Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Florian Arfert

Laufzeitanalyse von Main-Memory-Datenbanksystemen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Florian Arfert

Laufzeitanalyse von Main-Memory-Datenbanksystemen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Wirtschaftsinformatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr. Wolfgang Gerken

Abgegeben am 16.05.2017

Florian Arfert

Thema der Arbeit
Laufzeitanalyse von Main-Memory-Datenbanksystemen

Stichworte
Laufzeitanalyse, Benchmark, Main-Memory-Datenbanksystem, In-Memory-Datenbanksystem, Hauptspeicherdatenbanksystem

Kurzzusammenfassung
Diese Arbeit beschäftigt sich mit der Analyse von Hauptspeicherdatenbanksystemen im praktischen Einsatz. Der erste Teil führt in die Thematik der Hauptspeicherdatenbanksysteme ein. Danach erfolgt eine Vorstellung der Datenbanksysteme, die in den diversen Tests genutzt werden. Der dritte und letzte Teil beschäftigt sich mit dem Aufbau, der Durchführung und der Auswertung der Tests.

Florian Arfert

Title of the paper
Runtime analysis of main memory database systems

Keywords
Runtime analysis, benchmark, main memory database systems, in-memory database system

Abstract
This paper is about an analysis of main memory database systems in practical use. The first part of this paper is an introduction to main memory database systems. The next part presents a selection of database systems, which are used for the examination in various tests. The third and last part is about the setup, execution and evaluation of those test.

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abkürzung	Bedeutung
ACID	Atomicity, Consistency, Isolation, Durability
AG	Aktiengesellschaft
bzw.	beziehungsweise
CPU	Central Processing Unit
ERM	Entity-Relationship-Modell
ETL	Extract, Transform, Load; auch: Extract-Transform-Load
GB	Gigabyte
I/O	Input/Output
Inc.	Incorporation
IT	Informationstechnolgie
KB	Kilobyte
L1 (-Cache)	First-Level-Cache
L2 (-Cache)	Second-Level-Cache
L3 (-Cache)	Third-Level-Cache
ms	Millisekunde(n)
MB	Megabyte
MVCC	Multiversion Concurrency Control
NAT	Network Address Translation
NUMA	Non-Uniform Memory Access
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
PL/SQL	Procedural Language/Structured Query Language
RAM	Random Access Memory
SQL	Structured Query Language
STD-KOMP	Standard-Kompression der Datensätze

SPEZ-KOMP	Kompression der Datensätze anhand einer Spezifikation
T-SQL	Transact-SQL
TB	Terabyte
TPC	Transaction Processing Performance Council
UDF	User Defined Function
UMA	Uniform Memory Access

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Wirt- und Wissenschaft entwickeln fortlaufend neue Software-Lösungen, um ihre Prozesse zu unterstützen und zu verbessern. Viele dieser Entwicklungen, die zum Beispiel das operative Tagesgeschäft in Unternehmen unterstützen, indem essentielle Geschäftsdaten gespeichert oder großen Datenmengen zur Erkenntnisgewinnung analysiert werden, basieren auf dem Einsatz von Datenbanksystemen. Dabei wachsen die Ansprüche an diese Software-Lösungen und den damit verbundenen Datenbanksystemen.

Traditionelle Datenbanksysteme werden den steigenden Anforderungen an sie immer seltener gerecht, da sie auf der Annahme basieren, dass die Festplatte das primäre Speichermedium in der Datenhaltung darstellt. Dieser Grundsatz hat große Auswirkungen bei der stetig wachsenden Anzahl an Daten, da die Festplatte in der modernen Speicherhierarchie das langsamste Medium darstellt. Folglich wird ein Datenbanksystem in seiner Leistungsfähigkeit begrenzt, wodurch Anwendungen und Prozesse nicht optimal ablaufen.

Der Hauptspeicher als primärer Ort der Datenhaltung bietet aus technischer Sicht Potenzial für Leistungsverbesserungen in Datenbanksystemen. Dabei ist dieses Konzept kein neues, sondern war bereits in den 80er Jahren des 20. Jahrhunderts Gegenstand hauptsächlich theoretischer Untersuchungen (vgl. Eich, 1989, S. 251-268). Mit den fallenden Preisen für Arbeitsspeicher und der Entwicklung der 64-Bit-Technolgie stieg das Interesse, Daten primär im Hauptspeicher der Datenbanksysteme zu halten und in dieser Form für die Verarbeitung zu nutzen, an und dieser Technologie wird ein enormes Potenzial nachgesagt (vgl. Matt, 2012, S. 229-230).

Das Ziel der Arbeit ist die Untersuchung des praktischen Leistungsvermögens von Hauptspeicherdatenbanksystemen und den daraus folgenden Einsatzmöglichkeiten. Auf die Erreichung dieses Ziels wird in den drei Bereichen des Hauptteils dieser Arbeit hingeführt. In dem ersten Bereich wird eine Einführung zu dieser Art von Datenbanksystemen gegeben. Zu dieser gehören die zu den Hauptspeicher-datenbanksystemen entsprechende Definition, die Abgrenzung dieser Datenbanksysteme mit den festplattenbasierten Datenbanksystemen und eine Betrachtung von ausgewählten technischen Merkmalen. Im darauffolgenden Bereich wird eine Auswahl von zu untersuchenden Hauptspeicherdatenbanksystemen vorgestellt. Der dritte und letzte Bereich des Hauptteils stellt die Laufzeitanalyse dar. In diesem wird auf die Organisation und Durchführung der Versuche eingegangen und die Resultate der Messungen werden analysiert.

In dieser Arbeit werden grundlegende Kenntnisse aus dem Gebiet der Datenbanksysteme vorausgesetzt, da keine Einführung in dieses erfolgt. Die verwendeten Begriffe sind teilweise aus dem Englischen entnommen, da diese an entsprechender Stelle passender erscheinen und in der IT-Welt Englisch die Fachsprache darstellt.

2 Main-Memory-Datenbanksysteme

Ein Main-Memory-Datenbanksystem (auch: In-Memory-Datenbanksystem, dt.: Hauptspeicherdatenbanksystem) ist ein Datenbanksystem, in dem der Hauptspeicher das grundlegende Speichermedium für die Daten bildet. (vgl. Loos et al., 2011, S. 384)

In herkömmlichen Datenbanksystemen werden die Datensätze primär auf der Festplatte gespeichert. Diese Systeme verwenden einen Datenbankpuffer, um häufig benötigte Daten im Arbeitsspeicher bereitzustellen. Dadurch kann ein Lokalitätseffekt ausgenutzt werden, der eine geringere Anzahl an Zugriffen auf die Festplatte bewirkt. Die selteneren Zugriffe führen zu einer Verbesserung der Verarbeitungsgeschwindigkeit (vgl. Kemper und Eickler, 2015, S. 210).
Im Gegensatz zu den festplattenbasierten Datenbanksystemen laden Main-Memory-Datenbanksysteme den gesamten Datenbestand, in der Regel einmalig beim Systemstart, in den Hauptspeicher und führen Lese- und Schreiboperationen nur auf den dort befindlichen Daten durch. Auf die Festplatte wird in Hauptspeicher-datenbanken nur für Sicherungs- und Wiederherstellungszwecke zugegriffen (vgl. Garcia-Molina und Salem, 1992, S. 512).

2.1 Unterschiede in der Verarbeitung

Die unterschiedlichen Eigenschaften der Speichermedien Hauptspeicher und Festplatte beeinflussen die Leistung von Datenbanksystemen. Nachfolgend sollen zwei signifikante Aspekte betrachtet werden, die die Auswirkungen der Datenträger im Hinblick auf die Verarbeitung in Datenbanksystemen verdeutlichen.

2.1.1 Die Speicherhierarchie

Für das Verständnis der Verarbeitungsvorgänge in Datenbanksystemen ist eine Betrachtung der Speicherhierarchie von Computern von wesentlicher Bedeutung. Die pyramidenförmige Hierarchie in Abbildung 2.1 stellt dar, wie sich die Verarbeitung von Daten entlang der Speichermedien verhält (dieses Unterkapitel beruht im Wesentlichen auf (Kemper und Eickler. 2014, S. 211-214) und (Plattner, 2013, S. 23-25)).

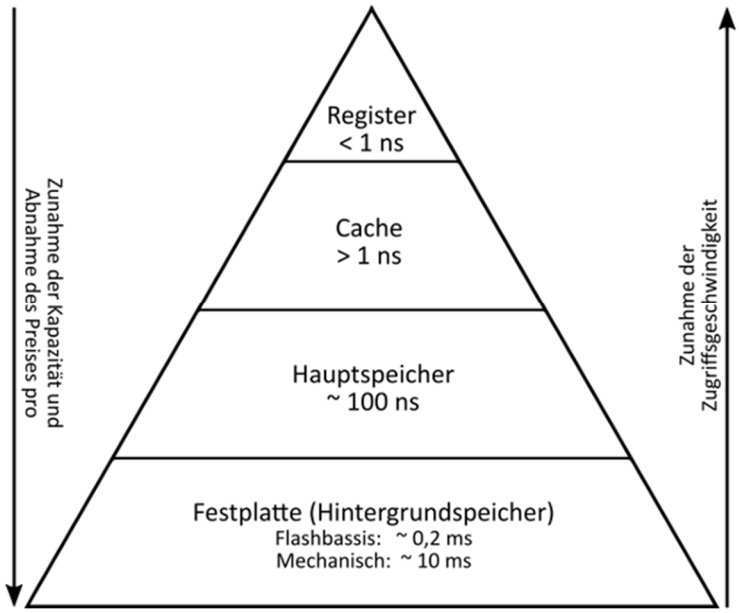

Abbildung 2.1 Speicherhierarchie[1]

Der Zugriff auf die Daten wird logisch von oben nach unten entlang der Stufenordnung durchgeführt. Dabei stellen höhere Schichten einen schnelleren und kürzeren Zugang zu den Speichermedien dar, während die unteren Ebenen im Vergleich zu den Oberen größere Kapazitäten und geringere Kosten pro Bit bzw. Byte aufweisen.

Die Daten, die sich nicht unmittelbar in der höchsten Schicht und somit direkt von der Recheneinheit eines Prozessors abrufbar sind, werden aus den unteren Schichten geholt. Bei der Ausführung von Anweisungen können die oberen Ebenen eine Pufferfunktion einnehmen. Häufig benötigte Daten werden in den entsprechenden Speichermedien gehalten, um die Leistung zu verbessern, da Lese- und/oder Schreibvorgänge minimiert beziehungsweise gänzlich vermieden werden.

[1] Vgl. (Plattner, 2013, S. 24) und (Kemper und Eickler, 2015, S. 213)

Auf der obersten Ebene befinden sich die Register. Dabei handelt es sich um kleine Speicher, die direkt mit der Recheneinheit eines Prozessorkerns verbunden sind. In den Registern werden die Anweisungen und Daten geladen und nach der Ausführung werden die Daten und Anweisungen wieder in den Registern gespeichert. Die übliche Größe für ein Register ist die Länge eines Datenwortes, bei modernen Prozessoren dementsprechend 64-Bit.

In der direkt darunterliegenden Ebene befinden sich die Caches. Diese übernehmen eine Pufferfunktion zwischen Registern und Arbeitsspeicher. Caches werden eingesetzt um die Verzögerung, die beim direkten Zugriff zwischen Registerspeicher und Hauptspeicher entstehen würden, zu reduzieren. Damit diese Pufferrolle möglichst effizient funktioniert, nutzen Caches unterschiedliche Strategien, um möglichst die nächstbenötigten Daten bereitzustellen. Die Speicherkapazitäten dieser Puffer sind mehrere hundert Kilobyte (insbesondere L1-Cache, aber auch L2-Cache) oder bis hin zu einigen Megabyte (L2-Cache und vor allem L3-Cache) groß. Die Caches sind intern in sogenannte Cache-Lines aufgeteilt, die jede für sich in modernen Architekturen 64 oder 128 Byte groß sind.

Die dritte Schicht bildet der Hauptspeicher mit Zugriffszeiten von 100 Nanosekunden. Diese Ebene bildet bei In-Memory-Datenbanksystemen den Kern der Datenhaltung, während bei Systemen auf Festplattenbasis diese Stufe den Datenbankpuffer darstellt. Dieser Puffer soll die Verzögerung, die bei der I/O-Kommunikation zwischen den oberen drei und den unteren Schichten entsteht, verringern. Zu diesem Zweck ist dieser Zwischenspeicher in der Regel viele Gigabyte groß und kann eine beträchtliche Anzahl von Daten aus dem Hintergrundspeicher im Primärspeicher bereitstellen. Die so bevorrateten Daten werden mittels einer Ersetzungsstrategie, falls die Notwendigkeit besteht, wieder auf die Festplatte zurückgeschrieben werden, sodass Platz für benötigte Daten geschaffen wird.

Die vierte und letzte Schicht ist die der Hintergrundspeicher. Auf diesen werden Daten geschrieben, die in einem persistenten Zustand verweilen. Werden diese Daten im Datenbanksystem benötigt, so werden sie vom Hintergrundspeicher gelesen und im Primärspeicher für die Verarbeitung zur Verfügung gestellt.

Unterzieht man nun die Zugriffszeiten zwischen Haupt- und Festplattenspeicher einem Vergleich, so ist ersichtlich, dass der durchschnittliche Zugriff auf mechanische Festplatte 100.000-mal, bei flashbasierten Festplatten immer noch 2.000-mal langsamer ist als bei dem Hauptspeicher, was diesen zu einer Option bei der Wahl eines Speichermediums für Daten macht.

2.1.2 Aufwand in der Abfrageverarbeitung

In der Arbeit von Harizopoulos et al. (vgl. Harizopoulos, 2007, S. 982) wird gezeigt, dass sich in den herkömmlichen Datenbanksystemen nur ein kleiner Teil der Verarbeitungsschritte sich direkt auf die durchzuführenden Abfragen beziehen und somit für diese nützlich sind. In der Abbildung 2.2 wird ein Überblick über den Aufwand in der Abfrageverarbeitung dargestellt (vgl. Gessert et al., 2016, S. 7):

- Pufferverwaltung (34,6%): Abfederung von langsamen Festplattenzugriffen.
- Locking (16,3%): Sicherstellung der logischen Isolation im Transaktionsbetrieb.
- Latching (14,2%): Schutz von gemeinsam genutzten Datenstrukturen in der parallelen Datenverarbeitung und Synchronisation von Threads.
- Logging (11,9%): Stellt die Dauerhaftigkeit des Datenbestandes sicher.
- Optimierungstechniken (16,2%): Verbesserung der Leistung durch Kenntnisse über den Anwendungsbereich des Datenbanksystems.
- Nützlich (6,8%): Verarbeitungsschritte, die im direkten Zusammenhang mit der Abfrage stehen.

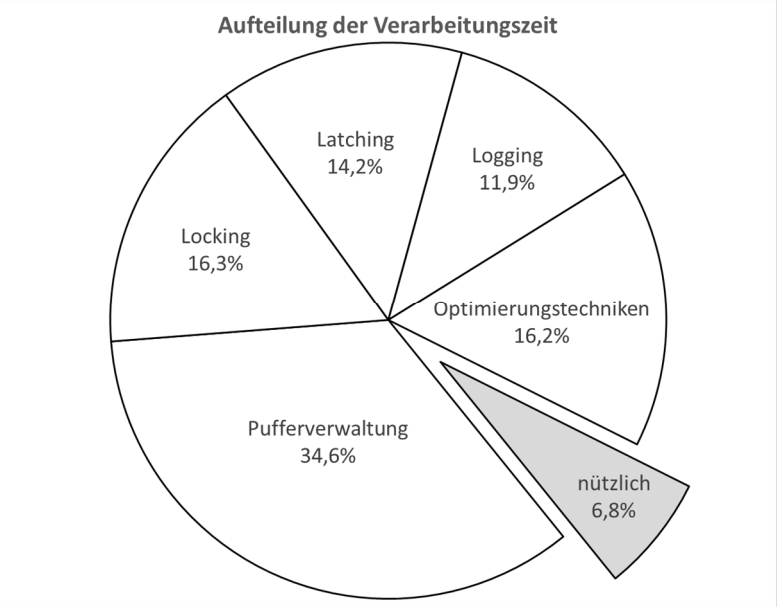

Abbildung 2.2 Aufwand in der Verarbeitung

Hauptspeicherdatenbanken bieten Potenziale, um die Effizienz in der Verarbeitung teilweise drastisch zu steigern. Ein Beispiel für eine Leistungssteigerung ist das Entfernen der Pufferverwaltung und des dazugehörigen Datenbankpuffers (dieser

und der nächste folgende Abschnitt basieren im Wesentlichen auf (Kemper und Eickler, 2015, S. 586-587)).

Eingangs in Kapitel zwei wird erwähnt, dass in den herkömmlichen Datenbanken die Pufferverwaltung die primär auf den Festplatten gespeicherten Daten für die Verarbeitung im Hauptspeicher organisiert. In Main-Memory-Datenbanken befinden sich die Daten aber grundsätzlich im Hauptspeicher. In diesen Systemen hätte die Pufferverwaltung mitsamt Datenbankpuffer keine Funktion und wird deshalb zu Optimierungszwecken weggelassen.

2.2 Auswahl technischer Aspekte

Hauptspeicherdatenbanksysteme ermöglichen eine deutlich effizientere Durchführung von Verarbeitungsvorgängen gegenüber Datenbanksystemen mit Festplatten als primären Speicherort (siehe Unterkapitel 2.1).

Damit die Effizienzsteigerung möglichst hoch ausfällt, sind die veränderten Grundlagen der Datenspeicherung zu berücksichtigen. Das Prinzip vom Arbeitsspeicher als primäres Speichermedium hat weitreichende Auswirkungen auf die Entwicklung und Architektur eines Datenbanksystems. Viele der für die herkömmlichen Datenbanken entwickelten technischen Lösungen lassen sich nur bedingt oder gar nicht verwenden, da diesen die Festplatte als primäres Speichermedium zugrunde liegt. Die geänderten Grundlagen bedingen die Anwendung passender Hardware- und Softwaretechniken für die Gestaltung von effizient funktionierenden Hauptspeicherdatenbanken (vgl. Kian-Lee et al., 2015, S. 35).

In den nachfolgenden Unterabschnitten wird eine Auswahl an technischen Aspekten betrachtet, die dem Verständnis, der Konzeption und der Umsetzung von Main-Memory-Datenbanksystemen förderlich sind.

2.2.1 Fortschritte in der Prozessortechnik

Die Einführung der 64-Bit-Technologie ermöglicht es theoretisch maximal 16 Exbibyte als Arbeitsspeicher zu nutzen. Diese Entwicklung ist von immenser Bedeutung für die Haltung von großen Datenmengen im Hauptspeicher, da im Vergleich die 32-Bit-Computersysteme gewöhnlich maximal 4 Gibibyte adressierbar können. (vgl. Matt, 2012, S. 229)

Die Entwicklung der Mehrkernprozessoren begünstigt die Verwendung von Parallelität in der Datenverarbeitung und ermöglicht somit weitere Leistungssteigerungen. So lassen sich zum Beispiel unter dem Aspekt der horizontalen Partitionierung[1] der Datenbasis die einzelnen Partitionen auf die Kerne verteilen. Anweisungen können dann parallel durchgeführt werden, wodurch eine gleichzeitige und folglich schnellere Datenverarbeitung erfolgen kann (vgl. Bog et al., 2011, S. 333)

2.2.2 NUMA-Architektur

Die NUMA-Architektur (Non-Uniform Memory Access) ermöglicht die Skalierung von Systemen mit vielen Prozessoren und weiter steigenden Hauptspeichermengen.

In der nachfolgenden Abbildung 2.2 wird dargestellt, wie bei dieser Systemgestaltung der gesamte Hauptspeicher aufgeteilt und mit den einzelnen Prozessoren in sogenannte NUMA-Knoten gruppiert wird. Die Einteilung erfolgt derart, dass der jeweils von der physischen Distanz nächste Speicher zu einem Prozessor mit diesem einen NUMA-Knoten bildet.

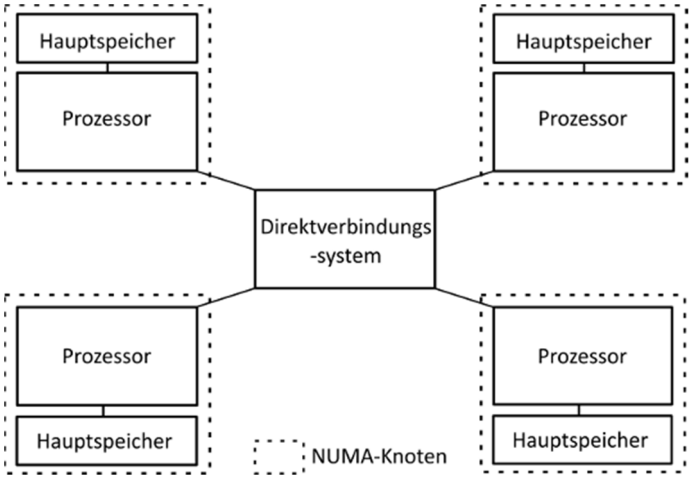

Abbildung 2.3 NUMA-Architektur

Der Teil des Hauptspeichers, der zusammen mit einem Prozessor einen Knoten bildet, stellt aus der Sicht des Prozessors den lokalen Speicher dar[2]. Dieser ermöglicht aufgrund seiner kurzen Distanz zu der CPU eine geringe Zugriffszeit. Jede CPU kann über Direktverbindungen auf den Hauptspeicher der anderen NUMA-Knoten, dem

[1] Partitionierung: Aufteilung einer Menge in nicht leere, disjunkte Teilmengen

[2] Es existieren auch Knoten, die mehrere Prozessoren umfassen und diese greifen gemeinsamen auf den lokalen Speicher zu.

sogenannten Fremdspeicher, zugreifen. Dann sind aber dafür höhere Zugriffszeiten in Kauf zu nehmen. Folglich kann jeder Prozessor den gesamten Hauptspeicher verwenden.

Durch die Verwendung der NUMA-Architektur lässt sich eine Verbesserung der Speicherbandbreite und folglich der Verarbeitung erreichen, wenn die Softwaresysteme derart angepasst werden, dass die CPUs hauptsächlich auf ihren eigenen Speicherbereich zugreifen. Das ist ein Vorteil im Gegensatz zu der UMA (Unifom Memory Access) - Architektur, bei der alle Prozessoren über ein gemeinsames Bussystem auf den gesamten Hauptspeicher zugreifen. Hierbei soll erwähnt werden, dass UMA relativ konstante Zugriffszeiten ermöglicht, bei NUMA die Zugriffszeiten jedoch abhängig sind von der Distanz zwischen Prozessor und angesprochenem Speicher (vgl. Plattner, 2013, S.25f sowie vgl. Kemper und Eickler, 2015, S. 583f).

2.2.3 Datenorganisation in relationalen Datenbanken

In Datenbanksystemen auf dem relationalen Datenmodell werden hauptsächlich zwei Arten der Datenspeicherung in Tabellenform betrachtet: das zeilen- und das spaltenorientierte Layout. Zu diesen existiert noch das hybride Tabellenlayout, das eine Kombination aus den beiden Layouts darstellt (vgl. Plattner, 2013, S. 60-62)

Mit Hilfe der Tabelle 2.1 wird erklärt, worin sich die unterschiedlichen Datenlayouts unterscheiden.

1	Sarah	Kurz	D
2	Frank	Klein	A
3	Dieter	Kante	CH

Tabelle 2.1 Basistabelle

In dem zeilenorientierten Layout werden die Datensätze der Tabellen den Attributen nach aufeinanderfolgend auf dem Datenträger gespeichert, wie es in Abbildung 2.4 dargestellt wird.

1	Sarah	Kurz	D	2	Frank	Klein	A	3	Dieter	Kante	CH

Abbildung 2.4 Zeilenorientierte Speicherung

Diese Art der Speicherung ist vorteilhaft, wenn einzelne Datensätze verarbeitet werden sollen, wie zum Beispiel beim Einfügen und Löschen. Dadurch können diese Operationen auf den zusammenhängenden Attributen sequentiell durchgeführt werden (vgl. Plattner, 2013, S. 61).

Bei festplattenbasierten Datenbanksystemen wirken sich sequenziell durchführbare Zugriffe auf den Datenträger positiv auf die Verarbeitungsgeschwindigkeit aus (vgl. Kemper und Eickler, 2015, S. 212). Abfragen auf einzelnen Attributen jedoch, wie sie bei Analysen häufig vorkommen, oder Updates lassen sich nur langsam durchführen wegen der bedingten Verteilung der Daten bei dieser Form der Speicherung. Dadurch lassen sich unter anderem Cache-Strategien, wie die Verwendung der räumlichen Lokalität mit ihrem positiven Effekt auf die Verarbeitungsgeschwindigkeit, nicht nutzen. (vgl. Plattner, 2013, S. 61)

Bei In-Memory-Datenbanksystemen auf Basis des relationalen Datenmodells wird überwiegend die spaltenorientierte Datenspeicherung verwendet. Bei dieser Form der Datenhaltung werden die Attributwerte der Spalten hintereinander gespeichert, wie in Abbildung 2.5 gezeigt. Daraus ergeben sich Vorteile, die von der zeilenorientieren Speicherung nicht genutzt werden können.

1	2	3	Sarah	Frank	Dieter	Kurz	Klein	Kante	D	A	CH

Abbildung 2.5 Spaltenorientierte Speicherung

Der Cache kann bei Analysen auf einzelnen Attributen die räumliche Lokalität nutzen, da die Attributwerte sequentiell angeordnet sind und Kompressionsverfahren (siehe Unterkapitel 2.2.4), die auf der Wörterbuch-Codierung basieren, erreichen einen besseren Kompressionsfaktor (vgl. Plattner, 2013, S. 61).

Nachteilig wirken sich große Mengen von Lesen- und/oder Schreibzugriffen in Form von vielen oder allen Spalten von Datensätzen aus, da diese in die einzelnen Spalten zerlegt bzw. aus diesen wieder rekonstruiert werden müssen (vgl. Plattner, 2013, S. 92-93). Hier besteht die Möglichkeit durch geeignete Datenstrukturen, wie z.B. Differential Buffer / Delta Buffer bei SAP, die Leistung bei Schreibzugriffen zu verbessern (vgl. Plattner, 2013, S. 167).

Eine dritte Art für das Speicherformat von Daten in relationalen Datenbanken stellt das sogenannte hybride Tabellenlayout dar. Dieses stellt eine Kombination aus dem zeilen- und dem spaltenorientierten Layout dar, bei dem die Vorteile aus beiden Formaten genutzt werden können. Attribute, die einzeln häufig abgefragt werden, können im einspaltigen Format gehalten werden, während Attribute, die oft zusammenhängend verarbeitet werden, als Spaltenverbund gespeichert werden. Es wird darauf hingewiesen, dass die möglichst optimale Kombination der Attribute oft erst während des laufenden Datenbankbetriebes mittels Statistikdaten und Laufzeitalgorithmen gefunden wird und daraufhin eine Reorganisation der Daten erfolgt (vgl. Plattner, 2013, S. 62)

2.2.4 Kompressionsverfahren

Mittels Kompressionsverfahren wird der Speicherplatzverbrauch der Daten auf einem Speichermedium verringert. Durch die Verkleinerung der Speichergröße der Daten lassen sich somit die Übertragungszeiten für Datenmengen reduzieren, was in verbesserten Verarbeitungszeiten resultiert (vgl. Plattner, 2013, S. 37). Darüber hinaus sind die komprimierten Daten in der Verarbeitung nutzbar, wie zum Beispiel beim Sort-Merge-Join (vgl. Plattner, 2013, S. 139-140).

Ein besonders häufig verwendetes Verfahren ist die die Wörterbuch-Codierung, die nachfolgend näher betrachtet wird (die folgenden Abschnitte in diesem Unterkapitel basieren im Wesentlichen auf (Plattner, 2013, S. 37-41)).

Bei der Wörterbuch-Codierung handelt es sich um eine Datenstruktur für Zeichenfolgen, bei der jede Zeichenfolge einen numerischen Wert zur Identifikation, enthält und in dem sogenannten Wörterbuch eingetragen wird. Soll nun eine Zeichenfolge gespeichert werden, so wird geprüft, ob sich diese Zeichenfolge im Wörterbuch befindet. Für den Fall, dass sich diese im Wörterbuch finden lässt, wird anstelle der Zeichenfolge der Identifikationswert gespeichert. Tritt jedoch die Variante ein, dass die Zeichenfolge nicht vorhanden ist, so wird diese neu in das Wörterbuch aufgenommen und erhält einen sie identifizierenden Wert. Anschließend wird anstelle der Zeichenfolge der Identifikationswert gespeichert.
Beim Lesen des Datensatzes wird zu einem passenden Zeitpunkt der Identifikationswert im Wörterbuch nachgeschlagen und dann durch die dazugehörige Zeichenfolge ersetzt, sodass sich der Ursprungsdatensatz wieder korrekt rekonstruieren lässt.

Bei Attributwerten, bei denen es wahrscheinlich ist, dass diese mehrfach vorkommen, bietet es sich an eine Wörterbuch-Codierung vorzunehmen. Denn je mehr Zeichenfolgen codiert werden, desto höher ist der Kompressionsfaktor und desto geringer der benötigte Speicher.

Die Wörterbuch-Codierung kann als Basis für weitere Kompressionsverfahren dienen, wie zum Beispiel der hier nicht weiter behandelten Lauflängenkomprimierung, mit der sich bei sortierten spaltenorientierten Tabellen der Kompressionsfaktor steigern lässt.

2.2.5 Sicherung und Wiederherstellung

Um die Konsistenz und Beständigkeit der Datenhaltung bei Main-Memory-Datenbanksystemen zu gewährleisten, ist es notwendig, dass die Datenbasis des flüchtigen Hauptspeichers regelmäßig gesichert wird. Mittels Wiederherstellungsmaßnahmen lassen sich die Daten nach einem Neustart des Systems, sei dieser nun geplant oder durch einen Fehler herbeigeführt, mittels des Sicherungsmechanismus wieder in den Arbeitsspeicher laden und auf den aktuellen und konsistenten Stand bringen. Dabei werden in der Praxis unterschiedliche Techniken zur Sicherung und Wiederherstellung von Daten in Hauptspeicherdatenbanken angewendet.

Häufig werden Verfahren genutzt, die auf der Basis von Checkpoints in Kombination mit Log-Dateien arbeiten. Diese Checkpoints stellen Speicherpunkte dar, die in periodischen Abständen vollständige Kopien der Datenbasis anlegen. Diese Abbilder sind sogenannte Snapshots (dt.: Schnappschüsse), die exakt den konsistenten Zustand der Daten widerspiegeln, der beim Start des Sicherungsvorgangs vorliegt und diese werden auf einem nichtflüchtigen Datenträger abgelegt. Die Log-Dateien protokollieren die Transaktionen, die während des Snapshot-Vorgangs erfolgreich durchgeführt wurden. Im Fall der Wiederherstellung lässt sich der aktuell gültige Datenbestand in zwei Schritten wiederherstellen:

1. Der zuletzt gesicherte Datenbestand in Form eines Snapshots wird in das Datenbanksystem geladen.
2. Die erfolgreich durchgeführten Transaktionen seit der Erstellung des aktuell gültigen Snapshots werden aus der Log-Datei in die Datenbank übernommen.

Die Kombination aus Snapshot und Log-Datei ermöglicht somit eine Wiederherstellung des aktuell gültigen Zustandes der Datenbank (vgl. Garcia-Molina und Salem, 1992, S.512-513).

Stehen mehrere Rechnersysteme n zur Verfügung, bieten sich Replikationsverfahren an, die den Datenbestand redundant auf den Systemen bereitstellen. Fallen nun bis zu n-1 Computer aus, so können diese bei einem Neustart ihre Daten mittels der Datenbestände der anderen Rechnersysteme auf den aktuellen Stand bringen (Gessert et al., 2016, S. 8).

2.2.6 Parallelisierte Datenverarbeitung

Die nachfolgenden Ausführungen über die unterschiedlichen Formen der Parallelisierung basieren im Wesentlichen auf (Plattner, 2013, S. 117).

Die Nutzung von Parallelität in der Datenverarbeitung erbringt in der Softwareentwicklung Leistungssteigerungen, von denen auch Hauptspeicherdatenbanken profitieren können.

Die Datenparallelität zeichnet sich dadurch aus, dass die Daten in disjunkte Teilmengen, sogenannte Partition, aufgeteilt werden. Auf diesen unterschiedlichen Datenpartitionen lassen sich dann die Operationen parallel ausführen und im Anschluss werden die Resultate zu einem Gesamtergebnis zusammengeführt. Bei dieser Art der Parallelität ist der Grad der Komplexität hoch, da die einzelnen Aktionen auf den Teilmengen koordiniert ablaufen müssen. Dem Aufwand steht gegenüber, dass bei steigender Datenmenge das Maß an Parallelisierung mitwachsen kann, sodass die Datenparallelität hervorragend skaliert.

Bei der Pipelineparallelität werden Operationen eines Ausführungsplans derart verknüpft, dass die auf einer aktuell ausgeführten Operation nachfolgende Weiterverarbeitung bereits auf Teilergebnissen des gegenwärtigen Arbeitsvorganges durchgeführt werden kann. Im Effekt bedeutet das, dass nicht auf das Gesamtergebnis einer Operation gewartet werden muss und freie Rechenkapazitäten mit der Abarbeitung fortfahren können. Somit kann durch die gezielte Hintereinanderschaltung von Verarbeitungsschritten in Form der Pipelineparallelität die benötigte Ausführungszeit für den Gesamtvorgang verringert werden.

Hierbei müssen aber einige Punkte beachtet werden, wie zum Beispiel, dass einige Operationen nicht für Teilmengen einer im Ausführungsplan vorangestellten Operation geeignet sind, da diese Ergebnisse erst weiterleiten können, wenn das Gesamtergebnis der Vorgängeroperation vorliegt. Dies hat zur Folge, dass unterschiedliche Ausführungszeiten von Operationen die Einsparung in der Verarbeitungszeit teilweise erheblich beeinflussen können.

Die Inter-Query-Parallelität zeichnet sich dadurch aus, dass Operationen von verschiedenen Anfragen parallel im Datenbanksystem ablaufen. Bei dieser Art der Query-Parallelität lassen sich durch Untersuchungen der Ausführungspläne der parallel umzusetzen Aktionen identische Teiloperationen ermitteln und somit gemeinsam ausführen.

Im Gegensatz zu der Inter-Query-Parallelität wird bei der Intra-Query-Parallelität eine einzelne Abfrage derart zerlegt, dass möglichst viele Operationen innerhalb dieser parallel ausgeführt werden.

2.2.7 Mehrbenutzer- und Threadsynchronisation

Von Hauptspeicherdatenbanken wird erwartet, dass sie neben dem transaktionalen Betrieb einen hohen Grad der Parallelverarbeitung gegenüber den festplattenbasierten Datenbanksystemen aufweisen. Die angestrebten Ziele dabei sind ein hoher Durchsatz und ein Minimum an Latenzzeit im Datenbankbetrieb. Um diese zu erreichen werden neue Verfahren angewendet, die den Aufwand für Locking und Latching (siehe 2.1.2) reduzieren oder gänzlich entfernen. Dabei variieren die Techniken zwischen den Herstellern. So existieren Methoden, die beispielsweise auf der Basis von Partitionen und serieller Transaktionsausführung auf diesen Partitionen arbeiten oder es kommen optimistische Sperrverfahren, wie MVCC, zum Einsatz (vgl. Larson und Levandovski, 2016, S. 1610).

2.3 Auswirkungen auf die Praxis

Die derzeitige betriebliche IT-Landschaft teilt die Datenbanksysteme in zwei Anwendungsbereiche ein: Zum einen in die Online Transaction Processing (OLTP)-Systeme, die für die effiziente Transaktionsverarbeitung ausgelegt sind. Zum anderen in die Online Analytical Processing (OLAP)-Systeme, die für intensive Analysen genutzt werden. Die Einteilung wird aufgrund der Annahme vorgenommen, dass die OLTP-Nutzung schreibintensiv sei, während die OLAP-Nutzung fast ausschließlich aus lesenden Zugriffen besteht. Für die Synchronisation der Daten zwischen dem OLAP- und dem OLTP-System sind aufwendige Extract-Transform-Load (ETL)-Prozesse notwendig (vgl. Plattner, 2013, S. 15-16).

Plattner argumentiert in (vgl. Plattner, 2013, S.16-17) für eine Zusammenlegung von OLAP- und OLTP-Systemen beim Einsatz von Hauptspeicherdatenbanken. So sind bei Analysen von Kundensystemen herausgekommen, dass sich OLTP- und OLAP-Systeme ähnlicher sind als angenommen. Die Anzahl von Einfüge-Vorgängen und die Abfrageraten von Daten in OLTP-Systemen sind ein wenig höher ist als bei OLAP-Systemen. Im Hinblick auf Aktualisierungen von Daten führt Plattner an, dass in High-Tech-Unternehmen etwa 88% der gespeicherten Daten in OLTP-Systemen nie aktualisiert werden, wobei andere Branchen noch niedrigere Änderungsraten aufweisen.

Durch die Kombination der OLTP- und OLAP-Arbeitslast auf Basis eines hauptspeicherbasierten Datenbanksystems ist es möglich sowohl transaktionale als auch analytische Verarbeitungen auf demselben Datenbestand durchzuführen. Dies führt dazu, dass nur noch ein Ort der Datenspeicherung notwendig ist, bei dem sämtliche Daten stets aktuell sind. Darüber hinaus wird die Verwaltung der Daten vereinfacht, da ETL-Prozesse überflüssig sind und aggregierte Daten, sowie materialisierte Views nicht mehr benötigt werden. Die benötigten Daten für die Analysen werden einfach nach Bedarf bereitgestellt (vgl. Plattner, 2013, S. 16-17).

In (vgl. Loos et al., 2011, S.385) weist Zeier auf den Aufwand der Anpassung der Softwareentwicklung an Hauptspeicherdatenbanken hin. Bestehende Software muss nach und nach neu bzw. umgeschrieben werden. Die Entwicklung neuer Anwendungen sollte bereits mit der Ausrichtung auf Hauptspeicherdatenbanken geschehen. Darüber hinaus sei zu prüfen, ob vorhandene Konzepte, die bisher nicht oder nur mit Spezialsystemen umsetzbar waren, mit Hauptspeicherdatenbanken realisierbar sind.

Die Schwierigkeit bei Hauptspeicherdatenbanken sei die Vorhersage, um wie viel sich die Leistung gegenüber festplattenbasierten Datenbanksystem verbessert. Die Prognose wird durch die verwendeten Daten und den Einsatzbereich bestimmt. Die Aussagen reichen dabei von ein wenig bis zu viel schneller und von Werten zwischen 50 und 100 bis hin zu 10.000 (vgl. Savage, 2014, S. 15).

3 Ausgewählte Datenbanksysteme

Die für die Laufzeitanalyse ausgesuchten, auf dem relationalen Datenbankmodell basierenden Hauptspeicherdatenbanksysteme TimesTen, SQL Server 2014 und 2016, EXASOL und VoltDB werden in diesem Kapitel näher betrachtet. Der Fokus liegt hierbei auf folgenden Aspekten:

- Ausrichtung des Datenbanksystems (OLTP oder OLAP),
- Bezug zu welchem SQL-Standard,
- Datenhaltung (Zeilen- oder Spaltenform, Kompressionsmöglichkeiten),
- Transaktionsbetrieb (ACID-fähig, Isolationsstufen),
- Sicherung und Wiederherstellung, sowie
- Möglichkeiten im Rechnerverbund (beispielsweise Replikation).

Darüber hinaus werden auch datenbankspezifische Funktionen und Eigenschaften tiefer gehend betrachtet, wenn diese besonders im Vergleich zu den jeweils anderen Datenbanksystemen sind und dem Verständnis dienen.

Die einzelnen Datenbanksystem-Unterkapitel enden mit der Begründung der Auswahl des jeweiligen Datenbanksystems. Den Abschluss dieses Kapitels bildet eine Zusammenfassung der Hauptspeicherdatenbanksysteme in Tabellenform, die die Systeme auf Basis der oben aufgeführten Aspekte gegenüberstellt.

3.1 TimesTen

Oracle bietet mit TimesTen eine Hauptspeicherdatenbank an, die sich nicht nur für den Transaktionsbetrieb eignet, sondern auch für die Durchführung analytischer

Prozesse genutzt werden kann. Neben der Verwendung als primäres Datenbanksystem besteht die Möglichkeit, dieses System als Cache für Transaktionen zu einem bereits bestehenden Oracle Datenbanksystem zu integrieren (vgl. Tirthankar et al., 2013, S. 6).

Die Speicherung der Daten erfolgt auf Basis des zeilenorientieren Tabellenlayouts. Der für die Datenhaltung zuständige TimesTen Storage Manager beinhaltet die Option die Wörterbuch-Codierung zu verwenden, um den Speicherverbrauch zu senken (vgl. Tirthankar et al., 2013, S. 8).

Das Datenbanksystem beinhaltet einen großen Teil des PL/SQL-Umfangs von Oracle Database 11.2.0.2 (vgl. Oracle, 2014c, S.1-1). Dementsprechend ist die Unterstützung des SQL 2008 Standard gegeben (vgl. Oracle, 2016, S. C-1).

Die Unterstützung des Transaktionsbetriebes nach dem ACID-Prinzip wird von TimesTen gewährleistet (vgl. Tirthankar et al., 2013, S. 7). Transaktion laufen standardmäßig in der Isolationsstufe read committed ab, weil diese Einstellung die beste Leistung bietet. Die Stufe lässt sich aber auch auf serializable einstellen, wenn es die Anwendung erfordert (vgl. Tirthankar et al., 2013, S. 8-9).

Die Sicherung des Datenbestandes wird durch ein besonderes Checkpoint-Verfahren sichergestellt. Die Aufzeichnungen der erfolgreich durchgeführten Transaktionen werden in TimesTen durch einen Logpuffer durchgeführt. Dieser beinhaltet einen multithread-basierten Logmechanismus. Die Transaktionen werden in der Art gespeichert, dass der Logmechanismus den Logpuffer in mehrere Partitionen unterteilt, sodass Transaktionen nicht nacheinander, sondern parallel gespeichert werden können. Die logisch sequentielle Reihenfolge der Transaktionen wird bei der Wiederherstellung garantiert. Die Abbildung des jeweils gültigen Zustands der Datenbank wird mittels zweier Checkpoint-Dateien ermöglicht. Sobald eine Checkpoint-Datei eine erfolgreich durchgeführte Sicherung darstellt, wechselt der Checkpoint-Mechanismus auf die andere Datei. Somit kann garantiert werden, dass eine korrekte, dauerhafte Kopie des Datenbestandes erfolgt ist. (vgl. Tirthankar et al., 2013, S. 8)

Für den Bereich der Hochverfügbarkeit bietet TimesTen mehrere Möglichkeiten. Eine Möglichkeit besteht darin, die Logeinträge der Transaktionen derart zu replizieren, dass bei erfolgreicher Durchführung der Transaktionen diese an die anderen Datenbanksysteme übermittelt werden. Eine striktere Variante besteht in der Nutzung der sogenannten 2-safe replication-Verfahrens. Bei der Anwendung gelten

lokal durchgeführten Transaktionen erst als erfolgreich, wenn diese bei den anderen Datenbanksystemen in den Logpuffer geschrieben und bestätigt wurden. Das ermöglicht die Vermeidung von festplattenbasierten Sicherungen, da die Daten redundant in den verbundenen gleichartigen Datenbanksystemen vorhanden sind. Zu beachten ist hierbei, dass bei einem Totalausfall von allen Systeme der Datenbestand verloren ist (vgl. Tirthankar et al., 2013, S. 9).

Die vorwiegend verwendete Art der Replikation ist die Verwendung einer Active-Standby-Kombination. Bei dieser Form werden sämtliche Schreibzugriffe nur auf der als Active-bezeichneten Datenbank durchgeführt. Diese wiederum gibt die Änderungen an die Standby-Systeme weiter. Die Standby-Systeme erlauben nur Lesezugriffe, können aber beim Ausfall des Active-Datenbanksystems dessen Rolle übernehmen (vgl. Oracle, 2014a, S. 17).

Die am Anfang dieses Unterkapitels ausgesagte Nutzungsmöglichkeit über TimesTen für sowohl den OLTP- als auch den OLAP-Bereich bestimmt die Auswahl dieses Systems. Deshalb soll dieses Datenbanksystem in der Leistungsanalyse einem Vergleich mit reinen OLTP- bzw. OLAP-Systemen unterzogen, um zu ermitteln, ob die Hybrid-Nutzung sinnvoll möglich ist.

3.2 SQL Server 2014 und 2016

Microsoft bietet mit SQL Server 2014 erstmals die Möglichkeit das hauseigene Produkt SQL Server als Hauptspeicherdatenbank zu betreiben. Zu diesem Zweck wurde die auf den OLTP-Anwendungsbereich ausgerichtete Hekaton Engine entwickelt, die erstmalig in SQL Server 2014 integriert wurde (vgl. Microsoft, In-Memory OLTP (Arbeitsspeicheroptimierung), 2016a).

Bei beiden Datenbanksystemen besteht im laufenden Betrieb die Möglichkeit sowohl den Hauptspeicher als auch die Festplatte als Datenspeicher gleichzeitig zu nutzen. Dadurch ist es möglich die leistungskritischen Daten im Hauptspeicher bereitzustellen, während weniger kritische Daten auf der Festplatte gespeichert werden (vgl. Larson et al., 2013, S. 34). Im Arbeitsspeicher erfolgt die zeilenorientierte Datenhaltung in den Tabellen (vgl. Microsoft, Tabellen- und Zeilengröße in speicheroptimierten Tabellen, 2016a). Weder in SQL Server 2014, noch in SQL Server 2016 ist es möglich, die im Hauptspeicher gehaltenen Daten zu komprimieren (vgl. Microsoft, Transact-SQL Constructs Not Supported by In-Memory OLTP, 2016c).

Veränderungen am Datenbestand werden nach dem Insert-Only-Ansatz durchgeführt, bei dem zu jeder Änderung ein neuer Datenbankeintrag erfolgt. Die in der Hekaton Engine enthaltene Garbage Collection sorgt dafür, dass nicht mehr gültige Einträge aus der Datenbank entfernt werden. Dadurch wird der Speicherverbrauch auf ein Minimum reduziert und eine effiziente Datenorganisation im Hauptspeicher gewährleistet (vgl. Diaconu et al., 2013, S. 1251).

Die Datenbanksprache für den Festplatten-, als auch für den Hauptspeicherbetrieb ist die Microsoft eigene SQL-Erweiterung T-SQL. Dabei ist bei der Verwendung von T-SQL für den Hauptspeicherbetrieb zu beachten, dass nur eine Teilmenge der Befehle zur Verfügung steht, der beim festplattenbasierten Datenbankeinsatz möglich ist. Abfragen können sowohl die auf regulären Festplatten als auch die im Hauptspeicher gehaltenen Daten in Kombination referenzieren. In T-SQL gespeicherte Prozeduren, die nur die im Hauptspeicher gehaltenen Tabellen referenzieren, lassen sich in nativen Maschinencode kompilieren. Die so erstellten Verarbeitungsfolgen bieten weitere Leistungssteigerungen und stellen die schnellste Art da, Daten abzufragen und zu modifizieren (vgl. Larson et al., 2013, S. 34).

Die Transaktionen im Hauptspeicherbetrieb sind ACID-konform (vgl. Weiner und Levin, 2014, S. 3). Für die Transaktionsabläufe werden die Isolationsstufen snapshot isolation, read committed und serzializable angeboten. Dabei kommt ein Multiversion Concurrency Control-Verfahren auf Basis von Zeitstempeln zum Einsatz. Das Verfahren ermöglicht blockierungsfreie Transaktionsausführungen und für schreibende Zugriffe wird die Serialisierbarkeit garantiert (vgl. Diaconu et al., 2013, S. 1248). Zusätzlich finden in Hekaton nur Datenstrukturen Verwendung, die keine Sperrmechanismen für die parallele Datenverarbeitung einsetzen (vgl. Larson et al., 2013, S. 35).

Für die primär im Hauptspeicher gehaltenen Daten werden weder in SQL Server 2014, noch in SQL Server 2016 Replikationsmöglichkeiten angeboten (vgl. Microsoft, Transact-SQL Constructs Not Supported by In-Memory OLTP, 2016c).

Die Dauerhaftigkeit von Transaktionen und Daten wird durch die Erstellung von Checkpoint- und Logdateien auf nichtflüchtigen Datenträgern sichergestellt. Eine erfolgreich durchgeführte Transaktion wird mitsamt den durchgeführten Änderungen in das Logverzeichnis aufgenommen. Die Checkpoints werden auf Basis des Logverzeichnisses, und nicht auf dem Datenbestand im Hauptspeicher, durchgeführt (vgl. Diaconu et al., 2013, S. 1250).

In SQL Server 2016 steht die Hekaton Engine mit einem erweiterten Funktionsumfang für den Hauptspeicherbetrieb zur Verfügung. Zu den wichtigsten Unterschieden bezüglich des Funktionsumfanges der Hekaton Engine in SQL Server 2016 gegenüber der Version in SQL Server 2014 zählen (vgl. De Bruin, In-Memory OLTP – what's new in SQL2016 CTP3, 2015):

- Foreign-Key-Unterstützung,
- Trigger,
- Check-Einschränkung, sowie
- Aufhebung des Limits der Gesamtmenge an Daten in hauptspeicher-optimierten Tabellen.

Die Wahl von SQL Server 2014 basiert auf dem Fakt, dass es sich dabei um das erste Datenbanksystem von Microsoft handelt, das neben der festplattenbasierten Speicherung auch die Nutzung des Hauptspeichers als primären Speicherort der Daten in einem einzigen Datenbanksystem anbietet. Des Weiteren bietet die OLTP-Ausrichtung die Möglichkeit der Gegenüberstellung zu OLAP-Datenbanksystemen in der Leistungsanalyse. Die zusätzliche Auswahl von SQL Server 2016 erfolgt aufgrund des unterschiedlichen Funktionsumfangs der zugrundeliegenden Engine des Hauptspeicherdatenbanksystems. Die beiden Versionen bieten die Gelegenheit die Unterschiede in der Leistung untereinander, als auch gegenüber den anderen Datenbanksystemen zu untersuchen.

3.3 EXASOL

Die EXASOL AG stellt mit EXASOL ein auf Analysen ausgerichtetes Hauptspeicherdatenbanksystem bereit (vgl. EXASOL AG, 2012, S. 3). Bereits hier seien zwei Besonderheiten dieses Systems zu erwähnen, auf die nachfolgend in einigen Punkten eingegangen wird: Zum einen wird EXASOL mit dem hauseigenen, auf Parallelität ausgerichteten Betriebssystem ausgeliefert (vgl. EXASOL AG, 2015a, S. 5). Zum anderen ist der Automatisierungsgrad dieses Datenbanksystems gegenüber den anderen Hauptspeicherdatenbanksystemen hoch (vgl. EXASOL AG, 2015a, S. 6).

Die Datenhaltung in EXASOL erfolgt in Form des spaltenorientierten Tabellenlayouts (vgl. EXASOL AG, 2012, S. 6f). Dabei kommt die in EXAClusterOS befindliche Speicherverwaltungssoftware EXAStorage zum Einsatz (vgl. EXASOL AG, 2015a, S. 5). Das Datenbanksystem wendet automatisch Kompressionsverfahren für die zu speichernden Daten an, die darüber hinaus auf die Datentypen angepasst sind (vgl. EXASOL AG, 2012, S. 6).

EXASOL bietet einen Großteil des SQL 2008 Standards an (vgl. EXASOL AG, 2015b, S. 421). Durch die Verwendung der sogenannten Scripting-Programmierung lassen sich SQL-Befehle kombinieren (vgl. EXASOL AG, 2015b, S. 266). Zusätzlich werden UDF-Skripte (UDF: User Defined Function) angeboten, die die Erstellung benutzerdefinierter Analyse-, Verarbeitungs- und Generierungsfunktionen und Ausführung dieser in einem Cluster ermöglichen (vgl. EXASOL AG, 2015b, S. 243).

Die Transaktionen werden in EXASOL ACID-konform durchgeführt (vgl. EXASOL AG, 2015b, S. 289). Dabei wird standardmäßig die Isolationsstufe serializable genutzt, die zudem die einzige ist (vgl. EXASOL AG, Transaction System, 2014).

Der Verbund von mehreren EXASOL-Systemen zu einem Parallelsystem ermöglicht die Nutzung der massiv-parallelen Datenverarbeitung. Die Parallelverarbeitung wird durch zwei Arten der Pipelineparallelität unterstützt. Zum einen werden Threads und Zwischenergebnisse mittels einer Execution Pipeline (dt.: Ausführungspipeline) verwaltet und die interne Parallelausführung des Systems wird verbessert. Zum anderen können Zwischenergebnisse an die Pipelines der anderen Knoten geschickt werden, was die verteilte parallele Datenverarbeitung effizienter gestaltet (vgl. EXASOL AG, 2016, S. 6).

Die Gestaltung als Parallelsystem bietet nicht nur Vorteile bei der Verarbeitung, sie bietet zudem eine Sicherheitsfunktion beim Ausfall eines der Datenbanksysteme im Verbundsystem. Im Fehlerfall kann ein als Hot Standby-Server definiertes System die Funktion eines ausgefallenen Servers übernehmen (vgl. EXASOL AG, 2016, S.13). In Kombination mit redundanter Datenhaltung durch Replikationen und dateibasierten Backups, sowie einer gleichmäßigen Verteilung der Daten wird die Funktionsfähigkeit des Rechnerverbundes sichergestellt. (vgl. EXASOL AG, 2016, S.13)

Für Einzelsysteme besteht die Möglichkeit der Datensicherung in Form von dateibasierten Backups, die entweder im System oder extern gespeichert werden können (vgl. EXASOL AG, 2016, S.13).

Durch den hohen Automatisierungsgrad führt das Datenbanksystem selbstständig Optimierung durch, ohne dass für den Benutzer die Möglichkeit besteht in die Tuning-Maßnahmen einzugreifen (vgl. EXASOL AG, 2012, S. 8). Zusätzlich zu der Funktionalität der automatisch durchgeführten Kompression ist in EXASOL ein sogenannter Query Optimizer enthalten. Dieser arbeitet autonom auf der Basis von zur Laufzeit gesammelten Statistiken und bezieht die vorhandene Einzel- bzw. Parallelsystemarchitektur in die Datenverarbeitung mit ein (vgl. EXASOL AG, 2016, S.

8). Beispielsweise werden Indizes automatisch vom Query Optimizer verwaltet und dieser kann Replikationen im Verbundsystem anlegen, wenn diese zu weiteren Verbesserungen in der Verarbeitung der Daten führen (vgl. EXASOL AG, 2015a, S. 6). Die Datenverarbeitungsschritte werden derart geplant, dass sie möglichst unabhängig auf den einzelnen Knoten ablaufen, sodass der Koordinationsaufwand auf ein Minimum reduziert wird (vgl. EXASOL AG, 2016, S. 8).

EXASOL wurde auf Basis der Platzierung im TPC-H-Benchmark[1] ausgewählt (vgl. TPC, TPC-H - Top Ten Performance Results Version 2 Results, 2017). In diesem Benchmark belegt die auf OLAP-Nutzung ausgelegte Datenbank jeweils den ersten Platz in den Kategorien mit Datenvolumen zwischen 100 und 100.000 Gigabyte.

3.4 VoltDB

Die VoltDB Inc. hat mit VoltDB ein Hauptspeicherdatenbanksystem für den OLTP-Anwendungsbereich entwickelt (vgl. VoltDB Inc., 2016b, S. 1). Dabei besteht dieses System aus einer auf Parallelität ausgerichteten Architektur, die auf dem Konzept sogenannter Virtueller Knoten basiert. Diese stellen die Ausführungseinheiten in der Datenverarbeitung dar (vgl. VoltDB Inc., 2016a, S. 3).

Im Kern besteht jeder virtuelle Knoten aus den ihm zugeteilten Daten aus den Tabellen, einem Thread und einer Queue. Die Zuordnung der Daten erfolgt auf zwei mögliche Arten. Zum einen können die Tabellen den Knoten in Form logisch disjunkter Teilmengen, sogenannter Partitionen, zugeteilt werden. Bei häufig zugegriffenen und/oder großen Tabellen trägt das zu einer effizienten Verarbeitung bei. Zum anderen können kleine Tabellen auf mehrere Knoten repliziert werden, was die Ausführung von Lesezugriffen verbessert. Der Thread verrichtet die Datenverarbeitung unter Verwendung der Queue. Die Queue beinhaltet die einzelnen, auszuführenden Verarbeitungsschritte und stellt die Reihenfolge dieser sicher (vgl. VoltDB Inc., o.J., S.3).

Jede Ausführungseinheit arbeitet nur auf den ihr zugeteilten Daten. Dadurch kann in der Datenverarbeitung auf Sperrmechanismen verzichtet werden, wodurch der Multithread-Verwaltungsaufwand auf ein Minimum reduziert (vgl. VoltDB Inc., 2016a, S. 4).

[1] Der TPC-H-Benchmark vergleicht Datenbanksysteme aufgrund ihres Leistungsvermögens im Decision-Support-Bereich.

Die Datenhaltung in VoltDB erfolgt auf Basis des zeilenorientierten Tabellenlayouts. In diesem Datenbanksystem besteht nicht die Möglichkeit, die Daten im Hauptspeicher zu komprimieren.

Der Sprachumfang von VoltDB entspricht einem Großteil von SQL 92. (vgl. VoltDB Inc., o.J., S. 3) Zusätzlich werden über diesen hinaus Erweiterungen, z.B. für räumliche Anwendungen, angeboten (vgl. VoltDB Inc., 2016a, S. 5). Der Transaktionsbetrieb wird unter Verwendung von Stored Procedures (dt.: gespeicherte Prozeduren) durchgeführt (vgl. VoltDB Inc., 2016b, S. 2). Die ACID-Fähigkeit wird für sämtliche Stored Procedures garantiert, da sie entweder gänzlich durchgeführt oder rückgängig gemacht werden (vgl. VoltDB Inc., 2016b, S. 30).

Im Datenbankbetrieb werden Transaktionen, die sich nur auf eine Partition beziehen, von der entsprechend passenden Ausführungseinheit verarbeitet. Für Transaktionen, die mehrere Partitionen betreffen, übernimmt eine Ausführungseinheit die Organisation der Verarbeitung. Diese Einheit verteilt die Ausführungspläne auf die zu den Partitionen gehörenden Knoten, sammelt die Ergebnisse ein und schließt die Transaktionen ab (vgl. VoltDB Inc., 2016b, S. 2-3).

VoltDB bietet die Möglichkeit, mehrere einzelne Datenbanksysteme als Cluster zu organisieren. Für den Bereich der Hochverfügbarkeit bedeutet dies, dass bei einem Ausfall eines Systems automatisch ein anderes dessen Funktion übernimmt und somit die Funktionsfähigkeit des Gesamtsystems gewährleistet ist. Dazu wird das sogenannte K-Safety-Verfahren genutzt. Dabei werden k-Replikationen im Cluster erzeugt, sodass die Ursprungspartition insgesamt (k+1)-mal vorhanden ist. Server, die nach ihrem Ausfall wieder dem Cluster hinzugefügt werden, stellen, sofern möglich, die ihr zugehörigen Partitionen wieder her. Im Anschluss an die Wiederherstellung nehmen die Server wieder am Clusterbetrieb teil (vgl. VoltDB Inc., 2016b, S. 77-78).

Neben der Möglichkeit durch Replikation den Datenbestand sicherzustellen, können Sicherung und Wiederherstellung der in VoltDB gespeicherten Daten auf Basis von Checkpoints mittels dem sogenannten Command-Logging durchgeführt werden. Die Besonderheit bei dem eingesetzten Verfahren ist, dass nicht die Auswirkungen der durchgeführten Transaktionen im Log protokolliert werden, sondern der Aufruf dieser. Dies hat zur Folge, dass die Logdateien ein Minimum an Größe aufweisen und der I/O-Zugriff reduziert wird. (vgl. VoltDB Inc., 2016a, S. 4)

Die Entscheidungsgrundlage für die Auswahl von VoltDB bildet die Architektur auf dem Konzept der virtuellen Knoten, die ohne Sperrmechanismen auskommt. In

diesem Zusammenhang soll ermittelt werden, wie sich diese Systemgestaltung auf die Datenbankleistung auswirkt.

3.5 Kurzfassung der Datenbanksysteme

Die nachfolgende Tabelle stellt in vereinfachter, kompakter Darstellung die in den Abschnitten 3.1 bis 3.4 vorgestellten Hauptspeicherdatenbanksysteme zusammen, um die Vergleiche zwischen den einzelnen Merkmalen übersichtlich zu gestalten.

	TimesTen	**SQL Server 2014**	**SQL Server 2016**	**EXASOL**	**VoltDB**
Ausrichtung	OLTP und OLAP	OLTP	OLTP	OLAP	OLTP
Bezug zu SQL-Standard	SQL 2008	Keine Angabe	Keine Angabe	SQL 2008	SQL 92
Hersteller-spezifische Erweiterung	PL/SQL	T-SQL	T-SQL	SCRIPT, UDF	Stored Procedure auf Java-Basis
Datenorganisation	Zeilenform	Zeilenform	Zeilenform	Spaltenform	Zeilenform
Kompression möglich	Ja	Nein	Nein	Ja	Nein
ACID	Ja	Ja	Ja	Ja	Ja
Transaktionsstufen	read committed, serializable	snapshot isolation, read committed, serializable	snapshot isolation, read committed, serializable	serializable	serializable
Sicherung und Wiederherstellung	Checkpoint, Replikation	Checkpoint	Checkpoint	Datei, Replikation	Checkpoint, Replikation
Rechnerverbund	Replikation, 2-safe Replication, Active-Standby-Kombination, Cache-Funktion	-	-	Hot-Standby, Replikation, Massive Parallelverar-beitung	K-Safety-Verfahren, Replikation

Tabelle 3.1 Kurzfassung der Datenbanksysteme

4 Laufzeitanalyse

Die in dem dritten Kapitel vorgestellten Hauptspeicherdatenbanksysteme werden in diesem Kapitel im praktischen Betrieb untersucht. In Form von Tests[1] soll ermittelt werden, wie die Leistung der Datenbanksysteme in unterschiedlichen Situationen ausfällt. Die Grundlage für die einzelnen Tests, sowie der Auswertungen dieser, bilden die folgenden Zielstellungen:

1. Die Auswirkung von Datenkompression, sofern diese verfügbar ist, auf die Verarbeitungsgeschwindigkeit der einzelnen Datenbanksysteme untersuchen,
2. bestimmen, ob die Trennung von OLTP- und OLAP-Systemen (noch) sinnvoll ist, sowie
3. die Untersuchung der Stärken und Schwächen der Datenbanksysteme.

Dieses vierte Kapitel ist gegliedert in vier aufeinander aufbauende Teile. Der Aufbau der Versuche stellt das erste Unterkapitel dar. In diesem werden die den Tests zugrundeliegenden Gegebenheiten beschrieben. Den zweiten Teil bilden die Durchführungen der Versuche. Diese werden ausführlich beschrieben in Konzeption und dem angestrebten Nutzen. Die Auswertungen der Versuche sind Bestandteil des dritten Unterkapitels. Das Fazit fasst die wichtigsten Erkenntnisse im Hinblick auf die eingangs gegebenen Zielstellungen zusammen und stellt den letzten Teil des vierten Kapitels dar.

[1] Test wird hier als Synonym zu Versuch und Experiment verwendet und umgekehrt.

4.1 Aufbau der Versuche

Um die Tests nachzuvollziehen und um die Abläufe reproduzierbar zu machen, werden die Voraussetzungen und die durchgeführten Schritte beschrieben. Den Beginn bildet die Unterteilung der Versuche anhand der grundlegenden Datenbankoperationen INSERT, SELECT, UPDATE und DELETE. Mit Hilfe dieser Einteilung erhalten die Tests zum einen eine Rahmenstruktur in Bezug auf die zu untersuchenden Funktionalitäten. Zum anderen ist die Kategorisierung der später erfolgenden Gegenüberstellung der Datenbanksysteme dienlich, sodass im Fazit ersichtlich wird, für welche Datenbankoperationen bzw. für welchen Art der Verwendung sich ein Datenbanksystem eignet oder nicht eignet. Darauf erfolgt eine Beschreibung des auf einem Warenwirtschaftssystem basierende Datenschema und der verwendeten Testdaten. Anschließend werden der softwaretechnische Testablauf in einer von einer Programmiersprache abstrahierten Form sowie die Messungen näher beschrieben. Den Abschluss bilden die Konfigurationen der verwendeten Hard- und Software.

4.1.1 Beschreibung der Testarten

Auf Basis einer zweistufigen Kategorisierung werden die Arten der durchzuführenden Tests beschrieben, sodass eine übersichtliche Rahmenstruktur für die Durchführung der Versuche und anschließender Analyse vorhanden ist. Die ausführliche Beschreibung der einzelnen Tests erfolgt in Unterkapitel 4.2.

Die erste Stufe der Einteilung bilden die vier grundlegenden SQL-Datenbank-operationen INSERT, SELECT, UPDATE und DELETE, die alle unabhängig voneinander verwendet werden. Das bedeutet, dass beispielsweise keine Subqueries bei den zu testenden Datenbankoperationen untereinander existieren, keine SELECT…INTO-Anweisungen oder unterschiedliche Datenbankoperationen bei dem jeweils auszuführenden Testfall vorhanden sind, außer sie dienen der Vorbereitung eines bestimmten Datenbankzustandes.

Die Nutzung der Testtabellen bildet die zweite Einteilungsstufe. Hierbei erfolgen zum einen Tests, die nur auf einer Tabelle basieren. Diese ermöglichen es zu bestimmen in welchem Umfang die einzelnen Datenbanksysteme bezüglich der verwendeten Datenbankoperationen skalieren. Als Ermittlungsgrundlage für die Skalierung dienen eine wachsende, auf der Basis 10 potenzierten Anzahl an Testdatensätzen und der damit verbundenen, benötigten Zeit für den entsprechenden Datenverarbeitungs-prozess, sowie die Menge an belegtem Hauptspeicher der durch die Testdaten entstehenden Datensätze.

Zum anderen existieren Versuche, die sämtliche Tabellen des Testdatenschemas nutzen. Bei diesen Tests wird keine Skalierung gemessen wird. Begründet wird dies mit der Tatsache, dass bei mehreren Tabellen die Schwierigkeit besteht, den Einfluss der Datensätze jeder einzelnen Tabelle auf die Skalierung korrekt zu ermitteln.

Für jede der vier grundlegenden SQL-Datenbankoperationen sind Versuche vorhanden, die die Skalierung auf Basis der Nutzung einer Tabelle testen. Jedoch werden nur bei den Versuchen, die in die Kategorie INSERT oder SELECT eingeteilt worden sind, auch Durchläufe durchgeführt, die alle Tabellen des Testdatenschemas nutzen.

4.1.2 Datenschema

Das in Abbildung 4.1 dargestellte Schema für die zu testenden Datenbanksysteme besteht aus einer vereinfachten Umsetzung eines Warenwirtschaftssystems. Dieses Datenbankschema ermöglicht zum einen das Testen von OLTP-Abläufen auf Basis von kleinen, zusammenhängenden Datenmengen, beispielsweise in Form von Anlegen von Kunden oder Bestellungen durch INSERT-Operationen. Zum anderen lassen sich auch lang andauernde analytische Abfragen durchführen, wobei hier exemplarisch die Auswertung der meistverkauften Artikel, zu denen man sich die Lieferanten ausgeben könnte zwecks Mengenrabatte beim Einkauf, genannt sei.

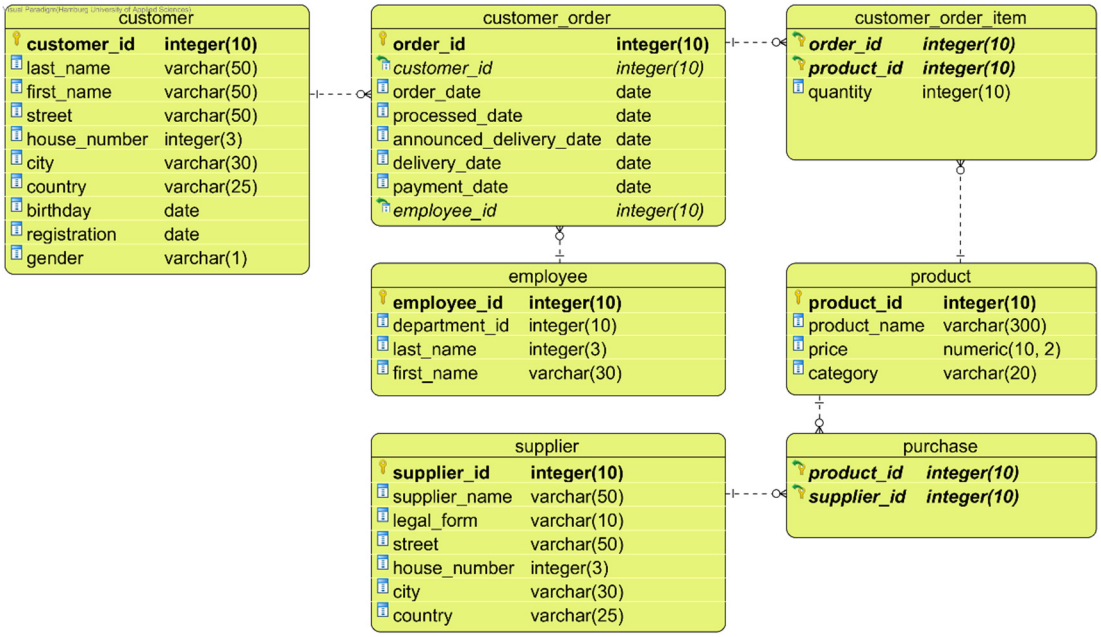

Abbildung 4.1 Datenschema

Ein weiterer wichtiger Punkt für dieses Schema ist die Möglichkeit des Testens von Auswirkungen des Datenlayouts auf die Verarbeitungsgeschwindigkeit. Es lassen sich Abfragen und Ablauffolgen erstellen, die nur wenige Spalten umfassen, wovon vor allem spaltenorientierte Datenbanksysteme profitieren können. Bei der Verarbeitung von mehreren Spalten einer Tabelle beziehungsweise von mehreren Attributen eines Datensatzes profitieren in der Regel die zeilenorientierten Hauptspeicherdatenbanken.

Dieses Datenbankschema bietet somit die Möglichkeit, neben OLTP- und OLAP-Nutzung auch die Auswirkungen der zeilen- bzw. spaltenorientierten Datenhaltung bei Hauptspeicherdatenbanksystemen zu testen und eignet sich demzufolge für die Testfälle.

Bei der Umsetzung des Schemas sind zwei Besonderheiten zu erwähnen: Erstens ist es nicht möglich in VoltDB und im Hauptspeicherbetrieb von SQL Server 2014 Foreign Keys zu verwenden, da diese Funktionalität nicht vorhanden ist. Zweitens wird das Schema in TimesTen in drei Ausführungen umgesetzt. Die erste Ausführung besteht in der Umsetzung ohne Kompressionseinstellungen bezüglich der Daten. Die zweite und dritte Ausführung erfolgen mittels der Kompressionsangabe der Spalten, dessen Daten komprimiert werden sollen. Dabei unterscheiden die beiden sich dadurch, wie die Spalten komprimiert werden. In der zweiten Ausführung passt das Datenbanksystem die Kompression anhand der jeweils eingefügten Daten grundsätzlich an 4 Bytes an. Die dritte Umsetzung erfolgt durch die Angabe eines Maximalwertes, der bestimmt wie viele unterschiedliche Ausprägungen höchstens komprimiert werden. Für weitere Details wird auf den Anhang verwiesen.

4.1.3 Testdaten

Die Grundlage für die Testdaten bilden Textdateien, deren Zeichenfolgen in den Zeilen die Datenbasis bilden. Diese Textdateien werden eingelesen und selbstentwickelten Datengeneratoren zur Verfügung gestellt. Die Generatoren erstellen pseudozufällig unter Verwendung eines Seed-*Wertes*[1] Testdaten-Objekte, die die programmtechnische Abbildung der einzufügenden Datensätze darstellen. Die so erstellten Testdaten-Objekte stellen ihre Daten anschließend den Objekten, die für die auszuführenden Datenbankoperationen zuständig sind, zur Verfügung, sodass diese ihren Part ausführen können.

[1] Die Verwendung des Seed-Wertes ermöglicht es, immer wieder die gleichen Datensätze zu generieren, sodass für alle Datenbanksysteme die gleichen Testdaten erstellt werden.

Die in mehreren Textdateien enthaltenen Zeichenfolgen setzen sich wie folgt zusammen:

- 1500 männliche und 1500 weibliche Vornamen,
- 1500 Nachnamen,
- 500 Firmennamen,
- 20 Rechtsformen für Unternehmen,
- 2600 Städte,
- 100 Länder,
- 400 Straßennamen und
- circa 547.000 Produkte mitsamt den Preisen.

Anzumerken ist, dass die Generierung der Daten der diskreten Gleichverteilung unterliegen. Das bedeutet, dass bei entsprechend großen Datenmengen jeder Wert beziehungsweise jede Zeichenfolge annähernd gleich häufig vorkommt.

4.1.4 Messungen

Um Auswertungen und Vergleiche der Datenbanksysteme durchführen zu können, sind folgende Kriterien festgelegt worden, zu denen Werte ermittelt werden:

- Die benötigte Zeit für den Iterationsdurchlauf,
- die Menge an belegtem Speicherplatz der Tabellen aus dem Datenschema und den damit zusammenhängenden Datensätzen vor und nach der Iteration, sowie
- die Menge an Speicherplatz der Indizes, die sich auf die Tabellen aus dem ERM-Schema beziehen, vor und nach der Iteration, sowie

Die benötigte Zeit wird in Millisekunden für jede Iteration gemessen. Der von den einzelnen Tabellen belegte Hauptspeicher wird aufsummiert und als Summe gespeichert. Für die Indizes gilt die gleiche Vorgehensweise. Die Speichereinheit ist je nach verwendetem Datenbanksystem Byte oder Kilobyte. Werden in einem Datenbanksystem die Angaben für die Speicherbelegung sowohl in Byte als auch in Kilobyte angegeben, so erfolgt die Messung des jeweiligen Wertes in Byte und die Umrechnung von Byte in Kilobyte erfolgt mit dem Wert 1024 als Divisor.

Die Art der Messungen ist eingeteilt in *Warmup* und *Measurement*. Die Ergebnisse der Measurements stehen dabei für die Analyse im Fokus, aber bei besonderen Vorkommnissen werden auch die Resultate der Warmups betrachtet.

Bei VoltDB, EXASOL, SQL Server 2014 und 2016 werden die Messungen des belegten Hauptspeichers bei Tabellen und Indizes getrennt voneinander durchgeführt.

TimesTen bietet nicht die Möglichkeit, die Hauptspeicherbelegung für Tabellen und Indizes gleichermaßen exakt zu ermitteln. Um einen annähernden Vergleich zwischen TimesTen und den anderen Datenbanksystemen bezüglich der Speichermessung zu ermöglichen, wurden daher die Messkriterien bei TimesTen angepasst. Die Messung des belegten Speichers der Tabellen erfolgt, wie bei den anderen Hauptspeicherdatenbanksystemen, über die Aufsummierung des belegten Speichers der einzelnen Tabellen. Die Messung der Indizes unabhängig von den Tabellen ist nicht möglich. Es besteht aber die Möglichkeit, die geschätzte Speicherbelegung auf Basis von vorhandenen Datensätzen für die Tabellen und Indizes aufsummiert zu ermitteln. Diese Schätzung wird bei TimesTen anstelle der Indexmessung vor und nach jeder Iteration durchgeführt. Zusätzlich wird der gesamte verwendete Hauptspeicher für die permanente Datenhaltung, der unter anderem die Tabellen und Indizes beinhaltet, gemessen.

4.1.5 Softwaretechnischer Aufbau und Ablauf der Tests

Die von einer Programmiersprache abstrahierte, softwaretechnische Umsetzung der Tests gestaltet sich in drei Teile: (1) die Organisationseinheit für den Testzustand, (2) die implementierte Testmethode, die die durchzuführenden Datenbankoperationen darstellt und (3) die Ausführungseinheit.

Die Organisationseinheit ist für alle Software-Einheiten und Abläufe zuständig, die erstens die Interaktionen mit dem Datenbanksystem ermöglichen, zweitens die Testdaten bereitstellen und drittens die Testsoftware und die Datenbanksysteme in den Zustand versetzen, der für die Iterationsdurchläufe benötigt wird.

Die implementierte Testmethode stellt die jeweils auszuführenden Operationen im Zusammenhang mit den Datenbanksystemen dar. Dabei erfolgt die Ausführung dieser Methode in Form von Iterationen, wobei vor Beginn und nach dem Abschluss der jeweiligen Iteration Zeit- und Speichermessungen erfolgen.

Die Ausführungseinheit ist zuständig für den Ablauf der Tests. Bevor der eigentliche Test gestartet wird, erfolgt das Laden der Konfiguration. Bei der Konfiguration lassen sich beispielsweise der auszuführende Test, sowie die Anzahl der Messungen festlegen. Sobald das Laden erfolgreich durchgeführt wurde, startet der Test. Nachdem alle Iterationen erfolgreich durchlaufen wurden, erfolgt die Speicherung der Messdaten in einer Textdatei.

In dem ersten der beiden nachfolgenden Unterkapitel wird näher auf Ablauf eines Tests eingegangen. Dabei werden die wichtigsten Schritte beschrieben, sodass die Zusammenhänge zwischen Organisationseinheit, Test-Methode und Ausführungseinheit besser verständlich werden. Im zweiten Unterkapitel werden die einzelnen Messkriterien der Tests näher beschrieben.

4.1.5.1 Programmierter Testablauf

Die Abbildung 4.2 auf der folgenden Seite stellt eine Übersicht über die wichtigsten Schritte des auszuführenden Programms dar, die nachfolgend näher erklärt werden.

Mit dem Start des Programms wird die Konfiguration geladen und daraufhin beginnt der eigentliche Ablauf des Tests. Zu Beginn wird die Organisationseinheit erzeugt. In diesem Zusammenhang werden sämtliche Software-Einheiten erstellt, die für eine Verbindung zu der Datenbank zuständig sind.

Mit der erfolgreichen Herstellung der Verbindung kann die Datenbank in einen für den Test definierten Zustand versetzt werden. Beispielsweise kann das durch das Einfügen von Testdaten in die Datenbank geschehen, sodass für SELECT-Anweisungen nicht für jeden Iterationsdurchlauf die Datensätze immer wieder neu angelegt werden müssen, sondern dies einmalig zu Beginn des jeweiligen Tests geschieht.

Im Anschluss können dann optional die Software-Einheiten erstellt, die Verwendung in jedem Iterationsdurchlauf finden, aber nur einmalig erstellt werden müssen. Hierzu zählen Daten, die bei den INSERT-Tests genutzt werden.

Sobald die Vorbereitungen erfolgreich durchgeführt worden sind, können Vorgänge durchgeführt werden, die die Datenbank in den Zustand versetzen, der für die Ausführung der jeweiligen Testmethode in den Iterationen definiert worden ist. Danach besteht, je nach durchzuführendem Test, die Möglichkeit, Objekte zu erstellen, die für die Ausführung der Datenbankoperationen notwendig sind, sofern diese nicht bereits zu einem früheren Zeitpunkt erstellt worden sind. Anschließend wird der Anteil am Hauptspeicher gemessen, der von den Testdaten und -tabellen, sofern diese vorhanden sind, belegt wird. Daraufhin wird die Testmethode gestartet, die die jeweiligen Datenbankoperationen des Testfalls durchführt. Nach Beendigung der Testmethode wird die Zeit festgehalten, die für die Ausführung der Operationen benötigt wurde. Danach erfolgt eine weitere Messung der Hauptspeicherbelegung der Testtabellen und Testdaten. Den Abschluss der Iterationen bildet die optionale Entfernung der Software-Einheiten, die für die Ausführung der

Datenbankoperationen benötigt wurden. Anschließend erfolgt der nächste Durchlauf, wobei immer mit der Erstellung des Zustandes der Datenbank für die jeweilige Iteration begonnen wird.

Ablauf

1) Start des Programms
2) Laden der Konfiguration durch Ausführungseinheit
3) Start des Testablaufs
4) Erstellung der Organisationseinheit
5) Erzeugung der benötigten Software-Einheiten für die Datenbankverbindung
6) Aufbau der Verbindung zu der Datenbank
7) *Optional: Erzeugung von Testdaten, die für den definierten Zustand der Datenbank benötigt werden*
8) *Optional: Datenbank in einen definierten Zustand versetzen, wobei dieser Zustand die Iterationen überdauern kann*
9) *Optional: Erzeugung der benötigten Software-Einheiten für die Iterationen, die über die gesamte Testdauer vorhanden sind*

----- Beginn der Iteration

10) *Optional: Herstellung des Zustandes der Datenbank, der für die Ausführung der Testmethode benötigt wird*
11) *Optional: Erzeugung von Software-Einheiten, die für die Ausführung der Datenbankoperationen benötigt werden*
12) Messung der Hauptspeicherbelegung von Testtabellen und -daten
13) Ausführung der Testmethode
14) Messung der benötigten Zeit für die Durchführung der Testmethode
15) Messung des Hauptspeichers von Testtabellen und -daten
16) *Optional: Entfernen von Software-Einheiten, die für die Ausführung der Datenbankoperationen benötigt wurden*

----- Ende der Iteration

17) *Optional: Datenbank in einen definierten Zustand versetzen*
18) Verbindung zu der Datenbank trennen
19) Messungen in Textdatei speichern
20) Ende des Programms

Abbildung 4.2 Programmablauf

Nachdem die erforderliche Anzahl an Testdurchläufen durchgeführt worden ist, erfolgt die Trennung der Verbindung zu der Datenbank. Daraufhin erfolgt die Speicherung der Messergebnisse in einer Textdatei. Das Ende des Speichervorganges ist zugleich auch das Ende des Programmes. Anzumerken ist, dass bei dem Auftreten eines Fehlers der Versuch abgebrochen wird und keine bereits vorliegenden Testergebnisse gespeichert werden.

4.1.5.2 Darstellung der Messergebnisse

Die Messergebnisse werden nach jedem erfolgreich durchgeführten Benchmark in eine Textdatei geschrieben. Die Zeilen basieren auf der von CSV-Dateien bekannten Zeilenform, wobei zusätzlich die wichtigsten Konfigurationseinstellungen und einige datenbankrelevante Werte der Tests gespeichert werden.

Die Darstellung der Messungen in der Textdatei bei VoltDB, EXASOL, SQL Server 2014 und 2016 ist in Abbildung 4.3 ersichtlich, wobei die Bedeutung der Überschriften in der darauffolgenden Tabelle 4.1 erklärt werden.

Darstellung der Messung in der Textdatei
Iteration; Measurement; Unit Of Measurement; Memory Data Before Iteration; Memory Data After Iteration; Memory Index Before Iteration; Memory Index After Iteration; Memory Size Unit

Auflistung der Ergebnisse pro Iteration

Konfigurationseinstellungen

Abbildung 4.3 Aufbau der Textdatei

Bezeichnung	Bedeutung
Iteration	Art und Nummer der Iteration
Measurement	Der gemessene Wert
Unit Of Measurement	Die Messeinheit in Bezug auf den gemessenen Wert bei Measurement
Memory Data Before Iteration	Die von den Daten in den erstellten Tabellen aus dem ERM-Schema belegte Menge an Hauptspeicher vor dem Iterationsdurchlauf

Memory Data After Iteration	Die von den Daten in den erstellten Tabellen aus dem ERM-Schema belegte Menge an Hauptspeicher nach dem Iterationsdurchlauf
Memory Index Before Iteration	Der vor der Iteration belegte Hauptspeicher der Indizes, die zu den erstellten Tabellen aus dem ERM-Schema gehören
Memory Index After Iteration	Der nach der Iteration belegte Hauptspeicher der Indizes, die zu den erstellten Tabellen aus dem ERM-Schema gehören
Memory Size Unit	Die Einheit, in der die Speichermessung erfolgt

Tabelle 4.1 Beschreibung der Textdatei

In der Abbildung 4.4 wird gezeigt, wie die Ergebnisse in TimesTen gespeichert werden. Anstelle der Speichermessung der Indizes werden die geschätzte Summe des belegten Speichers von Tabellen und Indizes zusammen vor und nach jeder Iteration, sowie der Hauptspeicheranteil der permanenten Daten abgespeichert (siehe 4.1.4). Die Tabelle 4.2 beschreibt die einzelnen Überschriften der gemessenen Werte.

Darstellung der Messung in der Textdatei bei TimesTen
Iteration; Measurement; Unit Of Measurement; Memory Data Before Iteration; Memory Data After Iteration; Memory ttSize Before Iteration; Memory ttSize After Iteration; Memory Perm In Use Before Iteration; Memory Perm In Use After Iteration; Memory Size Unit

Auflistung der Ergebnisse pro Iteration

Konfigurationseinstellungen

Abbildung 4.4 Aufbau der Textdatei bei TimesTen

Bezeichnung	**Bedeutung**
Iteration	Art und Nummer der Iteration
Measurement	Der gemessene Wert
Unit Of Measurement	Die Messeinheit in Bezug auf den gemessenen Wert bei Measurement

Memory Data Before Iteration	Die von den Daten in den erstellten Tabellen aus dem ERM-Schema belegte Menge an Hauptspeicher vor dem Iterationsdurchlauf
Memory Data After Iteration	Die von den Daten in den erstellten Tabellen aus dem ERM-Schema belegte Menge an Hauptspeicher nach dem Iterationsdurchlauf
Memory ttSize Before Iteration	Der vor der Iteration geschätzte belegte Hauptspeicher der Indizes und Tabellen zusammen summiert, die zu den erstellten Tabellen aus dem Datenschema gehören
Memory ttSize After Iteration	Der nach der Iteration belegte Hauptspeicher der Indizes, die zu den erstellten Tabellen aus dem Datenschema gehören
Memory Perm In Use Before Iteration	Die belegte Menge an Hauptspeicher der permanenten Daten vor jeder Iteration
Memory Perm In Use After Iteration	Die belegte Menge an Hauptspeicher der permanenten Daten nach jeder Iteration
Memory Size Unit	Die Einheit, in der die Speichermessung erfolgt

Tabelle 4.2 Beschreibung der Textdatei bei TimesTen

4.1.6 Konfiguration von Hard- und Software

In diesem Unterkapitel werden die verwendete Hardware und Software mitsamt ihrer Konfiguration beschrieben. Für die Versionsnummern und weiterführenden Informationen wird auf den Anhang verwiesen.

4.1.6.1 Hardware

Die Datenbanksysteme werden unter der Verwendung eines Laptops sowie eines Desktop-Rechners getestet. Der Laptop ist ein Acer Aspire 5750G-32354G32Mnkk, wobei die wichtigsten Ausstattungsmerkmale die Intel i3 2350M 2,3 GHz CPU, die 8 GB RAM sowie die 500 GB Samsung HDD sind. Die wesentlichen Kenndaten des

Desktop-PCs sind der Intel i5-3470 Prozessor, die 8 GB Hauptspeicher, sowie zwei 128GB SSD und eine 1TB HDD.

Der Desktop-Rechner bildet die hardwaretechnische Grundlage für die Hauptspeicherdatenbanksysteme. Die Tests werden von dem Laptop aus gestartet, wobei dieser auch die Messungen durchführt. Für die Datenübertragung zwischen Desktop-Rechner und Laptop wird ein 1 Gigabit-Netzwerk über Kabelverbindungen verwendet.

4.1.6.2 Betriebssysteme

Als Betriebssystem für die Datenbanksysteme TimesTen, SQL Server 2014 und 2016 kommt Windows Server 2012 R2 zum Einsatz. Das Hauptspeicherdatenbanksystem VoltDB wird unter Ubuntu Server 16.10 eingesetzt.

Mittels der Software clonezilla wurden von beiden Betriebssystemen Abbilder erstellt, nachdem diese aktualisiert worden sind. Unter der Verwendung der Abbilder kann somit für jedes Datenbanksystem, das dasselbe Betriebssystem verwendet, ein eigenes, zu den anderen Datenbanken identisches Grundsystem bereitgestellt werden. Dadurch wird verhindert, dass weitere Software-Installationen, die für ein bestimmtes Datenbanksystem notwendig sind, einen Einfluss auf die anderen Datenbanksysteme haben könnten.

Eine Besonderheit liegt bei EXASOL vor: Dieses Datenbanksystem läuft unter dem herstellereigenen Betriebssystem EXAClusterOS ab und wird mit diesem zusammen als virtuelle Maschine ausgeliefert. Für die Nutzung von EXASOL wird daher eine Virtualisierungssoftware benötigt. Zu diesem Zweck wurde VirtualBox ausgewählt, dass unter Windows Server 2012 R2 abläuft. Für die Ausführung der virtuellen Maschine wurden dieser 4 Rechenkerne und 5 GB Hauptspeicher zugeteilt. Die Netzwerkumgebung wurde als NAT konfiguriert und die entsprechenden Ports für den Zugriff auf das Datenbanksystem wurden freigegeben.

4.1.6.3 Datenbanksysteme

Bei den Datenbanksystemen wurde darauf geachtet, dass diese über annähernd gleiche bzw. gleichartige Grundeinstellungen[1] verfügen, sofern dies bestimmt werden kann. Die Einstellungen werden, sobald diese im jeweiligen System festgelegt

[1] Eine exakt gleiche bzw. gleichwertige Konfiguration der Datenbanksysteme ist nicht möglich, da zum einen nicht alle Einstellungsmöglichkeiten bei jedem zu testenden Datenbanksystem vorhanden sind und zum kann der Detaillierungsgrad der einzustellenden Werten variieren.

worden sind, nicht mehr verändert. Dadurch wird gewährleistet, dass jedem Test, der bei dem jeweiligen Datenbanksystem ausgeführt wird, derselbe Konfigurationszustand zugrunde liegt. Darüber hinaus werden, nachdem die Konfiguration durchgeführt wurde, die Aktualisierungen deaktiviert, sodass auch hier sichergestellt ist, dass die Datenbanksoftware dieselbe für alle Tests ist.
Der Wert für den Timeout bei der Ausführung von SQL-Befehlen wird auf 300 Sekunden gesetzt. Damit wird vermieden, dass Tests unnötig lange andauern. Die Menge an Hauptspeicher, die für die permanente Datenhaltung genutzt werden kann, wird auf 4 GB gesetzt. Dadurch wird ausreichend Speicher für die Datensätze und Tabellen bereitgestellt. Der temporäre Hauptspeicheranteil, der unter anderem für die Ausführung von Datenbankoperationen genutzt wird, wird auf 1 GB gesetzt.

Als Zeichensatz für VARCHAR bzw. VARCHAR2 wird UTF-8 verwendet und die entsprechende Längenangabe wird als maximale Anzahl an Zeichen konfiguriert. Das Erstellen von Sicherungen und Logdateien ist grundsätzlich eingeschaltet. Der Isolationslevel für Transaktionen wird auf serializable eingestellt.

In den nachfolgenden Absätzen werden die Umsetzungen der Konfigurationen im Zusammenhang mit den eben angeführten Einstellungen für jedes der verwendeten Datenbanksysteme erläutert. Die exakten Einzelschritte werden im Anhang aufgeführt.

Der Zugriff auf die TimesTen-Datenbank unter Windows wird mittels eines Data Source Names durchgeführt. In diesem Zusammenhang wird mit dem TimesTen Data Manager Treiber die Konfiguration der Datenbank durchgeführt. Dabei lassen sich in den Einstellungen alle vorher für die Grundeinstellung definierten Werte setzen.

Die Konfiguration von SQL Server 2014 und SQL Server 2016 erfolgt mit dem Microsoft SQL Server Management Studio. Hierbei ist vorweg zu erwähnen, dass aus den Dokumentation von SQL Server 2014 und SQL Server 2016 nicht ersichtlich ist, wie sich die Speicherverwendung im Hauptspeicherbetrieb exakt verhält. Die Einstellungen bezüglich der Hauptspeicherverwaltung erfolgen in den beiden Datenbanksystemen danach, was die Dokumentation (vgl. Microsoft, In-Memory OLTP (Arbeitsspeicheroptimierung), 2016a) an Informationen bereitstellt.
Die Einstellungen für die Verwendung des Hauptspeichers erfolgt in mehreren Schritten: Zuerst wird der Serverinstanz die minimale und maximale Menge an zu nutzendem Hauptspeicher zugewiesen. Beide Mengenangaben werden auf 5120 MB (entspricht 5 GB) gesetzt. Danach wird die Datenbank mitsamt der für den Hauptspeicherbetrieb notwendigen Dateigruppe und der Filestream-Datei.

Anschließend wird ein Ressourcenpool erstellt, der den für die Tabellen zur Verfügung stehenden Hauptspeicher auf annähernd 4 GB beschränkt. Hier ist zu erwähnen, dass bei der Anzeige der Hauptspeicheraufteilung mittels SELECT-Anweisung als maximale Menge 4864 MB[1] angezeigt wird und dass die Angabe für die zu nutzende Menge an Hauptspeicher bei Ressourcenpools in Prozentangaben erfolgt, wobei hier als Wert 84 angegeben wurde, was abgerundet 4085 MB entspricht. Daraufhin wird die Datenbank an den Ressourcenpool gebunden und daraufhin der für die Hauptspeicherverwaltung zuständige Resource Governor neukonfiguriert. Den Abschluss des Bindungsprozesses bildet eine Off- und anschließende Online-Schaltung der Datenbank und danach kann der Hauptspeicherbetrieb in Verbindung mit dem Ressourcenpool erfolgen.
In SQL Server 2014 und SQL Server 2016 ist es nicht möglich UTF-8 als Zeichensatz einzustellen, sodass der Wert für den Zeichensatz in Form der Collation auf Latin1_General_CI_AS belassen wird. Die Zeit für den Timeout bei SQL-Befehlen wurde auf 300 Sekunden gesetzt und die Längenangabe für VARCHAR erfolgt in Zeichen. Die Isolationsstufe für den Transaktionsbetrieb lässt sich nicht auf serializable einstellen[2].

In EXASOL lässt dich die Gesamtmenge an zu nutzendem Hauptspeicher angeben, wobei keine Beschreibung erfolgt, wie der Hauptspeicher genutzt wird. Die Menge wurde auf 4 GB[3] gesetzt. Der Timeout-Wert ist auf 300 Sekunden eingestellt worden. Die Längenangabe für VARCHAR stellt die Anzahl der Zeichen dar und der verwendete Zeichensatz ist auf UTF-8 eingestellt. EXASOL bietet nur die Isolationsstufe serializable für Transaktionen an, sodass hier keine Umstellung vorgenommen wurde.

Bei VoltDB handelt es sich, wie in 3.4 näher beschrieben, um ein Datenbanksystem, dessen Architektur auf Parallelität ausgelegt ist. Mit diesem Hintergrund wurde die

[1] Hier stellt sich die Frage, warum nicht die angegebenen 5120 MB genutzt werden bzw. wie die Differenz bezüglich der Hauptspeicherangabe erklärt wird.
[2] Beim Versuch die Transaktionen auf serializable für die jeweils aktuelle Verbindung einzustellen, erfolgt die Fehlernachricht: com.microsoft.sqlserver.jdbc.SQLServerException: Die folgenden Transaktionen müssen unter Verwendung der Momentaufnahmeisolation auf speicheroptimierte Tabellen und systemintern kompilierte Module zugreifen: RepeatableRead-Transaktionen, Serializable-Transaktionen und Transaktionen, die auf Tabellen zugreifen, die in der RepeatableRead-Isolation oder der Serializable-Isolation nicht speicheroptimiert sind.

[3] Angabe erfolgt als ganzzahliger Wert in GB und EXASOL empfiehlt ungefähr 90% des Hauptspeichers der virtuellen Maschine zu nutzen (vgl. EXASOL AG, EXASolo does not use more RAM after increase in VM RAM, 2015c)

Anzahl der Partitionen von Daten auf zwei festgelegt. Dabei wird die Menge an Arbeitsspeicher für die persistente Datenhaltung auf 4 GB gesetzt. Bei der Konfiguration der Menge an temporären Hauptspeicher besteht aufgrund der Architektur die Schwierigkeit, die Konfiguration der Hauptspeicherwerte so durchzuführen, dass diese annährend vergleichbar ist mit den Einstellungen der anderen Systeme. Es wurden zwei Konfigurationsmöglichkeiten identifiziert, die im Zusammenhang mit den Angaben für temporären Hauptspeicher stehen: temptables und die Einstellung des verwendeten Hauptspeichers der Java Virtual Machine. Der Wert für temptables stellt dabei die Menge an Speicher dar, die während den Transaktionen Daten aus Tabellen zwischenspeichert. Dieser Wert wird auf 1024 MB eingestellt. Die Java Virtual Machine wird in VoltDB für die Organisation der Prozeduren und die Zwischenspeicherung der Rückgabewerte der durchgeführten Prozeduren genutzt. Die Einstellung dieses Wertes wurde nicht verändert und somit auf 2048 MB belassen.
Als Zeichensatz wird UTF-8 genutzt und die Längenangabe ist als Anzahl von Zeichen bereits die Standardeinstellung. Der Timeout-Wert für SQL-Befehle ist auf 300 Sekunden eingestellt worden. Die einzige Transaktionsstufe in VoltDB ist serializable, sodass hier keine Einstellung vorgenommen werden kann.

4.1.6.4 Software für die Entwicklung und Ablauf der Tests

Die Tests wurden mit der Programmiersprache Java und der Entwicklungsumgebung Eclipse IDE for Java Developers erstellt. Dabei wurden das Benchmark-Framework JMH (Java Microbenchmarking Harness) und die Softwarebibliothek Chronicle-Queue verwendet. Das Framework stellt die Grundstruktur für die Tests zur Verfügung und ermöglicht zudem die Zeitmessung. Mittels der Verwendung der Klassen von Chronicle-Queue wurden die Messungen des durch die Daten belegten Hauptspeichers zwischengespeichert, sodass diese, neben den gemessenen Zeiten, am Ende des Tests für die Speicherung zur Verfügung stehen. Die Interaktionen mit den Datenbanksystemen werden mittels der entsprechenden JDBC-Treiber der Datenbankanbieter durchgeführt. Für Stored Procedures werden entsprechend CallableStatement-Objekte, für Prepared Statements PreparedStatement-Objekte und für die statischen SQL-Befehle Statement-Objekte genutzt. Bei VoltDB wird neben der Implementierung der Interfaces aus dem Packet java.sql ein eigener Client für die Verwendung in Java-Programmen zur Verfügung gestellt.

Die Tests werden mittels Eclipse gestartet. Dabei wird die Java Virtual Machine über die Konfigurationsmöglichkeiten bei Eclipse mit den Argumenten -Xms1024m und

-Xmx4096m ausgeführt. Damit wird bezweckt, dass ausreichend Hauptspeicher für die zu nutzenden Daten in den Testfällen bereitsteht und ein mögliches Eingreifen der Garbage Collection auf ein Minimum reduziert wird.

Mittels dem Test-Framework JUnit wurden die softwaretechnischen Umsetzungen der Datengeneratoren, Testdaten-Objekte, Zustände und auszuführenden Testmethoden auf Korrektheit überprüft. Dies schließt eine Überprüfung der zu erwartenden Resultate der durchzuführenden Operationen mit ein. Das bedeutet, dass bei SELECT-Operationen die Ergebnisse der Berechnungen überprüft werden und bei den Datenbankoperationen, die Daten einfügen, aktualisieren oder löschen, werden die Datensätze überprüft.

4.2 Durchführungen der Versuche

In diesem Unterkapitel werden die Versuche jeweils in Form einer Kurzbeschreibung dargestellt. Dabei wird erläutert was bei dem jeweiligen Test durchgeführt wird, welche Messkriterien relevant sind, welche Tabellen verwendet werden, wie die Nutzung der Testdaten erfolgt und welche weiteren, hervorzuhebenden Details vorhanden sind, die diesen Test ausmachen. Die Versuche sind anhand der Kategorien auf Basis der Datenbankoperationen eingeteilt (siehe hierzu 4.1.1).

Vorweg ist zu erwähnen, dass in TimesTen sämtliche Tests aufgrund der dreifachen Umsetzung des Testschemas (siehe hierzu 4.1.2) für jede der Umsetzungen je einmal ausgeführt werden.

Für die Durchführung der einzelnen Versuche gilt, dass nach dreimaligem Fehlschlagen der jeweilige Versuch abgebrochen wird

4.2.1 INSERT

Die Versuche in dieser Kategorie sollen ermitteln, wie sich das Einfügen der Datensätze auf die einzelnen Datenbanksysteme auswirkt. Dabei erfolgen die Iterationen derart, dass nach jeder Iteration die Datenbankobjekte gelöscht werden und für jeden Testdurchlauf eine leere Datenbank vorhanden ist.

4.2.1.1 INSERT CUSTOMER

Was wird durchgeführt:
Das Einfügen von Customer-Datensätzen im Batch-Modus.

Messung:
Die benötigte Zeit für das Einfügen von Customer-Datensätzen und den durch die hinzugefügten Datensätze belegten Hauptspeicher der Tabellen und der damit verbundenen Indizes.

Verwendete Tabelle(n):
Customer

Weitere Details:
Das Einfügen erfolgt dabei in Form von Statements, Stored Procedures und Prepared Statements, sofern diese bei den Datenbanksystemen nutzbar sind. Bei VoltDB wird zusätzlich der Client getestet. Die hier durchgeführten Versuche ermitteln zudem die Skalierung des jeweiligen Datenbanksystems bezüglich des Einfügens. Zu diesem Zweck wird eine Menge von Datensätzen auf Basis von Zehner-Potenzen verwendet, wobei die Menge nach jedem durchgeführten Versuch wächst.

4.2.1.2 INSERT CUSTOMER ONE BY ONE

Was wird durchgeführt:
Das Einfügen von Customer-Datensätzen einzeln hintereinander.

Messung:
Die benötigte Zeit für das Einfügen von Customer-Datensätzen und den durch die hinzugefügten Datensätze belegten Hauptspeicher der Tabellen und der damit verbundenen Indizes.

Verwendete Tabelle(n):
Customer

Weitere Details:
Das Einfügen erfolgt, wie schon in 4.2.1.1, in Form von Statements, Stored Procedures und Prepared Statements, sofern diese bei den Datenbanksystemen nutzbar sind. Bei VoltDB wird auch hier der Client getestet. Die hier durchgeführten Versuche ermitteln zudem die Skalierung des jeweiligen Datenbanksystems bezüglich des Einfügens. Zu diesem Zweck wird eine Menge von Datensätzen auf Basis von Zehner-Potenzen verwendet, wobei die Menge nach jedem durchgeführten Versuch wächst.

4.2.1.3 INSERT ALL

Was wird durchgeführt:
Einfügen von Datensätzen in alle Tabellen des Testschemas im Batch-Modus.

Messung:
Die benötigte Zeit für das Einfügen der Datensätze und den durch die hinzugefügten Datensätze belegten Hauptspeicher der Tabellen und der damit verbundenen Indizes.

Verwendete Tabelle(n):
Customer, Customer_Order, Customer_Order_Item, Employee, Supplier, Product, Purchase

Weitere Details:
Das Einfügen erfolgt bei diesen Versuchen in Form von Stored Procedures und Prepared Statements, sofern diese bei Datenbanksystemen sinnvoll nutzbar sind. Bei VoltDB wird der eigene Client genutzt.

Für die Ermittlung, welche Art des Einfügens den Aufbau des gesamten Datenbankzustandes in einem Datenbanksystem und im Vergleich zu den anderen Datenbanksystemen am schnellsten durchführt, wird die folgende Menge der jeweils einzufügenden Datensätze verwendet:

- 20 Employee-Datensätze,
- 5.000 Product-Datensätze,
- 35 Supplier-Datensätze,
- 149.431 Purchase Datensätze,
- 100.000 Customer-Datensätze,
- 250.000 Customer_Order-Datensätze und
- 2.499.721 Customer_Order_Item-Datensätze.

4.2.2 SELECT

Die SELECT-Statements werden für Analysen benutzt, bei denen die Durchführungszeit für die Auswertung der Anfragen ermittelt werden sollen.

Die für die Versuche SELECT 2 - 6 verwendeten Datensätze werden vor Testbeginn in die Datenbank eingefügt. Da es sich hierbei um Daten handelt, die nach dem Einfügen keinen Veränderungen unterliegen, kann somit vermieden werden, dass die Daten für jeden Versuch neu eingefügt werden müssen.

Dabei setzen sich diese Datensätze wie folgt zusammen:

- 1.000.000 Customer-Datensätze,
- 2.500.000 Customer_Order-Datensätze,
- 4.999.105 Customer_Order_Item-Datensätze,
- 50 Employee-Datensätze,
- 50.000 Product-Datensätze,
- 299.895 Purchase-Datensätze und
- 100 Supplier-Datensätze.

4.2.2.1 SELECT 1

Was wird durchgeführt:
Die Einteilung der Kunden in Altersgruppen mit abschließender Zählung der Anzahl der Kunden pro Altersgruppe.

Messung:
Die benötigte Zeit für die Abfrage.

Verwendete Tabelle(n):
Customer

Weitere Details:
Bei diesem Versuch erfolgt eine Ermittlung der Skalierung, sodass dieser Versuch mehrfach, mit einer jeweils größer werdenden Menge an Datensätzen durchgeführt wird. Die Anzahl der Datensätze basiert dabei auf Zehner-Potenzen. Für die Einteilung wird dabei nur die Spalte birthday in Kombination mit den, für die Berechnung notwendigen Funktionen CASE, MONTHS_BETWEEN (bzw. vergleichbare Funktion oder Berechnung), GROUP BY, ORDER BY und COUNT(*) genutzt.

4.2.2.2 SELECT 2

Was wird durchgeführt:
Ausgabe des Vor- und Nachnamens des Kunden zu der jeweiligen Bestellung, sowie den Namen, die Anzahl und den erzielten Umsatz des Produktes der einzelnen Bestellungen, das den höchsten Umsatz als Bestellposition in der jeweiligen Bestellung erzielt hat. Die Datensätze der Ergebnismenge sind nach dem höchsten Umsatzwert absteigend sortiert

Messung:
Die benötigte Zeit für die Abfrage.

Verwendete Tabelle(n):
Customer, Customer_Order, Customer_Order_Item, Product

Weitere Details:
Mehrfachnutzung von Spalten und die Multiplikation von größeren Datenmengen (Preis * Menge), sowie eine größere Menge an temporären Daten. Darüber hinaus werden die Funktionen ORDER BY und GROUP BY verwendet.

4.2.2.3 SELECT 3

Was wird durchgeführt:
Ausgabe der pro Kunde(-ID) erzielten Einnahmen.

Messung:
Die benötigte Zeit für die Abfrage.

Verwendete Tabelle(n)
Customer, Customer_Order, Customer_Order_Item, Product

Weitere Details:
Die Funktionen SUM und GROUP BY werden genutzt. Darüber hinaus erfolgt unter anderem die Multiplikation einer größeren Menge an Daten.

4.2.2.4 SELECT 4

Was wird durchgeführt:
Angabe der Top-20-Verkaufsprodukte im Sortiment und der Lieferanten, die diese Produkte liefern können. Die Lieferanten werden ihrer ID nach aufsteigend und die Produkte nach ihren erzielten Einnahmen absteigend sortiert, wobei die Produkte entsprechend zu den Lieferanten gruppiert werden.

Messung:
Die benötigte Zeit für die Abfrage.

Verwendete Tabelle(n):
Customer_Order_Item, Product, Purchase, Supplier

Weitere Details:
Nutzung der Funktionen ORDER BY, ROWNUM (bzw. vergleichbare Funktion), Multiplikation, SUM und GROUP BY.

4.2.2.5 SELECT 5

Was wird durchgeführt:
Gruppierung der Kunden ihren Vornamen nach zu den Produktkategorien, wobei die Datensätze nach dem erzielten Umsatz auf Basis der Vornamen in absteigender Reihenfolge pro Gruppierung sortiert sind.

Messung:
Die benötigte Zeit für die Abfrage.

Verwendete Tabelle(n)
Costumer, Customer_Order, Customer_Order_Item, Product

Weitere Details:
Nutzung der Funktionen SUM, GROUP BY und ORDER BY sowie die Multiplikation von Datensätzen.

4.2.2.6 SELECT 6

Was wird durchgeführt:
Ausgabe der ID der Mitarbeiter und die Ausgabe der Produkt-IDs, die die Mitarbeiter jeweils am häufigsten verarbeitet haben. Dies erfolgt in Form eines Top-3-Rankings, wobei ein Platz auch mehrere Produkte enthalten kann, wenn sie gleich oft vorkommend genannt werden in Kombination mit dem Mitarbeiter.

Messung:
Die benötigte Zeit für die Abfrage.

Verwendete Tabelle(n):
Customer_Order, Customer_Order_Item, Employee

Weitere Details:
Nutzung der Funktionen RANK(), SUM, TOP, ORDER BY und GROUP BY.

4.2.3 UPDATE

Die UPDATE-Versuche führen Veränderungen auf Datensätze der Customer-Tabelle durch. Die Gestaltung der Testiterationen erfolgt derart, dass vor jedem Durchlauf sämtliche benötigten Datenbankobjekte entfernt und neu angelegt werden. Die Aktualisierungen erfolgen somit immer auf den gleichen Datensätzen.

4.2.3.1 UPDATE SIMPLE

Was wird durchgeführt:
Aktualisierung von Geburtstag und Registrationsdatum der Customer-Datensätze.

Messung:
Die benötigte Zeit für den Aktualisierungsprozess und den durch die veränderten Datensätze belegten Hauptspeicher der Tabellen und der damit verbundenen Indizes.

Verwendete Tabelle(n):
Customer

Weitere Details:
Die hier durchgeführten Versuche ermitteln zudem die Skalierung. Dabei werden Mengen von Datensätzen verwendet, deren Anzahl sich auf Zehner-Potenzen beläuft. Darüber hinaus sind die Messungen der benötigten Zeit und des belegten Hauptspeichers der Datensätze vor und nach den Aktualisierungen insbesondere für den Einfluss von Kompression relevant, sofern die Datenbanksysteme die Kompression von Daten nutzen.

4.2.3.2 UPDATE COMPLEX

Was wird durchgeführt:
Ändern von mehreren Attributen der Customer-Datensätze.

Messung:
Die benötigte Zeit für den Aktualisierungsprozess und den durch die veränderten Datensätze belegten Hauptspeicher der Tabellen und der damit verbundenen Indizes.

Verwendete Tabelle(n):
Customer

Weitere Details:
Die hier durchgeführten Versuche ermitteln zudem die Skalierung. Dabei werden Mengen von Datensätzen verwendet, deren Anzahl sich auf Zehner-Potenzen beläuft. Darüber hinaus sind die Messungen der benötigten Zeit und des belegten Hauptspeichers der Datensätze vor und nach den Aktualisierungen insbesondere für den Einfluss von Kompression relevant, sofern die Datenbanksysteme die Kompression von Daten nutzen.

4.2.4 DELETE

Die Testfälle mit der DELETE-Anweisung nutzen nur die Tabelle Customer. Bedingt durch Änderungen an den Datensätzen werden auch hier, wie bei UPDATE, die Datenbestände auf den erwarteten, vorbereiteten Zustand für jede Iteration gebracht.

4.2.4.1 DELETE SIMPLE

Was wird durchgeführt
Löschen von Kundendatensätzen, wenn der Nachname sich aus lexikalischer Sicht zwischen A und Izzz befindet.

Messung:
Die benötigte Zeit für die Löschung der Datensätze und den durch die noch vorhandenen Datensätze belegten Hauptspeicher der Tabellen und der damit verbundenen Indizes.

Verwendete Tabelle(n):
Customer

Weitere Details:
Die hier durchgeführten Versuche ermitteln zudem die Skalierung. Dabei werden Mengen von Datensätzen verwendet, deren Anzahl sich auf Zehner-Potenzen beläuft. Darüber hinaus sind die Messungen der benötigten Zeit und des belegten Hauptspeichers der Datensätze vor und nach den Aktualisierungen insbesondere für den Einfluss von Kompression relevant, sofern die Datenbanksysteme die Kompression von Daten nutzen.

4.2.4.2 DELETE COMPLEX

Was wird durchgeführt:
Löschen von Kundendatensätzen bei Übereinstimmungen von einer oder mehreren Bedingungen bei der Betrachtung von acht Attributen.

Messung:
Die benötigte Zeit für die Löschung der Datensätze und den durch die noch vorhandenen Datensätze belegten Hauptspeicher der Tabellen und der damit verbundenen Indizes.

Verwendete Tabelle(n):
Customer

Weitere Details:
Die hier durchgeführten Versuche ermitteln auch die Skalierung. Dabei werden Mengen von Datensätzen verwendet, deren Anzahl sich auf Zehner-Potenzen beläuft. Darüber hinaus sind die Messungen der benötigten Zeit und des belegten Hauptspeichers der Datensätze vor und nach den Aktualisierungen insbesondere für den Einfluss von Kompression relevant, sofern die Datenbanksysteme die Kompression von Daten nutzen.

4.3 Analyse der Versuche

Die Untersuchungen erfolgen in der aus Unterkapitel 4.2 vorgenommenen Reihenfolge und Darstellung der Versuche. Der Fokus der Auswertungen liegt auf den Ergebnissen der Measurement-Iterationen für die benötigte Zeit der einzelnen Versuche und dem in den Tabellen den Testdaten direkt zuordenbaren Hauptspeicherverbrauch, sofern diese für die jeweilige Begutachtung bedeutsam ist. Dabei wird der Speicherverbrauch als Differenz der Angaben des durch die Daten belegten Speichers vor und nach der Datenbankoperation bzw. der -operationen. Aussagen bezüglich der benötigten Zeit oder Menge an Hauptspeicher im Zusammen der Skalierung bei steigenden Datenmengen beziehen sich im Allgemeinen auf größere Datenmengen, da bei kleineren Datenmengen der Einfluss von diversen Faktoren auf die Messergebnisse größer ist, als bei einer größeren Menge von Daten. Bezüglich der Betrachtung der zeitlichen beziehungsweise speichertechnischen Skalierung sei auf 4.1.1 hingewiesen.

Die in den Tabellen enthaltenen Messwerte sind grundsätzlich, sofern nicht anders angegeben, die entsprechenden Mittelwerte, die sich aus den Versuchsdurchläufen ergeben. Dabei erfolgen die Berechnungen dieser Werte auf Basis des arithmetischen Mittels.

Die grafische Darstellung in Form von Liniendiagrammen erfolgt, sofern nicht anders angegeben, auf Basis der doppelt-logarithmischen Skalierung, da zwischen den einzelnen Werten teilweise große Differenzen vorliegen. Zudem vereinfacht diese Form der Darstellung die Betrachtung der Skalierung der Datenbanksysteme bei den unterschiedlichen Datenbankoperationen, sofern dieser Aspekt relevant ist. Die Punkte in den Liniendiagrammen stellen im jeweiligen Betrachtungskontext den Wert des arithmetischen Mittels der Ergebnisse der Datenbanksysteme bei der an der X-Achse verzeichneten Datenmenge in den Versuchen dar.

Ein Fehlen von einem Datenbanksystem bei der der jeweiligen Ergebnisauswertung bedeutet, dass bei dem entsprechenden System dieser Test nicht durchgeführt wurde. Die dazugehörige Begründung wird in der jeweiligen Auswertung dargelegt und das System findet keine weitere Erwähnung bei der entsprechenden Ergebnisanalyse. Die Aussagen in den jeweiligen Auswertungen, die das Wort *alle* beinhalten, sind so zu verstehen, dass damit die Datenbanksysteme gemeint sind, die diesen Versuch erfolgreich durchgeführt haben.

Bei EXASOL beginnen die Tests, sofern nicht anders angegeben, bei einer Anzahl von 10 Datensätzen, da der Batch-Modus keine Verarbeitung eines einzelnen Datensatzes zulässt.

Für TimesTen erfolgen die Auswertungen dreifach, da dieses Datenbanksystem die Möglichkeit bietet für Daten optional eine Kompression durchzuführen. Die Resultate der Versuche ohne Kompressionsverwendung werden in den Darstellungen als TimesTen bezeichnet. Die Ergebnisse der Versuche mit der standardmäßigen Kompression der Datensätze werden als TimesTen STD-KOMP gekennzeichnet und die Resultate zu den Versuchen mit den anhand einer Spezifikation komprimierten Datensätzen werden unter TimesTen SPEZ-KOMP geführt. Für weitere Informationen bezüglich der Kompression in TimesTen sei auf 4.1.2 und den Anhang verwiesen.

Zu erwähnen ist, dass Aufgrund der Datenmenge der Messergebnisse diese auf der dieser Arbeit beiliegenden CD vorhanden sind und nicht im Anhang aufgeführt werden.

4.3.1 Auswertung der Versuche aus der Kategorie INSERT

In diesem Unterkapitel werden die wesentlichen Ergebnisse aus den Versuchen mit den unterschiedlichen Arten des Einfügens analysiert. Die Betrachtungen setzen sich zusammen aus den Resultaten für Tests, die sich nur auf Tabelle Customer beziehen, als auch auf das gesamte umgesetzte Schema aus 4.1.2.

Vorab sei zu erwähnen, dass der VoltDB eigene Client bei Auswertungen zu Stored Procedures, Prepared Statements und Statements in die Abbildungen und Tabellen mit aufgenommen wird, um diesen mit den anderen Möglichkeiten des Einfügens zu vergleichen.

4.3.1.1 Auswertungen zu INSERT CUSTOMER

Bei den Tests bezüglich dieser Art des Einfügens von Customer-Datensätzen im Batch-Modus ließen sich einige Versuche gänzlich nicht durchführen. In EXASOL sind Stored Procedures nicht vorhanden, sodass in diesem Fall keine Auswertung für dieses Datenbanksystem stattfindet. Für das System VoltDB ist ebenfalls für Stored Procedure keine Begutachtung möglich, da der entsprechende JDBC-Treiber den Batch-Modus für Callable-Statements nicht anbietet.

In der Abbildung 4.5 auf der nächsten Seite wird dargestellt, wie sich die Datenbanksysteme bezüglich der zeitlichen Skalierung beim Einfügen verhalten. Dabei werden auch die verschiedenen Arten des Einfügens im Batch-Modus gegenübergestellt. Diese werden nachfolgend näher betrachtet.

Zu erkennen ist, dass im Allgemeinen alle Datenbanksysteme, mit Ausnahme von EXASOL, bei einzufügenden Mengen größer als 1.000 Datensätzen annähernd linear bezüglich der benötigten Zeit skalieren. Davor ist der Zeitaufwand für das Einfügen teilweise deutlich besser als linear.

Für TimesTen lässt sich feststellen, dass das Einfügen mittels eines Prepared Statements die schnellste Möglichkeit darstellt, Datensätze dem Datenbanksystem hinzuzufügen. Darauf folgt die Variante mit einer Stored Procedure, die im Vergleich, insbesondere bei großen Datenmengen, deutlich langsamer ist.

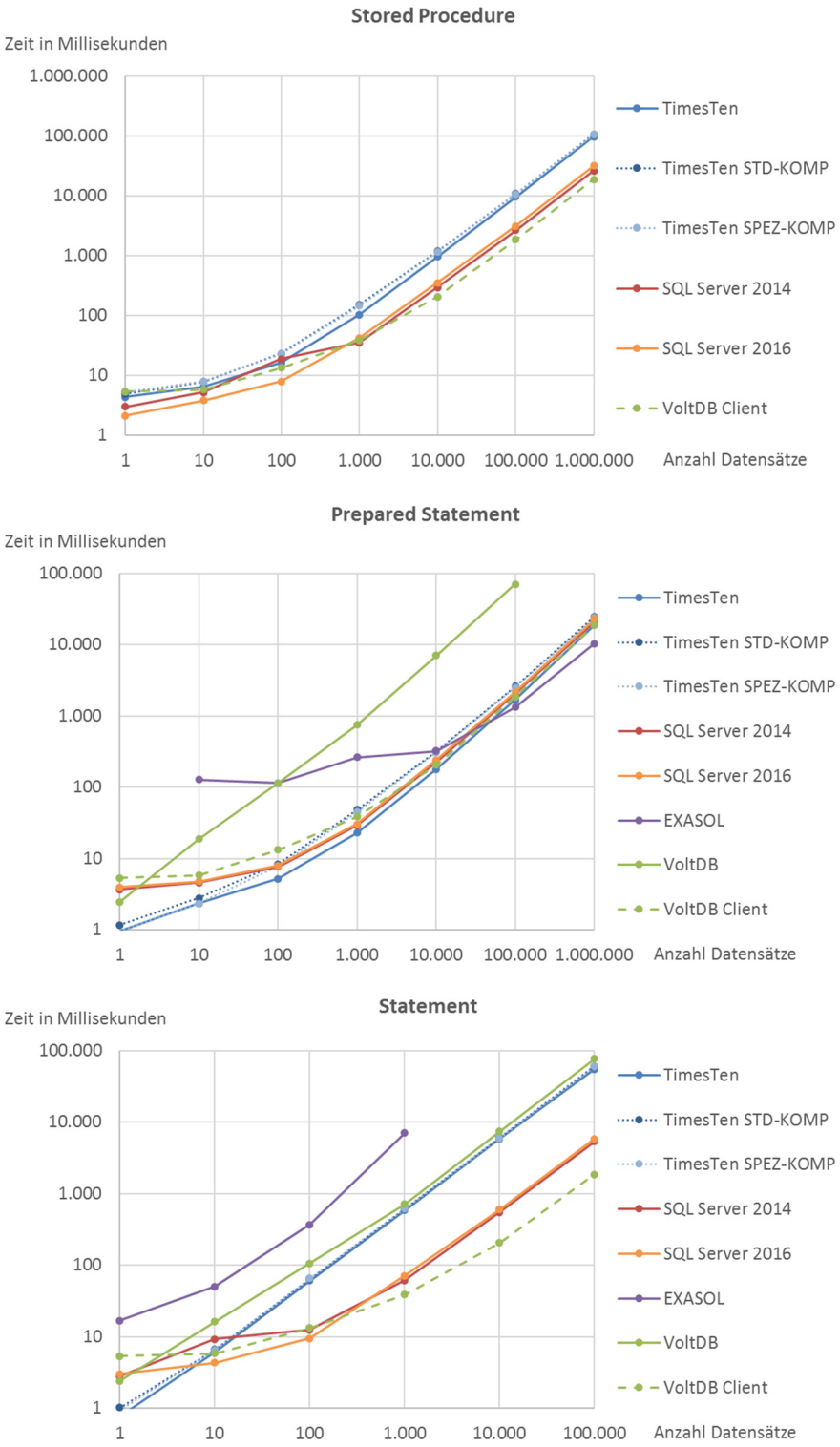

Abbildung 4.5 Benötigte Zeiten in ms bei INSERT CUSTOMER

Das Einfügen unter Verwendung eines Statements stellt die langsamste Form dar. Diese Beobachtungen gelten gleichermaßen für die nicht komprimierte und komprimierte Datenorganisation. Dabei lässt sich bei der Nutzung von Kompression festhalten, dass das Einfügen fast immer länger dauert, als bei der Nichtnutzung der Kompressionsmöglichkeit. Bei der Gegenüberstellung mit den anderen Datenbanksystemen stellt TimesTen beim Einfügen von Datensätzen mittels einer Stored Procedure unter Nutzung und Nichtnutzung von Kompression das langsamste System dar. Die Verwendung eines Prepared Statement zeigt, dass bei kleineren Datenmengen TimesTen das schnellste Datenbanksystem ist. Steigt die Anzahl der Daten an, wird das System von Oracle zu einem der langsameren Systeme im Vergleich mit den anderen Datenbanksystemen. Bei dem Gebrauch des Statements für das Hinzufügen von Daten befindet sich TimesTen im Mittelfeld bei der Gegenüberstellung der Zeitwerte.

Die Datenbanksysteme SQL Server 2014 und SQL Server 2016 haben bei der Gegenüberstellung untereinander ähnliche Messresultate bei der benötigten Zeit für diese Testreihe. Zu beobachten ist, dass SQL Server 2014 ab einer Anzahl von größer oder gleich 1.000 Datensätzen grundsätzlich schneller ist als die 2016er Version. Bei einer geringeren Anzahl ist SQL Server 2016 insbesondere beim dem Einfügen mittels einer Stored Procedure oder einem Statement deutlich schneller als SQL Server 2014. Die Verwendung eines Prepared Statements stellt in beiden Datenbanksystemen die schnellste Möglichkeit für das Hinzufügen von Datensätzen dar, gefolgt von der Stored Procedures. Das Statement bildet den Abschluss als langsamste Variante des Einfügens. Im Vergleich mit anderen Datenbanksystemen bezüglich der Nutzung einer Stored Procedure für das Einfügen von Daten stellen die beiden Systeme die schnellsten dar, wobei zu bemerken ist, dass bei einer kleineren Anzahl von Daten SQL Server 2016 das schnellste Hauptspeicherdatenbanksystem ist. Bei größeren Mengen ist jedoch SQL Server 2014 das System mit den besten Messergebnissen. Die Gegenüberstellung mit den anderen Datenbanksystemen beim Gebrauch eines Prepared Statements zeigt, dass SQL Server 2014 und 2016 unter dem Aspekt der benötigten Zeit im Mittelfeld liegen. Bei der Verwendung eines Statements zählen beide Systeme bei großen

Bei der Betrachtung von EXASOL bei der Verwendung eines Prepared Statements fällt erstens auf, dass keine Messung für einen einzigen Datensatz vorhanden ist, da das System hier mit einer Fehlermeldung reagiert. Zweitens benötigt EXASOL bei der

Verwendung eines Prepared Statements für das Einfügen von kleinen Datenmengen deutlich mehr Zeit als die den anderen Systemen. Drittens benötigt dieses System für 100 Datensätze weniger Zeit als für 10. Viertens und letztens wird bei steigenden einzufügenden Datenmengen der Effekt einer einsetzenden linearen Skalierung bezüglich der benötigten Zeit erst bei dem Übergang von 100.000 zu einer Millionen Datensätzen beobachtbar. Davor verhält sich das System deutlich besser.
Die Testreihe für das Einfügen mittels Statement wurde bei EXASOL nach den Messungen für 1.000 Datensätzen abgebrochen, da die benötigte Zeit stark ansteigt und das System deutlich erkennbar schlechter als linear skaliert. Somit stellt die Nutzung von Prepared Statement die beste Möglichkeit des Einfügens von Daten in EXASOL dar. Der Vergleich dieses Datenbanksystems mit den anderen zeigt, dass EXASOL unter der Verwendung eines Prepared Statements für große Datenmengen das schnellste Datenbanksystem darstellt, während es bei kleineren Mengen das langsamste bzw. zweitlangsamste System ist. Der Gebrauch von Statement für das Hinzufügen von Daten stellt die langsamste Form dar in der Gegenüberstellung mit den anderen Hauptspeicherdatenbanken.

Für VoltDB ist feststellbar, dass für das Hinzufügen von Datensätzen zur Datenbank der Client deutlich schneller ist als die Möglichkeiten mittels eines Prepared Statement oder eines Statements bei der Nutzung dieses Datenbanksystems. Die Nutzung des Clients im Vergleich zu den Möglichkeiten des Einfügens bei den anderen Hauptspeicherdatenbanksystemen stellt klar, dass VoltDB unter Anwendung der herstellereigenen Lösung bei größeren Mengen zu den schnellsten Datenbanksystemen in Bezug des Hinzufügens von Daten zählt. Bei kleineren Mengen befindet sich unter dem Zeitaspekt der Client im Mittelfeld. Die Gegenüberstellung der Resultate bei Prepared Statment und Statement mit den Resultaten der Systeme der anderen Hersteller zeigt, dass insbesondere bei größer werdenden Mengen VoltDB zu den langsamsten Systemen bei diesen Formen des Hinzufügens von Daten zählt.

In der folgenden Abbildung 4.6 wird der Speicherverbrauch der Datenbanksysteme bei den unterschiedlichen Arten des Einfügens dargestellt und diese stellen die Basis für die nachfolgenden Ausführungen dar.

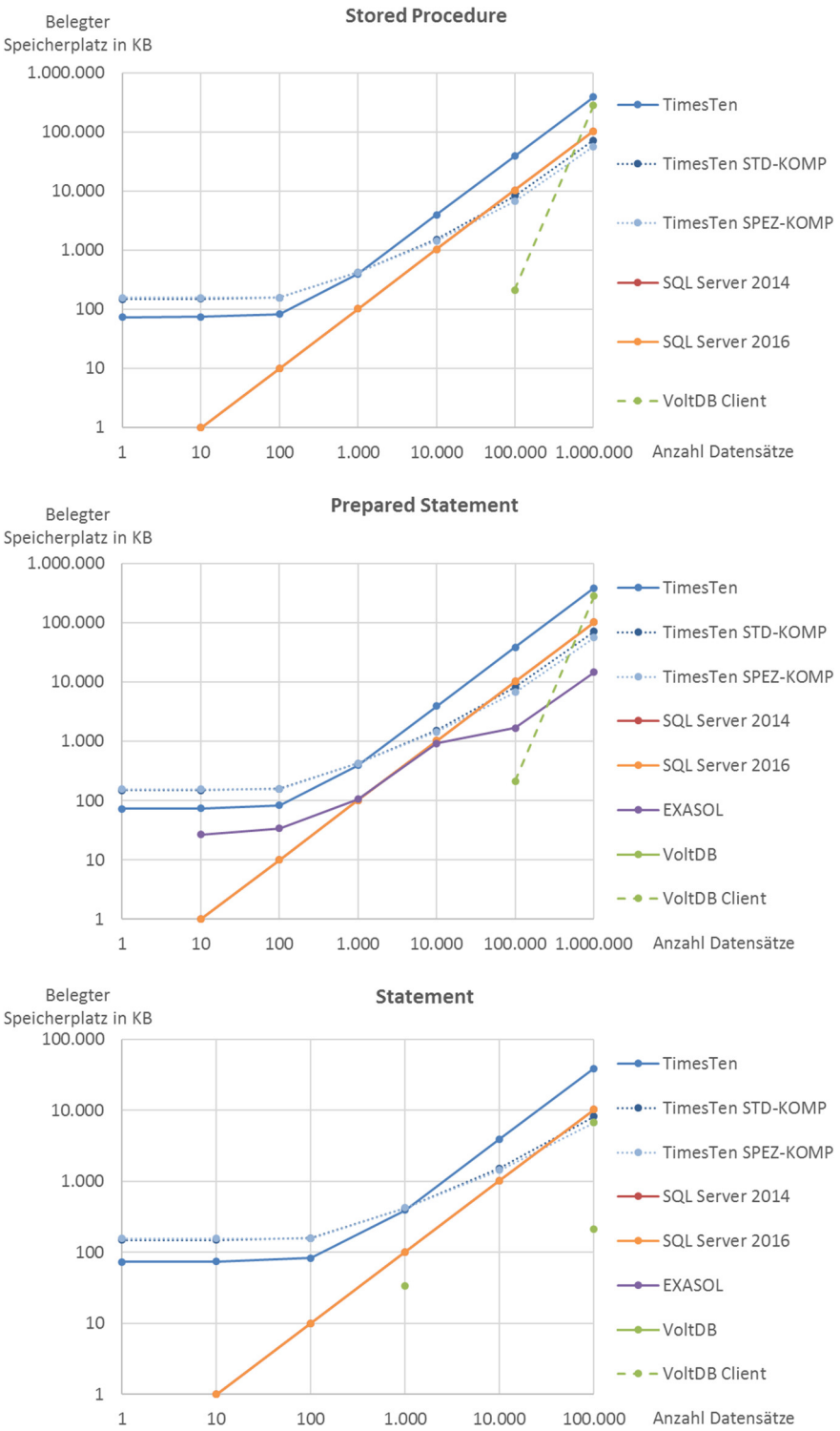

Abbildung 4.6 Belegter Speicherplatz in KB bei INSERT CUSTOMER

Bei der Betrachtung von TimesTen ist zu erkennen, dass bei allen Formen des Einfügens und unabhängig von der Nutzung bzw. Nichtnutzung von Kompression bei kleinen Datenmengen deutlich mehr Speicher für die Datenhaltung aufgewendet wird als bei den anderen Datenbanksystemen. Darüber hinaus festzustellen ist, dass die Nutzung von Kompression bis zu einer Anzahl von 1.000 Datensätzen mehr Speicher in Anspruch nimmt als die Nichtverwendung von Komprimierungsmöglichkeiten. Bei Datenmengen größer als der eben genannte Wert ist erkennbar, dass die Datenspeicherung unter Nutzung von Kompression deutlich weniger Speicher beanspruchen als die nicht komprimierten Daten.
Der Vergleich der einzelnen Messergebnisse zwischen den unterschiedlichen Arten des Einfügens ermöglicht die Betrachtung, dass es bezüglich des belegten Speichers untereinander Abweichungen gibt, die jedoch geringer als 1% sind. Bei der Anschauung der Graphen in Abbildung 4.6 lässt sich erkennen, dass bis zu einer Anzahl von 1.000 Datensätzen der Speicherverbrauch der Daten deutlich besser als linear skaliert und sich dann der linearen Skalierung annähert. Die Gegenüberstellung von TimesTen beim Speicherverbrauch mit den anderen Datenbanksystemen zeigt, dass ohne Datenkompression der belegte Speicher den höchsten und somit schlechtesten Wert aufweist. Wird die Betrachtung der Komprimierungsanwendungen mit in den Vergleich aufgenommen, so zeigt sich, dass bei größeren Datenmengen aus der Perspektive des von den Daten beanspruchten Speichers TimesTen zu den Systemen mit dem geringsten Verbrauch und somit zu den Besten gehört.

Die Betrachtung des Speicherverbrauchs bei den Datenbanksystemen SQL Server 2014 und SQL Server 2016 führt zu der Feststellung, dass die beiden Systeme bei den Tests die exakt gleiche Menge an Hauptspeicher für die Speicherung der Daten aufwenden. Aus den Verläufen der Graphen für SQL Server 2014 und SQL Server 2016 in der Abbildung 4.6 lässt sich erkennen, dass diese Systeme linear skalieren. Im Vergleich mit den anderen Datenbanksystemen lässt sich klar erkennen, dass für kleinen Datenmengen beide Systeme den geringsten Speicherverbrauch aufweisen. Bei einer steigenden Anzahl an Daten liegen SQL Server 2014 und SQL Server 2016 im Mittelfeld, wenn die Nutzung der Datenkompression von TimesTen in die Feststellung einbezogen wird. Ansonsten zählen beide zu den Hauptspeicherdatenbanksystemen mit den geringsten Speicherverbräuchen für die Daten.

Das Datenbanksystem EXASOL lässt, wie schon bei der der Zeitmessung, kein Einfügen eines einzelnen Datensatzes im Batch-Modus zu. Bei der Betrachtung der Anzahl der Datensätze ab 10 Stück bei der Nutzung eines Prepared Statement fällt auf, dass dieses System aufgrund der automatisch durchgeführten Komprimierung beim Einfügen mittels eines Prepared Statements deutlich weniger Speicher für die Datenhaltung in Anspruch nimmt als die anderen Datenbanksysteme. Bis zu einer Anzahl von 100.000 Datensätzen lässt sich feststellen, dass EXASOL erheblich besser als linear skaliert.
Das Einfügen mittels eines Statements weist die Besonderheit auf, dass die Messwerte für den Speicherverbrauch stets die gleichen Werte für die jeweils gleiche Anzahl an Datensätzen ausweisen, jedoch mit der Ausnahme, dass im jeweils ersten Warmup ein Unterschied zwischen den Speicherverbrauchsangaben gemessen wurde. Die Gegenüberstellung von EXASOL mit den anderen Datenbanksystemen bezüglich des Speicherverbrauchs der eingefügten Testdaten zeigt, dass insbesondere bei großen Datenmengen EXASOL deutlich weniger Speicher für die Datenhaltung aufwendet als die anderen Systeme. Bei kleinen Mengen ist EXASOL, aus der Perspektive des von den Daten beanspruchten Speichers, das drittbeste Datenbanksystem.

Bei VoltDB ist keine korrekte Darstellung des beanspruchten Speichers der Daten möglich, da die Messwerte vor und nach dem Einfügen bei jeder Form unterschiedlich sind und zu beobachten ist, dass die Werte vor dem Hinzufügen von Datensätzen teilweise höher sind als nach dem Hinzufügen. Die Graphen bzw. die dazugehörigen Punkte der Versuche mit den unterschiedlichen Arten des Einfügens in der Abbildung 4.6 dienen der Veranschaulichung, in welchem Bereich sich der Verbrauch möglicherweise tatsächlich befindet. Eine Gegenüberstellung von VoltDB mit den anderen Datenbanksystemen findet aufgrund der ungewöhnlichen Messwerte nicht statt.

4.3.1.2 Auswertungen zu INSERT CUSTOMER ONE BY ONE

Bei den Tests bezüglich dieser Art des Einfügens von Customer-Datensätzen im Batch-Modus ließen sich einige Versuche gänzlich nicht durchführen. In EXASOL sind Stored Procedures nicht vorhanden, sodass in diesem Fall keine Auswertung für dieses Datenbanksystem stattfindet.

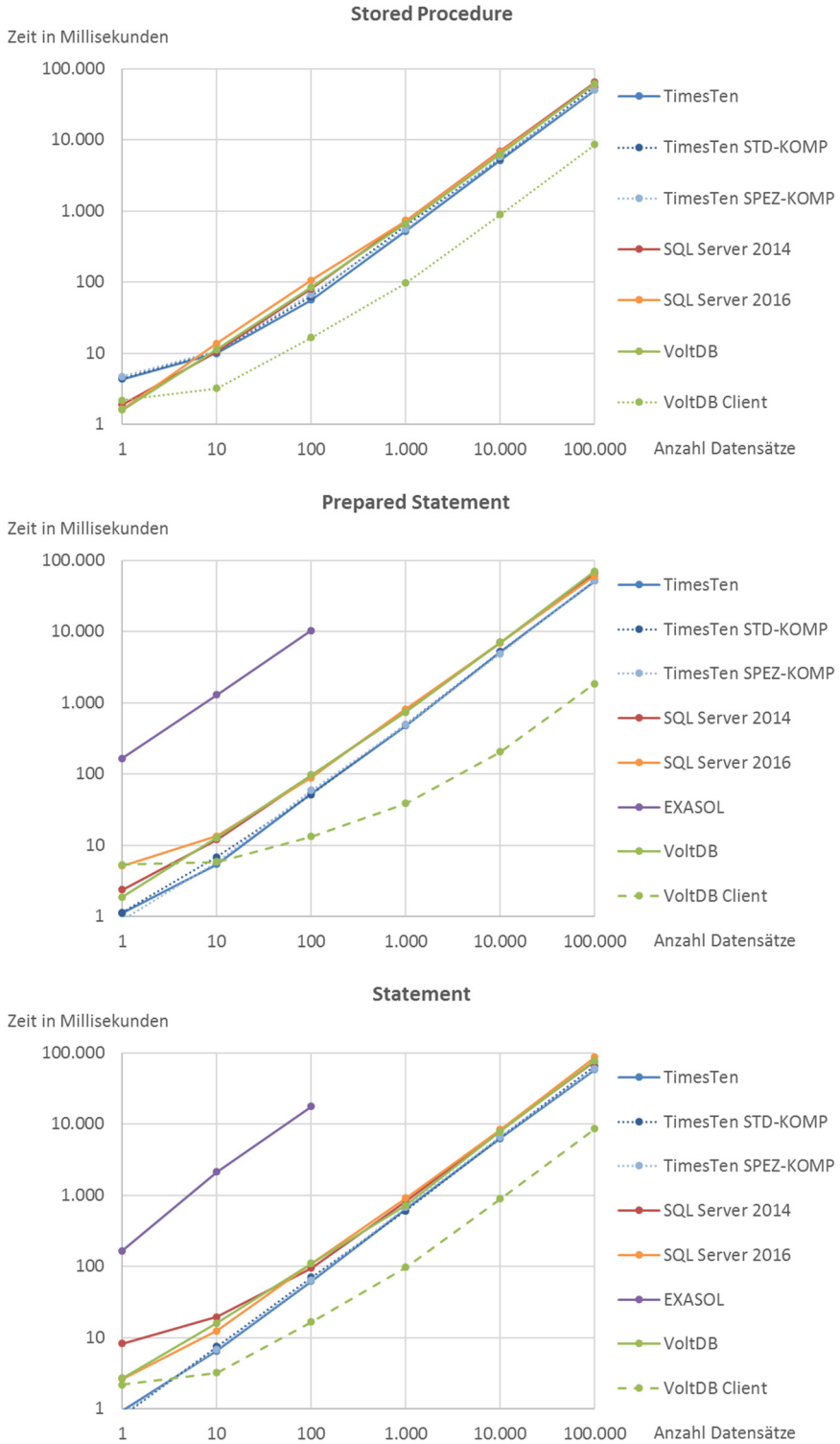

Abbildung 4.7 Benötigte Zeiten in ms bei INSERT CUSTOMER ONE BY ONE

Die Abbildung 4.7 auf der vorherigen Seite stellt dar, wie sich die Datenbanksysteme bezüglich der zeitlichen Skalierung beim Einfügen von steigenden Datenmengen verhalten.

Die Betrachtung der graphischen Auswertung für TimesTen zeigt, dass die zeitlichen Unterschiede zwischen dem Einfügen von Daten unter Verwendung von Kompression gegenüber der Nichtnutzung der Datenkomprimierung bei dem nacheinander erfolgenden Hinzufügen von einzelnen Datensätzen geringer ausfallen als im Batch-Modus. Die fast lineare Skalierung lässt sich sowohl bei dem Einsatz eines Statements als auch bei der Verwendung eines Prepared Statements, bereits ab 10 Datensätzen feststellen. Ab 100 Datensätzen ist diese Skalierung auch bei dem Gebrauch einer Stored Produre erkennbar. Dabei ist die Verwendung der Stored Procedure bei der Nichtnutzung der Kompression und der Datenkomprimierung anhand einer Spezifikation schneller als die Pendants bei der Nutzung eines Prepared Statements. Der Gebrauch von Statement stellt, wie beim Batch-Modus, die langsamste Form des Einfügens dar. Im Vergleich mit den anderen Datenbanksystemen stellt TimesTen bei allen Möglichkeiten des Einfügens bei größer werdenden Datenmengen und unter Nutzung bzw. Nichtnutzung der das schnellste System dar. Einzig beim Einfügen von weniger als 10 Datensätzen mittels einer Stored Procedure stellt sich TimesTen als das langsamste System heraus.

SQL Server 2014 und SQL Server 2016 haben im Vergleich untereinander eine ähnliche Zeitbeanspruchung beim Einfügen. Die markantesten Unterschiede sind bei der Nutzung eines Prepared Statements und eines Statements sichtbar. In SQL Server 2014 ist der Einsatz eines Prepared Statements schneller als SQL Server 2016 bei Datenmengen von bis zu zehn Datensätzen, während es sich bei der Verwendung von Statement genau umgekehrt verhält. Mit steigenden Datenmengen kehrt sich dieses Verhältnis um. In beiden Datenbanksystemen stellt das Einfügen mittels Statement bei größer werdenden Datenmengen die langsamste Form des Einfügens dar. In SQL Server 2014 ist die Verwendung einer Stored Procedure die schnellste Möglichkeit des Einfügens in Bezug auf große Datenmengen und Einzelverarbeitung dar. Bei SQL Server 2016 stellt stattdessen die Nutzung eines Prepared Statements die schnellste Variante des Hinzufügens von Daten dar. Für beide Systeme kann festgestellt werden, dass diese annähernd linear skalieren.
In der Gegenüberstellung von SQL Server 2014 und SQL Server 2016 mit den anderen Datenbanksystemen bezüglich des Einfügens von Daten mittels einer Stored

Procedure zeigt sich, dass SQL Server 2014 bei größeren werdenden Datenmengen langsamer ist als die anderen Systeme. SQL Server 2016 ist hierbei das zweitschnellste Datenbanksystem. Für den Gebrauch eines Prepared Statements bzw. eines Statements für das Hinzufügen von Daten lässt sich für beide Systeme feststellen, dass diese aus der Perspektive der benötigten Zeit im Mittelfeld liegen.

Bei EXASOL wurden die Testreihe für Statement und Prepared Statement nach dem Durchlauf mit 100 Datensätzen abgebrochen, da dieses Datenbanksystem, wie aus der Abbildung 4.7 erkennbar, gegenüber den anderen Datenbanksystemen deutlich mehr Zeit benötigt in der Einzelverarbeitung von Datensätzen und ein Fortführen der jeweiligen Testreihen aus zeitlicher Sicht unverhältnismäßig zum angestrebten Nutzen sind. Ein Vergleich der gemessenen Werte von EXASOL mit den jeweils entsprechenden Resultaten der anderen Datenbanksysteme zeigt auf, dass für das Einfügen von Daten unter Verwendung eines Prepared Statements bzw. eines Statements dieses Datenbanksystem das langsamste System darstellt.

Das Datenbanksystem VoltDB skaliert sowohl bei der Nutzung einer Stored Procedure, eines Prepared Statement und eines Statements als auch bei dem Client annähernd linear. Herausragend ist, dass der Client deutlich weniger Zeit für das Einfügen von Daten in der Einzelverarbeitung benötigt als die anderen Möglichkeiten. Die Betrachtung der Anwendung des Clients stellt klar, dass sich diese Software sich für das Hinzufügen von Datensätzen in der Einzelverarbeitung, insbesondere bei großen Datenmengen, eignet. Die Untersuchung der Ergebnisse bei der Nutzung einer Stored Procedure, eines Prepared Statements und eines Statements zeigen, dass diese Formen des Einfügens von Daten im Vergleich mit den Möglichkeiten der anderen Datenbanksysteme nur mittelmäßig schnell sind.

Die auf der nächsten Seite folgende Abbildung 4.8 stellt den Speicherbedarf der unterschiedlichen Datenmengen in den jeweiligen Datenbanksystemen dar. Für alle Datenbanksysteme außer EXASOL wird für die Auswertung der Speicher-beanspruchung auf Unterkapitel 4.3.1.1 verwiesen. Die Begründung dafür ist, dass sich die Testreihen zwar bezüglich der Art des Einfügens unterscheiden, jedoch in beiden die gleichen Daten genutzt werden und somit gleichwertige Betrachtungen bezüglich des Speicherverbrauchs der Daten bei allen Datenbanken mit Ausnahme von EXASOL gegeben sind.

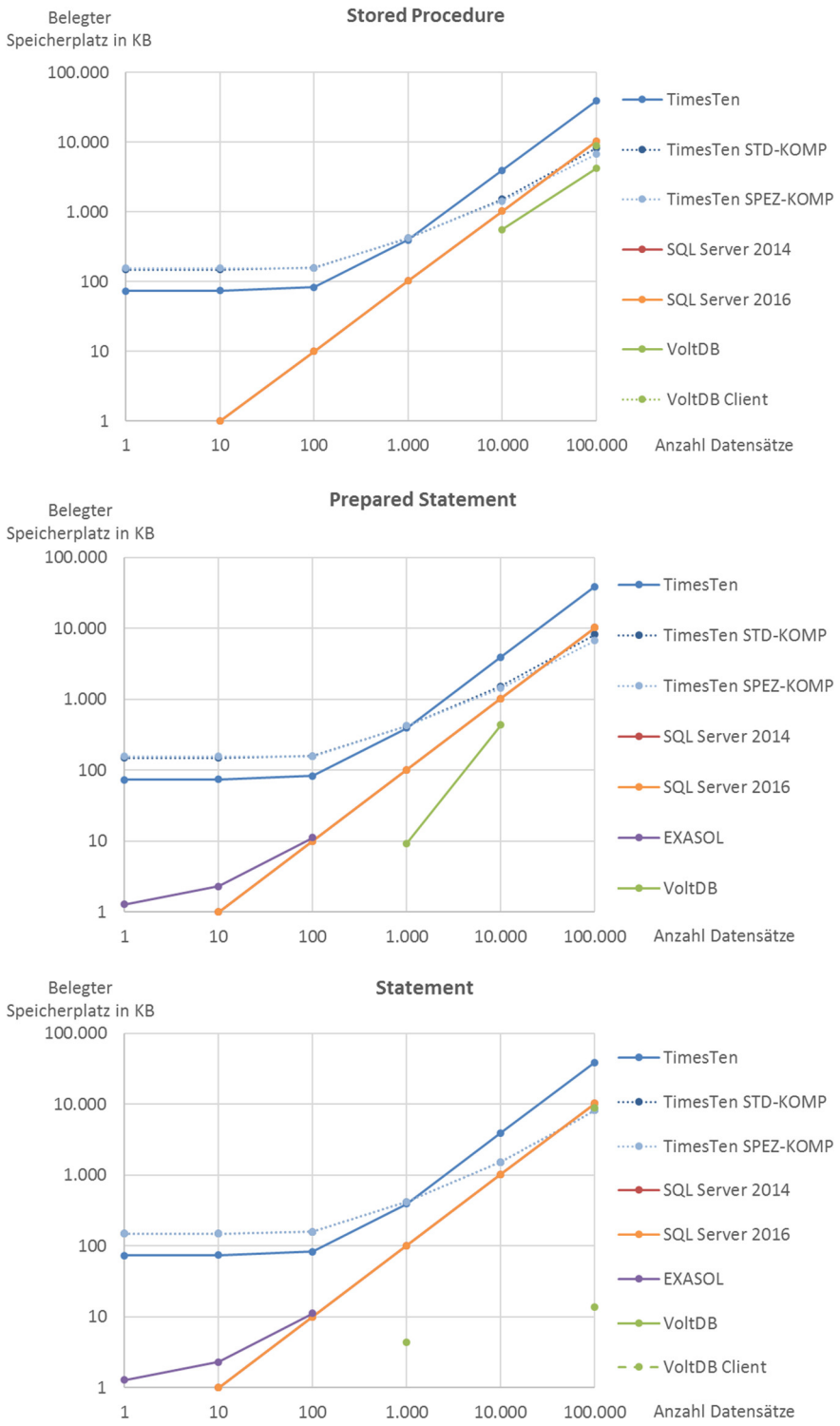

Abbildung 4.8 Belegter Speicherplatz in KB bei INSERT CUSTOMER ONE BY ONE

In der Auswertung bei EXASOL sind Unterschiede zwischen der Verwendung eines Prepared Statements und eines Statements feststellbar, die nachfolgend aufgeführt werden. Zum einen ist im Vergleich zum Batch-Modus auffallend, dass für die gleichen eingefügten Daten weniger Speicherplatz benötigt wird. Zum anderen ist es bei der Einzelverarbeitung in EXASOL möglich für einen einzigen Datensatz den Speicherbedarf zu bestimmen. Die Betrachtung der bezüglich der Skalierbarkeit beim Speicherverbrauch entfällt, da nicht ausreichend Daten diesbezüglich verfügbar sind.

4.3.1.3 Auswertungen zu INSERT ALL

Die Versuche unter Verwendung von Stored Procedures wurden in EXASOL nicht durchgeführt, da diese Funktionalität in diesem Datenbanksystem nicht vorhanden ist. Bei SQL Server 2014 wurden die Tests mit Foreign-Keys nicht ausgeführt, da diese im Hauptspeicherbetrieb nicht existieren. Für VoltDB wurden anstelle der Testreihen mit Prepared Statements und Stored Procedures der VoltDB eigene Client verwendet.

Die folgende Tabelle 4.3 beinhaltet die Messungen für die jeweils benötigte Zeit in Millisekunden für das Einfügen von Daten in die Tabellen des Testschemas.

Das Datenbanksystem TimesTen weist, wie schon beim Einfügen von Customer-Datensätzen in Unterkapitel 4.3.1.1 die Tatsache auf, dass das Hinzufügen von Daten mittels Prepared Statements deutlich schneller ist als die Verwendung von Stored Procedures. Dieser Sachverhalt lässt sich auch auf die Tests übertragen, die die Datenkompression dieser Hauptspeicherdatenbank nutzen. Im Vergleich der beiden verwendeten Möglichkeiten der Datenkomprimierung ist die Kompression anhand einer Spezifikation in den Versuchen stets die schnellere Variante. Auffallend ist die zeitliche Differenz zwischen den Tests, die die Auswirkung der Foreign-Key-Funktionalität untersuchen. Die Werte der einzelnen Differenz zwischen den Tests unter Verwendung von Prepared Statements bzw. Stored Procedures lassen sich dabei in ein Intervall von sechs bis sieben Sekunden eingrenzen.

Im Vergleich zu den anderen Datenbanksystemen stellt bei den Auswertungen zu dem Einfügen mittels Stored Procedures TimesTen die langsamste Datenbank dar. Bei der Verwendung von Prepared Statements ist TimesTen die zweitschnellste Hauptspeicherdatenbank.

Einfügen via	Stored Procedures	Stored Procedures	Prepared Statements	Prepared Statements	Client
Verwendung von Foreign Keys	Ja	Nein	Ja	Nein	Nein
TimesTen	67.525,80	60.584,24	12.930,00	6.737,59	-
TimesTen STD-KOMP	70.884,50	64.492,33	15.929,92	9.249,18	-
TimesTen SPEZ-KOMP	70.278,61	64.014,47	15.724,23	9.190,28	-
SQL Server 2014	-	15.594,74	-	13.990,41	-
SQL Server 2016	22.546,06	20.091,18	20.327,31	15.391,64	-
EXASOL	-	-	5.862,00	5.483,05	-
VoltDB	-	-	-	-	11.944,55

Tabelle 4.3 Benötigte Zeiten in ms bei INSERT ALL

Der Vergleich der Datenbanksysteme SQL Server 2014 und 2016 untereinander beschränkt sich auf die Versuche ohne Foreign-Keys. Die Betrachtung der Werte zeigt, dass SQL Server 2014 sowohl bei der Nutzung von Stored Procedures als auch bei der Verwendung von Prepared Statements das schnellere Datenbanksystem ist. Dabei ist auch erkennbar, dass in beiden Systemen das Einfüguen mittels Prepared Statements schneller ist im Vergleich zu dem Gebrauch von Stored Procedures. In der Einzelbetrachtung von SQL Server 2016 lässt sich feststellen, dass die Tests ohne die Foreign-Key-Funktionalität stets schneller sind als die Versuche, die diese Funktionalität nutzen. SQL Server 2014 ist beim Einfügen von Datensätzen mittels Stored Procedures in alle Tabellen ohne Nutzung von Foreign Keys die schnellste Datenbank und bei den Prepared Statements ist es die zweitlangsamste.

SQL Server 2016 ist beim Hinzufügen von Datensätzen mittels Stored Procedures in alle Tabellen unter Nutzung von Foreign Keys die schnellste Datenbank und ohne Nutzung von Foreign Keys die zweitschnellste. Bei der Verwendung von Prepared Statements ist SQL Server 2016 sowohl bei Tabellen mit Foreign Keys als auch ohne diese, die langsamste Datenbank.

Für EXASOL lässt sich bei der Untersuchung der Werte festhalten, dass auch hier das Weglassen von Foreign Keys das Einfügen der Datensätze positiv beeinflusst. Dieses Datenbanksystem ist im Vergleich mit den anderen Systemen bei der Nutzung von Prepared Statements in den dazugehörigen Versuchen das schnellste Hauptspeicherdatenbanksystem.

In VoltDB wird mittels des Clients das Einfügen von Datensätzen in alle Tabellen des Testschemas erfolgreich durchgeführt, wobei dies ohne die Verwendung von Foreign Keys erfolgt, da diese nicht unterstützt werden. Ein Vergleich des Clients mit den datenbankeigenen Möglichkeiten der Nutzung von Stored Procedures und Prepared Statements wird nicht durchgeführt, da das Einfügen von Datensätzen mittels Stored Procedures im Batch-Modus, wie in Unterkapitel 4.3.1.1 erläutert, nicht möglich ist und im Gegensatz zu dem Client die Stapelverarbeitung unter der Nutzung von Prepared Statements nur langsam verläuft. Die Gegenüberstellung der gemessenen Zeit des Clients mit den Werten der anderen Datenbanksysteme zeigt, dass dieser leistungstechnisch gut abschneidet.

Die nachfolgende Tabelle 4.4 beinhaltet die Messwerte für die in den Datenbanktabellen gespeicherten Daten in Kilobyte. Die Resultate werden für die nachfolgenden Auswertungen der einzelnen Datenbanksysteme in Bezug auf die Datenhaltung verwendet.

Bei TimesTen lässt sich beim Vergleich der Werte feststellen, dass zwischen den unterschiedlichen Formen des Einfügens und Nutzung bzw. Nichtnutzung von Foreign Keys bei allen Datenspeicherungsformen, d.h. ohne Datenkomprimierung, standardmäßiger Komprimierung und Komprimierung anhand einer Spezifikation, Abweichungen im Bereich von weniger als einem Kilobyte gibt. Im Vergleich mit den anderen Datenbanksystemen haben die Daten in den Tabellen des Testschemas ohne Datenkomprimierung den höchsten Speicherbedarf. Bei der Nutzung der standardmäßigen Kompression verringert sich der Bedarf an Speicher und TimesTen

hat den viertniedrigsten Speicheraufwand. Erfolgt die Datenhaltung unter Verwendung der Kompression anhand einer Spezifikation, so hat diese Form Speicherung den zweitniedrigsten Bedarf an Hauptspeicher.

Einfügen via	Stored Procedures	Stored Procedures	Prepared Statements	Prepared Statements	Client
Verwendung von Foreign Keys	Ja	Nein	Ja	Nein	Nein
TimesTen	88.303,58	88.303,60	88.303,55	88.303,37	-
TimesTen Def	54.203,62	54.203,80	54.204,12	54.203,84	-
TimesTen Spez	50.723,66	50.723,66	50.723,77	50.723,65	-
SQL Server 2014	-	50.999,00	-	50.999,00	-
SQL Server 2016	50.999,00	50.999,00	50.999,00	50.999,00	-
EXASOL	-	-	7.421,59	7.505,94	-
VoltDB	-	-	-	-	70.907,80

Tabelle 4.4 Belegter Speicherplatz in KB bei INSERT ALL

Die Betrachtung der Resultate für SQL Server 2014 und SQL Server stellt eindeutig klar, dass die Art des Einfügens keinen Einfluss auf die Datenspeicherung in den beiden Datenbanksystemen hat und beide Systeme exakt den gleichen Speicherverbrauch haben. Die Gegenüberstellung mit den anderen Hauptspeicher-

datenbanken zeigt, dass die beiden Systeme den zweitniedrigsten bzw. drittniedrigsten[1] Speicherverbrauch haben.

Die Werte der Messungen beim Datenbanksystem EXASOL stellen dar, dass das Einfügen unter Nutzung von Foreign Keys etwas weniger Speicherplatz benötigt als das Weglassen dieser. Bei den einzelnen Resultaten lässt sich feststellen, dass das Datenbanksystem für die leeren Tabellen bei der Nutzung von Foreign Keys bereits Werte größer 0 ausgibt, obwohl kein einziger Datensatz vorhanden ist. Werden nun die Einzelmessungen in der Variante ohne Foreign Keys betrachtet, so fällt auf, dass bei der ersten Warmup-Iteration auch ein Wert für Daten in den Tabellen ausgegeben wird, obwohl kein Datensatz vorhanden ist, und alle nachfolgenden Iterationen 0 als Resultat der jeweiligen Messung der Speicherbelegung vor der Iteration liefern. Im Vergleich zu den anderen Datenbanksystemen benötigt EXASOL den geringsten Speicherplatz für die Datensätze in allen Tabellen des Testschemas.

Für VoltDB ist ein Vergleich mit den anderen Datenbanksystemen bezüglich des Speicherbedarfs der Daten mit Schwierigkeiten verbunden, da in Unterkapitel 4.3.1.1 festgestellt wurde, dass die Speichermessungen ungewöhnliche Resultate liefern. In Anbetracht des Wertes lässt sich mit Vorsicht sagen, dass das ein realistischer Wert sein kann und sich dann zwischen den Messergebnissen von SQL Server 2014 bzw. 2016 und TimesTen eingliedern lässt. Daraus kann geschlussfolgert werden, dass unter diesem Gesichtspunkt VoltDB den zweithöchsten Speicherverbrauch für die Daten des Testschemas hat.

[1] Im Vergleich zu den Werten bei TimesTen, da hier die standardmäßige Kompression und das Weglassen der Kompression für die Daten einen höheren Speicherverbrauch bedeuten, aber die Kompression der Daten anhand einer Spezifikation einen geringeren Speicherbedarf zur Folge hat.

4.3.2 Auswertung der Versuche aus der Kategorie SELECT

In diesem Unterkapitel werden die Resultate der auf Abfragen basierenden Versuche betrachtet. Eine der Auswertungen ermittelt, ob die Skalierbarkeit bei dem jeweils betrachteten Datenbanksystem gegeben ist, wenn dieselbe Abfrage bei wachsenden Datenmengen ausgeführt wird. Die anderen Abfragen dienen der Feststellung der Eignung der Datenbanksysteme für Abfragen auf größeren Datenmengen.

An dieser Stelle ist vorab zu erwähnen, dass für EXASOL je zwei Messergebnisse ausgewertet werden. Der Grund dafür ist, dass EXASOL intern Abfrageergebnisse standardmäßig verfügbar hält. Dadurch wird eine Abfrage bei erstmaliger Abarbeitung durchgeführt und wenn die Daten, auf denen diese Abfrage durchgeführt wurde, nicht verändert wurden, während dieselbe Abfrage nochmal ausgeführt werden soll, so wird auf das gespeicherte Ergebnis zurückgegriffen und dieses ausgeliefert, anstelle dieses neu zu berechnen. Die beiden betrachteten Werte sind zum einen das Ergebnis der erstmaligen Berechnung, nachfolgend in der Legende als EXASOL Erstberechnung gekennzeichnet, zum anderen das Resultat der Messungen mit der Verfügbarkeit des Ergebnisses der Abfrage, welches nachfolgend einfach als EXASOL gekennzeichnet wird.

Die Darstellungen der Balken in den Abbildungen der Auswertungen zu SELECT 2 – 6 basieren auf dem arithmetischen Mittelwert der Resultate für die benötigte Zeit.

4.3.2.1 Auswertung zu SELECT 1

Mit Hilfe des Liniendiagramms in der Abbildung 4.9 auf der nächsten Seite, das den Verlauf der Verarbeitungszeiten für die Abfrage SELECT 1 bei steigenden Datenmengen darstellt, soll die zeitliche Skalierung des jeweiligen Datenbanksystems bei Abfragen ermittelt und anschließend mit den anderen Systemen verglichen werden.

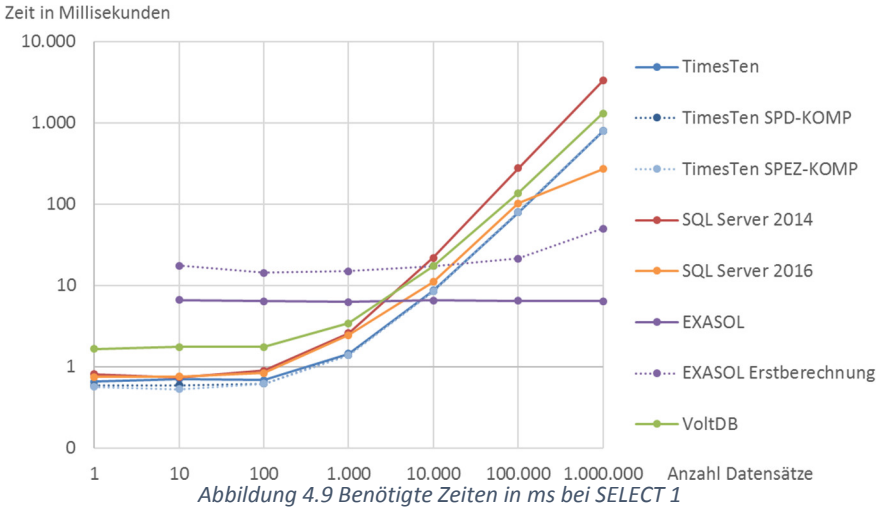

Abbildung 4.9 Benötigte Zeiten in ms bei SELECT 1

Das Diagramm zeigt, dass ein Grenzwert für die Anzahl der Datensätze, nämlich 100, existiert, ab dem alle Datenbanksysteme außer EXASOL bei steigenden Datenmengen anfangen linear zu skalieren.

In TimesTen sind die Differenzen der zeitlichen Messungen zwischen den Abfragen auf komprimierten und nicht komprimierten Datensätzen gering. Bei der Betrachtung stellt sich heraus, dass bei größer werdenden Datenmengen die Abfrage auf den nicht komprimierten Daten schneller ist als auf den komprimierten Daten. Im Gegensatz liefert die Abfrage auf den anhand einer Spezifikation komprimierten Daten bei kleinen Mengen am schnellsten das Ergebnis. Bei einer Anzahl von weniger als 10.000 Datensätzen liefert TimesTen im Vergleich zu den anderen Datenbanksystemen am schnellsten die Ergebnisse. Für größere Mengen als 10.000 Datensätze stellt TimesTen das zweit- bzw. drittschnellste System dar.

Bei SQL Server 2014 und SQL Server 2016 lässt sich feststellen, dass diese sich bei einer Anzahl von bis zu 1.000 Datensätzen ähnliche Werte für die benötigte Zeit liefern. Bei größeren werdenden Mengen von Datensätzen ist für SQL Server 2016 erkennbar, dass es im Vergleich zu den anderen Datenbanksystemen langsamer wird und die Annäherung an die lineare Skalierung hier eher stärker einsetzt als bei den anderen Datenbanksystemen. Bei der größten Datenmenge ist SQL Server 2014 das langsamste System. Im Gegensatz dazu ist SQL Server 2016 das zweitschnellte Datenbanksystem bei der größten Menge und lässt sich bei Mengen zwischen 10.000 und ungefähr 100.000 als das drittschnellste System identifizieren.

Die Betrachtung der Ergebnisse für EXASOL beginnt ab einer Menge von 10 Datensätze, da im Batch-Modus, wie in Unterkapitel 4.3 erwähnt, das Einfügen von einem Datensatz nicht möglich ist. Die Betrachtung der Messwerte bei dem jeweils bereits zur Verfügung stehenden Ergebnis der Abfragen auf den unterschiedlichen Datenmengen zeigt, dass die Resultate in fast konstanter Zeit abgerufen werden können. Die Analyse des gemessenen Einzelwertes für die Zeit bei der jeweils erstmaligen Verarbeitung der Abfrage auf den verschiedenen Mengen an Daten lässt erkennen, dass ab Datenmengen mit einer Anzahl von mehr als 10.000 Datensätzen EXASOL das schnellste Datenbanksystem ist.

VoltDB ist bei den vorhandenen Messergebnissen für Mengen mit einer Anzahl von weniger als 10 Datensätzen das langsamste Datenbanksystem. Im Vergleich zu den anderen Systemen ist VoltDB bei steigenden Datenmengen das zweitlangsamste System.

4.3.2.2 Auswertung zu SELECT 2

Das Diagramm in der nachfolgenden Abbildung 4.10, dass die Unterschiede in der benötigten Verarbeitungszeit der Datenbanksysteme für die Abfrage SELECT 2 darstellt, zeigt, dass EXASOL im Vergleich mit den anderen Datenbanksystemen sowohl bei der Betrachtung des Messwertes der Zeit bei der ersten Berechnung als auch beim Abruf des gespeicherten Ergebnisses das schnellste System darstellt.

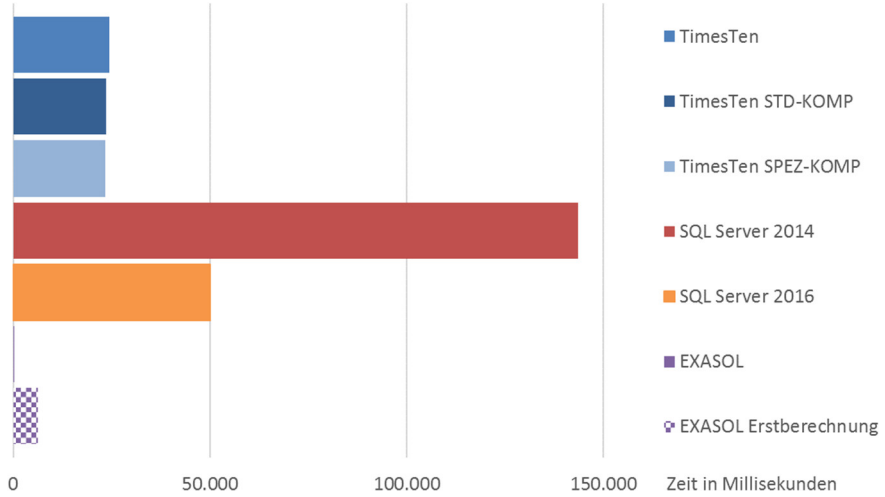

Abbildung 4.10 Benötigte Zeiten in ms bei SELECT 2

TimesTen ist das zweitschnellste Datenbanksystem, wobei die Abfrage auf den anhand einer Spezifikation komprimierten Daten am schnellsten durchgeführt wurde, gefolgt von der Verarbeitung auf den Daten, die unter Verwendung der standardmäßigen Kompression gespeichert wurden. Am langsamsten wurde die Abfrage auf den nicht komprimierten Daten durchgeführt.

Im Vergleich der beiden Datenbanksysteme von Microsoft untereinander ist erkennbar, dass SQL Server 2014 das signifikant schnellere System ist bei der Durchführung der Abfrage. Bei der Gegenüberstellung mit den anderen Datenbanksystemen ist SQL Server 2014 das zweitlangsamste System während SQL Server 2016 das langsamste darstellt. In SQL Server 2016 wurde dieser Versuch im ersten Durchgang abgebrochen, da 15 Minuten nach dem Start des Tests noch kein Ergebnis vorlag und kein Timeout auftrat. Ein zweiter Durchgang nach dem Neustart des Datenbankrechners verlief erfolgreich.

In VoltDB wurde die Verarbeitung der Abfrage SELECT 2 nach 5 Minuten abgebrochen, da die Grenze für die maximale Verarbeitungszeit überschritten worden ist.

4.3.2.3 Auswertung zu SELECT 3

Mit Hilfe des Balkendiagramms in der Abbildung 4.11 auf der nächsten Seite, das die Unterschiede in der benötigten Zeit für die Verarbeitung dieser Abfrage der Datenbanksysteme darstellt, wird eine Auswertung durchgeführt, die dazu beigeträgt festzustellen, in wie fern sich die Datenbanksysteme für analytische Zwecke eignen.

Bei der Auswertung der gemessenen Zeiten für TimesTen zeigt sich, dass die Durchführung der Abfrage auf den Daten, die unter Gebrauch der standardmäßigen Kompression gespeichert wurden, am schnellsten erfolgte. Die zeitlichen Differenzen zwischen der Verarbeitung der Abfrage auf den Daten, die anhand einer Spezifikation komprimiert gespeichert wurden, und den nicht komprimierten Daten sind äußerst gering. Dabei weist die Abfrage bei den Daten ohne Kompression die schlechteste Zeit im Vergleich untereinander auf. In der Gegenüberstellung zu den anderen Datenbanksystemen ist TimesTen, wenn für EXASOL, statt dem Messergebnis für vorgehaltene Ergebnisse, die Erstberechnung genommen wird, das schnellste System. Ansonsten stellt es das zweitschnellste System dar.

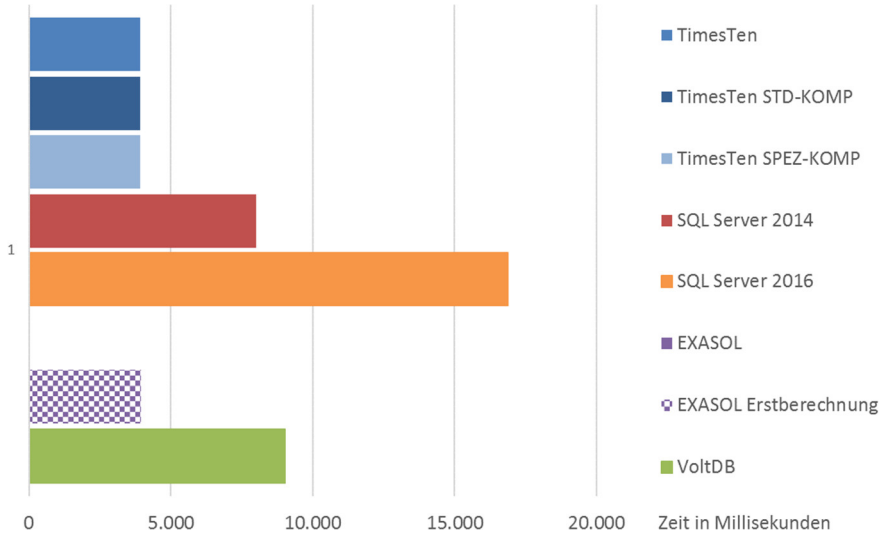

Abbildung 4.11 Benötigte Zeiten in ms bei SELECT 3

Die Betrachtung der Darstellung der Ergebnisse von SQL Server 2014 und SQL Server 2016 im Balkendiagramm stellt klar, dass SQL Server 2014 deutlich schneller ist als SQL Server 2016. Die Gegenüberstellung der Zeiten der beiden Datenbanksysteme mit den anderen Systemen zeigt, dass SQL Server 2014 das drittschnellste Hauptspeicherdatenbanksystem ist, während SQL Server 2016 das langsamste ist.

Für EXASOL lässt sich feststellen, dass es, wenn das Resultat der Zeitmessungen mit dem auf Abruf bereitstehendem Ergebnis der Abfrage als Bemessungsgrundlage genommen wird, das schnellste Hauptspeicherdatenbanksystem im Vergleich mit den anderen Systemen ist. Wird jedoch die Erstberechnung als Grundlage genommen, so ist EXASOL die das zweitschnellste Datenbanksystem.

VoltDB ist das zweitlangsamste Hauptspeicherdatenbanksystem bei der Bearbeitung der Abfrage SELECT 3.

4.3.2.4 Auswertung zu SELECT 4

Wie schon bei den beiden vorherigen Auswertungen wird unter Zuhilfenahme eines Balkendiagramms, nachfolgend in Abbildung 4.12, eine Auswertung durchgeführt werden, die dazu beigeträgt festzustellen, in wie fern sich die Datenbanksysteme für analytische Zwecke eignen. Dabei stellt das Diagramm die Unterschiede des Zeitaufwands für die Verarbeitung der Abfrage SELECT 4 der einzelnen Datenbanksysteme dar.

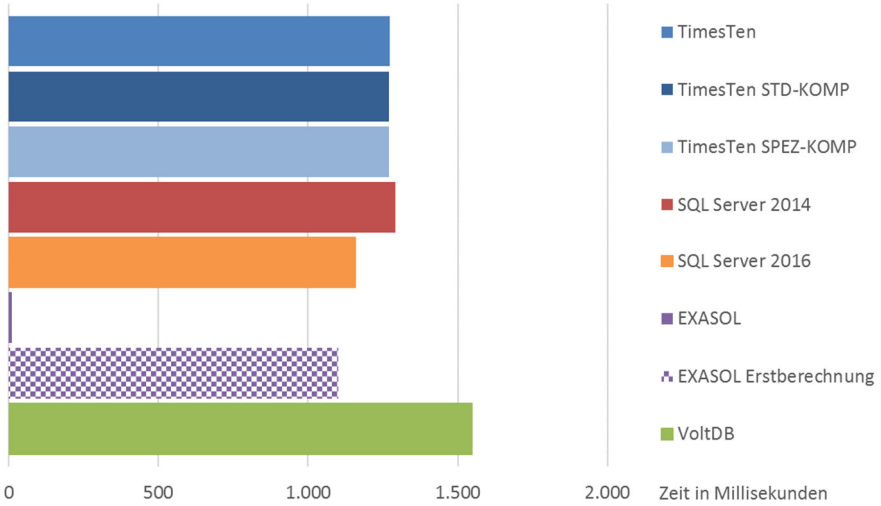

Abbildung 4.12 Benötigte Zeiten in ms bei SELECT 4

Bei der Betrachtung der Datenbanksysteme von Microsoft fällt auf, dass, im Gegensatz zu den vorherigen Abfragen, SQL Server 2016 schneller ist als 2014. Dabei stellt SQL Server 2016 in der Gegenüberstellung mit den anderen Datenbanksystemen das zweitschnellste System dar, während die 2014er Variante von SQL Server die zweitlangsamste darstellt. Anzumerken ist, dass bei den Tests auf die Korrektheit der Abfrage SELECT 4 in beiden Systemen Konvertierungsfehler auftreten und diese die den Test fehlschlagen ließen, aber die Versuche, die Messergebnisse liefern, ohne Fehler erfolgreich absolviert wurden.

EXASOL stellt im Vergleich mit den anderen Systemen sowohl beim Abrufen des gespeicherten Ergebnisses als auch bei der Betrachtung der gemessenen Zeit bei der ersten Berechnung das schnellste System dar.

In der Gegenüberstellung mit den anderen Hauptspeicherdatenbanksystemen ist VoltDB bei der Verarbeitung der vierten Abfrage das langsamste System.

TimesTen ist das drittschnellste Datenbanksystem. Die Abfrage auf den mittels der Kompression anhand einer Spezifikation gespeicherten Daten ist minimal schneller ist als die Verarbeitung der Abfrage bei den anderen zwei Varianten, wobei die langsamere der beiden die auf den nicht komprimierten Daten ist.

4.3.2.5 Auswertung zu SELECT 5

Aus der Betrachtung des Diagramms in der folgenden Abbildung 4.13, dass die Unterschiede in der benötigten Verarbeitungszeit der Datenbanksysteme für die Abfrage SELECT 5 darstellt, lässt sich feststellen, dass EXASOL unter beiden spezifischen Gesichtspunkten, die in 4.3.2 beschrieben sind, das schnellste System darstellt.

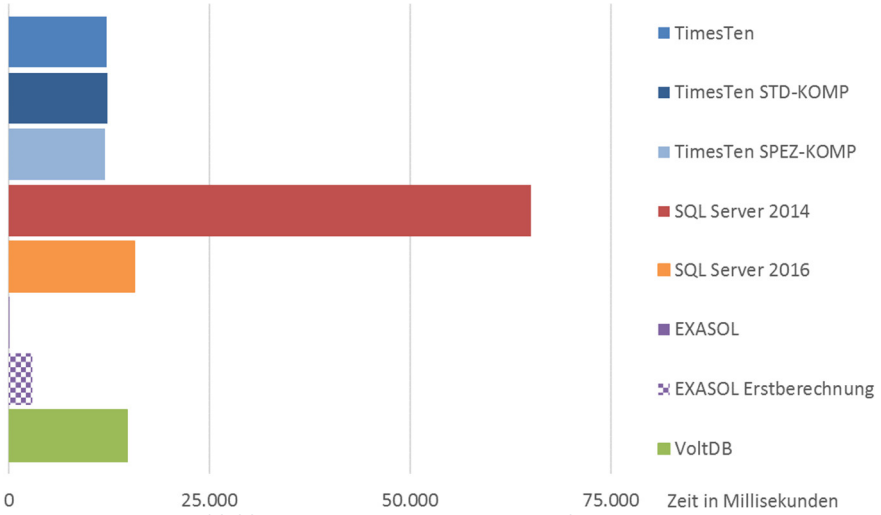

Abbildung 4.13 Benötigte Zeiten in ms bei SELECT 5

Für TimesTen lässt sich festhalten, dass die Verarbeitung der fünften Abfrage auf den Daten, die mittels der Kompression anhand einer Spezifikation gespeichert wurden, den besten Zeitaufwand im internen Vergleich aufweist. Darauf folgt Ergebnisberechnung auf den nicht komprimierten Daten. Die meiste Zeit benötigte die Ausführung der Abfrage auf den standardmäßig komprimierten Daten. In der Gegenüberstellung mit den anderen Datenbanksystemen ist TimesTen das zweitschnellste System.

In der Gegenüberstellung mit den anderen Hauptspeicherdatenbanksystemen ist VoltDB in der Verarbeitung das drittschnellste System.

SQL Server 2014 und SQL 2016 stellen die beiden Datenbanksysteme mit den höchsten Werten für die benötigte Zeit dar. Auffallend ist hier, dass SQL Server 2014 signifikant langsamer ist als SQL Server 2016 und somit das langsamste Datenbanksystem ist.

4.3.2.6 Auswertung zu SELECT 6

An der Betrachtung der Abbildung 4.14 lässt sich erkennen, dass, wie zuvor, unter Zuhilfenahme eines Balkendiagramms, eine Auswertung durchgeführt wird, die dazu beiträgt festzustellen, in wie fern sich die einzelnen Datenbanksysteme für analytische Operationen eignen.

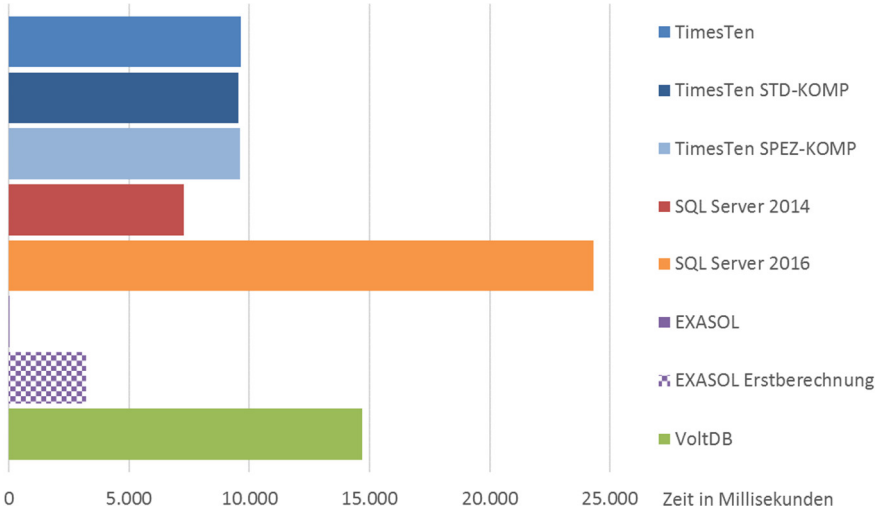

Abbildung 4.14 Benötigte Zeiten in ms bei SELECT 6

EXASOL ist bei dieser Abfrage, wie bereits mehrfach zuvor, das schnellste Hauptspeicherdatenbanksystem.

Der Vergleich zwischen den Datenbanksystemen von Microsoft zeigt, dass SQL Server 2014 deutlich schneller ist als SQL Server 2016. Bei der Gegenüberstellung mit den anderen Hauptspeicherdatenbanksystemen lässt sich feststellen, dass SQL Server 2014 das zweitschnellste System darstellt, während SQL Server 2016 das langsamste ist.

TimesTen ist das drittschnellste System bei der Betrachtung der fünf Datenbanksysteme. Dabei weisen im internen Vergleich die Zeiten für die Bearbeitung der Abfrage auf den Daten, die komprimiert worden sind, niedrigere und somit bessere Werte auf, als das Resultat der Abfrageverarbeitung auf den nicht komprimierten Daten. Das Ergebnis der Abfrage auf den Daten, die unter Gebrauch der standardmäßigen Kompression gespeichert wurden, stellt dabei das Ergebnis am schnellsten zur Verfügung.

VoltDB ist in im Vergleich mit den anderen Datenbanksystemen das zweitlangsamste System.

4.3.3 Auswertung der Versuche aus der Kategorie DELETE

Die Ergebnisse bezüglich des Löschens von Datensätzen anhand bestimmter Kriterien werden in diesem Unterkapitel näher betrachtet. Dazu werden Auswertungen bezüglich der Skalierbarkeit der dabei aufgewendeten Zeit und des freigegebenen Speicherplatzes bei den einzelnen Datenbanksystemen durchgeführt.

Die Aussagen bezüglich des Löschens von Datensätzen oder äquivalenter Formen beziehen immer die Überprüfung der Datensätze auf die entsprechenden Bedingungen mit ein.

4.3.3.1 Auswertung zu DELETE SIMPLE

Die Betrachtung der Abbildung 4.15, in der die zeitliche Skalierung der Datenbanksysteme dargestellt wird, zeigt dass die Systeme sowohl signifikante Unterschiede in der Gegenüberstellung als auch in der Einzelbetrachtung bei größer werdenden Datenmengen haben.

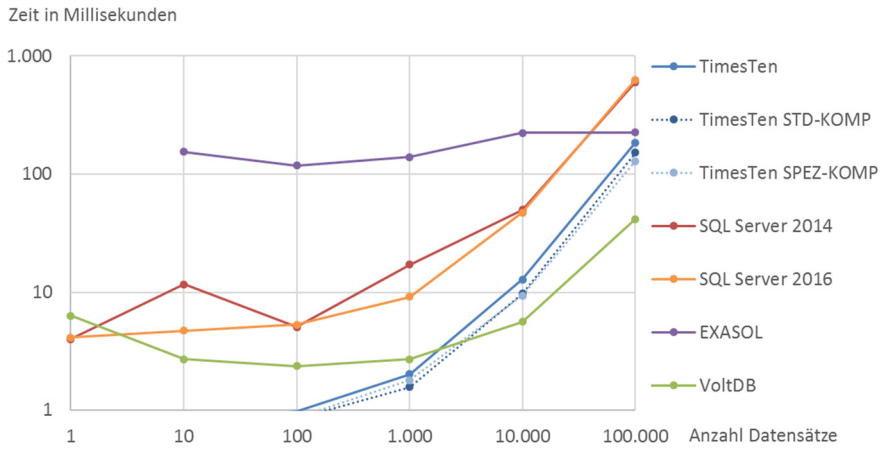

Abbildung 4.15 Benötigte Zeiten in ms bei DELETE SIMPLE

TimesTen ist bei Datenmengen bis etwas mehr als 1.000 Datensätzen bei der Durchführung des Löschvorgangs das schnellste System und bei größeren Datenmengen das zweitschnellste. Übersteigt die Menge der Datensätze die Anzahl von 10.000, ist zu beobachten, dass das System die Tendenz zeigt schlechter als linear zu skalieren. Allgemein lässt sich feststellen, dass in TimesTen das Entfernen

von komprimierten Datensätzen schneller durchgeführt als das von nicht komprimierten Datensätzen.

Das Datenbanksystem VoltDB weist die Besonderheit auf, dass es bei der Durchführung der Löschung bis zu einer Anzahl von bis zu 100 zu betrachtenden Datensätzen schneller wird und dann wieder langsamer. Bei großen zu untersuchenden Datenmengen, bei denen Datensätze auch gegebenenfalls gelöscht werden, ist VoltDB das schnellste Datenbanksystem und weist eine annähernd lineare Skalierung bezüglich der Verarbeitungszeit bei steigenden Datenmengen auf.

Für EXASOL lässt sich festhalten, dass das Ausführen des Löschvorgangs fast immer am langsamsten geschieht. Für das Durchführen des Entfernens von Datensätzen, deren Anzahl sich 100.000 nähert, lässt sich beobachten, dass dies in fast konstanter Zeit geschieht und EXASOL dann nicht mehr das System mit dem höchsten Zeitaufwand ist.

Der Vergleich der beiden Datenbanksysteme von Microsoft untereinander zeigt, dass bei einer Anzahl der Datensätze von bis zu 10.000 Stück SQL Server 2016 schneller ist als SQL Server 2014. Das ändert sich jedoch, wenn die Stückzahl sich 100.000 annähert, da dann festgestellt werden kann, dass SQL Server 2014 für das Löschen weniger Zeit braucht als SQL Server 2016. Bei der Untersuchung der benötigten Zeiten lässt sich bei SQL Server 2016 die Besonderheit feststellen, dass für das Entfernen von 10 Datensätzen mehr Zeit aufgewendet wird als für 100 Datensätze. Bis zu einer Anzahl von 10.000 Datensätzen stellen SQL Server 2014 und 2016 das zweit- bzw. drittlangsamste System dar und bei 100.000 das langsamste bzw. zweitlangsamste Datenbanksystem. Beide Systeme zeigen bei großen Datenmengen eine schlechter als lineare Skalierung bezüglich der benötigten Zeit beim Löschen auf. Darüber hinaus schlug ein Versuch der Datenlöschung bei 10.000 Datensätzen in SQL Server 2014 fehl.

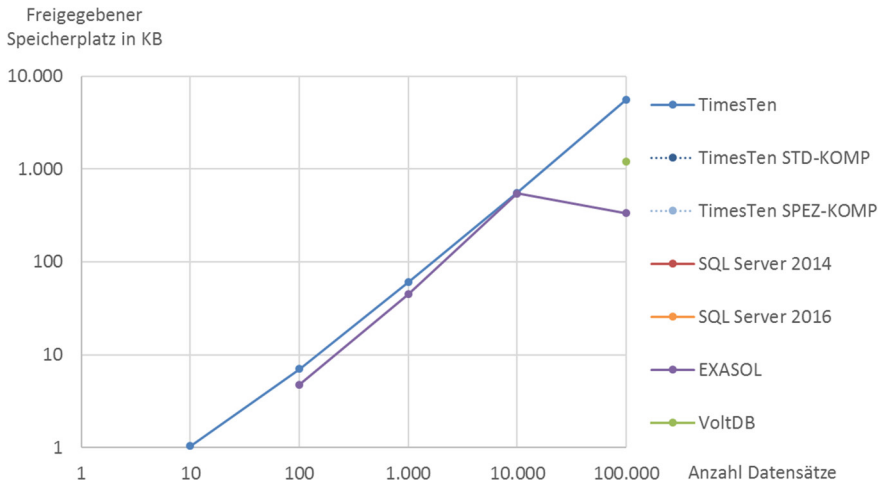

Abbildung 4.16 Freigegebener Speicherplatz in KB bei DELETE SIMPLE

Die obige Abbildung 4.16 stellt die grafische Auswertung der Datenbanksysteme bezüglich der Skalierung des freigegebenen Speichers dar, die nachfolgend näher untersucht wird.

Bei TimesTen ist festzustellen, dass das Freigeben von Speicherplatz bei der Löschung von einer wachsenden Anzahl von Datensätzen fast linear skaliert, wenn die Datensätze nicht komprimiert gespeichert sind. Bei komprimierten Datensätze wird kein Speicherplatz nach der Durchführung der Löschung freigegeben.

Für EXASOL lässt sich aussagen, dass das System bei der Freigabe von Speicher annähernd linear skaliert beim Löschen von Datensätzen, wenn der Löschvorgang auf einer Menge von bis zu 10.000 Datensätzen durchgeführt wird. Bei einer Menge von 100.000 Datensätzen lässt sich feststellen, dass weniger Speicher freigegeben wird als bei 10.000 Datensätzen.

Bei VoltDB wird auf die Besonderheit bei der Speichermessung in 4.3.1.1 hingewiesen, sodass keine Auswertung für dieses Datenbanksystem der Freigabe von Speicher erfolgt.

Bei Messungen bezüglich der Freigabe von Speicherplatz bei SQL Server 2014, SQL Server 2016 und TimesTen mit Nutzung der standardmäßigen Kompression bzw. der Kompression anhand einer Spezifikation in der Datenhaltung betragen die Werte 0, so dass angenommen wird, dass das Freigeben von Speicher nicht direkt nach dem Entfernen der Datensätze aus den Tabellen geschieht.

4.3.3.2 Auswertung zu DELETE COMPLEX

Unter Verwendung der nachfolgenden Abbildung 4.17 wird die Auswertung der Datenbanksysteme beim Löschen von Datensätzen durchgeführt. Dabei stellt die Verläufe der Graphen die Zeit dar, die die Datenbanken bei steigenden Datenmengen benötigen. Auffallend ist, dass diese grafische Auswertung der Abbildung 4.15 in Unterkapitel 4.3.3.1 ähnelt und sich vergleichbare Aussagen zu denen in der Auswertung von DELETE SIMPLE treffen lassen.

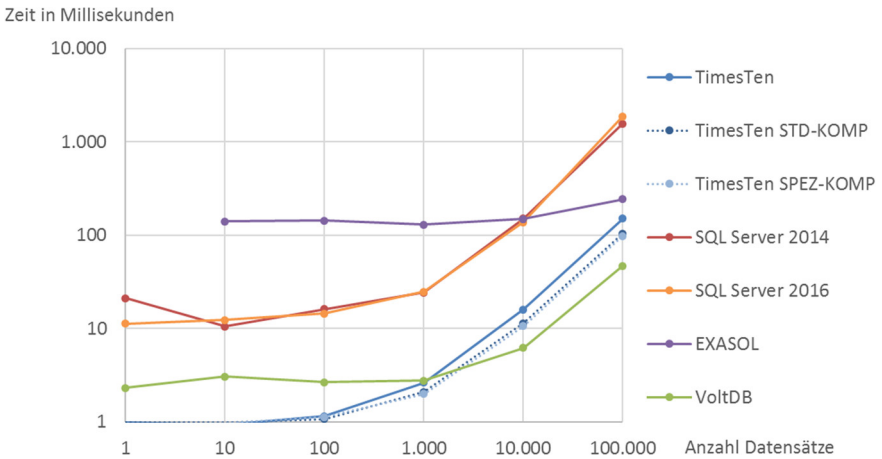

Abbildung 4.17 Benötigte Zeiten in ms bei DELETE COMPLEX

Für TimesTen lässt sich auch hier feststellen, dass es bei Datenmengen bis etwas mehr als 1.000 Datensätzen bei der Durchführung des Löschvorgangs das schnellste System und bei größeren Datenmengen das zweitschnellste. Im Gegensatz zu der Auswertung von TimesTen in 4.3.3.1 ist hier eine lineare Skalierung für den Zeitaufwand des Löschvorgangs bei steigenden Datenmengen zu erkennen. Ebenso wie in der vorhergehenden Auswertung bezüglich des Löschens von Datensätzen wird auch in diesem Versuch das Entfernen von komprimierten Datensätzen schneller durchgeführt als das von nicht komprimierten Datensätzen.

VoltDB weist auch in diesem Test die Besonderheit auf, dass es bei der Durchführung der Löschung ab einer bestimmten Anzahl von zu betrachtenden Datensätzen schneller wird und dann wieder langsamer. Wie zuvor in der Auswertung zu DELETE SIMPLE ist VoltDB bei großen Mengen das schnellste Datenbanksystem bezüglich der Durchführung von Löschungen und zeigt auch hier eine lineare Skalierung in der benötigten Zeit bei steigenden Datenmengen auf.

In der Betrachtung zu EXASOL lässt sich feststellen, dass zwischen 100 und 1.000 zu betrachtenden und gegebenenfalls zu löschenden Datensätzen dieses Datenbanksystem schneller wird und davor und danach langsamer ist. In der Gegenüberstellung zu der Untersuchung der Ergebnisse von DELETE SIMPLE lässt sich hier nicht beobachten, dass eine annähernd konstante Zeit für das Löschen von Datensätzen bei einer Annäherung von 100.000 Datensätzen erreicht wird. Dafür lässt sich eine Tendenz für eine lineare Skalierung feststellen. EXASOL ist auch bei den Löschvorgängen in diesem Versuch fast immer das langsamste Datenbanksystem.

Die Gegenüberstellung der Datenbanksysteme von Microsoft mit den Systemen der anderen Hersteller zeigt auch hier, dass SQL Server 2014 und 2016 bis zu einer Anzahl von 10.000 Datensätzen das zweit- bzw. drittlangsamste System darstellen und bei 100.000 das langsamste bzw. zweitlangsamste Datenbanksystem. Auffallend bei diesem Versuch ist, dass im Vergleich der Microsoft Datenbanksysteme untereinander, im Gegensatz zu der Auswertung von DELETE SIMPLE, hier keine signifikanten Abweichungen bei den benötigten Zeiten für das Löschen auf Basis unterschiedlicher Datenmengen zwischen SQL Server 2014 und SQL Server 16 gibt. Bei der Einzelbetrachtung von SQL Server 2014 ist zu beobachten, dass bei von potentiell 10 zu betrachtenden und gegebenenfalls zu löschenden Datensätze die Zeit besser ist als bei nur einem Datensatz. Wie schon in der Auswertung zu DELETE SIMPLE lassen sich für beide Datenbanksysteme eine schlechter als lineare Skalierung der benötigten Zeit bei steigendem Wachstum der Datenmenge feststellen. Zu dem schlugen bei SQL Server 2014 zwei Versuchsdurchführungen fehl.

Die Untersuchung der Skalierung der Datensätze bezüglich der Freigabe von Speicherplatz, wenn das Löschen auf einer wachsenden Anzahl von Datensätzen durchgeführt wird, basiert auf der Abbildung 4.18. Diese Abbildung stellt anhand von Graphen den Verlauf der Freigabe dar, wenn bei steigenden Datenmengen das Löschen erfolgt.

Für alle Datenbanksysteme außer TimesTen wird, aufgrund der Ähnlichkeit der Interpretation der Messergebnisse, auf die jeweiligen Aussagen zu den Systemen der Auswertung von DELETE SIMPLE in Unterkapitel 4.3.3.1 verwiesen.

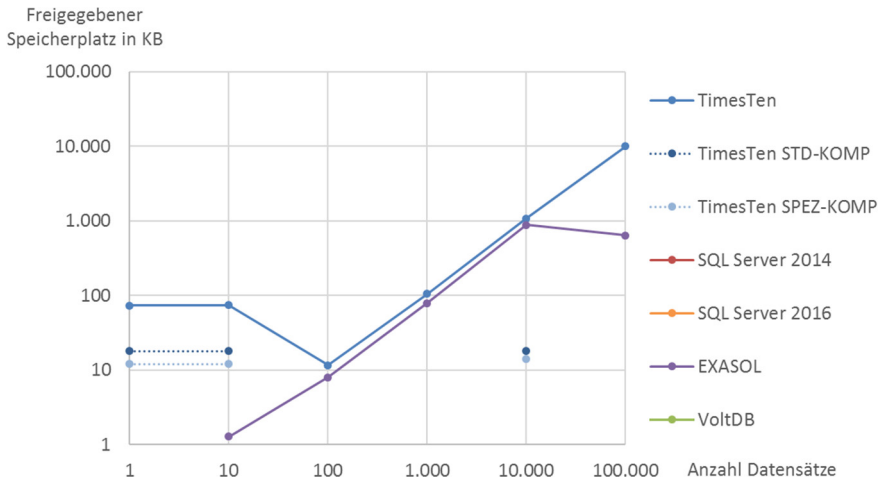

Abbildung 4.18 Freigegebener Speicherplatz in KB bei DELETE COMPLEX

Für TimesTen lassen sich, im Vergleich zu der Auswertung dieses Datenbanksystems in 4.3.3.1, Besonderheiten feststellen. Zum einen wird bis zu einer Menge von 10 Datensätzen signifikant mehr Speicher freigegeben im Löschvorgang als bei 100 Datensätzen. Zum anderen kann anhand der eingetragenen Graphen für Daten, die mittels der Datenkompression gespeichert wurden, festgestellt werden, dass unter Umständen auch bei diesen eine sofortige Freigabe von Speicher erfolgen kann. Ab einer Menge von 10 Datensätzen kann eine lineare Skalierung bei steigenden Mengen festgestellt werden.

4.3.4 Auswertung der Versuche aus der Kategorie UPDATE

In diesem Unterkapitel werden die Resultate bezüglich des Änderns von Datensätzen in diesem Unterkapitel näher betrachtet. Dazu erfolgen Auswertungen bezüglich der Skalierbarkeit der dabei aufgewendeten Zeit und der Veränderung des freigegebenen Speicherplatzes bei den einzelnen Datenbanksystemen.

4.3.4.1 Auswertung zu UPDATE SIMPLE

Auf der Basis der Abbildung 4.19 erfolgt die Auswertung bezüglich Zeitaufwandes. Dabei stellt die Abbildung die benötigte Zeit der Datenbanksysteme im Kontext der durchgeführten Änderungen bei steigenden Datenmengen in Form eines Liniendiagramms dar.

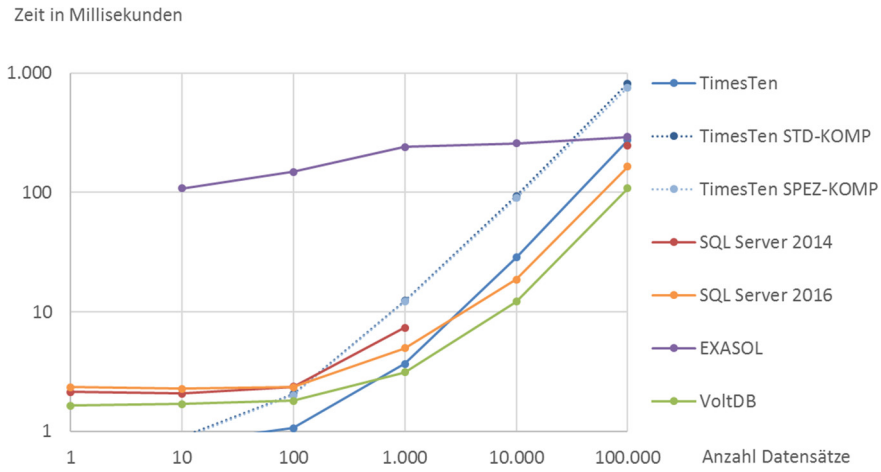

Abbildung 4.19 Benötigte Zeiten in ms bei UPDATE SIMPLE

Die Betrachtung von TimesTen zeigt, dass Änderungen bei kleinen Datenmengen in diesem Datenbanksystem am schnellsten durchgeführt werden. Steigt die Anzahl der Datenmengen, so wird TimesTen zu einem der langsamsten Datenbanksysteme. In der Gegenüberstellung der benötigten Zeit zwischen komprimierten und nicht komprimierten Daten ist erkennbar, dass auf den nicht komprimierten Daten Änderungen deutlich schneller durchgeführt werden. Für Veränderungen bei großen Datenmengen, die mittels einer der beiden Formen der Datenkompression gespeichert wurden, benötigt dieses Datenbanksystem die meiste Zeit. In der Abbildung 4.19 ist klar zu erkennen, dass TimesTen linear skaliert.

Der Vergleich der Datenbanksysteme von Microsoft untereinander zeigt, dass SQL Server 2016 Änderungen fast immer schneller durchführt als SQL Server 2014. Ab einer Menge von 10.000 Datensätzen gehören beide Systeme zu den schnellsten Datenbanksystemen in der Gegenüberstellung mit den Produkten der anderen Hersteller. SQL Server 2014 und SQL Server skalieren linear bezüglich der benötigten Zeit. Für SQL Server 2014 ist anzumerken, dass aufgrund von datenbankinternen Fehlern die Versuche bei einer Anzahl von 10.000 Datensätzen dreimal fehlgeschlagen sind und für diese Menge keine Messwerte vorhanden sind.

Das Datenbanksystem EXASOL ist in der Gegenüberstellung mit den anderen Systemen bei Aktualisierung das langsamste System. Die Ausnahme bildet die Betrachtung unter dem Aspekt, dass die Kompression von Daten bei der Datenspeicherung, sofern verfügbar, genutzt wird. Dann ist EXASOL bei einer Anzahl von mehr als 10.000 Datensätzen nicht mehr das langsamste Datenbanksystem.

Der Graph für EXASOL aus der Abbildung 4.19 lässt klar erkennen, dass EXASOL teilweise signifikant besser als linear bei der Verarbeitungszeit skaliert.

VoltDB ist ab einer Anzahl von 1.000 Datensätze das schnellste System, wenn Änderungen am Datenbestand durchgeführt werden sollen. Davor ist es das zweitschnellste Datenbanksystem. Für VoltDB kann eine lineare Skalierung bezüglich der benötigten Zeit bei steigenden Datenmengen festgestellt werden.

Unter Verwendung der nachfolgenden Abbildung 4.20 wird die Auswertungen der Ergebnisse bezüglich der Speichermessungen bei dem Versuch UPDATE SIMPLE durchgeführt. Dabei stellt die Abbildung die Veränderungen des Speicherplatzes in Kilobyte bei der Durchführung von Änderungen bei unterschiedlichen Datenmengen der Datenbanksysteme dar.

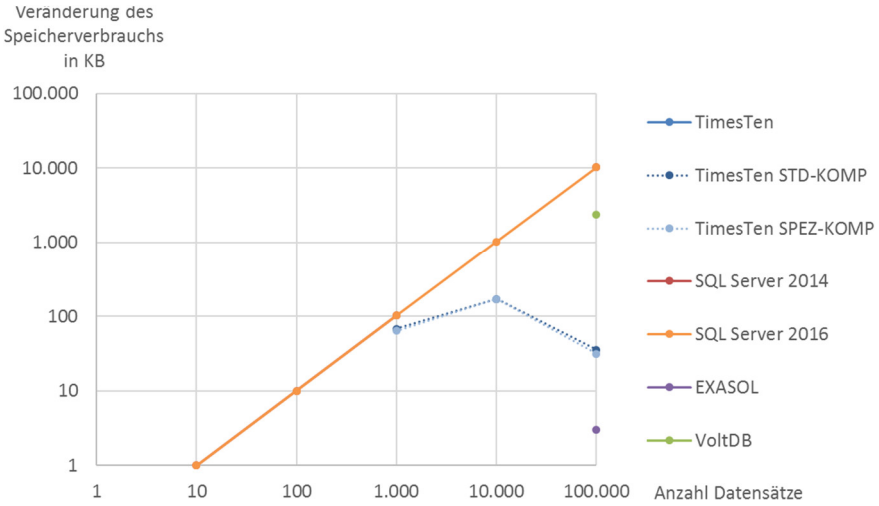

Abbildung 4.20 Veränderung des Speicherverbrauches in KB bei UPDATE SIMPLE

Bei der Betrachtung der Verläufe der Graphen für SQL Server 2014 und SQL Server 2016 lässt sich auch hier feststellen, dass diese, wie in 4.3.1.1 exakt gleich und linear sind. Dies hängt mit dem Verhalten der beiden Datensysteme zusammen, denn beide Systeme arbeiten bei Änderungen am Datenbestand nach dem Insert-Only-Ansatz, wie in Unterkapitel 3.2 beschrieben. Bei diesem Ansatz ist zu erkennen, dass Veränderungen an Datensätzen sich stets proportional auf Veränderungen am benötigten Speicherplatz auswirken. Daraus lässt sich schlussfolgern, dass die Skalierung bezüglich des Speicherverbrauches stets linear ist.

Für TimesTen lässt sich für Veränderungen an Datensätzen, die mittels Datenkompression gespeichert worden sind, erkennen, dass bei Änderungen von 100.000 Datensätzen weniger Speicher zusätzlich benötigt wird, um die Veränderungen durchzuführen als bei 10.000 oder 1.000 Datensätzen. Bei der Durchführung dieses Versuchs auf 1.000 und weniger Datensätzen, die mittels Kompression gespeichert wurden, sowie bei sämtlichen Mengen von nicht komprimierten Daten ließen sich keine Differenzen bei den Speichermessungen feststellen. Die Messergebnisse deuten nicht auf eine Skalierung bei der Speicherfreigabe hin.

Bei der Auswertung der Ergebnisse für EXASOL und VoltDB ist zu erkennen, dass für beide Datenbanksysteme nur je ein Messwert vorhanden ist und dieser jeweils keine Rückschlüsse auf die Skalierung der Systeme zulässt. Das bedeutet, dass in beiden Systemen nur bei 100.000 veränderten Datensätzen eine Zunahme des Speicherplatzes gemessen wurde.

4.3.4.2 Auswertung zu UPDATE COMPLEX

Unter Verwendung der Abbildung 4.21 wird der Zeitaufwand der einzelnen Datenbanksysteme für die Durchführung von UPDATE COMPLEX ausgewertet. Dabei zeigt die Abbildung die benötigte Zeit der Datenbanksysteme im Kontext der durchgeführten Änderungen bei steigenden Datenmengen in Form eines Liniendiagramms.

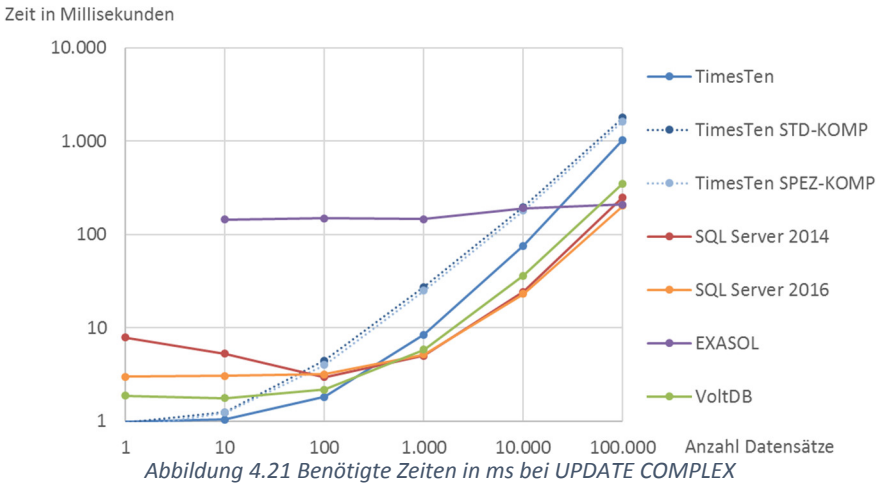

Abbildung 4.21 Benötigte Zeiten in ms bei UPDATE COMPLEX

Das Datenbanksystem EXASOL skaliert, wie sich erkennen lässt, signifikant besser als linear und ist bei Änderungen von 100.000 Datensätzen das zweitschnellste System.

Bei kleineren Mengen ist dieses Datenbanksystem das langsamste System, mit der Ausnahme, dass der Vergleich mit den anderen Systemen unter dem Aspekt der Kompressionsnutzung, sofern möglich, erfolgt. Dann ist EXASOL bei einer Anzahl von 10.000 Datensätzen schneller als TimesTen, wenn die standardmäßige Kompression bei der Datenspeicherung in TimesTen genutzt wird.

Der Vergleich der Datenbanksysteme von Microsoft untereinander zeigt, dass bei Mengen von weniger als 100 Datensätzen SQL Server 2016 schneller Änderungen durchführt als SQL Server 2014. Zu beobachten ist hier, dass SQL Server 2014 weniger Zeit für Änderungen bei 100 und 1000 Datensätzen benötigt als für kleinere Mengen. In der Gegenüberstellung mit anderen Datenbanksystemen ist zu erkennen, dass bei der Aktualisierung von großen Mengen an Datensätzen beide Datenbanksysteme zu den schnellsten Systemen gehören, während sie bei kleinen Datenmengen zu den langsamsten Hauptspeicherdatenbanksystemen gehören. Beide Systeme zeigen ein lineares Verhalten für den Zeitaufwand bei steigenden Datenmengen. Bei SQL Server 2014 trat während der Durchführung des Versuches mit 10.000 Datensätzen ein interner Fehler auf.

Die Auswertung bei TimesTen zeigt, dass Veränderungen an kleinen Datenmengen in diesem Datenbanksystem am schnellsten durchgeführt werden. Der Vergleich der benötigten Zeit für Änderungen zwischen komprimierten und nicht komprimierten Datenmengen bis zu einer Stückzahl von 1.000 Datensätzen lässt erkennen, dass bei den nicht komprimierten Daten Änderungen deutlich schneller durchgeführt werden. Die Differenzen der benötigten Zeit für die Manipulation von Datensätzen mit Kompression und denen ohne Kompression wird bei großen Datenmengen geringer. Der Versuch mit der höchsten Anzahl an zu ändernden Datensätzen wird von TimesTen am langsamsten ausgeführt im Vergleich zu den anderen Datenbanksystemen. Für Veränderungen an komprimierten Datensätzen lässt sich feststellen, dass bei steigenden Datenmengen für die benötigte Zeit eine lineare Skalierung erkennbar ist, während bei nicht komprimierten Datensätzen diese schlechter als linear ist.

Das Datenbanksystem VoltDB ist bei der Manipulation von kleinen Datenmengen das zweitschnellste System, während es bei Änderungen wachsenden Datenmengen langsamer wird. Bei diesem Hauptspeicherdatenbanksystem lässt sich eine annähernd lineare für die benötigte Zeit ab einer Menge von 1.000 zu ändernden Datensätzen erkennen.

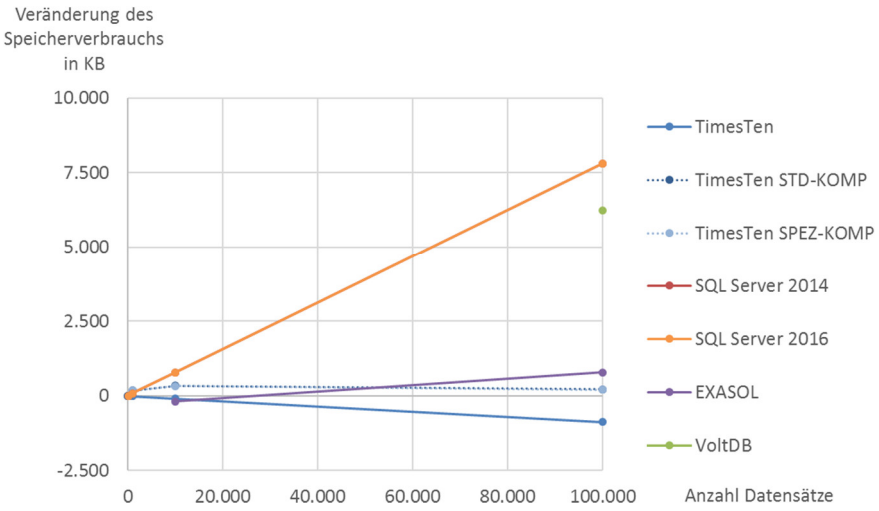

Abbildung 4.22 Veränderung des Speicherverbrauches in KB bei UPDATE COMPLEX

Mit Hilfe der vorangehenden Abbildung 4.22 wird die Auswertung der Ergebnisse bezüglich der Speichermessungen durchgeführt. Die Abbildung zeigt die Veränderungen des Speicherplatzes in Kilobyte bei der Durchführung von Änderungen bei unterschiedlichen Datenmengen der Datenbanksysteme. Bei dieser Abbildung ist zu beobachten, dass im Gegensatz zu den anderen Abbildungen der Auswertungen die Achsen keiner logarithmischen Skalierung unterliegen, da sonst negative Werte nicht in Kombination mit positiven Werten dargestellt werden können.

Die Auswertung von TimesTen zeigt, dass Veränderungen der Datensätze ohne Kompression zur Folge haben, dass sich der Speicherverbrauch sich verringert und bei Änderungen bei steigenden Datenmengen eine annähernd lineare Skalierung aufweist. Die Messergebnisse für die Speicherbeanspruchung der Datensätze, die mittels der standardmäßigen Kompression gespeichert wurden, weisen ab einer zu verändernden Menge von 1.000 Datensätze eine Zunahme auf, wobei diese bei 10.000 Datensätzen am größten ist. Bei der Betrachtung der Resultate der Speicherbeanspruchung von Datensätzen, die der Kompression anhand einer Spezifikation unterliegen und geändert wurden, weisen bereits ab einem Datensatz eine Zunahme im Speicherbrauch auf, wobei auch hier diese bei 10.000 geänderten Datensätzen am höchsten ist. Aufgrund der Zunahme und abschließender Abnahme des Speicherverbrauchs bei den Manipulationen von komprimierten Datensätzen, die anhand der standardmäßigen bzw. spezifikationsbasierten Kompression gespeichert wurden, lässt sich feststellen, dass keine Skalierung vorliegt.

Bei EXASOL lässt sich beobachten, dass bei Veränderungen von 10.000 Datensätzen eine Abnahme des Speicherbrauchs eben dieser Datensätze erfolgt und bei Änderungen von 100.000 Datensätzen eine Zunahme des Speicherverbrauchs erfolgt. Eine eindeutige Skalierung lässt sich dadurch nicht feststellen.

Für die Datenbanksysteme SQL Server 2014 und SQL Server 2016 wird auf die Auswertung zu UPDATE SIMPLE in 4.3.4.1 der beiden Systeme verwiesen, da die Interpretation der Ergebnisse die gleiche Aussage aufweist.

Die Auswertung für VoltDB beschränkt sich auf die Feststellung, dass in diesem System nur bei 100.000 veränderten Datensätzen eine Zunahme des Speicherplatzes gemessen wurde und keine Skalierung erkennbar ist.

4.3.5 Datenbankspezifische Besonderheiten der Auswertungen

Auf die Besonderheiten der Ergebnisse, die sich bei den einzelnen Datenbanksystemen über die unterschiedlichen Versuche erstrecken oder in einzelnen Versuchen herausragen und nicht an geeigneter Stelle behandelt wurden, sowie den datenbankspezifischen Auffälligkeiten, die nicht unbedingt der Architektur des jeweiligen Datenbanksystems zugeordnet werden können, wird in diesem Kapitel eingegangen. Dabei wird jedes Datenbanksystem separat behandelt.

4.3.5.1 Besonderheiten bei TimesTen

Bei der Betrachtung der Messergebnisse des Versuchs INSERT CUSTOMER mit der Verwendung eines Prepared Statements bzw. einer Stored Procedure fällt auf, dass für das Einfügen von 1.000.000 Datensätzen ohne Nutzung der Datenkompression die Standardabweichung für die benötigte Zeit der Messung-Iterationen signifikant höher ist als die errechnete Abweichung die der Warmup-Iterationen. Die gleiche Beobachtung kann bei der Verwendung eines Statements bei 100.000 Datensätzen, ebenfalls ohne Verwendung der Datenkompression, beobachtet werden. Die Betrachtung der gemessenen Zeiten der einzelnen Messung-Iterationen ergibt, dass beim Gebrauch eines Prepared Statements bzw. einer Stored Procedure jeweils ein Wert deutlich von den anderen abweicht. Die Untersuchung der Messung-Iterationen bei Statement zeigt, dass die Messwerte breiter gestreut sind.

Der Vergleich der Werte der Differenzen des den Daten direkt zuordenbaren Speicherverbrauches vor und nach jeder Iteration mit den Werten der Differenzen von ttSize für den geschätzten Speicherverbrauch vor und nach jeder Iteration zeigt insbesondere für große Datenmengen erhebliche Abweichungen bei den

Möglichkeiten der Bestimmung des Speicherverbrauchs in den Versuchen auf, die das Aktualisieren bzw. Löschen von Datensätzen thematisieren.

In der Auswertung des Versuchs UPDATE COMPLEX wurde festgestellt, dass bei großen, nicht komprimierten Datenmengen die benötigte Zeit schlechter als linear bei weiterem Wachstum der Menge skaliert. Die Betrachtung der Standardabweichung für die Zeitmessung bei sowohl den Messung-Iterationen als auch den Warmup-Iterationen für 100.000 zu ändernde Datensätze weist einen relativ hohen Wert auf. Bei der Untersuchung der einzelnen Messwerte für die benötigte Zeit lässt sich feststellen, dass die Werte deutlich voneinander abweichen.

Eine weitere Besonderheit stellen die gemessenen Werte für den von den Daten direkt zuordenbaren Speicherverbrauch und den Werte von Memory Perm In Use dar. Bei den Versuchen, die das Löschen von Datensätzen behandeln, zeigen die Werte teils widersprüchliche Angaben an, obwohl beide Messkriterien die in den Tabellen enthaltenen Daten beinhalten. So ist beispielweise beim Versuch DELETE COMPLEX mit 100 Datensätzen zu erkennen, dass die Differenzen der Werte für die Daten direkt zuordenbaren Speicherverbrauch vor und nach jeder Iteration eine Verkleinerung des Speicherverbrauchs zeigen, jedoch bei den Differenzen der Werte von Memory Perm In Use vor und nach jeder Iteration keine Verkleinerung des beanspruchten Hauptspeichers festgestellt werden kann.

4.3.5.2 Besonderheiten bei SQL Server 2014

Die Untersuchung der Messergebnisse des Versuchs INSERT CUSTOMER ONE BY ONE zeigen, dass bei allen drei genutzten Arten des Einfügens bei einer Anzahl von 100.000 einzufügenden Datensätzen die Standardabweichung der Zeitmessung relativ hoch ist. Die Betrachtung der benötigten Zeit in Messung- und Warmup-Iterationen bei dem Gebrauch einer Stored Procedure bzw. eines Prepared Statements stellt klar, dass jeweils ein einzelner Wert die hohe Abweichung erklärt. Bei der Untersuchung der Verwendung eines Statements für das Einfügen von Datensätzen wird erkennbar, dass die Werte der Zeitmessung weiter gestreut sind und es sich in diesem Fall nicht um einen Einzelwert handelt, der den Wert der Standardabweichung stark beeinflusst.

Die Auswertung von SQL Server 2014 in dem Versuch DELETE SIMPLE zeigt einen hohen Wert für die Standardabweichung bei der benötigten Zeit in den Messung-Iterationen für 100.000 Datensätzen. Die Analyse der Messwerte der Iterationen stellt klar, dass ein einzelner Messwert die hohe Standardabweichung erklärt.

4.3.5.3 Besonderheiten bei SQL Server 2016

Wie auch schon bei SQL Server 2014 ist bei den Betrachtungen der Messergebnisse für den Versuch INSERT CUSTOMER ONE BY ONE erkennbar, dass die Standardabweichung von den gemessenen Zeiten der Warmup- und/oder Messung-Iterationen bei einer Anzahl von 100.000 einzufügenden Datensätzen relativ hoch ist bei den drei Arten des Einfügens von Daten. Die Untersuchung der benötigten Zeit bei der Verwendung einer Stored Procedure bzw. eines Statements für das Hinzufügen von Daten zeigt, dass die einzelnen Werte sowohl bei den Warmup- als auch bei den Messung-Iterationen etwas auseinanderliegen. Bei dem Gebrauch eines Prepared Statements bedingt nur ein einziger Wert die hohe Standardabweichung der Warmup-Iterationen.

Die Analyse der Ergebnisse für SELECT 6 zeigt, dass die Standardabweichung für die benötigte Zeit bei Warmup- und Messung-Iterationen jeweils einen hohen Wert hat. Die Betrachtung der Einzelwerte der Zeitmessungen stellt klar, dass die Werte breiter gestreut sind in ihren Ausprägungen.

4.3.5.4 Besonderheiten bei EXASOL

Die Messwerte der benötigten Zeit in sämtlichen Versuche, die sich mit dem Einfügen, Verändern und Löschen von Datensätzen beschäftigen, zeigen, dass die gleichen durchgeführten Operationen in den jeweiligen Versuchen teils signifikant unterschiedlich lange für die Verarbeitung brauchen.

In den Versuchen SELECT 1 – 6 sind die Standardabweichungen oftmals höher als die Mittelwerte für die Zeiten der Durchführung. Hier wird erfolgt der Verweis auf Unterkapitel 4.3.2, das diesen Sachverhalt im Kontext mit der Erstberechnung verständlich macht.

4.3.5.5 Besonderheiten bei VoltDB

Zusätzlich zu der Besonderheit, die bei den Messungen des den Daten direkt zuordenbaren Speicherverbrauchs in Unterkapitel 4.3.1.1 wurde festgestellt, lässt sich das Gleiche auch bei den Messungen bezüglich des Speicherverbrauchs der Indize beobachten, so dass auch hier keine weitergehende Analyse durchgeführt werden kann.

Die Resultate der einzelnen Iterationen der Versuche SELECT 3 – 6 stellen klar, dass zu einem bestimmten Zeitpunkt die Werte für den Speicherverbrauch feststehen, da die Messwerte keine Veränderungen der Speicherbelegung zeigen.

Die Untersuchungen der einzelnen Messungen für 100 bzw. 1.000 Datensätze nach der jeweils ersten Warmup-Iteration der Versuche INSERT CUSTOMER und INSERT CUSTOMER ONE BY ONE zeigen, dass die Messwerte der benötigten Zeit der Warmup-Iterationen oftmals deutlich höher sind, als die der Messung-Iterationen. Die Verarbeitungsgeschwindigkeit wird bei dem Einfügen dieser Datenmengen somit tendenziell schneller.

4.4 Fazit

Die Auswertungen der Versuche zeigen, dass zwischen den einzelnen untersuchten Hauptspeicherdatenbanksystemen teils sehr deutliche Unterschiede bei den eingesetzten Datenbankoperationen vorhanden sind. Bei der Betrachtung der Ergebnisse für Versuchsdurchläufe, die kleine Datenmengen nutzen, lässt sich feststellen, dass die beiden Datenbanksysteme von Microsoft und VoltDB mit ihrer OLTP-Ausrichtung bzw. TimesTen mit der Hybrid-Ausrichtung deutlich schneller in der Ausführung der jeweiligen Operationen sind als das reine OLAP-Datenbanksystem EXASOL. Die Resultate zeigen aber auch, dass EXASOL bei größer werdenden Mengen deutlich besser bei Abfrage-, Lösch- und Änderungsoperationen skaliert bzw. sich verhält als die anderen Datenbanksysteme.

Aus der Betrachtung der Ergebnisse für das Einfügen von Datensätzen lässt sich erkennen, dass die Art, wie Datensätze eingefügt werden, einen signifikanten Einfluss auf die benötigte Zeit hat. Die Nutzung von Prepared Statements im Batch-Modus ist bei allen Datenbanksystemen außer VoltDB die beste Wahl. Für VoltDB stellt der Client die beste Art des Einfügens von Datensätzen dar.

Die Auswirkungen der Datenkompression auf die Verarbeitungsgeschwindigkeit lassen sich anhand des Datenbanksystems TimesTen analysieren. Die Messergebnisse lassen erkennen, dass die Löschung von Datensätzen, die komprimiert worden sind, insbesondere bei großen Mengen deutlich schneller durchgeführt wird als bei Datensätzen, die keiner Kompression unterliegen. Das Hinzufügen von Datensätzen zur der Datenbank und das Aktualisieren von Datensätzen wird teils deutlich langsamer ausgeführt auf komprimierten Datensätzen als bei nicht komprimierten Datensätzen. Bei der Betrachtung der Resultate für Abfragen auf den Datensätzen ist festzustellen, dass die Abfragen auf den komprimierten Datensätzen fast immer schneller durchgeführt werden, die Unterschiede aber gering sind. Der Vergleich der Kompressionsmöglichkeiten untereinander zeigt, dass die Kompression anhand einer Spezifikation generell die bessere Wahl darstellt. Zum einen ist der Speicherverbrauch der so komprimierten

Daten geringer als bei der standardmäßigen Kompression. Zum anderen sind die gemessenen Zeiten in fast allen Versuchen besser. Abschließend zur Verwendung von Datenkompression lässt sich festhalten, dass, wenn ein Hauptspeicherdatenbanksystem eine analytische Ausrichtung hat, sich die Datenkompression lohnt.

In den Gegenüberstellungen mit den Messergebnissen der anderen Datenbanksysteme lässt sich für TimesTen feststellen, dass bei der Nutzung der zuvor empfohlenen Art des Einfügens, das heißt mit der Verwendung von Prepared Statements, dieses System mit den anderen Hauptspeicherdatenbanksystemen mithalten kann. Bezüglich der Leistung bei Abfragen lässt sich das Datenbanksystem von Oracle zwischen dem reinen OLAP-System und den OLTP-Systemen eingliedern. Die Löschvorgänge erfolgen insbesondere bei kleinen Mengen an Daten am schnellsten und bei steigenden Datenmengen stellt TimesTen das zweitschnellste Datenbanksystem dar. Die Aktualisierung der Datensätze erfolgt in diesem System bei kleinen Mengen schneller als bei den anderen, jedoch entwickelt sich TimesTen bei steigender Anzahl von Datensätzen zu der langsamsten Hauptspeicherdatenbank. Letztlich lässt sich sagen, dass TimesTen als Hybrid-Datenbanksystem eine sinnvolle Option ist, wenn ein Hauptspeicherdatenbanksystem als OLTP- und OLAP-System genutzt werden soll und die oben aufgeführten Punkte bezüglich der Art des Einfügens und des Verhaltens bei Aktualisierungsvorgängen beachtet werden.

Im Vergleich von SQL Server 2014 und SQL Server 2016 untereinander lässt sich feststellen, dass das Einfügen von Datensätzen, außer bei der Nutzung einer Stored Procedure, in SQL Server 2014 schneller durchgeführt wird als bei SQL Server 2016. Auffallend ist bei der Betrachtung des den Daten direkt zuordenbaren Speicherverbrauchs der beiden Systeme, dass diese stets die gleichen Werte aufweisen. In der Gegenüberstellung zu den anderen Datenbanksystemen zeigt sich bei Aktualisierungen von Daten, dass diese für kleine Datenmengen von SQL Server 2014 und SQL Server 2016 vergleichsweise langsam durchgeführt werden. Bei steigenden Mengen von Daten ist zu erkennen, dass diese eine annähernd vergleichbare, wenn nicht sogar bessere Leistung als die anderen Hauptspeicher-datenbanksysteme für das Aktualisieren von Datensätzen aufweisen. Im Allgemeinen betrachtet stellt SQL Server 2016 das schnellere der beiden Systeme diesbezüglich dar. Die Durchführung von Löschvorgängen geschieht in beiden Systemen von Microsoft mit am langsamsten, wobei eine Tendenz zu erkennen ist, dass insbesondere bei größeren werdenden Datenmengen die zeitliche Skalierung schlechter als linear ist. Das Einfügen von Daten anhand der empfohlenen Art zeigt, dass die Leistung der beiden Datenbanksysteme vergleichbar ist mit denen der

anderen Systeme. Die Verarbeitung von Abfragen zählt nicht zu den Stärken der beiden Systeme, da diese in den Auswertungen der Versuche, die die Leistung in der analytischen Verarbeitung thematisieren, oftmals zu den langsamsten Systemen zählen. Hierbei ist anzumerken, dass die Fehler, die bei der Aktualisierung oder Löschung von Datensätzen in einigen Versuchen bei SQL Server 2014 auftraten, in SQL Server 2016 nicht mehr existent sind.

Die Resultate für EXASOL zeigen, dass dieses Datenbanksystem für die Verarbeitung großer Datenmengen gut geeignet ist. Insbesondere die Untersuchungen der zeitlichen Skalierung, sofern diese bei den jeweiligen Versuchen relevant ist, stellen klar, dass dieses Datenbanksystem für größere werdende Mengen an Daten oftmals besser als linear bezüglich der Datenverarbeitung skaliert. Die grafischen Auswertungen zeigen, dass bei weiter steigendenden Datenmengen die Tendenz besteht, das EXASOL schneller in der Datenverarbeitung sein wird als die anderen Datenbanksysteme, wobei bei dem Einfügen von großen Datenmengen EXASOL bereits das schnellste System ist. Die Betrachtung der Ergebnisse von Versuchen mit kleinen Mengen von Datensätzen lässt erkennen, dass EXASOL stets das langsamste der untersuchten Hauptspeicherdatenbanksysteme darstellt. Die analytische Ausrichtung dieses Datenbanksystems zeigt sich vor allem bei den Versuchen, die die Auswertung von Datenmengen behandeln. Bei diesen ist EXASOL, außer bei einem Versuch, stets das schnellste Datenbanksystem in den Gegenüberstellungen mit den anderen Systemen.

Das System VoltDB zeigt in dem Vergleich mit den anderen Datenbanksystemen bei der Durchführung von Aktualisierungen und Löschungen von Datensätzen gute Leistungen und zählt diesbezüglich zu den schnellsten Systemen. Für das Einfügen von Datensätzen, ist wie eingangs zu diesem Unterkapitel erwähnt, der Client von VoltDB zu empfehlen. Die Auswertungen zeigen, dass die die Nutzung von Stored Procedures, Prepared Statements oder Statements hinsichtlich des Hinzufügens von Datensätze bei VoltDB im Vergleich zu dem Client teils signifikant höhere Werte bezüglich der benötigten Zeit aufweisen. Die Leistung von VoltDB bei den Versuchen, die das Verhalten bei Analysen thematisieren, liegt auf dem Niveau von SQL Server 2014 und SQL Server 2016, die zu den langsamen Datenbanksystemen zählen. Insbesondere die Abfrage von großen Tabellen und/oder vielen Zwischenergebnissen bringen das System an seine Grenzen, sodass für VoltDB abschließend geäußert werden kann, dass Analysen, wenn überhaupt, auf kleinen Datenmengen durchgeführt werden sollten.

Die Betrachtung der Hauptspeicherdatenbanksysteme bezüglich des den Daten direkt zuordenbaren Speicherverbrauchs zeigt, dass diese annähernd linear oder besser bezüglich des benötigten Speichers skalieren, wenn die Datenmenge ansteigt. Die Gegenüberstellungen der Datenbanksysteme bezüglich der Hauptspeicherbelegung zeigen, dass, sofern die Messwerte nutzbar sind für Aussagen bezüglich der Skalierung oder Vergleiche mit anderen Systemen, Architektur und Kompressionsmöglichkeiten einen wesentlichen Einfluss haben. So hat VoltDB aufgrund der auf Partitionen basierenden Systemarchitektur, bei der die Daten redundant, sofern die Leistung positiv beeinflusst wird, gespeichert werden, einen höheren Speicherverbrauch als die anderen Systeme. Bei kleinen Mengen zeigen die beiden Datenbanksysteme von Microsoft einen geringeren Speicherverbrauch als die anderen Systeme, während die Datenbanksysteme mit Datenkompression einen geringeren Speicherverbrauch aufweisen bei großen Datenmengen, sofern die Kompression genutzt bzw. automatisch vom System durchgeführt wird.

Den letzten Teil des Fazits bildet eine Abhandlung über die OLTP- und OLAP-Trennung im Zusammenhang mit den untersuchten Hauptspeicherdatenbanksystemen. Die Betrachtung der Auswertungen zeigt, dass bei kleinen Datenmengen, wie für den OLTP-Betrieb typisch, die Datenbanksysteme mit der OLTP-Ausrichtung signifikant schneller in der Datenverarbeitung sind als das EXASOL mit der OLAP-Ausrichtung. EXASOL zeigt seine Stärken, wie schon in der Abhandlung zu EXASOL in diesem Unterkapitel dargelegt, bei der Verarbeitung von großen Datenmengen, wie es für OLAP-Systeme charakteristisch ist. TimesTen mit seiner hybriden Ausrichtung zeigt, dass Datenbanksysteme mit OLTP- und OLAP-Ausrichtung eine Alternative zu reinen OLTP- bzw. OLAP-Systemen sein können. Jedoch sind auch Defizite gegenüber Datenbanksystemen vorhanden, die nur auf den OLTP- bzw. OLAP-Betrieb ausgerichtet sind. Abschließend lässt sich festhalten, dass eine Trennung von OLAP- und OLTP-Nutzung auch bei Hauptspeicherdatenbanksysteme sinnvoll ist, selbst wenn die Verarbeitungsgeschwindigkeit aufgrund des Hauptspeichers als primären Speicherort der Daten deutlich besser als bei den festplattenbasierten Datenbanksystemen ist.

5 Zusammenfassung und Ausblick

Datenbanksysteme, die den Arbeitsspeicher als primären Ort der Datenhaltung nutzen, bilden viel Potenzial für den praktischen Einsatz. Im Rahmen dieser Arbeit werden die Grundlagen von Hauptspeicherdatenbanksysteme und ein Spektrum technischer Aspekte im Zusammenhang mit diesen Systemen erklärt. Eine Auswahl von fünf Datenbanksysteme erfolgt und die ausgewählten Systeme werden einer näheren Betrachtung unterzogen. Die Leistungsanalyse der Hauptspeicher-datenbanksysteme bildet den Kern dieser Arbeit. Diese ist in die vier Bereiche Aufbau, Durchführung und Auswertung der Versuche sowie dem Fazit eingeteilt. Der Aufbau beschäftigt sich mit den hardware- und softwaretechnischen Gegebenheiten der Versuche. Die Durchführung basiert auf einer Zusammenstellung und Ausführung von Möglichkeiten, die sich anhand der grundlegenden Datenbankoperationen ergeben und entweder in sämtlichen verwendeten Datenbanksystemen vorhanden oder äquivalent umsetzbar sind. Für jeden Versuch erfolgen immer Zeitmessungen, wobei in den Versuchen, in denen Datensätze eingefügt, aktualisiert oder gelöscht werden, auch immer Speichermessungen erfolgen. Die Auswertung der Versuche betrachtet die Resultate der Messungen sowohl einzeln für die Datenbanksysteme als auch in der Gegenüberstellung mit den anderen Datenbanksystemen. Ebenso werden im Analyseteil gravierende Auffälligkeiten für jedes Datenbanksystem aufgeführt. Im Fazit werden die Stärken und Schwächen der betrachteten Hauptspeicherdatenbanksysteme im praktischen Einsatz zusammenfassend dargestellt und die möglichen Einsatzgebiete lassen sich für jedes dieser Systeme erkennen.

Das Interesse von Wirt- und Wissenschaft an der schnellen und effizienten Verarbeitung der stetig ansteigenden Menge an Daten ist groß. Hauptspeicherdatenbanksysteme bieten eine lohnenswerte Möglichkeit, die vorhandenen und entstehenden Datenmengen sinnvoll zu organisieren und zu nutzen. In der Wirt- und Wissenschaft ist die wachsende Bedeutung der Hauptspeicherdatenbanksysteme zu erkennen. Die Zahl der wissenschaftlichen Veröffentlichungen, die sich diesen Datenbanksystemen widmen, ist seit dem Beginn des 21. Jahrhunderts deutlich gestiegen. In der Wirtschaft sehen nicht nur junge Unternehmen, wie die EXASOL AG und VoltDB Inc., eine Zukunft in der Nutzung von Hauptspeicherdatenbanksystemen, sondern auch große Softwareunternehmen wie SAP, Microsoft und Oracle, da diese ihr Produktportfolio um diese Art von Datenbanksystemen erweitert haben.

Für weitere Forschungen interessant ist die Ablösung festplattenbasierter OLTP- und OLAP Datenbanksysteme durch hybride Hauptspeicherdatenbanksysteme in bestehenden IT-Landschaften. Beispielweise kann eine Studie zeigen, wie hoch die dabei entstehenden Kosten sind und welche Vorteile, aber auch Nachteile ein Wechsel mit sich bringt.

Bei der Betrachtung der Sicherheit von Datenbeständen im Hinblick eines Stromausfalls können sich Untersuchungen weitergehend damit befassen, wie sich die Verwendung von Non-Volatile Random-Access Memory, das heißt nichtflüchtiger RAM, im Praxiseinsatz bei Hauptspeicherdatenbanksystemen bewähren kann. Hier ist denkbar, dass die Datensicherung auf Festplatten komplett entfällt und eine Abwägung unter organisatorischen und finanziellen Aspekten die Möglichkeiten aufzeigt.

Anhang

A SQL-Anweisungen

Die nachfolgenden SQL-Befehle basieren auf Oracle‘s PL/SQL, das in TimesTen verwendet wird. Für die anderen Hauptspeicherdatenbanksysteme wurden die Befehle entsprechend angepasst. In den folgenden Abschnitten werden nur die Anweisungen aus TimesTen behandelt, da sich diese aufgrund dem SQL-Standard ähnlich sind und daher nicht für jedes Datenbanksystem einzeln erwähnt werden.

SELECT 1

SELECT generation, COUNT(*) AS amount FROM (SELECT (CASE WHEN age < 18 THEN 'under 18' WHEN age BETWEEN 18 AND 25 THEN '18-25' WHEN age BETWEEN 26 AND 35 THEN '26-35' WHEN age BETWEEN 36 AND 45 THEN '36-45' WHEN age BETWEEN 46 AND 55 THEN '46-55' WHEN age BETWEEN 56 AND 65 THEN '56-65' END) AS generation FROM (SELECT *, FLOOR(MONTHS_BETWEEN(DATE '2016-12-31', birthday) / 12) AS age FROM customer) tmp) GROUP BY generation ORDER BY generation ASC;

SELECT 2

SELECT customer.first_name, customer.last_name, customer_order.order_id, coipResult.category, coipResult.product_name, coipResult.quantity, coipResult.price, coipResult.maxProduct FROM (SELECT coipTmp.order_id, coipTmp.product_name, coipTmp.category, coipTmp.price, coipTmp.quantity, coipMax.maxProduct FROM (SELECT coi.order_id, p.product_name, p.category, p.price, coi.quantity,coi.quantity*p.price tmpProduct FROM customer_order_item coi, product p WHERE coi.product_id = p.product_id) coipTmp , (SELECT coi.order_id, MAX(coi.quantity*p.price) maxProduct FROM customer_order_item coi, product p WHERE coi.product_id = p.product_id GROUP BY coi.order_id) coipMax WHERE

coipTmp.tmpProduct = coipMax.maxProduct AND coipTmp.order_id = coipMax.order_id) coipResult, customer, customer_order WHERE coipResult.order_id = customer_order.order_id AND customer_order.customer_id = customer.customer_id ORDER BY coipResult.maxproduct DESC;

SELECT 3

SELECT customer.customer_id, SUM(coip.orderValue) sales FROM (SELECT coi.order_id, SUM((coi.quantity*p.price)) orderValue FROM customer_order_item coi, product p WHERE coi.product_id = p.product_id GROUP BY coi.order_id) coip, customer_order , customer WHERE coip.order_id = customer_order.order_id AND customer_order.customer_id = customer.customer_id GROUP BY customer.customer_id

SELECT 4

SELECT supplier.supplier_id, supplier.supplier_name, topsales.product_id, topsales.sales_value FROM (SELECT * FROM (SELECT coisq.product_id, (coisq.sum_quantity*p.price) sales_value FROM (SELECT customer_order_item.product_id, SUM(customer_order_item.quantity) sum_quantity FROM customer_order_item GROUP BY customer_order_item.product_id) coisq, product p WHERE coisq.product_id = p.product_id ORDER BY SALES_VALUE desc) sales WHERE ROWNUM <= 20) topsales, supplier, purchase WHERE topsales.product_id = purchase.product_ID AND purchase.supplier_id = supplier.supplier_id ORDER BY supplier.supplier_id ASC, SALES_VALUE DESC;

SELECT 5

SELECT customer.first_name, product.category, SUM(res.posvalue) sales FROM (SELECT coc.customer_id, p.product_id, (p.price * coi.quantity) posValue FROM (SELECT order_id, co.customer_id FROM customer_order co, (SELECT customer_id FROM customer WHERE birthday >= '1965-01-01') c WHERE co.customer_id = c.customer_id AND co.order_date >= '2012-01-01') coc, product p, customer_order_item coi WHERE coc.order_id = coi.order_id AND p.product_id = coi.product_id) res, customer, product WHERE customer.customer_id = res.customer_id AND product.product_id = res.product_id GROUP BY customer.first_name, product.category ORDER BY customer.first_name ASC, sales DESC;

SELECT 6

SELECT * FROM (SELECT su.employee_id, su.product_id, RANK() OVER (PARTITION BY su.employee_id ORDER BY sales_quantity DESC) top FROM (SELECT e.employee_id, coi.product_id, SUM(coi.quantity) sales_quantity FROM employee e, customer_order co, customer_order_item coi WHERE e.employee_id = co.employee_id AND co.order_id = coi.order_id GROUP BY e.employee_id, coi.product_id) su) WHERE TOP <= 3 ORDER BY employee_id ASC, top ASC;

UPDATE SIMPLE

UPDATE customer SET birthday = birthday - 365, registration = registration + 30

UPDATE COMPLEX

UPDATE customer SET last_name = SUBSTR(last_name, 1, 3), first_name = SUBSTR(first_name, 1, 3), street = SUBSTR(street, 1, 3), house_number = house_number - 1 , city = SUBSTR(city, 1,3), country = SUBSTR(country, 1, 5), birthday = birthday - 365, registration = registration - 1, gender = '?';

DELETE COMPLEX

DELETE FROM customer WHERE last_name BETWEEN 'A' AND 'Mz' OR first_name BETWEEN 'B' AND 'Nz' OR (street BETWEEN 'M' AND 'Zz' AND house_number BETWEEN 100 AND 400) OR city LIKE 'K%' OR country BETWEEN 'A' AND 'Ezz' OR birthday BETWEEN '1970-01-01' AND '1990-12-31' OR registration BETWEEN '2012-01-01' AND '2012-12-31';

DELETE SIMPLE

DELETE FROM customer WHERE last_name BETWEEN 'A' AND 'Izzz' AND first_name BETWEEN 'T' AND 'Zzzz';

INSERT CUSTOMER – STORED PROCEDURE

CREATE OR REPLACE PROCEDURE proc_insert_customer(cstmrid IN NUMBER, lstnm IN VARCHAR2, frstnm IN VARCHAR2, strt IN VARCHAR2, hsnmbr IN NUMBER, cty IN VARCHAR2, cntry IN VARCHAR2, brthdy IN DATE, rgstrtn IN DATE, gndr IN VARCHAR2) IS BEGIN INSERT INTO customer(customer_id, last_name, first_name, street, house_number, city, country, birthday, registration, gender) VALUES (cstmrid, lstnm, frstnm, strt, hsnmbr, cty, cntry, brthdy, rgstrtn, gndr); END;

INSERT CUSTOMER_ORDER – STORED PROCEDURE

CREATE OR REPLACE PROCEDURE proc_insert_customer_order(orderid IN NUMBER, customerid IN NUMBER, orderdate IN DATE, processeddate IN DATE,

announceddeliverydate IN DATE, deliverydate IN DATE, paymentdate IN DATE, employeeid IN NUMBER) IS BEGIN INSERT INTO customer_order(order_id, customer_id, order_date, processed_date, announced_delivery_date, delivery_date, payment_date, employee_id) VALUES (orderid, customerid, orderdate, processeddate, announceddeliverydate, deliverydate, paymentdate, employeeid);END;

INSERT CUSTOMER_ORDER_ITEM – STORED PROCEDURE

CREATE OR REPLACE PROCEDURE proc_insert_customer_order_itm(orderid IN NUMBER, productid IN NUMBER, quant IN NUMBER) IS BEGIN INSERT INTO customer_order_item(order_id, product_id, quantity) VALUES (orderid, productid, quant);END;

INSERT EMPLOYEE – STORED PROCEDURE

CREATE OR REPLACE PROCEDURE proc_insert_employee(employee_id IN NUMBER, departmentnr IN NUMBER, lastname IN VARCHAR2, firstname IN VARCHAR2) IS BEGIN INSERT INTO employee VALUES (employee_id, departmentnr, lastname, firstname); END;

INSERT PRODUCT – STORED PROCEDURE

CREATE OR REPLACE PROCEDURE proc_insert_product(productid IN NUMBER, productname IN VARCHAR2, price IN NUMBER, ctgry IN VARCHAR2) IS BEGIN INSERT INTO product VALUES (productid, productname, price, ctgry); END;

INSERT PURCHASE – STORED PROCEDURE

CREATE OR REPLACE PROCEDURE proc_insert_purchase(productId IN NUMBER, supplierId IN NUMBER)IS BEGIN INSERT INTO purchase VALUES (productId, supplierId);END;

INSERT SUPPLIER – STORED PROCEDURE

CREATE OR REPLACE PROCEDURE proc_insert_supplier(supplierid IN NUMBER, suppliername IN VARCHAR2, legalform IN VARCHAR2, strt IN VARCHAR2, hsnmbr IN NUMBER, cty IN VARCHAR2, cntry IN VARCHAR2) IS BEGIN INSERT INTO supplier (supplier_id, supplier_name, legal_form, street, house_number, city, country) VALUES (supplierid, suppliername, legalform, strt, hsnmbr, cty, cntry); END;

INSERT CUSTOMER – PREPARED STATEMENT

INSERT INTO customer (customer_id, last_name, first_name, street, house_number, city, country, birthday, registration, gender) VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?);

INSERT CUSTOMER_ORDER – PREPARED STATEMENT
INSERT INTO customer_order (order_id, customer_id, order_date, processed_date, announced_delivery_date, delivery_date, payment_date, employee_id) VALUES(?, ?, ?, ?, ?, ?, ?, ?);

INSERT CUSTOMER_ORDER_ITEM – PREPARED STATEMENT
INSERT INTO customer_order_item (order_id, product_id, quantity) VALUES(?, ?, ?);

INSERT EMPLOYEE – PREPARED STATEMENT
INSERT INTO employee (employee_id, department_nr, last_name, first_name) VALUES(?, ?, ?, ?);

INSERT PRODUCT – PREPARED STATEMENT
INSERT INTO product (product_id, product_name, price, category) VALUES(?, ?, ?, ?);

INSERT PURCHASE – PREPARED STATEMENT
INSERT INTO purchase (product_id, supplier_id) VALUES(?, ?);

INSERT SUPPLIER – PREPARED STATEMENT
INSERT INTO supplier (supplier_id, supplier_name, legal_form, street, house_number, city, country) VALUES(?, ?, ?, ?, ?, ?, ?);

CREATE TABLE CUSTOMER
CREATE TABLE customer (customer_id NUMBER(10) NOT NULL PRIMARY KEY, last_name VARCHAR2(50) NOT NULL, first_name VARCHAR2(50) NOT NULL, street VARCHAR2(50) NOT NULL, house_number NUMBER(3) NOT NULL, city VARCHAR2(30) NOT NULL, country VARCHAR2(25) NOT NULL, birthday DATE NOT NULL, registration DATE NOT NULL, gender VARCHAR2(1) NOT NULL);

CREATE TABLE CUSTOMER COMPRESSED
CREATE TABLE customer (customer_id NUMBER(10) NOT NULL PRIMARY KEY, last_name VARCHAR2(50) NOT NULL, first_name VARCHAR2(50) NOT NULL, street VARCHAR2(50) NOT NULL, house_number NUMBER(3) NOT NULL, city VARCHAR2(30) NOT NULL, country VARCHAR2(25) NOT NULL, birthday DATE NOT NULL, registration DATE NOT NULL, gender VARCHAR2(1) NOT NULL) COMPRESS (last_name BY DICTIONARY, first_name BY DICTIONARY, street BY DICTIONARY, city BY DICTIONARY, country BY DICTIONARY, birthday BY DICTIONARY, registration BY DICTIONARY, gender BY DICTIONARY) OPTIMIZED FOR READ;

CREATE TABLE CUSTOMER_ORDER

CREATE TABLE customer_order (order_id NUMBER(10) NOT NULL, customer_id NUMBER(10) NOT NULL, order_date DATE NOT NULL, processed_date DATE NOT NULL, announced_delivery_date DATE NOT NULL, delivery_date DATE NOT NULL, payment_date DATE NOT NULL, employee_id NUMBER(10) NOT NULL, PRIMARY KEY (order_id), CONSTRAINT fk_employee FOREIGN KEY (employee_id) REFERENCES employee(employee_id), CONSTRAINT fk_customer FOREIGN KEY (customer_id) REFERENCES customer(customer_id));

CREATE TABLE CUSTOMER_ORDER COMPRESSED

CREATE TABLE customer_order (order_id NUMBER(10) NOT NULL, customer_id NUMBER(10) NOT NULL, order_date DATE NOT NULL, processed_date DATE NOT NULL, announced_delivery_date DATE NOT NULL, delivery_date DATE NOT NULL, payment_date DATE NOT NULL, employee_id NUMBER(10) NOT NULL, PRIMARY KEY (order_id), CONSTRAINT fk_employee FOREIGN KEY (employee_id) REFERENCES employee(employee_id), CONSTRAINT fk_customer FOREIGN KEY (customer_id) REFERENCES customer(customer_id)) COMPRESS (order_date BY DICTIONARY, processed_date BY DICTIONARY, announced_delivery_date BY DICTIONARY, delivery_date BY DICTIONARY, payment_date BY DICTIONARY) OPTIMIZED FOR READ;

CREATE TABLE CUSTOMER_ORDER WITHOUT FOREIGN KEYS

CREATE TABLE customer_order (order_id NUMBER(10) NOT NULL, customer_id NUMBER(10) NOT NULL, order_date DATE NOT NULL, processed_date DATE NOT NULL, announced_delivery_date DATE NOT NULL, delivery_date DATE NOT NULL, payment_date DATE NOT NULL, employee_id NUMBER(10) NOT NULL, PRIMARY KEY (order_id));

CREATE TABLE CUSTOMER_ORDER WITHOUT FOREIGN KEYS COMPRESSED

CREATE TABLE customer_order (order_id NUMBER(10) NOT NULL, customer_id NUMBER(10) NOT NULL, order_date DATE NOT NULL, processed_date DATE NOT NULL, announced_delivery_date DATE NOT NULL, delivery_date DATE NOT NULL, payment_date DATE NOT NULL, employee_id NUMBER(10) NOT NULL, PRIMARY KEY (order_id)) COMPRESS (order_date BY DICTIONARY, processed_date BY DICTIONARY, announced_delivery_date BY DICTIONARY, delivery_date BY DICTIONARY, payment_date BY DICTIONARY) OPTIMIZED FOR READ;

CREATE TABLE CUSTOMER_ORDER_ITEM

CREATE TABLE customer_order_item (order_id NUMBER(10) NOT NULL, product_id NUMBER(10) NOT NULL, quantity NUMBER(10) NOT NULL, PRIMARY KEY (order_id, product_id), CONSTRAINT fk_order FOREIGN KEY (order_id) REFERENCES customer_order(order_id), CONSTRAINT fk_product FOREIGN KEY (product_id) REFERENCES product(product_id));

CREATE TABLE CUSTOMER_ORDER_ITEM WITHOUT FOREIGN KEYS

CREATE TABLE customer_order_item (order_id NUMBER(10) NOT NULL, product_id NUMBER(10) NOT NULL, quantity NUMBER(10) NOT NULL, PRIMARY KEY (order_id, product_id));

CREATE TABLE EMPLOYEE

CREATE TABLE employee (employee_id NUMBER(10) NOT NULL, department_nr NUMBER(10) NOT NULL, last_name VARCHAR2(50) NOT NULL, first_name VARCHAR2(50) NOT NULL, PRIMARY KEY (employee_id));

CREATE TABLE EMPLOYEE COMPRESSED

CREATE TABLE employee (employee_id NUMBER(10) NOT NULL, department_nr NUMBER(10) NOT NULL, last_name VARCHAR2(50) NOT NULL, first_name VARCHAR2(50) NOT NULL, PRIMARY KEY (employee_id)) COMPRESS (last_name BY DICTIONARY, first_name BY DICTIONARY) OPTIMIZED FOR READ;

CREATE TABLE PRODUCT

CREATE TABLE product (product_id NUMBER(10) NOT NULL, product_name VARCHAR2(300) NOT NULL, price NUMBER(10,2) NOT NULL, category VARCHAR2(20) NOT NULL, PRIMARY KEY (product_id));

CREATE TABLE PRODUCT COMPRESSED

CREATE TABLE product (product_id NUMBER(10) NOT NULL, product_name VARCHAR2(300) NOT NULL, price NUMBER(10,2) NOT NULL, category VARCHAR2(20) NOT NULL, PRIMARY KEY (product_id)) COMPRESS (product_name BY DICTIONARY, category BY DICTIONARY) OPTIMIZED FOR READ;

CREATE TABLE PURCHASE

```
CREATE TABLE purchase (product_id NUMBER(10) NOT NULL, supplier_id NUMBER(10) NOT NULL, PRIMARY KEY ( product_id, supplier_id), CONSTRAINT fk_supplier FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id), CONSTRAINT fk_product FOREIGN KEY (product_id) REFERENCES product(product_id));
```

CREATE TABLE PURCHASE WITHOUT FOREIGN KEYS

```
CREATE TABLE purchase (product_id NUMBER(10) NOT NULL, supplier_id NUMBER(10) NOT NULL, PRIMARY KEY ( product_id, supplier_id));
```

CREATE TABLE SUPPLIER

```
CREATE TABLE supplier ( supplier_id NUMBER(10) NOT NULL, supplier_name VARCHAR2(50) NOT NULL, legal_form VARCHAR2(10) NOT NULL, street VARCHAR2(50) NOT NULL, house_number NUMBER(3) NOT NULL, city VARCHAR2(30) NOT NULL, country VARCHAR2(25) NOT NULL, PRIMARY KEY ( supplier_id));
```

CREATE TABLE SUPPLIER COMPRESSED

```
CREATE TABLE supplier (supplier_id NUMBER(10) NOT NULL, supplier_name VARCHAR2(50) NOT NULL, legal_form VARCHAR2(10) NOT NULL, street VARCHAR2(50) NOT NULL, house_number NUMBER(3) NOT NULL, city VARCHAR2(30) NOT NULL, country VARCHAR2(25) NOT NULL, PRIMARY KEY ( supplier_id)) COMPRESS (supplier_name BY DICTIONARY, legal_form BY DICTIONARY, street BY DICTIONARY, city BY DICTIONARY, country BY DICTIONARY) OPTIMIZED FOR READ;
```

B Software-Versionen und zusätzliche Informationen

In der nachfolgenden Tabelle B.1 werden die Betriebssysteme mit den darauf ablaufenden Datenbanksystemen dargestellt.

Betriebssystem	**Ablaufende Datenbanksysteme**
Windows Server 2012 R2 - Version: 6.3 (Build 9600)	TimesTen, SQL Server 2014, SQL Server 2016
Ubuntu Server - Version: 16.10	VoltDB
EXAClusterOS - Als virtuelle Maschine - Basierend auf einem modifiziertem CentOS 6.8 - Ausführung mittels VirtualBox 5.1.112 inklusive des Extension Packs	EXASOL

Tabelle B.1 Übersicht Betriebssysteme und Datenbanksysteme

Für die Erstellung der Abbilder wurde clonezilla in der Version clonezilla-live-2.4.9.17 genutzt.

Die Versionen der Datenbanksysteme sowie die Software für die Konfiguration, soweit erforderlich, werden in Tabelle B.2 angegeben.

Datenbanksystem	**Version**
TimesTen - SQL Developer 4.1.5.21	11.2.2.8
VoltDB - Java Version 1.8.112	6.9
EXASOL - EXAplus 6.0.rc1	V6.0 Free Small Business Edition RC1
SQL Server 2014 - SQL Management Studio 13.0.16100.1 - .NET Framework Version 3.5	12.0.5338.0
SQL Server 2016 - SQL Management Studio 13.0.16100.1 - .NET Framework Version 4.6.1	13.0.4411.0

Tabelle B.2 Versionen der Datenbanksysteme und dazugehöriger Software

Die nachfolgende tabellarische Darstellung B.3 gibt einen Überblick über die verwendete Software, die für die Programmierung und Durchführung der Versuche genutzt wurde.

Software	**Kurze Beschreibung inklusive Version**
Java	Programmiersprache, Version 8 Update 112.
Eclipse	Entwicklungsumgebung, Version Neon.1a Release 4.6.1.
m2e	Build-Management-Tool Maven Integration für Eclipse Version 1.7.0.20160603
JMH (Java Microbenchmarking Harness)	Framework für Benchmarks Version 1.17.3
Chronicle-Queue	Java-Library für eine persistente Queue Version 4.5.19
JUnit	Test-Framework Version 4.12
SLF4J-NOP	Unterbinden der Logger-Ausgabe Version 1.7.22

Tabelle B.3 Software für die Programmierung und Durchführung der Tests

C Datenkompression bei TimesTen

Die Nutzung der Kompression von Daten in Tabellen lässt sich durch die Verwendung der COMPRESS Anweisung am Ende der Tabellendefinition oder in Form von ALTER TABLE durchführen. Dabei besteht die Möglichkeit die Anzahl der einzigartigen Werte für ein Attribut anzugeben, für die eine Kompression durchgeführt werden wird. Erfolgt eine Angabe der unterschiedlichen Werte, so lassen sich die Daten unter Umständen stärker komprimieren. Sind weniger als 256 Werte vorhanden, so verweist ein 1 Byte großer Pointer auf die Daten. Ein 2 Byte großer Pointer wird verwendet, wenn weniger als 65.535 einzigartige Werte im Wörterbuch vorhanden sind. Ansonsten werden für bis zu 4.294.967.295 unterschiedliche Werte 4 Byte Pointer verwendet, wobei der 4 Byte Pointer der Standard ist, wenn keine Angabe für die einzigartigen Werte erfolgt. (vgl. Oracle, 2015, S. 6-146f.)

Als Beispiel für die Kompression ohne Angabe eines Maximalwertes dient folgender Code:

```
CREATE TABLE customer
        (customer_id NUMBER(10) NOT NULL PRIMARY KEY,
        last_name VARCHAR2(50) NOT NULL,
        first_name VARCHAR2(50) NOT NULL,
        street VARCHAR2(50) NOT NULL, house_number NUMBER(3) NOT NULL, city
        VARCHAR2(30)  NOT NULL,
        country VARCHAR2(25) NOT NULL,
        birthday DATE NOT NULL,
        registration DATE NOT NULL,
        gender VARCHAR2(1) NOT NULL)
        COMPRESS (last_name BY DICTIONARY, first_name BY DICTIONARY,
        street BY DICTIONARY, city BY DICTIONARY, country BY DICTIONARY,
        birthday BY DICTIONARY, registration BY DICTIONARY, gender BY
        DICTIONARY) OPTIMIZED FOR READ;
```

Hier werden die Daten für last_name, first_name, street, city, country, birthday, registration sowie gender im Wörterbuch gespeichert, wobei für jeden Verweis 4 Byte Pointer genutzt werden.

Betrachtet man nun die für Testdaten verwendeten Angaben 1500 männliche und 1500 weibliche Vornamen, 1500 Nachnamen, 2600 Städte, 100 Länder und 400 Straßennamen für Customer, so kann man einen höheren Kompressionsgrad erreichen, da für alle Mengenangaben unter 65535 2-Byte Pointer verwendet werden. So kann gegenüber der sonst standardmäßig 4 Byte Pointer-Nutzung pro Pointer 2 Byte eingespart werden, was insbesondere bei vielen Daten bedeutsam ist.

Die Bezeichnung TimesTen STD-KOMP steht für die standardmäßige Kompression der Daten mithilfe der 4 Byte Pointer-Nutzung. Das bedeutet, dass keine maximale Anzahl an unterschiedlichen Werten bei der Kompressionsanweisung angegeben wird und so unter Umständen nicht die bestmögliche Datenkomprimierung erreicht wird.

Für eine möglichst hohe Komprimierung der Datensätze kann anhand der Angabe der zu erwartenden Anzahl an unterschiedlichen Werte die Wahl der zu nutzenden Pointer beeinflusst werden. In dieser Arbeit wird diese Variante der Nutzung von

Kompression als TimesTen SPEZ-KOMP bezeichnet. Dabei erfolgt die Kompression anhand der nachfolgend näher beschriebenen Spezifikation.

Um den geringstmöglichen Speicherverbrauch unter Einsatz der Datenkompression zu erreichen, wird die Anzahl der einzigartigen Werte pro Spalte, bei der eine Kompression der Werte zu einem geringeren Speicherverbrauch führt beziehungsweise sehr wahrscheinlich zur Folge hat, ermittelt. Anschließend werden die so spezifizierten Werte als Anzahl der maximal zu erwartenden Ausprägungen für die entsprechende Spalte in der Kompressionsanweisung angegeben.

D Einrichtung der Datenbanksysteme

In den nachfolgenden Unterkapiteln werden die Konfigurationen der Datenbanksysteme näher beschrieben. Vorweg ist anzumerken, dass die erstellten Benutzer über sämtliche Datenbankrechte verfügen, was die Durchführung der Tests erleichtert.

D.1 TimesTen

Die Einrichtung einer TimesTen-Datenbank geschieht mittels der Erstellung eines *Data Source Name* (nachfolgend auch DSN). Über die *Systemsteuerung* erreicht man das Element *Verwaltung*. Mittels Doppelklick auf Verwaltung gelangt man zu diversen Verknüpfungen, die bei der Organisation des Betriebssystems hilfreich sind. Dort die Verknüpfung *ODBC-Datenquellen (64-Bit)* ausführen.

Sobald die Anwendung *ODBC-Datenquellen-Administrator (64-Bit)* gestartet wurde, zu dem Reiter *System-DSN* wechseln. Die Schaltfläche *Hinzufügen* anklicken und nachfolgend den *TimesTen Data Manager 11.2.2* auswählen und auf *Fertigstellen* klicken.

Im nun erscheinenden Fenster im Reiter *Data Store* bei *Data Source Name** einen Namen für die Datenbank eintragen. Die Pfadangabe für die Datenbankdateien auf der Festplatte bei *Data Store Path + Name** und die Pfadangabe für die Transaktionslogdateien bei *Transaction Log Directory*“ eintragen. Anschließend den Eintrag bei *Database Character Set*** auf *UTF 8* einstellen. Der *Type Mode* wird auf *0 – Oracle* gesetzt.

Im Reiter *First Connection* die Angabe für *Permanent Data Size* auf *4096 MB* setzen und bei *Temporary Data Size* den Wert auf *1024 MB* einstellen.

Anschließend auf den Reiter *General Connection* wechseln und für *Query Timeout* den Wert 300 eintragen.

Im Reiter *NLS Connection* für *Connection Character Set* den Eintrag *UTF8* setzen und bei *NLS_LENGTH_SEMANTICS* den Wert auf *CHAR* einstellen. Anschließend auf *OK* klicken und die Einrichtung bezüglich des DSN ist fertig.

Zum Abschluss soll die Datenbank so eingestellt werden, dass die Datenbank die Daten ständig im Hauptspeicher zur Verfügung hält und nicht die Daten bei jedem Verbindungsaufbau von der Festplatte liest, sofern keine Verbindung zu der Datenbank besteht. Zusätzlich soll auch ein Benutzer mit Administrationsrechten angelegt werden, damit die Datenbank auch von anderen Rechnern aus genutzt werden kann.

Über die Kommandozeile wird zuerst der Benutzer angelegt. Mittels der Ausführung von *ttisql* wird ein Kommandozeilenprogramm gestartet, das Operationen mit der Datenbank ermöglicht. Daraufhin mittels *CONNECT Datenbankname*[1]*;* sich mit der Datenbank verbinden. Mittels des Statements *CREATE USER userX IDENTIFIED BY passwordY;* wird ein neuer Benutzer angelegt. Anschließend werden diesem alle Datenbankrechte mittels *GRANT ALL PRIVILEGES TO userX;* gegeben und den Abschluss bildet die Ausführung eines *COMMIT;*-Statements. Das Programm *ttisql* kann mittels *EXIT;* verlassen werden.

Für das permanente Halten von Daten im Hauptspeicher wird über die Kommandozeile die Anweisung *ttAdmin -ramPolicy always DATENBANKNAME* abgesetzt und anschließend wird dies über eine Ausgabe auf der Kommandozeile bestätigt. Anschließend ist das Datenbanksystem fertig eingerichtet.

D.2 SQL Server 2014 und 2016

Die Einrichtung der Datenbank in SQL Server 2014 und SQL Server 2016 wird mittels der von Microsoft bereitgestellten Software *SQL Server Management Studio* durchgeführt. Dieses Programm wird auf dem Desktop-Rechner installiert, auf dem auch die SQL-Server-Instanz ausgeführt wird.

Nach dem Start von *SQL Server Management Studio* verbindet man sich mit der SQL-Server-Instanz mittels der bei der Installation eingestellten Art der Authentifizierung. Sobald die Verbindung hergestellt wurde, erweitert man die *Instanz*, falls noch nicht

[1] Der Datenbankname ist der Name, der bei *Data Source Name** eingetragen wurde.

geschehen, über den *Objekt-Explorer*. Anschließend führt man einen Rechtsklick auf den Ordner *Sicherheit* aus und navigiert im so entstandenen Kontextmenü über *Neu* zu *Anmeldung* und wählt diese mit einem Linksklick aus.

Im nun erscheinenden Fenster wird unter der Seite *Allgemein* die *SQL Server-Authentifizierung* ausgewählt und es wird ein Kennwort eingetragen. Unter *Anmeldename* wird der Name des neuen Benutzers angelegt. Den Haken bei *Ablauf des Kennworts erzwingen* entfernen, sofern dieser gesetzt ist. Anschließend werden unter der Seite *Serverrollen* dem so neu entstehenden Benutzer alle Rechte beziehungsweise alle Rollen erteilt. Den Abschluss bildet der Klick auf OK und der neue Benutzer ist vorhanden.

Die Verwaltung des Arbeitsspeichers für die SQL-Server-Instanz erfolgt mittels Rechtsklick auf den *Namen der Instanz* im *Objekt-Explorer*. Im so entstehenden Kontextmenü wird das Feld *Eigenschaften* ausgewählt. Die Seite *Arbeitsspeicher* wird angewählt und unter *Mindestmenge an Serverarbeitsspeicher (in MB)* und *Maximaler Serverarbeitsspeicher (in MB)* wird jeweils *5120* eingetragen. Der SQL-Server-Instanz sind somit 5120 MB Hauptspeicher fest zugeteilt.

Die Datenbank wird eingerichtet, in dem im *Objekt-Explorer* die Server-Instanz mittels eines Klicks auf das + erweitert wird, falls nicht bereits geschehen. Anschließend wird ein Rechtsklick auf dem Ordner *Datenbanken* durchgeführt und im Kontextmenü *Neue Datenbank…* ausgewählt mittels Linksklick. Im nun erscheinenden Fenster wird unter der Seite *Allgemein* ein Datenbankname vergeben. Daraufhin wird auf die Seite *Dateigruppen* gewechselt, wo im Bereich *MEMORY OPTIMIZED DATA* mittels der Schaltfläche *Dateigruppe hinzufügen* eine neue Dateigruppe erstellt wird, die einen Namen erhält. Anschließend wird auf die Seite *Allgemein* gewechselt und mittels *Hinzufügen* wird eine neue Datenbankdatei erstellt, die den Dateityp *FILESTREAM* und einen Namen zugewiesen bekommt. Anschließend wird mittels *OK* die Datenbank erstellt.

Den Abschluss der Einrichtung bilden die Hauptspeicher-Einstellungen der der speicheroptimierten Tabellen. Mittels Rechtsklick auf den Namen der Datenbank wird ein Kontextmenü geöffnet, bei dem das Feld *Neue Abfrage* ausgewählt wird. Im nun auftauchenden Fenster wird zuerst ein Ressourcenpool mittels *CREATE RESOURCE POOL Poolname WITH (MIN_MEMORY_PERCENT = 84, MAX_MEMORY_PERCENT = 84);* erstellt. Anschließend wird die Verwaltungseinheit des Speichers mittels *ALTER RESOURCE GOVERNOR RECONFIGURE;* neukonfiguriert. Daraufhin erfolgt noch eine Bindung der Datenbank an den Ressourcenpool mittels

EXEC sp_xtp_bind_db_resource_pool Datenbankname, Poolname; und den Abschluss bildet ein Neustart der Datenbank. Das Offline- und Online-Schalten der Datenbank wird mittels Rechtsklick auf den Datenbanknamen und anschließender Navigation über *Task* zu *Offline schalten* bzw. *Online schalten* durchgeführt. Danach ist die Datenbank für den Hauptspeicherbetrieb eingerichtet.

D.3 EXASOL

Die importierte virtuelle Maschine wird so eingestellt, dass diese *4 Prozessoren* und *5120 MB* Arbeitsspeicher erhält. Dazu werden im *Einstellungsmenü* der virtuellen Maschine von EXASOL unter der Auswahl *System* im *Reiter Hauptplatine* für Hauptspeicher *5120 MB* eingetragen und im *Reiter Prozessor* die Anzahl der Prozessoren auf 4 gesetzt.

Nach dem Start der virtuellen Maschine lässt sich über das im Fenster der virtuellen Maschine angegebene *Webinterface* die Hauptspeicherverwendung einstellen. Hier muss gegebenenfalls eine unsichere Verbindung zugelassen werden, um die Konfiguration vorzunehmen. Anschließend mit den Anmeldedaten, die in der virtuellen Maschine angezeigt werden, einloggen. Im *Webinterface* wird der Punkt *ExaSolution* angewählt und die *Checkbox* der Datenbankinstanz mit einem *Haken* versehen und mittels der Schaltfläche *Shutdown* heruntergefahren. Sobald die Datenbank ausgeschaltet ist, lässt sich mit einem Klick auf den Namen der Datenbank (im Normalfall *EXAone*) ein Menü mit erweiterten Informationen und Interaktionsmöglichkeiten aufrufen. Dort wird die Schaltfläche *Edit* ausgewählt und unter dem Punk *DB RAM (GiB)* wird der Wert *4* eingetragen. Anschließend mittels *Apply* die Änderungen übernehmen und im auftauchenden Menü *EXASolution Instance* unter dem Punkt *Actions* im Dropdown-Menü *Startup* auswählen. Mittels einem Klick auf den Button *Submit* wird die Datenbank neugestartet. Das Webinterface kann nun verlassen werden.

Für die Einrichtung eines neuen Datenbanknutzers wird *EXAplus* genutzt. Sobald *EXAplus* gestartet wurde, ist es möglich mit den Anmeldedaten, die für EXAplus gelten, eine Verbindung aufzubauen. Anschließend sollte der *SQL-Editor* offen sein. Ist das nicht der Fall, über *Datei* den Punkt *Neuer SQL-Editor* anklicken und nun sollte der Editor vorhanden sein. Anschließend den Befehl *CREATE USER userX IDENTIFIED BY "passwordY"*[1]*;* im Editor eingeben und ausführen lassen. Nun noch sämtliche Rechte mittels *GRANT ALL PRIVILEGES TO userX;* dem neu angelegten Benutzer geben und die Einrichtung ist abgeschlossen.

[1] Die Einführungszeichen müssen mit angegeben werden.

D.4 VoltDB

Bei VoltDB lässt sich die Konfiguration mittels eines Texteditors durchführen. Im ausgeschalteten Zustand des Datenbanksystems wird mittels eines entsprechenden Editors die Datei *deployment.xml* im Unterverzeichnis *config* des *initialisierten Datenbankordners* geöffnet. Anschließend wird die Konfiguration an die nachfolgenden Angaben angepasst:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<deployment>
    <cluster hostcount="1" sitesperhost="2" kfactor="0"
elastic="enabled" schema="ddl"/>
    <partition-detection enabled="true">
        <snapshot prefix="partition_detection"/>
    </partition-detection>
    <admin-mode port="21211" adminstartup="false"/>
    <heartbeat timeout="90"/>
    <httpd enabled="true">
        <jsonapi enabled="true"/>
    </httpd>
    <snapshot frequency="24h" retain="2"
prefix="AUTOSNAP" enabled="false"/>
    <commandlog synchronous="false" enabled="true"
logsize="1024">
        <frequency time="200" transactions="2147483647"/>
    </commandlog>
    <systemsettings>
        <temptables maxsize="1024"/>
        <snapshot priority="6"/>
        <elastic duration="50" throughput="2"/>
        <query timeout="300000"/>
        <resourcemonitor frequency="60">
            <memorylimit size="4" alert="70%"/>
        </resourcemonitor>
    </systemsettings>
    <security enabled="false" provider="hash"/>
</deployment>
```

Nachdem die Konfiguration durchgeführt wurde, kann die Datenbank anschließend neugestartet werden. Anzumerken ist, dass VoltDB ohne Sicherheitseinstellungen verwendet wird und in diesem Zusammenhang auch keine Benutzer existieren.

E Inhalt der CD

Die der Arbeit beiliegende CD hat folgenden Inhalt:

- BA_Arfert.pdf
- Laufzeitanalyse_Messungen_Graphen.xlsx
- Laufzeitanalyse_Messungen.xlsx
- README.docx
- Code.zip
- Messungen.zip
- Testdaten.zip
- Ueberpruefung.zip

Literaturverzeichnis

Bog, Anja; Plattner, Hasso; Zeier, Alexander: A mixed transaction processing and operational reporting benchmark, in: Information Systems Frontiers, Jahrgang 13, Ausgabe 13, Seiten 321-335, Hingham: Kluwer Academic Publishers, 2011.

De Bruijn, Jos: In-Memory OLTP – what's new in SQL2016 CTP3, 2015
https://blogs.msdn.microsoft.com/sqlserverstorageengine/2015/11/12/in-memory-oltp-whats-new-in-sql2016-ctp3/
(Abruf 25.01.2017)

Diaconu, Cristian; Freedman, Craig; Ismert, Erik; Larson, Per-Ake; Mittal, Pravin; Stonecipher, Ryan; Verma, Nitin; Zwilling, Mike: Hekaton: SQL server's memory-optimized OLTP engine, in: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13), Seiten 1243-1254, New York: ACM, 2013.

Eich, Margaret H.: Main Memory Database Research Directions, in: Database Machines, Sixth International Workshop, {IWDM} '89, Deauville, France, June 19-21, 1989, Proceedings, Seiten 251-268, Berlin: Springer-Verlag, 1989.

EXASOL AG: A Peek under the Hood, 2016.
http://www.exasol.com/info/downloads/technisches-whitepaper-exasol-1/?aliId=1155806 (Abruf: 08.11.2016)

EXASOL AG: EXASOL – Die schnellste analytische Datenbank der Welt, 2015a. http://www.exasol.com/info/downloads/business-whitepaper-exasol/?aliId=1155808 (Abruf: 08.11.2016)

EXASOL AG: EXASolo does not use more RAM after increase in VM RAM, 2015c, https://www.exasol.com/support/browse/SOL-282 (Abruf: 05.02.2017)

EXASOL AG: EXASolution, 2012 http://www.exasol.com/fileadmin/content-de/pdf/Whitepaper/EXASolution_TWP_EN_082012.pdf (Abruf: 08.11.2016)

EXASOL AG: Transaction System, 2014, https://www.exasol.com/support/browse/SOL-135 (Abruf: 22.01.2017)

EXASOL AG: EXASolution User Manual Version 5.0.13, 2015b https://www.exasol.com/support/secure/attachment/38778/EXASolution_User_Manual-5.0.13-en.pdf (Abruf: 08.11.2016)

Garcia-Molina, Hector; Salem, Kenneth: Main memory database systems: an overview, in: IEEE Transactions on Knowledge and Data Engineering, Jahrgang 4, Ausgabe 6, Seiten 509-516, Piscataway: IEEE Educational Activities Department, 1992.

Gessert, Felix; Wingerath, Wolfram; Friedrich, Steffen; Ritter, Norbert: NoSQL database systems: a survey and decisision guidance, in: Computer Science – Research and Development, Berlin: Springer-Verlag, 2016

Harizopoulos, Stavros; Abadi, Daniel J. et al.: OLTP through the looking glass, and what we found there, in: Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD '08), Seiten 981-992, New York, NY, USA: ACM, 2008.

Kemper, Alfons; Eickler, André: Datenbanksysteme: Eine Einführung, 10. Auflage, Berlin: De Gruyter Oldenbourg, 2015.

Kian-Lee, Tan; Qingchao, Cai; Beng Chin, Ooi; Weng-Fai, Wong; Chang, Yao; Hao, Zhang: In-memory Databases: Challenges and Opportunities From Software and

Hardware Perspectives, in: ACM SIGMOD Record, Jahrgang 44, Ausgabe 2, 2015, Seiten 35-40, New York, NY, USA: ACM, 2015.

Larson, Per-Åke; Levandoski, Justin: Modern main-memory database systems, in: Proceeding of the VLDB Endowment, Jahrgang 9, Ausgabe 13, 2016, Seiten 1609-1610, VLDB Endowment, 2016.

Larson, Per-Ake; Zwilling, Mike; Farlee, Kevin: The Hekaton Memory-Optimized OLTP Engine, in: IEEE Data Engineering Bulletin, Jahrgang 36, Ausgabe 2, Seiten 34-40, New York, NY, USA: ACM, 2013.

Loos, Peter; Lechtenbörger, Jens; Vossen, Gottfried et al.: In-memory Databases in Business Information Systems, in: Business & Information Systems Engineering, Jahrgang 3, Ausgabe 6, Seiten 389-395, Springer Gabler, 2011.

Matt, Christian: In-Memory-Technologien für Unternehmensanwendungen, in: Controlling & Management, Jahrgang 56, Ausgabe 4, Seiten 229-230, Springer-Gabler, 2012.

Microsoft: In-Memory OLTP (Arbeitsspeicheroptimierung), 2016a
https://msdn.microsoft.com/de-de/library/dn133186(v=sql.120).aspx
(Abruf: 24.01.2017)

Microsoft: Tabellen- und Zeilengrößen in speicheroptimierten Tabellen, 2016b
https://msdn.microsoft.com/de-de/library/dn205318(v=sql.120).aspx
(Abruf: 24.01.2017)

Microsoft: Transact-SQL Constructs Not Supported by In-Memory OLTP, 2016c
https://msdn.microsoft.com/en-us/library/dn246937(v=sql.130).aspx
(Abruf: 24.01.2017)

Oracle: Database SQL Language Reference 11gRelease 2 (11.2), 2016
https://docs.oracle.com/cd/E11882_01/server.112/e41084.pdf
(Abruf: 22.01.2017)

Oracle: Extreme Performance Using Oracle TimesTen In-Memory Database, 2014a.
http://www.oracle.com/technetwork/products/timesten/overview/wp-timesten-tech-132016.pdf
(Abruf: 07.11.2016)

Oracle: TimesTen In-Memory Database PL/SQL Developer's Guide 11g Release 2 (11.2.2), 2014c, https://docs.oracle.com/cd/E21901_01/timesten.1122/e21639.pdf (Abruf: 22.01.2017)

Oracle: TimesTen SQL Reference 11gRelease 2 (11.2.2), 2015, https://docs.oracle.com/cd/E21901_01/timesten.1122/e21642.pdf (Abruf: 05.02.2017)

Plattner, Hasso: Lehrbuch In-Memory Data Management: Grundlagen der In-Memory-Technologie, Wiesbaden: Springer Gabler, 2013.

Savage, Neil: The power of memory, in: *Communications of the ACM, Jahrgang* 57, Ausgabe 9, 2014, Seiten 15-17, New York, NY, USA: ACM, 2014

Tirthankar Lahiri, Marie-Anne Neimat, Steve Folkman: Oracle TimesTen: An In-Memory Database for Enterprise Applications, in: IEEE Data Engineering Bulletin, Jahrgang 36, Ausgabe 2, Seiten 6-13, 2013.

TPC: TPC-H - Top Ten Performance Results Version 2 Results, 2017 http://www.tpc.org/tpch/results/tpch_perf_results.asp (Abruf: 26.01.2017)

VoltDB Inc.: HIGH PERFORMANCE, SCALABLE RDBMS FOR BIG DATA AND REAL-TIME ANALYTICS, o. J. https://downloads.voltdb.com/datasheets_collateral/technical_overview.pdf (Abruf: 08.11.2016)

VoltDB Inc.: HIGH PERFORMANCE, SCALE-OUT RDBMS FOR FAST DATA APPS REQUIRING REAL-TIME ANALYTICS WITH TRANSACTIONS, 2016a, https://www.voltdb.com/hubfs/content/white_papers/hv_white_papers/hv-white-paper-voltdb-technical-overview.pdf?t=1474655628996 (Abruf: 08.11.2016)

VoltDB Inc.: Using VoltDB, 2016b, http://downloads.voltdb.com/documentation/UsingVoltDB.pdf (Abruf: 08.11.2016)

Weiner, Mike; Levin, Ami: In-Memory OLTP Common Workload Patterns and Migration Considerations, 2014. https://msdn.microsoft.com/library/dn673538.aspx (Abruf: 08.11.2016)

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den 16.05.2017 _____