Technology
Arts Sciences
TH Köln

Eignung von nichtrelationalen Datenbanksystemen zur Erstellung einer leichtathletischen Mehrkampfdatenbank

und Vergleich mit einer relationalen Umsetzung

Bachelorarbeit

ausgearbeitet von

Matthias Frank
Daniel Morady

zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

vorgelegt an der

Technischen Hochschule Köln
Campus Gummersbach
Fakultät für Informatik und
Ingenieurwissenschaften

im Studiengang
Informatik

Erster Prüfer: Prof. Dr. Birgit Bertelsmeier
Technische Hochschule Köln

Zweiter Prüfer: Prof. Dr. Heide Faeskorn-Woyke
Technische Hochschule Köln

Gummersbach, im Februar 2017

Adressen: Matthias Frank
Zum Heuenfeld 3
51597 Morsbach
matthias.frank@live.com

Daniel Morady
An der Schüttenhöhe 4
51643 Gummersbach
daniel-morady@web.de

Prof. Dr. Birgit Bertelsmeier
Technische Hochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
birgit.bertelsmeier@th-koeln.de

Prof. Dr. Heide Faeskorn-Woyke
Technische Hochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
heide.faeskorn-woyke@th-koeln.de

Inhaltsverzeichnis

1 Einleitung **6**

2 Die Eingrenzung nichtrelationaler Systeme **7**

2.1 Eine Auswahl im Vorfeld . . . 7

2.2 Die betrachteten Datenbankgruppen . . . 8

2.2.1 Graphendatenbanksysteme . . . 8

2.2.2 Spaltenorientierte Datenbanksysteme . . . 8

2.2.3 Key/Value-Datenbanksysteme . . . 10

2.2.4 Dokumentenorientierte Datenbanksysteme . . . 11

2.3 Eine Auflistung der Anforderungen des Projekts . . . 11

2.4 Eine erste Eignungseinschätzung . . . 13

2.4.1 Graphendatenbanksysteme . . . 13

2.4.2 Spaltenorientierte Datenbanksysteme . . . 13

2.4.3 Key/Value-Datenbanksysteme . . . 13

2.4.4 Dokumentenorientierte Datenbanksysteme . . . 14

2.5 Fazit . . . 14

3 Eine weitergehende Betrachtung der gewählten Systeme **15**

3.1 Die Eingrenzung der notwendigen Eigenschaften . . . 15

3.1.1 Aggregate . . . 15

3.1.2 Schemafreiheit . . . 18

3.1.3 Datenkonsistenz (Consistency) . . . 20

3.1.4 Datenaufteilung (Sharding) . . . 25

3.1.5 Datenspiegelung (Replication) . . . 26

3.1.6 Zusammenfassung . . . 27

3.2 Die Begutachtung der verbleibenden Datenbanksysteme . . . 29

3.2.1 Graphendatenbanksysteme . . . 29

3.2.2 Key/Value-Datenbanksysteme . . . 30

3.2.3 Dokumentenbasierte Datenbanksysteme . . . 32

3.3 Fazit . . . 33

4 Die Verwendung des ausgewählten Systems **34**

4.1 Die Bestimmung der Softwarelösung . . . 34

4.1.1 Die möglichen Softwarelösungen . . . 35

4.1.2 Der Vergleich der API-Unterstützungen . . . 37

4.1.3 Fazit . . . 38
4.2 Die nichtrelationale Umsetzung . . . 38
4.2.1 Die Modellierung in der Umsetzung . . . 39
4.2.2 Die Referenzierung in Dokumenten . . . 40

5 Der Vergleich mit der relationalen Umsetzung 42
5.1 Die Sortierung einer Kollektion / Tabelle . . . 43
5.1.1 in MongoDB . . . 43
5.1.2 in MariaDB . . . 45
5.1.3 Der Vergleich beider Umsetzungen . . . 47
5.2 Die Suche in einer Kollektion / Tabelle . . . 47
5.2.1 in MongoDB . . . 48
5.2.2 in MariaDB . . . 49
5.2.3 Der Vergleich beider Umsetzungen . . . 50
5.3 Die Suche in verschachtelten Dokumenten / Zuordnungstabellen . . . 51
5.3.1 in MongoDB . . . 51
5.3.2 in MariaDB . . . 52
5.3.3 Der Vergleich beider Umsetzungen . . . 53
5.4 Die Umstrukturierende Abfrage in Dokumenten / Tabellen mit JOINs . . . 53
5.4.1 Die Beschreibung des Testaufbaus . . . 54
5.4.2 in MongoDB . . . 55
5.4.3 in MariaDB . . . 59
5.4.4 Der Vergleich beider Umsetzungen . . . 60
5.5 Fazit . . . 60
5.5.1 Ein Grundsatzproblem beim Vergleich . . . 60
5.5.2 Zusammenfassung . . . 61

6 Fazit 63
6.1 Vergleich der nichtrelationalen Systeme . . . 63
6.2 Vergleich der Umsetzungen . . . 64

Abbildungsverzeichnis 65

Tabellenverzeichnis 66

Literaturverzeichnis 67

Internet-Quellen-Verzeichnis 69

Messreihen 71

Aufteilung der Ausarbeitung 83

Kurzfassung

Zielsetzung dieser Abschlussarbeit ist die Eignungsprüfung nichtrelationaler Datenbanksysteme für die Erstellung einer leichtathletischen Mehrkampfdatenbank. Sie ist in fünf Kapitel aufgeteilt.

Im ersten Kapitel wird einleitend auf die Entstehung dieser Arbeit und des zugrundeliegenden Projekts eingegangen.

Im zweiten Kapitel werden die vier großen Hauptgruppen nichtrelationaler Datenbanksysteme grob auf ihre generelle Eignung für die Verwendung und Umsetzung der Mehrkampfdatenbank anhand gegebener Anforderungen bewertet und aussortiert.

Im dritten Kapitel werden die verbliebenen Hauptgruppen eingehender betrachtet und das am besten geeignet erscheinende Gruppe für eine weitergehende Betrachtung ausgewählt.

Im vierten Kapitel werden mögliche Kandidaten aus der im vorherigen Kapitel bestimmten Hauptgruppe für die tatsächliche Umsetzung bewertet. Abschließend wird ein Kandidat für die Umsetzung der Mehrkampfdatenbank bestimmt.

Im fünften Kapitel wird ein Vergleich mit einer relationalen Umsetzung der Mehrkampfdatenbank gezogen, dabei werden verschiedene Datenbankanfragen auf unterschiedlichen Datenmengen verglichen. Außerdem einzelne Verfahrensweisen zur Abwicklung von Anfragen betrachtet.

In einem abschließenden Fazit werden die Ergebnisse und Erkenntnisse der Arbeit nochmals kurz zusammengefasst.

1 Einleitung

Im Verlauf des letzten Jahres wurde aus einem, ursprünglich, relativ kleinen und einfachen Informatikprojekt[1] eine immer umfangreichere Umsetzung. Die zu Beginn geplante und umgesetzte, sehr speziell ausgerichtete Darstellung eines Zehnkampfes wurde in ihrer Weiterentwicklung in einem Praxisprojekt[2] zu einer umfassenden Mehrkampfdatenbank ausgebaut.
Und während für die ersten Umsetzungen ein relationales Datenbanksystem nicht unbedingt die perfekte, aber doch die schnelle und funktionale Lösung war, wurde im Verlauf des Praxisprojektes bei der Umsetzung der Mehrkampfdatenbank die Begrenzungen und Einschränkungen dieses relationalen Systems schnell deutlich. Mit jeder weiteren Idee mussten erhebliche Änderungen am Datenbankmodell durchgeführt werden.

Es erschien daher nur logisch, die Mehrkampfdatenbank auch aus einem nichtrelationalen Ansatz heraus zu entwickeln und dabei zu prüfen, ob das schemafreie Konzept weniger Probleme bereitet oder ob nun andere Beschränkungen die Umsetzung und Erweiterung erschweren.
Obwohl während der Planung für diese Ausarbeitung eine tatsächliche Umsetzung der Mehrkampfdatenbank in einem NoSQL-System nicht zwingend im Raum stand (und allenfalls als Option gedacht war), wurde recht schnell klar, dass ein, zumindest rudimentär lauffähiger, Prototyp umgesetzt werden "muss".
Nur in der praktischen Umsetzung konnten gewisse Problemstellungen und die dazugehörigen Lösungsansätze genauer untersucht und ein direkter Vergleich mit einer relationalen Umsetzung ermöglicht werden.

Die in dieser Arbeit erlangten Erkenntnisse sollen gleichwohl keine generelle Bewertung der verschiedenen Datenbanksysteme darstellen. Die im Verlauf der Arbeit gemachten Erfahrungen sind vielfach durch die Anforderungen der Aufgabe bestimmt und können daher nur grob auf ähnlich gelagerte Probleme übertragen werden.

[1] [FM16a]
[2] [FM16b]

2 Die Eingrenzung nichtrelationaler Systeme

Der Bereich der nichtrelationalen Datenbanksysteme umfasst eine Vielzahl verschiedener Einsatzgebiete und Datenbanktypen. Neben Umsetzungen für spezielle und individuelle Anwendungsbereiche, wie z.B. für Biometrie, Fingerabdruck- und Gesichtserkennung, analytische Datenbanksysteme, geographische Daten und Big Data/Data Mining[1], haben sich vier Hauptgruppen als Struktureinteilung etabliert:

- Graphendatenbanksysteme (Graph Databases)
- Spaltenorientierte Datenbanksysteme (Wide Column Stores)
- Key/Value-Datenbanksysteme (Key-Value-Stores)
- Dokumentenorientierte Datenbanksysteme (Document Stores)

In diesem Kapitel werden diese vier Hauptgruppen zuerst in Abschnitt 2.2 kurz vorgestellt, wobei einige Eigenschaften und -arten der Systeme hervorgehoben werden.
Im Abschnitt 2.3 werden dann die Anforderungen genauer dargestellt, welche ein später umzusetzendes System erfüllen sollte.
Danach werden in Abschnitt 2.4 die vier Hauptgruppen auf ihre generelle Eignung für diese Umsetzung überprüft.
Im abschließenden Fazit erfolgt dann eine kurze Schlussbewertung. Diese ist Grundlage für eine eingehendere Betrachtung und Eignungsprüfung in Kapitel 3.

2.1 Eine Auswahl im Vorfeld

Als Grundlage für die Eignungsprüfung haben sich die Autoren für die Eingangs genannten vier Hauptgruppen entschieden. Die Gründe hierfür sind

[1] [Cel14]

1. die Ein- und Begrenzung der zu betrachtenden Systeme, um den zeitlichen Umfang überschaubar zu halten.
2. der Zugriff auf umfangreiche Literatur und Dokumentation in schriftlicher und digitaler Form.
3. die Ausrichtung der Systeme auf ein breit gefächertes Einsatzgebiet (im Gegensatz zu den Speziallösungen z.B. für Gesichtserkennung).

2.2 Die betrachteten Datenbankgruppen

In diesem Abschnitt werden die vier Hauptgruppen der nichtrelationalen Datenbanksysteme kurz in ihrer Funktionsweise und ihrem Einsatzschwerpunkt beschrieben.

2.2.1 Graphendatenbanksysteme (Graph Databases)

Graphendatenbanksysteme legen die gespeicherten Informationen in Form eines Graphen an, der aus Knoten (Nodes) und Kanten (Edges) besteht. Die Informationen werden meist in den Knoten abgelegt und die Kanten beschreiben die Verbindung der Knoten zueinander, es können aber auch hier Informationen hinterlegt werden. Um Knoten und Kanten besser einzuordnen, können sie mit Eigenschaften (Properties) in Gruppen unterteilt werden.
Graphendatenbanksysteme werden vor allem bei stark vernetzten Informationen (z.B. zur Darstellung sozialer Netzwerke) eingesetzt, da die Verbindungen zwischen den Informationen direkt auswertbar sind und nicht erst, wie bei relationalen Systemen, über Fremdschlüssel und Joins aus verschiedenen Tabellen zusammengesucht werden müssen.

2.2.2 Spaltenorientierte Datenbanksysteme (Wide-Column-Store)

Auf den ersten Blick scheinen spaltenorientierte und relationale Datenbanksysteme sehr ähnlich aufgebaut zu sein, da beide in Form von Tabellen dargestellt werden können. Während in relationalen Systemen die Informationen zeilenweise gespeichert werden, legen spaltenorientierte Systeme die Informationen spaltenweise ab. In Abbildung 2.1 ist dies beispielhaft dargestellt.

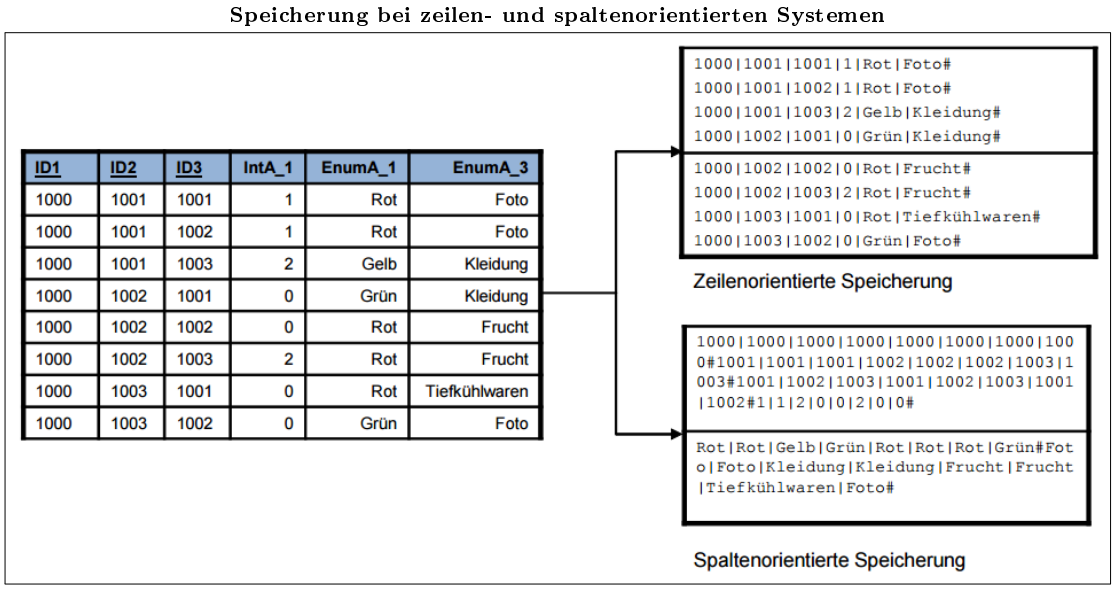

Abbildung 2.1: Die Informationen (links) werden entweder zeilenweise oder spaltenweise gespeichert. Quellenvorlage: [Her12]

Beide Verfahren haben aufgrund ihrer Speicherstruktur unterschiedliche Vor- und Nachteile. Zeilenorientierte Systeme eignen sich besser für die Abfrage von Datensätzen, da diese zusammenhängend im Speicher abgelegt sind. Analysierende Vorgänge über Spalten hingegen gestalten sich sehr aufwendig, da die Informationen im Speicher umständlich zusammengesucht werden müssen.
Spaltenorientierte Systeme können umgekehrt sehr gut analysierende Vorgänge ausführen, die Abfrage von Datensätzen erfordert ein Zusammensuchen der Teilinformationen.[2]

Soll aus dem Datenbestand in T 2.1 der Datensatz mit der ID-Kombination 1000 (*ID1*), 1001 (*ID2*) und 1002 (*ID3*) ausgelesen werden, finden sich alle Informationen bei der zeilenorientierte Speicherung in einer Zeile. In der spaltenorientierten Speicherung findet sich in jeder Datenzeile genau ein Teil der Gesamtinformation an der exakt gleichen Position.
Bei einer analysierenden Abfrage ist das Verhalten genau umgekehrt. Wenn die Anzahl des Wortes 'Rot' in *EnumA_1* gezählt werden soll, muss ein zeilenorientiertes System alle Zeilen nach den Teilinformationen durchsuchen, das spaltenorientierte System findet alle Informationen in einer Zeile vor.

Die modernen Spaltenorientierten Systeme beheben den beschriebenen Nachteil, indem Sie einen zweiten Schlüssel, den sogenannten 'row key', verwenden.

[2] [Edl10, Seite 53]

Verwendung von 'Row Keys' in spaltenorientierten Systemen

row key	column names					
	ID1	ID2	ID3	IntA_1	EnumA_1	EnumA_3
row_a	1000	1001	1001	1	Rot	Foto
row_b	1000	1001	1001	1	Rot	Foto
row_c	1000	1001	1002	1	Rot	Foto
row_d	1000	1001	1003	2	Gelb	Kleidung
row_e	1000	1002	1001	0	Grün	Kleidung
row_f	1000	1002	1002	0	Rot	Frucht
row_g	1000	1002	1003	2	Rot	Frucht
row_h	1000	1003	1001	0	Rot	Tiefkühlwaren
row_i	1000	1003	1002	0	Grün	Foto

Tabelle 2.1: Jede Teilinformation ist durch einen 'row key' und einen Spaltennamen eindeutig auffindbar.

Durch diese Ergänzung gestalten sich Abfragen auf Datensätze jetzt sehr viel einfacher. Jede Dateninformation kann über den 'Row Key' und Spaltennamen aufgefunden werden. Das Ergebnis ist in Tabelle 2.1 zu sehen.
Das Einfügen und Ändern von Datensätzen kann aber immer noch Probleme bereiten, da die Speicherung nach wie vor spaltenorientiert erfolgt und somit die Dateninformationen auf die verschiedenen Spalten aufgeteilt werden müssen.

2.2.3 Key/Value-Datenbanksysteme (Key-Value-Store)

In der Literatur werden die Key-Value-Systeme als die einfachsten Umsetzungen innerhalb der NoSQL-Datenbanksysteme gesehen,[3] da sie lediglich aus einem Schlüssel (Key) und der zugehörigen Informationen (Value) bestehen.
Der Aufbau der Information ist nicht weiter definiert, so dass eine Suche ausschließlich über den zugrundeliegenden Schlüssel möglich ist. Aufgrund dieser Eigenschaft kann das Datenbanksystem keinerlei Aussagen zur Struktur der gespeicherten Information machen. Die Analyse und Auswertung muss daher außerhalb des Datenbanksystems stattfinden.
Im allgemeinen sind nur wenige Operationen für Key/Value-Systeme vorgegeben:

- Speichern einer Information unter Angabe eines Schlüssels

[3] [SF13, Seite 81]

- Auslesen einer Information unter Angabe eines Schlüssel
- Löschen eines Schlüssels (und der abhängigen Information)

Da die Schlüssel automatisch indexiert werden und als einziges Auffindungsmerkmal von Informationen dienen, ist die Suche über die Schlüssel zwar sehr schnell, ohne entsprechenden Schlüssel ist die Suche im Datenbanksystem aber nahezu unmöglich.
Dieses grundsätzliche Manko wird von vielen Datenbanksystemen dadurch umgangen, dass sie, abseits der oben genannten Norm, Möglichkeiten zur Suche in abgelegten Informationen anbieten. Sie überschreiten damit die ohnehin sehr feine Abgrenzung zu den dokumentenorientierten Datenbanksystemen. Dies führt dazu, dass manche Umsetzungen nicht mehr klar zugeordnet werden. So wird DynamoDB von Amazon selbst als ein Datenbanksystem beschrieben, dass sowohl das Key/Value- als auch das Dokumenten-Modell umsetzen kann.[4]

2.2.4 Dokumentenorientierte Datenbanksysteme (Document Store)

Dokumentenorientierte Datenbanksysteme sind in ihrem Aufbau den Key/Value-Datenbanksystemen sehr ähnlich,[5] denn auch hier werden Informationen über einen Schlüssel abgelegt, wobei es zwei gravierende Erweiterungen gibt:

- Schlüssel werden automatisch vom System erzeugt
- die Informationen können indexiert werden und sind damit auch durchsuch- und direkt aufrufbar

Diese Erweiterungen stellen sicher, dass Schlüssel durch die automatische Erzeugung einzigartig bleiben und dass komplexere Suchanfragen auf die Informationen direkt vom Datenbanksystem durchgeführt und nicht auf eine andere Ebene ausgelagert werden müssen.

2.3 Eine Auflistung der Anforderungen des Projekts

Um die Eignung einer nichtrelationalen Systemgruppe für den Einsatz als leichtathletische Mehrkampfdatenbank festzustellen, wird zuerst ein Anforderungsprofil erstellt. Dieses soll aufzeigen, welche Informationen und Zusammenhänge durch das Datenbanksystem dargestellt werden müssen.

[4] [17b]
[5] [Vai13, Seite 41]

Die zentrale Information ist der leichtathletische Mehrkampf (Wettkampf), zu dem die nachfolgenden Informationen gehören die

- **Grundinformationen** zum Wettkampf, wie
  - **Name** oder **Bezeichnung**
  - **Austragungsdatum**
  - **Austragungsort**
- **teilnehmende Athleten** mit ihren
  - **persönlichen Informationen**, wie Name, Geburtsdatum und Geschlecht
  - **erbrachten Leistungen** in den verschiedenen Disziplinen
  - **erzielten Punktzahlen** aus den erbrachten Leistungen
- Informationen zu **Disziplinen** im Wettkampf, insbesondere ihre
  - **Grundinformationen**, wie Bezeichnung und technische Vorgaben
  - **Formeln** zur Punkteberechnung der sportlichen Leistung
  - **Faktoren** zur Berechnung der Punkte mit den Formeln

Mithilfe dieser zentralen Informationen ließe sich bereits ein entsprechendes System umsetzen. Um aber eine gewisse Konsistenz zu gewährleisten, ist folgendes zu beachten:

- Ein Athlet kann an mehreren Wettkämpfen teilgenommen haben
- Es können Anfragen zu einem Athleten stattfinden (in welchen Wettkämpfen er teilnahm)
- Wettkämpfe unterteilen sich in Wettkampfgruppen, wie zum Beispiel den Zehn-, Sieben- oder Dreikampf, die sich in ihrer
  - Zusammenstellung (Anzahl und Art der Disziplinen)
  - Auswertung (verwendete Formeln und Faktoren)

  gleichen.

Um eine konsistente Darstellung von Athleten und Wettkämpfen einer Wettkampfgruppe zu erhalten und um ihre Bearbeitung zentralisiert durchführen zu können, sollten diese Informationen einmalig als eine Art Vorlage erfasst werden und daraus immer wieder übernommen werden. Auf den vorhandenen Daten werden hauptsächlich Suchanfragen (z.B. nach Wettkämpfen und

Athleten) durchgeführt. Vergleichende Analysen (z.B. der beste 100m-Läufer unter 30 Jahren) sind nicht explizit vorgesehen.

2.4 Eine erste Eignungseinschätzung

Die in Abschnitt 2.2 vorgestellten Datenbanksysteme werden im Folgenden auf ihre Eignung zur Umsetzung der in Abschnitt 2.3 geforderten Eigenschaften geprüft.

Diese erste Einschätzung soll die Eignung nur in groben Zügen beleuchten und gravierende Nachteile oder Einschränkungen darstellen, die eine Umsetzung im jeweiligen System erschweren könnte.

2.4.1 Graphendatenbanksysteme (Graph Databases)

Da in Graphendatenbanksystemen die Informationen auf einzelne Knoten verteilt werden, ist die Verknüpfung dieser Knoten über die Kanten (und damit die Erstellung von Zusammenhängen) von elementarer Bedeutung.
Grundsätzlich kann ein Graph daher jede Art von Datenstruktur abbilden und auch die Auswertung der entsprechenden Daten wird durch verschiedene Ansätze unterstützt.

2.4.2 Spaltenorientierte Datenbanksysteme (Wide-Column-Store)

Der Aufbau von spaltenorientierten Datenbanksystemen bietet vor allem Vorteile bei der Analyse der Daten, das Einfügen neuer Datensätze ist gegebenenfalls etwas aufwendiger, da die Informationen eines Datensatzes erst in die Gesamtinformation eingefügt werden müssen.[6]
Da in Bezug auf den Mehrkampf aber immer wieder neue Datensätze angelegt werden, wird eine Umsetzung in einem spaltenorientierten Datenbanksystem nicht weiter in Betracht gezogen.

2.4.3 Key/Value-Datenbanksysteme (Key-Value-Stores)

Prinzipiell kann jeder Wettkampf mit all seinen Einzelheiten als Wert (Value) über einen Schlüssel (Key) abgelegt werden. Unter diesem Aspekt wären Key/Value-Datenbanksysteme eine geeignete Lösung. Einschränkend wirkt sich aber die Limitierung des Systems bei der Abfrage von

[6] [Edl10, Seite 53]

bestehenden Informationen aus und muss in der genaueren Betrachtung dahingehend untersucht werden, denn generell könnten fehlende Funktionalitäten auch auf Anwendungsseite gelöst werden.

2.4.4 Dokumentenorientierte Datenbanksysteme (Document Store)

Das Konzept zur Ablage von Informationen in einem Dokument spiegelt die Anforderungsgliederung aus Abschnitt 2.3 wieder und scheint damit ideal geeignet, um die Mehrkampfdatenbank zu verwalten. Auch die Möglichkeit zur Indexierung von eingebetteten Daten und der damit verbundenen schnellen Abfrage erfüllt eine der Grundanforderungen aus Abschnitt 2.3 zur Erstellung diverser Auswertungen.
Inwieweit die Konsistenz von Daten, die in mehreren Dokumenten hinterlegt sind, gewahrt bleiben kann, muss eingehender betrachtet werden.

2.5 Fazit

Nach einer ersten Betrachtung kommen die spaltenorientierten Datenbanksysteme für eine Umsetzung nicht in Frage, da das Einfügen, Ändern und Löschen von Daten einen nicht unerheblichen Anteil bei der vorgesehenen Datenhaltung haben wird und dies als möglicher Schwachpunkt in Abschnitt 2.4.2 aufgeführt wurde.

Die anderen drei Datenbanksystemgruppen scheinen auf den ersten Blick durchaus geeignet und werden im nachfolgenden Kapitel eingehender betrachtet.

3 Eine weitergehende Betrachtung der gewählten Systeme

In diesem Kapitel werden die verbliebenen Datenbankgruppen eingehender auf ihre Eignung zur Umsetzung einer leichtathletischen Mehrkampfdatenbank geprüft.
Im ersten Abschnitt werden einige Eigenschaften nichtrelationaler Datenbanksysteme vorgestellt und ihre Notwendigkeit für die Umsetzung der Mehrkampfdatenbank geprüft.
Im zweiten Abschnitt wird die Unterstützung der notwendigen Eigenschaften durch die einzelnen Datenbankgruppen geprüft. Abschließend wird in einem Fazit die am besten geeignet erscheinende Datenbankgruppe ausgewählt.

3.1 Die Eingrenzung der notwendigen Eigenschaften

In Bezug auf Datenbanksysteme werden immer wieder einige Eigenschaften genannt, deren Nutzen für die Umsetzung der Mehrkampfdatenbank in diesem Abschnitt geprüft wird.
Die hier betrachteten Themen sind

- das Speichern zusammenhängender Informationen (Aggregate, Transaktionen),
- das Umsetzen neuer Anforderung an die Datenhaltung (Schemafreiheit),
- der Grad der Datenkonsistenz,
- das Verwalten großer Datenmengen (Datenverteilung) und
- die Bearbeitung vieler gleichzeitiger Anfragen (Datenspiegelung).

3.1.1 Aggregate - die Bündelung zusammenhängender Daten

Ein Problem der relationalen Datenbanken ist, dass zusammengehörige Informationen häufig auf mehrere Relationen aufgeteilt sind. Diese Aufteilung der Informationen erschwert oft das

Erkennen zusammenhängender Daten in den verschiedenen Relationen und wird als "Impedance Mismatch" (siehe Abbildung 3.1) bezeichnet.

Das 'Impedance Mismatch'-Problem

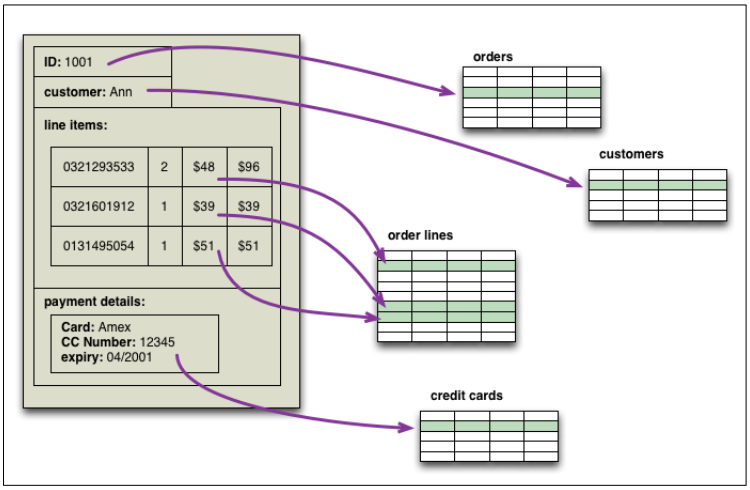

Abbildung 3.1: Das 'Impedance Mismatch'-Problem: Eine Information, beispielsweise eine Rechnung (links), wird über verschiedene Relationen aufgeteilt abgelegt. Bei einer Abfrage muss die Information erst wieder zusammengesucht werden. *Quelle:* [Fow15]

Um diese Informationen wieder zu einer Einheit zusammenzufügen, sind entweder mehrere Einzelabfragen notwendig (bei denen viele unnötige Informationen mit übertragen werden) oder die Relationen werden durch JOIN-Operationen verbunden. Je nachdem, wie hoch der Verteilungsgrad der angefragten Information ist, sind viele JOIN-Operationen notwendig, deren Reihenfolge erhebliche Unterschiede in der Laufzeit ergeben können.

Das relationale Modell der leichtathletischen Mehrkampfdatenbank

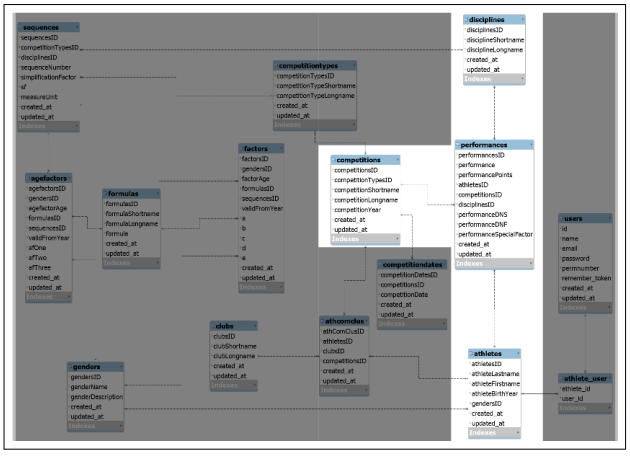

Abbildung 3.2: Die Grundinformationen zu einem Wettkampf sind über verschiedene Relationen verteilt und müssen durch JOIN-Operationen zusammengeführt werden. Allein um die Informationen zu einem Wettkampf, den erbrachten Leistungen, die dazugehörigen Disziplinen und Athleten zu ermitteln müssen die vier hervorgehobenen Relationen in einer Abfrage verknüpft werden.

Bei der Darstellung von leichtathletischen Mehrkämpfen werden sehr häufig die Informationen aus verschiedenen Relationen zusammengesucht, dementsprechend müssen immer wieder JOIN-Operationen durchgeführt werden.
Um aus dem relationalen Modell in Abbildung 3.2 die Informationen zu den Disziplinen, den erbrachten Leistungen und den teilnehmenden Athleten eines bestimmten Wettkampfes zu erhalten, müsste der notwendige SQL-Befehl in etwa dem in Abbildung 3.3 gleichen.

SQL-Befehl zur Abfrage eines Wettkampfes

```
SELECT * FROM competitions JOIN performances JOIN athletes JOIN disciplines WHERE
↪   competitions_id = 1;
```

Abbildung 3.3: Über den SQL-Befehl werden alle Informationen (Wettkampf-, Leistungs-, Athleten- und Disziplindaten) abgefragt und durch drei JOINs zusammengeführt.

Bei einer aggregierten Speicherung können alle notwendigen Informationen über eine einfache Abfrage ermittelt werden. In Abbildung 3.4 sind die Informationen in Wettkampf-Aggregaten (Competition) zusammengefasst. Ein Abfrage zu einem Wettkampf liefert automatisch alle darunterliegenden Daten mit. Im Gegensatz zu einer relationalen Lösung werden keine JOIN-Operationen benötigt.

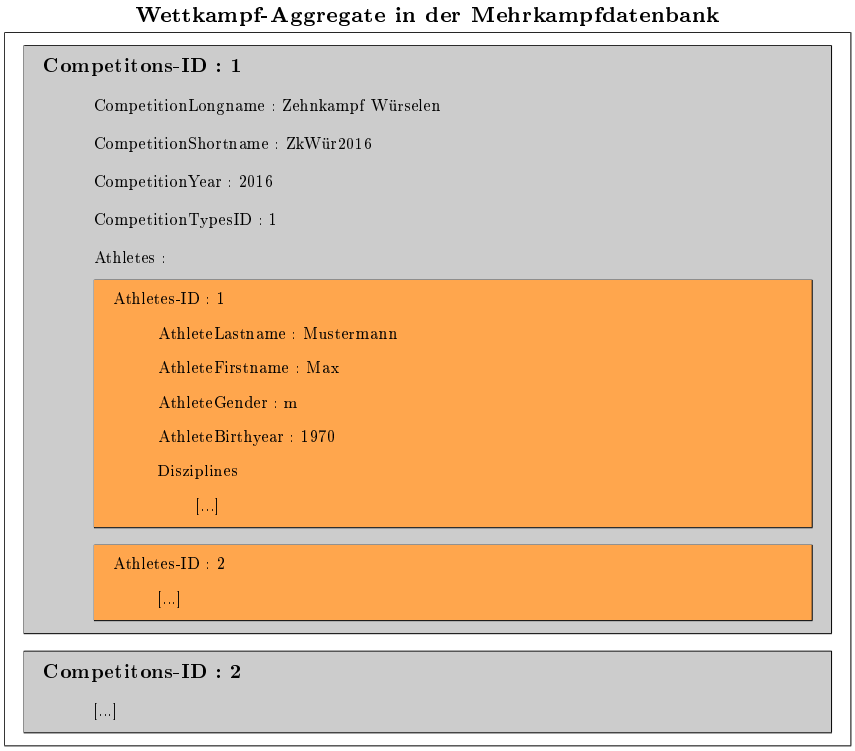

Abbildung 3.4: Beispielhafte Darstellung der Wettkampf-Aggregate in der nichtrelationalen Mehrkampfdatenbank. Eine Abfrage nach einem bestimmten Wettkampf liefert alle zugehörigen Informationen zu Athleten, Disziplinen und Leistungen mit.

Die Hierarchie der Informationen im Aggregat kann insofern wichtig sein, wenn sie nicht der Struktur einer Suchanfrage entspricht.
Wenn in den Aggregaten in Abbildung 3.4 nach einem bestimmten Athleten gesucht wird und das Ergebnis der Anfrage dem Aufbau in Abbildung 3.5 entsprechen soll, sind Umformungen der betroffenen Aggregate durch Aggregationsoperationen notwendig.[1]
Hierbei wird die Abfolge und Hierarchie der Informationen umgestellt, um sie dem gewünschten Ergebnis anzupassen.

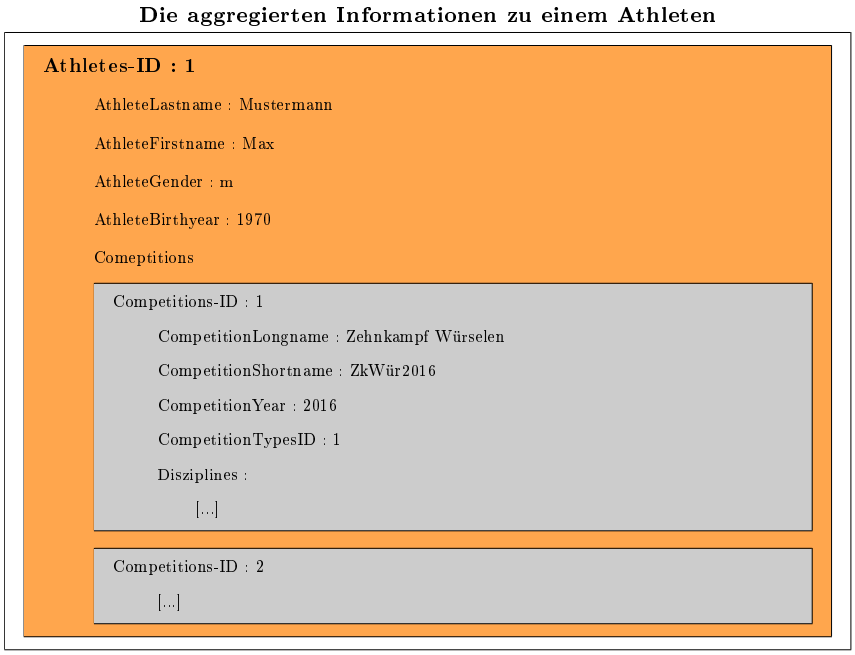

Abbildung 3.5: Um die Informationen zu einem Athleten aus den Aggregaten in Abbildung 3.4 zu erhalten sind MAP-REDUCE-Operationen notwendig.

Die Aggregate haben einen großen Vorteil bei der Planung und Umsetzung von Clustern, denn zusammengehörige Daten sind bereits zusammen und müssen nicht aufwendig über mehrere Server zusammengesucht werden. Ob eine Verteilung der Aggregate auf verschiedene Cluster notwendig ist, wird in Abschnitt 3.1.4 genauer betrachtet.

3.1.2 Die Einbindung neuer Anforderungen (Schemafreiheit)

Ein häufiges Problem bei der Planung und Umsetzung von Speicherstrukturen ist die Frage, welche Informationen möglicherweise zu einem späteren Zeitpunkt relevant sind. Im Gegensatz zu relationalen Datenbanksystemen, die ein striktes Schema oder Modell voraussetzen, haben

[1] [SF13, Kapitel 7]

NoSQL Datenbanksysteme keine solche Restriktion. Sie werden daher gemeinhin als schemafrei bezeichnet.

Die Tatsache, dass das Datenbanksystem schemafrei ist, bedeutet allerdings nicht, dass es kein Schema für die gespeicherten Daten gibt. Das Schema wird vielmehr implizit außerhalb des Datenbanksystems festgelegt.

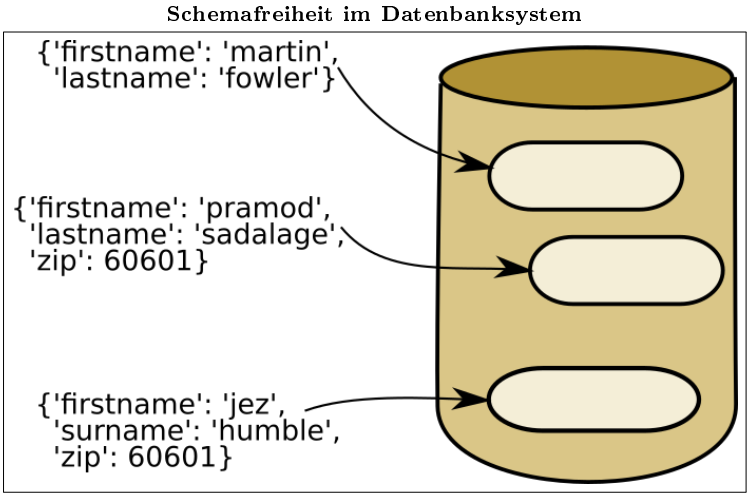

Abbildung 3.6: In einem schemafreien System können die Daten in beliebiger Form und ohne Vorgaben abgelegt werden. *Quellenvorlage: [Fow13]*

In Abbildung 3.6 werden Daten in unterschiedlichen Konstellationen an das Datenbanksystem übergeben.

Das Datenbanksystem verwaltet lediglich die Datensätze, führt jedoch keinerlei Prüfungen auf die Einhaltung eines Schemas durch, da ein solches im Datenbanksystem nicht existiert.

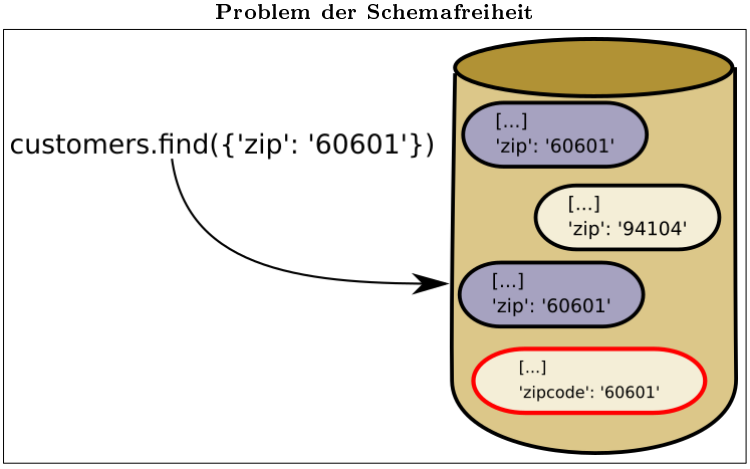

Abbildung 3.7: Trotz Schemafreiheit im Datenbanksystem, muss außerhalb ein einheitliches Schema existieren. Andernfalls können die Daten nicht korrekt abgerufen werden. Der unterste Datensatz wird von der Anfrage nicht erfasst. *Quellenvorlage: [Fow13]*

Ein Nachteil eines impliziten Schemas ist die Nutzung des Datenbanksystems über verschiedene Anwendungen hinweg (Integration Database), da jede Anwendung das implizite Schema korrekt umsetzen muss. Andernfalls kann es passieren, dass eine Anwendung nicht in der Lage ist, Datensätze auszulesen. In Abbildung 3.7 ist ein solcher Problemfall dargestellt, die Suchanfrage liefert nicht alle zu erwartenden Datensätze.

In Bezug auf das Projekt stellt die Anforderungsbeschreibung in Kapitel 2.3 den endgültigen Umfang dar, daher ist das Vorhandensein von Schemafreiheit nicht notwendig. Sollte der Umfang der Anwendung in der Zukunft noch erweitert werden, könnte die Schemafreiheit eine erhebliche Arbeitserleichterung darstellen.

3.1.3 Die Zusicherung von Datenkonsistenz (Consistency)

Generell wird mit dem Begriff der Datenkonsistenz der Zustand von Datensätzen bei Abfragen verbunden. Während die Informationen aus dem System ausgeliefert werden, sollen keine anderen Operationen die Daten manipulieren können. Während dies beim Betrieb des Datenbanksystems auf einem einzelnen Server (Single-Server) tatsächlich die einzige Gefahr für Datenkonsistenz darstellt, entsteht bei der Verteilung der Daten (Multi-Server) noch ein weiteres Problem in Bezug auf Dateninkonsistenz.
Um diese Inkonsistenzprobleme zu lösen wurden im Laufe der Zeit viele Verfahren umgesetzt, die heute durch verschiedene Konzepte beschrieben werden.

"ACID" und "BASE" (Datenkonsistenz gegen -verfügbarkeit)

> "BASE is an acronym that is so bad, it makes ACID look meaningful!"
>
> — Martin Fowler[2]

Das Grundkonzept von ACID wurde durch GrayEnde der Siebziger erstmals ausführlicher beschrieben und etwas später von Haerder und Reuter mit dem entsprechenden Akronym versehen.[3,4]
Das Akronym BASE wurde von Brewer Ende der Neunziger erfunden und wird inzwischen von ihm selber als missverständlich angesehen.[5]

[2] [13]
[3] [Gra81]
[4] [HR83]
[5] [Bre12]

Beide Akronyme beschreiben Designphilosophien, die entweder auf Datenkonsistenz (ACID) oder Datenverfügbarkeit (BASE) ausgerichtet sind. Nach Brewer selbst, stellen beide die äußeren Enden einer Skala dar, auf der das Verhältnis zwischen Konsistenz von Daten (Consistency) und Verfügbarkeit von Daten (Availability) festgelegt werden kann.

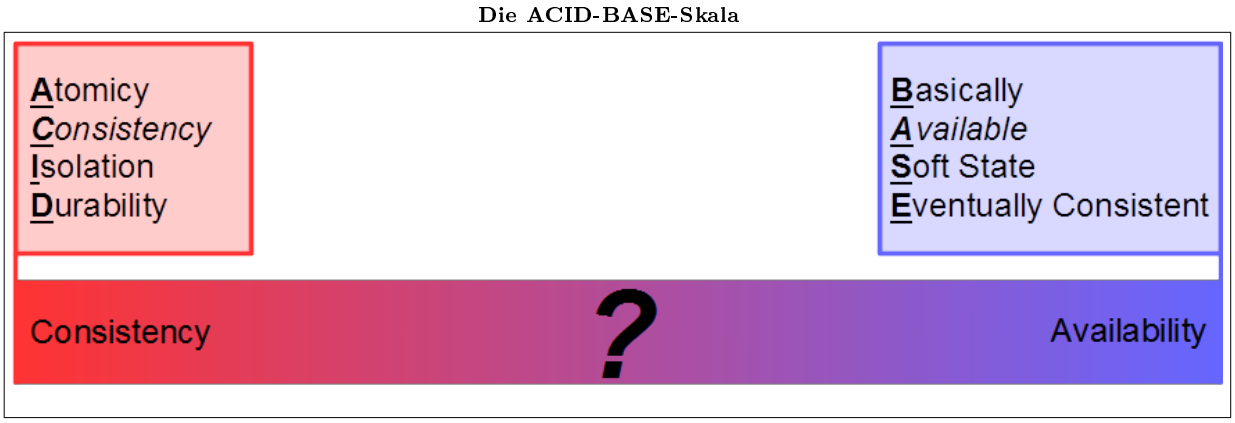

Abbildung 3.8: Zwischen ACID und BASE gibt es, vergleichbar mit einer ph-Skala, viele Abstufungen, die durch die beiden Konzepte nicht dargestellt werden können.

Irrtümlicherweise werden die Akronyme häufig als Synonym für Gruppen von Datenbanksystemen verwendt. ACID repräsentiert dabei die SQL-Systeme und BASE die NoSQL-Systeme. Diese Einteilung ist jedoch irreführend, da zum Beispiel Graphendatenbanksysteme zwingend das ACID-Konzept umsetzen müssen.[6]

Ein besserer Ansatz ist die Frage, ob die Daten auf mehrere Server verteilt oder dupliziert werden. Sofern dies der Fall ist, tendiert das verwendete Datenbanksystem zu BASE, ansonsten folgt die Umsetzung mehr dem ACID-Ansatz. Da aber auch in diesen Fällen immer noch Abstufungen möglich sind, ist die Verwendung von ACID und BASE weiterhin leicht irreführend.

Das CAP-Theorem

Um die Jahrtausendwende wurde ein neuer Konzeptansatz durch Brewer beschrieben.

Im Gegensatz zu ACID und BASE werden beim CAP-Theorem drei Anforderungen (Consistency, Availability und Partition tolerance) gegenübergestellt, von denen immer nur zwei durch das System erfüllt werden können.[7,8]

Partition tolerance bezeichnet dabei die Ausfalltoleranz des Datenbanksystems bei einem Wegfall einzelner Teilknoten in einem verteilten System.

[6] [13]
[7] [FB99]
[8] [Bre12]

Während CA-Systeme (Consistency and Availability) nur bei Einzelserver-Lösungen, wo es keine Partition tolerance geben muss, umsetzbar sind, beschreiben CP- und AP-Systeme nur den Zustand bei einem Teilausfall aber nicht im Normalbetrieb.

PACELC-Theorem

Um die Schwachstellen des CAP-Theorems zu eliminieren, veröffentlichte Abadi in 2012 einen Artikel zum PACELC-Theorem.

> *"A more complete portrayal of the space of potential consistency tradeoffs for DDBSs can be achieved by rewriting CAP as PACELC (pronounced 'pass-elk'): if there is a partition (P), how does the system trade off availability and consistency (A and C); else (E), when the system is running normally in the absence of partitions, how does the system trade off latency (L) and consistency (C)?"* [9]

Es behandelt ausschließlich verteilte Systeme, beschreibt dabei aber das Verhalten des Systems bei einem Teilausfall **und** im Normalbetrieb. Beim PACELC-Theorem verwischt auch die binäre Unterscheidung zwischen Konsistenz und Verfügbarkeit der Daten und der Begriff der Antwortzeit (Latency) wird eingeführt.[10]

Konsistenzprobleme jenseits des Datenbankkonzepts

Im Allgemeinen sind alle modernen Datenbanksysteme bestrebt, ein Mindestmaß an Konsistenz zu gewährleisten.
Relationale Datenbanksysteme garantieren dieses Mindestmaß in Form von Transaktionen, die implizit einzelne Anfrage- und Manipulationsbefehle umfassen und explizit auch Befehlsketten einschließen können.
Nichtrelationale Datenbanksysteme bieten diese Konsistenzeigenschaft in Form von Aggregaten, allerdings wird dies nicht von allen unterstützt. Bezogen auf die vier großen Hauptgruppen fehlt den Graphendatenbanksystemen diese Eigenschaft. Die anderen Gruppen werden daher auch als 'Aggregate-Databases' bezeichnet.[11]

[9] [Aba12]
[10] [OPS15]
[11] [Fow15]

Abseits der Konsistenzproblematik innerhalb der Datenstruktur, gleich ob das gewählte System ACID, BASE, CAP oder PACELC nutzt, können Konsistenzprobleme auch von außen auf das System einwirken. Solche Konsistenzprobleme sind nicht für das Datenbanksystem erkennbar und können daher auch nicht auf dieser Ebene korrigiert oder gelöst werden.
Eines der klassischen (und wohl am häufigsten auftretenden) Probleme ist das des 'Lost-Update', welches in Mehrbenutzersystemen anzutreffen ist. Dabei überschreiben Benutzer die Änderungen anderer Benutzer, wie in Abbildung 3.9 gezeigt, ohne sich dessen bewusst zu sein.

Aggregate können diese Problematik nicht lösen, transaktionsbasierte relationale Systeme wären dazu theoretisch in der Lage, wenn die Transaktion auf die gesamte Befehlsfolge ausgedehnt wird. Da aber dies eventuell zu einer längeren Sperrung des Datenbanksystems für andere Zugriffe führt, wenn zum Beispiel ein Benutzer eine Transaktion startet und erst nach dem Mittagessen wieder beendet, haben die Datenbanksysteme den Zeitraum, in dem eine Transaktion geöffnet ist, begrenzt.[12]

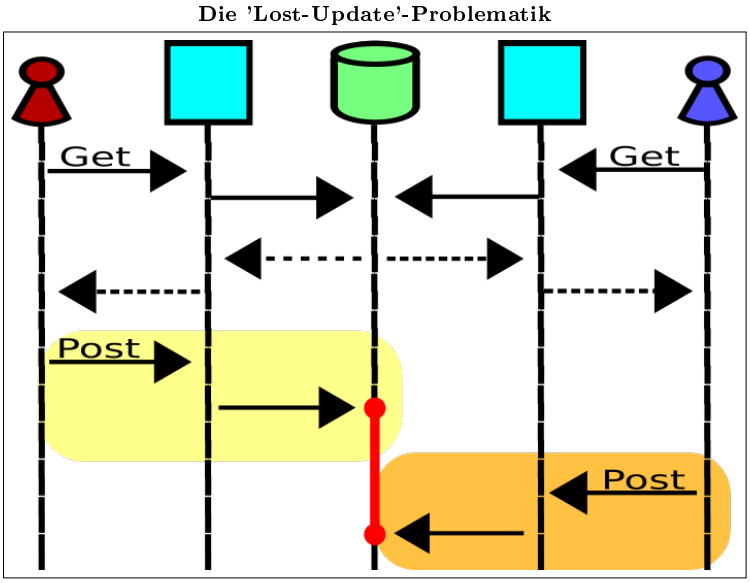

Abbildung 3.9: Ein Beispiel für die 'Lost-Update'-Problematik im Mehrbenutzerbetrieb. Benutzer Blau überschreibt das Update von Benutzer Rot.

Eine Lösungsmöglichkeit ist die Verwendung von sogenannten Offline-Locks, die eine Sperrung einzelner Datenbankbereiche für einen längeren Zeitraum ermöglichen.
Dabei werden ein pessimistischer und ein optimistischer Ansatz unterschieden. Deren Einsatz hängt davon ab, ob ein Konflikt sehr wahrscheinlich (pessimistisch) oder eher unwahrscheinlich

[12][Fow03, Seite 426]

(optimistisch) ist.
Beide Ansätze müssen außerhalb des Datenbanksystems gesteuert werden. Sie können entweder ein bestehendes Konzept im Datenbanksystem verwenden oder ein eigenes Modell etablieren.[13]

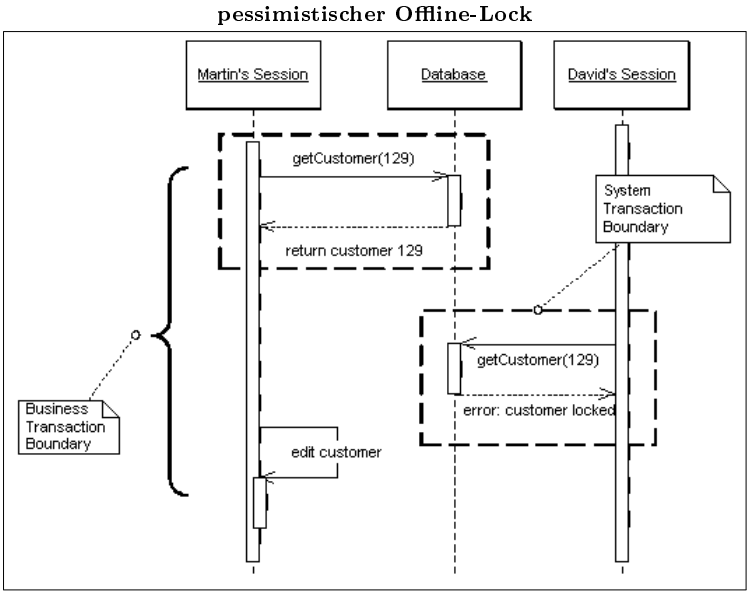

Abbildung 3.10: Beispiel für einen pessimistischen Offline-Lock: Vor der Abfrage wird durch Martins Sitzung ein Lock gesetzt, der einen Zugriff, egal ob lesend oder schreibend, durch andere Sitzungen, in diesem Fall Davids, verhindert. Nach Beendigung aller notwendigen Aufgaben wird von Martins Sitzung der Lock wieder aufgelöst, um den Zugriff wieder freizugeben.

Der pessimistische Ansatz, in Abbildung 3.10 beispielhaft dargestellt, ähnelt sehr stark einer Transaktion, da sämtliche anderen Zugriffe, auch Lesezugriffe, auf die betroffenen Daten unterbunden werden. Die Verwendung sollte daher auf solche Vorgänge beschränkt werden, für die diese Nutzung unumgänglich ist. Eine zu häufige Verwendung schränkt die generelle Verfügbarkeit für die Benutzer stark ein.

Bei der Verwendung von optimistischen Offline-Locks sollte immer bedacht werden, dass ein vom Benutzer angestoßenes Update aufgrund des Updates eines anderen Benutzers, wie in Abbildung 3.11 gezeigt, nicht ausgeführt werden kann.
Dem Benutzer sollte daher immer per Rückmeldung der Erfolg oder Misserfolg einer Transaktion mitgeteilt werden. Weitere Techniken und Lösungsansätze zur Überarbeitung und erneuten Übermittlung der Informationen sind optional.

[13] [Fow03, Kapitel 16]

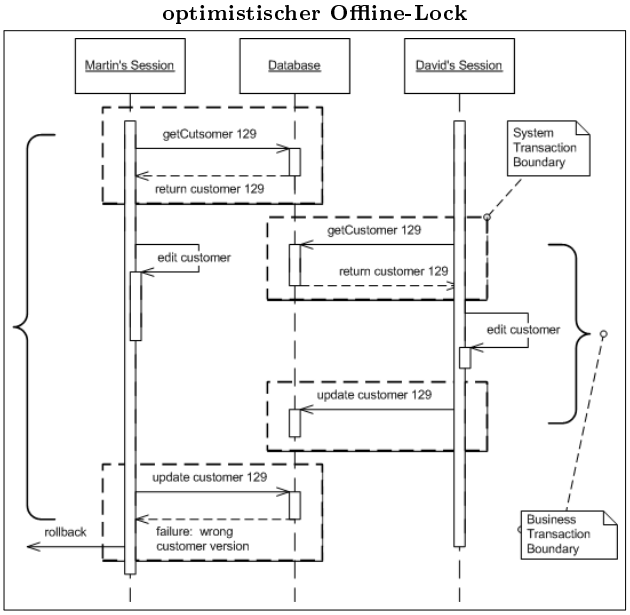

Abbildung 3.11: Beispiel für einen optimistischen Offline-Lock: Das Update von Martin wird abgelehnt, da die Versionsnummer der von ihm abgerufenen Daten nicht mehr mit der aktuellen Versionsnummer übereinstimmt. Der Grund hierfür ist das erfolgreich durchgeführte Update von David.

3.1.4 Die Aufteilung der Daten (Sharding)

Die Notwendigkeit zur Aufteilung der Daten auf mehrere Server hängt in den meisten Fällen vom Umfang der Datenmenge ab. Daneben kann eine Verteilung der Daten auch nach Zugehörigkeit der Informationen zu einer geographischen Region erfolgen, um die Gesamtmenge der Anfragen auf die jeweiligen Regionen zu verteilen.

Wenn ein Datenbanksystem zur Speicherung sozialer Kontakte genutzt wird, müssen die anfallenden Daten auf weltweit verfügbare Server verteilt werden. In Abbildung 3.12 sind zwei Möglichkeiten dargestellt, wobei die Verteilung nach regionaler Zugehörigkeit zum einen die Latenzzeiten bei Anfragen klein hält, zum anderen liegen in den meisten Fällen die sozialen Kontakte einer Person in derselben Region, wodurch auch hier kleine Latenzzeiten möglich sind.

Eine alphabetische Verteilung könnte einen europäischen Benutzer zwingen, sich mit dem australischen Server verbinden zu müssen, sein bester Freund hingegen wird auf dem südamerikanischen Server verwaltet.

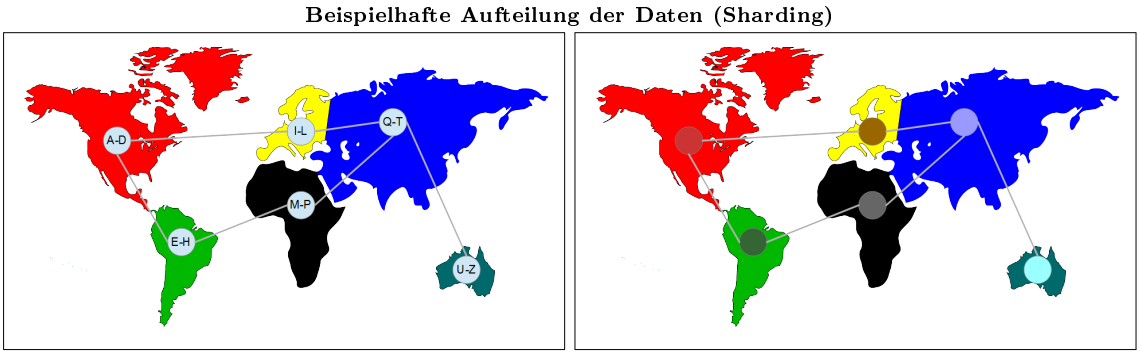

Abbildung 3.12: Datenbanksystem "People of the world": In der linken Grafik werden die Daten alphabetisch über die weltweit vorhandenen Server verteilt. In der rechten Grafik sind die Daten nach Regionen auf die Server aufgeteilt.

Die in dieser Arbeit umgesetzte Mehrkampfdatenbank soll für die Datenhaltung in einem einzelnen Verein genutzt werden.
Ausgehend von der in Kapitel 2.3 dargestellten Struktur, umfasst ein Wettkampfdatensatz alle Informationen zum Wettkampf selbst, den stattfindenden Disziplinen und den teilnehmenden Athleten, inklusive ihrer Leistungen und Punktewertungen.
Der Umfang an Daten wird anhand der nachfolgenden Annahmen geschätzt.

- Es finden 12 Wettkämpfe pro Jahr statt
- An jedem Wettkampf beteiligen sich Wettkämpfer aus 10 Vereinen
- Aus jedem Verein treten 20 Mitglieder an

Aus dieser Annahme ergibt sich eine Gesamtmenge von 2.400 Teildatensätzen von Athleten die in 12 Dokumenten aufgeteilt sind. Ausgehend von der Annahme, dass auch häufig nach bestimmten Athleten gesucht wird, muss jeder Teildatensatz auf Übereinstimmung geprüft werden. Damit eine Größenordnung von 100.000 Athleten entstehen kann, müsste ein Zeitraum von über 40 Jahren verstreichen.

Sofern keine nachträgliche Erweiterung des Datenmodells die Menge der zu speichernden Informationen massiv erhöht, ist eine Verteilung des Datenbestandes auf mehrere Serverknoten nicht notwendig.

3.1.5 Die Spiegelung der Daten (Replication)

Eine Spiegelung der Daten erfolgt meist dann, wenn die Anzahl gleichzeitiger Anfragen durch die Benutzer ein einzelnes Serversystem überlasten würde, die Gesamtmenge der Informationen

aber weiterhin durch ein einzelnes System gehandhabt werden könnte.

Die Anzahl der gleichzeitigen Anfragen ist schwerer abzuschätzen, da dies von verschiedenen Faktoren abhängt.

- Sind Informationen für alle zugänglich oder gibt es einen abgegrenzten Benutzerkreis.
- Wie viele Benutzer operieren durchschnittlich und in Spitzenzeiten gleichzeitig auf dem Datenbestand.

Um trotzdem eine Aussage zu treffen, kann aus dem Einsatzzweck des Systems und vergleichbaren verfügbaren Systemen eine Annahme zu den beiden oben genannten Punkten gemacht werden.
Teile der gespeicherten Informationen sind sicherlich der Allgemeinheit zugänglich, insbesondere die Ergebnisübersichten eines Wettkampfes. Insofern ist der Benutzerkreis nicht abgrenzbar.
Tatsächliche Benutzer sind im Allgemeinen direkt involvierte Teilnehmer und ein kleiner Kreis von sportlich interessierten Beobachtern. Aus diesen beiden Gruppen lässt sich eine Annahme über die Größenordnung der gleichzeitigen Anfragen treffen.
Insbesondere in den Tagen nach einem Wettkampf werden gehäuft Anfragen durchgeführt, da die teilnehmenden Sportler ihre Ergebnisse einsehen möchten. Daneben werden auch noch einige weitere interessierte Benutzer zugegen sein.
Ausgehend von einer Anzahl von 200 Teilnehmern pro Wettkampf und genauso vielen interessierten Benutzern, sollte ein einzelner Server für die Datenhaltung ausreichen.

3.1.6 Zusammenfassung

Zum Abschluss dieses Abschnittes werden die zuvor genannten Eigenschaften nochmals auf ihre Notwendigkeit in der Projektumsetzung zusammengefasst.

Sharding

Die prognostizierte Datenmenge sollte von einem einzelnen Serversystem verwaltbar sein. Daher ist das Sharding für die Projektumsetzung unwichtig.

Replication

Die voraussichtliche Anzahl von gleichzeitigen Benutzerzugriffen sollte so gering ausfallen, dass

eine Spiegelung (Replication) der Daten auf mehrere Server keinen nennenswerten Geschwindigkeitsvorteil bringt.

Aggregate

Der momentane Aufbau der Anwendungsoberfläche lässt zwei Ansichten auf die Gesamtdatenmenge zu. Der Aufruf der Wettkampfübersicht mit seinen Athleten und Leistungen entspricht dabei dem gewünschten Aufbau aus Abschnitt 2.3 und ließe sich wunderbar als Aggregat abbilden. Die Erstellung der Athletenübersicht dagegen könnte nur durch ein Zusammensuchen aus den einzelnen Aggregaten erfolgen. Dieser Vorgang darf nicht sehr viel komplexer und zeitintensiver ausfallen, als ein vergleichbarer JOIN-Vorgang im RDBMS.

Schemafreiheit

Die Schemafreiheit ist für die Projektumsetzung nicht zwingend notwendig. Sie bietet aber einen erheblichen Vorteil, sofern zu einen späteren Zeitpunkt doch noch eine Erweiterung der Datenstruktur gewünscht wird.

Konsistenz und Verfügbarkeit

Weder Sharding noch Replication sind für die Projektumsetzung notwendig. Daher gibt es auch kein verteiltes System (Distributed Database System - DDBS). Das Prinzip der Verfügbarkeit, wie es nach CAP, BASE und PACELC definiert wird, ist damit unerheblich. Nach PACELC wäre dies ein PC/EC System[14], bei dem die Konsistenz der Daten Vorrang hat. Es entspricht dem ACID Modell der RDBMS.

Zwei zusätzliche Anforderungen, die sich aus der Benutzung der Mehrkampfdatenbank im Praxisbetrieb ergeben würden, sind:

CRUD-Operationen

CRUD-Operationen (Create, Read, Update & Delete) haben einen hohen Stellenwert, da unter anderem ständig neue Daten hinzugefügt werden. Insofern sollte ein Datenbanksystem diese Operationen generell unterstützen und keine Performanzprobleme bei ihrer Ausführung verursachen.

Sprachumfang

Die Abfragesprache des Datenbanksystems sollte die unterschiedlichsten Abfrage-Konstellationen

[14] [OPS15]

unterstützen. Neben klassischen Suchanfragen über indexierte Schlüssel, sollte auch eine direkte Suche im Datenbestand möglich sein.

3.2 Die Begutachtung der verbleibenden Datenbanksysteme

Auf Grundlage der im vorhergehenden Abschnitt definierten und geprüften Eigenschaften, werden die drei noch verbliebenen nichtrelationalen Datenbankgruppen (Graphendatenbank-, Key/Value- und dokumentenbasierte Datenbanksysteme) auf ihre Eignung hin untersucht.

3.2.1 Graphendatenbanksysteme (Graphdatabase)

Sharding

Da ein Graphdatenbanksystem seinen Informationen, stärker als ein RDBS, aufteilt und in Knoten und Kanten ablegt, ist eine horizontale Skalierung, also die Verteilung der Informationen auf mehrere Server (Sharding) nur schwer umsetzbar.[15] Auch wenn diese Eigenschaft für das Projekt nicht notwendig ist, sollte die Entscheidung für ein Graphendatenbanksystem wohlüberlegt sein. Eine später vielleicht doch notwendige Umstellung auf ein verteiltes System ist nur über eine komplette Überführung in ein anderes Datenbanksystem möglich.

Replication

Die Spiegelung der Daten auf mehrere Server (Replication) ist für die Projektumsetzung nicht notwendig, ließe sich aber in Graphendatenbanksystemen nachträglich einfügen, da diese Eigenschaft unterstützt wird.

Aggregate

Aufgrund der Verteilung der Informationen auf verschiedene Knoten und Kanten, können Graphendatenbanksysteme keine Aggregate bilden. Durch die weitreichende Verknüpfung der Informationen im Graphen können jedoch die verschiedensten Abfragen durchgeführt werden. Jede Abfrage erzeugt ihr eigenes "Aggregat". Ausschlaggebend für eine Suchanfrage ist das schnelle Auffinden des jeweiligen Startknotens im Graphen. Durch die Indexierung verschiedener Knotengruppen, anhand ihrer Eigenschaften (Properties) spezifiziert, kann ein schnelles Auffinden des Startknotens sichergestellt werden.

[15] [11]

Schemafreiheit

Wie alle anderen NoSQL-Systeme sind auch Graphendatenbanksysteme schemafrei. Durch die weitläufige Aufteilung der Informationen auf Knoten und Kanten ist es noch schwerer, das implizite Schema im Datenbestand zu erkennen. Dies erfordert, mehr noch als bei den anderen Gruppen, die klare Dokumentation des impliziten Schemas.

Datenkonsistenz

Graphendatenbanksysteme setzen zwingend die Konsistenzeigenschaften nach ACID voraus. Grund hierfür ist die Aufteilung der Daten, die im Vergleich zu relationalen Systemen noch stärker ausfällt.[16]

CRUD-Operationen

Obwohl die Graphendatenbanksysteme in Bezug auf die bisher betrachteten Eigenschaften alle Anforderungen erfüllen, sind sie mit einem nicht unerheblichen Manko behaftet.
Das Ändern bestehender Daten ist in Graphendatenbanksystemen mit einem nicht gewissen Aufwand verbunden, wenn viele Knoten gleichzeitig durch die Änderung betroffen sind.[17] Abhängig davon, wie stark das relationale Modell weiter aufgeteilt wird, könnte dies ein Problem darstellen.

Sprachumfang

Der Sprachumfang ist stark vom Einsatzschwerpunkt eines Graphendatenbanksystems abhängig. Weit verbreitete Systeme, wie Neo4j, bieten einen Sprachumfang, der in Teilen dem der relationalen Systeme entspricht. Zusätzlich können auch analytische Anfragen formuliert werden, die in anderen NoSQL- oder relationalen Systemen nur schwer umsetzbar sind.

3.2.2 Key/Value-Datenbanksystemem (Key-Value-Store)

Sharding und Replication

Sowohl das Verteilen (Sharding), wie auch das Spiegeln (Replication) von Daten, wird durch die Key/Value-Datenbanksysteme unterstützt.

Aggregate

Über die Struktur der gespeicherten Informationen (Values) werden grundsätzlich keinerlei

[16] [Fow15]
[17] [SF13, S. 121]

Vorgaben gemacht, somit können auch aggregierte Informationen in Form von JSON-Objekten abgelegt werden.

Schemafreiheit

Da die Key/Value-Systeme keinerlei Vorgaben für den Aufbau und Inhalt der Informationen (Values) vorgeben, kann jede Datenstruktur gespeichert werden.

Datenkonsistenz

Key/Value-Systeme unterstützen die Konsistenzanforderungen in Bezug auf atomare Änderungen innerhalb eines Key/Value-Paares. Die Änderung mehrerer Paare kann nur durch eine Sperrung (Locks), sofern unterstützt, konsistent gehalten werden.

CRUD-Operationen

Das Erfassen, Ändern und Löschen von Informationen ist sehr performant durchführbar, sofern der dafür notwendige Schlüssel (Key) vorliegt.

Sprachumfang

Grundsätzlich unterstützt das Key/Value-Prinzip lediglich Operationen in Bezug auf den Schlüssel (Key). Diese sind durch Hashing (siehe Abbildung 3.13) in Laufzeiten der Ordnung O(1) durchführbar.

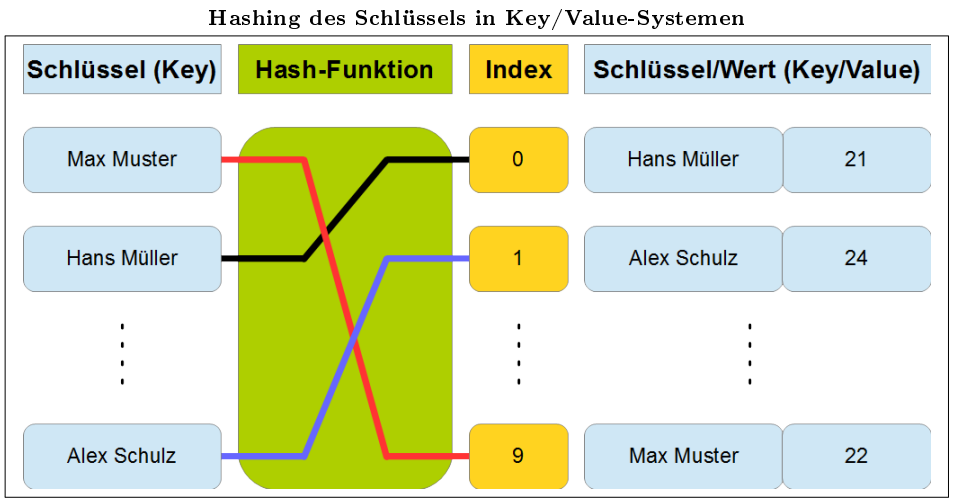

Abbildung 3.13: Die Schlüssel (ganz links) werden mittels einer Hash-Funktion umgewandelt. Diese Hashes dienen als Index für die Ablage und das Suchen der eigentlichen Schlüssel und Werte (ganz rechts).

Ohne einen Schlüssel können direkte Anfragen auf die gespeicherten Informationen nicht effizient durchgeführt.

Zwar bieten einige Umsetzungen von Key/Value-Systemen erweiterte Suchmethodiken, dies kann aber nicht generell garantiert werden, zudem schwankt der Umfang der Methoden von Umsetzung zu Umsetzung.
Da in der Projektumsetzung auch Anfragen auf Dateninhalte durchgeführt werden sollen, sind Key/Value-Systeme nur bedingt geeignet, sofern sie eine erweiterte Anfragesprache anbieten.

3.2.3 Dokumentenbasierte Datenbanksysteme (Document-Store)

Sharding und Replication

Dokumentenbasierte Datenbanksysteme unterstützen sowohl das Verteilen (Sharding), wie auch das Spiegeln (Replication) von Daten. Insbesondere das Sharding ist durch die Bündelung zusammenhängender Informationen in Aggregaten sehr einfach durchzuführen.

Aggregate

Ähnlich wie die Key/Value-Systeme können auch dokumentenbasierte Datenbanksysteme Informationen in Form von Aggregaten abspeichern. Dabei werden meist Datentypen für die Speicherung der Informationen genutzt. Häufig verwendete Formate sind JSON/BSON oder XML.

Schemafreiheit

Auch in Bezug auf Schemafreiheit gibt es keinen Unterschied zwischen den Key/Value- und den dokumentenbasierten Datenbanksystemen.

Datenkonsistenz

Einzelne Dokumente werden grundsätzlich als eine Einheit angesehen und somit auch vom System atomar behandelt. Dadurch wird Datenkonsistenz innerhalb der Dokumente gewährleistet. Datenmanipulationen, die mehrere Dokumente betreffen, können nur über Sperren (Locks) den Konsistenzanforderungen gerecht werden. Zu beachten ist, dass das Sperrprinzip kein genereller Bestandteil der dokumentenbasierten Datenbanksysteme darstellt und daher nicht zwingend von jeder Umsetzung unterstützt werden muss.

CRUD-Operationen

Das Einfügen neuer Daten, aber auch das Ändern und Löschen bestehender Daten ist in dokumentenbasierten Systemen ohne Probleme möglich. In den beiden letzteren Fällen kann dies sowohl über den Schlüssel, wie auch über Teilinformationen geschehen.

Sprachumfang

Die Variation zur Ausführung von Operationen ist sehr umfangreich. Neben den von Key/Value-Systemen unterstützten Operationen über einen Schlüssel, können auch Anfragen und Anweisungen in Bezug auf die gespeicherten Informationen durchgeführt werden. In den Informationen können auch weitere geschachtelte Dokumente (nested documents) existieren. Generell unterstützen aktuelle Systeme auch Operationen innerhalb dieser geschachtelten Informationen. Um schnelle Abfragen zu ermöglichen, ist das Erstellen von Indexen auf Daten möglich.

3.3 Fazit

Von den drei betrachteten Datenbankgruppen haben die Key/Value-Datenbanksysteme ein besonders großes Manko, die sie lediglich Datenbankoperationen über die Schlüssel zusichern können.
Es besteht zwar grundsätzlich die Möglichkeit, dieses Manko durch entsprechende Konzepte auf Anwendungsseite zu kompensieren, der dabei entstehende hohe Datenaustausch zwischen Anwendung und Datenbankserver wäre aber unerwünscht.

Graphendatenbanksysteme bieten mit ihrer Struktur zwar den höchsten Grad an Flexibilität in Bezug auf Datenanfragen, dafür muss aber, aufgrund der starken Verteilung der Informationen auf viele Knoten, das Konzept auch auf die praktische Anforderung übertragen werden. Es muss vorab geklärt werden, inwieweit die zusammenhängenden Informationen aus der Praxis im Graphen auf einzelne Knoten verteilt werden.
Aufgrund des zeitlichen Umfangs dieses Projektes wird daher auf den Einsatz eines Graphendatenbanksystems verzichtet.

Die dokumentenbasierten Datenbanksysteme speichern zusammenhängende Informationen als Einheit. Dieser Ansatz unterstützt die Funktionsweise der Benutzeroberfläche, die ebenfalls zusammengehörige Informationen als Einheit darzustellen versucht. In Bezug auf das umgesetzte Projekt einer Mehrkampfdatenbank können auch Teilinformationen innerhalb der Dokumente ermittelt werden.

Als Umsetzung für die leichtathletische Mehrkampfdatenbank wird daher ein dokumentenbasiertes Datenbanksystem gewählt.

4 Die Verwendung des ausgewählten Systems

In diesem Kapitel werden im ersten Abschnitt verschiedene Softwarelösungen dokumentenbasierten Datenbanksysteme vorgestellt und verglichen. Danach erfolgt die Auswahl einer Softwarelösung für die praktische Umsetzung der leichtathletischen Mehrkampfdatenbank in einem NoSQL-System.
Abschließend wird kurz auf einige Grundentscheidungen bei der Erstellung eines Datenmodells eingegangen.

4.1 Die Bestimmung der Softwarelösung

Die Auswahl an dokumentenorientierten Datenbanksystemen ist recht umfassend. Auf der Internetseite DB-Engine Ranking werden allein 45 verschiedene Ausführungen (Stand Dezember 2016) genannt.[1]

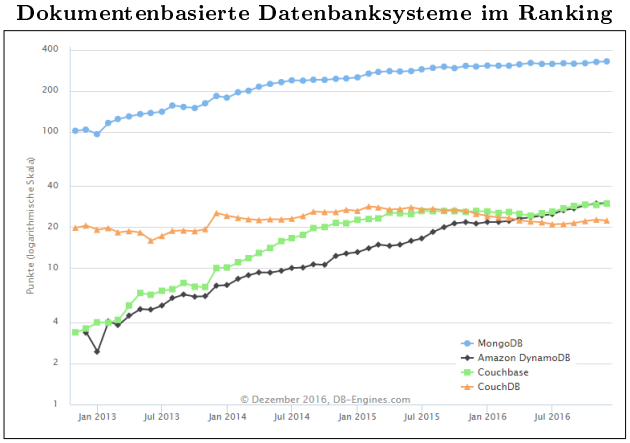

Abbildung 4.1: Die vier führenden dokumentenbasierten Datenbanksysteme im Ranking von 11.2012 bis 12.2016. Grundlage ist u.a. die Häufigkeit der Nennung in Suchanfragen.
Quelle: [16a] (Stand:06.12.2016)

[1] [16a]

Da eine umfassende Betrachtung und Einschätzung aus zeitlichen Gründen nicht möglich ist, wird als Vorselektion die Verbreitung der Datenbanksysteme herangezogen. Die Grundidee hierbei ist die Tatsache, dass zu einem stark verbreiteten System, auch wenn es vielleicht nicht die idealste Lösung sein mag, eine Vielzahl an Informationen, Hilfestellungen und Lösungsansätzen zu finden ist.

4.1.1 Die möglichen Softwarelösungen

Auf Grundlage des Rankings aus Abbildung 4.1 werden die vier "stärksten" Datenbanksysteme für die engere Auswahl in Betracht gezogen.

- MongoDB von MongoDB Inc.
- Amazon DynamoDB von Amazon Web Services Inc.
- Couchbase von Couchbase Inc.
- CouchDB von The Apache Software Foundation

Die Auswahl wird durch einen Bericht der Analysten von Forrester[2] gestützt, der drei von vier genannten Datenbanksystemen (MongoDB, Amazon DynamoDB, Couchbase) als *"Leader"* aufführt.[3]

Eine von den Autoren durchgeführte, nicht repräsentative, Abfrage im Shop des Medienverlages O'Reilly[4] ergibt für Anfragen zu "mongodb" 64 Treffer, zu "couchdb" 20 Treffer, zu "couchbase" 15 Treffer und zu "dynamodb" lediglich 5 Treffer.

MongoDB

MongoDB wurde bis 2016 nach eigenen Angaben ca. 20 Millionen mal heruntergeladen. Dies umfasst allein den Teil der Benutzer, die MongoDB im Rahmen der kostenlosen Lizenz verwenden. Daneben betreut die MongoDB Inc. über 1.000 gewerbliche Kunden, u.a. Expedia, Forbes, Otto und Bosch.[5]

[2] [16b]
[3] [Yuh16]
[4] www.oreilly.com (Stand 26.12.2016)
[5] [17f]

Abseits der umfangreichen Dokumentation und Hilfestellung durch MongoDB Inc. finden sich eine Vielzahl von Beispielen und Lösungsvorschlägen für beinahe jeden Anwendungsfall im Internet. Außerdem existiert für die gängigsten Programmiersprachen eine entsprechende API zur Einbindung des Datenbanksystems, so dass bei der Umsetzung der Anwendungssoftware keinerlei Einschränkungen vorliegen.[6]
Aufgrund der großen Verbreitung von MongoDB ist auch eine stetige Weiterentwicklung und Fehlerbereinigung der vorhandenen Assets gegeben, sei es durch den Hersteller selber oder die weltweite Community.

Amazon DynamoDB

Amazon DynamoDB definiert sich selbst als 'NoSQL-Cloud-Datenbank'. Dabei werden die Ressourcen für das Datenbanksystem auf den Servern des Dienstleisters Amazon, auf Basis eines Preissystems, zur Verfügung gestellt. Daneben wird eine lokale Version für Entwicklungs- und Testzwecke kostenlos angeboten.
Die Dokumentation ist recht umfangreich, es werden neben den Handbüchern auch diverse Tutorien zur Einarbeitung angeboten. Es existieren Schnittstellen zu diversen Programmiersprachen.[7]
All dies wird vornehmlich durch Amazon abgedeckt. Lösungsanfragen und -beispiele durch andere Personenkreise auf anderen Webseiten (z.B. Stackoverflow) finden sich nicht ganz so häufig, gerade im Vergleich mit den anderen betrachteten Datenbanksystemen in diesem Kapitel.

Couchbase

Couchbase ähnelt vom Aufbau sehr stark MongoDB, seine Abfragesprache ist den SQL-Sprachen relationaler Systeme nicht unähnlich.
Die Verbreitung ist, gerade im Vergleich mit MongoDB, allerdings nicht so stark. Es existieren APIs für die gängigsten Programmiersprachen.[8]
Die Hilfestellung und Unterstützung durch Dritte im Internet fällt ebenfalls geringer aus.

[6] [17h]
[7] [17b]
[8] [17d]

CouchDB

CouchDB definiert sich selbst als ein Datenbanksystem, welches vornehmlich auf separaten Servern laufen soll. Um eine breite Basis an Sprachen und Frameworks zu unterstützen, stellt CouchDB lediglich das HTTP-Protokoll für die Kommunikation und das JSON-Datenformat für die Speicherstruktur der Dokumente zur Verfügung.[9] Es existieren allerdings APIs für verschiedene Sprachen (z.B. JAVA), die jedoch ausschließlich von der Web-Community entwickelt und gepflegt werden.
Im Vergleich mit den drei zuvor genannten Umsetzungen, orientiert sich CouchDB in einigen Funktionen stark an relationalen Systemen. Neben der Erstellung von Views kann für Dokumente datenbankseitig ein Grundschema festgelegt werden, welche bei der Übermittlung von Daten auf Einhaltung geprüft wird.[10]

4.1.2 Der Vergleich der API-Unterstützungen

Die Verfügbarkeit einer geeigneten API für die Kommunikation zwischen Anwendung und Datenbanksystem kann durchaus Einfluss auf die Entscheidung von Entwicklern haben.
Die Abbildung 4.1 gibt die Auswahlmöglichkeiten aufgrund der Nennungen auf den Herstellerseiten wieder, dort nicht genannte APIs wurden nicht berücksichtigt.
CouchDB fällt in der Darstellung aus dem Rahmen, da die Entwickler lediglich das HTTP-Protokoll und das JSON-Datenformat als Kommunikationsschnittstelle vorgeben. Obwohl es durchaus Community-Projekte, z.B. für eine JAVA-API, gibt, werden diese nicht auf den Seiten von CouchDB gesammelt aufgeführt.

[9] [17c]
[10] [ALS10, S. 53-73]

API-Unterstützung der dokumentenbasierten Datenbanksysteme

	MongoDB	Amazon DynamoDB	Couchbase	CouchDB
Android	-	Amazon	Couchbase	
C	MongoDB	-	Couchbase	
C++	MongoDB	Amazon	Couchbase	
Erlang	Community	-	-	
Go	Community	Amazon	Couchbase	
Hadoop Connector	MongoDB	-	-	
Haskell	MongoDB	-	-	
iOS	-	Amazon	-	
Java	MongoDB	Amazon	Couchbase	
JavaScript	-	Amazon	-	
Lua	Experimental	-	-	
.NET	-	Amazon	Couchbase	
Node.JS	MongoDB	Amazon	Couchbase	
PHP	MongoDB	Amazon	Couchbase	
Perl	MongoDB	-	-	
Python	MongoDB	Amazon	Couchbase	
Ruby	MongoDB	Amazon	Couchbase	
Scala	MongoDB	-	-	

Tabelle 4.1: Die vier betrachteten Umsetzungen eines dokumentenbasierten Datenbanksystem unterstützen eine Vielzahl von verschiedenen Sprachen und Frameworks. CouchDB setzt einfach nur das HTTP-Protokoll und das JSON-Format voraus. Angezeigt wird, durch wen die Unterstützung erfolgt.

4.1.3 Fazit

Als Datenbanksystem für die Umsetzung wird das Datenbanksystem MongoDB ausgewählt. Gründe hierfür sind

- die große Reichweite, insbesondere die Möglichkeiten zur Lösungssuche und Erklärung der Dokumentation anhand diverser Beispiele im Internet.
- die Verwendung einer SQL-ähnlichen Sprachkonstruktion.
- die freie Verfügbarkeit aller Komponenten.
- die direkte Unterstützung von PHP durch eine entsprechende Schnittstelle, in Verbindung mit einer modifizierten PHP-Schnittstelle für das Laravel-Framework, zur Einbindung in die Anwendungsoberfläche.

4.2 Die nichtrelationale Umsetzung

Die Umsetzung der leichtathletischen Mehrkampfdatenbank als nichtrelationaler Ansatz in MongoDB erfolgt auf Grundlage des relationalen Datenmodells, dargestellt in Abbildung 3.2, und der

bereits existierenden Anwendungsoberfläche aus dem zuvor umgesetzten Praxisprojekt.
In den folgenden Abschnitten wird kurz auf das Modell für die Umsetzung in MongoDB, sowie auf ein grundsätzliches Problem dokumentenbasierten Datenbanksysteme in Bezug auf redundant auftretende Dateninformationen eingegangen.

4.2.1 Die Modellierung in der Umsetzung

In MongoDB können ähnlich strukturierte Dokumente in sogenannten Kollektionen (Collections) abgelegt werden. Diese Kollektionen werden von MongoDB als Container betrachtet, es gibt keinerlei Vorgaben welche Dokumente in welchem Container abgelegt werden. Diese Entscheidung wird auf Anwendungsebene durch ein implizites Modell getroffen.

Für die Umsetzung der Mehrkampfdatenbank werden die zu speichernden Informationen in vier Kollektionen aufgeteilt. In Abbildung 4.2 sind drei dieser Kollektionen dargestellt.

Benutzer (Users)

Diese Kollektion enthält die Benutzerdaten für die Weboberfläche und wird ausschließlich vom PHP-Framework Laravel genutzt. Die Daten stehen in keinem direkten Zusammenhang mit den Daten in den anderen Kollektionen.

Wettkampftypen (CompetitionTypes)

Alle Wettkämpfe gehören einem bestimmten Wettkampftyp (competitiontype) an, welcher die Disziplinen und die anzuwendenden Formeln und Faktoren bestimmt. Die Kollektion beinhaltet Dokumente, die als Schablone genutzt werden, um bei gleichartigen Wettkämpfen dieselben Grundlagen zu verwenden.

Wettkämpfe (Competitions)

Diese Kollektion verwaltet den größten Teil an Dateninformationen. Jeder Wettkampf wird als eigenes Dokument abgelegt und enthält, neben seinen Grundinformationen und dem Verweis auf einen Wettkampftyp, alle Athleten, ihre erbrachten Leistungen und die daraus resultierenden Punkte.
Die Athleten sind dabei jeweils durch eine Referenz repräsentiert. Der Grund für diese Entscheidung wird in Abschnitt 4.2.2 erläutert.

Athleten (Athletes)

Alle notwendigen, personenbezogenen Daten der Athleten werden in dieser Kollektion gespeichert. Sie dienen als Grundlage für die Referenzhinweise in der Wettkampf-Kollektion.

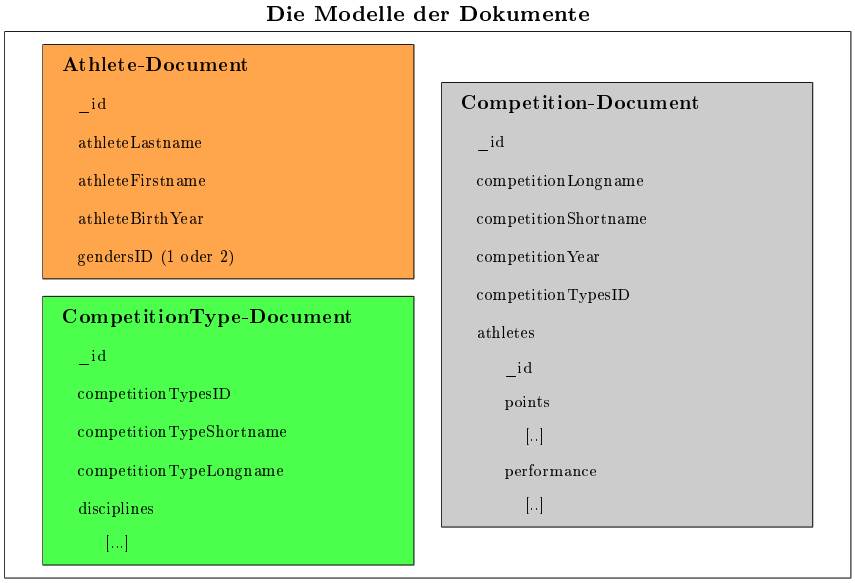

Abbildung 4.2: Für jedes Dokument existiert ein implizites Modell, welches die Anwendungsseite definiert. Unter den Punkten disciplines, points und performances bestehen weitere Schachtelungen.

4.2.2 Die Referenzierung in Dokumenten

Ein großes Problem bei der Bündelung von Informationen in Dokumenten ist das Vorhandensein redundanten Daten. Die Dokumentation von MongoDB verweist selbst auf die Kontroverse zwischen der Aufteilung von Informationen und der Bündelung.[11]

Die Entscheidung, Athleteninformationen nicht im Wettkampfdokument einzubetten, hat zwei hauptsächliche Gründe. Zum einen kann über die Athleten-Kollektion sehr schnell nach einem bestimmten Athleten gesucht werden, um bei Neuanlagen eine Athletenauswahl zu bieten.

Zum anderen müssen Änderungen und Korrekturen nicht umständlich auf mehreren redundanten Kopien in den Wettkämpfen ausgeführt werden. Erschwerend hierbei wäre zusätzlich, dass erst alle Wettkämpfe ermittelt werden müssten, an denen der Athlet teilgenommen hat. Außerdem werden nur die einzelnen Aktualisierungen in jedem Dokument atomar ausgeführt. Im Falle eines Abbruches nach der Hälfte aller Aktualisierungen wäre somit kein "Rollback" auf den ursprünglichen Zustand mehr möglich.

[11] [17h]

Ein Nachteil der Referenzierung ist die Notwendigkeit, die Daten durch mehrere Anfragen zu sammeln.

5 Der Vergleich mit der relationalen Umsetzung

> "If you're comparing an RDBMS to a NoSQL database, then you're comparing apples to motorbikes!"
>
> — Fowler[1]

In diesem Kapitel wird die Umsetzung der Mehrkampfdatenbank in einem nichtrelationalen, dokumentenorientierten Datenbanksystem (MongoDB[2]) mit der Umsetzung in einem relationalen Datenbanksystem (MariaDB[3]) in Bezug auf Laufzeitperformanz bei Datenabrufen verglichen. Das Hauptaugenmerk liegt dabei auf den Antwortzeiten der Systeme bei immer größer werdenden Datenmengen. Dabei soll auch kurz auf Unterschiede bei der Ausführung der Anfragen, sofern solche erkennbar sind, eingegangen werden.

Den Autoren ist bewusst, dass durch Optimierung der entsprechenden Anfragen weitere Leistungsgewinne durchaus möglich sind. Dieser Vergleich soll daher keine generelle Leistungsaussage über die jeweiligen Systeme liefern, sondern vorrangig die Eigenheiten des jeweiligen Lösungsansatzes bewerten.

Eine vollständige Übersicht aller Messergebnisse ist im Anhang unter dem Punkt *Messreihen*, ab Seite 71, zu finden.

[1] [Fow15, S. 411]
[2] [16c]
[3] [16d]

5.1 Die Sortierung einer Kollektion / Tabelle

In der ersten Testreihe wird eine Kollektion, beziehungsweise eine Tabelle, mit Informationen zu Athleten anhand ihrer Nach- und Vornamen sortiert. Dabei wird die vergangene Zeit von der Übergabe der Suchanfrage bis zu Rückmeldung gemessen und, um etwaige Leistungseinbrüche auszuschließen, der Median aus mehreren Durchläufen verwendet.
Die Testreihe wird anschließend mit einer immer größeren Anzahl von Datensätzen wiederholt. Parallel wird geprüft, ob eine Indexierung der Nach- und Vornamen zu einer Leistungssteigerung im Suchverlauf führt. Eine kurze Übersicht der Testparameter ist in Tabelle 5.1 zu sehen.

Während der Offset-Parameter einen klar erkennbaren Ausschlag für die Dauer einer Anfrage verursacht, hat die Limitierung der auszugebenden Datensätze keinen signifikanten Einfluss. Alle vier Teildiagramme zeigen, dass die Gesamtmenge der Datensätze ebenfalls die Dauer der Anfrage beeinflusst.

Zusammenfassung der Testparameter für die Athletensortierung

	Parameter	Werte
1.	Menge der zu sortierenden Datensätze	10.000 / 100.000 / 1.000.000 Athleten
2.	Offset (Anzahl der übersprungenen Daten im sortierten Ergebnis vor der Ausgabe)	kein Offset / 10% / 50% / 90%
3.	Limitierung der Ausgabe	keine Limitierung / 1.000 / 100 / 10 Datensätze
4.	Indexierung der Athleten	ja / nein

Tabelle 5.1: Parameter für die Erstellung der Sortierungsanfrage auf die Athletendatensätze. Die aufgeführten Parameter werden so miteinander kombiniert, dass jede Wertekombination aller vier Parameter durchgeführt wird.

5.1.1 Die Sortierung einer Kollektion in MongoDB

In Abbildung 5.1 sind die Durchschnittswerte der einzelnen Durchgänge in Diagrammform dargestellt. Die Ergebnisse werden dabei nach dem Offset-Parameter gruppiert dargestellt.

Auffällig an den Ergebnissen ist, dass die Abfragen ohne Indexierung schneller ablaufen als ihre Vergleichsabfragen mit Index, auch wenn die Abstände immer recht klein ausfallen (maximal 0,3 Sekunden). Dies steht eigentlich im Gegensatz zur Dokumentation von MongoDB die für eine schnellere Sortierung die Verwendung von Indexen empfehlen:

"In MongoDB, sort operations can obtain the sort order by retrieving documents based on the ordering in an index. If the query planner cannot obtain the sort order from an index, it will sort the results in memory. Sort operations that use an index often have better performance than those that do not use an index. In addition, sort operations that do not use an index will abort when they use 32 megabytes of memory." [4]

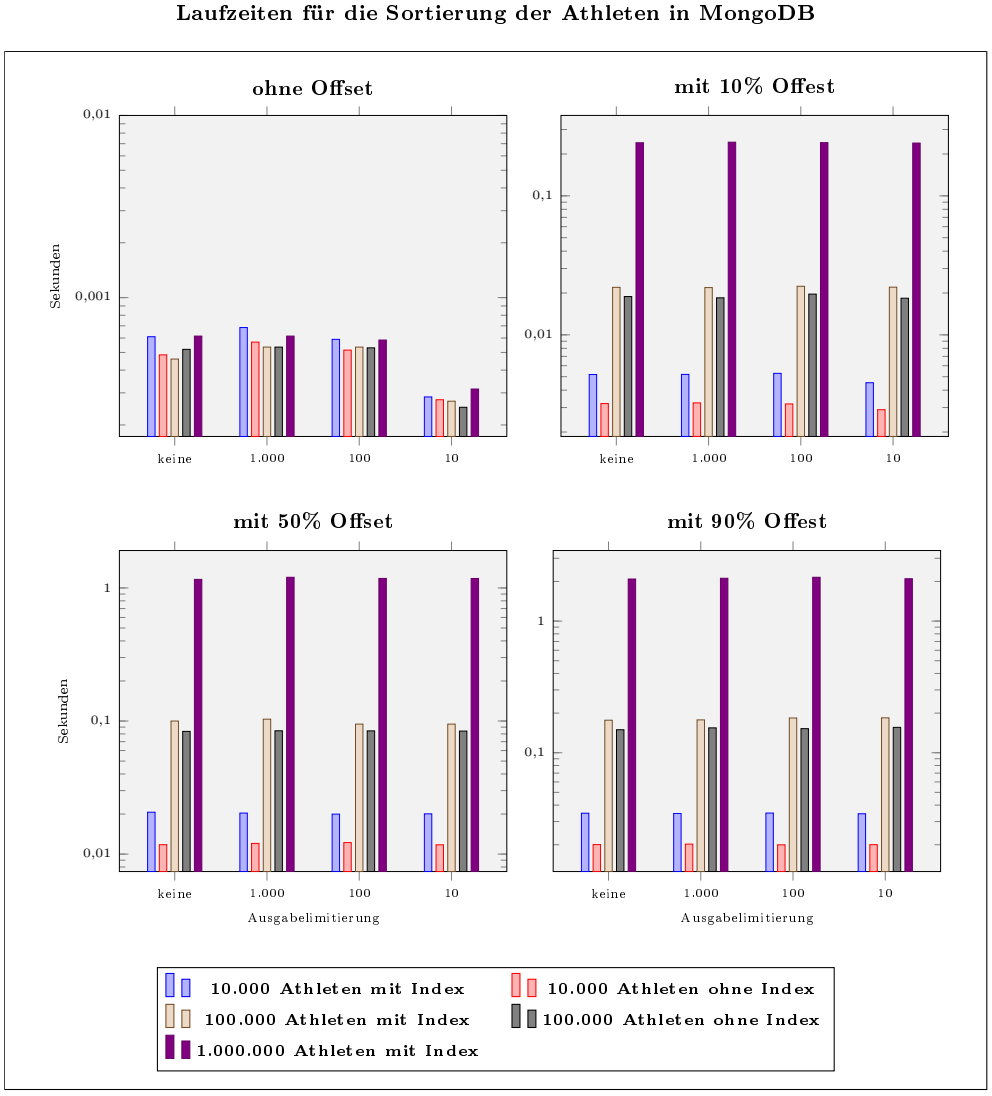

Abbildung 5.1: Die vier Diagramme zeigen die Dauer der Sortierung von unterschiedlich großen Datenmengen an Athleten. Auf der X-Achse ist die Mengenbegrenzung der rückgemeldeten Datensätze, auf der Y-Achse (logarithmisch) die Dauer der Anfrage dargestellt. Der Offset beschreibt, wieviel der sortierten Datensätze vor der Ausgabe übersprungen werden.

Indexe werden aber dann notwendig, wenn die Datenmenge beim Sortieren die Speicherlimitierung überschreitet. Dies tritt in den durchgeführten Testläufen bei der Sortierung von 1.000.000 Athleten auf, daher kann für diesen Testfall keine Zeitmessung durchgeführt werden.

[4] [16e]

Insgesamt ist aber zu erkennen, dass MongoDB die Sortierungen recht schnell ausführt, die längste Anfrage braucht knapp über zwei Sekunden.

5.1.2 Die Sortierung einer Tabelle in MariaDB

Anders als bei der Testreihe in MongoDB fallen die Messergebnisse in MariaDB viel differenzierter aus. Insbesondere der Einsatz beziehungsweise das Weglassen eines Indexes weist einen erheblichen Leistungsunterschied auf. Die Messergebnisse sind in Abbildung 5.2 dargestellt. Auffallend ist, dass bei kleinen Datenmengen der Zeitunterschied zwischen Abfragen mit und ohne Index recht gering ausfällt, bei einer Millionen Datensätzen aber von 2 Sekunden für die indexierte Abfrage auf etwa 7 Stunden für die nicht indexierte Abfrage ansteigt.

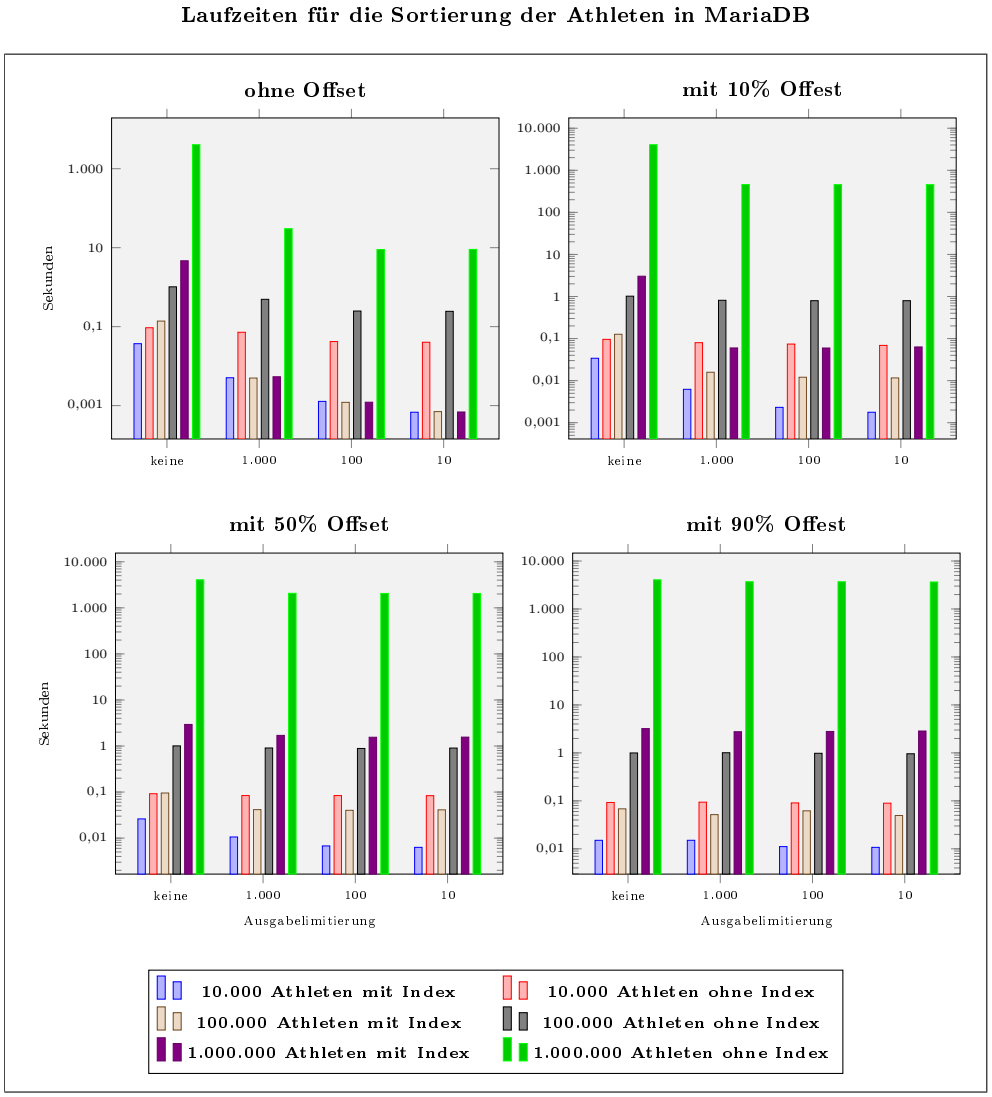

Abbildung 5.2: Die vier Diagramme zeigen die Dauer der Sortierung von unterschiedlich großen Datenmengen an Athleten. Auf der X-Achse ist die Mengenbegrenzung der rückgemeldeten Datensätze, auf der Y-Achse (logarithmisch) die Dauer der Anfrage dargestellt. Der Offset beschreibt, wieviel der sortierten Datensätze vor der Ausgabe übersprungen werden.

Die Verwendung der Profiling-Funktionen von MariaDB liefert hierzu ein interessantes Ergebnis. Die beiden zeitrelevanten Aktivitäten, in Tabelle 5.2 dargestellt, sind das Sortieren der Daten (Sorting result) und das Ausliefern des Ergebnisses (Sending data).
Während bei Datenmengen von 10.000 und 100.000 Datensätzen die Verwendung eines Indexes keinen Zeitvorteil bringt, steigt bei einer Millionen Datensätze nicht nur der Zeitaufwand für die Sortierung, sondern auch für die Übertragung der Daten.

Auszug zum Profiling in MariaDB

	10.000 Athleten		100.000 Athleten		1.000.000 Athleten	
	mit Index	ohne Index	mit Index	ohne Index	mit Index	ohne Index
Sorting result	0,00002	0,08359	0,00002	0,84836	0,00002	61,88438
Sending data	0,01971	0,00880	0,09714	0,18662	4,11891	1.000,00000

Tabelle 5.2: Dargestellt ist ein Auszug des Profiling-Ergebnisses. Alle Werte sind in Sekundenangaben dargestellt. Grundlage ist die Sortierungsanfrage ohne Offset und ohne Ausgabelimitierung.

Es ist zu vermuten, dass für das Sortieren nicht mehr genug Arbeitsspeicher zur Verfügung gestellt werden kann und eine Zwischenlagerung auf Festplattenspeicher vorgenommen wird. Eine ausreichend gute Erklärung für das Ansteigen der Auslieferungszeit liegt bisher nicht vor, hierzu wäre vermutlich eine genauere Betrachtung des Datenbankmanagementsystems und seiner Programmierung notwendig.

5.1.3 Der Vergleich beider Umsetzungen

In allen Durchgängen ist die Abfrage in MongoDB schneller abgewickelt. Der Abstand bei "kleinen" Datenbeständen ist jedoch so gering, dass andere Einflüsse im Produktivbetrieb dies überdecken würden.

Beide Systeme sollten bei großen Datenmenge für die Sortierung auf Indexe zurückgreifen können. Während MariaDB dadurch erheblich an Geschwindigkeit zulegt, ist ein Index für MongoDB unerlässlich.

5.2 Die Suche in einer Kollektion / Tabelle

Die zweite Testreihe liefert eine sortierte Tabelle bzw. Kollektion der Athleten, deren Inhalte einem bestimmten Suchmuster auf Nach- und/oder Vorname entspricht.

Zusammenfassung der Testparameter für die Athletensuche

	Parameter	Werte
1.	Menge der zu durchsuchenden Datensätze	10.000 / 100.000 / 1.000.000 Athleten
2.	Suchbereich	Nachname / Vorname / Nach- und Vorname
3.	Suchkriterium	vollständiger Name / Namensteil / Namensanfang

Tabelle 5.3: Parameter für die Erstellung der Suchanfragen nach Athleten. Die aufgeführten Parameter werden so miteinander kombiniert, dass jede Kombination aller drei Parameter durchgeführt wird.

Eine Zusammenstellung der verschiedenen Parameter ist in Tabelle 5.3 zu sehen.
Die Suchparameter für die Anfrage werden im Vorfeld aus einem vorhandenen Datensatz erstellt und bestehen bei den Anfragen auf Namensanfänge und Namensteile aus zwei aufeianderfolgenden Buchstaben.

5.2.1 Die Suche in einer Kollektion in MongoDB

Ein Vergleich für die Laufzeiten der Suchanfragen ist in Abbildung 5.3 dargestellt.
Allgemein lässt sich feststellen, dass mit einem Ansteigen der zu durchsuchenden Datenmenge die Laufzeit der Suche ebenfalls steigt. Eine Ausnahme bildet das Suchergebnis auf einen Vornamensteil, bei dem die Anfrage auf eine Million Athleten schneller durchgeführt wird als bei kleineren Datenmengen.
Eine ausreichend gut Erklärung kann hierfür nicht gegeben werden, auch mehrfache Wiederholungen liefern dasselbe Ergebnis. Auch die Anzahl der Treffer in den Suchen liefert keinen erklärenden Ansatz, fällt sie doch bei den größeren Datenmengen ebenfalls größer aus.

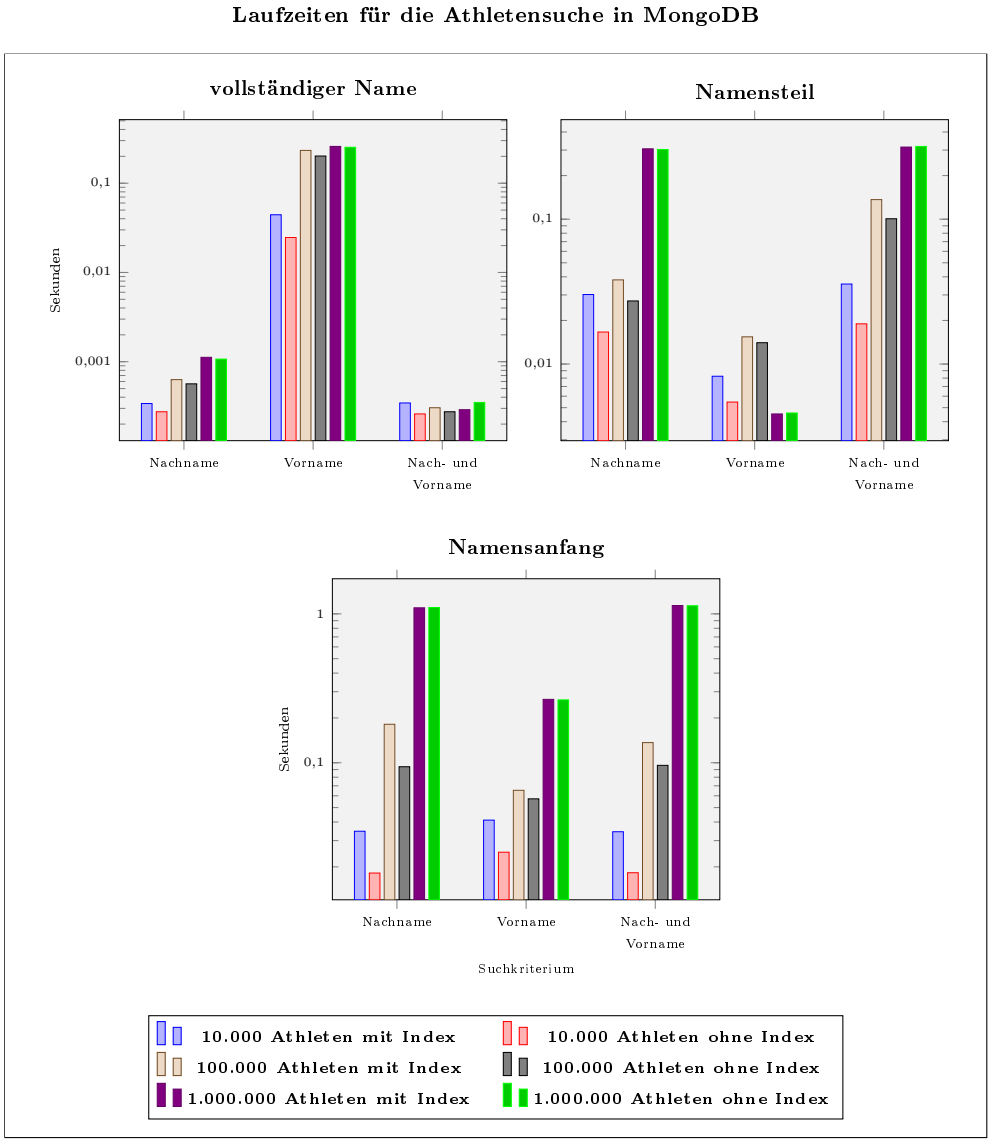

Abbildung 5.3: Die vier Diagramme zeigen die Dauer der Athletensuche in unterschiedlich großen Datenmengen. Auf der X-Achse ist das Suchkriterium, auf der Y-Achse (logarithmisch) die Dauer der Anfrage dargestellt.

Eine genauere Betrachtung der Suchalgorithmen und Heuristiken von MongoDB könnte vielleicht eine Erklärung liefern.

5.2.2 Die Suche in einer Tabelle in MariaDB

In Abbildung 5.4 sind die Laufzeiten der Suchanfragen zusammengefasst.
Wie schon bei MongoDB steigt auch hier die Dauer einer Anfrage mit der Anzahl der zu durchsuchenden Datensätze. Während bei kleineren Datenmengen der Index keinen großen Unterschied ausmacht, gehen die Ergebnisse bei einer Million Datensätze teils weit auseinander.

Bei der Suche nach einem Namensteil liegen sie aber wieder gleichauf. Dies lässt vermuten, das der Suchalgorithmus in MariaDB keine weitergehenden Heuristiken hierfür verwendet.

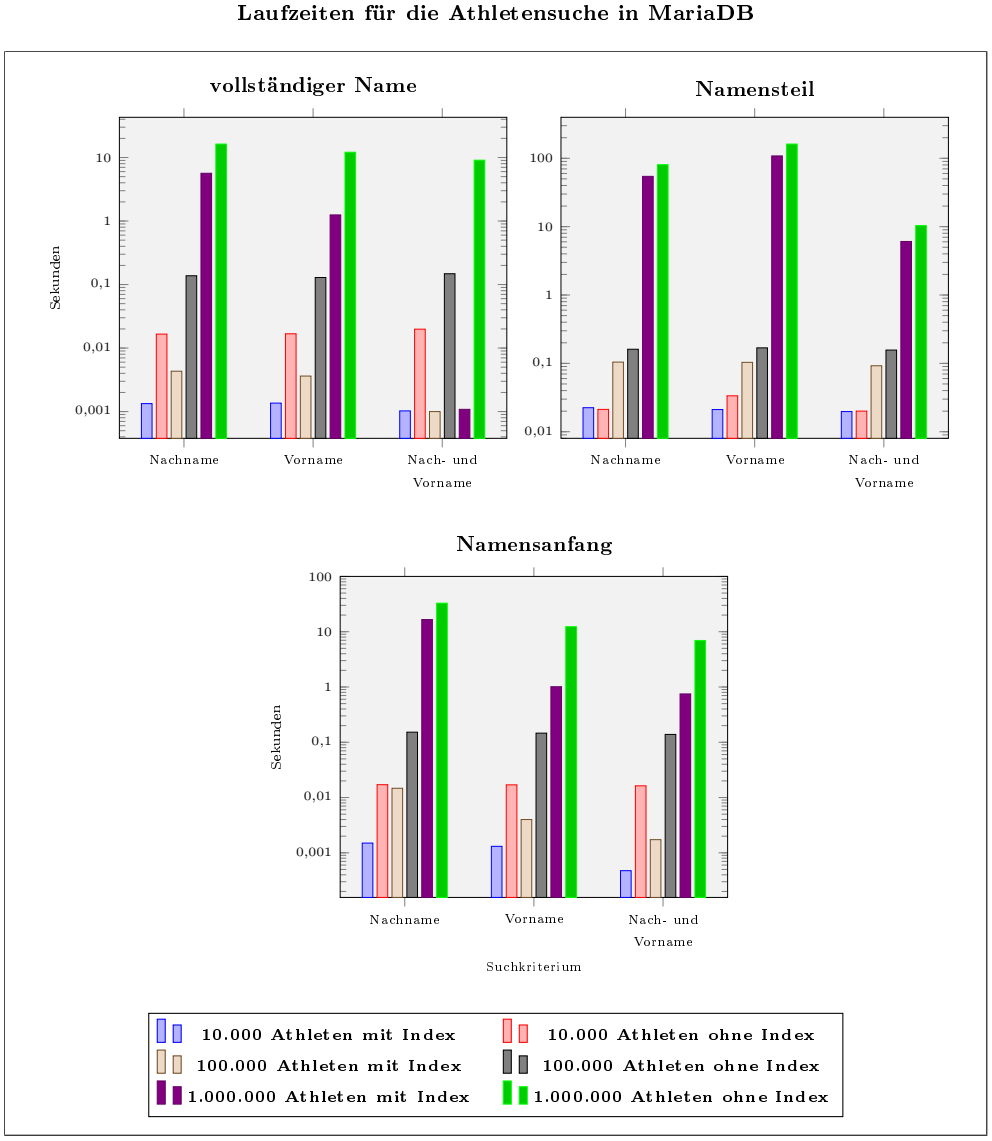

Abbildung 5.4: Die vier Diagramme zeigen die Dauer der Athletensuche in unterschiedlich großen Datenmengen. Auf der X-Achse ist das Suchkriterium, auf der Y-Achse (logarithmisch) die Dauer der Anfrage dargestellt.

5.2.3 Der Vergleich beider Umsetzungen

MongoDB ist auch in der zweiten Testreihe bei großen Datenmengen klar schneller. Insbesondere bei der Suche nach Namensteilen übertrifft er MariaDB klar. Wie schon in der ersten Testreihe ist der Abstand bei "kleinen" Datenmenge aber vernachlässigbar.

Während für MariaDB die Verwendung von Indexen einen Geschwindigkeitsvorteil bringt, scheint MongoDB dadurch ausgebremst zu werden.

5.3 Die Suche in verschachtelten Dokumenten / Zuordnungstabellen

Die dritte Testreihe führt Anfragen aus, die nach Informationen in geschachtelten Dokumenten, insbesondere in den sogenannten "nested Documents", sucht.
In den relationalen Systemen finden sich diese Informationen in entsprechenden Zuordnungstabellen, deren Durchsuchen mittels entsprechender Indexe recht einfach durchführbar ist. Demgegenüber steht die Suche in "nested Documents" und die Frage, wie diese durchzuführen ist und welche Hilfsmittel zur Verfügung gestellt werden.
Im konkreten Fall wird nach einem bestimmten Athleten gesucht, welcher anhand einer eindeutigen Identifikationsnummer selektiert werden kann.

5.3.1 Die Suche in verschachtelten Dokumenten in MongoDB

Wie schon in den vorhergehenden Testreihen sind die Suchanfragen in MongoDB ohne Verwendung eines Indexes schneller durchgeführt. Die Abbildung 5.5 zeigt außerdem, dass alle Anfragen in weniger als einer Sekunde das gewünschte Ergebnis liefern. Auch die Tatsache, dass die gesuchte Information innerhalb eines Dokumentes erst aufgefunden werden muss, hat in den durchgeführten Abfragen keine besondere Auswirkung.

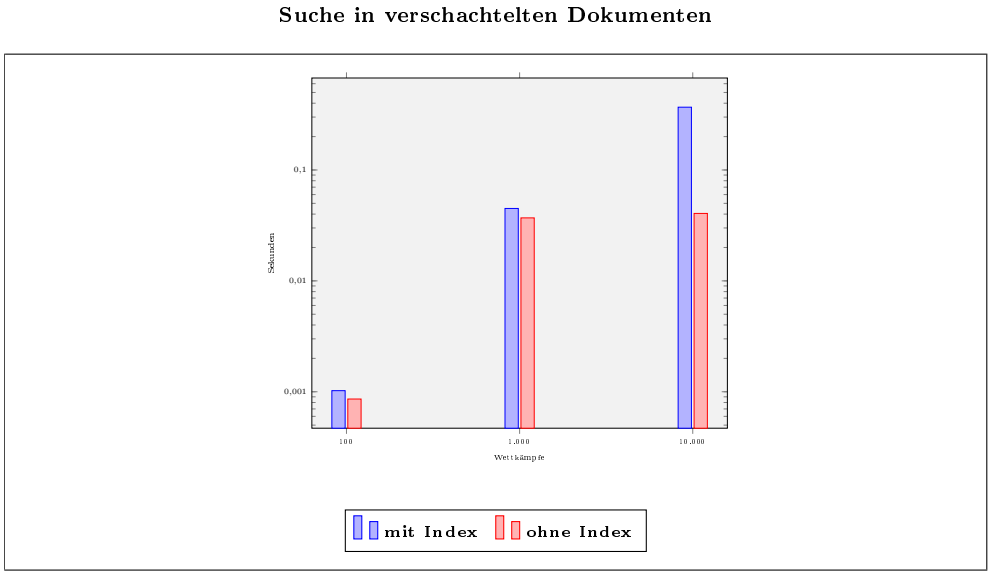

Abbildung 5.5: Die Grafik zeigt die Laufzeiten für die Suchanfragen nach einem bestimmten Athleten innerhalb der Wettkampf-Dokumente. Dabei werden die Athleten in einem Fall mit einem Index versehen.

5.3.2 Die Suche in Zuordnungstabellen in MariaDB

Das von MariaDB standardmäßig verwendete Speichersubsystem legt automatisch einen Index auf Fremdschlüssel an. Da die zu suchende Information in dieser Testreihe als Fremdschlüssel in einer Zuordnungstabelle abgelegt ist, ist eine Suche ohne Index nicht möglich.
Aus diesem Grund beinhaltet die Ergebnisgrafik in Abbildung 5.6 keine Informationen zu nicht indexierten Suchen. Die verbliebenen Anfragen sind aufgrund des vorhandenen Indexes, wie nicht anders zu erwarten, sehr schnell ausgeführt.

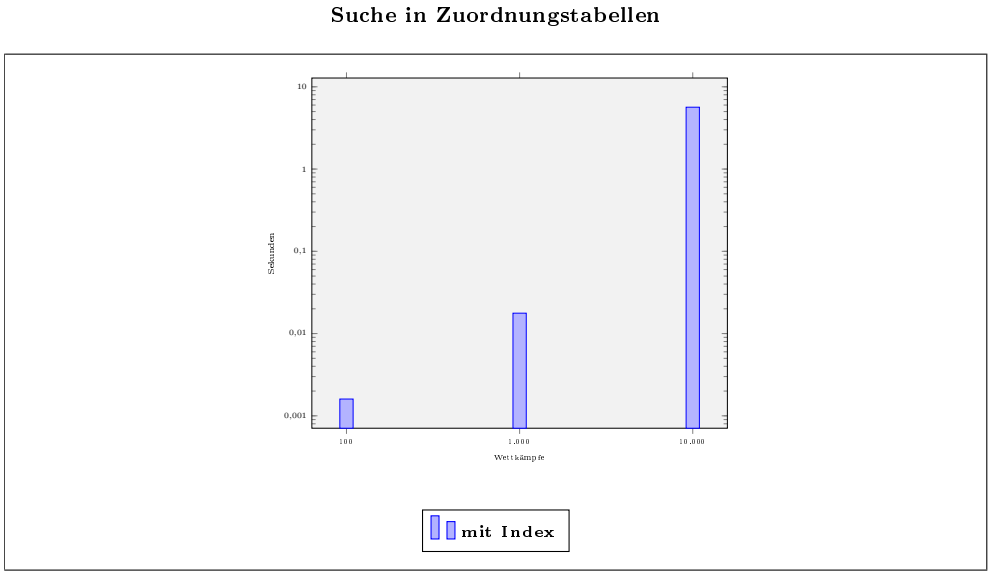

Abbildung 5.6: Die Grafik zeigt die Laufzeiten für die Suchanfragen nach einem bestimmten Athleten innerhalb einer Zuordnungstabelle über einen Fremdschlüsselverweis. Dieser ist standardmäßig immer indexiert.

5.3.3 Der Vergleich beider Umsetzungen

Da in MariaDB, bei Verwendung von XtraDB[5] als Speichersubsystem, Fremdschlüssel automatisch indexiert werden, können keine Anfragen ohne Index durchgeführt werden. Daher können nur die Anfragen unter Nutzung eines Indexes verglichen werden.
Beide Systeme sind bei kleinen Datenmengen sehr schnell und liefern in weniger als einer Zehntelsekunde ihre Ergebnisse aus. Bei großen Datenmengen wächst der Abstand, so dass MongoDB einen klaren Vorteil vorweisen kann.
Die Indexierung innerhalb von Dokumenten bringt in MongoDB keinen messbaren Vorteil, ist aber aufgrund der bereits genannten Speicherproblematik empfehlenswert.

5.4 Die Umstrukturierende Abfrage in Dokumenten / Tabellen mit JOINs

Häufig kommt es vor, dass das Ergebnis einer Anfrage eine ganz bestimmte Ausgabestruktur verlangt.
In diesem Abschnitt werden daher die JOIN-Operationen in MariaDB mit vergleichbaren Funk-

[5] [17i]

tionen von MongoDB anhand unterschiedlich großer Datenmengen getestet.

In relationalen Systemen können durch das Verbinden von Relationen über JOIN-Operationen, die Reihenfolge der JOINs und die Angabe der auszugebenden Spalten unterschiedliche Ergebnisse erreicht werden.

Dokumentenbasierte Datenbanksysteme wie MongoDB kennen in der Regel das Konzept von JOINs nicht, da alle zusammengehörigen Informationen in einem Dokument zusammengefasst werden. Die Hierarchie der Daten im Dokument geben die Struktur der Ausgabe vor.
Um eine andere Struktur zu erhalten, müssten die abgerufenen Informationen nachträglich umgestellt werden. Viele dokumentenbasierte Systeme bieten aber auch Funktionalitäten, um diese Umstrukturierung innerhalb des Datenbanksystems durchzuführen. Im Fall von MongoDB werden zwei verschiedene Möglichkeiten, die MAP-REDUCE-Funktion und die Aggregation-Pipeline,[6] genauer untersucht.

5.4.1 Die Beschreibung des Testaufbaus

Das Ziel der Anfrage ist es, zu einem einzelnen Athleten alle Wettkämpfe, an denen er teilgenommen hat, und die dort erzielten Leistungen anzuzeigen.
In Kapitel 4.2.1 ist in Abbildung 4.2 der Dokumentenaufbau für einen Wettkampf (Competition-Document) dargestellt. Um nun ein Dokument von MongoDB zu erhalten, welches die zuvor genannten Anforderungen aufweist, ist eine Umstrukturierung notwendig.
Ein vergleichbares Ergebnis, in der relationalen Umsetzung von MariaDB, wäre ein JOIN der Wettkampf- (competitions) und Leistungs-Relationen (performances), wie sie in Kapitel 3.1.1 in Abbildung 3.2 dargestellt sind, zusammenführt.
Die Menge der zu bearbeitenden Daten wird sukzessive gesteigert (siehe Tabelle 5.4), um die Bearbeitungszeiten bei unterschiedlichen Datenmengen zu betrachten.

[6] [17a]

Datenumfang für JOIN- und Aggregationstests

	Athleten gesamt	Anzahl Wettkämpfe	Athleten je Wettkampf
1. Fall	100	100	10
2. Fall	1.000	1.000	100
3. Fall	10.000	10.000	1.000

Tabelle 5.4: Darstellung des Datenumfangs für die Durchführung der umstrukturierenden Abfragen.

5.4.2 Die Umstrukturierende Abfrage in Dokumenten in MongoDB

Entspricht das gewünschte Ergebnis einer Abfrage nicht der hinterlegten Dokumentenstruktur, gibt es zwei Ansätze die Umstrukturierung vorzunehmen.
Der sehr einfache Ansatz besteht darin, die Informationen an die Anwendungsseite zu übertragen und dort die Daten neu zu strukturieren. Der Nachteil besteht darin, dass vielleicht unnötige Daten mit übertragen werden, wodurch der Kommunikationsaufwand zwischen Datenbanksystem und Anwendung erheblich steigen kann.
Die Alternative ist die Verwendung "bordeigener" Mittel im Datenbanksystem, um die notwendige Umstrukturierung durchzuführen. MongoDB stellt hierfür die MAP-REDUCE-Funktion und die Aggregation-Pipeline zur Verfügung. Beide nachfolgend kurz beschrieben werden.

MAP-REDUCE-Funktion

Das Konzept der MAP-REDUCE-Funktion ist ein allgemeines Programmiermodell, welches vor allem für die parallele Verarbeitung von Daten auf mehreren Maschinen verwendet wird.[7]
Bei der Implementierung der MAP-REDUCE-Funktion kann es in verschiedenen Datenbanksystemen Unterschiede in der Ausführung geben. Daher wird an dieser Stelle ausschließlich der Funktionsablauf erläutert.

Die MAP-REDUCE-Funktion lässt sich grob in zwei Teilschritte, den MAP-Vorgang und den REDUCE-Vorgang unterteilen. Die Teilschritte müssen immer in dieser Reihenfolge durchgeführt werden. Allerdings können mehrere MAP-REDUCE-Funktionen aufeinander aufbauend ausge-

[7] [DG04]

führt werden.

Im MAP-Vorgang wird immer ein einzelnes Aggregat übergeben. Sofern mehrere Aggregate vorliegen, wird der MAP-Vorgang für jedes separat ausgeführt. Hier setzt auch die Möglichkeit der Parallelisierung an. Da jeder MAP-Vorgang unabhängig durchgeführt werden kann, ist eine Verteilung verschiedenen Rechnern möglich. Jeder MAP-Vorgang liefert eine Menge von KEY-VALUE-Paare aus. In Abbildung 5.7 wird dies beispielhaft dargestellt.

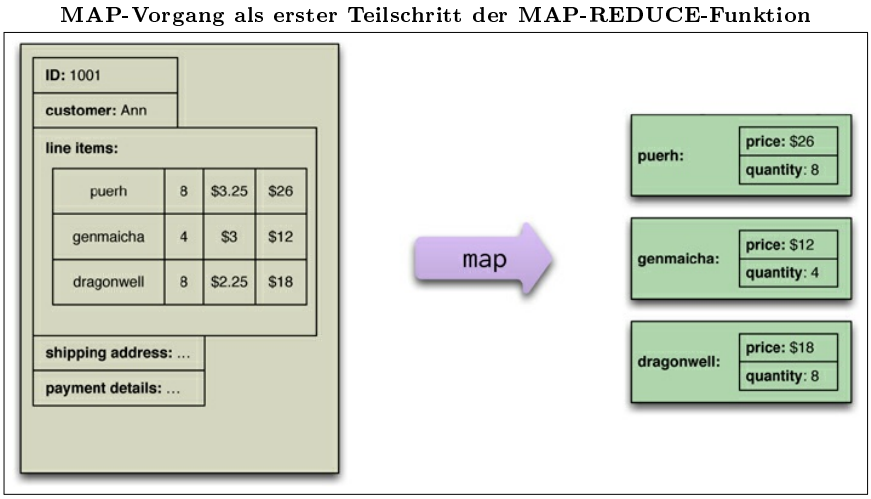

Abbildung 5.7: Der MAP-Vorgang liest ein Aggregat aus dem Datenbanksystem und liefert ein Menge an KEY-VALUE-Paaren zurück. In diesem Beispiel werden aus einer Rechnung die einzelnen Artikel mit ihrer Anzahl und dem Gesamtpreis ausgelesen. Die Artikelbezeichnung dient dabei als Schlüssel (KEY), die Stückzahl und der Preis werden als Wertpaar (VALUE) kombiniert.
Quellenvorlage: [SF13, Kapitel 7]

Bevor der REDUCE-Vorgang die Ergebnisse des MAP-Vorgangs aufnimmt, werden Key/Value-Paare mit identischen Schlüsseln gruppiert.
Der REDUCE-Vorgang führt auf jedem dieser gruppierten Key/Value-Paare eine vorher durch den Anwender festgelegte Operation aus, zum Beispiel die Summierung von bestimmten Werten, und gibt das Ergebnis zurück. Damit ist die MAP-REDUCE-Funktion am Ende angelangt.

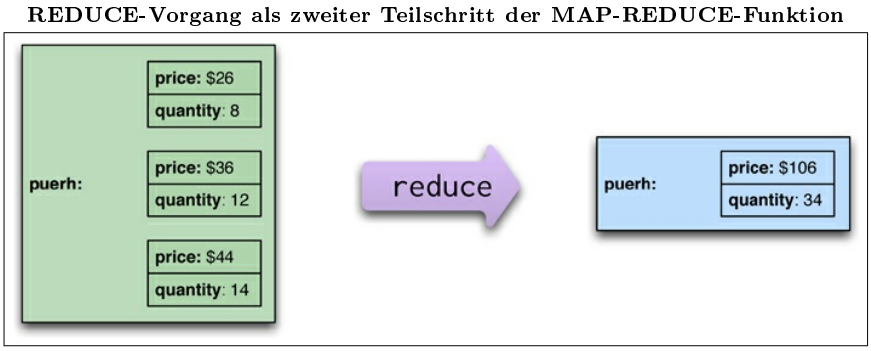

Abbildung 5.8: Der REDUCE-Vorgang erhält die nach Schlüsseln gruppierten Ergebnisse der MAP-Vorgänge und führt auf diesen eine vorher definierte Operation aus. In diesem Fall werden die im MAP-Vorgang ausgelesenen Artikelinformationen gruppiert und für jeden Artikel der Gesamtpreis und die Gesamtstückzahl zurückgegeben.
Quellenvorlage: [SF13, Kapitel 7]

In MongoDB kann die MAP-REDUCE-Funktion um zwei weitere Teilschritte erweitert werden. Dabei wird vor der eigentlichen MAP-REDUCE-Funktion eine Abfrage (QUERY) und ganz am Ende eine Finalisierung (FINALIZE) des Ergebnisses durchgeführt.
Der QUERY-Teilschritt soll nicht benötigte Dokumente im Vorfeld ausfiltern, der FINALIZE-Teilschritt führt Operationen durch, die erst nach dem MAP-REDUCE ausgeführt werden können.[8]

Die Aggregation-Pipeline von MongoDB

Die Aggregation-Pipeline ist ein hauseigener Aggregationsansatz von MongoDB. In ihrer Funktionsweise ist sie dem Modell von MAP-REDUCE nicht unähnlich, bietet aber eine eigene Semantik und ist dadurch stärker an die Datenstruktur von MongoDB angepasst.
Die Möglichkeiten sind sehr weitreichend und umfassend, daher wird an dieser Stelle auf die Dokumentation von MongoDB verwiesen.[9]
In der Aggregation-Pipeline werden, vereinfacht ausgedrückt, Anweisungen nacheinander ausgeführt, wobei das Ergebnis einer Anweisung den Dateninput für die nächste Anweisung darstellt.

Der Vergleich der Aggregatsfunktionen

Aus Abbildung 5.9 ist erkennbar, dass die Aggregation-Pipeline die geforderte Aufgabe etwas schneller erledigt. Aufgrund der logarithmischen Darstellung scheint der Abstand zum MAP-Reduce aber zu schrumpfen.

[8] [17g]
[9] [17a]

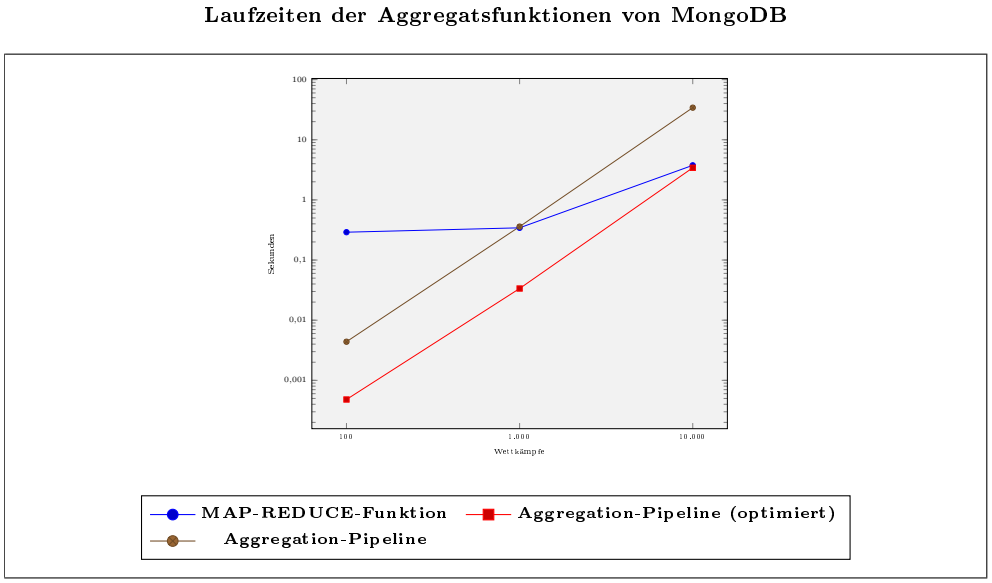

Abbildung 5.9: Die Grafik zeigt die Laufzeiten für die umstrukturierende Abfrage auf die Wettkampf-Kollektion in MongoDB. Dabei werden unterschiedliche Aggregatsfunktionen gegenübergestellt.

Tatsächlich ist der zeitliche Unterschied für die verschiedenen Datenmengen aber fast konstant. Bezogen auf die Gesamtlaufzeit der Anfragen wird der Abstand, im Verhältnis betrachtet, sogar immer geringer. In Tabelle 5.5 kann dies nachvollzogen werden.

Zeitvergleich der MongoDB Aggregatsfunktionen

	100 Wettkämpfe	1.000 Wettkämpfe	10.000 Wettkämpfe
Map-Reduce	0,2904	0,3433	3,7580
Aggregation-Pipeline (optimiert)	0,0005	0,0336	3,4221
Zeitabstand total	0,2899	0,3097	0,3359
Verhältnisfaktor	604,9792	10,2121	1,0982

Tabelle 5.5: Vergleich der Aggregatsfunktionen in MongoDB. Der Verhältnisfaktor zeigt an, um wieviel schneller die Aggregation-Pipeline ist. Bei einem Gleichstand hat der Verhältnisfaktor den Wert Eins.

Bei steigenden Datenmengen wird der zeitliche Abstand nebensächlicher, weitaus wichtiger ist die optimierte Umsetzung der Aggregatsfunktion. Tabelle 5.6 zeigt dies beispielhaft für die Aggregation-Pipeline, gleiches gilt aber auch für die MAP-REDUCE-Funktion.

Optimierung der Aggregation-Pipeline

	100 Wettkämpfe	1.000 Wettkämpfe	10.000 Wettkämpfe
Aggregation-Pipeline	0,0044	0,3596	34,1537
Aggregation-Pipeline (optimiert)	0,0005	0,0336	3,4221
Zeitabstand total	0,0039	0,3260	30,7316
Verhaältnisfaktor	9,1354	10,6971	9,9804

Tabelle 5.6: Wichtig für die Aggregation-Pipeline ist eine Optimierung der Anfrage. Bei der optimierten Anfrage werden im Vorfeld unnötige Dokumente aussortiert.Der Verhältnisfaktor zeigt an, um wieviel schneller die optimierte Aggregation-Pipeline ist. Bei einem Gleichstand hätte der Verhältnisfaktor den Wert Eins.

5.4.3 Die umstrukturierende Abfrage in Tabellen mit JOINs

JOIN-Operationen werden gemeinhin als zeitaufwendig angesehen, dies scheint aber stark von Komplexität und Anzahl abzuhängen. In dem hier dargestellten Beispiel werden lediglich zwei Relationen mittels einer JOIN-Operation zusammengefügt. Der JOIN wird dabei über eine einzige Spalte durchgeführt, die in beiden Relationen indexiert ist.
Zusätzlich sorgt die Abfrage-Optimierung von MariaDB dafür, dass vor dem JOIN möglichst viele Datensätze aussortiert werden.
In Abbildung 5.10 sind die Laufzeiten für verschieden große Datenmengen dargestellt.

Laufzeiten für JOIN-Operationen in MariaDB

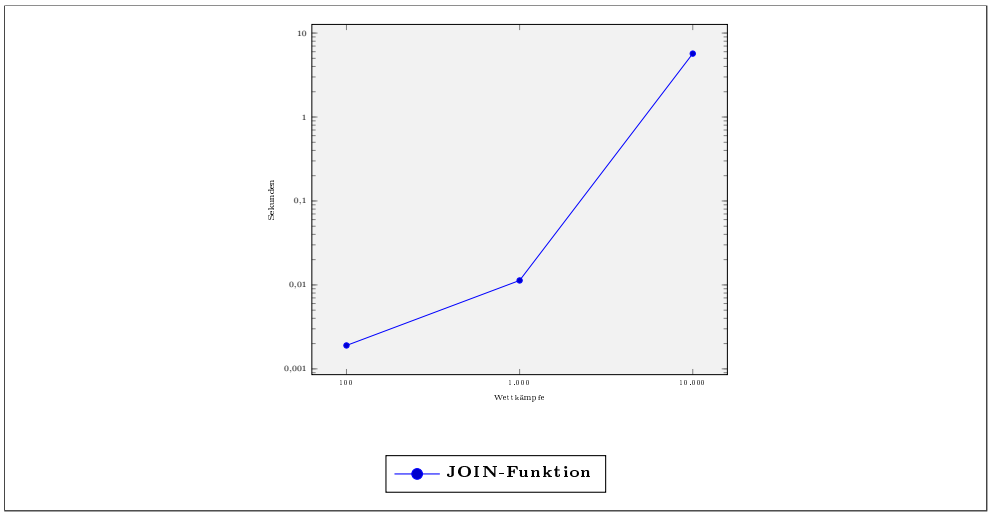

Abbildung 5.10: Die Grafik zeigt die Laufzeiten für eine per JOIN-Operation verbundene Abfrage auf zwei Relationen. Die Anzahl der Datensätze in den Relationen wird dabei immer mehr gesteigert.

5.4.4 Der Vergleich beider Umsetzungen

Das relationale MariaDB kann durch entsprechende JOIN-Operationen jede Art von Ergebnismenge liefern, da keinerlei Hierarchie in den Relationen vorliegen. Die Menge der JOINs für das Zusammensuchen der notwendigen Informationen kann jedoch einen nicht unerheblichen Zeitaufwand verursachen, abhängig davon, wie komplex das Ganze ausfällt.
Auch das Erstellen der JOIN-Operationen für die Zusammenführung der Informationen aus vielen Relationen kann durch das 'Impedance Mismatch'-Problem sehr aufwendig ausfallen.

MongoDB fällt mit seinen Aggregatsfunktionen erstmals hinter die Laufzeiten von MariaDB zurück. Die Zeitunterschiede sind allerdings bei kleinen Datenmengen überschaubar. Mit dem Anwachsen der Datenmenge ändert sich aber die Reihenfolge.

Interessant wäre ein alternativer Vergleich, bei dem die Informationen im relationalen System sehr viel stärker verteilt vorliegen, so dass die Anzahl der JOIN-Operationen notwendigerweise höher ausfällt.

5.5 Fazit

Zum Abschluss dieses Kapitels wird auf eine Grundproblematik bei der Vergleichbarkeit von MongoDB mit MariaDB eingegangen.
Danach werden die in diesem Kapitel gewonnenen Erkenntnisse nochmal kurz zusammengefasst.

5.5.1 Ein Grundsatzproblem beim Vergleich

Ein grundsätzliches Vergleichsproblem wurde innerhalb der vorherigen Abschnitte nicht betrachtet - die Übergabe des Abfrageergebnisses.
MariaDB übermittelt sämtliche Daten des Ergebnisses auf einmal, dies ist nicht unerheblich dem ACID-Konzept geschuldet. Würde das Ergebnis in Teilen gesendet, müssten die Restinformationen im Datenbanksystem entweder zwischengespeichert oder für Änderungen gesperrt werden.
In MongoDB werden große Datenmengen in Teilen übertragen, dabei fungiert ein zu Beginn übermittelter Cursor als Anforderungsmanager.

Eine solche Cursor-Systemik bietet MariaDB leider nicht. Es existiert zwar ein Cursor, dieser ist jedoch nur ein temporäres Konstrukt und ist ausschließlich innerhalb von MariaDB nutzbar.[10] Um einen direkten Vergleich der Abfragezeiten zu ermöglichen, wird der MongoDB-Cursor solange iteriert, bis das gesamte Ergebnis der Abfrage übermittelt wurde. Die Ergebnisse dieses Vergleichs sind in Tabelle 5.7 aufgeführt.

Die Iteration des MongoDB-Cursors

MongoDB			MariaDB
Cursor	gespeichert	iteriert	
0,0005	0,6104	0,5934	0,1384

Tabelle 5.7: Es werden die Abfragezeiten für eine Sortierabfrage in MariaDB und MongoDB bei 100.000 Athleten, ohne Offset und ohne Limit bei einem vorhandenen Index mit der Laufzeit eines iterierten MongoDB-Cursors (gespeichert und ausschließlich iteriert) verglichen. Die Testbedingungen sind in Kapitel 5.1 genannt.

Ohne Iteration siegt MongoDB auf ganzer Linie. Wird der Cursor hingegen iteriert, liegt MariaDB vorne.
In beiden Fällen hinkt der Vergleich, da MariaDB standardmäßig alle Daten auf einmal sendet, MongoDB dagegen portionsweise die Daten liefert.
Wird der MongoDB-Cursor iteriert, hat die verwendete Programmsprache (in diesem Fall PHP) Einfluss auf den Ausgang, da ihre Ausführungsgeschwindigkeit das Tempo der Iteration bestimmt.
Die Abfragegeschwindigkeit sollte daher nur marginal in den Vergleich einfließen.

5.5.2 Zusammenfassung

Das zu Beginn dieses Kapitels genannte Zitat von Fowler kann auf Grund der gewonnenen Erkenntnisse durchaus bestätigt werden.
Ein direkter und unmittelbarer Vergleich zweier Datenbanksysteme scheitert in diesem Fall an der kompletten Unterschiedlichkeit von MongoDB und MariaDB. Beide Systeme haben ihre Stärken und Schwächen, die für ihren jeweiligen Einsatzzweck optimiert sind.

Trotzdem soll an dieser Stelle ein kurzer Eignungsvergleich für die zuvor durchgeführten Testreihen gezogen werden.

[10] [17e]

In allen Testreihen können MongoDB und MariaDB gute bis sehr gute Leistungsergebnisse vorweisen. Weitaus wichtiger als die Menge der hinterlegten Daten, ist die Vorbereitung auf mögliche Anfragen. Die Verwendung von Indexen sollte, insbesondere bei steigenden Datenmengen, als verpflichtend betrachtet werden.

MongoDB hat hier einen Vorteil, da die Indexe auch nachträglich erfasst werden können. Beim standardmäßigen Speichersubsystem von MariaDB muss der Index vor dem ersten Eintrag vorhanden sein.

Sofern eine Abfrage eine sehr große Datenmenge liefert, die jenseits der hier betrachteten Größen liegt, hat MongoDB mit seinem Cursor einen klaren Vorteil. Da das Ergebnis portionsweise übermittelt wird, kann mit dieser Teilinformation schon weitergearbeitet werden. In MariaDB muss erst auf die Übermittlung des gesamten Ergebnisses gewartet werden, bevor dieses genutzt werden kann.

6 Fazit

Zum Abschluss dieser Ausarbeitung sollen ein paar Erkenntnisse und Gedanken zu den zwei Schwerpunkten dieser Arbeit genannt werden.
Während der Vergleich der verschiedenen nichtrelationalen Systeme zu Beginn einen nicht unerheblichen Anteil an Zeit benötigte, verbrauchte die Realisierung eines lauffähigen Prototypen in MongoDB und der anschließende Vergleich mit der relationalen Umsetzung einen ebenso großen Anteil.

6.1 Vergleich der nichtrelationalen Systeme

Der Bereich der nichtrelationalen Datenbankensysteme stellte sich als weitaus umfassender und weitläufiger dar, als es den Autoren im Vorfeld und in der Planung zu dieser Ausarbeitung bewusst war.

Trotz der Eingrenzung auf einen bestimmten Bereich der nichtrelationalen Systeme blieb nicht genug Zeit, um ihnen allen eine umfassende Analyse und Betrachtung zukommen zu lassen. Daraus resultierend wurde ein einfaches und mehrstufiges Verfahren konzipiert, um ausreichend schnell ein Auswahlergebnis zu erhalten.
Die Festlegung auf dokumentenbasierte Datenbanksysteme und insbesondere MongoDB folgte der simplen Zugänglichkeit der Dokumentenstruktur und der weitreichenden Verfügbarkeit an Ressourcen.

Die Graphendatenbanksysteme boten auch einen gewissen Reiz, erforderten jedoch einen völlig anderen Denkansatz zur Strukturierung, hier besser zur Fragmentierung, der Daten. Da der dafür notwendige Einarbeitungszeitraum nicht ausreichend abgeschätzt werden konnte, wurden es nicht weiter verfolgt.

Die anderen beiden Gruppen ließen keinen unmittelbaren Vorteil gegenüber der relationalen Lösung erkennen oder ihnen schienen gewisse Eigenschaften zu fehlen, die für Umsetzung aber als notwendig erachtet wurden.

Es lässt sich aber sagen, dass ein dokumentenbasiertes System wie MongoDB ein sehr naheliegender Ansatz zur Umsetzung ist.

6.2 Vergleich der Umsetzungen

In Kapitel 5 wurden bereits vergleichende Betrachtungen von MongoDB und MariaDB gemacht. Daher soll an dieser Stelle noch kurz auf die Eignung im Rahmen der Projektanforderung als leichtathletische Mehrkampfdatenbank eingegangen werden.
Ausgehend von der in Kapitel 3.1.4 genannten Datenmenge, die es zu verwalten gilt, sind beide Systeme ausreichend stark dimensioniert.

Beide Systeme haben, in den Augen der Autoren, eine Eigenschaft, die besonders hervorsticht, da sie dem jeweils anderen System fehlt.
In einem relationalen Systemen, wie MariaDB, kann durch Normalisierung das Vorkommen redundanten Dateninformationen vermieden werden. Das dokumentenbasierte MongoDB ermöglicht hingegen durch seine Schemafreiheit eine einfache nachträgliche Erweiterung, basierend auf dem impliziten Schema der darüber liegenden Anwendung.
Welche der beiden Eigenschaften einen höheren Stellenwert besitzt, ist von der jeweiligen Projektanforderung abhängig. Aus Sicht der Autoren lässt sich durch entsprechende Konstrukte in einer darüber liegenden Anwendung, eine (zumindest teilweise) redundanzfreie Struktur in MongoDB leichter implementieren, als das Konzept der Schemafreiheit in MariaDB. Letzteres ist nämlich nicht möglich.

Abbildungsverzeichnis

2.1 Speicherung bei zeilen- und spaltenorientierten Systemen 9

3.1 Das 'Impedance Mismatch'-Problem 16
3.2 Das relationale Modell der leichtathletischen Mehrkampfdatenbank 16
3.3 SQL-Befehl zur Abfrage eines Wettkampfes 17
3.4 Wettkampf-Aggregate in der Mehrkampfdatenbank 17
3.5 Die aggregierten Informationen zu einem Athleten 18
3.6 Schemafreiheit im Datenbanksystem 19
3.7 Problem der Schemafreiheit 19
3.8 Die ACID-BASE-Skala 21
3.9 Die 'Lost-Update'-Problematik 23
3.10 pessimistischer Offline-Lock 24
3.11 optimistischer Offline-Lock 25
3.12 Beispielhafte Aufteilung der Daten (Sharding) 26
3.13 Hashing des Schlüssels in Key/Value-Systemen 31

4.1 Dokumentenbasierte Datenbanksysteme im Ranking 34
4.2 Die Modelle der Dokumente 40

5.1 Laufzeiten für die Sortierung der Athleten in MongoDB 44
5.2 Laufzeiten für die Sortierung der Athleten in MariaDB 46
5.3 Laufzeiten für die Athletensuche in MongoDB 49
5.4 Laufzeiten für die Athletensuche in MariaDB 50
5.5 Suche in verschachtelten Dokumenten 52
5.6 Suche in Zuordnungstabellen 53
5.7 MAP-Vorgang als erster Teilschritt der MAP-REDUCE-Funktion 56
5.8 REDUCE-Vorgang als zweiter Teilschritt der MAP-REDUCE-Funktion 57
5.9 Laufzeiten der Aggregatsfunktionen von MongoDB 58
5.10 Laufzeiten für JOIN-Operationen in MariaDB 59

Tabellenverzeichnis

2.1 Verwendung von 'Row Keys' in spaltenorientierten Systemen 10

4.1 API-Unterstützung der dokumentenbasierten Datenbanksysteme 38

5.1 Zusammenfassung der Testparameter für die Athletensortierung 43
5.2 Auszug zum Profiling in MariaDB 47
5.3 Zusammenfassung der Testparameter für die Athletensuche 48
5.4 Datenumfang für JOIN- und Aggregationstests 55
5.5 Zeitvergleich der MongoDB Aggregatsfunktionen 58
5.6 Optimierung der Aggregation-Pipeline 59
5.7 Die Iteration des MongoDB-Cursors 61

Literaturverzeichnis

[Aba12] Daniel Abadi. „Consistency Tradeoffs in Modern Distributed Database System Design: CAP Is Only Part of the Story“. In: *Computer* 45.2 (Feb. 2012), S. 37–42. ISSN: 0018-9162. DOI: 10.1109/MC.2012.33. URL: http://ieeexplore.ieee.org/document/6127847/ (besucht am 18. 01. 2017).

[ALS10] J. Chris Anderson, Jan Lehnardt und Noah Slater. *CouchDB: The Definitive Guide*. 1st ed. OCLC: ocn320189640. Sebastopol, CA: O'Reilly, 2010. 245 S. ISBN: 978-0-596-15589-6.

[Cel14] Joe Celko. *Joe Celko's Complete Guide to NoSQL: What Every SQL Professional Needs to Know about Nonrelational Databases*. Amsterdam ; Boston: Elsevier/Morgan Kaufmann, 2014. 224 S. ISBN: 978-0-12-407192-6.

[DG04] Jeffrey Dean und Sanjay Ghemawat. „MapReduce: Simplified Data Processing on Large Clusters“. In: *Sixth Symposium on Operating System Design and Implementation*. OSDI'04. San Francisco, Dez. 2004. URL: https://www.usenix.org/legacy/publications/library/proceedings/osdi04/tech/full_papers/dean/dean.pdf.

[Edl10] Stefan Edlich, Hrsg. *NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*. OCLC: 680727498. München: Hanser, 2010. 289 S. ISBN: 978-3-446-42355-8.

[FB99] A. Fox und E.A. Brewer. „Harvest, Yield, and Scalable Tolerant Systems“. In: IEEE Comput. Soc, 1999, S. 174–178. ISBN: 978-0-7695-0237-3. DOI: 10.1109/HOTOS.1999.798396. URL: http://ieeexplore.ieee.org/document/798396/ (besucht am 17. 01. 2017).

[FM16a] Matthias Frank und Daniel Morady. *Konzeption Und Umsetzung Einer Relationalen Zehnkampfdatenbank Mit Dem Laravel-Framework*. 2016.

[FM16b] Matthias Frank und Daniel Morady. *Planung Und Umsetzung Einer Erweiterten, Relationalen Mehrkampfdatenbank Mit Mehrbenutzerunterstützung Unter Verwendung Des Laravel-PHP-Frameworks Und MariaDB*. 2016.

[Fow03] Martin Fowler. *Patterns of Enterprise Application Architecture*. The Addison-Wesley signature series. Boston: Addison-Wesley, 2003. 533 S. ISBN: 978-0-321-12742-6.

[Fow15] Adam Fowler. *NoSQL for Dummies*. OCLC: 904288512. 2015. ISBN: 978-1-118-90574-6.

[Gra81] Jim Gray. „The Transaction Concept: Virtues and Limitations“. In: *Proceedings of Seventh International Conference on Very Large Databases*. Sep. 1981, S. 144–154. URL: http://research.microsoft.com/en-us/um/people/gray/papers/theTransactionConcept.pdf (besucht am 17. 01. 2017).

[Her12] Olaf Herden. *Spaltenbasierte Datenbanken - Ein Konzept Zur Handhabung Großer Datenmengen*. Nov. 2012. URL: http://www.gil-net.de/Publikationen/25_135.pdf.

[HR83] Theo Haerder und Andreas Reuter. „Principles of Transaction-Oriented Database Recovery“. In: *ACM Computing Surveys* 15.4 (2. Dez. 1983), S. 287–317. ISSN: 03600300. DOI: 10.1145/289.291. URL: http://portal.acm.org/citation.cfm?doid=289.291 (besucht am 17. 01. 2017).

[OPS15] Chinedu Kingsley Obasi, Oghenekaro Asagba Prince und Abasiama Ita Silas. „A Comparative Study of Consistency Theorems in Distributed Databases“. In: *African Journal of Computing & ICT*. Bd. 8. Sep. 2015. URL: https://static.secure.website/wscfus/5655211/1375818/v8n3p24-2015-ajocict.pdf (besucht am 18. 01. 2017).

[SF13] Pramod J. Sadalage und Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Upper Saddle River, NJ: Addison-Wesley, 2013. 164 S. ISBN: 978-0-321-82662-6.

[Vai13] Gaurav Vaish. *Getting Started with NoSQL: Your Guide to the World and Technology of NoSQL*. OCLC: 856803207. Birmingham: Packt Publishing, 2013. 123 S. ISBN: 978-1-84969-498-8.

[Yuh16] Noel Yuhanna. *Document Stores, Q3 2016*. 8. Sep. 2016, S. 14. URL: https://kloudrydermcaas.s3.amazonaws.com/Forrester/RES125581.pdf (besucht am 12. 12. 2016).

Internet-Quellen-Verzeichnis

[11] *Datenbanken / Partitionierung Eines Graphen | Datenbanken Online Lexikon*. 11. Juli 2011. URL: http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.GraphPartitionierung (besucht am 25. 01. 2017).

[13] *NoSQL Distilled to an Hour by Martin Fowler*. Köln, 2013. URL: https://www.youtube.com/watch?v=ASiU89GlOFO&feature=youtu.be (besucht am 13. 01. 2017).

[16a] *DB-Engines Ranking - Die Rangliste Der Populärsten Document Stores*. 2016. URL: http://db-engines.com/de/ranking/document+store (besucht am 06. 12. 2016).

[16b] *Forrester*. 2016. URL: https://go.forrester.com/ (besucht am 12. 12. 2016).

[16c] *MongoDB for GIANT Ideas*. 2016. URL: https://www.mongodb.com/index (besucht am 24. 11. 2016).

[16d] *Open Source Database, Enterprise Database | MariaDB*. 2016. URL: https://mariadb.com/ (besucht am 24. 11. 2016).

[16e] *Use Indexes to Sort Query Results — MongoDB Manual 3.2*. 2016. URL: https://docs.mongodb.com/v3.2/tutorial/sort-results-with-indexes/ (besucht am 02. 12. 2016).

[17a] *Aggregation — MongoDB Manual 3.4*. 2017. URL: http://docs.mongodb.com/manual/aggregation (besucht am 03. 02. 2017).

[17b] *Amazon DynamoDB Product Details – Amazon Web Services*. 2017. URL: //aws.amazon.com/dynamodb/details/ (besucht am 26. 01. 2017).

[17c] *Apache CouchDB 2.0 Documentation - 1.5. The Core API*. 2017. URL: http://docs.couchdb.org/en/2.0.0/intro/api.html (besucht am 27. 01. 2017).

[17d] *Couchbase*. 2017. URL: https://developer.couchbase.com/open-source-projects (besucht am 27. 01. 2017).

[17e] *Cursor Overview*. 2017. URL: http://mariadb.com/kb/en/mariadb/cursor-overview/ (besucht am 10. 02. 2017).

[17f] *Do What You Could Never Do Before*. 2017. URL: https://www.mongodb.com/what-is-mongodb (besucht am 06. 02. 2017).

[17g] *mapReduce — MongoDB Manual 3.4*. 2017. URL: http://docs.mongodb.com/manual/reference/command/mapReduce (besucht am 07. 02. 2017).

[17h] *MongoDB API*. 2017. URL: http://api.mongodb.com/ (besucht am 27. 01. 2017).

[17i] *XtraDB and InnoDB*. 2017. URL: `http://mariadb.com/kb/en/mariadb/xtradb-and-innodb/` (besucht am 06. 02. 2017).

[Bre12] Eric Brewer. *CAP Twelve Years Later: How the "Rules"Have Changed*. 30. Mai 2012. URL: `https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed` (besucht am 17. 01. 2017).

[Fow13] Martin Fowler. *Schemaless Data Structures*. 7. Jan. 2013. URL: `https://martinfowler.com/articles/schemaless/` (besucht am 20. 01. 2017).

Messreihen

verwendetes System

CPU: Intel Core i5-4460 3.20 GHz (4 Kerne)

RAM: 8 GB

Betriebssystem: Windows 10 Home Edition 64-Bit

Software:

XAMPP (v3.2.2) für den MariaDB-Server (v10.1.13)

PHP (v7.0.6)

MongoDB (v3.2.10)

Ergebnisse

MariaDB - Sortierung Athleten - 10.000 Datensätze - mit Index

Messung		Messergebnisse (*in Sekunden*)					Median
ohne Offset	ohne Limit	0,0108	0,0403	0,0389	0,0274	0,0385	0,0370
		0,0349	0,0356	0,0388	0,0400	0,0255	
	1.000 Limit	0,0051	0,0051	0,0050	0,0052	0,0052	0,0051
		0,0051	0,0050	0,0050	0,0050	0,0053	
	100 Limit	0,0020	0,0012	0,0019	0,0013	0,0006	0,0013
		0,0008	0,0014	0,0012	0,0014	0,0013	
	10 Limit	0,0007	0,0008	0,0007	0,0007	0,0007	0,0007
		0,0008	0,0007	0,0007	0,0009	0,0007	
10% Offset	ohne Limit	0,0333	0,0383	0,0339	0,0268	0,0373	0,0341
		0,0337	0,0342	0,0374	0,0373	0,0339	
	1.000 Limit	0,0062	0,0062	0,0062	0,0062	0,0062	0,0062
		0,0064	0,0062	0,0063	0,0062	0,0065	
	100 Limit	0,0023	0,0025	0,0023	0,0023	0,0023	0,0023
		0,0023	0,0023	0,0023	0,0024	0,0024	
	10 Limit	0,0018	0,0018	0,0018	0,0019	0,0018	0,0018
		0,0020	0,0018	0,0018	0,0018	0,0018	
50% Offset	ohne Limit	0,0265	0,0262	0,0207	0,0140	0,0261	0,0261
		0,0262	0,0262	0,0262	0,0260	0,0261	
	1.000 Limit	0,0105	0,0106	0,0106	0,0108	0,0106	0,0106
		0,0106	0,0110	0,0107	0,0105	0,0112	
	100 Limit	0,0067	0,0068	0,0068	0,0068	0,0068	0,0068
		0,0068	0,0068	0,0071	0,0068	0,0068	
	10 Limit	0,0062	0,0066	0,0063	0,0063	0,0064	0,0063
		0,0063	0,0066	0,0062	0,0063	0,0063	
90% Offset	ohne Limit	0,0151	0,0150	0,0152	0,0151	0,0151	0,0151
		0,0155	0,0150	0,0150	0,0150	0,0152	
	1.000 Limit	0,0152	0,0150	0,0154	0,0150	0,0152	0,0151
		0,0150	0,0151	0,0151	0,0153	0,0150	
	100 Limit	0,0053	0,0112	0,0070	0,0112	0,0111	0,0111
		0,0111	0,0111	0,0112	0,0112	0,0113	
	10 Limit	0,0106	0,0108	0,0109	0,0107	0,0107	0,0108
		0,0107	0,0106	0,0109	0,0110	0,0108	

MariaDB - Sortierung Athleten - 10.000 Datensätze - ohne Index

	Messung	Messergebnisse (*in Sekunden*)					Median
ohne Offset	ohne Limit	0,1080	0,0792	0,0922	0,0850	0,0837	0,0938
		0,0995	0,0947	0,0951	0,1176	0,0929	
	1.000 Limit	0,0724	0,0818	0,0736	0,1133	0,0837	0,0721
		0,0534	0,0690	0,0691	0,0676	0,0717	
	100 Limit	0,0426	0,0408	0,0475	0,0414	0,0408	0,0420
		0,0408	0,0407	0,0456	0,0476	0,0510	
	10 Limit	0,0410	0,0521	0,0270	0,0392	0,0305	0,0403
		0,0404	0,0405	0,0405	0,0401	0,0401	
10% Offset	ohne Limit	0,0989	0,1054	0,0807	0,1072	0,0943	0,0958
		0,0974	0,0839	0,0915	0,1166	0,0914	
	1.000 Limit	0,0838	0,0827	0,0794	0,0775	0,0688	0,0799
		0,0770	0,0804	0,0787	0,0998	0,0909	
	100 Limit	0,0762	0,0741	0,0724	0,0714	0,0606	0,0741
		0,0743	0,0945	0,0741	0,0863	0,0677	
	10 Limit	0,0729	0,0689	0,0689	0,0598	0,0694	0,0689
		0,0668	0,0824	0,0800	0,0662	0,0689	
50% Offset	ohne Limit	0,1043	0,0904	0,0931	0,1157	0,0908	0,0920
		0,0817	0,0793	0,0903	0,1150	0,1065	
	1.000 Limit	0,0613	0,0973	0,0834	0,0878	0,0828	0,0840
		0,0858	0,0826	0,0846	0,0833	0,0845	
	100 Limit	0,0811	0,0848	0,0840	0,0812	0,1054	0,0839
		0,0910	0,0823	0,0838	0,0822	0,1286	
	10 Limit	0,0861	0,0884	0,0836	0,0812	0,0836	0,0836
		0,0725	0,0830	0,0900	0,0888	0,0817	
90% Offset	ohne Limit	0,0976	0,0986	0,0918	0,0891	0,0933	0,0926
		0,0921	0,0858	0,1365	0,0931	0,0905	
	1.000 Limit	0,0962	0,0826	0,0802	0,0915	0,0889	0,0940
		0,0919	0,1034	0,0997	0,1119	0,1226	
	100 Limit	0,1240	0,1013	0,0875	0,0900	0,0876	0,0907
		0,0889	0,0967	0,0976	0,0872	0,0914	
	10 Limit	0,0719	0,0580	0,0888	0,1006	0,1458	0,0895
		0,0896	0,0873	0,0978	0,1119	0,0894	

MariaDB - Sortierung Athleten - 100.000 Datensätze - mit Index

	Messung	Messergebnisse (*in Sekunden*)					Median
ohne Offset	ohne Limit	0,6014	0,1386	0,1241	0,3948	0,1446	0,1384
		0,1382	0,1302	0,1435	0,1289	0,1111	
	1.000 Limit	0,0050	0,0053	0,0051	0,0050	0,0050	0,0050
		0,0051	0,0050	0,0055	0,0050	0,0050	
	100 Limit	0,0110	0,0012	0,0012	0,0012	0,0012	0,0012
		0,0012	0,0012	0,0012	0,0013	0,0012	
	10 Limit	0,0006	0,0009	0,0007	0,0007	0,0009	0,0007
		0,0007	0,0007	0,0009	0,0007	0,0007	
10% Offset	ohne Limit	0,1336	0,1158	0,1348	0,1259	0,1303	0,1266
		0,1273	0,1117	0,1290	0,1142	0,1245	
	1.000 Limit	0,0154	0,0159	0,0158	0,0159	0,0158	0,0158
		0,0159	0,0159	0,0157	0,0160	0,0158	
	100 Limit	0,0121	0,0120	0,0121	0,0121	0,0122	0,0121
		0,0119	0,0121	0,0120	0,0120	0,0120	
	10 Limit	0,0114	0,0117	0,0115	0,0116	0,0116	0,0116
		0,0115	0,0117	0,0116	0,0116	0,0116	
50% Offset	ohne Limit	0,0943	0,0984	0,1165	0,0958	0,0951	0,0954
		0,0940	0,0945	0,1012	0,1065	0,0937	
	1.000 Limit	0,0424	0,0319	0,0415	0,0427	0,0514	0,0416
		0,0331	0,0416	0,0416	0,0516	0,0351	
	100 Limit	0,0425	0,0402	0,0499	0,0364	0,0400	0,0402
		0,0401	0,0402	0,0371	0,0523	0,0401	
	10 Limit	0,0423	0,0521	0,0443	0,0398	0,0398	0,0411
		0,0540	0,0399	0,0331	0,0516	0,0397	
90% Offset	ohne Limit	0,0784	0,0575	0,0698	0,0668	0,0618	0,0683
		0,0810	0,0643	0,0740	0,0605	0,0698	
	1.000 Limit	0,0596	0,0515	0,0520	0,0500	0,0561	0,0517
		0,0472	0,0556	0,0500	0,0561	0,0322	
	100 Limit	0,0620	0,0717	0,0658	0,0659	0,0601	0,0621
		0,0651	0,0562	0,0548	0,0558	0,0622	
	10 Limit	0,0535	0,0555	0,0598	0,0421	0,0311	0,0496
		0,0322	0,0930	0,0456	0,0438	0,0592	

MariaDB - Sortierung Athleten - 100.000 Datensätze - ohne Index

	Messung	Messergebnisse (*in Sekunden*)					Median
ohne Offset	ohne Limit	1,2861	1,0354	1,0261	1,0231	1,0063	1,0201
		1,0201	1,0163	1,0201	1,0010	1,0064	
	1.000 Limit	0,4829	0,5003	0,4992	0,4853	0,4830	0,4922
		0,5212	0,4814	0,5272	0,4709	0,5125	
	100 Limit	0,2419	0,2402	0,2568	0,2507	0,2527	0,2487
		0,2538	0,2615	0,2388	0,2428	0,2467	
	10 Limit	0,2429	0,2321	0,2378	0,2478	0,2082	0,2444
		0,2663	0,2477	0,2459	0,2389	0,2531	
10% Offset	ohne Limit	1,1482	1,0169	1,0200	1,0216	1,0212	1,0184
		0,9969	1,0138	1,0142	1,0312	1,0103	
	1.000 Limit	0,8008	0,7368	0,7649	1,1610	0,8263	0,8126
		0,8863	0,8030	0,7294	0,8330	0,8221	
	100 Limit	0,7801	0,8948	0,7374	0,7459	0,7782	0,7959
		0,8754	0,8096	0,8036	0,8453	0,7882	
	10 Limit	0,8797	0,7936	0,7114	0,8703	0,8480	0,7960
		0,7953	1,1573	0,7966	0,7113	0,7138	
50% Offset	ohne Limit	1,0061	1,0077	1,0059	1,0945	0,9928	1,0069
		1,0535	0,9850	1,0948	0,9754	1,0812	
	1.000 Limit	0,8948	0,9359	0,8950	0,9996	0,9358	0,9061
		0,8979	0,9625	0,9143	0,8158	0,8367	
	100 Limit	1,0181	0,8630	0,9557	0,9304	0,8859	0,8849
		0,8824	0,9324	0,8839	0,8261	0,8800	
	10 Limit	0,9449	0,8796	0,9406	0,8711	0,9107	0,9025
		0,8205	0,9286	0,8639	1,0679	0,8943	
90% Offset	ohne Limit	0,9850	0,9909	0,9886	1,0217	0,9965	0,9983
		1,0000	0,9719	3,4495	1,1247	1,1717	
	1.000 Limit	0,9999	1,1428	0,9669	1,1256	0,9928	1,0081
		1,0162	0,9813	1,1325	0,9538	1,0804	
	100 Limit	0,9682	0,9813	0,9812	1,0744	0,9537	0,9820
		1,0254	0,9746	0,9827	0,9851	0,9943	
	10 Limit	0,9816	0,9685	0,8851	0,8879	1,3401	0,9599
		0,9767	0,9043	0,8878	0,9770	0,9513	

MariaDB - Sortierung Athleten - 1.000.000 Datensätze - mit Index

	Messung	Messergebnisse (*in Sekunden*)					Median
ohne Offset	ohne Limit	13,2168	6,3943	5,4398	4,8515	4,5281	4,6654
		4,6182	4,7127	3,9910	4,3682	4,1611	
	1.000 Limit	0,0191	0,0055	0,0056	0,0053	0,0053	0,0054
		0,0053	0,0052	0,0154	0,0053	0,0054	
	100 Limit	0,0012	0,0012	0,0013	0,0012	0,0012	0,0012
		0,0012	0,0012	0,0016	0,0012	0,0014	
	10 Limit	0,0007	0,0008	0,0007	0,0007	0,0008	0,0007
		0,0007	0,0007	0,0007	0,0007	0,0007	
10% Offset	ohne Limit	3,0740	3,0731	3,0338	3,0422	2,9809	3,0380
		3,0701	2,9729	3,2068	2,9161	2,6014	
	1.000 Limit	0,2474	0,0654	0,0593	0,0580	0,0537	0,0599
		0,0491	0,0678	0,0648	0,0482	0,0605	
	100 Limit	0,0654	0,0682	0,0549	0,0485	0,0796	0,0597
		0,0625	0,0485	0,0521	0,0570	0,0636	
	10 Limit	0,0745	0,0691	0,0711	0,0633	0,0610	0,0633
		0,0580	0,0632	0,0486	0,0618	0,0675	
50% Offset	ohne Limit	2,6957	2,9951	2,6052	2,8644	2,8720	2,9412
		3,0369	3,0604	3,4074	3,4753	2,8873	
	1.000 Limit	1,8891	1,7678	1,5044	1,5643	1,8033	1,7014
		1,8115	1,5484	1,7685	1,6351	1,5883	
	100 Limit	1,4834	1,5023	1,8463	1,5886	1,5287	1,5511
		1,5735	1,6190	1,4290	1,4974	1,6804	
	10 Limit	1,7044	1,5336	1,6377	1,5152	1,5557	1,5599
		1,5641	1,6586	1,5973	1,4753	1,4689	
90% Offset	ohne Limit	2,4798	3,6319	3,2469	3,5180	3,1370	3,2122
		3,2234	3,0417	3,2009	3,1291	3,2757	
	1.000 Limit	3,0896	3,1408	2,8898	3,1086	2,6289	2,7657
		2,6291	2,8133	2,7160	2,5139	2,7181	
	100 Limit	2,4925	2,7897	3,0361	2,8095	2,8598	2,8008
		2,8686	2,7921	2,7581	2,8303	2,7752	
	10 Limit	2,8826	3,0103	2,7301	2,5154	3,5683	2,8604
		2,8747	2,4667	2,2909	2,8462	3,0908	

MariaDB - Sortierung Athleten - 1.000.000 Datensätze - ohne Index

	Messung	Messergebnisse (*in Sekunden*)					Median
ohne Offset	ohne Limit	4.056,5684	4.059,2816	4.054,2722	4.057,2110	4.055,2720	4.057,3801
		4.057,7680	4.061,2262	4.057,5491	4.066,3465	4.056,1238	
	1.000 Limit	30,0989	32,9063	30,1311	31,0973	32,3888	30,2559
		31,0062	29,9387	30,1396	30,3721	29,8139	
	100 Limit	8,7370	9,1064	8,9904	9,1228	8,9653	8,9779
		8,2816	9,0315	8,7235	8,6062	9,0738	
	10 Limit	8,7597	9,0237	8,4311	9,0825	9,0321	9,0524
		9,2079	9,0316	9,0904	9,0727	9,1250	
10% Offset	ohne Limit	4.054,4901	4.057,7719	4.055,3891	4.055,8430	4.055,5423	4.056,5909
		4.055,3679	4.057,8488	4.057,9049	4.058,4103	4.057,3388	
	1.000 Limit	462,1031	458,9834	462,9840	458,0084	460,5758	459,9126
		461,7170	459,9335	459,4169	457,7005	459,8918	
	100 Limit	455,1759	455,1586	457,2586	454,3341	459,3177	456,6133
		456,7260	458,3756	455,1504	459,4254	456,5006	
	10 Limit	454,3835	457,7597	455,1341	457,0093	457,1914	456,4334
		459,5513	455,8839	456,9830	455,5107	455,3917	
50% Offset	ohne Limit	4.058,2406	4.058,2655	4.056,4264	4.053,7105	4.059,9624	4.064,1289
		4.068,2954	4.081,4753	4.083,8496	4.082,0511	4.082,4278	
	1.000 Limit	2.071,0503	2.071,1297	2.069,4298	2.069,3130	2.067,9463	2.069,3797
		2.069,9465	2.069,1796	2.069,3296	2.067,3218	2.070,6134	
	100 Limit	2.058,0544	2.056,0740	2.054,0808	2.060,5459	2.057,2890	2.055,8728
		2.053,9233	2.055,6715	2.051,4482	2.056,6879	2.052,5630	
	10 Limit	2.053,7057	2.054,1395	2.051,1189	2.055,2017	2.058,5540	2.053,9595
		2.048,4631	2.056,4394	2.053,7795	2.051,7890	2.056,5883	
90% Offset	ohne Limit	4.053,2955	4.056,9526	4.058,4090	4.053,2436	4.056,7844	4.056,3746
		4.055,9649	4.058,3680	4.057,0799	4.053,6897	4.053,8419	
	1.000 Limit	3.701,4083	3.706,1212	3.703,5785	3.700,4083	3.704,1542	3.703,3516
		3.700,7790	3.703,1247	3.698,9589	3.706,0657	3.704,3546	
	100 Limit	3.695,4513	3.699,7836	3.699,7752	3.702,7933	3.700,1430	3.700,0800
		3.700,4251	3.701,4259	3.695,9181	3.700,0170	3.705,7839	
	10 Limit	3.658,1751	3.659,9998	3.656,8432	3.659,7841	3.658,0255	3.659,6760
		3.663,7761	3.658,1593	3.663,1262	3.659,5678	3.660,4436	

MongoDB - Sortierung Athleten - 10.000 Datensätze - mit Index

	Messung	Messergebnisse (*in Sekunden*)					Median
ohne Offset	ohne Limit	0,0123	0,0007	0,0005	0,0006	0,0007	0,0006
		0,0006	0,0006	0,0006	0,0009	0,0006	
	1.000 Limit	0,0011	0,0008	0,0005	0,0005	0,0006	0,0007
		0,0007	0,0006	0,0007	0,0008	0,0009	
	100 Limit	0,0009	0,0006	0,0005	0,0006	0,0006	0,0006
		0,0005	0,0006	0,0006	0,0006	0,0006	
	10 Limit	0,0003	0,0003	0,0003	0,0003	0,0003	0,0003
		0,0003	0,0003	0,0003	0,0003	0,0003	
10% Offset	ohne Limit	0,0061	0,0047	0,0052	0,0049	0,0049	0,0052
		0,0052	0,0052	0,0052	0,0049	0,0053	
	1.000 Limit	0,0053	0,0055	0,0047	0,0057	0,0049	0,0052
		0,0050	0,0056	0,0051	0,0054	0,0049	
	100 Limit	0,0056	0,0048	0,0052	0,0055	0,0048	0,0053
		0,0054	0,0055	0,0048	0,0048	0,0058	
	10 Limit	0,0043	0,0051	0,0045	0,0051	0,0047	0,0045
		0,0045	0,0043	0,0049	0,0045	0,0046	
50% Offset	ohne Limit	0,0225	0,0200	0,0207	0,0217	0,0208	0,0206
		0,0207	0,0206	0,0202	0,0203	0,0200	
	1.000 Limit	0,0219	0,0217	0,0208	0,0203	0,0199	0,0203
		0,0201	0,0204	0,0202	0,0210	0,0198	
	100 Limit	0,0202	0,0200	0,0213	0,0200	0,0194	0,0200
		0,0201	0,0200	0,0198	0,0195	0,0198	
	10 Limit	0,0194	0,0200	0,0202	0,0205	0,0200	0,0201
		0,0214	0,0213	0,0200	0,0199	0,0201	
90% Offset	ohne Limit	0,0350	0,0348	0,0347	0,0347	0,0344	0,0347
		0,0348	0,0353	0,0335	0,0352	0,0336	
	1.000 Limit	0,0346	0,0340	0,0342	0,0347	0,0347	0,0346
		0,0346	0,0346	0,0345	0,0356	0,0341	
	100 Limit	0,0343	0,0348	0,0362	0,0351	0,0353	0,0348
		0,0347	0,0349	0,0356	0,0340	0,0347	
	10 Limit	0,0350	0,0347	0,0349	0,0355	0,0341	0,0344
		0,0340	0,0339	0,0340	0,0340	0,0357	

MongoDB - Sortierung Athleten - 10.000 Datensätze - ohne Index

	Messung	Messergebnisse (*in Sekunden*)					Median
ohne Offset	ohne Limit	0,0124 0,0005	0,0007 0,0005	0,0005 0,0005	0,0005 0,0005	0,0005 0,0005	0,0005
	1.000 Limit	0,0005 0,0007	0,0004 0,0006	0,0004 0,0007	0,0005 0,0007	0,0007 0,0005	0,0006
	100 Limit	0,0007 0,0005	0,0007 0,0005	0,0006 0,0005	0,0005 0,0005	0,0005 0,0005	0,0005
	10 Limit	0,0004 0,0003	0,0003 0,0002	0,0003 0,0003	0,0003 0,0002	0,0003 0,0003	0,0003
10% Offset	ohne Limit	0,0064 0,0033	0,0038 0,0031	0,0033 0,0028	0,0033 0,0032	0,0028 0,0031	0,0032
	1.000 Limit	0,0036 0,0034	0,0028 0,0029	0,0031 0,0024	0,0033 0,0033	0,0032 0,0034	0,0032
	100 Limit	0,0031 0,0034	0,0036 0,0032	0,0034 0,0034	0,0030 0,0029	0,0031 0,0031	0,0032
	10 Limit	0,0025 0,0031	0,0022 0,0035	0,0028 0,0030	0,0027 0,0037	0,0021 0,0038	0,0029
50% Offset	ohne Limit	0,0245 0,0124	0,0117 0,0117	0,0116 0,0114	0,0122 0,0116	0,0124 0,0118	0,0118
	1.000 Limit	0,0122 0,0122	0,0125 0,0120	0,0120 0,0115	0,0117 0,0123	0,0120 0,0120	0,0120
	100 Limit	0,0122 0,0122	0,0121 0,0132	0,0120 0,0135	0,0119 0,0124	0,0128 0,0118	0,0122
	10 Limit	0,0120 0,0123	0,0115 0,0116	0,0112 0,0112	0,0120 0,0119	0,0114 0,0119	0,0117
90% Offset	ohne Limit	0,0327 0,0196	0,0196 0,0200	0,0201 0,0205	0,0209 0,0201	0,0197 0,0199	0,0200
	1.000 Limit	0,0198 0,0215	0,0201 0,0206	0,0203 0,0202	0,0197 0,0203	0,0202 0,0202	0,0202
	100 Limit	0,0204 0,0196	0,0201 0,0201	0,0205 0,0200	0,0199 0,0198	0,0200 0,0196	0,0200
	10 Limit	0,0192 0,0203	0,0193 0,0202	0,0201 0,0200	0,0197 0,0204	0,0200 0,0200	0,0200

MongoDB - Sortierung Athleten - 100.000 Datensätze - mit Index

	Messung	Messergebnisse (*in Sekunden*)					Median
ohne Offset	ohne Limit	0,0036 0,0004	0,0006 0,0004	0,0005 0,0005	0,0005 0,0005	0,0005 0,0006	0,0005
	1.000 Limit	0,0006 0,0004	0,0005 0,0004	0,0007 0,0004	0,0007 0,0007	0,0005 0,0006	0,0005
	100 Limit	0,0005 0,0006	0,0004 0,0005	0,0005 0,0005	0,0006 0,0007	0,0005 0,0006	0,0005
	10 Limit	0,0003 0,0003	0,0003 0,0003	0,0002 0,0003	0,0003 0,0003	0,0003 0,0003	0,0003
10% Offset	ohne Limit	0,0249 0,0225	0,0223 0,0216	0,0219 0,0219	0,0216 0,0213	0,0220 0,0224	0,0220
	1.000 Limit	0,0216 0,0216	0,0218 0,0224	0,0220 0,0222	0,0227 0,0215	0,0217 0,0225	0,0219
	100 Limit	0,0233 0,0226	0,0223 0,0221	0,0228 0,0224	0,0218 0,0221	0,0225 0,0216	0,0224
	10 Limit	0,0220 0,0220	0,0223 0,0216	0,0227 0,0221	0,0221 0,0226	0,0214 0,0215	0,0220
50% Offset	ohne Limit	0,1014 0,0999	0,1000 0,0999	0,1005 0,0987	0,1006 0,0984	0,1015 0,0979	0,1000
	1.000 Limit	0,1033 0,1042	0,1032 0,1048	0,1034 0,1009	0,1025 0,1007	0,1036 0,0996	0,1033
	100 Limit	0,0946 0,0943	0,0932 0,0943	0,0970 0,0996	0,1134 0,0950	0,0977 0,0946	0,0948
	10 Limit	0,0957 0,0946	0,0946 0,0935	0,0955 0,0944	0,0965 0,0956	0,0950 0,0941	0,0948
90% Offset	ohne Limit	0,1742 0,1770	0,1689 0,1805	0,1688 0,1774	0,1720 0,1776	0,1814 0,1764	0,1767
	1.000 Limit	0,1779 0,1757	0,1783 0,1822	0,1774 0,1757	0,1789 0,1755	0,1772 0,1788	0,1777
	100 Limit	0,1815 0,1838	0,1839 0,1818	0,1830 0,1866	0,1838 0,1835	0,1834 0,1850	0,1837
	10 Limit	0,1842 0,1825	0,1829 0,1862	0,1859 0,1842	0,1855 0,1840	0,1843 0,1836	0,1842

MongoDB - Sortierung Athleten - 100.000 Datensätze - ohne Index

	Messung	Messergebnisse (*in Sekunden*)					Median
ohne Offset	ohne Limit	0,0042	0,0006	0,0006	0,0005	0,0005	0,0005
		0,0006	0,0005	0,0005	0,0005	0,0005	
	1.000 Limit	0,0005	0,0005	0,0007	0,0006	0,0005	0,0005
		0,0007	0,0006	0,0005	0,0005	0,0005	
	100 Limit	0,0006	0,0005	0,0005	0,0005	0,0004	0,0005
		0,0004	0,0005	0,0005	0,0006	0,0006	
	10 Limit	0,0003	0,0003	0,0003	0,0003	0,0002	0,0003
		0,0003	0,0002	0,0003	0,0003	0,0003	
10% Offset	ohne Limit	0,0216	0,0192	0,0189	0,0179	0,0184	0,0189
		0,0188	0,0181	0,0180	0,0189	0,0202	
	1.000 Limit	0,0185	0,0190	0,0184	0,0183	0,0178	0,0185
		0,0207	0,0179	0,0202	0,0188	0,0185	
	100 Limit	0,0189	0,0190	0,0186	0,0190	0,0209	0,0196
		0,0205	0,0204	0,0212	0,0198	0,0195	
	10 Limit	0,0174	0,0189	0,0192	0,0192	0,0174	0,0183
		0,0189	0,0201	0,0178	0,0175	0,0174	
50% Offset	ohne Limit	0,0871	0,0829	0,0830	0,0837	0,0846	0,0836
		0,0809	0,0835	0,0821	0,0857	0,0837	
	1.000 Limit	0,0830	0,0854	0,0842	0,0844	0,0845	0,0845
		0,0845	0,0841	0,0830	0,0856	0,0846	
	100 Limit	0,0836	0,0830	0,0842	0,0828	0,0859	0,0843
		0,0860	0,0845	0,0849	0,0827	0,0861	
	10 Limit	0,0823	0,0822	0,0844	0,0861	0,0838	0,0841
		0,0852	0,0900	0,0836	0,0850	0,0825	
90% Offset	ohne Limit	0,1484	0,1496	0,1516	0,1490	0,1494	0,1495
		0,1484	0,1513	0,1488	0,1513	0,1505	
	1.000 Limit	0,1533	0,1538	0,1553	0,1580	0,1543	0,1545
		0,1579	0,1549	0,1548	0,1538	0,1511	
	100 Limit	0,1490	0,1582	0,1555	0,1568	0,1542	0,1524
		0,1540	0,1476	0,1500	0,1508	0,1500	
	10 Limit	0,1573	0,1560	0,1550	0,1568	0,1567	0,1559
		0,1511	0,1558	0,1569	0,1552	0,1552	

MongoDB - Sortierung Athleten - 1.000.000 Datensätze - mit Index

	Messung	Messergebnisse (*in Sekunden*)					Median
ohne Offset	ohne Limit	0,0036	0,0007	0,0007	0,0006	0,0005	0,0006
		0,0006	0,0006	0,0006	0,0006	0,0006	
	1.000 Limit	0,0008	0,0006	0,0007	0,0006	0,0006	0,0006
		0,0006	0,0006	0,0006	0,0006	0,0007	
	100 Limit	0,0006	0,0006	0,0008	0,0007	0,0007	0,0006
		0,0006	0,0005	0,0006	0,0005	0,0007	
	10 Limit	0,0005	0,0005	0,0003	0,0003	0,0003	0,0003
		0,0003	0,0003	0,0003	0,0004	0,0003	
10% Offset	ohne Limit	0,2477	0,2351	0,2432	0,2350	0,2519	0,2409
		0,2370	0,2398	0,2418	0,2401	0,2446	
	1.000 Limit	0,2454	0,2449	0,2418	0,2430	0,2423	0,2432
		0,2435	0,2410	0,2371	0,2471	0,2479	
	100 Limit	0,2430	0,2428	0,2399	0,2359	0,2456	0,2414
		0,2392	0,2481	0,2426	0,2379	0,2403	
	10 Limit	0,2485	0,2566	0,2357	0,2397	0,2391	0,2397
		0,2441	0,2380	0,2390	0,2397	0,2398	
50% Offset	ohne Limit	1,1609	1,1588	1,1640	1,1587	1,1717	1,1598
		1,1628	1,1689	1,1568	1,1535	1,1500	
	1.000 Limit	1,2026	1,1779	1,1894	1,2282	1,2255	1,2028
		1,2122	1,2207	1,2030	1,1791	1,1839	
	100 Limit	1,1726	1,1733	1,1560	1,1797	1,2158	1,1793
		1,2096	1,2090	1,2067	1,1675	1,1789	
	10 Limit	1,1767	1,1782	1,1818	1,1797	1,1850	1,1781
		1,1922	1,1736	1,1759	1,1781	1,1747	
90% Offset	ohne Limit	2,1561	2,1540	2,1315	2,0743	2,0896	2,0873
		2,0736	2,0717	2,0864	2,0881	2,0662	
	1.000 Limit	2,0703	2,1124	2,0947	2,0945	2,1185	2,1154
		2,1770	2,1818	2,1507	2,1628	2,1089	
	100 Limit	2,1574	2,1246	2,1471	2,1134	2,1144	2,1523
		2,1277	2,1589	2,1748	2,1718	2,1612	
	10 Limit	2,1055	2,0969	2,0957	2,1293	2,0993	2,0981
		2,1075	2,1059	2,0867	2,0963	2,0706	

MariaDB - Athletensuche - 10.000 Datensätze - mit Index

	Messung	Messergebnisse (*in Sekunden*)					Median
Komplett	Nachname	0,0185	0,0013	0,0012	0,0013	0,0014	0,0013
		0,0015	0,0013	0,0012	0,0014	0,0012	
	Vorname	0,0026	0,0013	0,0013	0,0014	0,0015	0,0014
		0,0016	0,0015	0,0013	0,0012	0,0014	
	Nach- und Vorname	0,0010	0,0011	0,0010	0,0012	0,0009	0,0010
		0,0010	0,0010	0,0011	0,0014	0,0011	
Teil	Nachname	0,0337	0,0231	0,0233	0,0225	0,0200	0,0225
		0,0224	0,0226	0,0222	0,0223	0,0223	
	Vorname	0,0131	0,0191	0,0190	0,0211	0,0212	0,0211
		0,0211	0,0213	0,0211	0,0215	0,0212	
	Nach- und Vorname	0,0197	0,0198	0,0195	0,0198	0,0195	0,0197
		0,0199	0,0200	0,0178	0,0196	0,0202	
Anfang	Nachname	0,0015	0,0015	0,0012	0,0015	0,0017	0,0015
		0,0015	0,0015	0,0015	0,0016	0,0015	
	Vorname	0,0013	0,0013	0,0013	0,0014	0,0013	0,0013
		0,0013	0,0013	0,0079	0,0030	0,0006	
	Nach- und Vorname	0,0005	0,0005	0,0005	0,0004	0,0004	0,0005
		0,0005	0,0005	0,0004	0,0005	0,0005	

MariaDB - Athletensuche - 10.000 Datensätze - ohne Index

	Messung	Messergebnisse (*in Sekunden*)					Median
Komplett	Nachname	0,0040	0,0069	0,0169	0,0163	0,0168	0,0166
		0,0165	0,0166	0,0171	0,0166	0,0170	
	Vorname	0,0174	0,0164	0,0168	0,0166	0,0167	0,0168
		0,0168	0,0168	0,0168	0,0103	0,0168	
	Nach- und Vorname	0,0203	0,0199	0,0172	0,0182	0,0200	0,0198
		0,0203	0,0182	0,0199	0,0198	0,0197	
Teil	Nachname	0,0177	0,0259	0,0231	0,0100	0,0101	0,0212
		0,0226	0,0226	0,0227	0,0181	0,0199	
	Vorname	0,0324	0,0342	0,0343	0,0341	0,0272	0,0335
		0,0343	0,0330	0,0325	0,0272	0,0342	
	Nach- und Vorname	0,0204	0,0184	0,0202	0,0200	0,0202	0,0200
		0,0201	0,0199	0,0185	0,0202	0,0200	
Anfang	Nachname	0,0218	0,0171	0,0069	0,0165	0,0170	0,0171
		0,0171	0,0171	0,0172	0,0171	0,0166	
	Vorname	0,0172	0,0169	0,0174	0,0170	0,0167	0,0170
		0,0170	0,0162	0,0171	0,0169	0,0165	
	Nach- und Vorname	0,0061	0,0162	0,0168	0,0167	0,0164	0,0163
		0,0164	0,0160	0,0164	0,0161	0,0162	

MariaDB - Athletensuche - 100.000 Datensätze - mit Index

	Messung	Messergebnisse (*in Sekunden*)					Median
Komplett	Nachname	0,1747	0,0043	0,0042	0,0043	0,0042	0,0043
		0,0044	0,0046	0,0043	0,0042	0,0046	
	Vorname	0,0491	0,0039	0,0040	0,0035	0,0037	0,0036
		0,0054	0,0035	0,0034	0,0033	0,0033	
	Nach- und Vorname	0,0010	0,0010	0,0010	0,0010	0,0010	0,0010
		0,0010	0,0010	0,0010	0,0010	0,0010	
Teil	Nachname	0,1393	0,0973	0,1040	0,0916	0,1049	0,1044
		0,1066	0,1040	0,1109	0,0873	0,1077	
	Vorname	0,1169	0,1074	0,0832	0,0954	0,1032	0,1035
		0,1073	0,1038	0,1005	0,1000	0,1087	
	Nach- und Vorname	0,0826	0,1197	0,0973	0,0730	0,1047	0,0923
		0,0964	0,1053	0,0852	0,0865	0,0882	
Anfang	Nachname	0,0278	0,0191	0,0220	0,0179	0,0133	0,0147
		0,0145	0,0150	0,0133	0,0135	0,0130	
	Vorname	0,0241	0,0044	0,0026	0,0041	0,0039	0,0040
		0,0046	0,0045	0,0035	0,0034	0,0033	
	Nach- und Vorname	0,0027	0,0017	0,0017	0,0017	0,0016	0,0017
		0,0016	0,0019	0,0017	0,0020	0,0017	

MariaDB - Athletensuche - 100.000 Datensätze - ohne Index

	Messung	Messergebnisse (*in Sekunden*)					Median
Komplett	Nachname	0,0853	0,1378	0,1372	0,1345	0,1274	0,1375
		0,1318	0,1493	0,1543	0,1543	0,1416	
	Vorname	0,1575	0,1523	0,1168	0,1524	0,1723	0,1290
		0,1191	0,1320	0,1260	0,1216	0,1243	
	Nach- und Vorname	0,1468	0,1447	0,1491	0,1566	0,1550	0,1480
		0,1437	0,1589	0,1597	0,1263	0,1462	
Teil	Nachname	0,1736	0,1599	0,1607	0,1582	0,1448	0,1610
		0,2007	0,1646	0,1561	0,1625	0,1612	
	Vorname	0,1627	0,2028	0,1760	0,1671	0,1702	0,1687
		0,1527	0,1579	0,2053	0,1771	0,1554	
	Nach- und Vorname	0,1550	0,1444	0,1600	0,1613	0,1587	0,1566
		0,1618	0,1582	0,1415	0,1356	0,1365	
Anfang	Nachname	0,1899	0,1570	0,1528	0,1437	0,1520	0,1524
		0,1405	0,1552	0,1727	0,1495	0,1254	
	Vorname	0,1528	0,1448	0,1567	0,1665	0,1814	0,1463
		0,1471	0,1455	0,1317	0,1307	0,1451	
	Nach- und Vorname	0,1400	0,1808	0,1339	0,1380	0,1350	0,1387
		0,1366	0,1512	0,1458	0,1384	0,1389	

MariaDB - Athletensuche - 1.000.000 Datensätze - mit Index

	Messung	Messergebnisse (*in Sekunden*)					Median
Komplett	Nachname	5,6468	5,3579	5,9049	5,8375	5,6259	5,6470
		5,5336	5,6472	5,7816	5,5776	5,8373	
	Vorname	2,5964	0,8335	1,4191	0,7058	1,3822	1,2479
		0,5619	1,1136	0,7930	3,0650	2,1046	
	Nach- und Vorname	0,0774	0,0011	0,0011	0,0011	0,0013	0,0011
		0,0011	0,0013	0,0011	0,0011	0,0011	
Teil	Nachname	54,0794	52,1891	53,8087	54,7780	59,5402	54,4236
		60,0954	53,4085	54,5618	54,2854	54,6179	
	Vorname	108,2153	104,8426	108,4225	107,7111	108,7665	108,3296
		112,7873	102,7618	111,0297	108,2367	108,9418	
	Nach- und Vorname	5,6775	6,0341	6,1135	5,8613	5,9165	6,0738
		6,3494	6,8494	6,8746	6,2226	5,9963	
Anfang	Nachname	16,9895	16,7580	16,4969	16,6044	16,4092	16,5853
		16,5662	16,5109	16,1200	16,8118	16,7273	
	Vorname	0,7555	0,9916	1,1241	0,9838	1,0932	1,0115
		0,8864	1,0472	1,0825	1,0315	0,9561	
	Nach- und Vorname	0,6931	0,8323	0,7047	0,7555	0,7428	0,7491
		0,8667	0,7758	0,6802	0,7230	0,9238	

MariaDB - Athletensuche - 1.000.000 Datensätze - ohne Index

	Messung	Messergebnisse (*in Sekunden*)					Median
Komplett	Nachname	17,0883	16,9894	16,3877	16,2960	16,2715	16,2839
		16,2718	16,1039	16,2196	16,3403	16,0795	
	Vorname	11,5120	12,1836	12,0428	12,2169	12,2016	12,1132
		11,8504	11,6097	11,9262	12,3299	12,8934	
	Nach- und Vorname	8,3598	8,2295	9,0185	9,1726	9,1422	9,0803
		8,9812	40,1604	83,3115	17,0676	9,0093	
Teil	Nachname	78,5490	77,2979	75,6759	78,9601	83,0351	80,6510
		81,1110	81,6293	80,8401	81,1533	80,4619	
	Vorname	161,3683	161,4915	161,9985	161,5323	163,4704	161,5941
		159,4324	161,4583	162,3415	161,6558	164,2117	
	Nach- und Vorname	10,6771	11,3386	11,0716	11,6137	10,2717	10,3587
		10,3387	9,9796	9,7559	10,3787	9,6073	
Anfang	Nachname	32,7430	32,6939	33,0932	33,1523	32,9682	32,8556
		33,1112	32,9759	32,3805	32,5940	32,5692	
	Vorname	12,1289	12,2632	12,0179	12,8176	12,3968	12,3323
		12,6637	12,6928	13,2009	12,0803	12,2678	
	Nach- und Vorname	6,9356	6,1228	6,0914	6,0977	6,0991	6,9472
		6,9587	7,1716	7,2795	8,1242	8,2905	

MongoDB - Athletensuche - 10.000 Datensätze - mit Index

	Messung	Messergebnisse (*in Sekunden*)					Median
Komplett	Nachname	0,0006	0,0005	0,0005	0,0004	0,0003	0,0003
		0,0003	0,0003	0,0003	0,0004	0,0003	
	Vorname	0,0463	0,0438	0,0445	0,0442	0,0454	0,0443
		0,0441	0,0444	0,0439	0,0431	0,0449	
	Nach- und Vorname	0,0005	0,0003	0,0004	0,0005	0,0005	0,0003
		0,0004	0,0003	0,0003	0,0003	0,0003	
Teil	Nachname	0,0316	0,0303	0,0321	0,0312	0,0305	0,0302
		0,0294	0,0299	0,0290	0,0301	0,0300	
	Vorname	0,0084	0,0081	0,0079	0,0081	0,0078	0,0082
		0,0079	0,0084	0,0092	0,0089	0,0084	
	Nach- und Vorname	0,0388	0,0373	0,0345	0,0358	0,0361	0,0357
		0,0356	0,0348	0,0352	0,0348	0,0359	
Anfang	Nachname	0,0356	0,0374	0,0374	0,0349	0,0338	0,0347
		0,0348	0,0341	0,0343	0,0346	0,0333	
	Vorname	0,0431	0,0424	0,0408	0,0403	0,0401	0,0412
		0,0412	0,0412	0,0415	0,0404	0,0418	
	Nach- und Vorname	0,0342	0,0346	0,0356	0,0350	0,0351	0,0344
		0,0340	0,0338	0,0347	0,0340	0,0342	

MongoDB - Athletensuche - 10.000 Datensätze - ohne Index

	Messung	Messergebnisse (*in Sekunden*)					Median
Komplett	Nachname	0,0004	0,0003	0,0003	0,0003	0,0003	0,0003
		0,0003	0,0003	0,0003	0,0003	0,0003	
	Vorname	0,0264	0,0244	0,0246	0,0256	0,0246	0,0246
		0,0246	0,0243	0,0253	0,0251	0,0246	
	Nach- und Vorname	0,0004	0,0003	0,0003	0,0003	0,0003	0,0003
		0,0003	0,0002	0,0003	0,0002	0,0003	
Teil	Nachname	0,0169	0,0160	0,0167	0,0169	0,0167	0,0166
		0,0169	0,0166	0,0166	0,0165	0,0164	
	Vorname	0,0054	0,0056	0,0053	0,0055	0,0057	0,0055
		0,0054	0,0056	0,0055	0,0052	0,0053	
	Nach- und Vorname	0,0193	0,0186	0,0192	0,0190	0,0187	0,0190
		0,0190	0,0190	0,0187	0,0189	0,0190	
Anfang	Nachname	0,0184	0,0181	0,0180	0,0179	0,0180	0,0182
		0,0182	0,0182	0,0183	0,0183	0,0186	
	Vorname	0,0250	0,0249	0,0253	0,0251	0,0251	0,0251
		0,0251	0,0253	0,0246	0,0244	0,0255	
	Nach- und Vorname	0,0180	0,0190	0,0183	0,0183	0,0185	0,0183
		0,0179	0,0187	0,0189	0,0181	0,0179	

MongoDB - Athletensuche - 100.000 Datensätze - mit Index

	Messung	Messergebnisse (*in Sekunden*)					Median
Komplett	Nachname	0,0006	0,0005	0,0006	0,0005	0,0006	0,0006
		0,0006	0,0008	0,0008	0,0006	0,0007	
	Vorname	0,2315	0,2298	0,2366	0,2316	0,2356	0,2330
		0,2333	0,2328	0,2316	0,2343	0,2360	
	Nach- und Vorname	0,0004	0,0003	0,0003	0,0003	0,0003	0,0003
		0,0004	0,0003	0,0003	0,0003	0,0003	
Teil	Nachname	0,0374	0,0386	0,0385	0,0385	0,0387	0,0381
		0,0372	0,0378	0,0371	0,0387	0,0377	
	Vorname	0,0148	0,0150	0,0160	0,0163	0,0153	0,0154
		0,0148	0,0158	0,0157	0,0156	0,0144	
	Nach- und Vorname	0,1362	0,1339	0,1369	0,1350	0,1346	0,1365
		0,1393	0,1382	0,1375	0,1348	0,1385	
Anfang	Nachname	0,1514	0,1835	0,1825	0,1851	0,1813	0,1814
		0,1815	0,1858	0,1795	0,1487	0,1321	
	Vorname	0,0642	0,0650	0,0663	0,0652	0,0666	0,0653
		0,0660	0,0655	0,0660	0,0651	0,0651	
	Nach- und Vorname	0,1323	0,1364	0,1346	0,1380	0,1372	0,1366
		0,1339	0,1368	0,1370	0,1371	0,1355	

MongoDB - Athletensuche - 100.000 Datensätze - ohne Index

	Messung	Messergebnisse (*in Sekunden*)					Median
Komplett	Nachname	0,0006	0,0006	0,0006	0,0006	0,0007	0,0006
		0,0005	0,0006	0,0005	0,0006	0,0007	
	Vorname	0,2050	0,2019	0,2033	0,2013	0,1991	0,2016
		0,2037	0,1997	0,2047	0,2007	0,1945	
	Nach- und Vorname	0,0004	0,0004	0,0003	0,0003	0,0003	0,0003
		0,0003	0,0003	0,0003	0,0003	0,0003	
Teil	Nachname	0,0278	0,0273	0,0276	0,0270	0,0272	0,0273
		0,0276	0,0272	0,0276	0,0271	0,0270	
	Vorname	0,0124	0,0138	0,0131	0,0136	0,0142	0,0140
		0,0146	0,0140	0,0142	0,0141	0,0141	
	Nach- und Vorname	0,1014	0,1025	0,1029	0,1006	0,1005	0,1007
		0,1006	0,1008	0,0992	0,1004	0,1012	
Anfang	Nachname	0,1487	0,0933	0,0925	0,0940	0,0956	0,0941
		0,0942	0,0931	0,1475	0,0929	0,0950	
	Vorname	0,0561	0,0569	0,0565	0,0572	0,0573	0,0572
		0,0574	0,0566	0,0578	0,0588	0,0576	
	Nach- und Vorname	0,0994	0,0960	0,0963	0,0968	0,0961	0,0961
		0,0957	0,0959	0,0957	0,0964	0,0961	

MongoDB - Athletensuche - 1.000.000 Datensätze - mit Index

	Messung	Messergebnisse (*in Sekunden*)					Median
Komplett	Nachname	0,0007	0,0013	0,0011	0,0012	0,0013	0,0011
		0,0011	0,0011	0,0012	0,0012	0,0010	
	Vorname	0,2580	0,2548	0,2609	0,2605	0,2549	0,2584
		0,2618	0,2650	0,2579	0,2589	0,2564	
	Nach- und Vorname	0,0004	0,0003	0,0003	0,0003	0,0003	0,0003
		0,0003	0,0003	0,0003	0,0003	0,0003	
Teil	Nachname	0,4586	0,3076	0,3103	0,3178	0,3140	0,3060
		0,3045	0,2982	0,3043	0,3001	0,3029	
	Vorname	0,0043	0,0048	0,0045	0,0048	0,0044	0,0045
		0,0043	0,0046	0,0045	0,0045	0,0046	
	Nach- und Vorname	0,3074	0,3084	0,3132	0,3200	0,3112	0,3147
		0,3143	0,3255	0,3236	0,3218	0,3152	
Anfang	Nachname	1,1387	1,0942	1,0837	1,1041	1,0961	1,0983
		1,0879	1,1004	1,1055	1,1006	1,0895	
	Vorname	0,2687	0,2612	0,2653	0,2621	0,2634	0,2665
		0,2728	0,2754	0,2649	0,2678	0,2729	
	Nach- und Vorname	1,1442	1,1547	1,1085	1,1176	1,1451	1,1379
		1,1751	1,1618	1,1282	1,1315	1,1134	

MongoDB - Athletensuche - 1.000.000 Datensätze - ohne Index

	Messung	Messergebnisse (*in Sekunden*)					Median
Komplett	Nachname	0,0010	0,0014	0,0011	0,0011	0,0010	0,0011
		0,0015	0,0010	0,0011	0,0010	0,0013	
	Vorname	0,2600	0,2513	0,2621	0,2527	0,2514	0,2527
		0,2478	0,2547	0,2506	0,2527	0,2554	
	Nach- und Vorname	0,0006	0,0004	0,0003	0,0004	0,0004	0,0004
		0,0004	0,0003	0,0003	0,0003	0,0003	
Teil	Nachname	0,4532	0,3068	0,3088	0,4292	0,2971	0,3028
		0,3008	0,3021	0,2967	0,3036	0,3021	
	Vorname	0,0047	0,0048	0,0042	0,0046	0,0046	0,0046
		0,0044	0,0042	0,0048	0,0042	0,0047	
	Nach- und Vorname	0,3176	0,3135	0,3182	0,3215	0,3165	0,3180
		0,3178	0,3312	0,3160	0,3193	0,3217	
Anfang	Nachname	1,1038	1,1086	1,1165	1,1039	1,1225	1,1034
		1,0942	1,0742	1,1030	1,1023	1,0926	
	Vorname	0,2684	0,2609	0,2610	0,2645	0,2718	0,2644
		0,2637	0,2643	0,2627	0,2694	0,2690	
	Nach- und Vorname	1,1139	1,1203	1,1679	1,1236	1,1008	1,1340
		1,0978	1,1445	1,1521	1,1473	1,1490	

Anzahl Treffer bei Namenssuchen in MariaDB und MongoDB

	Messung	10.000 Athleten	100.000 Athleten	1.000.000 Athleten
Komplett	Nachname	17	212	2.016
	Vorname	12	104	1.004
	Nach- & Vorname	1	1	2
Teil	Nachname	167	1.819	17.848
	Vorname	614	1.994	53.712
	Nach- & Vorname	9	33	929
Anfang	Nachname	55	612	6.022
	Vorname	59	422	1.004
	Nach- & Vorname	2	3	7

MariaDB - Wettkampfabfrage zu einem Athleten

Messung	Messergebnisse (*in Sekunden*)					Median
100 Wettkämpfe	0,0959	0,0016	0,0016	0,0874	0,0016	0,0016
	0,0014	0,0248	0,0014	0,4240	0,0015	
1.000 Wettkämpfe	0,3501	0,0143	0,0214	0,0180	0,0177	0,0177
	0,0177	0,0192	0,0167	0,0172	0,0051	
10.000 Wettkämpfe	5,7319	5,5808	5,6391	5,7144	5,6217	5,6491
	5,6199	5,6385	5,6645	5,6776	5,6592	

MongoDB - Wettkampfanfrage zu einem Athleten

	Messung	Messergebnisse (*in Sekunden*)					Median
mit Index	100 Wettkämpfe	0,0011	0,0010	0,0010	0,0010	0,0010	0,0010
		0,0010	0,0010	0,0010	0,0010	0,0012	
	1.000 Wettkämpfe	0,0538	0,0446	0,0447	0,0460	0,0447	0,0450
		0,0450	0,0453	0,0465	0,0449	0,0449	
	10.000 Wettkämpfe	0,4320	0,3674	0,3748	0,3669	0,3696	0,3684
		0,3688	0,3686	0,3682	0,3676	0,3650	
ohne Index	100 Wettkämpfe	0,0009	0,0008	0,0008	0,0009	0,0008	0,0009
		0,0010	0,0010	0,0008	0,0008	0,0009	
	1.000 Wettkämpfe	0,0365	0,0375	0,0370	0,0373	0,0370	0,0370
		0,0370	0,0367	0,0371	0,0364	0,0370	
	10.000 Wettkämpfe	0,0691	0,0414	0,0406	0,0400	0,0397	0,0406
		0,0406	0,0403	0,0406	0,0406	0,0406	

MariaDB - Leistungsabfrage zu einem Athleten

Messung	Messergebnisse (*in Sekunden*)					Median
100 Wettkämpfe	0,0854	0,0019	0,0019	0,0019	0,0111	0,0019
	0,0020	0,0020	0,0018	0,0520	0,0018	
1.000 Wettkämpfe	0,0030	0,0113	0,0115	0,0113	0,0113	0,0113
	0,0114	0,0113	0,0130	0,0115	0,0113	
10.000 Wettkämpfe	5,7548	5,8026	5,7092	5,6903	5,6865	5,6921
	5,7387	5,6899	5,6661	5,6939	5,6823	

MongoDB - Leistungsabfrage zu einem Athleten

Messung			Messergebnisse (*in Sekunden*)					Median
100 Wettkämpfe	MAP-REDUCE		0,2763	0,2913	0,2895	0,2992	0,4543	0,2904
			0,2750	0,2968	0,3422	0,2663	0,2690	
	Aggregation-Pipeline		0,0051	0,0041	0,0043	0,0045	0,0042	0,0044
			0,0040	0,0046	0,0046	0,0043	0,0045	
	(optimiert)		0,0080	0,0006	0,0007	0,0005	0,0005	0,0005
			0,0005	0,0005	0,0005	0,0005	0,0005	
1.000 Wettkämpfe	MAP-REDUCE		0,3541	0,4174	0,3353	0,3228	0,4753	0,3433
			0,3514	0,3233	0,4691	0,3352	0,3322	
	Aggregation-Pipeline		0,3618	0,3662	0,3436	0,3466	0,3452	0,3596
			0,3608	0,3619	0,3597	0,3596	0,3585	
	(optimiert)		0,0346	0,0338	0,0344	0,0334	0,0336	0,0336
			0,0336	0,0330	0,0334	0,0336	0,0338	
10.000 Wettkämpfe	MAP-REDUCE		3,7544	3,7445	3,7585	3,8081	3,8136	3,7580
			3,8050	3,7744	3,7268	3,7506	3,7574	
	Aggregation-Pipeline		34,5371	34,1585	34,4457	34,3062	34,0721	34,1537
			34,1074	34,0502	34,3158	34,1489	34,1135	
	(optimiert)		3,4797	3,3692	3,3398	3,3628	3,4213	3,4221
			3,4266	3,4299	3,4242	3,4229	3,3925	

Aufteilung der Ausarbeitung

Kapitel / Abschnitt	Verfasser
1 Einleitung	Matthias Frank
2 Die Eingrenzung nichtrelationaler Systeme	Matthias Frank & Daniel Morady
2.1 Eine Auswahl im Vorfeld	Daniel Morady
2.2 Die betrachteten Datenbankgruppen	Daniel Morady
2.3 Eine Auflistung der Anforderungen des Projekts	Matthias Frank
2.4 Eine erste Eignungseinschätzung	Matthias Frank
2.5 Fazit	Matthias Frank & Daniel Morady
3 Eine weitergehende Betrachtung der gewählten Systeme	Matthias Frank & Daniel Morady
3.1 Die Eingrenzung der notwendigen Eigenschaften	Matthias Frank & Daniel Morady
3.1.1 Aggregate	Daniel Morady
3.1.2 Schemafreiheit	Matthias Frank
3.1.3 Datenkonsistenz (Consistency)	Daniel Morady
3.1.4 Datenaufteilung (Sharding)	Matthias Frank
3.1.5 Datenspiegelung (Replication)	Matthias Frank
3.1.6 Zusammenfassung	Matthias Frank
3.2 Die Begutachtung der verbleibenden Datenbanksysteme	Daniel Morady
3.3 Fazit	Matthias Frank & Daniel Morady
4 Die Verwendung des ausgewählten Systems	Matthias Frank & Daniel Morady
4.1 Die Bestimmung der Softwarelösung	Matthias Frank
4.2 Die nichtrelationale Umsetzung	Daniel Morady
5 Der Vergleich mit der relationalen Umsetzung	Matthias Frank & Daniel Morady
6 Fazit	Matthias Frank & Daniel Morady
Umsetzung der Datenbankanbindung an die Weboberfläche	Daniel Morady
Ausarbeitung und Ausführung der Messreihen	Matthias Frank

Eidesstattliche Erklärung

Wir versichern, die von uns vorgelegte Arbeit selbständig verfasst zu haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, haben wir als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die wir für die Arbeit benutzt haben, sind angegeben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Gummersbach, 15. Februar 2017

Matthias Frank

Daniel Morady