung 43 visualisiert die Ergebnisse der Small Range Abfrage. Auffallend sind hierbei die deutlichen Unterschiede zwischen der Anzahl der Knoten. Die Laufzeit verkürzt sich beim Einsatz von 2 Knoten um etwas mehr als 1 Sekunde, was in Anbetracht der relativ kurzen Laufzeit als signifikanter Unterschied erwähnenswert ist.

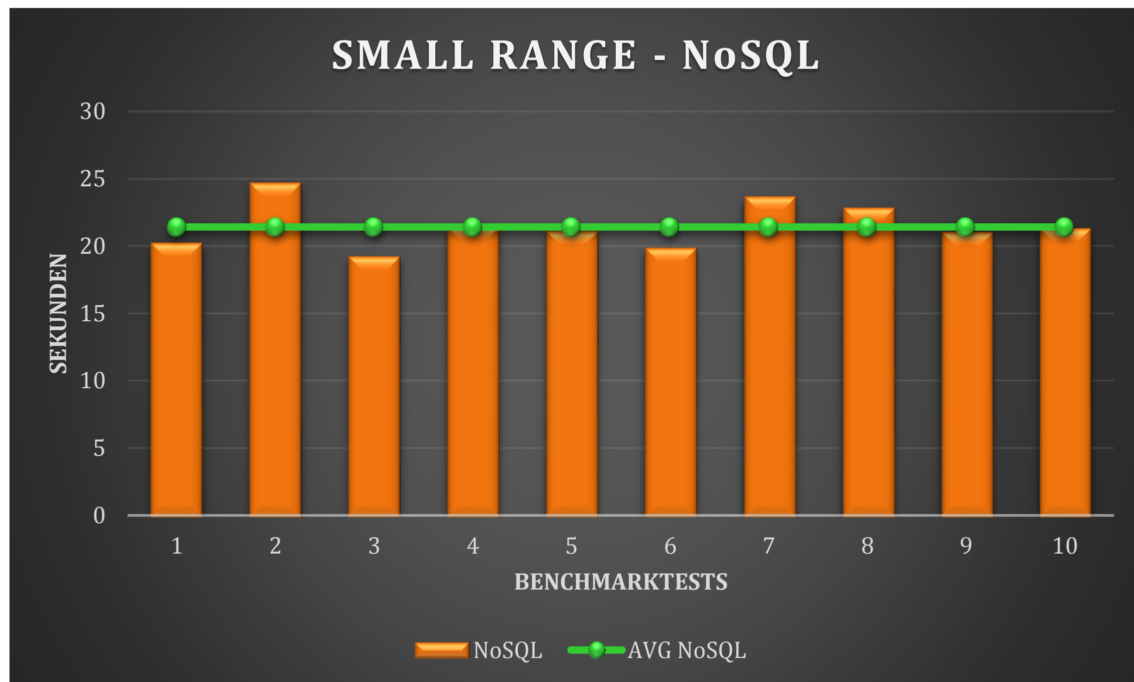


Abbildung 44: Auswertung der Small Range Abfrage

5.4.3.3 Auswertung

Bei der kleinen Bereichsabfrage werden alle Distrikte in Chicago gesucht, die größer gleich 30 sind. Hier können erste Annäherungen der Laufzeiten beider Systeme ausgemacht werden. MySQL bedarf dafür im Schnitt 6,9 Sekunden und zeigt kaum messbare Abweichungen dieses Mittelwerts. Apache Cassandra benötigt im Schnitt 21,4 Sekunden und liegt damit zum ersten Mal nur um das dreifache höher als eine identische Abfrage unter MySQL. Dabei ist deutlich zu sehen, dass bei steigender Knotenanzahl die Laufzeit geringfügig sinkt. Die Abweichungen um den Mittelwert nehmen jedoch zu.

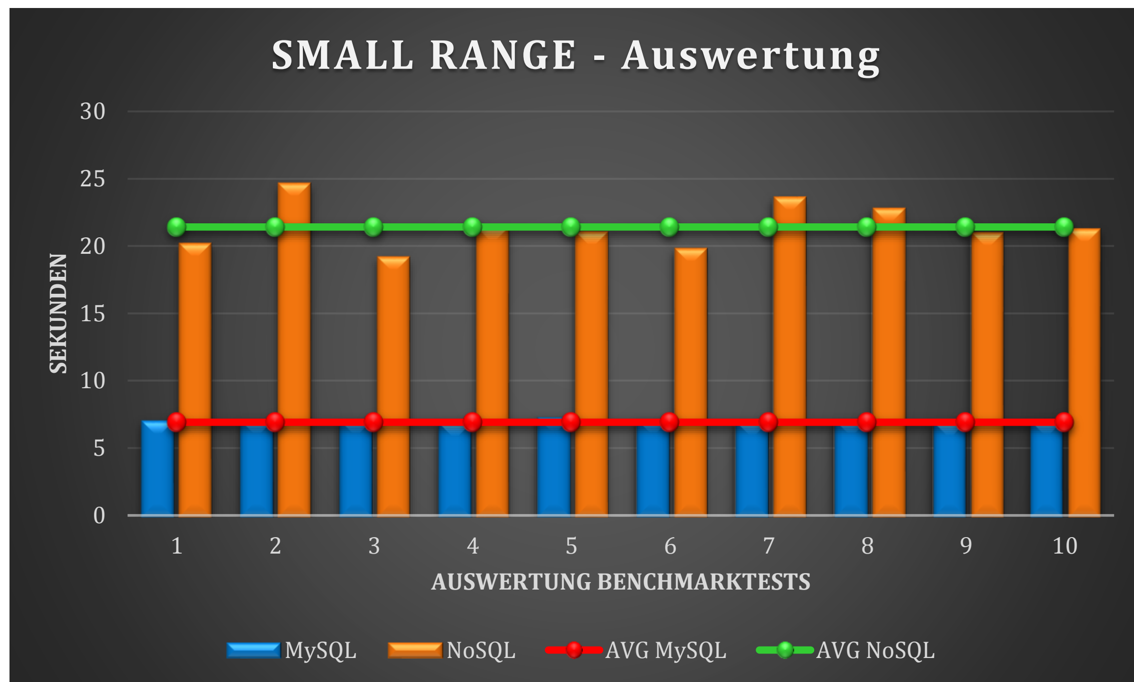


Abbildung 45: Gegenüberstellung von MySQL und NoSQL – Small Range

5.4.4 Test 4 Large-Range – Größere Bereichsabfrage

5.4.4.1 MySQL

Die SQL-Abfrage bezieht sich gleichermaßen auf einen großen Bereich des Datensatzes. Hierbei wird eruiert, dass das Verhältnis zwischen Datensätzen sowie kleiner und großer Bereichsabfragen nicht proportional ansteigt. Mit fortlaufender SQL-Abfrage über die Kriminalitätsstatistik von Chicago wird der Benchmarktest vollzogen.

```
SET @n=0;
SELECT @n:=@n +1 AS ‘Ereignis’, Date, Description, ‘Location Description’, Arrest
FROM `crimes_-_2001_to_present`
WHERE Year >= ‘2005’ AND Year <= ‘2014’
```

Die Bereichsabfrage gibt neben der Nummerierung der Datensätze: Datum, Description, Location Description sowie Arrest aus. Dabei wird der Datensatz auf die Jahre von 2005 bis 2014 limitiert, sodass ein möglichst großer Bereich ausgegeben wird und nicht der komplette Datensatz von 2001 bis 2017.

LARGE RANGE MySQL		
	Dauer	Neustart des PC´s/ Datenbank
1.)	10,639 sec.	Ja
2.)	10,623 sec.	Nein
3.)	10,622 sec.	Ja
4.)	10,670 sec.	Nein
5.)	10,695 sec.	Ja
6.)	10,742 sec.	Nein
7.)	10,725 sec.	Ja
8.)	10,643 sec.	Nein
9.)	10,734 sec.	Ja
10.)	10,726 sec.	Nein
AVG.	**10,6819 sec.**	

Tabelle 9: Ergebnisse der Large Range Abfrage MySQL

Die Visualisierung im Diagramm veranschaulicht die Bearbeitungszeit zum Zurückliefern der 3764139 Datensätze.

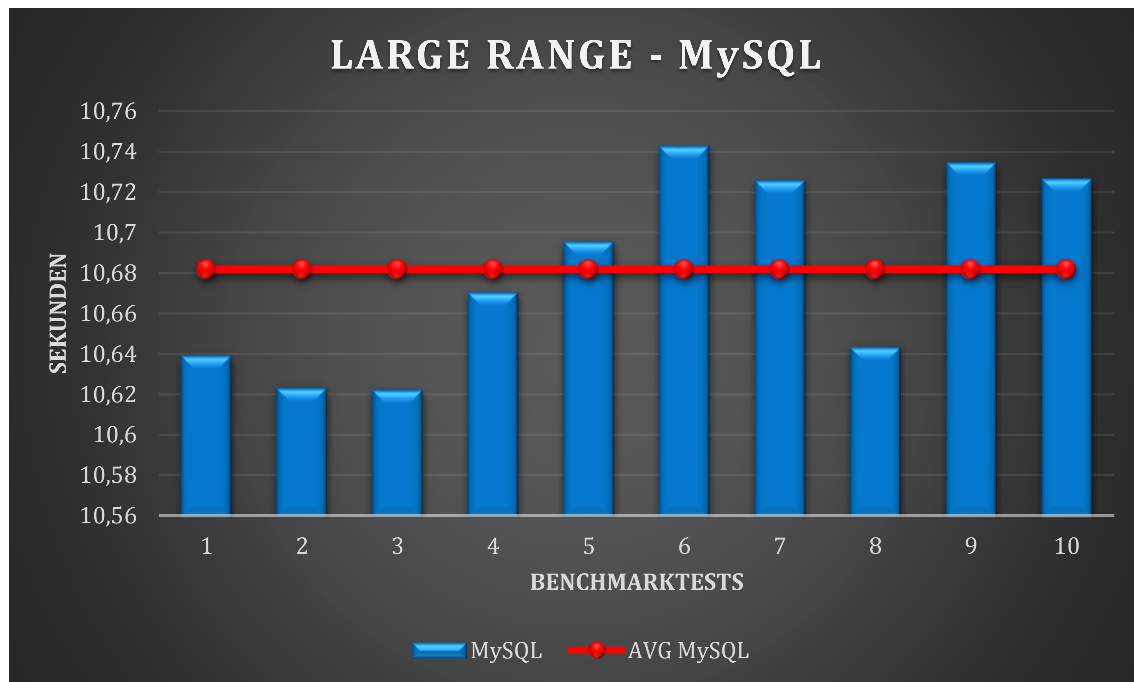


Abbildung 46: Auswertung der Large Range Abfrage MySQL

5.4.4.2 NoSQL

Der letzte Test unter CQL indiziert die Laufzeiten, die sich für Cassandra bei einer großen Bereichsabfrage ergeben. Es wird hierbei eine untere wie auch eine obere Schranke definiert. Die große Bereichsabfrage erfolgt mit der CQL-Anweisung:

```
SELECT Date, Description, Location_Description, Arrest
FROM crimes
WHERE year >= `2005´AND year <= `2014´
LIMIT 4000000
ALLOW FILTERING;
```

Diese Anfrage liefert ebenfalls, die zuvor unter MySQL errechneten 3764139 Datensätze zurück. Zwischen der kleinen und großen Bereichsabfrage sind keine signifikanten Unterschiede auszumachen. Eine exponentielle Steigerung im Verhältnis zu den Datensätzen ist nicht erkennbar.

LARGE RANGE NoSQL		
	1-Knoten	2-Knoten
1.)	25,31 sec.	23,76 sec.
2.)	27,20 sec.	25,80 sec.
3.)	24,79 sec.	28,13 sec.
4.)	24,88 sec.	21,45 sec.
5.)	26,13 sec.	23,17 sec.
6.)	25,97 sec.	24,01 sec.
7.)	23,54 sec.	22,98 sec.
8.)	24,81 sec.	23,56 sec.
9.)	25,33 sec.	22,71 sec.
10.)	26,70 sec.	23,05 sec.
AVG.	**25,47 sec.**	**23,86 sec.**

Tabelle 10: Ergebnisse der Large Range Abfrage NoSQL

Die große Bereichsabfrage zeigt ebenso wie auch die anderen Benchmarktests, dass unter Verwendung eines zweiten Knotens die Streuung der Einzelwerte zunimmt, jedoch der Mittelwert sinkt. Anhand des Diagramms werden die Werte der oberen Tabelle veranschaulicht sowie die Steigerung des Mittelwerts unter Zuhilfenahme eines zweiten Knotens verdeutlicht.

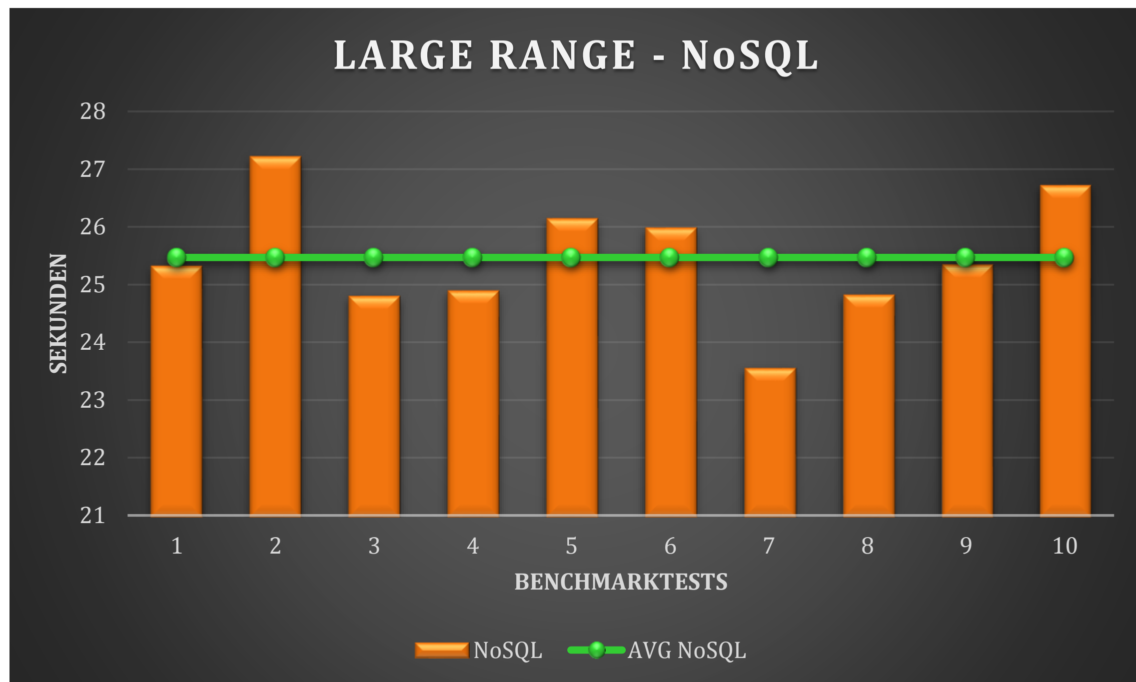


Abbildung 47: Auswertung der Large Range Abfrage NoSQL

5.4.4.3 Auswertung

Die große Bereichsabfrage bildet abschließend den letzten Benchmarktest. Hierbei wird ein Datensatz ausgegeben, der durch eine obere sowie eine untere Schranke begrenzt ist. Interessant ist, dass erstmals in dieser Testreihe NoSQL etwas mehr als doppelt so lange benötigt wie das MySQL-System. Der Durchschnitt liegt unter Cassandra bei 25,4 Sekunden und MySQL bei 10,6 Sekunden.

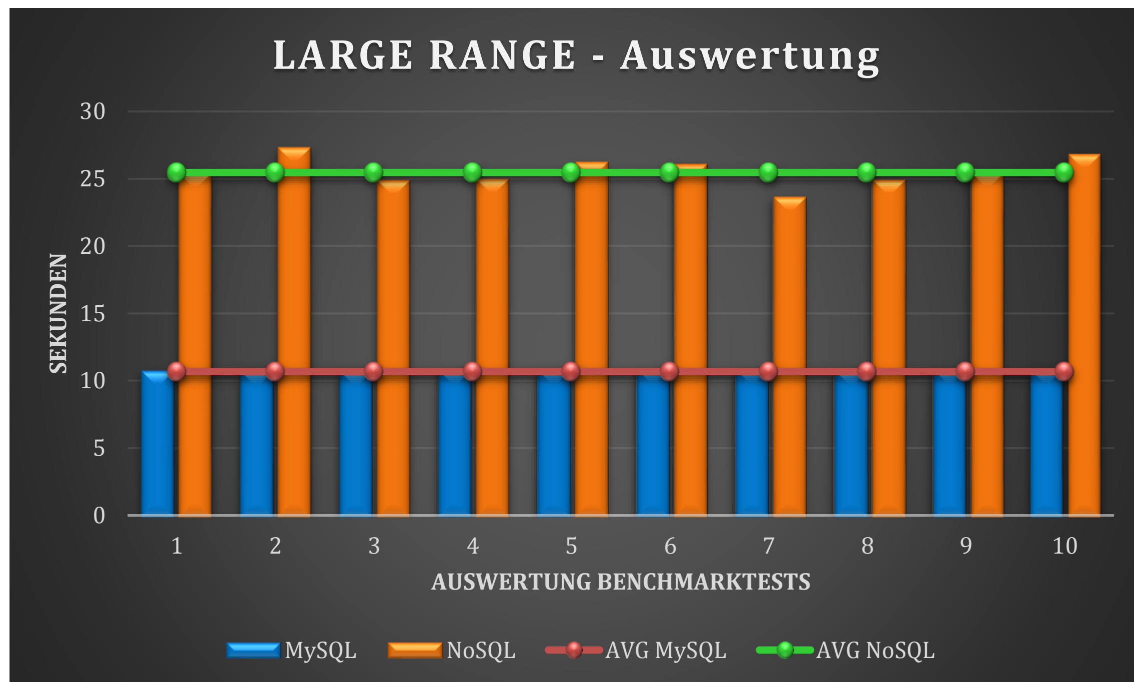


Abbildung 48: Gegenüberstellung von MySQL und NoSQL – Large Range

5.5 Analyse und Bewertung der Ergebnisse

In den vorherigen Kapiteln sind verschiedene Benchmarktests zu den beiden Systemen MySQL sowie Apache Cassandra durchgeführt worden. Dabei sind die Laufzeitunterschiede beider Datenbanksysteme bei gleicher Anfrage recht different ausgefallen. Auf Grundlage der vorangegangenen Theorie in den Kapiteln eins und zwei werden nachfolgend die Ergebnisse eingeordnet..

Prinzipiell werden zwei Hauptfaktoren für die unterschiedlichen Laufzeiten ausgemacht. Zum einen die Datenhaltung im Datenbanksystem und zum anderen das Reduktionsverfahren, welches die gesamte Datenmenge auf den gewünschten Umfang reduziert.

Unter MySQL erfolgt die Datenhaltung in zweidimensionalen Tabellen, wobei jeder Datensatz exakt eine Zeile in der Tabelle einnimmt. Dabei ist unerheblich, dass jedes Tupel innerhalb der Tabelle auch dieselbe Anzahl an Attributen beinhaltet. Existiert ein Datensatz mit weniger Attributen als andere, wird hierzu dennoch das Attribut mit einem NULL-Wert angelegt. Das kann dazu führen, dass Speicherplatz allokiert wird und somit bei Anfragen jedes Mal ausgewertet werden muss, wodurch sich die Laufzeit maßgeblich erhöht.

Die Datenhaltung unter Apache Cassandra unterscheidet sich grundlegend von dem relationalen Ansatz. Zwar existieren die Begriffe: Zeile (Row) und Spalte (Column) ebenso, dennoch weisen sie andere Eigenschaften auf. Jede Zeile wird hierbei anhand ei-

nes Schlüssels identifiziert und kann eine beliebige Anzahl an Spalten beinhalten. Der Datensatz wird im Gegensatz zu relationalen Datenbanken nicht standardmäßig auf einem Heap gespeichert, sondern anhand des Hash-Werts in der Datenbank sortiert abgelegt, um das spätere Wiederfinden im verteilten System problemlos zu ermöglichen. Die unterschiedliche Datenhaltung von MySQL und Cassandra indiziert schon zu Beginn des Benchmarktests große Leistungsunterschiede. Nachfolgend werden die jeweils durchgeführten Tests gründlich analysiert.

Erster Test: Alle Daten, die im Datenbanksystem enthalten sind, sollen ausgegeben werden. Durch die Datenhaltung im Heap unter MySQL bedarf es im Mittel 31,2 Sekunden zur Ausgabe aller Datensätze, während das NoSQL-System Cassandra im Mittel 159,3 Sekunden benötigt. MySQL nutzt bei diesem Test die Grundoperation der Projektion bei der alle Spalten selektiert und ausgegeben werden. Cassandra hingegen kann die Vorteile, die aus dem Sortieren des Datensatzes resultieren, nicht zum Einsatz bringen. Zudem startet Cassandra stets mit dem Reduktionsverfahren Map/Reduce, sodass aufgrund dessen Komplexität zusätzliche Performanceeinbußen zu verzeichnen sind. Da in diesem Test keine Reduktion der Daten stattfindet, wird unnötige Rechenlast beansprucht, die sich auf die Laufzeit der Anfrage deutlich auswirkt.

Zweiter Test: Die Abfrage des Datensatzes erfolgt auf Gleichheit.

Hierbei werden alle Einträge ausgegeben, bei denen es sich um Autodiebstähle handelt, die im Jahr 2005 in Chicago verübt wurden. MySQL verwendet zur Ausgabe der geforderten Datensätze eine weitere Grundoperation der relationalen Algebra. Mithilfe der Intersection (Schnittmenge) werden alle Tupel zurückgeliefert, die sowohl Autodiebstähle als auch das Jahr 2005 beinhalten. Grundsätzlich werden hierfür zwei Relationen zur Bildung der Schnittmenge in Anspruch genommen. Da der verwendetete Datensatz jedoch nur aus einer Relation besteht, kopiert MySQL diese Relation intern und bildet anschließend die Schnittmenge. Dies geschieht im Mittel in 6,69 Sekunden.

Cassandra verfolgt wiederum denselben Ansatz wie bei Test eins. Dabei wird die Gesamtdatenmenge mithilfe der Map/Reduce-Funktion reduziert, sodass das gewünscht Ergebnis ausgegeben wird. Die Map/Reduce-Funktion erhält zunächst den gesamten Datensatz, der in der Map-Phase in verschiedene gleich große Blöcke aufteilt wird. Die Daten werden in der Map-Phase zudem mit einem Zähler versehen. Wird das Element „AUTOMOBILE" sowie die Jahreszahl „2005" erkannt, wird dieses Element mit 1 (true) kenntlich gemacht. Jedes andere Element im Datensatz erhält in dieser Phase den Wert 0 (false). In der nachfolgenden Reduce-Phase wird somit der Datensatz auf die Elemente reduziert, die zuvor mit „true" gekennzeichnet wurden. Im Anschluss erfolgt die Zusammenfassung des Ergebnisses und die Ausgabe an den Nutzer. Da auch hier das vorherige Sortieren des Datensatzes irrelevant ist, wird eine Antwort im Durchschnitt nach 34,44 Sekunden gegeben.

Dritter Test: Ein kleiner Bereich des Datensatzes wird abgefragt.

Unter MySQL geschieht dies mit der sogenannten Restriktion. Dabei werden alle Zeilen selektiert, die in der Spalte „District“ einen Wert größer gleich 30 besitzen. Dies erfolgt innerhalb von 6,9 Sekunden. Cassandra benötigt bei gleicher Anfrage hingegen 21,4 Sekunden, um die Datensätze zurückzuliefern. Die sortierte Datenhaltung innerhalb des NoSQL-Systems kann auch bei Test drei keinen entscheidenden Vorteil gegenüber den relationalen Datenbanken liefern.

Vierter Test: Die große Bereichsabfrage soll alle Verbrechen in den Jahren 2005 bis 2014 zurückliefern. Zudem sollen die Informationen: Date, Description, Location_Description sowie Arrest ausgegeben werden. MySQL verfährt hierbei nach einem weiteren Operator der relationalen Algebra. Mithilfe der Differenz kann das relationale System alle Einträge der Jahre 2005-2014 zurückgeben. Dies erfolgt nach 10,68 Sekunden, wobei es bedeutungslos ist, ob die Daten sortiert vorliegen oder die Speicherstruktur des Heaps genutzt wird. Das NoSQL-System arbeitet hingegen nach dem bereits bekannten Map/Reduce-Schema. Dabei wird der gesamte Datensatz in der Map-Phase in gleich große Blöcke unterteilt und in Schlüssel-Wert-Paare gespeichert. Die Reduce-Phase reduziert anschließend den Datensatz und gibt alle Werte zurück, die im Bereich der unteren „2005“ und oberen „2014“ Schranke liegen. Der errechnete Mittelwert liegt auch hier mit 25,4 Sekunden deutlich über dem von MySQL.

Die Vorzüge von Cassandra können jedoch bei diesem Benchmarktest nicht ihre volle Effektivität erzielen. Grundsätzlich ist Cassandra für große bis sehr große Datensätze konzipiert. Der hier zum Einsatz gebrachte Datensatz mit 5,6 Millionen Datensätzen ist nicht ausreichend, um deren vollständige Stärke zu repräsentieren. Dennoch kann mit der Verwendung eines zweiten Knotens gezeigt werden, dass sich die Antwortzeit etwas verkürzt. Bei Milliarden von Datensätzen und unzähligen Knoten ist Cassandra dem relationalen System deutlich überlegen. Dies konnte im Benchmarktest jedoch aufgrund fehlender Hardware sowie der begrenzten Bearbeitungszeit der Masterarbeit nur ansatzweise dargestellt werden. Trotz allem decken sich die Ergebnisse mit ähnlichen Benchmarktests, die bereits in anderen wissenschaftlichen Projekten zu den beiden Systemen durchgeführt wurden.

5.6 Probleme und Messfehler

Wissenschaftliche Arbeiten, die praktische Bezüge zwischen verschiedenen Systemen in Form von Benchmarktests herstellen, müssen sich sehr häufig mit unterschiedlichsten Problemen wie auch Messfehlern auseinandersetzen.

Gleichermaßen sind auch in dieser Arbeit einige Komplikationen aufgetreten, die zu Beginn der Testphase gelöst werden mussten, um die Vergleichbarkeit beider Tests zu gewährleisten.

Der Datensatz zur Kriminalitätsstatistik der Stadt Chicago verursachte erste Schwierigkeiten beim Import in die beiden Datenbanken. Unter MySQL wurde der Import des Datensatzes jedes Mal nach exakt 23 Minuten abgebrochen und lieferte die Fehlermeldung: „ It is not allow to import this data set, Error 53“. Diese Fehlermeldung war wenig aufschlussreich, da sie allgemeingültig formuliert ist und keinerlei Rückschlüsse auf das eigentliche Problem zuließ. Nach einiger Zeit konnte festgestellt werden, dass MySQL als Trennzeichen einer CSV-Datei ein Semikolon erwartet, woraus diese Fehlermeldung resultierte. Bei der Konfiguration wurde das falsche Trennzeichen verwendet, sodass das Komma ersetzt wurde und der Import erfolgen konnte. Ebenso kam es zu Schwierigkeiten beim Import des Datensatzes unter Apache Cassandra. Der Header des Datensatzes enthielt zahlreiche Bezeichnungen, die sich aus zwei Wörtern zusammensetzten und durch ein Leerzeichen voneinander getrennt waren. Beim ersten Einfügen des Datensatzes importierte Cassandra jedes Wort des Headers als separate Spalte, wodurch der Datensatz verfälscht wurde. Zur Gewährleistung des korrekten Imports musste eine Datenmanipulation durchgeführt werden. Dazu wurden alle Bezeichnungen, die sich aus zwei Wörtern zusammensetzten mithilfe eines Unterstrichs verbunden. Die dadurch entstandene Manipulation am Datensatz war jedoch so geringfügig, sodass diese auf die Laufzeiten des späteren Benchmarktests keine Auswirkungen hatte.

Des Weiteren stellte der Datentyp „Date“ unter Cassandra beim verwendeten Datensatz ein Problem dar. In der CSV-Datei wurde das Datum im Format MM/DD/YYYY angegeben, welches von Cassandra nicht verarbeitet werden konnte. Dieses musste durch „August 31, 2017“ ersetzt werden. Erst nach der Änderung des Datentyps „Date“ in „varchar“ konnte das in der CSV-Datei verwendete Format in die Datenbank importiert werden.

Weitere Obstruktionen ergaben sich zunächst durch die Anfragesprache CQL, da einzelne Funktionen sowie deren Bedeutung weder in der Fachliteratur noch im Online-Wiki von Apache Cassandra auffindbar waren. Funktionen wie PAGING OFF erschwerten beispielsweise den Umgang mit der Datenbank, da diese standardmäßig aktiviert ist und die Ausgabe der Datensätze auf maximal 100 Einträge begrenzte. Ebenso existieren zahlreiche weitere Funktionen, die mithilfe des Wikis sowie der Literatur nicht gelöst werden konnten, sodass es notwendig wurde neben den offiziellen Quellen Wissensblogs hinzuzuziehen, um entsprechende Funktionen ausfindig zu machen.

Messfehler konnten bei dem Test beider Datenbanken auch durch nebenläufige Prozesse verursacht werden. Die Anzahl an nebenläufigen Prozessen wurde daher zu Beginn der Tests minimiert. Es war jedoch nicht möglich alle Prozesse innerhalb des Systems zu

eliminieren, da fundamentale Prozesse wie Kernel Tasks für die Lauffähigkeit des Systems unumgänglich sind. Dennoch konnten durch die Minimierung größere Messfehler im Benchmarktest dezimiert werden, sodass diese die Ergebnisse nur geringfügig beeinflussten.

Alles in allem wurden für den Test die notwendigen Konfigurationen vorgenommen und jegliche Probleme mit direkten Auswirkungen auf den Test behoben. Die verbleibenden Messfehler wurden durch mehrmaliges Wiederholen des Tests zu einem Mittelwert approximiert.

5.7 Zusammenfassung

Der praktische Teil dieser Masterarbeit beinhaltet die Durchführung des Benchmarktests anhand dessen die beiden Datenbanksysteme: MySQL (Relationale Datenbank) als auch Apache Cassandra (NoSQL-Datenbank) miteinander verglichen werden.

Zunächst wird die Begrifflichkeit *Benchmarktest* genauer erläutert und die Kriterien an einen erfolgreichen Test definiert. Im weiteren Verlauf wird die Installation sowie die Konfiguration beider Datenbanksysteme beschrieben. Mittels des notwendigen SQL bzw. CQL-Codes werden die grundlegenden Unterschiede aufgezeigt, die für einen erfolgreichen Import des Datensatzes unverzichtbar sind. Ebenso werden die für das Betriebssystem MacOS erforderlichen Parameter konkretisiert sowie wichtige Konfigurationsschritte ausführlich erklärt.

Für den anschließenden Benchmarktest kommt dem Testdatensatz eine große Bedeutung zu, da dieser eine entsprechende Größe und Komplexität aufweisen muss, um unter den gegebenen Hardwarevoraussetzungen überhaupt durchführbar zu sein. Ebenso ist ein Bezug der Daten zur realen Welt wichtig, um hierdurch bessere Analysen vorzunehmen. Im weiteren Verlauf wird die verwendete Hardware genauer beschrieben, um Rückschlüsse auf die Ergebnisse unter der Leistungsfähigkeit der Hardware zu ziehen.

Daran anschließend erfolgen die jeweils vier unterschiedlichen Benchmarktests. Jeder Test wird unter gleichen Bedingungen durchgeführt und zehnmal wiederholt, um eine Mittelwertberechnung zu ermöglichen und Zufallsergebnisse auszuschließen. Dabei werden von jedem der vier Tests die unterschiedlichsten Ansprüche an die Datenbank gestellt. Beim ersten Test soll die Ausgabe aller Datensätze des Datensatzes erfolgen. Hierbei entstehen beachtliche Unterschiede, bei der das relationale System mindestens fünfmal so schnell eine Ausgabe erzeugt. Im zweiten Test wird der Datensatz auf Gleichheit geprüft, um die Ausgabe aller Autodiebstähle zurückzuliefern, die im Jahr 2005 in der Stadt Chicago verzeichnet wurden. Dazu ist ein zweistufiger Prozess innerhalb der

Datenbank notwendig, der zunächst nach einer Bedingung selektiert und anschließend die erhaltenen Werte mit der zweiten Bedingung vergleicht und weiter selektiert, bis beide Bedingungen erfüllt sind. Test drei und Test vier sind Bereichsabfragen, die alle Daten ausgeben, die in einem zuvor definierten Bereich liegen. Dazu wird in beiden Fällen eine obere bzw. eine untere Schranke verwendet, die den Datensatz eingrenzt. Alle Tests zeigen deutliche Laufzeitunterschiede, die so in dieser Größenordnung nicht erwartet wurden. Das NoSQL-System benötigt bei allen Tests ein Vielfaches mehr an Zeit als das relationale System von MySQL.

Die Analyse der Ergebnisse beschreibt verschiedene Faktoren genauer, um das Resultat besser einordnen zu können. Dabei kann festgehalten werden, dass die Datenhaltung der beiden Systeme sich deutlich voneinander unterschiedet, was unter anderem ein Grund für die errechneten Antwortzeiten darstellt. Ebenso gibt es gravierende Unterschiede in der Anfrageverarbeitung, wobei NoSQL das Map/Reduce- Verfahren zum Einsatz bringt, das speziell für große bis sehr große Datensätze geeignet, jedoch für kleinere Datensätze eher inadäquat ist. Es wird in diesem Fall viel Rechenleistung in Anspruch genommen, die jedoch nicht gebraucht wird. Dies zeigt auch der direkte Vergleich mit dem relationalen System, das bei Abfragen auf die Verwendung der relationalen Algebra setzt und somit bei verhältnismäßig kleinen Datensätzen deutlich schneller zu einen Ergebnis gelangt.

6 Fazit

Die vorliegende Masterarbeit soll eine Orientierungshilfe im Dschungel der Datenbanksysteme darstellen. Zu diesem Zweck findet eine Gegenüberstellung des relationalen Datenbanksystems MySQL und des NoSQL-Systems Cassandra statt, um mittels Durchführung von Benchmarktests die theoretischen Fakten zu untermauern.

Die Ergebnisse der verschiedenen Benchmarktests zeigen jedoch, dass sich Theorie und Praxis deutlich voneinander unterscheiden. So wird zu Beginn der Masterarbeit angenommen, dass das NoSQL-System Cassandra dem MySQL-System gegenüber eindeutig überlegen ist. Auch die theoretischen Vorgaben sowie das Hinzuziehen von Fachliteratur, ebenso wie die Auseinandersetzung mit schon angewandten Benchmarktests zum Thema Datenbanken bestärken diese Vermutung zunächst.

Im praktischen Teil der Arbeit sind hingegen aller Erwartungen, die Ergebnisse explizit zum Nachteil von Cassandra ausgefallen. So zeigt das System bereits zu Beginn einige Tücken. Besonders im Internet wird damit geworben, dass NoSQL-Systeme „Out of the Box“ oder „Lauffähig in nur 10 Minuten“ sind. Dies kann nicht bestätigt werden, da selbst mit Vorkenntnissen im Bereich der relationalen Datenbanken nicht darauf zurückgegriffen werden kann, da die Funktionsweise sowie die Anfragesprache CQL eine andere ist. Neben der Komplexität von Cassandra beansprucht vor allem das Erlernen der Anfragesprache CQL einiges an Zeit.

Die Benchmarkergebnisse präsentieren Cassandra gegenüber MySQL als deutlichen Verlierer. Im ersten und zweiten Test benötigt Cassandra die fünffache Zeit, um zum selben Ergebnis zu gelangen. Prognostiziert wurde ursprünglich, dass Cassandra als klarer Favorit bei einem Datensatz von 5,6 Millionen Daten hervorgeht. Aber auch die weiteren Benchmarktests zeigen deutlich, dass die Antwortzeit unter Cassandra um ein Vielfaches höher ist als bei MySQL. Die Verteilung der Daten auf mehrere Systeme (Knoten) kann das Ergebnis nur maginal im Zehntel-Sekundenbereich verbessern.

Die Auswertung der Ergebnisse bestätigt, dass bei verhältnismäßig kleinen Datensätzen das NoSQL-System Cassandra gegenüber dem relationalen System MySQL nicht mithalten kann, auch nicht bei einer Erhöhung der Anzahl der Knoten. Ähnliche Studien zum Thema haben gezeigt, dass die Leistungsfähigkeit von Cassandra erst ab einem Datensatz in der Größenordnung von 100 Millionen und unter Verwendung von mindestens 12 Knoten aufholen und deutliche Unterschiede zu MySQL nachgewiesen werden können. Anhand dieser Masterarbeit kann im speziellen Fall gezeigt werden, dass das relationale System durchaus seine Berechtigung besitzt und im Bereich von kleinen Datensätzen gegenüber dem Konkurrenten in Führung liegt.

Es kann aufgrund der Ergebnisse jedoch keine allgemeingültige Empfehlung zu den Datenbanksystemen vorgenommen werden, da wegen des zeitlichen Rahmens des Projektes nur Teilgebiete und Funktionen betrachtet wurden, was lediglich eine Tendenz zu MySQL bei kleineren Datensätzen zulässt.

7 Zusammenfassung und Ausblick

Der heutige Trend in der Informations- und Kommunikationstechnologie geht hin zu Big Data Anwendungen, wodurch extrem große Mengen an strukturierten und unstrukturierten Daten mit hoher Geschwindigkeit verarbeitet werden müssen. Die hierfür benötigten Anforderungen durch Sozial-Media-Plattformen überschreiten häufig die Kapazität sowie Leistungsfähigkeit, der bis dato verwendeten relationalen Datenbanksysteme. Aufgrund dieser Entwicklung sind in den 2000er Jahren zahlreiche NoSQL-Systeme entstanden, die für Big Data konzipiert wurden. Diese verfügen im Gegensatz zu MySQL-Anwendungen über eine hohe Skalierbarkeit, wodurch diese vor allem durch redundante Datenhaltung der Ausfallsicherheit vorbeugen können.

Dazu wurden in den ersten Kapiteln die verschiedenen Formen der Datenhaltung sowie die Datenreduktion bei Anfragen im Hinblick auf relationale Datenbanken sowie NoSQL-Systeme aufgegriffen und ausführlich beschrieben. Es wurden zahlreiche NoSQL-Systeme ausgemacht, die in ihrer Grundfunktionalität kontinuierlich auf das Verteilen der Daten in großen Netzwerken ausgelegt sind. Sie sind in der Regel in Dokumentenorientierte Datenbanken, Key/Value-Datenbanken, Spaltenorientierte Datenbanken und Graphdatenbanken unterteilt, wobei in dieser Arbeit besondere Aufmerksamkeit auf die namhaften Vertreter Redis, CouchDB, Cassandra und Neo4j gelegt wurde, im Hinblick auf den späteren Benchmarktest.

Im Kapitel „Kritik an NoSQL-Datenbanken“ wurde dargelegt, dass diese eine deutlich geringere Entwicklungsreife gegenüber den relationalen Systemen besitzen, die schon seit mehreren Jahren erfolgreich eingesetzt und sich weiterentwickelt haben. Zusätzliche Kritikpunkte waren auch der unzureichende Support und die geringe Standardisierung. Darüber hinaus existiert keine verlässliche und qualitativ hochwertige Fachliteratur. Dies erschwert vielen Anwendern den Einstieg, da sie zum einen im Dschungel der zahllosen Systeme leicht den Überblick verlieren können und zum anderen gerade zu Beginn den effizienten Einsatz der Systeme nicht gewährleisten. Aufgrund des Mangels werden viele Anwender zum Prinzip „Learing by doing“ gezwungen und greifen dadurch zu den ihnen besser bekannten relationalen Datenbanken.

Daraus ist abzuleiten, dass relationale Datenbanken für die Anwender meist geeigneter erscheinen, da sich Modelle aus der realen Welt unmittelbar in der Datenbank abbilden lassen. So können beispielsweise in einem Entity-Relationship-Modell die exakten Anforderungen an die spätere Datenbank festgelegt werden. Ebenso ist die Kommunikation zwischen Entwickler und künftigem Anwender deutlich einfacher, da eine Visualisierung der späteren Datenbank in Form eines Modells erfolgt. Positiv ist die ausrei-

chende Fachliteratur hervorzuheben, die in verschiedenen Entwicklungsstufen das System genau beschreibt. Dank stetiger Innovationen in den Strukturen wie Datenhaltung und der Anfragesprache SQL sind diese Systeme keineswegs zum alten Eisen gehörig, sondern finden auf dem heutigen Markt immer noch zahlreiche Anwendungsgebiete.

Im dritten Teil der Masterarbeit wurde zunächst der Begriff Benchmark anhand aktueller Literatur definiert. Im Anschluss erfolgten verschiedene Benchmarktests, bei denen sich MySQL zum Thema Antwortzeiten deutlich vom Kontrahenten Cassabdra lösen konnte. Die dort erhaltenen Ergebnisse führten durchaus zu einigen Überraschungen, die zu Beginn der Arbeit nicht angenommen wurden. So konnte nachgewiesen werden, dass in allen vier Benchmarktests MySQL bis zu fünf Mal schneller als das NoSQL-System Cassandra waren. Dabei zeigte das System der Apache Foundation bei einem Benchmarktest mit 5,6 Millionen Datensätzen deutliche Geschwindigkeitsunterschiede, die auf die Datenhaltung sowie auf die Anfrageverarbeitung, selbst bei kleinen Datenmengen zurückzuführen waren. Laut Studien und anderen wissenschaftlichen Publikationen hat sich gezeigt, dass NoSQL-Systeme ihre volle Stärke erst bei etwa 100 Millionen Datensätzen vollständig entfalten, wenn diese in einem hochperformanten Netzwerk auf mindestens 12 Knoten verteilt werden. Dies ist ein erster Anknüpfungspunkt, der in zukünftigen Benchmarktests weiter verfolgt werden sollte.

Aufbauend auf den Ergebnissen dieser Arbeit lassen sich Bereiche identifizieren, in denen weitere Forschungsanstrengungen unternommen werden sollten. Beispielsweise könnte zur Auslastung von Cassandra der Datensatz ausgeweitet werden, um die Erkenntnisse zu überprüfen die andere wissenschaftliche Studien liefern, um deren These zu stärken. Des Weiteren sollten zukünftig Datensätze verwendet werden, die aus mehreren Tabellen bestehen, um Join Abfragen zu ermöglichen.

Darüber hinaus bieten NoSQL-Datenbanken weitere interessante Möglichkeiten, die Testergebnisse zu beeinflussen. Hier lohnt es sich neben Cassandra, auch weitere Vertreter wie „Redis, „CouchDB“ oder etwa „Googles Big Table“ genauer zu betrachten, da diese Systeme durchaus einige Vorteile vorweisen. In der Welt der relationalen Datenbanksysteme ist der Spielraum hingegen deutlich geringer. Neben MySQL könnten möglicherweise kommerzielle Systeme einem Benchmarktest unterzogen werden. Diese unterscheiden sich jedoch weder in der Datenhaltung noch in der Anfragesprache deutlich von MySQL.

Wünschenswert wäre auch die Änderung der Speichermethode von MySQL. So könnte statt der Speicherstruktur des Heaps die B-Baum oder Hash-Funktion verwendet werden, was Auswirkungen auf die Ergebnisse des Benchmarktests nehmen sollte.

Zusätzlich müsste ein weiteres Augenmerk auf die verteilte Datenhaltung unter NoSQL gelegt werden. Wie im Test deutlich wurde, konnten mit steigender Knotenanzahl

die Antwortzeiten minimiert werden. Dazu wurde eine virtuelle Maschine (VM) zum Einsatz gebracht, da enorme Schwankungen bei der Verteilung der Daten über das Netzwerk auf mehrere Rechner sichtbar wurden. Vor diesem Hintergrund erscheint es sinnvoll, eine leistungsfähige Netzwerkstruktur zu verwenden, die große bis sehr große Datenmengen verarbeiten kann. Zudem sollte die Hardware der Server eine identische Leistungsfähigkeit aufweisen, um den Test nicht zu beeinflussen.

Es wäre sicherlich sehr aufschlussreich bei einem erneuten Benchmarktest die Testanfragen zu erweitern, da die in der Masterarbeit verwendeten Anfragen nur einen geringen Teil abdecken. Beispielsweise könnten bei einem entsprechenden Datensatz ebenso Join Operationen Anwendung finden, um die Datensätze kreuzweise miteinander zu verknüpfen.

Alles in allem soll die Masterarbeit Orientierungspunkte für künftige Arbeiten liefern, wobei aufgrund der Komplexität des Themas nur ein kleiner Teil der Systeme beschrieben wurde..

8 Literaturverzeichnis

[1] B. Schwartz, S. Z. (2012). *High Performance MySQL.* Californien USA: O´Reilly Media.

[2] B.F. Cooper, A. S. (2010). *Benchmarking Cloud Serving Systems with YCSB.* ACM Symposium on Cloud Computing: SOCC´10.

[3] Corporation, O. (2017, Mai 29). *MySQL*. Retrieved from https://dev.mysql.com/doc/workbench/en/wb-command-line-options.html

[4] DataStax. (2009, Juni 17). *DataStax*. Retrieved März 09, 2017, from http://www.datastax.com/documentation/cassandra(2.0/pdf/cassandra20.pdf

[5] DataStax. (2013). *Benchmarking Top NoSQL Databases*. Retrieved Mai 7, 2017, from http://www.datastax.com/resources/whitepapers/benchmarking-top-nosql-databases

[6] DataStax. (2014). *Apache Cassandra 1.1 Documentation*. Retrieved Mai 19, 2017, from htt://www.datastax.com/doc-source/pdf/cassandra11.pdf

[7] Downing, A. (2011). *Debunking the NoSQL Hype.* California USA: Oracle, Redwood Shores.

[8] Dresden, T. U. (2017, Januar 30). *Technische Universität Dresden*. Retrieved from h

[9] Erhard Rahm, G. S.-U. (2015). *Verteiltes und Paralleles Datenmanagement.* Leipzig: Springer Verlag Berlin Heidelberg.

[10] F. Bry, T. F. (2004). *Datenströme Forschungsprojekt.* München: Universität Münschen.

[11] Foundation, A. S. (2014). *Apache Software Foundation*. Retrieved März 23, 2017, from http://hbase.apache.org/book.html.

[12] Foundation, T. A. (2016, August 20). *Apache Cassandra*. Retrieved from http://cassandra.apache.org/

[13] G. DeCandia, D. H. (2007, März 19). *Dynamo: Amazon's Highly Available Key-value Store.* Retrieved from http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf

[14] Groningen, M. v. (2009). *Introduction to Hadoop.* Triforg Blog.

[15] H. Faeskorn-Woyke, B. B. (2007). *Datenbanksysteme: Theorie und Praxis mit SQL3, Oracle und MySQL.* München : Pearson Studium München.

[16] J. Dean, S. G. (2008). *MapReduce: Simplified Data Processing on Large Clusters.* San Francisco: ACM.

[17] Koch, S. (2011, Juni 01). *Eliteinformatiker*. Retrieved from http://www.eliteinformatiker.de/2011/06/01/nosql-key-value-datenbanken-memcachedb-project-voldemort-redis

[18] L. Cagliero, A. F. (2013). *A Rule-Based Flickr Tag Recommendation System.* London UK: Springer-Verlag.

[19] Luke Welling, L. T. (2004). *MySQL Tutorial.* München: Addison-Wesley.

[20] M.T. Özsu, P. V. (2011). *Principles of Distributed Database Systems.* New York: USA Springer-Verlag.

[21] Matthiessen, G. (2007). *Relationale Datenbanken und Standard SQL.* Sybase iAnywhere.

[22] Mehling, M. (2010, April 14). *10 things you Need to Know About NoSQL Databases*. Retrieved März 24, 2017, from http://databasejournal.com/features/article.php/3905531/10-things-you-Need-to-Know-Abou-NoSQL-Databases

[23] Metz, D. (2013). *The Concept of a Real-Time Enterprise in Manufacturing.* Siegen: Springer Gabler.

[24] Nalinda, C. (2017, März 04). *TechSprio*. Retrieved from http://techspiro.blogspot.de/2014/10/cap-theorem-nosql-big-data.html

[25] Oracle. (2017, April 25). *MySQL*. Retrieved from https://www.mysql.com/de/

[26] Portal, C. D. (2017). *Chicago Data Portal*. (C. o. Cicago, Editor) Retrieved Juni 05, 2017, from https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2

[27] Prof. Dr. Stefan Edlich, A. F. (2011). *Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken.* München: Carl Hanser Verlag München.

[28] R. Elmasri, S. N. (2002). *Grundlagen von Datenbanksystemen.* München: Pearson Studium München.

[29] Rahm, E. (1994). *Mehrrechner-Datenbanksysteme: Grundlagen der verteilten und parallelen Datenverarbeitung.* Bonn: Addison-Wesley.

[30] Ramez A. Elmasri, S. B. (2009). *Grundlagen von Datenbanksystemen* (Vol. 3). Hallbergmoos: Parson Deutschland GmbH.

[31] Rolland, F. (2003). *Datenbanksysteme.* München: Pearson Studium München.

[32] Sauer, H. (2002). *Relationale Datenbanken.* München : Addison-Wesley .

[33] Scientiest, K. D. (2017, März 05). *KB Data Scientist*. Retrieved from https://hadoop4usa.wordpress.com/2012/04/13/scale-out-up/

[34] Stompe, F. (2009). *Real-Time Data Mining.* Hamburg: Diplomica Verlag.

[35] team, T. p. (2017, Mai 25). *PhpMyAdmin*. Retrieved from https://docs.phpmyadmin.net/de/latest/

[36] Tiemeyer, E. (2016). *IT-System-Management.* München: Carl Hanser Verlag München.

[37] Tiwani, S. (2011). *Professional NoSQL.* Indianapolis: John Wiley & Sons, Inc.

[38] University, S. (2003, Januar 21). *Stream. The Stanford Stream Data Manager*. Retrieved from http://ilpubs.stanford.edu:8090/583/1/2003-21.pdf

[39] Versick, D. (2010). *Verfahren und Werkzeuge zur Leistungsmessung, -analyse und -bewertung der Ein-/Ausgabeeinheiten von Rechnersystemen.* Rostock: Logos Verlag Berlin.