HAW HAMBURG

BACHELORTHESIS
Yasmin Kohi

Ein experimenteller Vergleich und eine Evaluation von MariaDB, MongoDB und CockroachDB

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Yasmin Kohi

Ein experimenteller Vergleich und eine Evaluation von MariaDB, MongoDB und CockroachDB

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr. Martin Hübner

Eingereicht am: 23. September 2022

Yasmin Kohi

Thema der Arbeit

Ein experimenteller Vergleich und eine Evaluation von MariaDB, MongoDB und CockroachDB

Stichworte

NoSQL, NewSQL, SQL, RDBMS, DBMS, ACID, CockroachDB, MariaDB, MongoDB, CAP-Theorem, BASE, Big Data

Kurzzusammenfassung

Big Data ist in der jetzigen Zeit ein bekanntes Problem geworden. Aufgrund der enormen Datenmenge haben traditionelle Datenbanken Schwierigkeiten, die Daten zu verarbeiten. NoSQL und NewSQL sind Vorreiter für die Verarbeitung von Big Data. In dieser Bachelorarbeit wird ein experimenteller Vergleich und eine Evaluation zwischen CockroachDB, einer NewSQL-Datenbank, MariaDB, einer SQL-Datenbank, und MongoDB, einer NoSQL-Datenbank, durchgeführt. Die Tests prüfen die Systeme nach definierten Kriterien: die Leistung, den Index, die Transaktion, den Trigger, die CRUD-Operationen und die Verarbeitung großer Datenmengen.

Yasmin Kohi

Title of Thesis

An experimental comparison and evaluation of MariaDB, MongoDB and CockroachDB

Keywords

NoSQL, NewSQL, SQL, RDBMS, DBMS, ACID, CockroachDB, MariaDB, MongoDB, CAP-Theorem, BASE, Big Data

Abstract

Big data has become a known issue of the time. Due to the enormous amount of data, traditional databases have difficulty processing it. NoSQL and NewSQL are pioneers in processing big

data. This bachelor thesis conducts an experimental comparison and evaluation between CockroachDB, a NewSQL database; MariaDB, a SQL database; and MongoDB, a NoSQL database. The tests examined these systems according to defined criteria: the performance, the indices, the transactions, the triggers, the CRUD-Operations, and the processing of large amounts of data.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

ACID	Atomicity, Consistency, Isolation and Durability
BASE	Basically Available, Soft State and Eventually Consistent
BSON	Binary JSON
BTree	Balanced Tree
CAP Theorem	Consistency, Availability and Partition Tolerance Theorem
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
DBMS	Datenbankmanagementsystem
DML	Data Manipulation Language
JSON	JavaScript Object Notation
LSM-Tree	Log-Structured Merge-Tree
MQL	MongoDB Query Language
MVCC	Multiversion Concurrency Control
NewSQL	New Structured Query Language
NoSQL	Not Only SQL
NTP	Network Time Protocol
OLAP	Online Analytical Processing
OLTP	Online Transactional Processing

RTree	Real Tree
RDBMS	Relationales Datenbankmanagementsystem
SQL	Structured Query Language
WAN	Wide Area Network

Glossar

Transaktion Eine Transaktion besteht aus einer Logikeinheit, die wiederum aus mehreren Operationen besteht.

Sharding Sharding ist das Aufteilen von Daten in kleine Einheiten, die über das System verteilt und auf verschiedenen Systemen gespeichert werden.

Clustering Clustering ist der Prozess einer Kombination von mehr als einem Server, die eine Verbindung zu einer einzelnen Datenbank erstellt.

Caching Beim Caching werden die Daten in einem Cache, der eine Hochgeschwindigkeitsspeicherebene ist, temporär gespeichert, um für einen schnelleren Zugriff auf die Daten im Vergleich zum primären Speicherort zu sorgen.

Skalierung Skalierung beschreibt eine Eigenschaft eines Datenbanksystems, die linear wächst, z. B. das Steigen der Benutzeranzahl oder Datenmengen in das System.

Normalisierung Normalisierung wird in der relationalen Datenbank eingesetzt, um redundante Daten im Datenbanksystem zu vermeiden. Dabei

werden auf die Tabellen Normalisierungsregeln und die Normalformen angewendet.

Latenz Unter Latenz wird die Verzögerungszeit verstanden.

Replikation Unter Replikation wird das Kopieren der Daten von einem System auf ein anderes System bezeichnet.

Partitionierung Unter Partitionierung wird der Prozess des Aufteilens sehr großer Tabellen in mehrere kleine bezeichnet.

Open Source Eine Software wird als Open Source definiert, deren Quelltext öffentlich ist und von anderen verwendet werden kann.

Standalone Unter Standalone wird das Verwenden eines Servers verstanden.

Entität Eine Entität ist ein Objekt, das durch Angaben von Attributen beschrieben wird.

Schema Ein Schema definiert die Organisation der Daten in einer relationalen Datenbank.

1 Einleitung

1.1 Motivation

Datenbanken wurden für die Analyse, Verwaltung und Speicherung von Daten entwickelt, wovon zahlreiche Unternehmen profitieren [1, S. 1]. Durch die Zunahme von Nutzenden im Internet und das Herausbringen neuer Technologien nehmen die Datenmengen jährlich exponentiell zu, was die Verwaltung in den relationalen Datenbanken erschwert. Demnach erweitern Entwickelnde stetig die vorhandenen Datenbanken und entwickeln neue Systeme, um das Problem zu lösen. Ein neues Datenbankkonzept ist das NoSQL, das für die Skalierung und Handhabung großer Datenmengen geeignet ist. Da NoSQL-Systemen Transaktionen und andere Eigenschaften der SQL-Datenbanken fehlen, wurde das NewSQL-System entworfen, um große Datenmengen mithilfe der Eigenschaften einer relationalen Datenbank zu verarbeiten [2, S. 1].

1.2 Ziele der Bachelorarbeit

In der Forschung fehlt ein Vergleich zwischen den folgenden Datenbanksystemen: MongoDB, eine NoSQL-Datenbank, MariaDB, eine SQL-Datenbank, und CockroachDB, eine NewSQL-Datenbank. Deshalb ist das Ziel dieser Bachelorarbeit, einen Einblick in die einzelnen Datenbanksysteme und -modelle zu geben. Dafür werden verschiedene Testszenarien auf die Systeme angewendet, um die Stärken und Schwächen sowie Unterschiede und Ähnlichkeiten der Systeme zu untersuchen. Diese Systeme werden auf Trigger, Performance, Index und Transaktionseigenschaften von ACID (Atomicity, Consistency, Isolation, Durability) geprüft und miteinander verglichen, um die geeignetsten Anwendungsfälle für sie zu finden.

1.3 Aufbau der Bachelorarbeit

Die Bachelorarbeit ist in fünf Kapitel aufgeteilt. In Kapitel 2 werden die Eigenschaften und Unterschiede zwischen SQL, NoSQL und NewSQL beschrieben. In Kapitel 3 werden die Datenbanken MariaDB, MongoDB und CockroachDB, die als Repräsentanten für SQL, NoSQL und NewSQL gewählt wurden, vorgestellt. In diesem Kapitel werden die Funktionalitäten, Anwendungsgebiete und Architekturen der Systeme beschrieben. Daraufhin wird in Kapitel 4 das Experiment dargestellt. Darin werden die drei Systeme CockroachDB, MongoDB und MariaDB mittels bestimmter Anwendungsfälle getestet und miteinander verglichen. In Kapitel 5 wird die Bachelorarbeit schließlich zusammengefasst.

2 Hintergrund von Big Data und der Datenbankmodelle

Bereits in der Antike wurden Informationen aufgezeichnet und archiviert, jedoch war das Beschaffen und Ändern von Informationen ein beschwerlicher Vorgang. Um den Zugriff zu vereinfachen und zu beschleunigen, wurden mit dem Fortschritt der Technologien die handschriftlichen Daten in digitale Datenbankeinträge überführt [3]. Das Modell einer Datenbank wird durch eine Software namens Datenbankmanagementsystem (DBMS) festgelegt. Das DBMS ist für den Aufbau, die Verwaltung und den Zugriff von Datenbanken zuständig. Das erste DBMS wurde von Charles Bachman im Jahr 1960 entwickelt [4]. Im Jahr 1970 wurde ein neues DBMS-Modell entwickelt, das relationale Datenbankmanagementsystem (RDBMS) [1, S. 1]. Durch den wachsenden Konsum der Weltbevölkerung in Form der Nutzung Sozialer Medien, Geo-Tracking, Internet etc. werden mehr Daten denn je produziert, wodurch eine neue Form von Daten entstanden ist. Dies sind sogenannte Big Data. Das RDBMS besitzt nicht die nötigen Funktionen wie Sharding, Clustering und Caching, um Big Data horizontal skalieren zu können [2, S. 1]. Zusätzlich ist die Architektur von RDBMS nicht geeignet für eine große Datenmenge. Deshalb nimmt die Leistung und Effizienz von Datenabfragen bei Big Data ab. Zudem können die Abfragen aufgrund des festgelegten, relationalen Schemas lang und komplex werden [5, S. 1]. Das Speichern und die Verwaltung von Big Data stellt für das RDBMS eine Herausforderung dar [1, S. 1]. Um einige Einschränkungen von RDBMS sowie die Probleme von Big Data lösen zu können, wurde ein neues DBMS-Konzept eingeführt, das Not Only SQL (NoSQL) [2, S. 1]. NoSQL dient zum Speichern und Verwalten außergewöhnlich vieler Daten, da im Vergleich zu einer relationalen Datenbank keine festgelegte Datenstruktur für die Daten benötigt wird. Dadurch können die Daten leicht gespeichert und schnell aus der Datenbank abgerufen werden [1, S. 1]. Zudem besitzt NoSQL Funktionen wie eine geringe Latenz, die automatische Replikation und Sharding von Daten. Während NoSQL die Skalierungs- und

Flexibilitätsprobleme einer traditionellen RDBMS löst, bringt sie auch neue Probleme mit sich, z. B. die Inkonsistenz der Daten, keine ACID, keine Unterstützung für Transaktionen und Indexe, einen fehlenden Standardzugriff auf die Daten mithilfe der SQL-Sprache. Da NoSQL die Einschränkungen von RDBMS nicht behebt, wurde eine weitere Art von RDBMS entwickelt. Diese andere Art moderner RDBMS wird als NewSQL bezeichnet. NewSQL bietet sämtliche Funktionalitäten eines RDBMS mit Skalierungen und Leistungen eines NoSQL-Systems an und löst mit ihrer Architektur die Big-Data-Probleme [1, S. 1], [2, S. 1], [6, S. 3].

2.1 Big Data

Big Data wird als eine riesige, komplexe Datenmenge bezeichnet, die aus unterschiedlichen, heterogenen Datenquellen besteht. Heutzutage sind die meisten Technologien online und der Nutzende ist von unterschiedlichen Anwendungen und Geräten, z. B. Facebook, Twitter und YouTube, umgeben, die täglich unzähligen Mengen an Daten produzieren. Zum Beispiel erzeugt Google 1,2 Billionen Suchanfragen täglich und 40 000 Suchanfragen pro Sekunde [7, S. 1]. Dieses Phänomen wurde im Jahr 2000 von Francis Diebold in seiner wissenschaftlichen Arbeit „‘Big Data‘ Dynamic Factor Models for Macroeconomic Measurment and Forecasting“ mit dem Begriff ‚Big Data‘ beschrieben, wobei ‚Big Data‘ als eine explosionsartige Zunahme von quantitativen, verfügbaren, aktuellen, relevanten Daten definiert wurde [8]. Big Data entstehen aus unterschiedlichen, heterogenen Datenquellen als strukturierte, unstrukturierte und semistrukturierte Datentypen.

Strukturierte Daten:

Die strukturierten Daten werden in festgelegten Formaten definiert und in dieser Struktur gespeichert, verarbeitet und zugegriffen.

Unstrukturierte Daten:

Die unstrukturierten Daten besitzen keine Struktur und können nicht in RDBMS gespeichert werden. Für Analysezwecke können die unstrukturierten Daten so lange nicht genutzt werden, bis sie in strukturierte Daten umgewandelt werden.

Semistrukturierte Daten:

Semistrukturierte Daten besitzen keine formale Struktur, aber haben einige Merkmale, mit deren Hilfe die Daten analysiert und strukturiert werden können [9].

Die größten Herausforderungen bei der Verarbeitung von Big Data können nach Bali, Chaudhary und Singla [7, S. 65–66] in vier Merkmale aufgeteilt werden: Volume, Variety, Veracity und Velocity. Diese vier Merkmale von Big Data fordern viele konventionelle Datenbanken heraus und erschweren die Verarbeitung dieser Daten.

Volume (Volumen):

Volume steht für das Speichern vieler Daten. Diese Daten wachsen durch den Verbrauchenden kontinuierlich von Terabytes bis zu Hunderten von Petabytes, wodurch für das Verarbeiten dieser Daten Skalierungsmechanismen benötigt werden [10, S. 107].

Velocity (Geschwindigkeit):

Unter Velocity wird die hohe Geschwindigkeit der Datenerzeugung aus heterogenen Quellen und die Verarbeitung von Big Data bezeichnet.

Variety (Vielfalt):

Variety bezieht sich auf die unterschiedlichen Typen von Big Data, die aufgrund heterogener Quellen entstehen. Big Data besteht aus strukturierten, unstrukturierten und semistrukturierten Datentypen [11].

Veracity (Wahrhaftigkeit):

Veracity bezieht sich auf die Qualität und Wahrhaftigkeit der Daten. Die zur Verfügung gestellten Daten sollten für die Analysen und Entscheidungsprozesse präzise, relevant und aktuell sein [10, S. 107][12].

2.2 SQL

Das RDBMS wurde im Jahr 1970 von Dr. Edger F. Codd durch das IBM-Forschungslabor eingeführt [13, S. 4] und beruht auf dem relationalen Modell, in dem sämtliche Daten als Zeilen (Tupeln) in einer Tabelle abgelegt werden. Die Tabelle bildet eine Relation und besteht aus Spalten, wobei jede Spalte mit einem Attribut (Eigenschaft) assoziiert wird. Eine Zeile innerhalb einer Tabelle enthält einen Primärschlüssel, um diese Zeile eindeutig identifizieren zu können [13, S. 4,6], [14, S. 3]. RDBMS nutzen SQL (‚Structured Query Language', auf Deutsch ‚Strukturierte Abfragesprache') als Programmiersprache [13, S. 8]. Mithilfe von SQL-Anweisungen kann der Nutzende auf die Tabellen in einer relationalen Datenbank zugreifen, die Daten manipulieren, speichern und löschen. Vor dem Speichern der Daten in den Tabellen werden die Datentypen der Attribute und die Beziehung zwischen den Daten mittels SQL festgelegt [14, S. 3–4]. Mit dem festgelegten Schema kann die relationale Datenbank strukturierte Daten verarbeiten. Da semistrukturierte und unstrukturierte Daten keine formalen Strukturen besitzen, kann die relationale Datenbank diese Daten ohne eine Säuberung und Transformation nicht in Tabellen einführen [9]. RDBMS unterstützen die Normalisierung, um Duplikate zu beseitigen [15, S. 41], und die ACID-Eigenschaften, die beschreiben, dass eine Transaktion atomar, konsistent, isoliert und dauerhaft ist.

Atomicity (Atomizität):

Atomizität garantiert die Vollständigkeit einer Transaktion.

Consistency (Konsistenz):

Konsistenz sorgt für Stabilität der Daten.

Isolation (Isolation):

Isolation gewährleistet mehrere parallel ausgeführte Transaktionen, die unabhängig voneinander laufen können.

Durability (Dauerhaftigkeit):

Dauerhaftigkeit sorgt dafür, dass die Daten nach einer Transaktion für immer gespeichert werden [6, S. 1].

Die Architektur eines RDBMS ist ein zentrales System mit einer vertikalen Skalierung (Scale Up) [2, S. 1]. Durch die vertikale Skalierung werden die Ressourcen, z. B. CPU (Central Processing Unit) oder Speicher, und damit die Kapazität und Leistung des zentralen Systems verbessert [6, S. 2].

2.3 NoSQL

NoSQL ist ein nichtrelationales Datenbankmanagementsystem und wurde im Jahr 2009 als die nächste Generation von Datenbanken eingeführt [2, S. 1]. NoSQL wurde entworfen, um in einer verteilten Datenbank die Daten horizontal zu skalieren (Scale Out). Bei der horizontalen Skalierung werden neue Server zum bestehenden Cluster hinzugefügt, um die Last zu verteilen [2, S. 2], [6, S. 2]. NoSQL kann aufgrund der horizontalen Skalierung große Mengen von Daten auf den Servern verteilen und speichern. Dabei werden Kopien von Daten auf den neuen Servern zur Verfügung gestellt [1, S. 1–2]. Damit schafft NoSQL eine hohe Performance, Replikation und Verfügbarkeit mit geringer Latenz [2, S. 1]. Im Vergleich zu RDBMS senkt NoSQL mit der horizontalen Skalierung die Kosten von Unternehmen durch das Hinzufügen kostengünstiger Server [1, S. 1–2].

2.3.1 BASE

Im Vergleich zum RDBMS bietet NoSQL keine ACID-Transaktionen. Stattdessen basiert NoSQL auf dem Konzept von BASE (Basically Available, Soft State und Eventually Consistent). BASE stützt sich auf das CAP-(Consistency, Availability and Partition Tolerance-)Theorem, von dem nur zwei Eigenschaften ausgewählt werden können.

Basically Available:

Basically Available steht für die hohe Verfügbarkeit; sollten mehrere Server ausfallen, so ist immer mindestens ein Server weiterhin erreichbar.

Soft State:

Soft State steht für die Datenänderungen, die sich mit oder ohne eine Eingabe verändern.

Eventually Consistent:

Eventually Consistent lässt die Daten irgendwann in der Zukunft konsistent werden [1, S. 3], [6, S. 1].

BASE ist flexibler als ACID, weil die Sicherstellung konsistenter Daten über mehrere Server verteilt schwierig einzuhalten ist [1, S. 3]. NoSQL ist im Vergleich zu RDBMS schemafrei; damit können jegliche Typen von Big Data gespeichert werden. Aufgrund der Architektur können Informationen unabhängig von ihrem Inhalt und ihrer Struktur aus den NoSQL-Datenbanken schnell abgefragt werden [2, S. 1–3], [6, S. 3].

2.3.2 Kategorien von NoSQL-DBMS

NoSQL kann in die folgenden vier Kategorien eingeteilt werden:

Key-Value Database (Schlüssel-Werte-Datenbank):

Schlüssel-Werte-Datenbanken bestehen aus Schlüsseln und Werten, die in einer ‚Hash-Tabelle' geführt werden. Der Schlüssel wird als Index genutzt, der auf einen Datensatz verweist. Jeder Schlüssel ist einzigartig, womit jeder Datensatz eindeutig identifiziert werden kann. Die Schlüssel-Werte-Datenbanken eignen sich für die Verwaltung von Beständen und Produkten sowie Echtzeitanalysen, die Wert auf eine hohe Datenabrufgeschwindigkeit legen. Amazon DynamoDB und Redis sind Schlüssel-Werte-Datenbanken [1, S. 4], [16, S. 1].

Document Oriented Database (Dokumentenorientierte Datenbank):

Die dokumentenorientierte Datenbank nutzt Dokumente in JSON- und BSON-Formaten als Werte in der Datenbank. Dabei sind die Dokumente in der Datenbank strukturiert, unstrukturiert oder semistrukturiert. Die dokumentenorientierte Datenbank hat eine eigene Sprache, die SQL ähnelt, um Abfragen auf die Dokumente zu ermöglichen. Sie eignet sich für die Verarbeitung und Speicherung großer Mengen von Dokumenten. Hierzu gehört u. a. MongoDB, die sich für Textdokumente oder E-Mails eignet [1, S. 4], [16, S. 1–2].

Column-Family Database (Spaltenfamilien-Datenbank):

Die Spaltenfamilien-Datenbank wurde im Jahr 2006 von Googles ‚Big Table' eingeführt. Sie besteht aus einer Menge von Spalten, die ähnliche Daten speichern. Neue Spalten und Daten können jederzeit in die Spaltenfamilie eingefügt werden [16, S. 2]. Die Spaltenfamilien-Datenbank wird beispielsweise für Logging verwendet [1, S. 4]. Cassandra ist eine Spaltenfamilien-Datenbank [16, S. 2].

Graph Database (Graphdatenbank):

Die Graphdatenbanken basieren auf Knoten und Kanten. Die Knoten repräsentieren die Entitäten und die Kanten stellen die Beziehungen der Knoten zueinander dar [16, S. 2]. Die Graphdatenbank wird für die Darstellung sozialer Verbindungen, z. B. bei Transportsystemen oder Straßenkarten, verwendet [1, S. 4]. Neo4J ist eine Graphdatenbank [16, S. 2].

2.4 NewSQL

NewSQL ist eine Erweiterung der traditionellen RDBMS und wurde im Jahr 2011 von Matthew Aslett eingeführt [17, S. 1]. Verglichen zu NoSQL zielen die NewSQL-Datenbanken darauf, die Eigenschaften einer relationalen Datenbank, z. B. das relationale Datenmodell, beizubehalten und gleichzeitig die Leistung, Skalierung und Verteilung eines NoSQL-Systems anzubieten. Für die Interaktion mit den Anwendungen unterstützt NewSQL die Sprache SQL und ist aufgrund der Architektur für große Mengen von Online Transactional Processing-(OLTP-)Workloads geeignet. Die Architektur von NewSQL-Datenbanken beruht auf der Shared-Nothing-Scale-Out-Architektur, die die horizontale Skalierung unterstützt [2, S. 1,5]. Die Tabellen werden mithilfe von Sharding oder Partitionierung horizontal in mehrere Fragmente aufgeteilt. NewSQL unterstützt mittels verschiedener Timestamp-Ordering-Strategien die Concurrency Control. Durch die Concurrency Control können mehrere Nutzer auf die Datenbank zugreifen und unter Wahrung der ACID-Eigenschaften parallele Transaktionen ausführen. Um eine hohe Verfügbarkeit und Lebensdauer der Daten zu sichern, bietet NewSQL eine strenge konsistente, Wide-Area-Network (WAN) Replikation durch die synchrone und asynchrone Replikationsmethode an. Zusätzlich unterstützen die NewSQL-

Datenbanken Fehlertoleranz- und Wiederherstellungsmechanismen. Im Falle eines Master-Server-Ausfalls wird ein anderer erreichbarer Server als Master ausgewählt, der die Transaktionen weiterverarbeitet. Die meisten NewSQL-Datenbanken arbeiten mit einem In-Memory, indem die Daten im Hauptspeicher gespeichert werden. Dadurch muss die Datenbank nicht mit dem Schieben der Daten zwischen einer Festplatte und dem Hauptspeicher beschäftigt sein. Mithilfe von Monitoring-Mechanismen werden die Daten, die nicht häufig benutzt werden, aus dem Hauptspeicher auf eine Festplatte verlegt. Durch das In-Memory wird ein schnelles Ausführen von Abfragen ermöglicht und die hohen Kosten sowie der Arbeitsspeicher werden reduziert [18, S. 2–3]. Der Non-Locking-Concurrency-Control-Mechanismus vermeidet in NewSQL den Konflikt zwischen Lese- und Schreiboperationen in Echtzeit [2, S. 5].

2.5 Vergleich zwischen SQL, NoSQL und NewSQL

In Tabelle 2.1 ist ein allgemeiner Vergleich zwischen SQL, NoSQL und NewSQL aufgelistet. Die Vergleichskriterien orientieren sich an den Funktionalitäten und Eigenschaften der verschiedene Datenbankmodelle.

Tabelle 2. 1: Ein Vergleich von Eigenschaften zwischen SQL, NoSQL und NewSQL[1]

Datenbankmodell	**SQL**	**NoSQL**	**NewSQL**
Schema	Festes, relationales Schema	Schemafrei	Beides
Datentypen	Strukturierte Daten	Strukturierte, unstrukturierte, semistrukturierte Daten	Strukturierte, unstrukturierte, semistrukturierte Daten

[1] [63], [64]

Speicherung der Daten	Tabellen	Dokumenten, Schlüssel-Werte-Paare, Graphen, Spalten	Tabellen
Skalierung	Vertikal	Horizontal	Horizontal
Normalisierung	Ja	Nein[2]	Ja
Verteilung	Nein	Ja	Ja
Verfügbarkeit	Leidet an Single Point of Failure	Hohe Verfügbarkeit	Hohe Verfügbarkeit
Sicherheit	Strenge Sicherheitsmechanismen	Sicherheit ist nicht Teil von der Datenbank	Strenge Sicherheitsmechanismen
Transaktion	ACID	BASE	ACID
Anfragesprache	SQL	Abhängig vom System	SQL
OLTP	Teilweise	Nein	Ja
Abfragen	Abfragen mit geringer Komplexität	Abfragen mit hoher Komplexität	Beides
JOIN	Ja	Abhängig vom System	Ja
Datenmenge	Klein bis mittlere Datenmenge	Big Data	Big Data

[2] [35]

Kosten	Teuer	Günstig[3]	Günstig
Vorkommen	Open Source und Closed Platforms[4]	Open Source und Closed Platforms	Open Source
Beispiele	MariaDB, PostgresSQL, Oracle, MariaDB	MongoDB, Redis, Neo4J, Voldemort, Apache Cassandra	VoltDB, Apache Ignite, Google Spanner, CockroachDB

2.6 Existierende Forschungen

In den vergangenen Jahren wurden viele Vergleiche und Evaluationen von DBMS durchgeführt. Die meisten Forschungen verglichen dabei SQL- und NoSQL-DBMS. Wenige Evaluationen betrachteten dabei SQL-, NoSQL- und NewSQL-Systeme.

Die wissenschaftliche Arbeit „ANALYSIS AND COMPARISON OF DOCUMENT-BASED DATABASES WITH SQL RELATIONAL DATABASES: MONGODB VS MYSQL“ [14] führte eine Analyse und einen Vergleich zwischen MongoDB, einer dokumentenorientierten Datenbank, und MySQL, einer relationalen Datenbank, durch. Dabei bewerteten die Autoren die Datenspeicherung und Datenverwaltung der Systeme. Durch verschiedene Szenarien bewerteten sie zusätzlich die CRUD(Create, Read, Update, Delete)-Operationen und nannten die Vor- und Nachteile der Datenbankmodelle. Im Gesamtvergleich schnitt MongoDB bei großen Datenmengen mit mehrfachen parallelen Zugriffen besser ab als MySQL.

Die Autoren von „MongoDB vs Oracle – Database comparison“ [19] verglichen das SQL-DBMS Oracle mit MongoDB. Die Vergleichskriterien umfassten den theoretischen Aspekt der Systeme wie die Funktionalitäten, die Einschränkungen, die Integrität, die Verteilung, die

[3] [1, S. 2]

[4] [6, S. 2]

Systemanforderungen und die Geschwindigkeit des Abfragens und Einfügens der Daten. Die Ergebnisse zeigten, dass MongoDB wesentlich schneller und flexibler als die Oracle-Datenbank war. MongoDB unterstützt die horizontale Skalierung, weshalb das Kopieren eines Servers zum anderen Server mithilfe von Tools einfach ist. Aufgrund des festgelegten Schemas war die Oracle-Datenbank wesentlich langsamer als MongoDB. Die Replikation von Daten ist im Vergleich zu MongoDB schwieriger, jedoch ermöglicht die Oracle-Datenbank komplexe Abfragen mittels der ‚Join'-Operation.

In der wissenschaftlichen Arbeit „Inner Join Query Performance: MariaDB vs PostgreSQL" [20] wurden 50 000 bis 1 050 000 Datensätze in MariaDB und das PostgreSQL-DBMS geladen und 21-mal auf ein, zwei und drei ‚Join'-Relationen geprüft und bewertet. Basierend auf den Ergebnissen des T-Tests erzielte PostgreSQL eine schnellere Antwortzeit und eine bessere Leistung als MariaDB.

Kaur und Sachdeva verglichen in der Dissertation „Performance Evaluation of NewSQL Databases" [21] die folgenden vier NewSQL-Systeme miteinander: NuoDB, VoltDB, MemSQL und CockroachDB. Dabei wurde der Fokus auf die Vorteile, Eigenschaften und Klassifizierungen von NewSQL-Datenbanken für die OLTP im Big-Data-Management gelegt. Im Experiment wurden die Geschwindigkeit von Lese-, Schreib- und Update-Operationen sowie die Ausführung der Operationen miteinander verglichen. Das Resultat des Experiments ergab, dass NuoDB bei sämtlichen Operationen bis auf die Update-Operation weniger Zeit benötigte. Bei der Update-Operation benötigte MemSQL weniger Zeit. Im Gesamtvergleich lieferte NuoDB in den meisten Testfällen eine bessere Leistung.

3 Analyse

3.1 Repräsentanten der Datenbanken

In diesem Abschnitt werden drei verschiedene Datenbanken als Repräsentanten vorgestellt. Dabei werden die Leistung, die Funktionalität sowie die Architektur genannt und erläutert, um Gemeinsamkeiten und Unterschiede der Systeme darstellen zu können. Unter den NoSQL-Datenbanken haben Amazon DynamoDB und MongoDB im Laufe der Jahre an Popularität gewonnen [22]. MongoDB ist eine dokumentenorientierte NoSQL-Datenbank [16, S. 1]. Da dokumentenorientierte Datenbanken für die Herausforderungen von neuen und sich schnell ändernden Datentypen geeignet sind, wird MongoDB aufgrund ihrer Architektur und Funktionalitäten in den meisten empirischen Forschungen und Vergleichen als NoSQL-System genommen [14, S. 1–2]. Amazon DynamoDB ist eine Schlüssel-Werte-Datenbank. Im Vergleich zu MongoDB, die Lese- und Schreiboperationen in denselben Dokumenten und Feldern in einer einzigen Transkation erlaubt, unterstützt Amazon DynamoDB nicht mehrere Operationen innerhalb einer einzelnen Transaktion. Zudem ist DynamoDB auf die Amazon Web Services beschränkt [23]. Aus diesem Grund wird MongoDB als Repräsentantin für eine NoSQL-Datenbank gewählt. MariaDB ist Repräsentantin für ein RDBMS. MariaDB gehört zu den besten Optionen einer als Open Source verfügbaren relationalen Datenbank und wird von vielen bekannten Unternehmen, z. B. Facebook und Google, genutzt [24]. Neben dem Open-Source-Aspekt bietet MariaDB die Funktionalitäten einer relationalen Datenbank und unterscheidet sich mit ihrer besonderen Architektur von anderen RDBMS. Ursprünglich wurde MariaDB als ‚Fork' von MySQL bezeichnet, weshalb sich die Datenbanken ähneln. Allerdings bietet MariaDB mehr Storage Engines an und liefert eine bessere Performance ab als MySQL, weshalb MariaDB als Repräsentantin für den Vergleich gewählt wird [25]. NuoDB, ClustrixDB und CockroachDB sind populäre NewSQL-Datenbanken [26]. Da CockroachDB gegenüber den anderen NewSQL-Systemen noch jung ist, gibt es kaum Vergleiche zwischen CockroachDB

und den anderen Datenbanksystemen. Aus diesem Grund wird CockroachDB als Repräsentantin für NewSQL-Datenbanken gewählt.

3.1.1 MariaDB

MariaDB ist ein als Open Source verfügbares RDBMS. Zurzeit ist MariaDB eine der weitverbreitetsten RDBMS, die Daten aus heterogenen Quellen in strukturierte Informationen umwandelt und in Tabellen ablegt. MariaDB unterschützt die ACID-Eigenschaften für eine Transaktion und nutzt die Sprache SQL, um auf Daten zugreifen und sie manipulieren [27], [28].

Architektur

Die Architektur von MariaDB beruht auf einer Multi-Master-Replikation, der Galera-Clustering-Lösung, wodurch MariaDB hochverfügbar und skalierbar ist. Bei der Multi-Master-Replikation werden alle Server in ein Cluster synchronisiert, sodass das Einfügen, Löschen und Ändern der Daten gleichzeitig auf allen Servern durchgeführt werden kann. Die Server dürfen nicht voneinander abweichen und müssen alle den aktuellen Datenbestand haben. MariaDB wird mit MaxScale ergänzt, der ein Datenbankrouter für das Load Balancing von Galera-Clustern ist und die Master-Slave-Replikation sowie Replikations-Relay unterstützt. MaxScale steuert die Kommunikation zwischen der Anwendung und der Datenbank. Die Änderungen an den Datenbank-Clustern haben keine Auswirkung auf die Anwendung. Damit kann die Datenbank von der Anwendungsschicht unabhängig skaliert werden. MaxScale gleicht MariaDB bei der Verarbeitung von Abfragen und hat die Geschwindigkeitsvorteile von MariaDB [28].

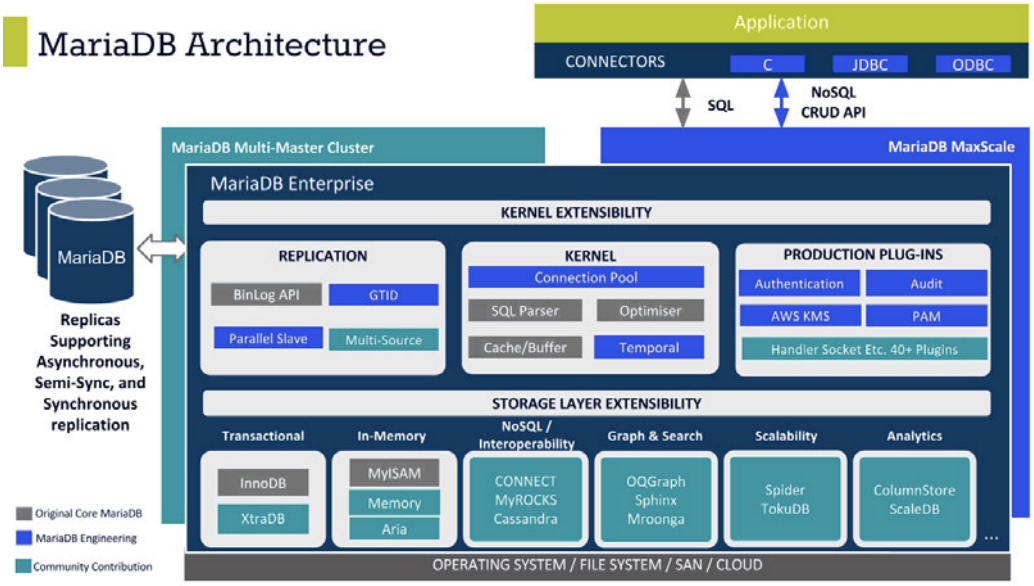

Abbildung 3. 1: MariaDB-Architektur © 2022 MariaDB[5]

Storage Engine

MariaDB basiert auf einer modularen Speicherarchitektur, die für jeden Anwendungsfall eine optimale Storage Engine anbietet. Die InnoDB-Engine eignet sich für die Transaktionen. Zusätzlich unterstützt die InnoDB-Engine die Komprimierung und Verschlüsselung sowie Schemaänderungen [29]. Bei massiven parallelen Architekturen wird ColumnStore verwendet, das nach einem spaltenbasierten Modell arbeitet. Die Connect-Engine erlaubt MariaDB aufgrund ihrer universellen Kommunikation, mit anderen DBMS zu kommunizieren. Spider bietet eine Sharding-Funktion von Daten an [28]. Spider in der Kombination mit InnoDB führt zu einer horizontalen Skalierung von Transaktionen. Aria ist eine ausfallsichere, nichttransaktionale Storage Engine. Die Storage Engine MyRocks bietet eine bessere Komprimierung mit einem geringere Schreibfaktor als InnoDB an [29].

[5] [28]

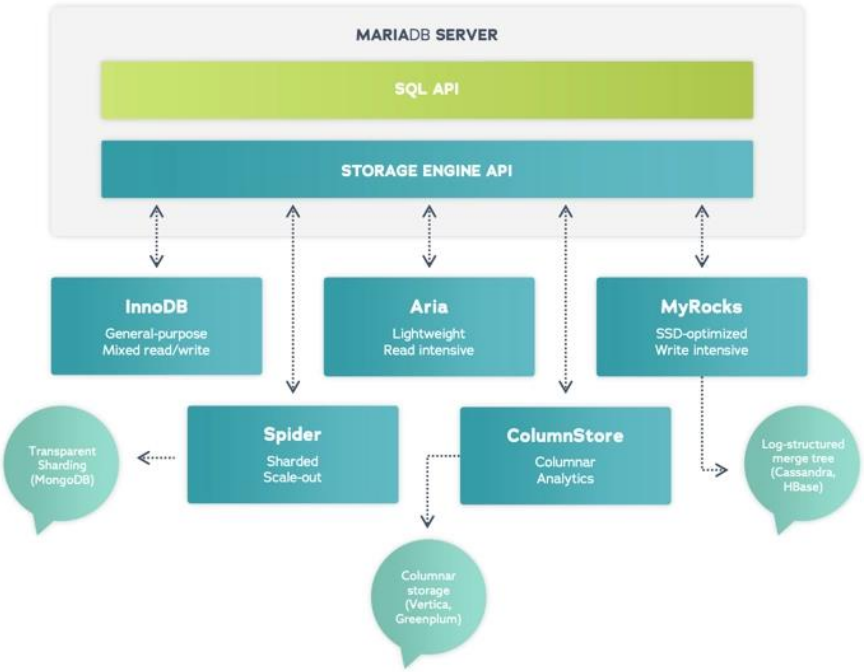

Abbildung 3. 2: Storage Engine © 2022 MariaDB[6]

3.1.2 MongoDB

MongoDB ist eine als Open Source verfügbare dokumentenorientierte NoSQL-Datenbank und unterstützt die SQL-Sprache nicht. Für die Manipulation, das Löschen und das Einfügen von Daten bietet MongoDB eine ähnlich leistungsfähige Abfragesprache, die MongoDB Query Language (MQL) [30, S. 144], [31]. MongoDB unterstützt keine ACID-Transaktionen und basiert stattdessen auf BASE, um eine hohe Verfügbarkeit für die Nutzende zu ermöglichen. Ab der MongoDB-Version 4.0 werden Funktionalitäten angeboten, die die ACID-Transaktionen unterstützen [32, S. 1559].

[6] [29]

Dokumentenorientierte Datenbank

MongoDB setzt sich im Vergleich zu einem RDBMS aus Datenbanken, Collections und Dokumenten zusammen. Eine MongoDB-Instanz kann aus einer oder mehreren Datenbanken bestehen. In diesen Datenbanken können eine oder mehrere Collections enthalten sein. Diese Collections ähneln den Tabellen in einem RDBMS. Jede Collection kann ein oder mehrere Dokumente besitzen, die als eine Menge von Schlüssel-Wert-Paaren gespeichert werden [30, S. 145], [33, S. 4]. Diese Schlüssel bestehen aus Strings und die Werte aus unterschiedlichen Datentypen. MongoDB speichert die Dokumente als JSON- oder BSON-Format, die kein vordefiniertes Schema benötigen. Damit können Daten mit unterschiedlichen Strukturen und komplexe Datentypen gespeichert werden. Ein Dokument ähnelt einer Zeile in einem RDBMS. Jedes Dokument besteht aus Feldern, die einer Spalte in einem RDBMS ähneln. Die Dokumente können mehr Informationen enthalten als eine Zeile aus einem RDBMS [32, S. 1556], [33, S. 4]. Jedes neu hinzugefügte Dokument erhält einen generierten Schlüssel von MongoDB, um die Dokumente schneller zu finden und zu sortieren [34].

Architektur

Die Architektur von MongoDB beruht auf einer Single-Master-Architektur, in der die Replica Sets aus einem Master- und mehreren Slave-Servern bestehen. Der Master ist der primäre Server, der für alle Schreiboperationen zuständig ist. Dieser aktualisiert die Slave-Server (sekundäre Server). Die sekundären Server können nur die Leseoperationen ausüben und dienen als Back-up, sollte der primäre Server ausfallen. Dann wird aus dem erreichbaren sekundären Server ein neuer primärer Server ausgewählt [35].

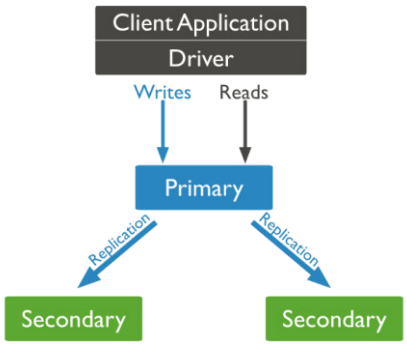

Abbildung 3. 3: MongoDB-Replica-Set-Architektur © 2022 MongoDB[7]

MongoDB unterstützt die horizontale Skalierung [5, S. 2]. Mithilfe von Sharding werden die großen Datenmengen auf mehreren Shards partitioniert und gespeichert. Jeder Shard kann als ein Replica Set fungieren. Die Daten werden nach Collections auf die Shards gespeichert. Jedes Dokument besitzt einen eindeutigen ‚Shard Key'. Die Datenaufteilung kann entweder durch Range-Sharding oder Hash-Sharding erfolgen. Bei der Hash-Sharding werden Hash-Werte durch die Hash-Funktion und den ‚Shard Key' definiert. Die Daten werden abhängig von ihrem Hash-Wert den jeweiligen Bereichen (Ranges) zugewiesen. Bei der Range-Sharding werden die Daten basierend auf dem ‚Shard Key' in Bereiche unterteilt, worin sich die Daten zu Datenblöcken (Chunks) bilden. Der Config-Server speichert die Metadaten und Konfigurationseinstellungen für den Cluster [36].

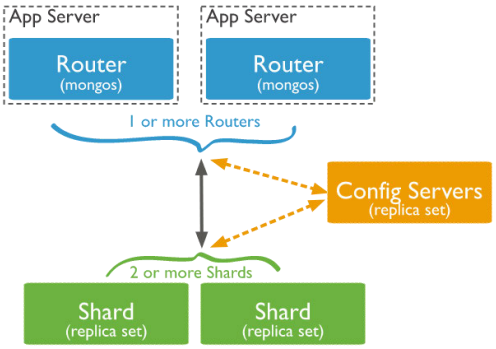

Abbildung 3. 4: Sharded Cluster in MongoDB © 2022[8]

[7] [65]

[8] [36]

Mithilfe der Metadaten kann der Router die richtigen Shards identifizieren und die jeweilige Benutzerabfrage zu den entsprechenden Shards weiterleiten [37].

Storage Engine

MongoDB unterstützt WiredTiger und die In-Memory Storage Engine. WiredTiger wird ab der MongoDB-Version 3.2 als Standard-Storage-Engine genutzt und eignet sich für die meisten Workloads. In-Memory wird in der MongoDB-Enterprise-Version unterstützt, in der die Dokumente im Arbeitsspeicher gespeichert werden [38].

3.1.3 CockroachDB

CockroachDB ist ein als Open Source verfügbares kommerzielles RDBMS, das für globale OLTP-Workloads geeignet ist. Daten aus heterogenen Quellen werden in Tabellen eingefügt und gespeichert. Der Zugriff auf die Tabellen wird durch die Sprache SQL ermöglicht. Zudem unterstützt CockroachDB bei einer Transaktion die ACID-Eigenschaften.

Architektur

Die Architektur von CockroachDB basiert auf einer Shared-Nothing-Architektur. Die großen Datenmengen werden partitioniert und mindestens drei Datenreplikate werden in verschiedenen geografischen Zonen gespeichert. Die Daten werden auf den Servern, die sich in der Nähe der Nutzenden befinden, gespeichert. Dadurch ermöglicht CockroachDB einen schnelleren Zugriff auf die Daten und reduziert so die Latenz. Die Server befinden sich in einem Cluster, der sich entweder in einem Rechenzentrum befindet oder in verschiedenen geografischen Zonen verteilt ist. Beim Hinzufügen neuer Server, die geografisch verteilt sind, unterstützt CockroachDB die horizontale Skalierung. Dadurch wird die Kapazität automatisch erhöht und die Daten werden auf die erreichbaren Server verteilt; somit erzielt CockroachDB eine hohe Leistung und sorgt für eine Fehlertoleranz. Der Nutzende kann sich in CockroachDB mit einem Server im Cluster verbinden und über das SQL-Interface Transaktionen ausführen. Durch die geografische Verteilung der Server können die Nutzende aus unterschiedlichen geografischen Zonen Transaktionen ausführen. Damit ermöglicht CockroachDB eine verteilte Transaktion, die mithilfe der Unterstützung der ACID-Eigenschaften für konsistente Daten in der Datenbank sorgt.

Der CockroachDB-Server beruht auf der Schichtenarchitektur, die aus der SQL-, der transaktionalen und der Verteilungsschicht besteht. Die SQL-Schicht dient als Schnittstelle zwischen dem Nutzenden und der Datenbank. Sie umfasst den Parser, den Optimierer und die Execution Engine, die High-Level-SQL-Abfragen für die darunter liegenden Schichten zu Low-Level-Lese- und Schreibabfragen umwandelt. Die beiden Schichten unter der SQL-Schicht bilden gemeinsam einen einzigen monolithischen Schlüssel-Wert-Speicher. Die Abfragen aus der SQL-Schicht werden an die transaktionale Schicht geleitet. Die transaktionale Schicht ist für die atomaren Daten sowie die Isolation der Transaktion zuständig. Die Verteilungsschicht bildet einen monolithischen logischen Schlüsselraum, in dem alle Daten über einen Schlüssel adressiert und geordnet sind. Mithilfe der Range-Partitionierung (Bereichspartitionierung) werden die Schlüssel mit den dazugehörigen Daten in zusammenhängende geordnete Blöcke unterteilt, die im gesamten Cluster gespeichert werden. Diese werden auch als Bereiche bezeichnet. Die Verteilungsschicht ist für die Identifizierung der Bereiche zuständig. Die Abfrage eines bestimmten Datensatzes wird an den entsprechenden Bereich weitergeleitet und ausgeführt. Aufgrund der Architektur von CockroachDB werden automatische Wiederherstellungsmechanismen und Replikationen unterstützt, um gegen sämtliche Serverausfälle widerstandsfähig zu sein und für eine hohe Verfügbarkeit zu sorgen [39, S. 1493–1495].

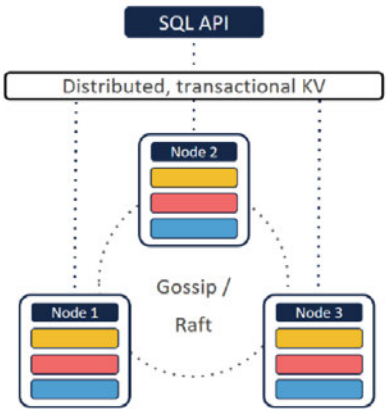

Abbildung 3. 5: CockroachDB-Server-Architektur[9]

[9] [18, S. 4]

Storage Engine

Jeder Server in CockroachDB enthält mindestens einen Speicher, von dem die Daten gelesen und auf die Festplatte geschrieben werden. Die Daten werden in der Storage Engine als Schlüssel-Wert-Paare auf der Festplatte gespeichert. CockroachDB verwendet die Storage Engine Pebble, die in der Programmiersprache Go geschrieben ist und ähnliche Funktionen wie RocksDB anbietet. Jeder Speicher enthält zwei Instanzen der Storage Engine: eine Instanz für das Speichern temporärer verteilter SQL-Daten und eine weitere Instanz für alle anderen Daten auf dem Server [40].

3.2 Anwendungsgebiete

MariaDB, MongoDB und CockroachDB unterscheiden sich in der Architektur und besitzen dadurch unterschiedliche Funktionalitäten, die für unterschiedliche Anwendungsgebiete geeignet sind. In Tabelle 3.1 sind die Eigenschaften und Funktionen der drei Datenbanksysteme zusammengestellt.

Tabelle 3. 1: Vergleich von Eigenschaften zwischen MariaDB, MongoDB und CockroachDB[10]

Datenbank	**MariaDB**	**MongoDB**	**CockroachDB**
Datenmodell	SQL	NoSQL	NewSQL
Speicherung der Daten	Tabelle	Collection	Tabelle
Einträge	Zeilen	Dokumente	Zeilen
OLTP-Workload	Ja	Ja	Ja
Entwickler	MariaDB Corporation/ Foundation	MongoDB, Inc	Cockroach Labs

[10] [60]

Open Source	Ja	Ja	Ja
Konsistente Daten	Ja	Eventuelle Konsistenz	Ja
Verfügbarkeit	Ja	Ja	Ja
Transaktion	ACID-Transaktion	BASE, ACID ab MongoDB 4.0	ACID-Transaktion
Skalierbarkeit	Horizontale Skalierung	Horizontale Skalierung	Horizontale und vertikale Skalierung
SQL	Ja	Nein, MQL	Ja
On Premises und Cloud-Based	Ja	Ja	Ja
Dauerhaftigkeit der Daten	Ja	Abhängig von der Konfiguration	Ja
Veränderungen am Datenbankschema	Online[11]	Offline	Online
Geo-Partitionierung	Ja[12]	Ja, geografische Sharding[13]	Ja
Implementierungssprache	C, C++ [14]	C++	Golang

[11] [66]

[12] [67]

[13] [68]

[14] [69]

Betriebssystem	Debian, Linux, Ubuntu, CentOS, Windows[15]	Linux, MacOS, Solaris, Windows	Linux, MacOS, Windows
Programmiersprache	C, C++, Java, JavaScript (Node.js), .NET, PHP, Python, Ruby, etc.[16]	C, C#, C++, Java, JavaScript, JSON, Perl, PHP, Python, Ruby, etc.	C#, C++, Java, JavaScript (Node.js), PHP, Python, Ruby, etc.

3.3 Vergleichskriterien

Die Vergleichskriterien bestehen aus den folgenden drei Analysen:

1. strukturelle Analyse,
2. funktionale Analyse,
3. nichtfunktionale Analyse.

Der Fokus der Analyse und Evaluation der Systeme liegt darin, die Eigenschaften und Funktionen darzustellen und miteinander zu vergleichen.

3.3.1 Strukturelle Analyse

Bei der strukturellen Analyse wurde ein konzeptioneller Vergleich zwischen den Datenbanksystemen über die Indexe durchgeführt. Dabei wurden die verschiedenen Indizierungstechniken genannt und miteinander verglichen.

[15] [70]

[16] [71]

3.3.2 Funktionale Analyse

Bei der funktionalen Analyse wurden die Systeme auf die Trigger-Funktion geprüft, um die Umsetzung, die Unterschiede und die Ähnlichkeiten der Trigger miteinander zu vergleichen. Ein Trigger wird automatisch ausgelöst, wenn ein bestimmtes Ereignis in der Datenbank aufgetreten ist. In einem Trigger wird festgelegt, auf welche Weise die Daten geändert werden sollen [41]. Zudem wurden bei der funktionalen Analyse die Eigenschaften einer Transaktion geprüft. Beim Vergleich lag der Fokus darauf, zu prüfen, ob die Datenbanken eine Transaktion mit den ACID-Eigenschaften unterstützen und inwiefern diese Transaktionen umgesetzt werden. CockroachDB und MariaDB unterstützen beide die ACID-Eigenschaften in einer Transaktion, um konsistente und korrekte Daten zu sichern. MongoDB hingegen basiert auf BASE, was für eine hohe Verfügbarkeit und inkonsistente Daten sorgt. Dennoch bietet MongoDB Funktionen an, um atomare, konsistente, dauerhafte und isolierte Transaktionen zu ermöglichen.

3.3.3 Nichtfunktionale Analyse

Bei der nichtfunktionalen Analyse wurde die Leistung der Systeme miteinander verglichen. Dafür wurden 1 bis 1 000 000 strukturierte Datensätze im System eingefügt und gespeichert. Auf die Datensätze wurden CRUD-Operationen angewendet. CRUD besteht aus den Operationen ‚Create', ‚Read', ‚Update' und ‚Delete'. Bei ‚Create' wurden die jeweiligen Tabellen bzw. Collections erstellt. Dabei wurde die Geschwindigkeit beim Einfügen der Daten in die Datenbanken gemessen und verglichen.

Bei ‚Read' wurden fünf Vergleiche durchgeführt:

1. Alle Datensätze werden ausgegeben.
2. Ein spezifischer Datensatz in der Datenbank wird ausgelesen.
3. Ein Index wird erstellt und über den Index wird ein bestimmter Datensatz gesucht und ausgelesen.
4. Zwei Tabellen/Collections werden mithilfe der ‚Join'-Operation verbunden und sämtliche Daten ausgegeben.

5. Drei Tabellen/Collections werden mithilfe der ‚Join'-Operation verbunden und sämtliche Daten ausgegeben.

Bei den fünf verschiedenen ‚Read'-Abfragen wurde die Geschwindigkeit gemessen und damit die Leistung der Read-Operationen geprüft. Ziel der ‚Read'-Abfragen war es, den Unterschied zwischen einer Suche ohne Verwendung eines Index und einer optimierten Suche mit Verwendung eines Index darzustellen. Zusätzlich wurde bei den ‚Read'-Abfragen geprüft, ob die ‚Join'-Operation unterstützt wird und welchen Einfluss das Nutzen der ‚Join'-Operation auf die Leistung des Systems hat.

Hinsichtlich ‚Update' wurden zwei Änderungsoperationen auf die Daten angewendet:

1. Ein spezifischer Datensatz wird verändert.
2. Die gesamten Daten in der Datenbank werden verändert.

Bei den beiden Änderungsoperationen wurden die Geschwindigkeit und die Umsetzung der Änderungen geprüft.

Bei ‚Delete' wurden zwei Löschoperationen auf die Datenmenge angewendet:

1. Ein spezifischer Datensatz in der Datenbank wird gelöscht.
2. Alle Daten in der Datenbank werden gelöscht.

Bei dieser Analyse wurde die Korrektheit der Ausführung der Abfrage geprüft und die Geschwindigkeit verglichen.

4 Experiment

4.1 Performancemetrik

Sämtliche Tests wurden mittels eines Laptops mit den folgenden Konfigurationen durchgeführt:

- Betriebssystem: Windows
- CPU: Intel Core i7-10510U mit 4 Cores
- RAM: 16GB

Die Python-Skripte wurden mit PyCharm 2021.2.3 mit Python Interpreter ‚Python 3.10' ausgeführt. Zudem wurden die Tests auf dem Localhost durchgeführt.

4.2 Struktureller Test

In diesem Abschnitt werden verschiedene Indizierungstechniken von MariaDB, MongoDB und CockroachDB vorgestellt. Indexe sind essentiell für die Datenbanken und ermöglichen eine Zugriffsstruktur für die Daten in der Datenbank. Ohne Indexe muss die gesamte Tabelle nach den relevanten Daten durchsucht werden, weshalb Indexe zur Optimierung der Leistung bei der Suche von Datensätzen beitragen. Durch das Erstellen von Indexen wird für sortierte Daten und eine schnellere Suche gesorgt [42].

4.2.1 Indexe in MariaDB

Die Performanceabfrage in MariaDB kann durch verschiedene Arten von Indexen optimiert werden. Einige Typen von Indexen, die MariaDB unterstützt, sind:

- Primary Key (Primärschlüssel),
- Unique Index,
- Plain Index.

Ein Primärschlüssel in MariaDB besteht aus einem einzelnen Feld oder aus einer Kombination von Feldern, die einen Datensatz innerhalb einer Tabelle eindeutig macht. Jedem Datensatz ist

nur ein Primärschlüssel zugeordnet, der keinen Null-Wert enthalten darf. Ein Unique Index ist einzigartig und kann Null-Werte enthalten. Plain Indexes müssen hingegen nicht einzigartig sein [43], [44]. Die Storage Engine InnoDB in MariaDB nutzt für den Index einen balancierten Baum (BTree). Aria und MyISAM verwenden den BTree und den Real Tree (RTree) als Indexstruktur. Die Storage Engine Memory/HEAP nutzt den Hash oder den BTree [45].

4.2.2 Indexe in MongoDB

In MongoDB wird für das Erstellen eines Index ein spezifisches Feld oder eine Gruppe von Feldern aus einem Dokument als Index festgelegt und gespeichert. MongoDB bietet verschiedene Indextypen an. Einige davon sind:

- Primärschlüssel,
- Single-Field und Multi-Field Index,
- Unique Index.

Für den Primärschlüssel generiert MongoDB mit ‚ObjectId' automatisch ein eindeutiges Feld ‚_id' in einer Collection. Beim Single-Field Index wird ein Index auf ein Feld in einem Dokument erzeugt, der nicht einzigartig sein muss. Dabei wird festgelegt, ob die Dokumente auf- oder absteigend sortiert werden sollen. Diese Funktion ähnelt dem Plain Index in MariaDB. Neben dem Erstellen des Index auf einem Feld können mehrere Felder aus einem Dokument kombiniert werden. Beim Erstellen des Index kann mithilfe von ‚Unique' der Index einzigartig gemacht werden, um Duplikate zu verhindern [34]. Die Storage Engine WiredTiger unterstützt den BTree und den Log-Structured Merge Tree (LSM-Tree) bei dem Sortieren von Dokumenten, um die Einträge schnell und effizient zu finden [46].

4.2.3 Indexe in CockroachDB

CockroachDB erstellt beim Index für die entsprechenden Spalten eine Kopie, in der die Daten sortiert sind. Die eigentlichen Tabellen in der Datenbank bleiben unsortiert. Dadurch wird die Anzahl der Zeilen bei der Suche des entsprechenden Datensatzes reduziert und das komplette Scannen der Tabelle wird vermieden. Die Indexe werden von CockroachDB direkt in den Schlüssel-Wert-Speicher eingefügt. CockroachDB unterstützt folgende Indextypen:

- Primärschlüssel,
- Sekundärer Index (Secondary Index),
- Unique Index.

Der Primärschlüssel wird in CockroachDB automatisch als Index gewählt. Wird kein Primärschlüssel festgelegt, so wird ein einzigartiger Schlüssel für jede Zeile als ‚rowid' genutzt. Der sekundäre Index dient der Suche nach Spalten, die nicht im Primärschlüssel enthalten sind. Dieser sekundäre Index kann als ein Unique Index oder nicht eindeutiger Index festgelegt werden [47]. Die Storage Engine Pebble von CockroachDB basiert auf einem LSM-Tree [40].

4.2.4 Zusammenfassung der Ergebnisse

Indexe dienen dazu, die Suche nach bestimmten Dateneinträgen zu optimieren, indem nur die zur Abfrage passenden Einträge sortiert werden, wodurch eine schnellere Suche ermöglicht wird. Besonders bei großen Datenmengen mit häufigen Datenabfragen ist die Nutzung von Indexen empfehlenswert. Nach dem Durchlaufen der verschiedenen Indizierungstechniken stellte sich Folgendes heraus:

- Die drei Datenbanken bieten gleiche Indexfunktionen an, agieren aber auf unterschiedliche Weise.
- LSM-Tree ist unter den NoSQL-Datenbanksystemen populär aufgrund der hohen Speicherplatznutzung, der überlegenen Schreibleistung und der Unveränderlichkeit der Daten auf der Festplatte [48, S. 416].
- BTree ist unter den Datenbanksystemen, insbesondere RDBMS, weitverbreitet, da die meisten Abfragen auf den Bereichs- und Gleichheitsvergleichen basieren und den Datenzugriff beschleunigen [49, S. 7].

Beim Vergleich der Performance zwischen den drei verschiedenen Datenbanken wird in Abschnitt 4.4 *Nicht-Funktionaler Test* der Fokus auf den Plain Index, einen nicht einzigartigen Index, gelegt.

Tabelle 4. 1: Zusammenfassung der Indexeigenschaften

Datenbank	**MariaDB**				**MongoDB**	**Cockro-achDB**
Primary Key	Primary Key				_id	rowid
Unique Index	CREATE UNIQUE INDEX…;				createIndex({<Fieldname>}, {unique: true})	CREATE UNIQUE INDEX…;
Index	CREATE INDEX…;				createIndex()	CREATE INDEX…;
Storage Engine	Aria	My-ISAM	InnoDB	Memmory /HEAP	WiredTiger	Pebble
Index-struktur	BTree, RTree	BTree, RTree	BTree	Hash, BTree	LSM-Tree, BTree	LSM-Tree

4.3 Funktionaler Test

In diesem Abschnitt werden die Funktionen Trigger und ACID-Transaktion erläutert und miteinander verglichen.

4.3.1 Versuchsaufbau

Die Trigger-Analyse erfolgte nur für die Datenbanken MariaDB und MongoDB, da CockroachDB die Trigger-Funktion nicht unterstützt. Für die Analyse wurden zwei Entitäten, ‚Mitarbeiter' und ‚Abteilung', benötigt. Die Entität ‚Abteilung' besitzt zwei Attribute: ‚Abteilung_Name' und ‚Mitarbeiter_Anzahl'. Die Entität ‚Mitarbeiter' besitzt folgende Attribute: ‚Vorname', ‚Nachname', ‚Gehalt', ‚Berufserfahrung' und ‚Abteilungsname'. Die folgende Tabelle 4.2 beinhaltet die genutzten Systeme für die Durchführung der Trigger-Analyse.

Tabelle 4. 2: Datenbanken und Software

Datenbanken	***MariaDB***	***MongoDB***	***CockroachDB***
Software/ Plattform	HeidiSQL	MongoDB Atlas	-

4.3.2 Trigger

Beim Vergleich wurde ein Trigger erstellt, der die Mitarbeiteranzahl in der Tabelle/Collection ‚Abteilung' automatisch erhöht, wenn der jeweiligen Abteilung ein neuer Mitarbeiter hinzugefügt wird.

Trigger in MongoDB

MongoDB hat drei Arten von Triggern: Database Trigger, Scheduled Trigger und Authentication Trigger.

Authentication Trigger (Authentifizierungsauslöser):
Der Authentication Trigger erzeugt Aktionen, die beim Authentifizieren eines Benutzers entstehen, z. B. das Löschen, Anmelden oder Erstellen eines Benutzers.

Database Trigger (Datenbankauslöser):
Der Database Trigger führt beim Ändern, Löschen oder Hinzufügen eines Dokuments eine bestimmte Aktion aus.

Scheduled Trigger (geplanter Auslöser):
Der Scheduled Trigger führt geplante Aktionen zu einem bestimmten Zeitpunkt aus.

Im Rahmen dieses Vergleichs wird der Fokus nur auf den Database Trigger gelegt. Die drei Arten von Triggern sorgen in MongoDB für Überwachung, Datenkonsistenz, Datenintegrität und Datenereignisse.

Überwachung:
Bei der Überwachung können die Nutzende, die ein Dokument verändert haben, identifiziert werden.

Datenkonsistenz:
Datenkonsistenz kann durch die Trigger erzeugt werden, um sicherzustellen, dass die eingefügten Daten dem vordefinierten Format entsprechen.

Datenintegrität:
Die Datenintegrität wird durch die Trigger erzeugt, um sicherzustellen, dass Dokumente eine gültige Kombination von Informationen enthalten.

Datenereignisse:
Die Trigger in MongoDB erzeugen eine verschachtelte Reihe von Ereignissen, die nacheinander ausgelöst werden [50].

Trigger in MariaDB

In MariaDB werden drei Arten von Triggern unterstützt: Insert-, Update- und Delete-Trigger. Diese Trigger werden automatisch ausgelöst, wenn ein bestimmtes Ereignis eingetreten ist.

Zudem wird jeder Trigger einem bestimmten Zeitpunkt mit dem Before- und dem After-Trigger zugeordnet.

Before-Trigger:
Der Before-Trigger sorgt für das Ausführen einer festgelegten Aktion, bevor ein bestimmtes Ereignis eintritt.

After-Trigger:
Der After-Trigger wird ausgelöst, wenn ein bestimmtes Ereignis eingetreten ist [51].

Im Vergleich zu MongoDB bietet MariaDB die Funktion ‚EVENT' an, die dem Scheduled Trigger in MongoDB als Funktion ähnelt. ‚EVENTS' sind zeitliche Trigger, die mithilfe von SQL-Anweisungen als Nicht-‚Data Manipulation Language'(DML)-Trigger erstellt werden und eine SQL-Anweisung zu einem bestimmten Zeitpunkt ausführen lassen [52].

4.3.3 Durchführung

MongoDB

In MongoDB Atlas wird vor dem Einfügen der Daten ein Projekt mit einem Cluster erzeugt. In diesem Cluster werden die zwei Collections ‚Mitarbeiter' und ‚Abteilung' erstellt. Für das Erstellen eines Triggers bietet MongoDB Atlas die Funktion ‚Add Trigger' unter den Data Services ‚Triggers' an. Daraufhin wird eine Konfigurationsseite geöffnet, die einen Datenbank-Trigger erzeugt. Dafür muss das jeweilige Cluster, die Datenbank und die Collection ausgewählt werden. Zudem werden für den Trigger die Operationen ‚Insert', ‚Update', ‚Replace' und ‚Delete' festgelegt. Im Vergleichsszenario wurde nur die ‚Insert'- und die ‚Update'-Operation benötigt. Für das Verhalten der Trigger wurde unter ‚Function' eine ‚node.js'-Funktion implementiert, die die Mitarbeiteranzahl beim Hinzufügen eines neuen Mitarbeiters erhöht. Mithilfe der ‚Insert'-Methode wurde ein Mitarbeiter eingefügt und mit der ‚Update'-Methode die Mitarbeiteranzahl der jeweiligen Abteilung erhöht (siehe Listing 4.1).

Listing 4. 1: Trigger in MongoDB mit node.js – automatische Erhöhung der Mitarbeiteranzahl in ‚Abteilung'

```
exports = async function(changeEvent) {

const abt= context.services.get("Cluster0").db("db_test").collection("Ab-
teilung");

const mit= context.services.get("Cluster0").db("db_test").collection("Mi-
tarbeiter");

await mit.insertOne({"Vorname":"Yasmin","Nachname":"Kohi","Berufserfah-
rung":5,"Gehalt":1450, "Abteilung_Name":"IT"});

await abt.updateOne({"Abteilung_Name":"IT"},{"$inc":{"Mitarbeiter_An-
zahl":1}});

return; };
```

MariaDB

In MariaDB wurden zwei Tabellen, eine für ‚Mitarbeiter' und eine für ‚Abteilung', erstellt. Mithilfe der SQL-Abfrage wurde die Anzahl der Mitarbeiter beim Hinzufügen eines neuen Mitarbeiters automatisch erhöht (siehe Listing 4.2). Dabei wurde für den Trigger der Name, das Ereignis mit den dazugehörigen Tabellen und die Operation, die ausgeführt werden soll, festgelegt.

Listing 4. 2: Trigger in MariaDB mit SQL – automatische Erhöhung der Mitarbeiteranzahl in ‚Abteilung'

```
CREATE TRIGGER inc_Abteilung_Mitarbeiter

AFTER INSERT ON Mitarbeiter FOR EACH ROW UPDATE

Abteilung, Mitarbeiter SET Abteilung.Anzahl_Mitarbeiter = Abteilung.An-
zahl_Mitarbeiter + 1

WHERE Abteilung.Abteilung_Name = Mitarbeiter.Abt_Name;
```

4.3.4 Zusammenfassung der Ergebnisse

Das Erstellen eines Triggers in MariaDB ist im Vergleich zu MongoDB unkompliziert und schnell. Mithilfe einer SQL-Anweisung konnte ein Trigger für das Ereignis ‚neuer Mitarbeiter' erstellt werden, der automatisch die Mitarbeiteranzahl zu der passenden Abteilung erhöht. Bei der SQL-Anweisung konnten zwei Tabellen über den Fremdschlüssel ‚Abt_Name' in der Tabelle ‚Mitarbeiter' und über den Primärschlüssel ‚Abteilung_Name' in ‚Abteilung' verknüpft werden. MongoDB unterstützt keine Relationen oder ‚Join'-Operationen; dadurch ist das Verknüpfen von zwei Collections über einen Fremdschlüssel nicht möglich. Deshalb ist der Aufwand für das Erstellen eines Triggers, der zwei Collections umschließt, größer als bei MariaDB. Nur durch die Funktionen ‚Embedding' oder ‚Reference' können zwei Dokumente miteinander verbunden und mit der ‚Update'-Methode die Anzahl der Mitarbeiter verändert werden. Außerdem kann in MariaDB mithilfe einer SQL-Anweisung über verschiedene Programme und Tools, z. B. MySQL Client (MariaDB), HeidiSQL oder das Python-Modul, ein

Trigger erstellt werden. Hingegen bietet MongoDB die Trigger-Funktion nur in MongoDB Atlas an, die Verfügbarkeit der Funktion ist also einschränkt. MariaDB hat unter anderem folgende Limitationen in der Trigger-Funktion:

- Kein ‚ResultSet' und keine Ergebnisse werden zurückgegeben.
- Der Trigger wird nicht durch die Aktionen von Fremdschlüsseln ausgelöst.
- Ein Trigger wird für jede Zeile in einer Tabelle ausgeführt.
- Bis MariaDB-Version 10.2.3 konnte nur eine Tabelle einen Trigger besitzen [53]

Des Weiteren haben auch die Trigger in MongoDB unter anderem folgende Limitationen:

- Die Anzahl von Triggern, die über die Anzahl der Change Streams festgelegt werden, ist begrenzt.
- Der MongoDB Atlas App Service schränkt die Ausführung von Triggern auf 2500-mal pro Sekunde ein [54].
- Die Nutzung ist kostenpflichtig.

Tabelle 4. 3: Zusammenfassung der Trigger-Eigenschaften

Datenbank	**MariaDB**	**MongoDB**	**CockroachDB**
Trigger	Ja	Ja	-
Typen	Events, Trigger	Database Trigger, Scheduled Trigger, Authentication Trigger	-
Operationen	INSERT, UPDATE, DELETE	INSERT, UPDATE, DELETE, REPLACE	-
Sprache	SQL	Node.js	-
Syntax	CREATE TRIGGER <identifier>	exports = function(changeEvent){...};	-

4.3.5 ACID-Transaktion

Im folgenden Abschnitt wird geprüft, ob die Datenbanken CockroachDB, MariaDB und MongoDB Transaktionen mit der ACID-Eigenschaft unterstützen. Dabei wurden die Datenbanksysteme auf ihre Atomizität, Konsistenz, Isolation und Dauerhaftigkeit untersucht und miteinander verglichen.

4.3.6 Versuchsaufbau

Bei der Analyse der ACID-Eigenschaften wurde die MongoDB-Version 5.0.3 verwendet, weil MongoDB ab Version 4.0 die Multi-Document-ACID-Transaktion unterstützt. Für den Transaktionsvergleich wurden in MongoDB drei Standalone-Instanzen zu einem Single-Node Replica Set umgewandelt, wobei eine als Master und die anderen zwei als Slaves fungierten. Über den Mongo Shell wurden die Transaktionen in MongoDB ausgeführt. Die folgende Tabelle 4.4 beinhaltet die Datenbanken und ihre jeweiligen Konnektoren.

Tabelle 4. 4: Datenbankenversionen und Konnektoren

Datenbank	***MariaDB***	***MongoDB***	***CockroachDB***
Datenbankversion	10.6	5.0.3	21.2.9
Interface/Connector	HeidiSQL	Mongo Shell	Psycopg2

Beim ACID-Vergleich wurden zwei Tabellen und Collections für die Entitäten ‚Mitarbeiter' und ‚Abteilung' erstellt. Die Entität ‚Mitarbeiter' besitzt folgende Attribute: ‚Vorname', ‚Nachname', ‚Gehalt', ‚Berufserfahrung' und ‚Abteilungsname'. Die Entität ‚Abteilung' besteht aus den zwei Attributen ‚Abteilung_Name' und ‚Anzahl_ Mitarbeiter'. Die Tabelle ‚Abteilung' wurde mit einem Datensatz befüllt. Nach dem Ausführen einer Transaktion wurden die Tabellen/Collections gelöscht, um Duplikate und vorhandene Daten zu vermeiden. Zudem wurde das automatische Ausführen einer Transaktion in MariaDB ausgeschaltet, damit die Transaktionen manuell ausgeführt werden konnten.

4.3.7 Durchführung der ACID-Transaktion

Atomizität

MariaDB:

Bei der Atomizität müssen alle Statements innerhalb einer Transaktion erfolgreich ausgeführt werden. Im Falle eines Fehlers müssen sämtliche Statements innerhalb einer Transaktion gemeinsam fehlschlagen [55]. Als Statements wurden die ‚Update'- und die ‚Insert'-Operation verwendet. Die ‚Insert'-Operation sollte einen neuen Mitarbeiter einfügen, während die ‚Update'-Operation die Anzahl der Mitarbeiter in einer Abteilung erhöhen sollte.

In MariaDB wurde eine Transaktion mit der Methode ‚START TRANSACTION' gestartet und mit ‚COMMIT' ausgeführt. Innerhalb dieser Transaktion wurde mithilfe der ‚Insert'-Operation ein Datensatz in die Tabelle ‚Mitarbeiter' eingefügt und die Anzahl der Mitarbeiter in der Abteilung ‚Informatik' durch die ‚Update'-Operation erhöht. Diese Transaktion wurde erfolgreich ausgeführt und alle Änderungen in den Tabellen wurden umgesetzt (siehe Listing 4.3).

Listing 4. 3: Erfolgreiche Transaktion in MariaDB

```
START TRANSACTION;
INSERT INTO Miterarbeiter(Mitarbeiter_ID, Vorname, Nachname, Berufserfah-
rung, Gehalt, Abt_Name) VALUES (1, ,'Yasmin' ,'Kohi',0,100,'Informatik');
UPDATE abteilung a INNER JOIN  mitarbeiter m ON
a.Abteilung_Name=m.Abt_Name SET  a.Anzahl_Mitarbeiter=(a.Anzahl_Mitarbei-
ter+1) ;
COMMIT;
```

Als Gegenvergleich wurde in der ‚Insert'-Operation eine Information ausgelassen und der Datentyp verändert. Beim Ausführen der Transaktion wurde ein Fehler geworfen, weil die Anzahl der übergebenen Informationen und die Anzahl der Spalten einer Tabelle nicht übereinstimmten. Zudem war der Datentyp für das Attribut ‚Berufserfahrung' falsch, da die Information als ein String übergeben wurde. Als Ergebnis wurde die Transaktion nicht ausgeführt und die Änderungen wurden verworfen (siehe Listing 4.4).

Listing 4. 4: Fehlerhafte Transaktion in MariaDB

```
START TRANSACTION;
INSERT INTO Miterarbeiter(Mitarbeiter_ID, Vorname, Nachname, Berufserfahrung, Gehalt, Abt_Name) VALUES (1, ,'Yasmin','Kohi','0','Informatik');
UPDATE abteilung a INNER JOIN  mitarbeiter m ON a.Abteilung_Name=m.Abt_Name SET  a.Anzahl_Mitarbeiter=(a.Anzahl_Mitarbeiter+1) ;
COMMIT;
```

MongoDB:

In MongoDB wurde mit ‚startTransaction()' in einer Session eine Transaktion gestartet. In dieser Session wurden ‚readConcern' und ‚writeConcern' als ‚majority' festgelegt. Beim Festlegen von ‚writeConcern' als ‚majority' gilt eine Transaktion erst dann als erfolgreich, wenn die Mehrheit der Nodes in einem Replica Set diese Schreiboperation bestätigt. Beim Festlegen von ‚readConcerns' als ‚majority' werden für eine Transaktion die Daten gelesen, die die Mehrheit von Nodes in einem Cluster bestätigt haben [56]. In der Transaktion wurde mit der Methode ‚insertOne()' in der Collection ‚Mitarbeiter' ein neuer Datensatz eingefügt. Zudem wurde mit der Methode ‚updateOne' das Attribut ‚Anzahl_Mitarbeiter' in der Collection ‚Abteilung' angepasst. Mit der Methode ‚session.commitTransaction()' wurde eine Transaktion erfolgreich ausgeführt und die Änderungen wurden permanent in der Collection umgesetzt (siehe Listing 4.5).

Listing 4. 5: Erfolgreiche Transaktion in MongoDB

```
> var session = db.getMongo().startSession()
> session.startTransaction({      "readConcern": { "level": "snapshot" }, "writeConcern": { "w": "majority" } })
> var mit = session.getDatabase('db_test').getCollection('Mitarbeiter')
> mit.insertOne({"Vorname": "Yasmin","Nachname": "Kohi","Berufserfahrung": 2, "Gehalt": 34, "Abteilung_Name": "Informatik" })
```

```
{
        "acknowledged" : true,
        "insertedId" : ObjectId("62d34a04390806f2bee06206")
}

> var abt = session.getDatabase('db_test').getCollection('Abteilung')

>   abt.updateOne({"Abteilung_Name":"Informatik"},{"$set":{"Anzahl_Mitar-
beiter":1}})

{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

> mit.find()

{  "_id"  :  ObjectId("62d34a04390806f2bee06206"),"Vorname"  :  "Yasmin",
"Nachname" : "Kohi", "Berufserfahrung" : 2, "Gehalt" : 34, "Abteilung_Name"
: "Informatik" }

> abt.find()

{ "_id" : ObjectId("62d34532390806f2bee06204"), "Abteilung_Name" : "Infor-
matik", "Anzahl_Mitarbeiter" : 1 }

> session.commitTransaction()
```

Bei einer fehlerhaften Transaktion, in der ein Datentyp falsch eingetragen und ein Feld im Dokument ausgelassen wird, wurde kein Fehler geworfen (siehe Listing 4.6).

Listing 4. 6: Fehlerhafte Transaktion in MongoDB

```
> var mit = session.getDatabase('db_test').getCollection('Mitarbeiter')
> mit.insertOne({"Vorname": "Yasmin","Nachname": "Kohi","Berufserfah-
rung": "2", "Abteilung_Name": "Informatik" })

{    "acknowledged" : true,
     "insertedId" : ObjectId("63029250f8368d21d48bff07")
}
> var abt = session.getDatabase('db_test').getCollection('Abteilung')
> abt.updateOne({"Abteilung_Name":"Informatik"},{"$set":{"Anzahl_Mitar-
beiter":1}})
{ "acknowledged" : true, "matchedCount" : 0, "modifiedCount" : 0 }

> session.commitTransaction()
```

CockroachDB:

Die Syntax für eine Transaktion in CockroachDB fängt mit ‚BEGIN' an und endet mit ‚COMMIT'. Bei einer erfolgreichen Transaktion wurden sämtliche Transaktionen ausgeführt und im Falle eines Fehlers wurden die Änderungen verworfen (siehe Listing 4.7 und 4.8).

Listing 4. 7: Erfolgreiche Transaktion in CockroachDB

```
BEGIN;
INSERT INTO Miterarbeiter(Mitarbeiter_ID, Vorname, Nachname, Berufserfah-
rung, Gehalt, Abt_Name) VALUES (1, ,'Yasmin' ,'Kohi',0,100,'Informatik');
UPDATE abteilung a INNER JOIN  mitarbeiter m ON
a.Abteilung_Name=m.Abt_Name SET  a.Anzahl_Mitarbeiter=(a.Anzahl_Mitarbei-
ter+1) ;
COMMIT;
```

Listing 4. 8: Fehlerhafte Transaktion in CockroachDB

```
BEGIN;
INSERT INTO Miterarbeiter(Mitarbeiter_ID, Vorname, Nachname, Berufserfah-
rung, Gehalt, Abt_Name) VALUES (1, ,'Yasmin' ,'Kohi',0,'Informatik');
UPDATE abteilung a INNER JOIN  mitarbeiter m ON
a.Abteilung_Name=m.Abt_Name SET  a.Anzahl_Mitarbeiter=(a.Anzahl_Mitarbei-
ter+1) ;
COMMIT;
```

Isolation

MariaDB:

Für die Überprüfung der Isolation-Eigenschaft in MariaDB wurden zwei Instanzen erzeugt. Eine Instanz sollte einen neuen Datensatz in die Tabelle ‚Mitarbeiter' einfügen und die Mitarbeiteranzahl in der jeweiligen Abteilung erhöhen, während die zweite Instanz drei ‚Select'-Statements mit fünf Sekunden Abstand auf die Tabelle ‚Mitarbeiter' ausführen sollte. Die erste Transaktion auf der ersten Instanz wurde ohne ein ‚Commit' ausgeführt (siehe Listing 4.9).

Listing 4. 9: Instanz-1: Transaktion ohne ‚Commit' in MariaDB

```
START TRANSACTIN;
INSERT INTO Miterarbeiter(Mitarbeiter_ID, Vorname, Nachname, Berufserfah-
rung, Gehalt, Abt_Name) VALUES (1, ,'Yasmin' ,'Kohi',0,100,'Informatik');
UPDATE abteilung a INNER JOIN  mitarbeiter m ON
a.Abteilung_Name=m.Abt_Name SET  a.Anzahl_Mitarbeiter=(a.Anzahl_Mitarbei-
ter+1) ;
```

Parallel zur ersten Instanz wurden auf der zweiten Instanz die ‚Select'-Statements ausgeführt. Alle drei ‚Select'-Statements gaben den Zustand der Daten vor dem Ausführen der Transaktion zurück. Dies liegt daran, dass eine Transaktion ohne ein ‚Commit' nicht ausgeführt wird (siehe Listing 4.10).

Listing 4. 10: Instanz-2: ‚Select'-Abfrage auf eine Transaktion ohne ‚Commit' in CockroachDB und MariaDB

```
SELECT * FROM Abteilung;

SELECT * FROM Mitarbeiter;

SELECT SLEEP (5);

SELECT * FROM Abteilung;

SELECT * FROM Mitarbeiter;

SELECT SLEEP (5);

SELECT * FROM Abteilung;

SELECT * FROM Mitarbeiter;
```

Mit einem ‚Commit' in einer Transaktion wurden beim zweiten ‚Select'-Statement auf der zweiten Instanz die Veränderungen ausgegeben (siehe Listing 4.11). Somit konnte die zweite Instanz auf das Ergebnis anderer Transaktionen auf anderen Instanzen zugreifen, wenn die Transaktion erfolgreich beendet wurde.

Listing 4. 11: Instanz-2: ‚Select'-Abfrage bei einer erfolgreichen Transaktion in CockroachDB und MariaDB

```
SELECT * FROM Abteilung;
```

Abteilung_Name	Anzahl_Mitar-beiter
Informatik	1

```
SELECT * FROM Mitarbeiter;
```

Mitarbeiter_ID	Vorname	Nachname	Berufser-fahrung	Gehalt	Abt_Name
1	Yasmin	Kohi	0	100	Informatik

MongoDB:

In MongoDB wurden für die Isolationsprüfung zwei MongoDB-Shell-Instanzen verwendet. Eine Shell-Instanz sollte die Transaktion ausführen. Die zweite Shell-Instanz sollte mit der ‚Select'-Operation die Änderungen der ausgeführten Transaktion prüfen. Eine Transaktion wurde auf der ersten Shell-Instanz ohne ‚session.commitTransaction()' ausgeführt. Nach dem Ausführen der ‚Find'-Methode wurden die Änderungen auf der ersten Instanz ausgegeben (siehe Listing 4.12).

Listing 4. 12: Instanz-1: Transaktion ohne ‚Commit' in MongoDB

```
> var session = db.getMongo().startSession()
> session.startTransaction({     "readConcern": { "level": "majority" },
"writeConcern": { "w": "majority" } })

> var mit = session.getDatabase('db_test').getCollection('Mitarbeiter')

> mit.insertOne({"Vorname": "Yasmin","Nachname": "Kohi","Berufserfahrung":
2, "Gehalt": 34, " Abteilung_Name":"Informatik"})

{
        "acknowledged" : true,
        "insertedId" : ObjectId("6302994cf8368d21d48bff0a")
}
```

```
> mit.find()

{ "_id" : ObjectId("6302994cf8368d21d48bff0a"), "Vorname" : "Yasmin",
"Nachname" : "Kohi", "Berufserfahrung" : 2, "Gehalt" : 34, "Abtei-
lung_Name" : "Informatik" }

> var abt = session.getDatabase('db_test').getCollection('Abteilung')

> abt.updateOne({"Abteilung_Name":"Informatik"},{"$set":{"Anzahl_Mitar-
beiter":1}})

{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

> abt.find()

{ "_id" : ObjectId("63029504f8368d21d48bff09"), "Abteilung_Name" : "In-
formatik", "Anzahl_Mitarbeiter" : 1 }
```

Die Änderungen auf der ersten Instanz wurden bei der zweiten Instanz nicht ausgegeben. Dies liegt daran, dass die Transaktion nicht mit ‚session.commitTransaction()' ausgeführt wurde (siehe Listing 4.13).

Listing 4. 13: Instanz-2: ‚Find'-Abfrage bei einer fehlerhaften Transaktion in MongoDB

```
> db.getCollection('Abteilung').find()

> db.getCollection('Mitarbeiter').find()
```

Nach der zweiten Ausführung einer Transaktion mit ‚session.commitTransaction()' waren die Änderungen permanent in der Collection gespeichert und auf beiden Shell-Instanzen zu sehen (siehe Listing 4.14 und 4.15).

Listing 4. 14: Instanz-1: Transaktion mit Commit in MongoDB

```
> var mit = session.getDatabase('db_test').getCollection('Mitarbeiter')

> mit.insertOne({"Vorname": "Yasmin","Nachname": "Kohi","Berufserfah-
rung": 2, "Gehalt": 34, "Abteilung_Name": "Informatik" })

{
        "acknowledged" : true,
```

```
        "insertedId" : ObjectId("630294e8f8368d21d48bff08")
}
> mit.find()
{ "_id" : ObjectId("630294e8f8368d21d48bff08"), "Vorname" : "Yasmin",
"Nachname" : "Kohi", "Berufserfahrung" : 2, "Gehalt" : 34, "Abtei-
lung_Name" : "Informatik" }
> var abt = session.getDatabase('db_test').getCollection('Abteilung')
> abt.updateOne({"Abteilung_Name":"Informatik"},{"$set":{"Anzahl_Mitar-
beiter":1}})
{ "acknowledged" : true, "matchedCount" : 0, "modifiedCount" : 0 }

> abt.find()
{ "_id" : ObjectId("63029504f8368d21d48bff09"), "Abteilung_Name" : "In-
formatik", "Anzahl_Mitarbeiter" : 1 }

> session.commitTransaction()
```

Listing 4. 15: Instanz-2: ‚Find'-Abfrage bei einer erfolgreichen Transaktion in MongoDB

```
> db.getCollection('Abteilung').find()

{ "_id" : ObjectId("63029504f8368d21d48bff09"), "Abteilung_Name" : "Infor-
matik", "Anzahl_Mitarbeiter" : 1 }

> db.getCollection('Mitarbeiter').find()

{ "_id" : ObjectId("630294e8f8368d21d48bff08"), "Vorname" : "Yasmin",
"Nachname" : "Kohi", "Berufserfahrung" : 2, "Gehalt" : 34, "Abteilung_Name"
: "Informatik" }
```

CockroachDB:

In CockroachDB wurden zwei Instanzen erstellt. Auf der einen Instanz wurde die Transaktion ohne ein ‚Commit' ausgeführt (siehe Listing 4.16) und auf der zweiten Instanz wurden die ‚Select'-Abfragen nacheinander ausgeführt (siehe Listing 4.10).

Listing 4. 16: Instanz-1: Transaktion ohne ‚Commit' in CockroachDB

```
BEGIN;
INSERT INTO Miterarbeiter(Mitarbeiter_ID, Vorname, Nachname, Berufserfah-
rung, Gehalt, Abt_Name) VALUES (1, ,'Yasmin' ,'Kohi',0,100,'Informatik');
UPDATE abteilung a INNER JOIN  mitarbeiter m ON
a.Abteilung_Name=m.Abt_Name SET  a.Anzahl_Mitarbeiter=(a.Anzahl_Mitarbei-
ter+1) ;
```

Auch in CockroachDB wurden die Änderungen nicht umgesetzt, da die Transaktion kein ‚Commit' beinhaltete. Bei einer erfolgreichen Transaktion auf der ersten Instanz mit ‚Commit' waren die Änderungen auf der ersten sowie auf der zweiten Instanz zu sehen (siehe Listing 4.11).

Konsistenz

Die Konsistenz in einer Transaktion bedeutet, dass die Ausführung einer Transaktion die Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand überführt [55]. Dies erfolgt durch die Überprüfung des Datenzustands vor und nach der Ausführung einer Transaktion mithilfe der ‚Select'-Abfrage. Dabei werden die festgelegten Datentypen überprüft. CockroachDB und MariaDB wurfen einen Fehler beim Speichern von Daten mit falschen Datentypen. MongoDB ließ das Variieren von Datentypen auf ein Feld zu. Zudem wurde mittels des ‚Update'-Statements dafür gesorgt, dass ein Widerspruch zwischen den Daten aus den Tabellen ‚Abteilung' und ‚Mitarbeiter' vermieden wurde, indem die Anzahl der Mitarbeiter in der Tabelle ‚Abteilung' an die neu eingefügten Mitarbeiter angepasst wurde.

Dauerhaftigkeit

Bei der Überprüfung der Dauerhaftigkeit von Daten wurden die Veränderungen der Datensätze und die Umsetzung einer Transaktion mittels Leseabfragen geprüft.

4.3.8 Zusammenfassung der Ergebnisse

MariaDB, MongoDB und CockroachDB unterstützen die ACID-Transaktionen, indem sämtliche Statements innerhalb einer Transaktion gemeinsam, isoliert von anderen Transaktionen ausgeführt werden, wobei sie dauerhaft in der Datenbank gespeichert werden. MongoDB lässt aufgrund des flexiblen Datenschemas das Verändern von Datentypen zu. Dennoch ermöglicht MongoDB mit ‚readConcern' und ‚writeConcern' einen konsistenten Datenzustand, indem nur mit den erfolgreichen Transaktionen, die von den anderen Replica Set Nodes bestätigt wurden, gearbeitet wird. Eine Transaktion innerhalb einer Session kann ohne einen ‚Commit' gelesen werden, jedoch kann sie nicht von anderen Mitgliedern im Replica Set gesehen werden. Diese nicht festgeschriebenen Schreibvorgänge werden nicht auf die sekundären Server repliziert. Erst wenn eine Transaktion mit ‚Commit' festgeschrieben wird, wird sie repliziert und atomar an alle sekundären Server weitergegeben. Dies sorgt für einen aktuellen konsistenten Zustand, da nur der primäre Server die Schreiboperation an die sekundären Server weitergeben kann. Zudem wird für eine gleiche Datenkonsistenz zwischen dem primären Server und dem sekundären Server gesorgt. Dadurch kann ein sekundärer Server als neuer primäre Server ausgewählt werden, falls der primäre Server ausfällt. MariaDB nutzt als Standard-Storage-Engine die InnoDB für Transaktionen, die durch Isolationstechniken die Transaktionen voneinander isoliert und die Zeilen, Tabellen und Indexe in der Datenbank für andere Nutzende sperrt. Mittels ‚Logging'-Mechanismen sorgt MariaDB dafür, dass sämtliche Änderungen dokumentiert werden. Im Falle eines Systemausfalls können mithilfe von ‚Audit Trail', Back-up und ‚Binary Transaction Log File' die Daten wiederhergestellt werden und einen konsistenten Datenzustand sicherstellen [57]. Mit der Methode ‚Begin Transaction' wird in CockroachDB für eine Transaktion ein Transaktionseintrag erstellt. Jeder Transaktionseintrag besitzt einen eindeutigen Schlüssel und den Zustand der Transaktion. Alle Schreibvorgänge einer Transaktion werden dauerhaft in den gleichen Bereich mit dem eindeutigen Schlüssel gespeichert. CockroachDB verwendet für die Isolation der Transaktionen Variationen von Multi-Version Concurrency Control (MVCC) und den Standard-Clock-Synchronisation-Mechanismus, wie Network Time Protocol (NTP) in der transaktionalen Schicht [39, S. 1494,1496]. Als Standard für die Isolation verwendet CockroachDB den ‚Serializable Snapshot', der die Daten wiederherstellt und eine verteilte, serialisierbare Transaktion ohne Sperre ermöglicht [58].

Tabelle 4. 5: Zusammenfassung der Transaktionsergebnisse

Datenbank	**MariaDB**	**MongoDB**	**CockroachDB**
Eigenschaft	ACID	ACID	ACID
Syntax	START TRANSACTION; <transaction Statements> COMMIT;	session.startTransaction() <transaction Statements> session.commitTransaction()	BEGIN; <transaction Statements> COMMIT;

4.4 Nichtfunktionaler Test

In diesem Abschnitt wird die Leistung der Systeme MariaDB, MongoDB und CockroachDB anhand verschiedener Anwendungsszenarien gemessen und ausgewertet.

4.4.1 Testmetrik und Versuchsaufbau

Beim nichtfunktionalen Test wurden sieben strukturierte CSV-Dateien von ‚Mitarbeiter', die eine vordefinierte Struktur haben, in die Systeme eingefügt. Die sieben CSV-Dateien bestehen aus den folgenden Datenmengen:

- 1
- 10
- 100
- 1000
- 10 000
- 100 000
- 1 000 000

Jeder Datensatz in der ‚Mitarbeiter'-CSV-Datei besteht aus den Attributen ‚Vorname', ‚Nachname', ‚Berufserfahrung', ‚Gehalt' und ‚Abteilungsname'. Alle CSV-Dateien zusammen entsprechen 31 520 KB. Die folgende Tabelle 4.6 listet die Versionen der Datenbanken und die benötigten Python-Module, um aus PyCharm mit den Datenbanken zu interagieren, auf.

Tabelle 4. 6: Datenbanken und Python-Module

Datenbanken	***MariaDB***	***MongoDB***	***CockroachDB***
Datenbankversion	10.6	5.0.3	21.2.9
Python Module	Mariadb	Pymongo	Psycopg2

Fairness beim Einfügen der Daten

Beim Einfügen der Daten in die verschiedenen Datenbanksysteme ist die Struktur der Python Scripts gleich. Für CockroachDB und MariaDB werden gleiche Methoden verwendet. Bei MongoDB werden ähnliche verfügbare Methoden für den Vergleich genutzt. Das Einfügen und Anwenden von Abfragen für die jeweiligen Testszenarien wurde dreimal wiederholt. Dieser Vorgang diente zur Konsistenzprüfung der Ergebnisse. Vor dem Einfügen der Daten wurden die Tabellen/Collections gelöscht und neu erstellt, um Duplikate oder eine falsche Anzahl von Datensätzen beim Einfügen zu vermeiden.

Fairness bei der ‚Join'-Abfrage

MongoDB hat keine ‚Join'-Funktion. Damit der Vergleich zwischen den Datenbanken fair bleibt, wurde die Funktion ‚Reference' benutzt. ‚Reference' eignet sich für die ‚Many-to-Many'-Beziehung. Die Dokumente werden separat gehalten. Nur ein Dokument enthält eine Referenz auf ein anderes Dokument. Durch diesen Verweis können die Informationen aus beiden Dokumenten abgerufen werden [59]. Dafür mussten sämtliche eingefügte Daten in MongoDB verändert werden, indem ein neues Feld ‚Projekt_Name' hinzugefügt wurde. Dadurch konnte MongoDB mithilfe der ‚$lookup'-Methode die ‚Projekt'-, ‚Mitarbeiter'- und ‚Abteilung'-Collection miteinander verknüpfen, indem die referenzierten Felder in ein neues Dokument eingebettet wurden.

4.3.2 Durchführung der Performanceanalyse

Um die Performance der drei Datenbanken mit unterschiedlichen Testszenarien zu prüfen, wurden die drei Entitäten ‚Mitarbeiter', ‚Abteilung' und ‚Projekt' benötigt. Die Entitäten ‚Projekt' und ‚Abteilung' dienten den ‚Join'-Abfragen, um die Leistung beim Verknüpfen von Entitäten mit verschiedenen Beziehungsarten zu messen.

Mehrere Mitarbeiter können in einer Abteilung arbeiten und an mehreren Projekten teilnehmen. Zwischen den Entitäten ‚Abteilung' und ‚Mitarbeiter' herrscht eine ‚1:N'-Beziehung. Hierfür besitzt die Entität ‚Mitarbeiter' den Primärschlüssel der Entität ‚Abteilung' als Fremdschlüssel. Mit dem Fremdschlüssel können die Entitäten ‚Mitarbeiter' und ‚Abteilung' miteinander verknüpft werden und an die Informationen aus beiden Entitäten zugreifen. Zwischen den Entitäten ‚Mitarbeiter' und ‚Projekt' herrscht eine ‚M:N'-Beziehung. Für die ‚M:N'-Beziehung wurde eine weitere Tabelle ‚beteiligt' benötigt, die die Primärschlüssel von ‚Projekt' und ‚Mitarbeiter' als Primär- und Fremdschlüssel besitzt (siehe Abbildung 4.1).

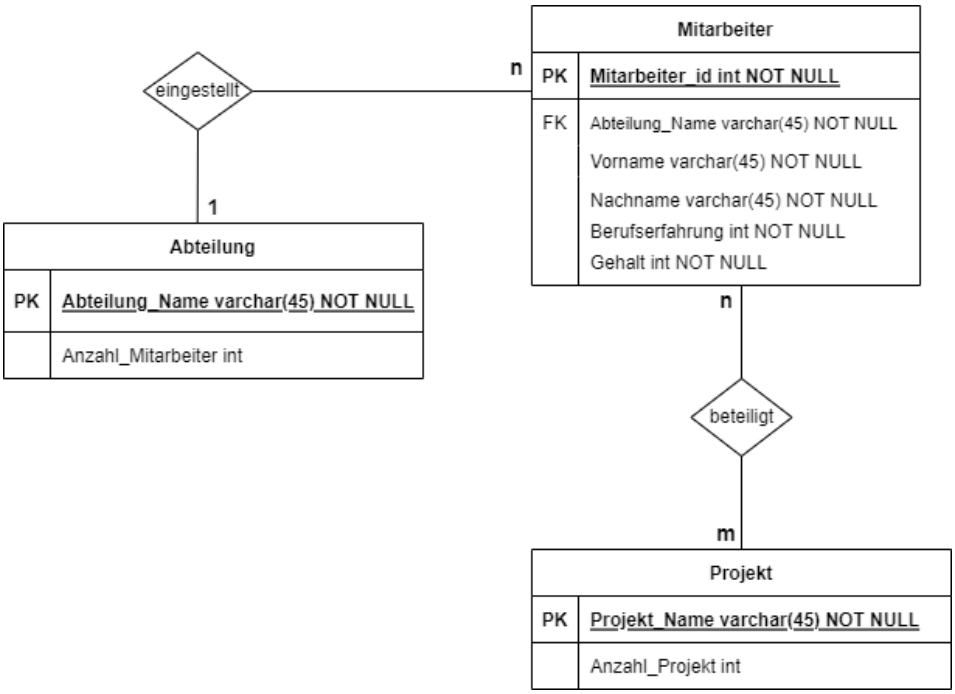

Abbildung 4. 1: Entity-Relationship-Modell von ‚Mitarbeiter', ‚Abteilung' und ‚Projekt'

‚Create' und ‚Insert'

In den relationalen Datenbanken MariaDB und CockroachDB wurden die Entitäten als Tabellen erzeugt, während in MongoDB die Entitäten als Collections erstellt wurden. Mit dem ‚Insert'-Befehl wurden die Collections und Tabellen mit Daten befüllt. Die sieben CSV-Dateien von ‚Mitarbeiter' wurden nacheinander in die Systeme eingefügt und die verschiedenen Abfragen der Testszenarien wurden angewendet. Dabei wurde die Anzahl der Mitarbeiter in ‚Abteilung' stetig an die Anzahl der eingefügten Daten in ‚Mitarbeiter' angepasst.

Listing 4. 17: Ein Beispieldokument aus der ‚Mitarbeiter'-Collection in MongoDB

```
{  "Vorname":"Keven",
   "Nachname":"Kohi",
    "Berufserfahrung":5,
    "Gehalt":1500,
    "Abteilung_Name":"Informatik",
    "Projekt_Name": "IT"}
```

‚Select'

Als Testszenarien für die Leseabfrage wurden drei ‚Select'-Abfragen auf die Tabelle ‚Mitarbeiter' angewendet:

1. Sämtliche Datensätze werden ausgegeben.
2. Nur die Datensätze, die ‚Keven' als Vornamen haben, werden ausgegeben.
3. Nur die Datensätze, die ‚Keven' als Vornamen haben, werden mithilfe des Index gesucht und ausgegeben.

CockroachDB und MariaDB bieten für die Suche nach sämtlichen Datensätzen das ‚Select'-Statement an, während MongoDB die Methode ‚find()' unterstützt (siehe Listing 4.18). Hierbei wurde die Ausführungszeit gemessen, d. h., wie lange die Datenbanksysteme benötigen, um sämtliche Daten auszugeben.

Listing 4. 18: Eine SQL-Leseabfrage sämtlicher Einträge aus der Tabelle ‚Mitarbeiter'

```
SELECT * FROM Mitarbeiter;
```

Im zweiten Testszenario wurde mittels der ‚Where Clause' nach einem spezifischen Datensatz in der Datenbank gesucht (siehe Listing 4.19).

Listing 4. 19: Das Suchen nach spezifischen Einträgen, die den Vornamen ‚Keven' enthalten

```
SELECT * FROM Mitarbeiter WHERE Mitarbeiter.Vorname ='Keven';
```

Das dritte Testszenario umfasste die Suche nach bestimmten Daten mithilfe eines nicht eindeutigen Index. Dabei wurde untersucht, inwiefern das Erstellen eines Index einen Einfluss auf die Suche nach einen Datensatz hat, und diese Suche optimiert. Der Index agiert als ein Pointer, der schnell festlegen kann, welche Datensätze den Bedingungen der ‚Where Clause' entsprechen. Diese Daten werden herausgefiltert und ausgegeben. Alle drei Datenbanken unterstützen die Index-Funktion. Während MongoDB einen Index für das Feld ‚Vorname' erstellte, wurde in den relationalen Datenbanken ein Index für die Spalte ‚Vorname' generiert (siehe Listing 4.20).

Listing 4. 20: Das Erstellen eines Index in MariaDB

```
CREATE INDEX Vorname_INDEX ON Mitarbeiter(Vorname);
```

‚Join'-Operation

Als weiteres Testszenario für die Leseabfrage wurde die ‚Join'-Funktion geprüft. Die relationalen Datenbanken CockroachDB und MariaDB unterstützen die ‚Join'-Funktion. Mithilfe der Primär- und der Fremdschlüssel konnte auf sämtliche Informationen aus den verknüpften Tabellen zugegriffen werden. Bei der Überprüfung der Leistung der ‚Join'-Operation wurden zwei Testszenarien geprüft:

1. die ‚1:N'-Beziehung, die zwei Tabellen (‚Abteilung' und ‚Mitarbeiter') verknüpft, und
2. die ‚M:N'-Beziehung, die drei Tabellen (‚beteiligt', ‚Projekt' und ‚Mitarbeiter') verknüpft.

Listing 4. 21: Eine SQL-Abfrage mit einer ‚1:N'-Beziehung

```
SELECT Mitarbeiter.Mitarbeiter_ID, Mitarbeiter.Berufserfahrung, Mitarbei-
    ter.Gehalt, Mitarbeiter.Vorname, Mitarbeiter.Nachname, Mitarbei-
 ter.Abt_Name, Abteilung.Anzahl_Mitarbeiter FROM Mitarbeiter JOIN Abtei-
        lung ON Mitarbeiter.Abt_Name = Abteilung.Abteilung_Name;
```

Mithilfe der ‚Join'-Funktion konnten die Tabellen in MariaDB und CockroachDB verbunden werden. Dies geschah bei der ‚1:N'-Beziehung über den Fremdschlüssel ‚Abt_Name' in der Tabelle ‚Mitarbeiter' mit dem Primärschlüssel ‚Abteilung_Name' in der Tabelle ‚Abteilung' (siehe Listing 4.21). Bei der ‚M:N'-Beziehung wurden zwei ‚Join'-Operationen benötigt, um die drei Tabellen ‚Projekt', ‚Mitarbeiter' und ‚beteiligt' miteinander zu verbinden. Die Tabelle ‚Mitarbeiter' konnte mit der Tabelle ‚beteiligt' über den Fremdschlüssel ‚Mitarbeiter_ID' verbunden werden. Mit einer weiteren ‚Join'-Funktion konnte die Tabelle ‚beteiligt' über den Fremdschlüssel ‚Projekt_Name' auf die Tabelle ‚Projekt' zugreifen und mit ihr verknüpft werden (siehe Listing 4.22).

Listing 4. 22: Eine SQL-Abfrage mit einer ‚M:N'-Beziehung

```
 SELECT Beteiligt.Mitarbeiter_ID, Mitarbeiter.Berufserfahrung, Mitarbei-
    ter.Gehalt, Mitarbeiter.Vorname, Mitarbeiter.Nachname, Mitarbei-
 ter.Abt_Name, Beteiligt.Projekt_Name,Projekt.Anzahl_Projekt FROM Mitar-
beiter JOIN Beteiligt ON Mitarbeiter.Mitarbeiter_ID = Beteiligt.Mitarbei-
   ter_ID JOIN Projekt ON Beteiligt.Projekt_Name= Projekt.Projekt_Name;
```

MongoDB unterstützt die ‚Join'-Funktion eines relationalen Datenbanksystems nicht; stattdessen unterstützt MongoDB ‚Embedding' oder ‚Reference' zwischen Collections. Für das Verknüpfen der Collection wurde Referenz (‚Reference') verwendet, wobei beim Leistungsvergleich der Fokus auf die Anzahl der Collections gelegt wurde. Da in den relationalen Datenbanken eine ‚1:N'-Beziehung zwei Tabellen umfasst, wurden in MongoDB mithilfe der Aggregate-Methode ‚$lookup' zwei Collections eingebettet (siehe Listing 4.23). Die Methode ‚$lookup' suchte nach den referenzierten Feldern in den Collections, die sich in einem Cluster befinden, und bettete diese referenzierten Dokumente als ein Array in die Dokumente der Collection ‚Mitarbeiter' ein.

Listing 4. 23: Einbettung von zwei Collections

```
db.get_collection('Mitarbeiter').aggregate([{"${"$lookup":
{   "from" : "Abteilung",
    "localField" : "Abt_Name",
    "foreignField" : "Abteilung_Name",
    "as" : "Abteilungen"}}])
```

In einer ‚M:N'-Beziehung wurden drei Collections mithilfe der ‚$lookup'-Methode eingebettet (siehe Abbildung 4.2).

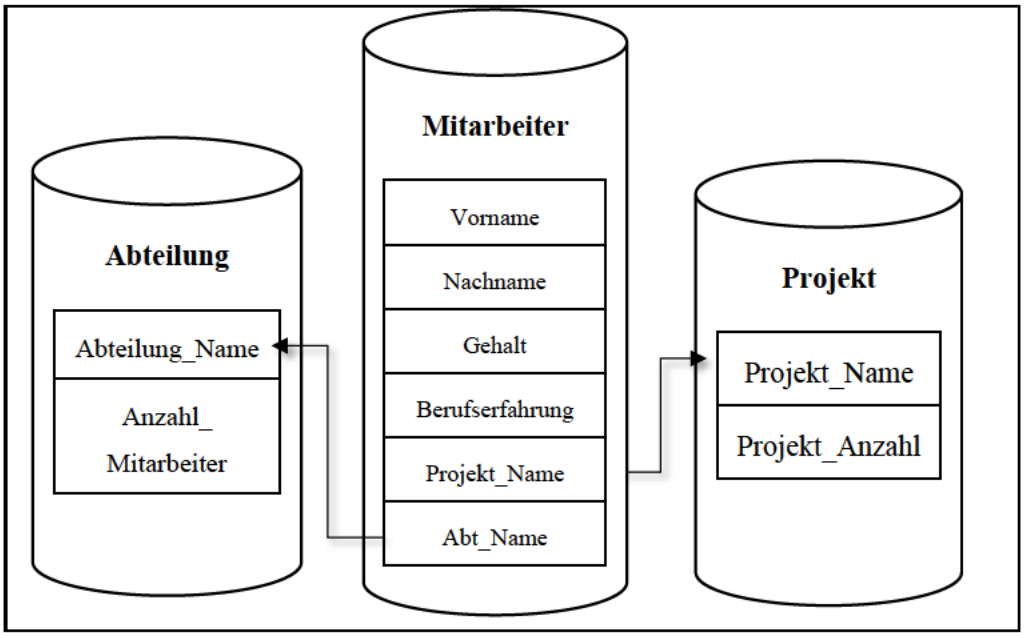

Abbildung 4. 2: Einbettung von drei Collections

‚Update'

Die Testszenarien beim Leistungsvergleich von Änderungsabfragen umfassten Folgendes:

- das Ändern bestimmter Datensätze, die den Vornamen ‚Keven' enthalten,
- das Ändern sämtlicher Datensätze.

Beim Ändern bestimmter Datensätze wurden sämtliche Daten durchsucht, die für ‚Berufserfahrung' einen Wert größer oder gleich 4 enthielten. Daraufhin wurden nur die zutreffenden Datensätze verändert. Dabei wurde das Gehalt um 50 und die Berufserfahrung um 1 erhöht. MariaDB und CockroachDB bieten hierfür die ‚Update'-SQL-Abfrage an (siehe Listing 4.24). MongoDB bietet hierfür die Methode ‚update_many' an, in der die Bedingung festgelegt wird, welche Daten verändert werden sollen.

Listing 4. 24: ‚Update'-Abfrage auf alle Datensätze, die Berufserfahrung größer oder gleich 4 haben

```
UPDATE Mitarbeiter SET Berufserfahrung =(Berufserfahrung + 1),
       Gehalt = ( gehalt + 50) WHERE Berufserfahrung>=4;
```

Beim Ändern sämtlicher Datensätze wurde das Gehalt um 50 und die Berufserfahrung um 1 erhöht. Als Bedingung wurde die Berufserfahrung größer oder gleich 0 übergeben, damit sämtliche Datensätze verändert werden (siehe Listing 4.25).

Listing 4. 25: ‚Update'-Abfrage auf sämtliche Daten in der Tabelle ‚Mitarbeiter'

```
UPDATE Mitarbeiter SET Berufserfahrung =(Berufserfahrung + 1),
       Gehalt = ( gehalt + 50) WHERE Berufserfahrung>=0;
```

‚Delete'

Beim Leistungsvergleich hinsichtlich des Löschens der Datensätze wurden zwei Testszenarien verglichen:

- das Löschen bestimmter Daten, die den Vornamen ‚Keven' haben,
- das Löschen sämtlicher Daten.

Mit dem SQL-Statement ‚DELETE' in MariaDB und CockroachDB wurden die Datensätze gelöscht, die ‚Keven' als Vorname enthielten (siehe Listing 4.26).

Listing 4. 26: Das Löschen der Datensätze mit dem Vornamen ‚Keven'

```
DELETE FROM Mitarbeiter WHERE Mitarbeiter.Vorname ='Keven';
```

MongoDB bietet die Funktion ‚delete_one' an, in der ein Filter als Schlüssel-Wert-Paar übergeben wird, um bestimmte Daten aus der Datenbank zu löschen (siehe Listing 4.27).

Listing 4. 27: Das Löschen der Datensätze mit dem Vornamen ‚Keven' in MongoDB

```
db.get_collection('Mitarbeiter').delete_one({„Vorname" : „Keven"})
```

Im Gegenzug wurden alle Datensätze in MariaDB und CockroachDB nur mit dem Befehl ‚DELETE' gelöscht (siehe Listing 4.28).

Listing 4. 28: Das Löschen sämtlicher Datensätze in MariaDB und CockroachDB

```
DELETE FROM Mitarbeiter;
```

MongoDB bietet die Funktion ‚delete_many' an, um sämtliche Daten aus der Datenbank zu löschen. Dabei wird kein Filter in der Funktion übergeben (siehe Listing 4.29).

Listing 4. 29: Das Löschen sämtlicher Datensätze in MongoDB

```
db.get_collection('Mitarbeiter').delete_many({})
```

4.3.3 Auswertung der Ergebnisse

Dieser Abschnitt enthält die experimentellen Ergebnisse der Performanceanalyse.

Ergebnis der ‚Insert'-Abfrage

Beim Einfügen der Daten erzielte MariaDB eine bessere Leistung als MongoDB und CockroachDB. MariaDB benötigte im Durchschnitt ~98,59 Sekunden weniger Zeit als MongoDB, die im Durchschnitt ~407,51 Sekunden mehr benötigte. CockroachDB benötigte mit ~1038,12 Sekunden mehr Zeit als die anderen zwei Datenbanken.

Tabelle 4. 7: ‚Insert'-Abfrage auf die Tabelle ‚Mitarbeiter'

Antwortzeit (in Sekunden)

Einträge	MariaDB	MongoDB	CockroachDB
1	0,02	0.004	0,01
10	0,03	0,05	0,11
100	0,11	0,83	0,46
1000	0,81	5,94	10,97
10 000	11,52	16,01	42,24
100 000	128,80	209,71	569,85
1 000 000	548,87	2620,00	6643,18
Durchschnitt	~ 98,59	~ 407,51	~ 1038,12

Ergebnis der ‚Select'-Abfrage

Bei der Ausgabe sämtlicher Einträge der Tabelle ‚Mitarbeiter' benötigte MariaDB im Durchschnitt nur ~3,30 Sekunden. Im Vergleich hierzu benötigten MongoDB und CockroachDB über ~5 Sekunden mehr Zeit, um die Einträge auszugeben. CockroachDB benötigte für 1 bis 100 000 Datensätze weniger Zeit als MongoDB. Jedoch brauchte MongoDB für 1 000 000 Datensätze 2 Sekunden weniger Zeit als CockroachDB.

Tabelle 4. 8: Ausgabe sämtlicher Einträge der Tabelle ‚Mitarbeiter' mit dem ‚Select'-Befehl

Antwortzeit (in Sekunden)			
Einträge	MariaDB	MongoDB	CockroachDB
1	0,001	0,04	0,005
10	0,001	0,04	0,002
100	0,02	0,06	0,01

1000	0,05	0,09	0,06
10 000	0,45	0,42	0,24
100 000	4,37	3,42	2,03
1 000 000	18,19	32,09	34,97
Durchschnitt	~ 3,30	~ 5,17	~ 5,33

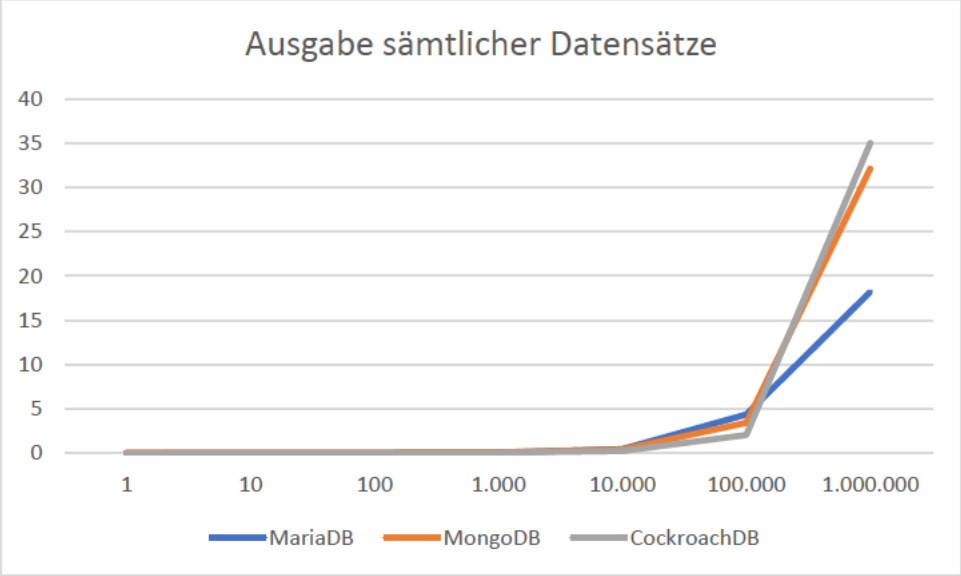

Abbildung 4. 3: Suche nach sämtlichen Datensätzen

Bei der Suche nach sämtlichen Daten, die ‚Keven' als Vorname enthalten, benötigte MariaDB mit ~0,39 Sekunden weniger Zeit als CockroachDB und MongoDB. Obwohl CockroachDB für 1 bis 100 000 Einträge weniger Zeit als MongoDB benötigte, brauchte MongoDB für 1 000 000 Einträge weniger Zeit als CockroachDB.

Tabelle 4. 9: ‚Select'-Abfrage nach Datensätzen mit dem Vornamen ‚Keven'

Antwortzeit (in Sekunden)			
Einträge	MariaDB	MongoDB	CockroachDB

1	0,001	0,002	0,006
10	0,002	0,07	0,005
100	0,001	0,07	0,008
1000	0,008	0,08	0,02
10 000	0,07	0,07	0,05
100 000	0,18	0,36	0,28
1 000 000	2,47	4,05	4,58
Durchschnitt	~ 0,39	~ 0,67	~ 0,71

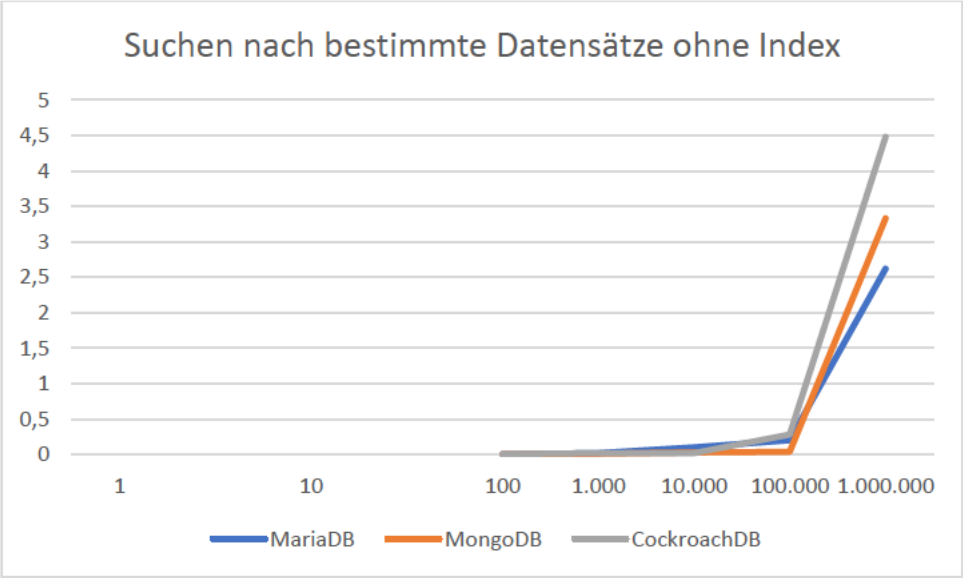

Abbildung 4. 4: Suche nach Datensätzen mit dem Vornamen ‚Keven' ohne Index

Bei der Suche nach dem Vornamen ‚Keven' mithilfe des Index benötigten MongoDB und MariaDB im Durchschnitt gleich viel Zeit. Besonders bei 10 bis 100 000 Einträgen lieferte MongoDB eine bessere Leistung bei der Suche nach den Datensätzen als MariaDB. Das Verwenden eines Index optimierte die Leistung in MongoDB um ~0,18 Sekunden, wodurch die Zeit für das Suchen der Einträge reduziert wurde. Die Leistung von CockroachDB wurden mit dem

Index bei 1000 bis 10 000 Einträgen verbessert. Jedoch nahm die Optimierung der Suche bei größeren Datensätzen ab. MariaDB wies mit und ohne Index eine höhere Geschwindigkeit als MongoDB und CockroachDB auf. Jedoch verschlechterte sich die Leistung von MariaDB im Vergleich ohne einen Index im Durchschnitt um ~0,03 Sekunden.

Tabelle 4. 10: ‚Select'-Abfrage nach Datensätzen mit Vornamen ‚Keven' mittels Index

Antwortzeit (in Sekunden)			
Einträge	MariaDB	MongoDB	CockroachDB
1	0,0001	0,001	0,001
10	0,002	0,001	0,004
100	0,003	0,002	0,005
1000	0,02	0,01	0,02
10 000	0,10	0,03	0,02
100 000	0,20	0,04	0,28
1 000 000	2,62	3,33	4,48
Durchschnitt	~ 0,42	~ 0,49	~ 0,69

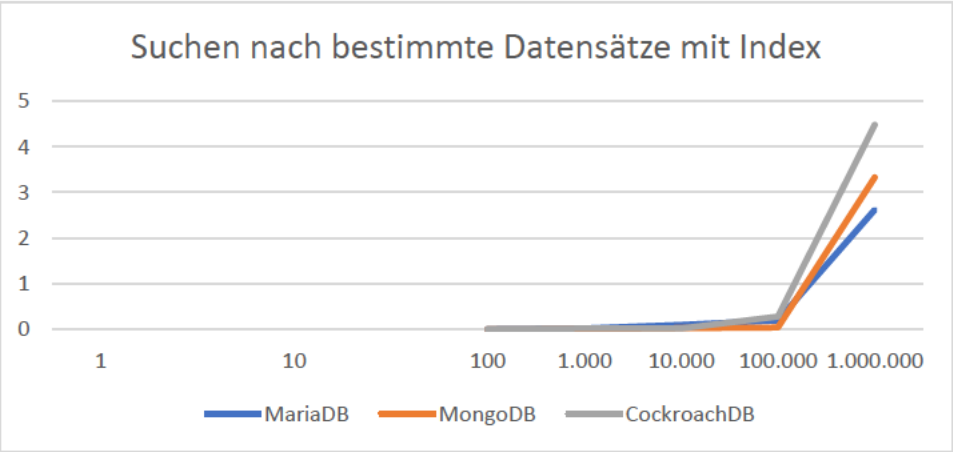

Abbildung 4. 5: Suche nach bestimmten Datensätzen mit dem Vornamen ‚Keven' mittels Index

Ergebnis der ‚Join'-Abfrage

MongoDB schnitt beim Verknüpfen von zwei Tabellen mit durchschnittlich ~20 Sekunden am schlechtesten ab im Vergleich zu den relationalen Datenbanken. Dieses Ergebnis verdeutlicht, dass die relationalen Datenbanken bei den Beziehungen zwischen den Entitäten eine bessere Leistung liefern als ein NoSQL-Datenbanksystem. MariaDB benötigte für die Datensätze mit 1 bis 100 Einträgen weniger Zeit als CockroachDB. Hingegen lieferte CockroachDB bei Datensätzen mit 1000 bis 100 000 Einträgen eine bessere Leistung als die anderen zwei Datenbanken. Bei 1 000 000 Einträgen lieferte MariaDB mit ~19 Sekunden die bessere Leistung.

Tabelle 4. 11: ‚Join'-Abfrage mit einer Relation

Antwortzeit (in Sekunden)			
Einträge	MariaDB	MongoDB	CockroachDB
1	0,001	0,07	0.003
10	0,005	0,04	0,01
100	0,01	0,08	0,02
1000	0,08	0,37	0,08
10 000	0,35	1,20	0,20
100 000	3,60	13,85	1,97
1 000 000	19,43	131,16	34,94
Durchschnitt	~ 3,35	~ 20,97	~ 5,32

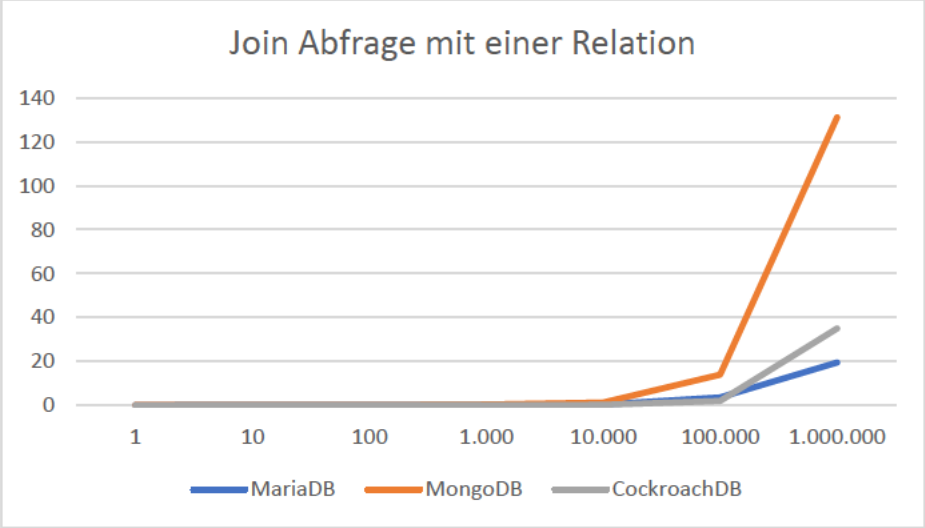

Abbildung 4. 6: ‚Join'-Abfrage mit einer Relation

Bei der Verknüpfung von drei Tabellen benötigte MariaDB mit durchschnittlich ~3,63 Sekunden weniger Zeit als CockroachDB mit ~6,12 Sekunden. MongoDB schnitt mit ~41,07 Sekunden am schlechtesten ab. Bei Datensätzen mit 100 000 Einträgen benötigte CockroachDB mit ~0,28 Sekunden weniger Zeit als MariaDB, jedoch lieferte MariaDB bei den anderen Datensätzen eine bessere Leistung.

Tabelle 4. 12: ‚Join'-Abfrage mit zwei Relationen

Antwortzeit (in Sekunden)			
Einträge	MariaDB	MongoDB	CockroachDB
1	0,001	0,07	0,004
10	0,001	0,06	0,01
100	0,02	0,21	0,02
1000	0,10	0,75	0,07
10 000	0,27	2,19	0,27
100 000	4,03	25,02	0,28

1 000 000	20,97	259,19	41,37
Durchschnitt	~ 3,63	~ 41,07	~ 6,12

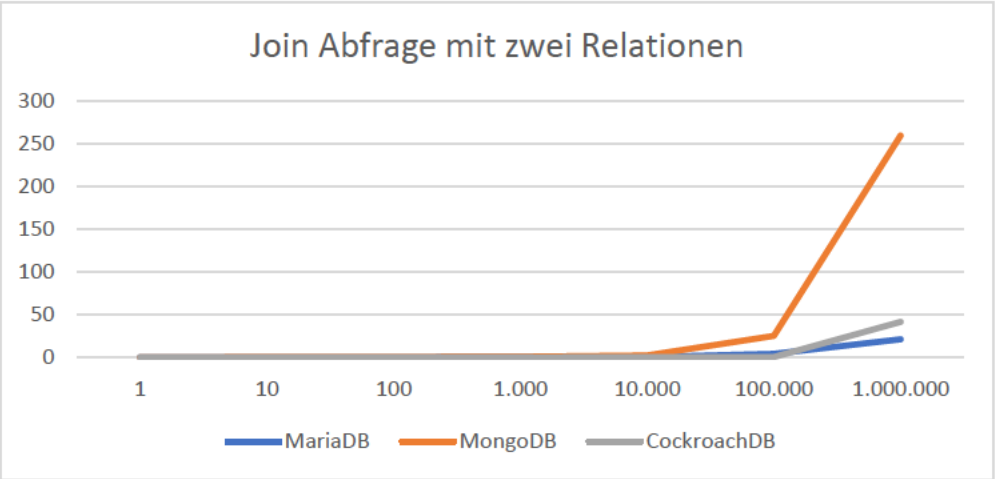

Abbildung 4. 7: ‚Join'-Abfrage mit zwei Relationen

Ergebnis der ‚Update'-Abfrage

Beim Verändern von Daten mit dem Vornamen ‚Keven' benötigte MariaDB mit ~0,37 Sekunden im Durchschnitt weniger Zeit als MongoDB und CockroachDB. MongoDB folgte nach MariaDB mit ~3,01 Sekunden, während CockroachDB ~6,68 Sekunden benötigte. Verglichen mit den anderen Datenbanken hat MariaDB bei sämtlichen Einträgen eine sehr gute Leistung mit wenig Zeit geliefert.

Tabelle 4. 13: ‚Update'-Abfrage auf Daten mit dem Vornamen ‚Keven'

Antwortzeit (in Sekunden)			
Einträge	MariaDB	MongoDB	CockroachDB
1	0,002	0,001	0,04
10	0,0007	0,001	0,006
100	0,001	0,002	0,02
1000	0,004	0,01	0,02

10 000	0,03	0,14	0,06
100 000	0,75	1,70	0,78
1 000 000	1,78	19,18	45,80
Durchschnitt	~ 0,37	~ 3,01	~ 6,68

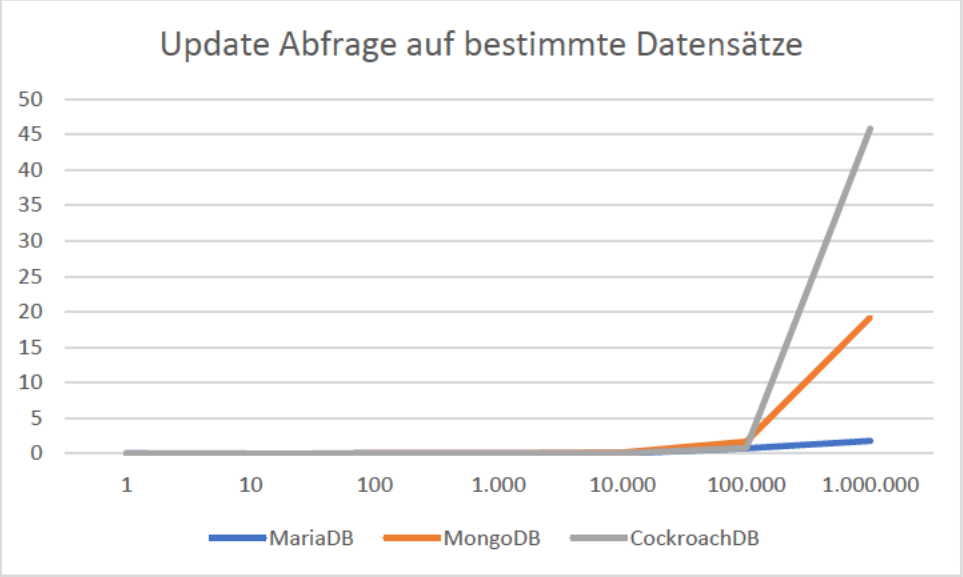

Abbildung 4. 8: Das Verändern von Daten mit dem Vornamen ‚Keven'

Beim Verändern sämtlicher Datensätze in der Tabelle ‚Mitarbeiter' benötigte CockroachDB mit ~6 Sekunden im Durchschnitt mehr Zeit als MariaDB mit ~0,42 Sekunden und MongoDB mit ~3,86 Sekunden. Besonders bei Datensätzen mit 1 000 000 Einträgen lieferte MariaDB mit nur ~2 Sekunden eine gute Performance, während MongoDB ~25 Sekunden und CockroachDB ~40 Sekunden Zeit benötigte.

Tabelle 4. 14: ‚Update'-Abfrage auf sämtliche Datensätze

Antwortzeit (in Sekunden)			
Einträge	MariaDB	MongoDB	CockroachDB

1	0,001	0,002	0,004
10	0,001	0,003	0,007
100	0,001	0,003	0,02
1000	0,004	0,02	0,08
10 000	0,04	0,22	0,18
100 000	0,65	2,21	1,30
1 000 000	2,22	24,54	40,38
Durchschnitt	~ 0,42	~ 3,86	~ 6,00

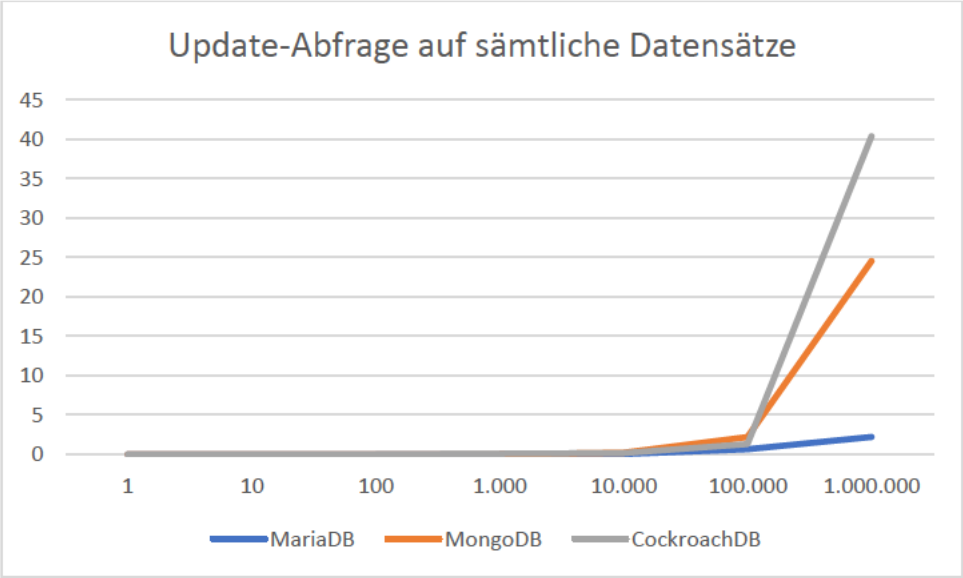

Abbildung 4. 9: Änderung sämtlicher Datensätze

Ergebnis der ‚Delete'-Abfrage

Für das Löschen von Datensätzen mit dem Vornamen ‚Keven' beanspruchte MongoDB mit ~0,001 Sekunden im Durchschnitt weniger Zeit als MariaDB mit ~0,27 Sekunden. Eine im Vergleich mangelhafte Leistung lieferte CockroachDB mit ~245,86 Sekunden. MariaDB und

MongoDB gaben bei Datensätzen mit bis zu 100 Einträgen die gleiche Performance ab. Ab 10 000 Einträgen holte MongoDB MariaDB mit ihrer gleichbleibenden Leistung auf.

Tabelle 4. 15: Löschung von Datensätzen mit dem Vornamen ‚Keven'

Antwortzeit (in Sekunden)			
Einträge	MariaDB	MongoDB	CockroachDB
1	0,001	0,001	0,02
10	0,001	0,001	0,01
100	0,001	0,002	0,01
1000	0,002	0,001	0,03
10 000	0,02	0,001	0,19
100 000	0,26	0,001	0,21
1 000 000	1,62	0,0005	1720,57
Durchschnitt	~ 0,27	~ 0,001	~ 245,86

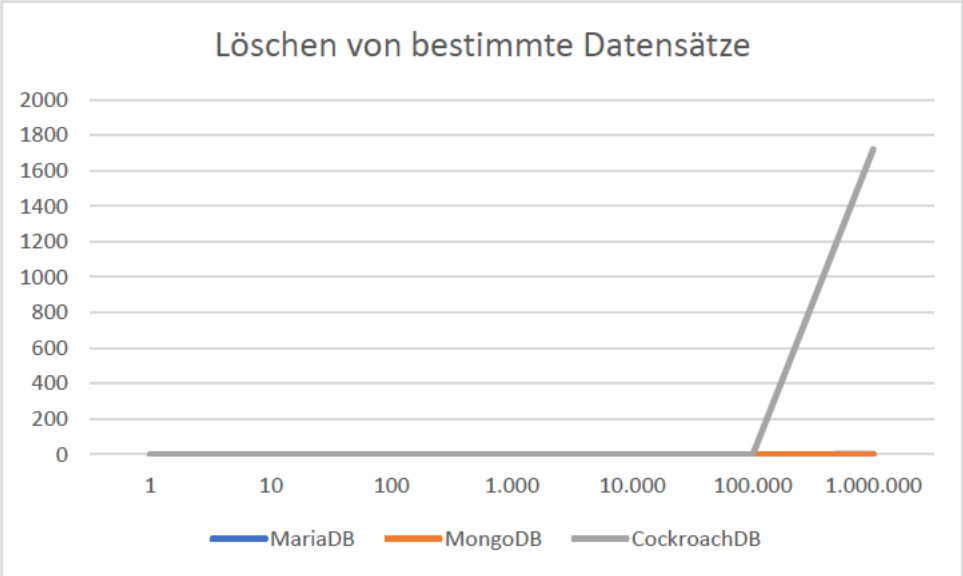

Abbildung 4. 10: Löschung von Datensätzen mit dem Vornamen ‚Keven'

Für das Löschen sämtlicher Daten benötigte MariaDB mit durchschnittlich ~0,89 Sekunden weniger Zeit als MongoDB und CockroachDB. CockroachDB benötigte hierfür mit ~24,2 Sekunden deutlich weniger Zeit als für das Löschen bestimmter Datensätze, die ‚Keven' als Vornamen enthielten. MongoDB benötigte mit ~2,72 Sekunden deutlich mehr Zeit im Vergleich zum Löschen bestimmter Datensätze.

Tabelle 4. 16: Löschung sämtlicher Datensätze

Antwortzeit (in Sekunden)			
Einträge	MariaDB	MongoDB	CockroachDB
1	0,0008	0,001	0,03
10	0,001	0,001	0,03
100	0,001	0,004	0,02
1000	0,003	0,02	0,03
10 000	0,18	0,13	0,26
100 000	1,20	1,86	3,98
1 000 000	4,84	17,04	165,56
Durchschnitt	~ 0,89	~ 2,72	~ 24,2

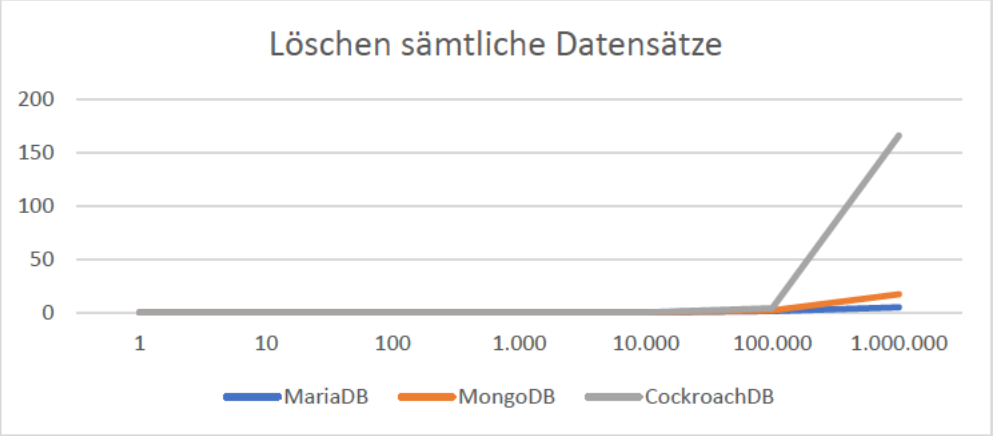

Abbildung 4. 11: Löschung sämtlicher Datensätze

4.3.4 Zusammenfassung der Ergebnisse

Im nichtfunktionalen Test lieferte MariaDB beim Ausführen der einzelnen Testszenarien eine bessere Leistung als MongoDB und CockroachDB. Das Erstellen der Lese- und Schreibabfragen in MariaDB ist einfach und die Geschwindigkeit für das Ausführen der Abfragen schnell. Der Grund hierfür liegt in der Architektur von MariaDB, die auf einer modularen Storage-Architektur beruht. Da Storage Engines einen Einfluss auf die Performance von Datenbanksystemen haben und MariaDB für jeden Anwendungsfall eine passende Storage Engine besitzt, die für jedes Szenario eine optimale Strategie anbietet, lieferte MariaDB bei sämtlichen Testszenarien eine vergleichsweise gute Leistung ab. Im Gegensatz zu den Ergebnissen aus Hairahs und Budimans [20] Untersuchung benötigte MariaDB ab dem Einfügen von 100 000 Datensätzen mehr Zeit als PostgreSQL. Beim Prüfen von einer, zwei und drei inneren ‚Join'-Relationen benötigte MariaDB mehr Zeit als PostgreSQL. Dessen ungeachtet lieferte MariaDB im Vergleich zu MongoDB und CockroachDB eine bessere Leistung bei der ‚Join'-Operation auf eine und zwei Relationen. CockroachDB brachte im nichtfunktionalen Test eine ungenügende Leistung. Die Ursache hierfür liegt an der Shared-Nothing-Architektur, die für die globalen OLTP-Workloads und eine wachsende Benutzeranzahl in einer verteilten Umgebung geeignet ist. Obwohl CockroachDB bei der ‚Join'-Abfrage schneller als MongoDB war, war CockroachDB im Vergleich zu MariaDB langsamer. Dies liegt daran, dass CockroachDB für komplexe SQL-Anweisungen wie ‚Join'-Operationen nicht geeignet und damit nicht die optimale Datenbank für umfangreiche Analysen oder Online Analytical Processing (OLAP) ist [60]. Trotz Optimierung der Suche mithilfe eines Index benötigte CockroachDB mehr Zeit als MariaDB und MongoDB. Auch in der wissenschaftlichen Arbeit von Sachdeva und Kaur [21] war CockroachDB in ‚Write', ‚Read' und ‚Update' langsamer als die anderen NewSQL-Systeme VoltDB, MemSQL und NuoDB. Nur bei der Ausführungszeit war CockroachDB schneller als MemSQL und VoltDB. Nach MariaDB benötigte MongoDB für sämtliche Testszenarien weniger Ausführungszeit, bis auf das Testszenario mit dem ‚Join'-Operator. Dies liegt daran, dass MariaDB und CockroachDB mehrere Optionen von ‚Join'-Strategien unterstützen. Zum Beispiel unterstützt MariaDB die Option ‚Block Hash Join' oder ‚Block-Based Join'[17] als eine

[17] [72]

‚Join'-Strategie, während CockroachDB den ‚Vectorized Merge Join'[18] als ‚Join'-Option unterstützt. MongoDBs ‚Join'-Operation basiert auf dem Iterieren sämtlicher Datensätze in einer Collection, bis die referenzierten Datensätze gefunden werden [61]. Zudem ist der Code für das Verschachteln mehrerer Dokumente in anderen Dokumenten komplexer als die SQL-Anweisungen von MariaDB und CockroachDB. Grund hierfür ist die Low-Level-Programmiersprache, die MongoDB unterstützt. Besonders bei komplexen ‚Join'-Abfragen müssen mehrere Aggregate-Methoden in MongoDB kombiniert werden, damit die Abfragen einer SQL-Abfrage gleichen. Außerdem muss für den Wechsel von einer ‚1:N'- zu einer ‚M:N'-Beziehung die Semantik der ‚Join'-Operation in MongoDB mit ‚$unwind' verändert werden [5, S. 3], [62]. Zudem wird über die ‚$lookup'-Methode mit aggregierten Daten gearbeitet, was für redundante Daten sorgt und damit nicht der klassischen ‚Join'-Operation in einer relationalen Datenbank entspricht. Dennoch benötigte MongoDB weniger Zeit beim Speichern großer Datenmengen und Ausführen der CRUD-Operationen als CockroachDB. In der Forschung von Boicea, Agapin und Radulescu [19] lieferte MongoDB auch beim Einfügen, Löschen und Verändern großer Datenmengen ab 100 Datensätzen eine bessere Leistung als die Oracle-Datenbank. Insbesondere in der wissenschaftlichen Arbeit von Deari, Zenuni, Ajdari, Ismaili und Raufi [14], in der MongoDB mit MySQL verglichen wurde, lieferte MongoDB beim Ausführen der einzelnen CRUD-Operationen eine bessere Leistung mit einer geringeren Ausführungszeit als MySQL. Auch bei der parallelen Ausführung der CRUD-Operationen fiel MySQL bei 10 000 Datensätzen aus und erbrachte verglichen zu MongoDB eine schlechte Leistung. Damit bewies MongoDB, dass es besonders bei der Verarbeitung großer Datenmengen mit parallelen Zugriffen und der gleichzeitigen Ausführung von CRUD-Operationen ein optimales System ist. Im Experiment wurden CockroachDB, MariaDB und MongoDB nicht auf parallele Zugriffe und die Ausführung paralleler CRUD-Operationen getestet. Deshalb muss eine Benchmark für die Theorie, dass MongoDB bei parallelen Zugriffen auf große Datenmengen eine bessere Leistung als MariaDB und CockroachDB liefert, durchgeführt werden. Zudem wurde die Leistungsanalyse zwischen MariaDB, MongoDB und CockroachDB mit einfachen festgelegten Datenschemata in einer nicht verteilten Umgebung mit nur einem Nutzenden durchgeführt. Die Ergebnisse könnten variieren, wenn die Analyse mit komplexen und sich dynamisch

[18] [73]

verändernden Datentypen und Datenstrukturen in einer verteilten Umgebung mit stetig wachsender Nutzeranzahl durchgeführt werden würde. Tabelle 4.17 enthält eine Zusammenfassung der Ergebnisse aus der nichtfunktionalen Analyse.

Tabelle 4. 17: Zusammenfassung der nichtfunktionalen Ergebnisse

Datenbank	**MariaDB**	**MongoDB**	**CockroachDB**
CRUD	CREATE, INSERT, UPDATE, DELETE	CREATE, INSERT, UPDATE, DELETE	CREATE, INSERT, UPDATE, DELETE
Query	SELECT … WHERE	FIND()	SELECT … WHERE
INDEX	Ja	Ja	Ja
JOIN	Ja	Nein, eingebettete Dokumente oder Referenz	Ja

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

In dieser Bachelorarbeit wurde ein Vergleich zwischen der SQL-Datenbank MariaDB, der NewSQL-Datenbank CockroachDB und der NoSQL-Datenbank MongoDB durchgeführt. Dafür wurden die Systeme anhand unterschiedlicher Anwendungsfälle getestet und deren Leistung mithilfe der Ausführungszeit verglichen. Für die Anwendungsfälle wurden unterschiedliche Vergleichskriterien und Testszenarien konzipiert, um die üblichen Funktionen traditioneller Datenbanken abzudecken. Diese Funktionen wurden in drei Analysen aufgeteilt: die strukturelle, die funktionale und die nichtfunktionale Analyse. Beim konzeptionellen Vergleich der Datenbanken hinsichtlich Indexe in der strukturellen Analyse wurde festgestellt, dass alle drei Datenbanksysteme den ‚Primary Key', ‚Unique Index' und ‚Secondary Index' unterstützten. Besonders in der nichtfunktionalen Analyse optimierte MongoDB durch das Nutzen eines nicht eindeutigen Index die Performance bei der Suche nach bestimmten Datensätzen. Obwohl CockroachDB und MariaDB relationale Datenbanken sind, lieferten die beiden Systeme bei den CRUD-Operationen eine unterschiedliche Performance. MariaDB schnitt in sämtlichen Testszenarien in der nichtfunktionalen Analyse am besten ab. Beim Einfügen von 1 000 000 Datensätzen sowie bei der Suche, Änderung und Löschung der Daten lieferte MariaDB verglichen zu MongoDB und CockroachDB eine schnellere und bessere Leistung. Zudem unterstützte MariaDB in der funktionalen Analyse die ACID-Transaktion und ermöglichte ein einfaches Erstellen von Trigger-Operationen. MongoDB leistete nach MariaDB in der nichtfunktionalen Analyse eine gute Performance. Bis auf das Testszenario mit der ‚Join'-Operation war MongoDB bei den CRUD-Operationen schneller als CockroachDB. Zudem unterstützte MongoDB als NoSQL-Datenbanksystem Funktionen aus den traditionellen Datenbanken, z. B. Trigger, Join und die ACID-Transaktion. CockroachDB unterstützte die ACID-Eigenschaften in einer Transaktion; dennoch schnitt CockroachDB bei den anderen Anwendungsfällen in der nichtfunktionalen Analyse am schlechtesten ab. Hinzufügend enthält CockroachDB einige Features nicht, die MongoDB und MariaDB unterstützen, z. B. die Trigger-Funktion. Der Grund hierfür liegt darin, dass MariaDB und MongoDB länger auf dem Markt sind und über

die Jahre optimiert wurden, um Nutzerinnen und Nutzern viele leistungsfähigere Features anzubieten.

5.2 Ausblick

Das Experiment umfasste einige Funktionen konventioneller Datenbanken. Dennoch deckte es nicht die Komplexität und Vielfalt der verfügbaren Datenbankfunktionen ab. Daher bleiben noch viele Funktionen offen, die verglichen werden können. Bei der strukturellen Analyse können die anderen Indexarten, die im Vergleich nicht genannt wurden, miteinander verglichen werden. Zudem wurde beim Trigger nicht geprüft, ob die Anzahl der Trigger einen Einfluss auf die Performance der Datenbanken hat. Außerdem war die Transaktion in der funktionalen Analyse nicht komplex; komplexe Transaktionen wurden im Vergleich also nicht berücksichtig. In der nichtfunktionalen Analyse könnten mehr Datensätze für die CRUD-Operationen verwendet werden und es könnte mit unterschiedlichen Datenstrukturen, z. B. mit semi- und unstrukturierten Daten, gearbeitet werden. Des Weiteren ist CockroachDB momentan in ihrer frühen Phase, in der viele Anwendungsbereiche und Funktionen noch nicht weiterentwickelt sind, weshalb viele weitere Releases mit neuen Features bevorstehen. Letztlich wurden die Tests in einer Testumgebung mit einem Server und begrenzten Hardwaretypen durchgeführt. Daher würde ein zukünftiger Vergleich mit mehreren Servern und unterschiedlichen Hardwaretypen inklusive Read-Write-Konflikten, unterschiedlichen Datenstrukturen sowie komplexen Transaktionen und Abfragen aufschlussreich sein.

Literaturverzeichnis

[1] V. Abramova, J. Bernardino, und P. Furtado, „Experimental Evaluation of NoSQL Databases“, *International Journal of Database Management Systems*, Bd. 6, Nr. 3, S. 01–16, Juni 2014, doi: 10.5121/ijdms.2014.6301.

[2] R. Kumar, N. Gupta, S. Charu, und S. Kumar Jangir, „Manage Big Data through NewSQL“, in *National Conference on Innovation in Wireless Communication and Networking Technology - 2014*, Apr. 2014, S. 1–5. doi: 10.13140/2.1.3965.3768.

[3] P. Baldwa, „Evolution of Databases - A Fascinating Journey - SG Analytics“, Feb. 14, 2019. https://www.sganalytics.com/blog/evolution-of-databases/ (zugegriffen Aug. 18, 2022).

[4] K. D. Foote, „A Brief History of Database Management - DATAVERSITY“, Okt. 25, 2021. https://www.dataversity.net/brief-history-database-management/ (zugegriffen Aug. 18, 2022).

[5] S. Kanoje, V. Powar, und D. Mukhopadhyay, „Using MongoDB for Social Networking Website Deciphering the Pros and Cons“, *International Journal of Database Management Systems (IJDMS) Vol.6, No.3, June 2014* , Bd. 6, Juni 2014, [Online]. Available: https://www.researchgate.net/publication/273260515_Using_MongoDB_for_Social_Networking_Website_Deciphering_the_Pros_and_Cons/ (zugegriffen Aug. 18, 2022).

[6] D. Kunda und H. Phiri, „A Comparative Study of NoSQL and Relational Database“, *ZAMBIA INFORMATION COMMUNICATION TECHNOLOGY (ICT) JOURNAL,Volume 1 (Issue 1) (2017) Pages 1-4*, Bd. 1, Nr. 1, S. 1–4, 2017, Zugegriffen: Aug. 19, 2022. [Online]. Available: https://ictjournal.icict.org.zm/index.php/zictjournal/article/view/8/3

[7] A. Singla, Nishu Bali, und Deepika Chaudhary, „Big Data and Its Applications“, *Journal of Technology Management for Growing Economies*, Bd. 11, Nr. 2, S. 63–67, Nov. 2020, doi: 10.15415/jtmge.2020.112008.

[8] TechVidvan, „The History, Evolution, & Technologies of Big Data [with use cases] - TechVidvan“, *TechVidvan*, 2022. https://techvidvan.com/tutorials/big-data-history/ (zugegriffen Aug. 18, 2022).

[9] TechVidvan, „What is Big Data - A Complete Comprehensive Guide - TechVidvan“, *TechVidvan*, 2022. https://techvidvan.com/tutorials/big-data-complete-guide/ (zugegriffen Aug. 18, 2022).

[10] P. Shukla und K. Atkotiya, „(PDF) NoSQL Databases : Big data characteristics and comparison of traditional and big data analytics.“, *Quest International Multidisciplinary Research Journal Vol 4, Issue 11, December 2015*, Zugegriffen: Aug. 18, 2022. [Online]. Available: https://www.researchgate.net/publication/290542203_NoSQL_Databases_Big_data_characteristics_and_comparison_of_traditional_and_big_data_analytics

[11] Oracle, „Was versteht man unter Big Data? | Oracle Deutschland“, *Oracle*, 2022. https://www.oracle.com/de/big-data/what-is-big-data/ (zugegriffen Aug. 18, 2022).

[12] SAP, „Was ist Big Data? | Fortschrittliche Big-Data-Analysen | SAP Insights“, *SAP*, 2022. https://www.sap.com/germany/insights/what-is-big-data.html (zugegriffen Aug. 18, 2022).

[13] A. Beaulieu, *Einführung in SQL*. O'Reilly Germany, 2009. Zugegriffen: Sep. 18, 2022. [Online]. Available: https://books.google.de/books?hl=de&lr=&id=FuCxAgAAQBAJ&oi=fnd&pg=PP5&dq=%09Beaulieu,+Alan+Einf%C3%BChrung+in+SQL+Daten+erzeugen,+bearbeiten+und+abfragen&ots=zFKRBzsnvY&sig=MTS17IhLfeJyb7XZP5RS2nKPMXQ&redir_esc=y#v=onepage&q&f=false

[14] R. Deari, X. Zenuni, J. Ajdari, F. Ismaili, und B. Raufi, „ANALYSIS AND COMPARISON OF DOCUMENT-BASED DATABASES WITH SQL RELATIONAL DATABASES: MONGODB VS MYSQL“, 2018.

[15] S. Chickerur, A. Goudar, und A. Kinnerkar, „Comparison of Relational Database with Document-Oriented Database (MongoDB) for Big Data Applications“, in *Proceedings*

- 8th International Conference on Advanced Software Engineering and Its Applications, ASEA 2015, März 2016, S. 41–47. doi: 10.1109/ASEA.2015.19.

[16] D. Ramesh, A. Sinha, und S. Singh, „Data modelling for discrete time series data using Cassandra and MongoDB“, in *2016 3rd International Conference on Recent Advances in Information Technology, RAIT 2016*, Juli 2016, S. 598–601. doi: 10.1109/RAIT.2016.7507966.

[17] J. Oliveira und J. Bernardino, „NewSQL databases: MemSQL and VoltDB experimental evaluation“, in *IC3K 2017 - Proceedings of the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, 2017, Bd. 2, S. 276–281. doi: 10.5220/0006518902760281.

[18] G. A. Schreiner, R. Knob, D. Duarte, P. Vilain, und R. dos Santos Mello, „NewSQL through the looking glass“, Dez. 2019. doi: 10.1145/3366030.3366080.

[19] A. Boicea, F. Radulescu, und L. I. Agapin, „MongoDB vs Oracle - Database comparison“, *Proceedings - 3rd International Conference on Emerging Intelligent Data and Web Technologies, EIDWT 2012*, S. 330–335, 2012, doi: 10.1109/EIDWT.2012.32.

[20] U. Hairah und E. Budiman, „Inner Join Query Performance: MariaDB vs PostgreSQL“, in *Journal of Physics: Conference Series*, März 2021, Bd. 1844, Nr. 1. doi: 10.1088/1742-6596/1844/1/012021.

[21] K. Kaur und M. Sachdeva, „Performance Evaluation of NewSQL Databases“, in international Conference on Inventive Systems and Control (ICISC-2017), 2017.

[22] Solid IT GmbH, „DB-Engines Ranking - popularity ranking of document stores“. https://db-engines.com/en/ranking/document+store (zugegriffen Aug. 20, 2022).

[23] S. Wickramasinghe, „MongoDB vs DynamoDB: Comparing NoSQL Databases – BMC Software | Blogs“, *bmc blogs*, Juli 14, 2021. https://www.bmc.com/blogs/mongodb-vs-dynamodb/ (zugegriffen Aug. 20, 2022).

[24] A. Pomponio, „MariaDB Overview: Key Features, Benefits, and FAQ | OpenLogic by Perforce“, Juli 29, 2021. https://www.openlogic.com/blog/mariadb-overview (zugegriffen Aug. 20, 2022).

[25] M. Chojrin, „MariaDB Vs MySQL: Performance, Licensing & Support“, *Panoply Blog*, Juni 03, 2021. https://blog.panoply.io/a-comparative-vmariadb-vs-mysql (zugegriffen Aug. 20, 2022).

[26] PAT RESEARCH, „Top 13 NewSQL Databases in 2022 - Reviews, Features, Pricing, Comparison - PAT RESEARCH: B2B Reviews, Buying Guides & Best Practices“, 2022. https://www.predictiveanalyticstoday.com/newsql-databases/ (zugegriffen Aug. 20, 2022).

[27] MariaDB, „Transactions - MariaDB Knowledge Base“, 2022. https://mariadb.com/kb/en/transactions/ (zugegriffen Aug. 21, 2022).

[28] J. Schulze, „MariaDB und MySQL – Vergleich der Features | Informatik Aktuell“, Juli 26, 2016. https://www.informatik-aktuell.de/betrieb/datenbanken/mariadb-und-mysql-vergleich-der-features.html (zugegriffen Aug. 20, 2022).

[29] MariaDB, „Arbeitslastoptimierte Speicher-Engines | MariaDB“, 2022. https://mariadb.com/de/database-topics/storage-engines/ (zugegriffen Aug. 20, 2022).

[30] C. Yujun, Y. Lou, und F. Ye, „Research on Data Query Optimization Based on SparkSQL and MongoDB“, *Proceedings - 2018 17th International Symposium on Distributed Computing and Applications for Business Engineering and Science, DCABES 2018*, S. 144–147, Dez. 2018, doi: 10.1109/DCABES.2018.00046.

[31] MongoDB, „Examples For Using MongoDB: Syntax, Shell And Querying Data | MongoDB“, 2022. https://www.mongodb.com/basics/examples (zugegriffen Aug. 21, 2022).

[32] S. Singh, „Security Analysis of MongoDB“, *International Journal for Digital Society*, Bd. 10, Nr. 4, S. 1556–1561, Dez. 2019, doi: 10.20533/ijds.2040.2570.2019.0193.

[33] K. Seguin, *The Little MongoDB Book*. 2013. Zugegriffen: Aug. 21, 2022. [Online]. Available: http://github.com/karlseguin/the-little-mongodb-book.

[34] MongoDB, „Indexes — MongoDB Manual“, 2022. https://www.mongodb.com/docs/manual/indexes/ (zugegriffen Aug. 21, 2022).

[35] R. Sharma, „MongoDB Architecture: Structure, Terminologies, Requirement & Benefits | upGrad blog“, *upGrad blog*, Dez. 28, 2020. https://www.upgrad.com/blog/mongodb-architecture/ (zugegriffen Aug. 21, 2022).

[36] MongoDB, „Sharding — MongoDB Manual“, 2022. https://www.mongodb.com/docs/manual/sharding/ (zugegriffen Aug. 21, 2022).

[37] MongoDB, „Config Servers — MongoDB Manual“, 2022. https://www.mongodb.com/docs/manual/core/sharded-cluster-config-servers/ (zugegriffen Aug. 21, 2022).

[38] MongoDB, „Storage Engines — MongoDB Manual“, 2022. https://www.mongodb.com/docs/manual/core/storage-engines/ (zugegriffen Aug. 21, 2022).

[39] R. Taft *u. a.*, „CockroachDB: The Resilient Geo-Distributed SQL Database“, in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Juni 2020, S. 1493–1509. doi: 10.1145/3318464.3386134.

[40] Cockroach Labs, „Storage Layer | CockroachDB Docs“, 2022. https://www.cockroachlabs.com/docs/stable/architecture/storage-layer.html (zugegriffen Aug. 21, 2022).

[41] Study.com, „Database Triggers: Examples & Overview - Video & Lesson Transcript | Study.com“, 2022. https://study.com/academy/lesson/database-triggers-examples-overview.html (zugegriffen Aug. 21, 2022).

[42] MariaDB, „The Essentials of an Index - MariaDB Knowledge Base“, 2022. https://mariadb.com/kb/en/the-essentials-of-an-index/ (zugegriffen Aug. 21, 2022).

[43] MariaDB, „Getting Started with Indexes - MariaDB Knowledge Base“, 2022. https://mariadb.com/kb/en/getting-started-with-indexes/ (zugegriffen Aug. 21, 2022).

[44] Tech ON THE Net, „MariaDB: Primary Keys“, 2022. https://www.techonthenet.com/mariadb/primary_keys.php (zugegriffen Aug. 21, 2022).

[45] MariaDB, „Storage Engine Index Types - MariaDB Knowledge Base“, 2022. https://mariadb.com/kb/en/storage-engine-index-types/ (zugegriffen Aug. 21, 2022).

[46] Database of Databases, „WiredTiger - Database of Databases“, 2022. https://dbdb.io/db/wiredtiger (zugegriffen Aug. 21, 2022).

[47] Cockroach, „Indexes - cockroachdb | DeepKB“, Feb. 16, 2020. https://deepkb.com/CO_000018/en/kb/IMPORT-95b5bd6b-aa6c-3907-887f-88c8ca031d7a/indexes (zugegriffen Aug. 21, 2022).

[48] C. Luo und M. J. Carey, „LSM-based storage techniques: a survey“, *The VLDB Journal*, Bd. 29, S. 393–418, 2020, doi: 10.1007/s00778-019-00555-y.

[49] M. Kumar Gupta und D. Badal, „COMPARATIVE STUDY OF INDEXING TECHNIQUES IN DBMS Selection of Initial Cluster Centroids for K-means Algorithm View project Efficient Indexing Technique to Improve the Performance of ‚LIKE‘ Operator based Comparisons View project COMPARATIVE STUDY OF INDEXING TECHNIQUES IN DBMS“, Zugegriffen: Aug. 21, 2022. [Online]. Available: https://www.researchgate.net/publication/333844844

[50] N. Samuel, „MongoDB Triggers Simplified: A Comprehensive Guide 101“, *Database Management Systems, MongoDB*, Feb. 21, 2022. https://hevodata.com/learn/mongodb-triggers/ (zugegriffen Aug. 21, 2022).

[51] MariaDB Tutorial, „MariaDB Triggers“, 2022. https://www.mariadbtutorial.com/mariadb-triggers/ (zugegriffen Aug. 21, 2022).

[52] MariaDB, „Events Overview - MariaDB Knowledge Base“, 2022. https://mariadb.com/kb/en/events/ (zugegriffen Aug. 21, 2022).

[53] MariaDB, „Trigger Limitations - MariaDB Knowledge Base“, 2022. https://mariadb.com/kb/en/trigger-limitations/ (zugegriffen Aug. 21, 2022).

[54] MongoDB, „Atlas Triggers Overview — Atlas App Services“, 2022. https://www.mongodb.com/docs/atlas/app-services/triggers/overview/ (zugegriffen Aug. 21, 2022).

[55] N. Litzel und S. Luber, „Was ist ACID?“, Nov. 15, 2018. https://www.bigdata-insider.de/was-ist-acid-a-776182/ (zugegriffen Sep. 14, 2022).

[56] MongoDB, „Read Isolation, Consistency, and Recency — MongoDB Manual“, 2022. https://www.mongodb.com/docs/manual/core/read-isolation-consistency-recency/ (zugegriffen Aug. 21, 2022).

[57] J. Griffin, „MySQL and the ACID Properties“, *LogicalRead*, Sep. 24, 2015. https://logicalread.com/mysql-acid-properties-mc13/#.YwF8by7P02w (zugegriffen Aug. 21, 2022).

[58] M. Tracy, „Serializable, Lockless, Distributed: Isolation in CockroachDB“, *Cockroach Labs*, Mai 04, 2016. https://www.cockroachlabs.com/blog/serializable-lockless-distributed-isolation-cockroachdb/ (zugegriffen Aug. 21, 2022).

[59] A. Sugandhi, „Data Relationships in MongoDB – A quick guide“, *Knowledgehut*, Jan. 19, 2022. https://www.knowledgehut.com/blog/web-development/data-relationships-in-mongodb (zugegriffen Aug. 21, 2022).

[60] DEVATHON TEAM, „CockroachDB vs (MySQL, PostgreSQL, MongoDB & Cassandra)“, Apr. 29, 2021. https://devathon.com/blog/cockroachdb-vs-mysql-vs-postgresql-vs-mongodb-vs-cassandra/ (zugegriffen Aug. 21, 2022).

[61] MongoDB, „$lookup (aggregation) — MongoDB Manual“, 2022. https://www.mongodb.com/docs/manual/reference/operator/aggregation/lookup/#mongodb-pipeline-pipe.-lookup (zugegriffen Aug. 21, 2022).

[62] M. Stonebraker, „Comparison of JOINS: MongoDB vs. PostgreSQL“, *EDB*, März 31, 2020. https://www.enterprisedb.com/blog/comparison-joins-mongodb-vs-postgresql (zugegriffen Sep. 14, 2022).

[63] M. Dancuk, „What is NewSQL? {+Best NewSQL Databases} | phoenixNAP KB“, 2022. https://phoenixnap.com/kb/newsql (zugegriffen Aug. 20, 2022).

[64] ConSol GmbH, „Technologie-Vergleich SQL NoSQL NewSQL | ConSol GmbH“, 2022. https://www.consol-software.at/software-engineering/software-architektur/technologie-vergleich-sql/ (zugegriffen Aug. 20, 2022).

[65] MongoDB, „Replication — MongoDB Manual“, 2022. https://www.mongodb.com/docs/manual/replication/#automatic-failover (zugegriffen Aug. 21, 2022).

[66] MariaDB, „Online Schema Changes with MariaDB Enterprise Cluster — MariaDB Enterprise Documentation“, 2022. https://mariadb.com/docs/sql/statements/schema/online-schema-changes/enterprise-server/galera/ (zugegriffen Sep. 19, 2022).

[67] K. Ksiazek, „How to Design a Geographically Distributed MariaDB Cluster | Severalnines“, Juli 20, 2020. https://severalnines.com/blog/how-design-geographically-distributed-mariadb-cluster/ (zugegriffen Aug. 21, 2022).

[68] MongoDB, „Segmenting Data by Location — MongoDB Manual“, 2022. https://www.mongodb.com/docs/manual/tutorial/sharding-segmenting-data-by-location/ (zugegriffen Aug. 21, 2022).

[69] MariaDB, „Coding in MariaDB - MariaDB Knowledge Base“, 2022. https://mariadb.com/kb/en/coding-in-mariadb/ (zugegriffen Aug. 21, 2022).

[70] MariaDB, „OS Compatibility — MariaDB Enterprise Documentation“, 2022. https://mariadb.com/docs/deploy/os-compatibility/ (zugegriffen Aug. 21, 2022).

[71] MariaDB, „Application Programming Interfaces - MariaDB Knowledge Base“, 2022. https://mariadb.com/kb/en/connectors/ (zugegriffen Aug. 21, 2022).

[72] MariaDB, „Block-Based Join Algorithms - MariaDB Knowledge Base“, 2022. https://mariadb.com/kb/en/block-based-join-algorithms/ (zugegriffen Aug. 21, 2022).

[73] G. Utsin, „Vectorizing the Merge Joiner in CockroachDB“, *Cockroach Labs*, Juni 18, 2019. https://www.cockroachlabs.com/blog/vectorizing-the-merge-joiner-in-cockroachdb/ (zugegriffen Aug. 21, 2022).

A Anhang

Folgende Dateien befinden sich auf der beigefügten CD:

- die Bachelorarbeit als PDF-Dokument.

3.3 Funktionale Analyse

3.3.1 Trigger

- Der Trigger von MongoDB ist in einer JavaScript-Datei mit der Bezeichnung „Trigger-MongoDB“.
- Der Trigger von MariaDB ist in einer SQL-Datei mit der Bezeichnung „Trigger-MariaDB“.

3.3.2 ACID

- MongoDB-Befehle für die Instanzen 1 und 2 befinden sich in den Textdokumenten „ACID-MongoDB-Instanz-1“ und „ACID-MongoDB-Instanz-2".
- Die CockroachDB-Instanzen 1 und 2 befinden sich im PyCharm-Projekt „bachelor_thesis“ im Ordner „ACID“.
- Die MariaDB-Instanzen 1 und 2 befinden sich im PyCharm-Projekt „bachelor_thesis“ im Ordner „ACID“ oder sind als SQL-Dateien „ACID-MariaDB-Instanz-1“ und „ACID-MariaDB-Instanz-2“ zu finden.

3.4 Nichtfunktionale Analyse

- Die Performanceanalyse von MariaDB, MongoDB und CockroachDB befindet sich im PyCharm-Projekt „bachelor_thesis“ unter „performance_check“.
- Excel-CSV-Datei „Mitarbeiter(1000000)“

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

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

Hamburg	23.09.2022	[redacted]
Ort	Datum	Unterschrift im Original