technische universität dortmund

IT in Produktion und Logistik

Analyse von NoSQL-Datenbanken zur bedarfsgerechten Darstellung von Supply-Chain-Daten

Bachelorarbeit

der Technischen Universität Dortmund

der Fakultät Maschinenbau

dem Fachgebiet IT in Produktion und Logistik

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.) Logistik

Betreuer: Univ.-Prof. Dr.-Ing. Markus Rabe

Dr.-Ing. Dipl.-Inf. Anne Antonia Scheidler

Vorgelegt von: Artur Lupatsiy

Matrikelnummer: 138117

Dortmund, 17. September 2018

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

Abb. Abbildung

APS Advanced Planning and Scheduling

d.h. das heißt

DBS Datenbanksystem

DB Datenbank

ERP Enterprise Resource Planning

MRP Material Resource Planning

SC Supply Chain

SCM Supply Chain Management

PPS Produktionsplanung und -steuerung

z.B. zum Beispiel

1. Einleitung

Durch den vermehrten Einsatz von Informationstechnologie (IT) in der logistischen Industrie entstehen immer umfangreichere Datenmengen in Supply Chains. Zudem bringen unterschiedliche Trends in der Logistik die Entstehung schnelllebiger und heterogener Datenmengen in Supply Chains mit sich. Dabei stoßen relationale Datenbanken bei der Verarbeitung dieser Daten an ihre Grenzen. Um die gesammelten Informationen bedarfsgerecht aufbereiten und speichern zu können, wird aus diesem Grund ein nicht relationaler Ansatz im Datenmanagement verwendet [Mei16, S3ff]. Ein wesentlicher Bestandteil von Supply Chains sind Transaktionsdaten, die in Transaktionssystemen, wie ERP-Systemen, als Entscheidungsgrundlage zur Prozessoptimierung genutzt werden. Die Daten werden unter anderem in der Bedarfs-, Distributions- und Produktionsplanung einzelner Unternehmen der Supply Chain erzeugt. Hier liefert nicht nur die einzelne Verwaltung der Daten, sondern auch die Nutzung verschiedener Software-Lösungen zur Datenorganisation, Daten in unterschiedlicher Qualität und Form [Kuh02, S10ff; Hei11, S.23ff; Sch16, S.101ff; Mei16, S.17ff]. Die so entstehenden heterogenen und umfangreichen Transaktions-daten können nicht mehr alleine durch relationale Datenbanksysteme zweckmäßig gespeichert und aufbereitet werden. Folglich ist eine Erweiterung des bestehenden relationalen Ansatzes zur Datenorganisation und -analyse um eine neue IT-Lösung, wie NoSQL-Datenbanken, gefordert [Sch16, S. 104]. Der Vorteil einer NoSQL-Datenbank besteht unter anderem in der Schemafreiheit, in der Vermeidung sogenannter Joins und der horizontalen Skalierung. Damit sind NoSQL-Datenbanken für große Datenmengen besonders geeignet.
Momentan sind mehr als 225 NoSQL-Datenbanken auf dem internationalen Markt zu finden. [Edl18] Die besondere Herausforderung besteht in der Auswahl einer passenden Datenbank zur Verwertung der anfallenden Transaktionsdaten. Aktuell gibt es keine Empfehlung in der Literatur für eine zweckmäßige Auswahl, eine solche könnte eine Hilfestellung für Unternehmen zur bedarfsgerechten Darstellung von Transaktionsdaten sein.

Das Ziel dieser Arbeit ist es, eine Empfehlung für eine NoSQL-Datenbank zur bedarfsgerechten Darstellung von Supply-Chain-Daten zu geben. Voraussetzung dafür ist der Vergleich von gängigen NoSQL-Datenbanken im Kontext von Supply-Chain-Daten. Um dies zu erreichen, wird aufgezeigt, wo und welche Daten in der Supply Chain anfallen und wie sie sich klassifizieren lassen. Ebenso werden

Möglichkeiten zur Datenspeicherung und -aufbereitung in Form von NoSQL-Datenbanken erläutert. Anschließend werden relevante Vergleichskriterien abgeleitet, um NoSQL-Datenbanken mithilfe einer Vergleichsmatrix gegeneinander abzuwägen. Mit der auf diesem Wege ausgewählten NoSQL-Datenbank wird exemplarisch die Darstellung eines konkreten Transaktionsdatensatzes demonstriert.

Als Basis für die Analyse von NoSQL-Datenbanken dienen gängige Daten, die im Datenmanagement von Supply Chains gespeichert und aufbereitet werden. Hierzu werden zuerst die Prozesse und Ziele vom Supply Chain Management aufgeführt, um darauf folgend Informationssysteme der Supply Chain vorzustellen. Schließlich werden die in den Informationssystemen entstehenden Daten der Supply Chain aufgezeigt und anhand ihrer Relevanz für Transaktionen innerhalb der Supply Chain abgegrenzt (Kapitel 2.1). Zusätzlich werden grundlegende Datenmodellierungs-methoden und die Eigenschaften relationaler Datenbanken vorgestellt (Kapitel 2.2). Es werden die Eigenschaften und Kernklassen von NoSQL-Systemen erläutert, um darauf folgend die in der Literatur erwähnten Vergleichskriterien zur Auswahl einer NoSQL-Datenbank aufzulisten (Kapitel 2.3). Es wird jeweils eine NoSQL-Datenbank der jeweiligen Kernklasse ausgewählt und anhand zuvor eingegrenzter Kriterien mittels einer Vergleichsmatrix untereinander verglichen, um so die für die bedarfsgerechte Darstellung von Transaktionsdaten geeignete Datenbank finden zu können (Kapitel 3.1). Die praktische Verwertbarkeit dieser NoSQL-Datenbank wird mit Hilfe eines konkreten Datensatzes demonstriert (Kap. 3.2). Zuletzt wird ein Ausblick auf mögliche Grenzen und Potentiale der NoSQL-Datenbanksysteme gegeben.

2. Grundlagen zum Datenmanagement

In diesem Kapitel soll ein grundlegender Überblick über die Möglichkeiten des Datenmanagements in der Supply Chain gegeben werden. Zu Beginn wird auf die Prozesse und Ziele des Supply Chain Managements eingegangen, um darauf folgend die Möglichkeiten zur Unterstützung des wirtschaftlichen Handelns von Unternehmen durch Informationssysteme vorzustellen. Hierauf aufbauend werden anfallende Supply-Chain-Daten in Transaktionssystemen einzelner Unternehmen beschrieben, klassifiziert und nach relevanten Transaktionsdaten abgegrenzt. Zum Ende des Kapitels werden Möglichkeiten der Speicherung und Aufbereitung der Transaktionsdaten durch NoSQL-Datenbanken aufgezeigt.

2.1 Daten in der Supply Chain

Ziel dieses Kapitels ist es, eine Übersicht über die in der Literatur erwähnten Daten in Supply Chains zu geben. Wie in der Einleitung bereits erwähnt, ist eine zentrale Herausforderung, diese Daten bedarfsgerecht darstellen zu können. Um die passende Darstellungsform durch eine NoSQL-Datenbank zu finden, ist die Definition der Begriffe *Daten und Datenmanagement* in der Supply Chain notwendig. Im folgenden Unterkapitel werden zunächst die Prozesse und Ziele des Supply Chain Management erklärt.

2.1.1 Prozesse und Ziele des Supply Chain Managements

Um Schlüsse daraus ziehen zu können, welche Daten in der Supply Chain entstehen und wie sie verarbeitet werden, ist es hilfreich, ein allgemeines Verständnis über die Supply Chain und die Ziele und Prozesse des Supply Chain Managements zu erlangen.
Das Supply Chain Management ist die unternehmensübergreifende Planung, Gestaltung und Realisierung sowie die Koordination und Optimierung der Material- und Informationsflüsse entlang der gesamten in einer Wertschöpfungskette organisierten Unternehmen. Das Ziel des Supply Chain Managements ist die optimale und unternehmensübergreifende Gestaltung der Gesamtprozesse. [Arn08 S.59, Haa18, S.18] Hierbei wird entlang der gesamten Wertschöpfungskette insbesondere Wert gelegt auf:

- Kostenvorteile
- Zeitvorteile

- Qualitätsvorteile

Durch die Wettbewerbsfaktoren Qualität, Kosten, Zeit und die Forderung der globalen Märkte nach hoher Produktqualität und schneller Reaktionsfähigkeit, kann dieser Anspruch nur erreicht werden, indem alle wertschöpfenden Prozesse integriert und synchronisiert ablaufen. Weil die Befriedigung des Endkundenbedarfes Entscheidungsgrundlage für sämtliche Steuerungs- und Planungprozesse innerhalb einer Supply Chain ist, ist die Synchronisierung dieses globalen Netzwerkes oberstes Ziel des SCM. Abgeleitete Ziele des Supply Chain Managements sind die Minimierung der Gesamtdurchlaufzeit und der Lieferzeiten, die Bestandsreduzierung sowie die Maximierung der Liefertreue, der Planungseffizienz und eine verbesserte Ressourcenplanung. Wesentliche Voraussetzung zur Realisierung der Ziele des SCM ist die informationstechnische Verknüpfung und der Informationsaustausch (engl. Information Sharing) aller Teilnehmer der Supply Chain, um so einen durchgängigen Informationsfluss zu ermöglichen und Informationsbarrieren abzubauen. [Haa18, S.33ff; Sch06, S.23ff; Law00, S.71ff]
Entscheidend bei der Erreichung dieser Ziele ist, die enge, auf unternehmens-übergreifenden IT-Systemen basierende Vernetzung mit Kunden und Lieferanten des produzierenden Unternehmens transparent und synchron zu gestalten. Hierbei kann die Vernetzung mit den Kunden durch die Verbindung zwischen dem produzierenden Unternehmen und den nachgelagerten Distributoren, insbesondere ihrer Lagerbeständen, erreicht werden. Neben einer Erhöhung der Prognosequalität durch verbesserte Abschätzung des Verbrauchsverhaltens der Endkunden ergibt sich vor allem eine frühzeitige Erkennung von Bedarfsschwankungen, die für eine Erhöhung der verfügbaren Reaktionszeiten für die Beschaffung, Produktion und Distribution sorgen. Die Vernetzung mit den Lieferanten kann durch die Verbindung zwischen dem Produzenten und den vorgelagerten Lieferanten erreicht werden. Dies gewährleistet, dass der Bestellaufwand durch frühzeitige Offenlegung der Materialverfügbarkeit reduziert werden kann. [Kuh02, S.15ff; Law01, S.58ff] Diese Transparenz und Verfügbarkeit von Bedarfs- und Kapazitätsinformationen entlang der gesamten Supply Chain wird durch unternehmensübergreifende IT-Systeme erreicht, die eine wechselseitige Versorgung der Unternehmen mit relevanten Supply-Chain-Daten sicher stellen.
In einer Supply Chain arbeiteten mehrere Unternehmen an der Produktion eines bestimmten Endproduktes zusammen. Zu Beginn einer Wertschöpfungskette stehen die Rohstofflieferanten, welche mehrere Teilelieferanten versorgen, die wiederum die Komponentenlieferanten versorgen und diese zuletzt die Produzenten des Endproduktes beliefern. Schließlich erreicht das Produkt den Endkunden über die

verschiedenen Absatzkanäle, die der Produzent beliefert. Hierbei ist der Ausgangspunkt der SC eine konstante Kundenorientierung, da das Konzept durch eine durchgängige Orientierung an den Bedürfnissen des Endkunden gekennzeichnet ist. [Sch06, S.25] Der wertschöpfende Prozess jedes Teilnehmers einer Supply Chain besteht aus den Teilprozessen ‘Beschaffung’, aus einer vorgelagerten Stufe, der innerbetrieblichen ’Produktion’, sowie der nachgelagerten Stufe ’Absatz’. Der Informationsfluss verläuft wechselseitig über jede Stufe der Supply Chain, wobei der Materialfluss stets in Richtung Endkunden orientiert ist. Dieses Prinzip ist in Abbildung 2-1 dargestellt.

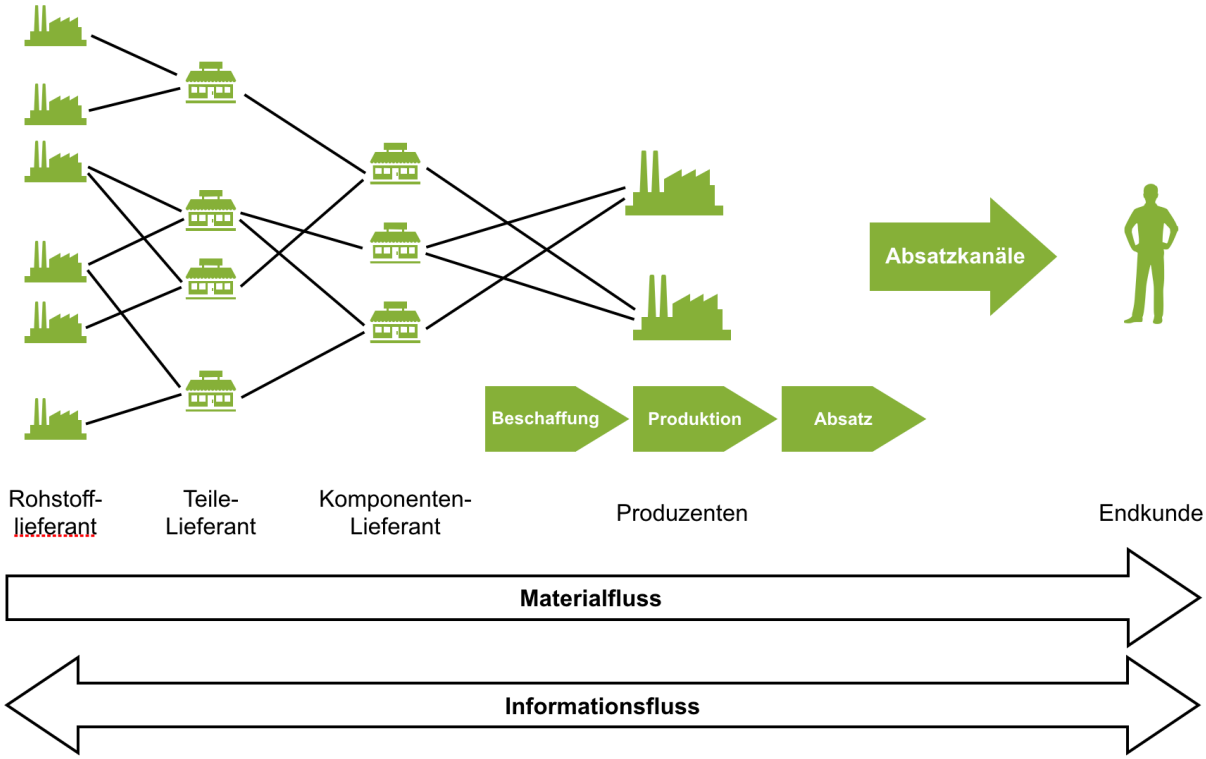

Abbildung 2-1: Aufbau einer Supply Chain [Sch16, S.22]

Hierbei sind die genannten Prozesse Beschaffung, Produktion und Absatz zugleich die Kernprozesse eines produzierenden Unternehmens. In ihnen fallen die Supply-Chain-Daten an, die in verschiedenen Informationssystemen, die nun vorgestellt werden, verarbeitet werden.

2.1.2 Informationssysteme in der Supply Chain

Die vorherige Ausführung zu den Prozessen und Zielen vom SCM verdeutlicht, dass die Supply Chain als Netzwerk von verbundenen, aber zumeist unabhängigen Unternehmen gesehen werden kann. Die Notwendigkeit einer engen, inhaltlichen Kopplung von Planungs- und Steuerungsaufgaben entlang der gesamten SC resultiert hieraus. Es ergeben sich entsprechende Anforderungen an die zum Einsatz kommenden Informationssysteme im Informationsmanagement eines Unternehmens in der SC. Ihr Einsatz unterstützt das wirtschaftliche Handeln der Unternehmen, indem sie den Unternehmen Informationen für Durchführungs-, Analyse-, und Entscheidungsaufgaben bereitstellen. Das informationslogische Prinzip besagt, dass Informationssysteme die richtigen Informationen, in der richtigen Menge, in der richtigen Form und Qualität, zur richtigen Zeit und am richtigen Ort zur Verfügung stellen müssen. [Sch10, S.9f] In diesem Unterkapitel werden drei Arten von Informationssystemen vorgestellt und thematisiert:

- Produktionsplanungs- und steuerungssysteme (PPS-Systeme)
- Enterprise Ressource Planning (ERP-Systeme)
- Advanced Planning Systems (APS)

Um 1960 wurde das Verfahren MRP I (Material Requirements Planning) entwickelt, welches als Softwarelösung für spezifische Standardproblemstellungen, wie der Materialbedarfsplanung eines Unternehmens, dient. Hierbei geht das MRP I von einem vorgegebenen Produktionsprogramm aus und ermöglicht so eine zeitliche Grobplanung der Produktion. Als rechnergestützte Materialbedarfsplanung sorgt es nicht nur dafür, dass Fertigungs- und Beschaffungsaufträge die richtigen Teile und Mengen enthalten, sondern auch die genauen Bedarfszeitpunkte angegeben werden. Nachteilig an diesem Verfahren ist, dass keine Berücksichtigung von Produktionskapazitäten stattfindet und die Funktionsweise nur für eine geringe Variantenvielfalt von Produkten ausgelegt ist. [Kuh02, S.128, Hei11, S. 338ff]
Dieser Ansatz wurde um 1980 durch das MRP II (Manufacturing Resource Planning) um weitere Module erweitert, die unter anderem folgende Funktionen mit einschließen: Termin-, Mengen- und Kapazitätsplanung, Auftragsveranlassung sowie die Auftragsüberwachung. Das Hauptziel ist eine planerische Berücksichtigung einer möglichst hohen Auslastung der Produktionskapazitäten. Beide Verfahren gehören zur Klasse der Produktionsplanungs und -steuerungssysteme (PPS), die große Mengen an aktuellen Planungs- und Steuerungsinformationen bereitstellen und verwalten. Die Ziele der PPS-Systeme sind unter anderem eine termingerechte Belieferung, eine effiziente Lagerverwaltung sowie die Bereitstellung aktueller

Fertigungsinformationen zur optimalen Steuerung der Fertigung. Da die Datengrundlage der Produktionsplanung nicht mit der Datenbasis der Beschaffungs-, Distributions- oder Absatzplanung abgestimmt ist, lässt sich nur eine bedingt optimierte Produktionsplanung erstellen. [Law01, S.55] Ein Nachteil von PPS-Systemen ist die lange Planungsdauer durch sequenzielle Abarbeitung der Planungsschritte, lange Planungszyklen und dadurch häufig veraltete Planungsergebnisse sowie schlechte Kapazitätsausnutzung durch statische Durchlaufzeiten, die die Gesamtdurchlaufzeit des Prozesses erhöhen. Hinzu kommen schwankende Nachfrageentwicklungen und gesättigte Märkte, die immer kürzere Planungszyklen erfordern, wodurch PPS-Systeme an ihre Grenzen stoßen, weil sie nicht flexibel genug sind. In modernen IT-Systemen von Unternehmen sind PPS-Systeme häufig als Module in Enterprise-Resource-Planing-Systeme (ERP Systeme) integriert. [Kuh02, S.127; Sch10, S.180; Sch12, S.195ff]

Seit den 90er Jahren wird eine Integration der einzelnen Planungs- und Steuerungssysteme zu einem logisch zusammenhängenden System angestrebt. Die Einführung von Transaktionssystemen, beispielsweise ERP-Systemen, ermöglicht eine Einbindung nahezu aller innerbetrieblicher Aufgabenbereiche und Prozesse von Unternehmen. Hierbei unterstützen sie die Abwicklung von Geschäftsaktionen in der Forschung und Entwicklung, der Materialwirtschaft, der Produktionsplanung und -steuerung, dem Marketing und dem Vertrieb. Dazu greifen sie auf eine umfangreiche Datenbank zu, die zur Bearbeitung von Geschäftsvorfällen durch Benutzereingaben abgefragt oder geändert werden kann. [Sch10, S.12] Dabei wird insbesondere Wert auf die Datenintegration gelegt, worunter man die Nutzung einer gemeinsamen Datenbasis versteht. Anfallende Daten verschiedener betriebswirtschaftlicher Bereiche, wie z.B. Beschaffung, Produktion, Vertrieb, Rechnungswesen und Personalwirtschaft, werden zentral in einer Datenbank gespeichert. Hierdurch ist eine bereichsübergreifende Nutzung der Daten ohne Mehrfacheingabe oder Pflege von Daten möglich. [Sch10, S. 152f, Kuh02, S.128] Sowohl PPS als auch ERP-Systeme knüpfen schwerpunktmäßig an die unternehmensinternen Prozessen der Produktion und Logistik an. Hinsichtlich des Funktionsumfangs stellen ERP-Systeme eine Erweiterung der klassischen PPS-Systeme dar. Die zeitliche Abfolge und die Erweiterung der Aufgaben- und Funktionsumfänge der einzelnen Informationssysteme in der Supply Chain sind in Abbildung 2.2 dargestellt.

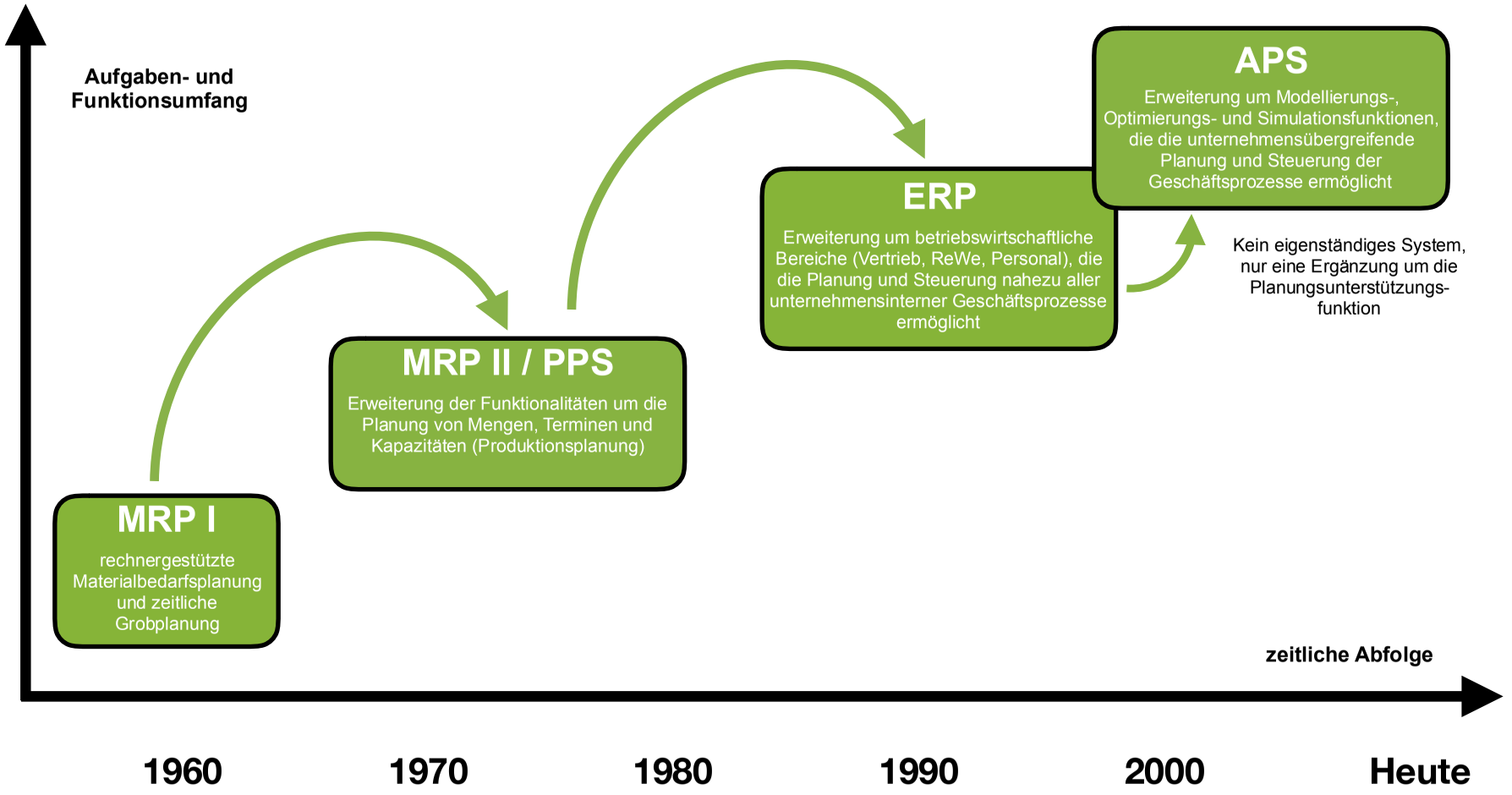

Abbildung 2-2: Zeitliche Abfolge der Aufgaben- und Funktionserweiterungen von Informationssystemen

PPS-Systeme sind für die Unterstützung der unternehmensübergreifenden Planung und Steuerung nur bedingt geeignet, da ihre Aufgaben auf ein einzelnes Unternehmen fokussiert und somit limitiert sind. Demgegenüber konzentrieren sich APS (Advanced Planning and Scheduling) Systeme auf die unternehmens-übergreifende Zusammenarbeit. Dabei kann es als fortgeschrittene Weiter-entwicklung zu bestehenden Planungssystemen gesehen werden und ergänzt ERP-Systeme um Planungsunterstützungsfunktionen, wobei der volle Nutzen nur in Verbindung mit ERP-Systemen erreicht werden kann. APS-Systeme sind modular aufgebaute Softwaresysteme zur integrativen Unterstützung einer unternehmens-übergreifenden, synchronen Planung und Steuerung von Geschäftsprozessen entlang der gesamten Supply Chain. Die Erstellung und Verwendung eines Modells der SC ermöglicht APS-Systemen, komplizierte logistische Strukturen abzubilden sowie den Bedarf und die Kapazitäten simultan zu planen. Integrierte Funktionen zur Modellierung, Optimierung und Simulation ermöglichen hierbei eine ganzheitliche Verfügbarkeitsplanung im System mit den Zielen der unternehmensübergreifende Kostenoptimierungen, der Reduzierung der Lagerbestände und der Verkürzung der Durchlaufzeiten. Dabei gehen APS-Systeme von einer begrenzten Verfügbarkeit von unternehmerischen Ressourcen aus und können mit ihren integrierten Planungs-funktionen als effektive Lösung zur Optimierung unternehmensübergreifender Geschäftsprozesse entlang der Supply Chain gesehen werden. [Hau14, S.117; Kuh02, S.129]

Zusammenfassend ist zu sagen, dass jedes Unternehmen der Supply Chain seine eigenen Transaktionssysteme, wie PPS und ERP-Systeme, für operative Tätigkeiten nutzt. Sie sind nur bedingt flexibel, unterstützen die Planung in einem geringen Umfang und die unternehmensübergreifende Kopplung der Informationssysteme ist nur mit SCM-Software möglich. Wie bereits zuvor beschrieben unterstützen die MRP I und II-Konzepte die Planungsentscheidungen nur in einem begrenztem Umfang. Zwar berechnen sie den Materialbedarf, die Durchlaufzeiten und Kapazitäten, jedoch ist kein Abgleich der Kapazitäten vorgesehen, auch Rückkopplungsmöglichkeiten existieren nicht. [Kuh02, S.133] Beispielsweise erfordern Änderungen durch Eilaufträge einen erneuten Durchlauf aller Planungsstufen, somit reichen solch sequenzielle Planungsansätze nicht aus, um ein effektives Supply Chain Management zu ermöglichen. APS-Systeme übernehmen die Rolle eines unternehmensübergreifenden SCM-Systems, welche jedoch keine eigenständigen Systeme sind und ihr volles Potential nur in Verbindung mit ERP-Systemen nutzen können. Bei der Verwendung der ERP-Daten, wie beispielsweise Lagerbestände oder Produktionskapazitäten der einzelnen Unternehmen der Supply Chain, setzt die

APS eine hohe Datenqualität voraus. Hinzu kommt, dass alle Unternehmen einer Supply Chain im Rahmen von APS auf die gleiche Datenbasis zugreifen, was sich als problematisch erweist, da unzureichende Datengrundlagen und unterschiedliche informationstechnischen Niveaus in den einzelnen Unternehmen der Supply Chain vorherrschen. Nicht nur ungleiche Datenformate und unterschiedliche Grade der Aktualisierung, sondern auch die Inkonsistenz der Daten in einer weltweiten Supply Chain erschweren die Optimierung. [Law01, S.60]

2.1.3 Transaktionsdaten in der Supply Chain

Die Grundlage zur effektiven Entscheidungsfindung in der Logistik bildet das Datenmanagement, welches ein Teilgebiet des Informationsmanagements ist. Hierbei stellen Daten zu unternehmensinternen und -externen Sachverhalten, die in den zuvor genannten Transaktionssystemen entstehen, die Grundlage für Planungs-, Steuerungs- und Kontrollprozesse dar. Das primäre Ziel des Datenmanagements ist es, die wirtschaftliche und zielgerichtete Nutzung von Daten im Unternehmen zu ermöglichen. [Sch16, S.104] Dazu werden die anfallenden Daten in der Supply Chain in diesem Unterkapitel klassifiziert und hinsichtlich Transaktionsdaten abgegrenzt.

Zeichen, Daten, Information und Wissen stehen in hierarchischer Beziehung zu einander. Die unterste Ebene bilden Zeichen, wie Ziffern oder Sonderzeichen. Ihre Menge wird als Zeichenvorrat bezeichnet. Sie sind die einzelnen Bausteine zur Informationsdarstellung. Daten werden aus Zeichen gebildet und folgen einer Struktur (Syntax), ohne dass ein Verwendungszweck aufgezeigt wird. So haben sie für sich allein genommen keine Aussagekraft. Daten kann man erst dann als Information bezeichnen, wenn sie mit einem Kontext versehen sind. Somit ist Information zweckorientiertes Wissen, welches die oberste Stufe der Begriffshierarchie darstellt. Hierbei kann Wissen als zweckorientierte Vernetzung von Information bezeichnet werden kann, mit der Kenntnis über die Zusammenhänge der Informationen. [Sch10, S.7f, Sch09, S.9] Die Beziehungen zwischen den Ebenen der Hierarchie von Wissen, Information, Daten und Zeichen veranschaulicht Abbildung 2-3.

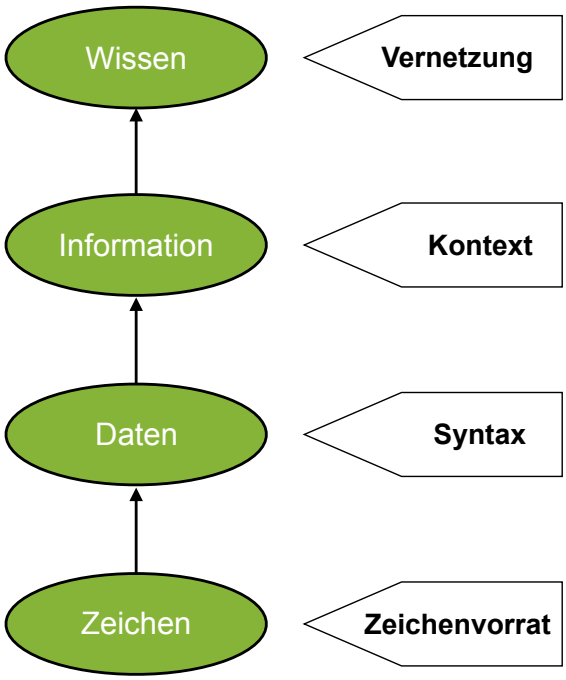

Abbildung 2-3: Beziehungen zwischen der Ebenen der Begriffshierarchie [Sch10, S.7]

Die in einer Supply Chain eingesetzten Informationssysteme sind interne ERP-Systeme eines Unternehmens, die an ein unternehmensübergreifendes APS-System geknüpft sind. Die ERP-Systeme der einzelnen Unternehmen tauschen Informationen zur Planung und Steuerung von Geschäftsprozessen mit dem zentralen, Supply-Chain-übergreifenden APS-System aus. Dabei stimmt das APS-System die Planung der Unternehmen untereinander ab und kann so die ERP-Systeme mit verbesserten Planungsinformationen versorgen. Diese Informationen unterstützen die Unternehmen bei der Integration ihrer Geschäftsprozesse in die SC, indem sie eine synchrone Planung und Steuerung unterstützen. Durch die in Kapitel 2.1.2 (Informationssysteme in der Supply Chain) vorgestellten Module (MRP I, II und PPS-Systeme) können alle relevanten Planungsaufgaben der Supply Chain über die Kopplung von ERP- und APS-Systeme optimiert werden. Alle relevanten Supply-Chain-Daten werden in den genannten Informationssystemen gespeichert und verarbeitet. Es ist wichtig, eine Abgrenzung der über ERP- und APS-Systeme abgewickelten Planungsprozesse zu treffen, um daraus auf relevante SC-Daten schließen zu können, die bei Transaktionsprozessen entstehen. Wie in Kapitel 2.1.1 (Prozesse und Ziele des Supply-Chain-Management) erwähnt, ist das zentrale Ziel einer Supply Chain die optimale, unternehmensübergreifende Gestaltung der Gesamtprozesse. Hierzu gehören in aller erster Linie die Planungsprozesse Beschaffungsplanung, Produktionsplanung und Absatzplanung. Hinzu kommt, dass unter den diversen Wettbewerbsfaktoren vor allem Wert auf die zwischenbetriebliche

Transportplanung gelegt wird. Nur die zuverlässige Versorgung der Unternehmen einer SC mit Roh- und Hilfsstoffen, Teilen und Produkten kann die Anwendbarkeit von optimierten Produktions- und Steuerungskonzepten realisieren. Somit sollen die in dieser Arbeit näher betrachteten Supply-Chain-Daten aus diesen Planungsprozessen stammen.
Im Folgenden werden allgemeine Datenarten vorgestellt. Es lassen sich vier grundlegende Datenarten nach dem Verwendungszweck und der Veränderbarkeit von Daten unterscheiden. Dabei zählt man *Stamm-* und *Bewegungsdaten* zu den zustandsorientierten Daten und *Bewegungs-* und *Änderungsdaten* gehören zu den abwicklungsorienttierten Daten.

- *Stammdaten* (engl. master data) sind die Kernobjekte eines Unternehmens, bestehend aus Informationen beispielsweise zu Kunden, Mitarbeitern, Lieferanten, Produkten oder Anlagegütern. Sie bleiben im Volumen über den Zeitablauf relativ konstant und weisen eine vergleichsweise geringe Änderungshäufigkeit auf.
- *Bestandsdaten* (engl. inventory data) beschreiben die betrieblichen Mengen- und Wertestrukturen, wie z.B. Lagerbestände und Produktionskapazitäten. Durch das kontinuierliche Betriebsgeschehen unterliegen Bestandsdaten einer systematischen Veränderungen, sodass sie im Zeitablauf eine hohe Änderungshäufigkeit aufweisen, wobei das Volumen ebenfalls konstant bleibt.
- *Bewegungsdaten* oder auch Transaktionsdaten (engl. transaction data) beschreiben betriebswirtschaftliche Vorgänge, wie Aufträge, Rechnungen oder Lager-bewegungen. Da Transaktionsdaten im Rahmen einer Geschäftstätigkeit immer wieder neu entstehen, verändern sie sich im Zeitablauf häufig und nehmen an Volumen zu. Hinzu kommt der wesentliche Faktor, dass Transaktionsdaten Bestandsdaten verändern können.
- *Änderungsdaten* (engl. change data) lösen Änderungen an Stammdaten aus. Dies geschieht beispielsweise durch die Änderung einer Kundenadresse oder Einführung eines neuen Produktes. Sie wachsen im Zeitverlauf, jedoch ist ihre Änderungs-häufigkeit und Volumensteigerung deutlich geringer als bei Bewegungsdaten. [Sch09, S.19f]

Die Typologie der vier grundlegenden Datenarten werden übersichtlich in Abbildung 2-4 dargestellt.

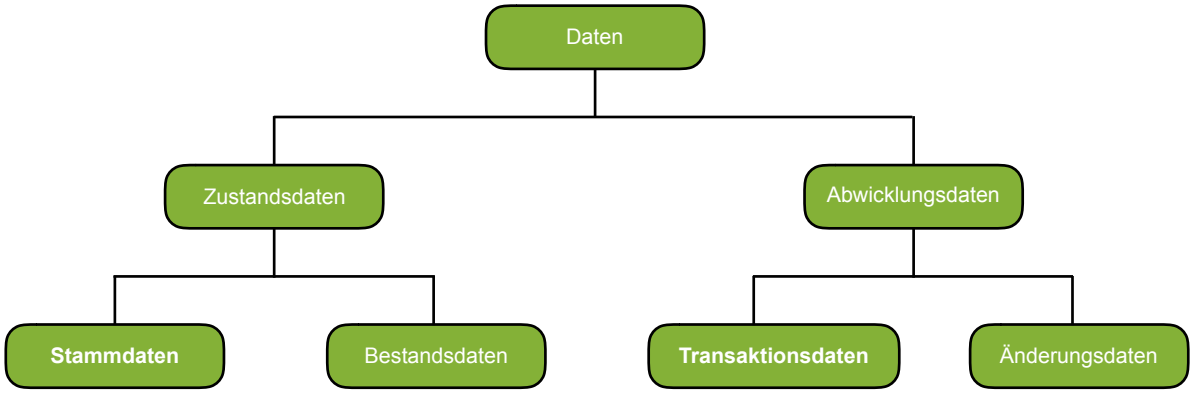

Abbildung 2-4: Typologie von Datenarten [Sch10, S.20]

Ziegler hat in seiner Arbeit diverse Quellen zu Kategorisierungen von Supply-Chain-Daten ausgewertet, eine umfassende Literaturrecherche zu gängigen Datenkategorisierungsmodellen durchgeführt und ein eigenes Kategorisierungsmodell konzipiert. In dieser Arbeit wird auf Zieglers Modell Bezug genommen, um im weiteren Verlauf Speicherungs- und Aufbereitungsmöglichkeiten dieser SC-Daten durch NoSQL-Datenbanken aufzuführen.

Hierzu wird zunächst eine Erweiterung der oben vorgestellten Typologie von Datenarten vorgenommen, indem weitere Unterkategorien eingeführt werden.
So lassen sich etwa Transaktionsdaten in zwei weitere Kategorien, nämlich *Transkaktionsaktivitäts*- sowie *Transaktionskontrolldaten* aufteilen.

Transaktionsaktivitätsdaten beschreiben zeitpunktbezogene Geschäftsaktivitäten eines Unternehmens, wie beispielsweise einzelne Zahlungen und Lieferungen. Zu ihnen gehören Mengeneinheiten der einzelnen Produkte, Preise oder auch verschiedene Transportmodi der Lieferungen.
Transaktionskontrolldaten sind die Protokolldaten der Transaktionsaktivitätsdaten, die den gesamten Transaktionsprozess nachvollziehbar werden lassen, indem sie Zeitstempel und Liefertermine beinhalten. [Zie15, S.11ff]

Die zustandsorientierten Stammdaten lassen sich in drei weitere Unterkategorien aufteilen: Referenz-, Unternehmensstruktur- und Transaktionsstrukturdaten.

Referenzdaten sind standardisierten Kodierungen zur Kategorisierung anderer Daten. Sie sind statisch, weisen eine geringe Änderungshäufigkeit auf und stammen aus externen, offiziellen Quellen, wie z.B. Länder oder Flughafenabkürzungen.
Unternehmensstrukturdaten bilden den strukturellen Aufbau eines Unternehmens ab, beispielsweise durch Organisationseinheiten und Kostenstellen.

Transaktionsstrukturdaten beschreiben alle Objekte, die an Transaktionen im Unternehmen beteiligt sind. Hierzu gehören Daten zu Kunden, wie Zahlungsbedingungen, Daten zu Lieferanten, wie Lieferadressen oder Lieferantennummer, aber auch Daten zu Produkten, wie die Produktkennung. [Zie15, S.29ff]

Hiermit sind alle relevanten SC-Daten hinsichtlich der diversen Transaktionsdaten abgegrenzt und werden als Datenbasis in dieser Arbeit zur exemplarischen Darstellung eines beispielhaften Transaktionsdatensatzes verwendet.
Die nachfolgende Abbildung 2-5 dient zur besseren Übersicht der zuvor beschriebenen einzelnen Kategorien von Supply-Chain-Daten und erweitert die Abbildung 2-4 Typologie von Datenarten um weitere Datenarten.

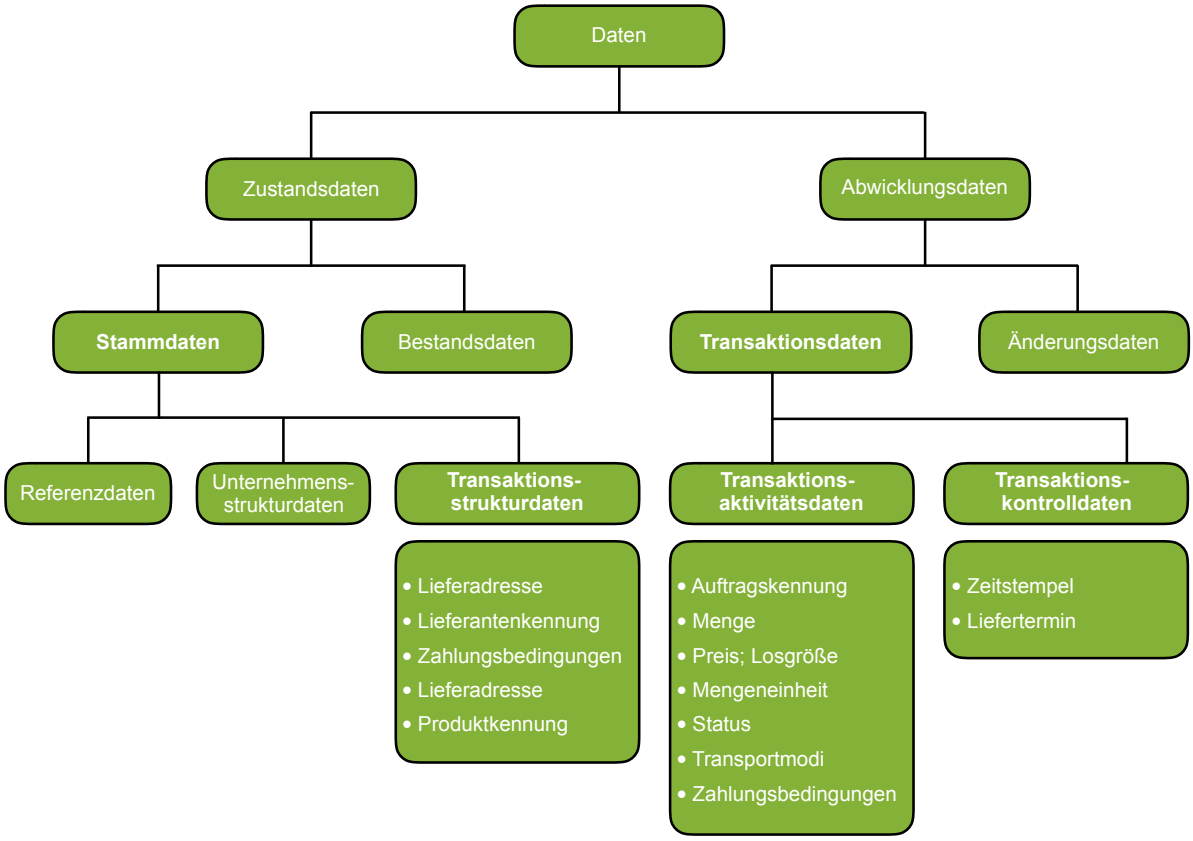

Abbildung 2-5: Übersicht der Kategorien von Supply-Chain-Daten [Zie15, S.31ff]

Die Prozesse und Ziele des Supply Chain Management (Kapitel 2.1.1) sind definiert, Informationssysteme in der Supply Chain (Kapitel 2.1.2) aufgeführt und die darin verarbeiteten Transaktionsdaten in der SC (Kapitel 2.1.3) abgegrenzt.
Jetzt gilt es, die Datenmodellierung und relationale Datenbanksysteme in Unternehmen vorzustellen und ihre Grenzen aufzuzeigen. Schließlich werden

NoSQL-Datenbanken als neuartige und effektive informationstechnische Lösung zur Speicherung und Aufbereitung umfangreicher, heterogener Transaktionsdaten vorgestellt und in Kapitel 3 analysiert.

2.2 Datenmodellierung und relationale Datenbanken

Das Ziel dieses Kapitels ist, einen grundlegenden Überblick über die Möglichkeiten zur Speicherung und Aufbereitung von Supply-Chain-Daten zu geben. Dabei werden zuerst allgemein die *Datenmodellierung im Unternehmen* aufgezeigt und danach eine *Einführung in relationale Datenbanken* gegeben.

2.2.1 Datenmodellierung im Unternehmen

Wie bereits in Kapitel 2.1.3 (Transaktionsdaten in der Supply Chain) beschrieben, sind Daten an sich nicht wertschöpfend. Erst durch die Interpretation und Nutzung der Daten durch die vorgestellten Informationssysteme, die die Informationen in Entscheidungen und Handlungen umsetzen, kommt es zur Wertschöpfung. Wie jegliche andere betriebliche Ressource benötigen Daten einen Managementprozess, der sie als Kernprozess im Unternehmen steuert. Das Datenmanagement, mit dem Fokus auf Datenmodellierung, -administration, -technik, -sicherheit und -konsistenz, hat als Ziel, die Bereitstellung und Nutzung der Daten im Unternehmen zu ermöglichen. Hierzu werden im Unternehmen Datenmanagementstrategien definiert, die festlegen, welche Daten für welche Systeme und Aufgaben auf welche Art und Weise zur Verfügung gestellt werden. Das Stammdatenmanagement im Unternehmen ist von hoher Bedeutung. Hinzu kommt die Festlegung der organisatorischen Verantwortlichkeit für Datenpflege und Erfassung, sowie die Bereitstellung der Daten in vordefinierter und abgestimmter Form. Beispielsweise werden Kundenstammdaten im Unternehmen oft in verteilten, heterogenen Systemen gehalten, was zur Datenredundanz führt. Denn diese Mehrfachhaltung von Stammdaten, sowohl in den Bereichen Finanzen, als auch in den Bereichen Logistik führt zu Problemen in Bezug auf Konsistenz und Qualität der Stammdaten. Konsistenz bezeichnet die Integrität der Daten, also die Korrektheit der in einer Datenbank oder verteiltem System gespeicherten Daten. Das Stammdaten-management hat als oberstes Ziel, die in den komplexen informationstechnischen Systemen eines Unternehmens vorherrschenden Stammdaten in einen

redundanzfreien, harmonisierten und an zentralen Stellen bereitgestellten und verwalteten Datenbestand zu überführen. [Krc15, S.42f; Sch10, S.43]
Im Folgenden werden hierauf passende Methoden der Modellierung, im speziellem das Referenzmodell und das grundlegende Entity-Relationship-Modell (ERM) vorgestellt.
Datenmodelle sind Voraussetzung für die Realisierung der bereits genannten Ansätze des Datenmanagements. Im Speziellen ist ein Referenzmodell ein Modell mit allgemeingültigen Charakter, der für einen Wirtschaftszweig eines Unternehmens erstellt ist. Sie dienen als Gestaltungsempfehlungen, und durch das Kriterium der Allgemeingültigkeit können sie für eine Klasse unternehmensspezifischer Modelle zum Einsatz kommen. Zusätzlich sind Referenzmodelle dabei inhaltlich konsistent und flexibel. So sind Änderungen am Modell ohne großen Aufwand realisierbar und die Robustheit, bedingt durch die Änderungen in der realen Welt, gewährleistet werden. Zusätzlich muss durch die Wahl des passenden Abstraktionsgrads das Kriterium der Allgemeingültigkeit erfüllt sein. [Krc15, S.44; Sch10, S.105] Die Vorteile der Nutzung von Referenzmodellen im Unternehmen sind Kostenreduktion sowie Zeitersparnisse durch verkürzte Organisations- und Implementierungszeiten. Zusätzlich kann dadurch die Datenintegrität sichergestellt werden, sodass unternehmensinterne Abläufe optimiert, Schwachstellen aufgedeckt und Prozesse dokumentiert werden. Als nachteilig kann die Reduzierung von strategischen Wettbewerbsvorteilen des Unternehmens durch die Standardisierung beim Einsatz solcher Modelle gesehen werden Denn hierdurch wird möglicherweise nicht genügend auf die Kernkompetenz des Unternehmens eingegangen. [Krc15, S.45; Sch09, S.25; Sch10, S.106] Wie bereits beschrieben, dienen Modellierungsansätze im Unternehmen zur Abbildung eines Systems von Objekten für einen bestimmten Zweck. So werden Ausschnitte aus der realen Welt durch ein Ist-Modell als Abbild der realen Welt dargestellt. Verschiedene Funktionsmodellierungen können für Informationssysteme erstellt werden, wobei sich im Bereich der Datenmodellierung das Entity-Relationship-Modell (ERM) mit den von Chen 1976 eingeführten Begriffen als Standard durchgesetzt hat. Hierdurch kann beispielsweise das Verhältnis der unternehmensrelevanten Daten zueinander dargestellt werden. Ein klarer Vorteil dieses Modells ist die exakte Definition und übersichtliche Darstellung der Objekte und Beziehungen in Form von drei Grundelementen:

- Ein *Entity* ist ein reales oder abstraktes Objekt, das für ein Unternehmen von Interesse ist. Dabei modellieren die Entitäten eindeutig, identifizierbar den Anwendungsbereich der realen Welt. Typische Beispiele hierfür sind Produkte, Kunden, Lieferanten oder Aufträge. Entitytypen sind zu einer Menge

zusammengefasste gleichartige Entitäten. Entitätstypen werden im ER-Diagramm als Rechtecke dargestellt.

- *Attribute* sind Eigenschaften von Entities, welche durch Werte geprägt sind. Dazu zählen z.B. Kundennummer, Name und Anschrift. Dabei liegen die Attribute in einem klar definiertem Wertebereich. Attributstypen werden als Rechtecke mit abgerundeten Ecken im ER-Diagramm dargestellt.
- *Beziehungen* sind logische Verknüpfungen von zwei oder mehreren Entitätstypen, somit stehen Entitäten mit Hilfe von Relationen zueinander in Beziehung. Sie werden dabei anhand ihrer Komplexität in 1:1, 1:n bzw. n:1 oder n:m Beziehungen unterschieden. Relationstypen werden als Raute im ER-Diagramm dargestellt. [Haa18, S.101; Sch10, S.100]

Zur Verdeutlichung soll ein Beispiel eines ERMs eines Unternehmens dienen, welches die internen Beziehungen zwischen Sachbearbeiter, Abteilungsleiter und Artikeln, sowie die externen Beziehungen zwischen Kunden, Lieferanten und Lägern als Entitäten verdeutlicht (Abbildung 2-6 ERM eines Unternehmens). Den Entitäten werden verschiedene Attributstypen zugewiesen und ihre Relationen zueinander beschrieben.

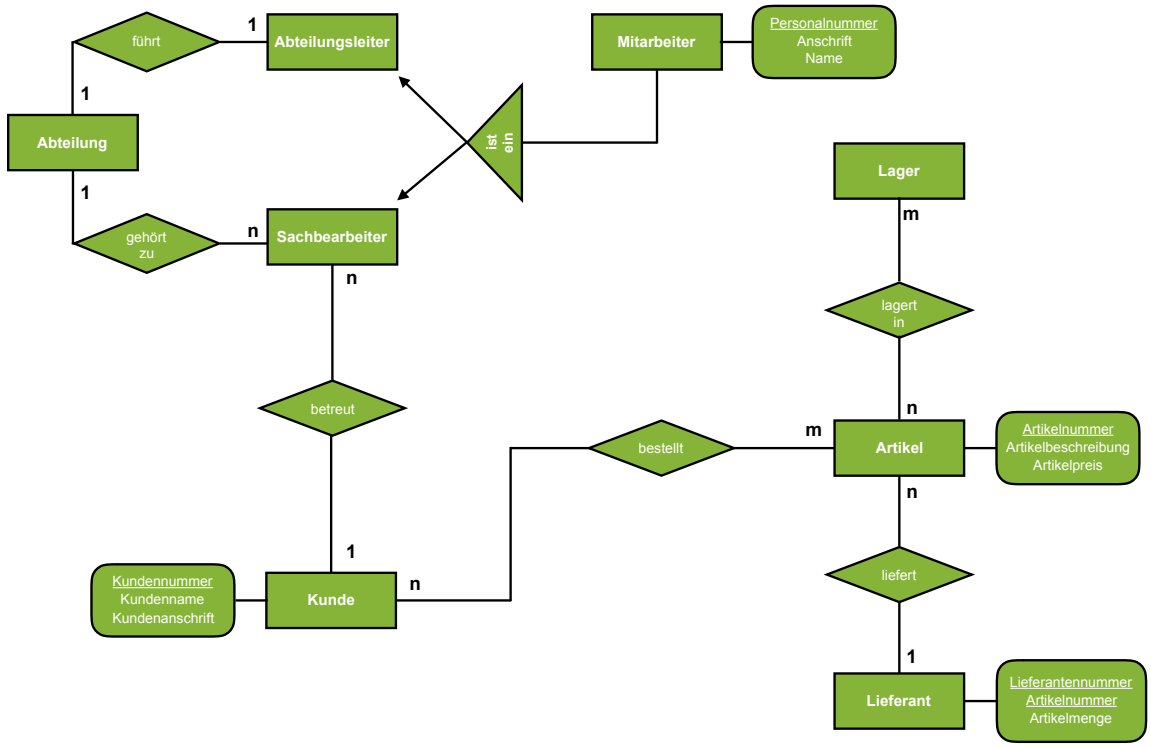

Abbildung 2-6: Entity-Relationship-Model eines Unternehmens [Krc15, S.47]

In Abbildung 2-6 ist jedem Mitarbeiter des Unternehmens genau ein Attributstyp *Personalnummer* zugeordnet. Dieser identifiziert ihn eindeutig. Bei solchen Attributstypen, die Entitätstypen eindeutig identifizierbar machen, spricht man von Primärschlüsseln. Sie werden in ER-Diagrammen unterstrichen dargestellt. Der Attributstyp *Anschrift,* sowie *Name* des Mitarbeiters sind nicht eindeutig identifizierend, da mehrere Mitarbeiter den gleichen Namen oder Anschrift haben können. So ist der Attributstyp *Name* und *Anschrift* kein Primärschlüssel und wird im Gegensatz zur *Personalnummer* nicht unterstrichen dargestellt.
Ein Mitarbeiter des Unternehmens, eindeutig identifizierbar über seine *Personalnummer*, kann entweder *Abteilungsleiter* oder *Sachbearbeiter* sein. Durch die Generalisierung (übergeordnete Menge) und Spezialisierung (untergeordnete Menge) ist eine Modellierung der Gegebenheit, dass ein Mitarbeiter entweder Abteilungsleiter oder Sachbearbeiter sein kann, durch die ist-ein-Generalisierung im ER-Diagramm möglich. Dabei wird über die Relationstypen ‚*führt*' und ‚*gehört zu*' verdeutlicht, wie die Entitäten Abteilungsleiter und Sachbearbeiter in Beziehung mit dem Entitätstyp *Abteilung* stehen.
Ein Abteilungsleiter führt eine Abteilung, welches eine 1:1-Beziehung begründet. Mehrere Sachbearbeiter gehören zu einer Abteilung. Hier liegt also eine n:1-Beziehung vor. Durch den Relationstypen ‚*betreut*' in Verbindung mit einer n:1-Beziehung wird verdeutlicht, dass mehrere Sachbearbeiter einen Kunden betreuen. Dieser ist durch seinen Attributstypen *Kundennummer* eindeutig identifizierbar, welcher im Gegensatz zu den weiteren Attributstypen *Kundenname* und - anschrift als Primärschlüssel gesehen wird. Mehrere Kunden können mehrere Artikel bestellen und stehen somit über den Relationstypen ‚*bestellt*' in einer n:m-Beziehung zum Entitätstypen *Artikel*.
Der Entitätstyp *Artikel* lässt sich über seinen Attributstypen *Artikelnummer* eindeutig identifizieren, somit ist letzterer wieder ein Primärschlüssel im ER-Diagramm. *Artikelbeschreibung* und -*preis* sind weitere Attributstypen, jedoch keine mit denen sich die Entität eindeutig identifizieren lässt. Verschiedene Artikel werden in mehreren *Lagern* aufbewahrt. Dies repräsentiert der Relationstyp *lagert in* über eine n:m-Beziehung. Zusätzlich liefert ein *Lieferant* mehrere Artikel, dies verdeutlicht der Relationstyp ‚*liefert*' über eine 1:n-Beziehung.
Dabei besitzt der Entitätstyp *Lieferant* zum einen den Attributstypen *Lieferantennummer*, der ihn eindeutig identifiziert und somit ein Primärschlüssel ist und zum anderen den Attributstypen *Artikelnummer* welcher auch ein Primärschlüssel ist. Ein weiterer Attributstyp der Entität Lieferant ist die *Artikelmenge*, welche keinen Primärschlüssel darstellt.

Mit dieser grundlegenden, aber auch anderen Modellierungsmethoden können komplexe unternehmensweite Datenmodelle erstellt werden, die nun durch die Verwendung eines geeigneten Datenbanksystems technisch Umgesetzt werden müssen.

2.2.2 Einführung in relationale Datenbanken

Zu Anfang sind Datenbanksysteme (DBS) zu definieren. Sie bestehen aus einer Menge von Daten, die eigentliche Datenbasis, sowie einer Menge von Programmen, die als Datenbankverwaltungssystem bezeichnet werden. Die Datenbasis enthält Informationseinheiten, die miteinander in Beziehung stehen und zur Steuerung und Kontrolle einzelner Aufgabenbereiche notwendig sind. Das Datenverwaltungssystem erlaubt die Administration, Nutzung und Modifizierung der Datenbasis. Hierbei wird vor allem Wert auf die Datenintegrität und -konsistenz gelegt. Zusätzlich muss die Sicherheit der Daten sichergestellt werden, wobei zwischen Anwendungssicherheit (engl. safety) und dem Schutz vor beabsichtigten Angriffen (engl. security) auf das Datenbanksystem und dessen Datenbasis unterschieden wird. Die Anwendungssicherheit eines DBS kann durch Maßnahmen, wie redundanter Datenhaltung, das Erstellen von Sicherheitskopien und der Nutzung von zertifizierter Software sichergestellt werden. Durch Zugriffskontrollen, Identifikationen, Authentifizierungen und Autorisierungen lässt sich der Schutz vor beabsichtigten Angriffen auf die Datenbasis sicherstellen. [Krc15, S.48f] Um im späteren Verlauf dieser Arbeit eine NoSQL-Datenbank für die bereits eingeführten Supply-Chain-Daten zu empfehlen, ist es wichtig die grundlegende Funktionsweise von Datenverwaltungssystemen innerhalb einer Datenbank und die Architektur von Datenbanksystemen zu verstehen. Ein Datenmodell, welches die Infrastruktur für die Modellierung der realen Welt zur Verfügung stellt (Kapitel 2.2.1), bildet die Basis für das Datenbankmanagementsystem. Das Datenmodell legt dabei Modellierungskonstrukte der realen Welt in eine computerbasiertes Informationsabbild, welches zum einen die Möglichkeit beinhaltet, Datenobjekte zu beschreiben und zum anderen anwendbare Operatoren und deren Wirkung festlegt. Das Datenmodell kann somit analog zu einer Programmiersprache gesehen werden, da es Strukturen und Operatoren festlegt, die man zur Modellierung einer bestimmten Anwendung ausnutzen kann. Beispielsweise legen Programmiersprachen Sprachkonstrukte fest, mit denen man Anwendungsprogramme realisieren kann. Selbiges Prinzip findet bei Datenmodellen eines DBMS ihre Anwendung. Das Datenmodell eines DBMS besteht aus zwei Teilsprachen, zum einen der Datendefinitionssprache (engl. Data Definition

Language, DDL) und zum anderen der Datenmanipulationssprache (engl. Data Manipulation Language, DML), wobei die DDL die Struktur der abzusichernden Datenobjekte beschreibt. Das Datenbankschema dient dabei als Strukturbeschreibung aller Datenobjekte, die die DDL erzeugt. Die DML besteht zum einen aus der Anfragesprache (engl. Query Language) und zum anderen aus der Datenmanipulationssprache, die zur Änderung von gespeicherten Datenobjekten, sowie zum Einfügen und Löschen von gespeicherten Daten verwendet wird. [Kem11, S.23] Als standardisierte Anfragesprache hat sich zur Einführung von relationalen Datenbanksystemen Anfang der 1970er Jahre SEQUEL (engl. Structured English Query Language) durchgesetzt, die später in SQL (engl. Structured Query Language) umbenannt wurde. Von der hohen Popularität von relationalen DBS ging die Notwendigkeit einer Normierung und Standardisierung der Sprache SQL aus, die 1986 von der ANSI-Kommission (American National Standards Institute) verabschiedet wurde. [Kem11, S.111] Wie bereits in Kapitel 2.1 beschrieben, hat das Datenmanagement als zentrale Funktion die Ressource Information im Unternehmen zu verwalten. Hier ergeben sich eine Vielzahl von Anforderungen an die Datenorganisation, die bei dem Entwurf von Datenbanken zu berücksichtigen sind. Zum einen sind das kurze Zugriffs- und Übertragungszeiten, um schnelle Verarbeitungsmöglichkeiten zu gewährleisten. Zum anderen eine möglichst geringe Redundanz der Daten im Unternehmen erforderlich, d.h. ein mehrfaches Aufkommen der selben Daten im Unternehmen soweit wie möglich zu unterbinden. Zusätzlich gilt es, die logische Integration der Daten, z.B. bei physisch verteilten Datenbeständen durch globale Produktionsstandorte, zusammenhängend verwalten zu können. Um die physische und logische Datenunabhängigkeit sicherzustellen, muss eine Trennung zwischen Datenorganisation und Anwendung statt finden. Es wird eine anwendungsneutrale Speicherung der Daten gefordert. Als letzte Anforderungen an die Datenorganisation können zum einen Datenintegrität, die durch Datenkonsistenz, Datensicherheit und Datenschutz gewährleistet wird, und zum anderen die Wirtschaftlichkeit gesehen werden. Es wird eine Vollständigkeit, Korrektheit und Widerspruchsfreiheit der Daten, sowie die exakte und aktuelle Wiedergabe der realen Bedingungen gefordert. Ergänzend zu diesen Anforderungen an die Datenorganisation werden in der Literatur weitere Anforderungen an Datenbanksysteme gestellt: Eine einfache Datenhandhabung soll durch eine leicht zu erlernende Benutzersprache zur Benutzerfreundlichkeit beitragen. Der Mehrfachzugriff soll jedem Berechtigten im Mehrbenutzerbetrieb den gleichzeitigen Zugriff auf die gespeicherten Daten mit anderen ermöglichen. Die Leistung des DBS soll durch kurze Antwortzeiten für die Abfrage und Verarbeitung der Daten sowie für

Änderungen und Ergänzungen des Datenbestandes erlauben. [Krc15, S.46; Sch16, S.108ff]
Um vorrangig den Anforderungen der minimalen Datenredundanz und -konsistenz gerecht zu werden, wird bei dem Entwurf der Datenbankanwendung zwischen drei Abstraktionsebenen unterschieden:

- Konzeptuelle Ebene,
- Implementationsebene,
- Physische Ebene.

Die konzeptuelle Ebene ist allgemein gehalten und beschreibt die Gesamtsicht der logischen Dateneinheiten unabhängig von Gesichtspunkten der Datenverarbeitung. Da diese Ebene unabhängig von dem zum Einsatz kommenden DBS modelliert wird, soll sie auch nur die Anwendersicht modellieren. Somit beschreibt sie die nur für eine Benutzergruppe interessante Datenmenge. Im konzeptuellen Schema wird vorwiegend das Entity-Relationship-Modell eingesetzt, welches zuvor vorgestellt wurden. Mittels einer geeigneten Datendefinitionssprache (DDL) wird dieses Schema spezifiziert.
Auf der Implementierungsebene wird die Datenbankanwendung in dem Datenmodell des zum Einsatz kommenden DBS modelliert. Das relationale Datenmodell beinhaltet Relationen, Tupel und Attribute.
Die physische und somit niedrigste Abstraktionsebene legt fest, in welcher Form die logisch beschriebenen Daten im Speicher abgelegt werden und welche Zugriffs-möglichkeiten bestehen sollen. Primär geht es hierbei darum, die Leistungsfähigkeit der Datenbank-anwendung zu erhöhen, wobei tiefergehende Kenntnisse des eingesetzten DBS und Hardware nötig sind. [Kem11, S.31; Sch10, S.46f]
Bei der logischen Datenorganisation, die sich mit der konzeptuellen Ebene befasst, geht es um die formale Beschreibung der Datenobjekte, ihrer Eigenschaften und der gegenseitigen Beziehung untereinander. Von hoher Bedeutung ist das um 1970 entwickelte relationale Datenbankmodell von Codd, welches das relationale Datenmodell in einer tabellarischen Dartstellungsform gebraucht. Das Grundmodell ist eine Tabelle (Relation) mit einer festen Anzahl von Spalten und einer beliebigen Anzahl von Zeilen, in der sämtliche Daten dargestellt werden. Die Zeilen entsprechen dabei den Datenobjekten. [Sch10, S.49] Die Daten die in den Tabellen gespeichert sind, werden durch Operatoren ausschließlich mengenorientiert verknüpft und verarbeitet. Unter Normalisierung versteht man die Umstrukturierung in der zusammenhängenden Daten, wie beispielsweise Adressen, in elementarer Beziehungen, wie beispielsweise Name, Straße und Nummer aufgelöst werden. So werden in einer Tabelle keine Wiederholungsgruppen auftreten. Dies wird auch als 1.

Normalform bezeichnet. Wenn z.B. 100 Bestellungen von einem Kunden abgearbeitet werden müssen, so könnte die Tabelle neben den Daten der Bestellung auch 100-mal die Adresse des Kunden beinhalten. Es entsteht hohe Redundanz durch die Wiederholungsgruppe der Adressen. Hierbei belastet die Datenredundanz unnötig den Speicher und bietet die Basis für häufige Fehlerquellen, da diese Daten nur mit erhöhtem Aufwand konsistent gehalten werden können. Aus diesem Grund wird eine neue Tabelle mit Kundennummern und dazugehörigen Kundenadressen angelegt. So beinhalten bei der Bestellung die Daten nur Informationen zu den Artikeln und lediglich die Kundennummer. Nun gilt es, die beiden Tabellen mittels sogenannter *Joints* zu verknüpfen. Die Verknüpfung wird dabei über die Kundennummern realisiert. Sie ein Schlüssel bzw. Attribut, welches den Eintrag eindeutig identifiziert. [Sch10, S. 49; Kem11, S.73] Bevor die den Datenmodellen zugrunde liegende Struktur der Daten in eine formale Beschreibungssprache transformiert wird, werden diese durch eine generelle Entwurfsprache beschrieben. Wie bereits in Kapitel 2.2.1 (Datenmodellierung im Unternehmen) erwähnt, geschieht dies durch das gängigste Beschreibungsverfahren Entity-Relationship-Modell. So

Bestellung			
Artikelnummer	Artikelbezeichnung	Bestellmenge [stk.]	Kundennummer
12345	O Ring	7	87912
67891	Spaltrohr	7	87912
6183	Gehäuse	14	7231
…	….	…	

Tabelle 2-1: Relationsmodell Bestellung

Lager		
Artikelnummer	Lagerort	Lagermenge
12345	23a	235
67891	23b	764
6183	5	98
…	….	…

Tabelle 2-2: Relationsmodell Lager

Kunde	
Kundenummer	Anschrift
87912	Gevelsberg 3, Germany
7231	Hack Way 1, California US
7421	Rue 32, Paris France
…	…

Tabelle 2-3: Relationsmodell Kunde

können, wie in der Abbildung 2-6 (Entity-Relationship-Modell eines Unternehmens),

Entitäten bestimmte Attributstypen in einer Tabelle zugeordnet werden. Dabei werden Relationen als Tabellen bezeichnet.

In der Tabelle 2-1 werden die Spalten als Attribute bezeichnet. Innerhalb einer Relation müssen Attribute eindeutig benannt sein. Die Zeilen der Tabelle *Bestellung* entsprechen den Tupeln der Relation. In diesem Beispiel enthält die Relation (Tabelle) vierstellige Tupel, deren Attributswerte aus dem Wertebereich integer und string stammen. Das Relationsschema wird dabei nach folgendem Muster spezifiziert:
Bestellung : { [Artikelnummer : integer , Artikelbezeichnung : string , Bestellmenge : integer, Kundennummer : integer] }

Die Informationen in den eckigen Klammern […] geben dabei den Aufbau der Tupel wieder, besagen also, welche Attribute vorhanden sind und welchen Typ (Wertebereich) diese haben. Die geschweiften Klammern { … } drücken aus, dass es sich bei dieser Relationsausprägung um eine Menge von Tupeln handelt. Der Unterstrich am Attribut *Artikelnummer* und *Kundennummer* symbolisiert dabei, dass es sich um einen Schlüssel handelt, der Verknüpfungen zu anderen Tabelle herstellt. Hier enthalten beispielsweise die Tabelle 2-2 relevante Informationen zum Lagerort und der gelagerten Menge und die Tabelle 2-3 Informationen zum Kunden, wie seine Anschrift.
Die in Tabelle 2-2 dargestellten Lager sind über das Attribut *Artikelnummer* - hier ein Schlüssel - mit Tabelle 2-1 verknüpft. Auch die Tabelle 2-3 ist über das Attribut Kundennummer mit Tabelle 2-1 verknüpft. Solche Verknüpfungen dienen zur Vermeidung von Datenredundanz und der Belastung von Speichermedien. Das relationale Datenbanksystem ist durch folgende Eigenschaften charakterisiert:

- *Modell:* Das Datenbankmodell unterliegt dem Relationsmodell. Alle Daten und Datenbeziehungen werden in Form von Tabellen ausgedrückt.
- *Schema:* Definitionen der Tabelle und Attribute werden in einem relationalen Datenbankschema abgelegt. Es beinhaltet auch Definitionen von Schlüsseln zur Identifikation und Regelungen zur Gewährung der Datenintegrität.
- *Sprache*: SQL.
- *Architektur:* große Datenunabhängigkeit durch Trennung von Daten und Anwendungsprogramm.
- *Mehrbenutzerbetrieb:* Das DBS unterstützt die gleichzeitige Abfrage und Bearbeitung der selben Datenbank durch mehrere Benutzer. Das relationale

DBS sorgt dafür, dass sich parallel ablaufende Transaktionen auf einer DB nicht behindern und die Korrektheit der Daten gewahrt wird.

- *Konsistenzgewährung:* Hilfsmittel zur Gewährleistung der Datenintegrität werden durch das DBS bereitgestellt. Durch ACID laufen Transaktionen stets atomar, also erst gültig nach vollständigem Abschluss der Transaktion, konsistent, also die Hinterlassung eines konsistenten Zustand der DB nach Beendigung der Transaktion, isoliert, also keine Beeinflussung durch nebenläufige Transaktionen, und dauerhaft, also die Garantie zur dauerhaften Speicherung der Daten, ab.
- *Datensicherheit und - Schutz:* Mechanismen zum Schutz vor Zerstörung der Daten, vor Verlust oder vor unbefugtem Zugriff werden bereit gestellt. [Mei16, S.10f]

Die Grenzen von relationalen Datenbanken liegen in der Verarbeitung von umfangreichen Datenmengen im Peta- bis Zetabyte-Bereich, sogenannten Big Data Anwendungen. [Mei16, S.11] Auch bei der Überprüfung der Datenintegrität stoßen relationale DBS aufgrund der umfangreichen Datenmengen an ihre Grenzen. Es stehen stark konsistenzorientierte Verarbeitungskomponenten einer Verarbeitung großer Datenmengen oft im Weg, gerade bei Anwendungen, bei denen die Leistung und nicht die Konsistenz im Vordergrund steht. [Mei16, S.221f] Ein grundlegendes Merkmal relationaler Datenbanktechnologien ist eine Transaktionsverarbeitung, die durch Atomarität (A), Konsistenz (C), Isolation (I) und Dauerhaftigkeit (D) gekennzeichnet ist. Somit gewährleistet die Konsistenzsicherung durch ACID bei relationalen Datenbanken stark konsistente Datenbestände, die jedoch nur in bestimmten Anwendungsfeldern erforderlich sind. Beispielsweise müssen Datenbanken im Finanzwesen und auf Kapitalmärkten immer korrekte Daten liefern, aber in Geschäftsanwendung ist dies nicht immer zwingend erforderlich. Bei dem Verzicht auf strikte Konsistenz kann die Skalierbarkeit und Verfügbarkeit der Datenbank erheblich verbessert werden. [Pok13, S.71f] Um den beschriebenen Anforderung der Verarbeitung umfangreicher Datenmengen, Skalierbarkeit und hoher Verfügbarkeit der Datenbankanwendung gerecht zu werden, sollen relationale Datenbanktechnologien um NoSQL-Datenbanktechnologien ergänzt werden. So können beispielsweise Webdienste immer und überall angeboten werden. Im folgenden Kapitel werden NoSQL-Datenbanken eingeführt und klassifiziert.

2.3 NoSQL-Datenbanken

Ziel dieses Kapitels ist es, eine neuartige informationstechnische Lösung durch eine *Einführung in NoSQL-Datenbanken* aufzuzeigen, um darauf folgend die vier grundlegenden *Klassen von NoSQL-Datenbanken* und ihre Eigenschaften zu erläutern. Am Ende des Kapitels werden *Vergleichskriterien* zur Auswahl einer NoSQL-Datenbank zur bedarfsgerechten Darstellung von Supply-Chain-Daten aufgeführt.

2.3.1 Einführung in NoSQL-Datenbanken

In diesem Unterkapitel werden nicht-relationale Ansätze zur Speicherung und Aufbereitung umfangreicher Datenmengen vorgestellt und darauf folgend klassifiziert. Bedingt durch die zunehmende Informationsflut des Internets und Trends wie Big Data, wurde nach neuartigen informationstechnischen Lösungswegen zur Speicherung und Aufbereitung von Daten in Informationssystemen gesucht. Diese Neuorientierung brachte den durch das Schlagwort NoSQL beschriebenen Ansatz nicht (nur) relationaler Datenbanken hervor. Nicht-relationale Datenbanken sind funktional weniger mächtig als relationale DBS, lassen sich aber dafür besser skalieren. [Kem11, S.515] Erst die verteilte Architektur der NoSQL-Datenbank, also die Replikation und Partitionierung der Daten über mehrere Server, ermöglicht eine große Anzahl an Schreib- und Leseoperationen pro Sekunde und gewährleistet so die Kerneigenschaft der horizontalen Skalierung. [Cat11, S.12] Nicht-relationale Ansätze, kurz NoSQL-Datenbanken, wobei das ‚No' in NoSQL für not-only steht, wurden parallel zu relationalen Systemen entwickelt. Schon 1979 entwarf Ken Thompson eine Key/Hash-Datenbank mit dem Namen DBM (Database Manager). In den 80er Jahren entstanden populäre Systeme wie Lotus Notes oder BerkleyDB. Doch erst in den 2000er Jahren mit Web 2.0 und der Notwendigkeit, immer größere Datenmengen zu verarbeiten, startete die Entwicklung von NoSQL-Technologien durch. Große Firmen wie Oracle, Facebook, Google oder Amazon entwickeln ihre eigenen NoSQL-Systeme, wie Hypertable, CouchDB oder Cassandra. [Edl10, S.1] Heutzutage zählen diese Systeme zu den klassischen NoSQL-Datenbanken, die in vier Grundklassen unterteilt werden.

Unter NoSQL-Datenbanken werden eine neue Generation IT-Lösungen für umfangreiche Datenmengen im Peta- bis Zetabyte-Bereich verstanden, die folgende Anforderungen ganz oder teilweise erfüllen:

- *nach Definition:* Datenmodell ist nicht relational

- *nach Skalierbarkeit:* verteilt und horizontal
- *nach Lizenz:* Open Source
- *nach Schemarestriktion:* schemalos oder schwächere Restriktionen
- *nach Replikation:* einfache Datenreplikation durch eine verteilte Architektur
- *nach Schnittstelle:* einfaches API (engl. Application Programming Interface)
- *nach Konsistenzmodell:* Eventually Consistent (BASE) [Edl10, S.2]

ACID	BASE
Priorität: Konsistenz (strong consistency)	Verzögerte Gewährleistung der Konsistenz (weak / eventually consistency)
weitgehend pessimistische Synchronisationsverfahren mit Sperrprotokollen	weitgehend optimistische Synchronisationsverfahren mit Differenzierungsoptionen
Verfügbarkeit (availability) bei geringen Datenmengen gewährleistet	hohe Verfügbarkeit bzw. Ausfalltoleranz bei massiv verteilter Datenhaltung
Integritätsregeln sind im Schema der Datenbank gewährleistet, z.B. referenzierte Integrität	Integritätsregeln sind im Schema der Datenbank gewährleistet, z.B. referenzierte Integrität

Tabelle 2-4: Vergleich zwischen ACID und BASE [Mei16, S.154]

Meier und Kaufmann führen als weitere Bedingung die drei Charakteristika für Big Data *Volume* (Umfang), *Variety* (Vielfalt) und *Velocity* (Echtzeitverarbeitung) auf, um nochmals die Anforderung für NoSQL-Datenbanken zur Verwaltung und Verwertung umfangreicher, heterogener und schnelllebiger Datenmengen zu betonen. Somit wird dieser Punkt zu den Anforderungen für NoSQL-Systeme ergänzt:

- *Mindestens 3 V:* Volume steht für die Bearbeitung umfangreicher Datenmengen, Variety für die Vielfalt der Daten, die sowohl strukturiert, semi-strukturiert als auch unstrukturiert anfallen können. Velocity bezeichnet, dass die Daten in Echtzeit zu verarbeiten sind. [Mei16, S.20]

Bei der Verteilung der Datenbank und der Ausrichtung auf Skalierbarkeit ist es wichtig, diese neuartigen Informationssysteme über verschiedene Dimensionen zu skalieren. Zum einen über die umfangreiche Menge der Daten. Hierbei ist es von Vorteil, wenn die Datenbanken von Beginn an auf Skalierung ausgelegt ist. Da viele

Nutzer auf die Dienste zugreifen, ist eine weitere Dimension Zuverlässigkeit (engl. availability) von hoher Bedeutung. Die Notwendigkeit von kontrollierter Redundanz durch Replikation ermöglicht ein hohes Maß an Zuverlässigkeit und Verfügbarkeit.
Die Schemafreiheit erleichtert es, neue Anforderungen an das DBS anzugliedern. So verlaufen etwa Web2.0 Projekte agiler. Bei NoSQL-Datenbanken kann ein Teil der Verantwortung im Umgang mit dem Schema der Anwendung übertragen werden. Es erfolgt keine lange Sperrung der Daten wie bei relationalen Datenbanksystemen. In sicherheitskritischen Transaktionssystemen ist ein Schema jedoch durchaus sinnvoll. Deswegen ist es entscheidend, ob die Anwendung konsistente Daten fordert und deshalb ein Schema unumgänglich ist oder ob man schemafrei mit leichten, seltenen inkonsistenten Zeitfenstern agiler entwickeln kann. Die Replikation der Daten ist eine logische Konsequenz der verteilten Architektur. Da in relationalen Systemen die Replikation nur umständlich umgesetzt werden konnte, wurde dies in NoSQL-Datenbanken bereits zu Anfang integriert.
Eine einfache Benutzerschnittstelle API (engl. Application Programming Interface) ist zwar auch bei der Abfragesprache SQL, da es der reifste Standard ist, der Fall. Jedoch nicht bei unüberlegtem Hinzufügen von Spalten und Tabellen, auch arbeitet SQL mit Strings, die fehleranfällig sind. [Edl10, S.3f]
Das Konsistenzmodell von NoSQL-Datenbanken benötigt nicht immer die strikte Konsistenzanforderung. Es wird also im Gegensatz zu relationalen DBS kein ACID gefordert, welches die Inkonsistenz von Daten verhindert. Das Konsistenzmodell BASE (basic, available, soft state, eventually consistent) reicht für NoSQL-Datenbanken aus und man spricht von Systemen mit Eventually Consistency oder Weak Consistency. [Mei16, S.20ff] Dabei erlaubt das DBS replizierten Knoten zwischenzeitlich unterschiedliche Daten-versionen zu halten, die erst zeitlich verzögert aktualisiert werden. Die Konsistenz wird also der Verfügbarkeit und der Ausfalltoleranz untergeordnet, um so eine horizontale Skalierung, Replikation und kurze Reaktionszeiten der Datenbank zu ermöglichen. [Edl10, S. 31ff] Ein Vergleich zwischen ACID und BASE in Tabelle 2-4 erklärt die wesentlichen Unterschiede beider Methoden zur Konsistenzsicherung innerhalb der Datenbankanwendung:
Im Gegensatz zur Konsistenzsicherung bei relationalen DBS, die ausschließlich ACID anbieten, können NoSQL-Systeme sowohl ACID als auch BASE gewährleisten. Sobald eine Änderung an der Datenbasis von massiv verteilten Datenhaltungssystemen vorgenommen wird, wird dies den Replikaten mitgeteilt. Das ist der Normalfall, allerdings kann es vorkommen, dass einige Knoten bei Anfragen des Benutzer aufgrund von zeitlicher Verzögerung nicht den aktuellen Zustand zeigen. So kann ein einzelner Knoten in einem verteilten Rechnernetz meistens verfügbar (Basically Available) und manchmal noch nicht konsistent (Eventually

Consistent) sein. Das bedeutet, dieser Knoten befindet sich in einem weichen Zustand (Soft State). [Mei16, S.154]

Bei relationalen Datenbankanwendungen werden innerhalb einer Transaktion die zu lesenden oder zu verändernden Objekte vor weiterem Zugriff geschützt und somit gesperrt und nach Abschuss der Transaktion wieder freigegeben. So verwenden relationale DBS im Gegensatz zu NoSQL-Datenbanken pessimistische Ansätze zur Synchronisation. [Mei16, S.141] Laut CAP-Theorem können massiv verteilte Datenhaltungssysteme, die hohe Verfügbarkeit und Ausfalltoleranz besitzen, nur verzögerte konsistente Zustände garantieren. Aufgrund des hohen Aufwandes bei Sperrung und Freigabe von Transaktionen auf replizierten Knoten verwenden NoSQL-Datenbanken optimistische Synchronisationsverfahren. [Mei, S.155]

Wie zuvor beschrieben wird die Konsistenz aufgrund des CAP-Theorems verzögert gewährleistet. Das von Eric Brewer im Jahre 2000 in einem Vortrag beschriebene CAP-Theorem sagt aus, dass alle verteilten Informationssysteme nur zwei der drei Zielgrößen erreichen können:

- Konsistenz (Consistency) bedeutet, dass alle Knoten einer verteilten Datenbank mit mehreren replizierenden Knoten zur selben Zeit dieselben Daten sehen. Das bedeutet, dass bei einer Transaktion, die auf einem Knoten Daten verändert, alle folgenden Lesezugriffe, egal über welchen Knoten, den aktuellen Wert zurückliefern. So können die Werte in großen Rechnerclustern mit vielen Knoten erst dann wieder gelesen werden, wenn alle replizierenden Knoten aktualisiert sind.
- Verfügbarkeit / Zuverlässigkeit (Availability) bezeichnet zum einen den un-unterbrochenen Betrieb der laufenden Anwendung und zum anderen akzeptable Antwort- oder Reaktionszeiten der DB.
- Ausfalltoleranz (Partition Tolerance) bedeutet, dass der Ausfall eines Knotens oder der Kommunikationsverbindung zwischen den Knoten einer verteilten DB keine Auswirkung auf das Gesamtsystem hat und es weiterhin auf Anfragen von außen reagieren kann. [Edl10, S.31f; Kem15, S.779; Mei16, S.148ff]

Die Abbildung 2-7 ‚CAP-Theorem' veranschaulicht eine Möglichkeiten zur Klassifizierung von NoSQL-Datenbanken:

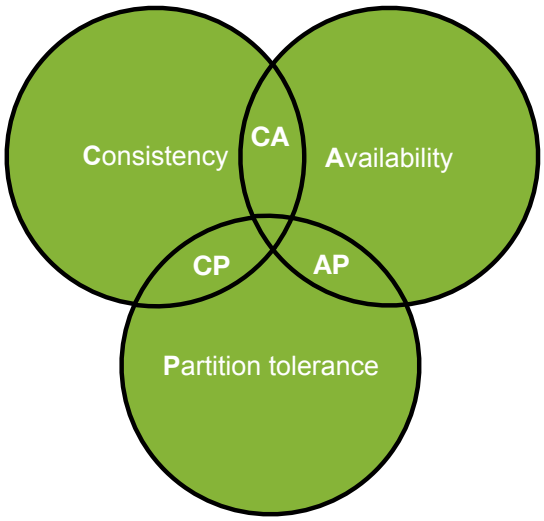

Abbildung 2-7: CAP-Theorem [Jin11, S.364]

Wie die Schnittfläche CA in der Abbildung 2-7 andeutet, erreichen Datenbanken nur eine starke Konsistenz und hohe Verfügbarkeit (CA), indem die Ausfalltoleranz weggelassen wird. Zu diesen CA-Systemen gehören vor allem relationale Datenbanken.
Sobald Konsistenz und Ausfalltoleranz (CP) erreicht werden soll, geht dies zu Lasten der Verfügbarkeit. Solche Datenbanken speichern Daten in verteilten Knoten, wobei die Konsistenz sichergestellt wird, aber geringe Unterstützung der Verfügbarkeit gewährleistet wird. NoSQL-Datenbanken, die zu den Haupt-CP-Systemen gehören, sind beispielsweise spaltenorientierte Datenbanken wie Big Table, Hypertable oder Hbase, auch dokumentenorientierte Datenbanken wie MongoDB oder Terrastore zählen zu CP-Systemen und Schlüssel-Wert-Datenbanken wie Redis, MemcacheDB oder BerkeleyDB sind Vertreter dieses Ansatzes.
Zu Datenbanksystemen, die Verfügbarkeit und Ausfalltoleranz (AP) gewährleisten und somit Konsistenz unterordnen, gehören beispielsweise dokumentenorientierte NoSQL-Datenbanken wie CouchDB, SimpleDB oder Riak, aber auch Schlüssel-Wert-Datenbanken wie Voldemort, Tokio Cabinet oder KAI gehören zu den AP-Systemen. [Jin11, S.364f]
Zusätzlich ist festzuhalten: sobald die Entscheidung für die Kerneigenschaft Konsistenz (C) und der Verfügbarkeit (A) getroffen wird, liegt der Vorteil in der einfachen Entwicklung der Anwendung und der vereinfachten Verwaltung der Daten. Nachteilig ist die Notwendigkeit der Implementierung einer komplexen Anwendungslogik, die die Inkonsistenzen der Daten erkennen und beheben soll. Die Priorität der Verfügbarkeit (A), bei dem gleichzeitigen Versuch der Maximierung der Konsistenz

(C), ist aus Sicht der Wirtschaftlichkeit gerechtfertigt, denn die Nichtverfügbarkeit einer Dienstleistung bedeutet finanzielle Verluste. Der Verzicht auf starke Konsistenz hingegen ermöglicht eine erhebliche Verbesserung der horizontalen Skalierung von NoSQL-Datenbanken. [Pok13, S.71ff] Die Konsistenzsicherung bei massiv verteilten und umfangreichen Datenmengen bildet ein Kernkriterium zur Auswahl einer NoSQL-Datenbank.

2.3.2 Kernklassen von NoSQL-Datenbanken

In diesem Kapitel werden die vier Kernklassen von NoSQL-Datenbanken vorgestellt und ihre Eigenschaften näher betrachtet. Das Ziel ist darauf folgend Vergleichskriterien zur Auswahl einer NoSQL-Datenbank abzuleiten.
Bei der Kategorisierung von NoSQL-Datenbanken lassen sich die Systeme in vier Kernklassen, sogenannte Core-NoSQL-Systeme und nachgelagerte, sogenannte Soft-NoSQL-Systeme unterscheiden.

Die vier Core-NoSQL-Datenbanksysteme sind:

- Spaltenorientierte Datenbanken (Column Stores)
- Dokumentenorientierte Datenbanken (Document Stores)
- Schlüssel-Wert-Datenbanken (Key Value Stores)
- Graphenorientierte Datenbanken (Graph Database)

gefolgt von nachgelagerten Soft-NoSQL-Datenbanksystemen:

- Objektdatenbanken
- XML-Datenbanken
- Grid-Datenbanken
- weitere nicht-relationale Systeme

Die verschiedenen Kategorisierungen von NoSQL-Datenbanksystemen im Vergleich zu relationalen Datenbanksystemen verdeutlicht Abbildung 2-8.

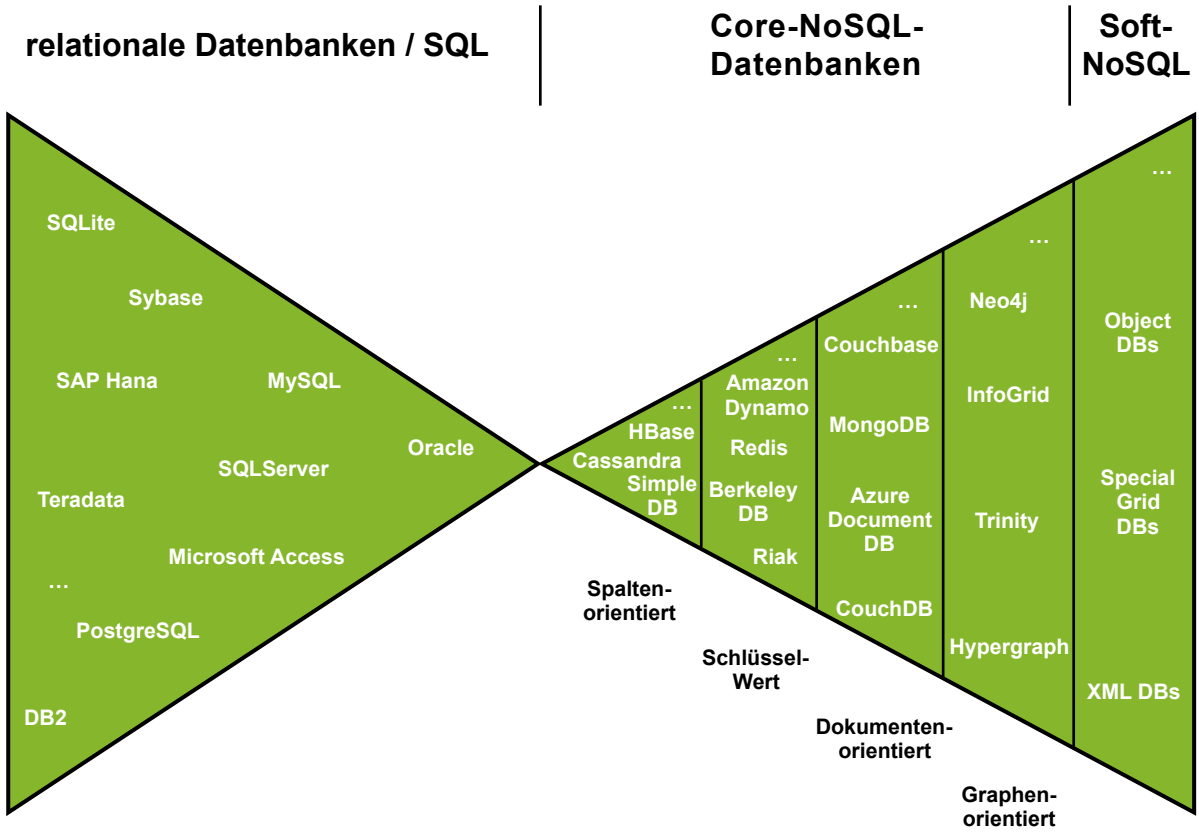

Abbildung 2-8: Relationale DB vs. Core-NoSQL-DB und Soft-NoSQL-DB [Edl10, S.6]

Folgende Eigenschaften klassifizieren spaltenorientierte Datenbanken zu einer der vier Kernklassen von NoSQL-Datenbanken:

- Daten: Speicherung in mehrdimensionalen Tabellen
- Adressierung der Datenobjekte: Zeilenschlüssel
- Adressierung der Objekteigenschaften: Spaltenschlüssel
- Zusammenfassung der Spalten einer Tabelle: Spaltenfamilien
- Tabellenschema: Bezug ausschließlich auf Spaltenfamilien, wobei innerhalb einer Spaltenfamilie beliebige Spaltenschlüssel verwendet werden können
- Optimierung der Antwortzeit: Bei verteilten und zerstückelten Architekturen werden Daten zu einer Spaltenfamilie am gleichen Ort gespeichert

Unter einer Spaltenfamilie (engl. column family) wird eine logische Verknüpfung von Spalten einer beliebigen Anzahl verstanden, auf die gemeinsam zugegriffen wird. Aus diesem Grund werden Daten in spaltenorientierten Datenbanken physisch in Spaltenfamilien anstatt in Zeilen gespeichert. Das flexible Schema ermöglicht, Änderungen zur Laufzeit durchzuführen. So können Spalten hinzugefügt oder

entfernt werden. Zusätzlichen besitzt eine Spalte oder Zeile einen Namen und zugehörigen Wert mit einer einfachen oder komplexen Struktur. Durch die Berücksichtigung der Struktur von Spaltenfamilien stellt jede Zeile ein stark strukturiertes Datenelement dar, welches durch einen Schlüssel eindeutig identifiziert wird. Zusammenfassend kann man spaltenorientierte Datenbanken als erweiterte Schlüssel-Wert-Datenbanken ansehen, in denen der Wert als eine Sequenz verschachtelter Paare dargestellt wird. [Dav18, S.7ff] Aus diesem Grund bieten spaltenorientierte NoSQL-Datenbankanwendungen durch massive Verteilung die gleichen Vorteile von hoher Skalierbarkeit und Verfügbarkeit wie Schlüssel-Wert-Datenbanken. Durch ein Schema mit Zugriffskontrolle und Lokalisierung von verteilten Daten auf Ebene der Spaltenfamilie bieten spaltenorientierte Datenbanken eine nützliche Struktur. Die Flexibilität innerhalb der Spaltenfamilie wird durch die Möglichkeit der Verwendung beliebiger Spaltenschlüssel ermöglicht. [Mei16, S.227f; Jin11, S.365f; Dav18, S. 6ff] Die Abbildung 2-9 zeigt ein Beispiel einer spalten-orientierten Datenbank, wo jede Datenzelle mit Spalten- und Zeilenschlüssel adressiert wird. Hier ist jedem Benutzer ein Zeitstempel (z.B. t1) und ein Zeilenschlüssel (z.B. U17547) zugeordnet. Zudem sind verschiedene Spalten, wie Mail und Name zur Spaltenfamilie *Contact* zusammengefasst.

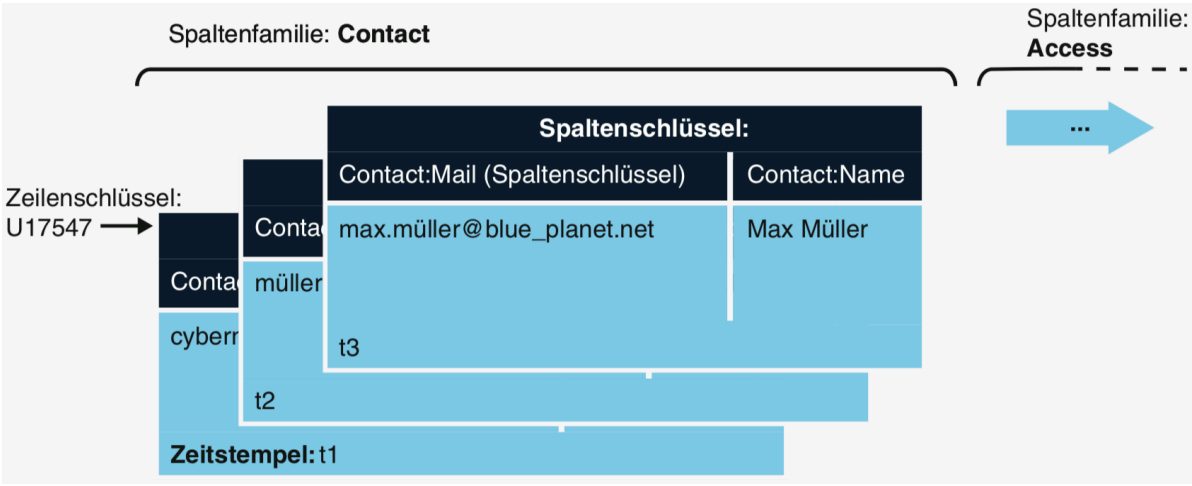

Abbildung 2-9: Beispiel einer spaltenorientierten Datenbank aus [Mei16, S.228]

Eine NoSQL-Datenbank mit folgenden Eigenschaften gehört zu den Schlüssel-Wert-Datenbanken:

- Menge von identifizierenden Datenobjekten: Schlüssel
- Genau ein Datenobjekt zu jedem Schlüssel: Wert
- Abfrage des Wertes: Durch Angabe der Schlüssel

Bei Schlüssel-Wert-Datenbanken können umfangreiche Datenmengen effizient geschrieben und wieder gelesen werden, da die Anforderung der referentiellen Integrität, d.h. die Bedingung zur Sicherung der Datenintegrität bei Nutzung, nicht überprüft wird. Durch die zusätzliche Speicherung der Schlüssel-Wert-Paare im Hauptspeicher (RAM) der Datenbank bei sogenannte In-Memory-Datenbanken erhöht sich die Geschwindigkeit der Datenverarbeitung nochmals. Zusätzlich kann die Skalierbarkeit bei Schlüssel-Wert-Datenbanken, durch die Zerstückelung der Datenbasis (Sharding) auf mehrere Speicher im Rechnercluster, beliebig ausgebaut werden. Um die Ausfallsicherheit innerhalb der verteilten Architektur zu erhöhen, kann die Datenbank auf andere Rechner kopiert und aktuell gehalten werden. Dabei findet die Synchronisation der eigentlichen Datenbasis, dem Master-Cluster, mit einer oder mehreren replizierten Datenbasen, den Slave-Clustern, statt. Neben dem Sharding für große Datenmengen, haben Schlüssel-Wert-Datenbanken den Vorteil der Flexibilität im Schema der Datenbank. Im Gegensatz zu relationalen DBS, wo jeder neu zu speichernde Datensatz ein Schema, in Form einer Relation mit Attributen, bestehen oder eine Schemadefinition vorgenommen werden muss, liegt der Vorteil bei Schlüssel-Wert-Datenbanken darin, dass keine Tabellen mit Spalten und Datentypen definiert werden müssen. So können Daten unter einem beliebigen Schlüssel abgelegt werden. Dabei hat auch die Einfachheit im Datenmodell eine schnelle und effiziente Verwaltung der Daten zur Folge [Edl10, S.7; Mei16, S.224f; Dav18, S.5ff]. Die Abbildung 2-10 veranschaulicht ein Beispiel einer Schlüssel-Wert-Datenbank mit massiv verteilter Architektur.

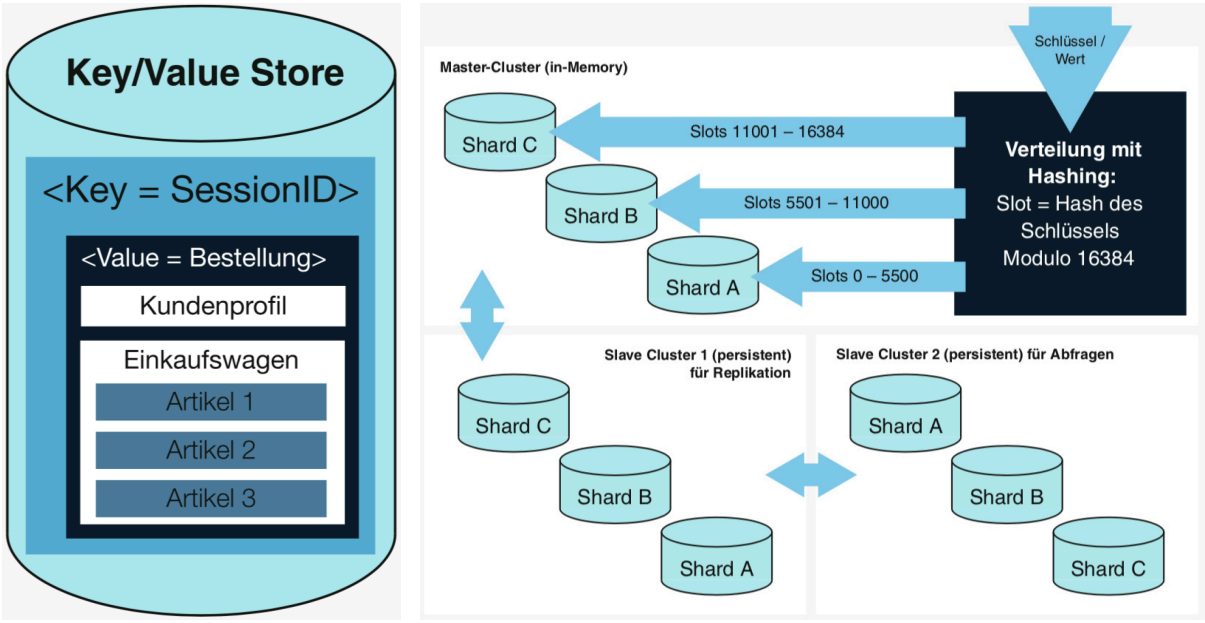

Abbildung 2-10: Beispiel einer Schlüssel-Wert-Datenbank aus [Mei16, S.19ff]

Jedem Schlüssel *SessionID* ist ein Wert *Bestellung* zugeordnet. Hier ist die zuvor beschriebene Möglichkeit des Shardings aufgezeigt. Der Master-Custer besteht aus drei Rechnern (Shard A-C) und die Datenbasis, in Form von Schlüssel-Wert-Paaren, wird direkt im Hauptspeicher gespeichert. Die Replizierung erfolgt über die Slave-Cluster-1 und ein zusätzlicher Slave-Cluster-2 erhöht die Leistung, da dieser extra für komplexe Abfragen und Analyse bereitgestellt wird.

Unter einer dokumentenorientierten Datenbank wird eine NoSQL-Datenbank bezeichnet, die folgende Eigenschaften erfüllt:

- Dokumente: gespeicherte Datenobjekte als Werte zu den Schlüsseln
- Identifikation: eindeutig über Schlüssel
- Enthaltene Datenstrukturen in Dokumenten: Form von rekursiv verschachtelten Attribut-Wert-Paaren ohne referenzielle Integrität
- Vergleichbar mit der Funktionsweise von Schlüssel-Wert-Datenbank
- Schemafreie Datenstruktur: In Dokumenten können beliebige Attribute ohne vorrangige Definition eines Schemas verwendet werden [Mei16, S.230]

Die Funktionsweise der dokumentenorientierten Datenbank ähnelt der der Schlüssel-Wert-Datenbanken, da zu einem beliebigen Schlüssel (Dokumenten-ID) ein Datensatz, sogenannte Dokumente, als Wert gespeichert werden kann. Ein Dokument im Sinne einer dokumentenorientierten Datenbank ist eine Datei mit strukturierten Daten. Diese Datenstruktur stellt eine Liste von Attribut-Wert-Paaren dar. Es existiert keine Beziehung zwischen den Dokumenten, sondern in sich abgeschlossene Sammlungen von Daten. Dokumentenorientierte Datenbanken vereinigen die Schemafreiheit von Schlüssel-Wert-Datenbanken mit der Möglichkeit zur Strukturierung der gespeicherten Daten. Hierbei werden strukturierte Datensätze als Dokumente bezeichnet. Dokumentenorientierte Datenbanken sind horizontal zu skalieren, indem verschiedene Rechner zu einem Gesamtsystem zusammengefasst werden. So wird die Menge der Daten mittels Sharding verteilt, womit sie auf die Verarbeitung großer, heterogener Datenmengen ausgelegt sind. Durch die gänzliche Schemafreiheit entfällt die Notwendigkeit, Datenstrukturen vor dem Einfügen in ein Schema zu definieren. So wird die Schemaverantwortung dem Benutzer übergeben. Ein Nachteil hieraus ist der Verzicht auf referenzierte Integrität und Normalisierung, was sich wiederum auf die Flexibilität der Speicherung heterogener Daten auswirkt. So kann die Zerstückelung der Datenbasis vereinfacht erfolgen. [Dav18, S.9ff; Mei16, S.229ff] Die Abbildung 2-11 stellt eine dokumentenorientierte Datenbank dar, wo Daten zu Nutzer einer Website gespeichert werden. Zu jedem Benutzerschlüssel

mit dem Attribut *_id* wird ein Dokument gespeichert, welches alle Daten zu einem Nutzer wie beispielsweise seinen Benutzername *userName* oder seinen Vornamen *firstName* speichert.

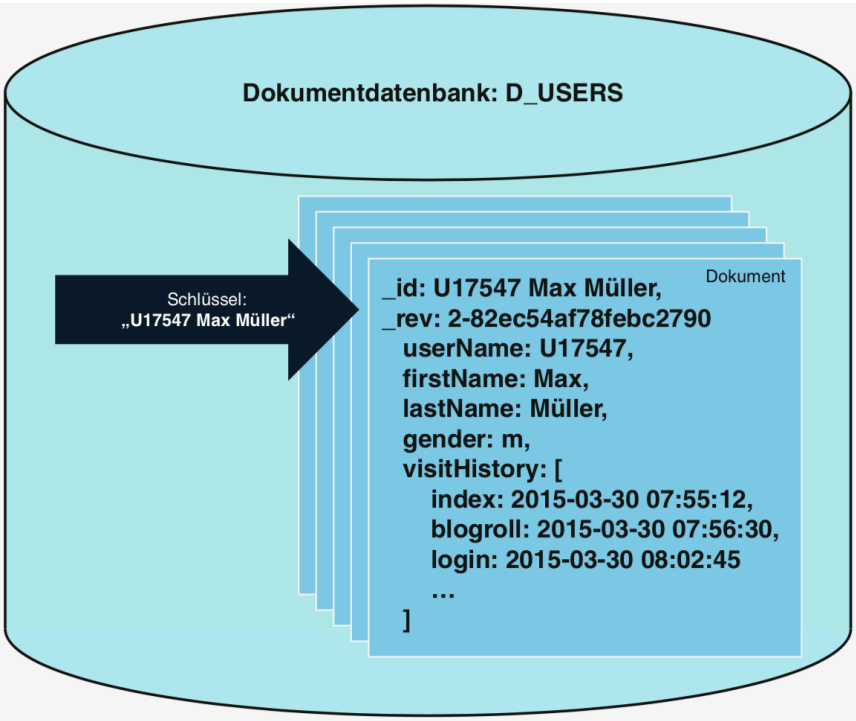

Abbildung 2-11: Beispiel einer dokumentenorientierten Datenbank aus [Mei16, S.230]

Unter graphenorientierten Datenbanken (Graph Database) wird eine NoSQL-Datenbank verstanden, die folgende Eigenschaften erfüllt:

- Abbildung von Daten oder Schema: graphenähnliche Strukturen
- Datenmanipulation: Ausgedrückt durch Transformation des Graphen oder Operationen, wobei Operationen direkt die typischen Eigenschaften, beispielsweise Pfade, Zusammenhänge, Subgraphen, etc., von Graphen ansprechen
- Sicherstellung der Konsistenz: Prüfung von Integritätsbedingungen, wobei sich die Definition von Konsistenz direkt auf die Struktur des Graphen, z.B. Knoten- und Kantentypen, Attribut-Wertebereich und referenzierte Integrität der Knoten bezieht

Im Gegensatz zu den bereits vorgestellten Datenmodellen der Key/Value-, spaltenorientierten- oder dokumentenorientierten NoSQL-Datenbanken, die zugunsten von einfacher Fragmentierung (Sharding) auf Schemata in der Datenbank und auf referenzierte Integrität verzichten, haben graphenorientierte Datenbanken ein strukturiertes Schema zugrunde liegend. Im Eigenschaftsgraphen werden Daten

in Form von Knoten und Kanten gespeichert, welche zu einem Knoten- oder Kantentyp gehören. Hierbei werden Daten in Form von Attribut-Wert-Paaren gehalten. Im Gegensatz zu relationalen DBS, die bei Einfügung neuer Datensätze ein festgelegtes Schema durchlaufen, können Datenobjekte zu einem bisher nicht existierenden Knoten- oder Kantentypen direkt in die Datenbank eingefügt werden. Hierbei muss keine Definition der Typen stattfinden. Eine entsprechende logische Schemaveränderung vollzieht das Datenbankverwaltungssystem und erstellt den entsprechenden Typ eigenständig. Das Einsatzgebiet von graphenorientierten Datenbanken ist die Verwaltung von Daten, die in Netzwerken organisiert sind. Hierbei ist nicht der einzelne Datensatz wichtig, sondern die Verknüpfung der Datensätze untereinander, wie beispielsweise bei sozialen Medien, aber auch bei der Analyse von Infrastrukturnetzen, wie bei der Wasser- oder Energieversorgung. Durch die Eigenschaft der indexfreien Nachbarschaft kann das DBS zu jedem Knoten den direkten Nachbarn finden, ohne sämtliche Kanten zu berücksichtigen. Unabhängig von der Datenmenge bleibt der Aufwand für die Abfrage von Beziehungen konstant, was als wesentlicher Vorteil von graphenorientierte nDatenbanken gesehen werden kann. Beziehungen zwischen den Datensätzen sind das zentrale Element von graphenorientierten Datenbanken, was die Fragmentierung erschwert. Im Gegensatz zu den drei bereits vorgestellten Arten von Kern-NoSQL-Datenbanken, die keine Beziehungen zwischen den Datensätzen sicherstellen, muss bei der Fragmentierung von graphenorientierten Datenbanken besonders Rücksicht auf die Zusammenhänge der Daten genommen werden. Hierfür werden Algorithmen verwendet, die keine effiziente Methode zur Zerlegung des Graphen in Teilgraphen bieten. Folglich bieten heutige Graphen-Datenbanken das Sharding nicht an. [Dav18, S.11ff; Mei16, S.237ff] Die Abbildung 2-12 verdeutlicht eine graphenorientierte Datenbank, die Daten eines Webportals mit Benutzern, Websiten und Beziehungen untereinander abbildet. Das zugrunde liegende Schema mit Knoten- und Kantentypen, denen Attribute zugewiesen sind, weist auf einen Eigenschaftsgraphen hin. Es gibt die zwei Knotentypen USER und WEBPAGE und die drei Kantentypen FOLLOWS, VISITED und CREATED_BY. Den beiden Knotentypen sind jeweils Attribute, wie beispielsweise *Name* oder *UserName* zugeordnet und auch dem Kantentyp VISITED ist ein Attribut *date* mit dem Wertebereich Datum zugeordnet. Zwar ähnelt diese graphenorientierte Datenbank der zuvor beschriebenen dokumentenorientierten Datenbank D_USERS (Abbildung 2-11), da sie vom Typ her ähnliche Daten speichert. Es werden aber beispielsweise Informationen zu Benutzern, die besuchten Websites und Zeitstempel gespeichert, mit dem wesentlichen Unterschied der Beziehungen zwischen den Datenobjekten, die explizit als Kanten vorhanden sind.

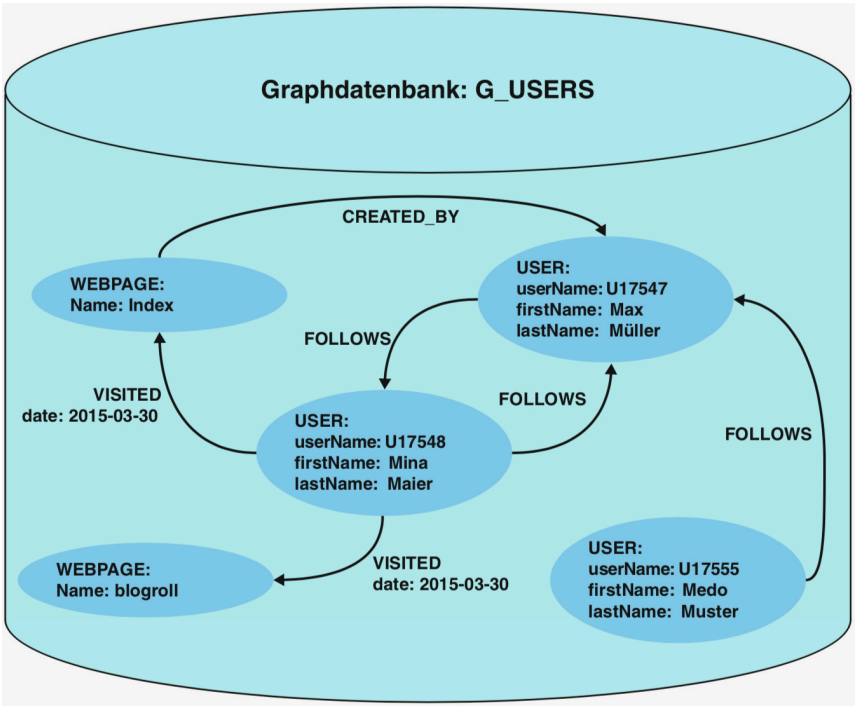

Abbildung 2-12: Beispiel einer graphenorientierten Datenbank aus [Mei16, S.236]

2.3.3 Vergleichskriterien zur Auswahl von NoSQL-Datenbanken

Ziel dieses Kapitels ist es, die in der Literatur erwähnten Vergleichskriterien zur Auswahl von NoSQL-Datenbanken aufzuführen, um im nächsten Kapitel die für diese Arbeit relevanten Vergleichskriterien abzugrenzen und ihre Gewichtung festzulegen.

Primär stellt die Forschung funktionale und nichtfunktionale Anforderungen an die vier Kern-NoSQL-Datenbankklassen, um diese so durch eine Vielzahl von Merkmalen untereinander vergleichbar zu machen. [Edl10, S.272ff; Gup17, S.295ff; Tud11, S.3] Zusätzlich werden Leistungsvergleiche einzelner NoSQL-Datenbanken mit Hilfe des laborbasierten Benchmarking-Werkzeuges YCSB (Yahoo! Cloud Serving Benchmark) durchgeführt. Dieses Messwerkzeug ist ein open source NoSQL-Benchmarking-System, welches erstmals 2010 durch Cooper et al. in dem Artikel ‚Benchmarking cloud serving systems with YCSB' vorgestellt wurde. [Coo10, S.143ff] Es erlaubt eine Bewertung der Leistung von NoSQL-Datenbanken anhand verschiedener Anwendungsszenarien mit variablen Betriebsfaktoren, sogenannten Workloads, wie beispielsweise dem Prozentsatz der Lese- und Aktualisierungsvorgänge, der Gesamtanzahl der Transaktionen und der Anzahl der verwendeten Datensätze. Der Benchmark YCSB besteht aus zwei Komponenten, zum einen dem

Datengenerator und zum anderen einer Reihe von Leistungstests zur Auswertung von Lese- und Aktualisierungsvorgängen innerhalb der einzelnen NoSQL-Datenbankanwendung. [Abr14, S.4ff; Tan16, S.106ff; Tud11, S.3ff] Allerdings lässt sich der Benchmark YCSB nur für drei der vier Kernklassen von NoSQL-Datenbanken durchführen. Somit kann nur die Leistung von NoSQL-Systemen der Kernklassen: spalten- und dokumentenorientierten, sowie von Schlüssel-Wert-Datenbanken sinnvoll beurteilt und untereinander verglichen werden. Da jedoch graphenorientierte Datenbanken andere Anwendungsfälle und Bewertungs-indikatoren haben, unterscheiden sie sich von den drei Kernklassen und werden vom YCSB ausgeschlossen. Zusätzlich erfordert die Verwendung von Ver-knüpfungen zwischen den Datensätzen andere Vergleichsansätze und somit benötigen graphenorientierte Datenbanken spezifische Benschmarking-Instrumente. Hier werden von der Literatur folgende Instrumente zur Leistungsbeurteilung von graphenorientierten Datenbanken vorgeschlagen: XGDBench und LinkBench. [Abr14, S.5; Tan16, S.106]

Edlich et al. unterteilen die Anforderungskategorien zur Auswahl von NoSQL-Datenbanksystemen in sechs Kriterien. Zum einen sind das die fünf funktionalen Anforderungen:

- *Daten*, wobei es wichtig ist, welche Datenart vorliegt, wie komplex diese sind und wie groß die Datenmenge ist.
- *Transaktionen* lassen sich von den zwei klassischen Transaktionsmodellen aus betrachten: ACID und BASE. Zusätzlich sollte die CAP-Abwägung, also in welchem Bereich (CA, CP oder AP) die NoSQL-Datenbank anzusiedeln ist, relevant sein.
- *Performance* oder die Leistung der NoSQL-Datenbank ist in Bezug auf Latenzzeit der Antworten, vorhersagbares Antwortverhalten, aber auch die Ausführungszeit für Lese- und Schreibvorgänge (YCSB Benchmark) zu untersuchen.
- *Abfrageanforderungen:* Intensität und Form der Abfragen (Query API)
- *Architektur*, die zum einen den Grad der Verteilung und das Einsatzgebiet der Datenbank beschreibt und zum anderen das Datenzugriffsmuster. Wenn es sich beim Einsatzgebiet um lokale oder parallele Anwendungen handelt, sind mobile Einsatzfelder wichtig oder gar offline Zugriffe durch eine replizierte Peer-2-Peer-Architektur erforderlich. Zusätzlich ist es wichtig, im Vorfeld das typische Datenzugriffsmuster (Data Access Patterns) herauszufinden, also ob es sich um Anwendungen mit vielen Schreib-, aber wenig Lesezugriffen

handeln wird, wie beispielsweise in der Spielindustrie. Oder anders herum wie bei online Einkaufsprotalen, wo nur wenige Daten geschrieben werden, dafür aber zahlreiche Lesezugriffe stattfinden. [Edl10, S.273ff]

Zum anderen beschreiben Edlich et al. das sechste Kriterium ‚nicht-funktionale Anforderungen', das wiederum in mehrere Unterkategorien unterteilt ist:

- *Replikationsart*, also wie die Konfigurierung der Anzahl der Replikate im System vollzogen wird. Bei vielen Anwendungen ist dies sehr einfach und geschieht automatisch, wiederum andere replizieren per Befehl. Die Unterscheidung erfolgt zwischen eager- (sofortiger) und lazy-Replikation (irgendwann).
- *Unterstützungsleistung für den Support*. Da es sich bei NoSQL-Datenbanken um eine junge technologische Entwicklung handelt, sind die verfügbaren Supportleistung der Hersteller nicht auf dem Niveau von relationalen DB, was bei der Auswahl der Anwendung berücksichtigt werden muss.
- *Qualifikation der DB-Entwickler* kann aufgrund der kurzen Entwicklungszeit Defizite aufweisen, die es zu beachten gibt.
- *Sicherheitsanforderungen (security).* Hier stellt sich die Frage nach dem Angebot eines umfangreichen Rollen- und Benutzer-Managements und ob integrierte Verschlüsselungen benötigt werden.
- *Unternehmensrestriktionen* gilt es zu prüfen, da viele Unternehmen bestehende relationale Infrastrukturen oder Vertragsbindungen gegenüber Herstellern haben. So kann die Integration in bestehende Datenbank- oder Data-Warehouse-Umgebungen nicht vollzogen werden.
- *Crash Resistance*, also die Unempfindlichkeit des Systems gegenüber Ausfällen. Auch die Frage nach einem Single Point of Failure ist erforderlich.
- *DB-Vielfalt im Unternehmen*. Hier gilt es herauszufinden, ob eine Kultur der Datenbankvielfalt im Unternehmen möglich ist und ob eine Vielfalt von Anwendungen auch genügend Vorteile mit sich bringt.
- *Sicherungs- / Wiederherstellungsmöglichkeiten*, die die Schnelligkeit und Einfachheit von Sicherungen und Wiederherstellungen (backup restore) der Anwendung ermöglichen.
- *Lizenzkosten / Open Source* sind bezüglich der Datenbankkosten und somit der Wirtschaftlichkeit der Integration eines neuen Systems hin zu untersuchen. [Edl10, S. 279f]

Zusätzlich führen die Autoren Gupta et al. folgende Anforderungskategorien auf der Konferenz ‚2017 International Conference on Computing and Communication Technologies for Smart Nation‘ ein:

- Leistung der Abfragen,
- Skalierbarkeit der Daten,
- Flexibilität des Schemas,
- Struktur der Datenbank,
- Komplexität der Werte. [Gup17, S.295]

Zwar werden diese Merkmale von Gupta et al. als nicht-funktional beschrieben, können jedoch als sinnvolle Ergänzung zu den von Edlich bereits erwähnten funktionalen Anforderungskategorien gesehen werden und zu ihnen hinzugefügt werden. So lassen sich die Kategorien ‚Skalierbarkeit der Daten‘ und ‚Komplexität der Werte’ zu der Anforderungskategorie ‚Daten‘ von Edlich hinzufügen, auch die ‚Leistung der Abfragen‘ lässt sich sinnvoll zur Kategorie ‚Abfrageanforderung‘ hinzufügen. Die ‚Struktur der Datenbank‘ ist durch die Anforderungskategorie ‚Architektur‘ beschrieben. Nur die ‚Flexibilität des Schemas‘ wird als gänzlich neue Anforderungskategorie aufgenommen.
Zudem vergleichen die Autoren Gupta et al. einzelne NoSQL-Datenbanken unter den folgenden Merkmalen, wie:

- Schreib-, Lese-, und Lösch-Operationen,
- Lizenz,
- Schema,
- Datenkomplexität,
- Partitionierung,
- Replikationsmethoden,
- Skalierung,
- Transaktionskonzepte,
- Sicherheitsaspekte. [Gup17, S.296]

Auch hier lassen sich einzelne Merkmale von Gupta et al. zu den von Edlich eingeführten funktionalen und nicht-funktionalen Anforderungskategorien ergänzen, es ergeben sich aber auch gänzlich neue Kriterien, wie beispielsweise die ‚Partitionierung‘. Die Kriterien ‚Schreib-, Lese-, und Lösch-Operationen‘ werden zur Anforderungskategorie ‚Performance‘ ergänzt. Alle andere Merkmale lassen sich sinngemäß ergänzen.

Tudorica et al. differenzieren auf der Konferenz ‚2011 RoEduNet International Conference: Networking in Education and Research' die Anforderungskategorien an NoSQL-Systeme in qualitative und quantitative Kriterien.

Zu den qualitative Kriterien legen die Autoren folgende Funktionen zum Vergleich der NoSQL-Datenbanken fest:

- Persistenz: dauerhafte Speicherung von Daten im DBS.
- Replikation,
- Hohe Verfügbarkeit,
- Transaktionen,
- Implementierungssprache,
- Einflüsse / Sponsoren,
- Lizenztyp. [Tud11, S.3]

Als quantitatives Kriterium wird von Tudorica et al. ein Leistungstest der einzelnen NoSQL-Datenbanken nach dem zuvor vorgestellten laborbasierten Benchmarking-Instrument YCSB durchgeführt. Auch hier lassen sich die Kriterien von Tudorica et al. zu denen von Edlich ergänzen.

In ihrem Artikel ‚Survey on NoSQL database' aus dem Jahre 2011 schlagen Jing et al. unter anderem folgende Optionen zur Auswahl einer NoSQL-Datenbank für Unternehmen vor:

- Datenmodell,
- CAP Unterstützung,
- Performance,
- Abfragen API,
- Zuverlässigkeit,
- Daten Persistenz,
- Support-Leistungen. [Jin11, S.366]

Auch diese Kriterien lassen sich sinngemäß ergänzen.

Auf der Konferenz ‚2017 IEEE 2nd International Conference on Big Data Analysis' führen Kalid et al. weitere Anforderfungskategorien zum Vergleich von NoSQL-Datenbanken ein:

- System Performance,
- Skalierbarkeit,

- Daten Partitionierung,
- Fehlertoleranz,
- CAP Einordnung,
- Lizenzierung. [Kal17, S.91]

Die Anforderungskategorien von Kalid et al. lassen sich sinngemäß ergänzen.

Somit ist eine umfangreiche Literaturrecherche zu den gängigsten Anforderungskategorien zum Vergleich von NoSQL-Datenbanken durchgeführt. Diese werden im folgenden Kapitel dazu verwendet, um für diese Arbeit relevante Vergleichskriterien festzulegen.

3. Vergleich von NoSQL-Datenbanken

In diesem Kapitel werden mehrere NoSQL-Datenbanken der vier Kernklassen untereinander verglichen, um so eine NoSQL-Datenbank auszuwählen, die die anfallenden Transaktionsdaten in der SC bedarfsgerecht darstellen kann. In Kapitel 3.1 (Vergleichsmatrix) werden zuerst relevante Vergleichskriterien in Bezug auf die in Kapitel 2 (Grundlagen zum Datenmanagement) aufgeführten Voraussetzungen festgelegt, um anschließend mit einer NoSQL-Datenbank eine exemplarische Darstellung eines beispielhaften Transaktionsdatensatzes in Kapitel 3.2 aufzuzeigen.

3.1 Vergleichsmatrix

Ziel dieses Kapitels ist es, eine NoSQL-Datenbank aus den vier Kernklassen mit Hilfe einer Vergleichsmatrix auszuwählen. Dazu werden die in Kapitel 2.3.3 (Vergleichskriterien zur Auswahl von NoSQL-Datenbanken) aufgelisteten Kriterien von unterschiedlichen Autoren in übersichtlicher Form dargestellt und die für diese Arbeit relevanten Kriterien festgelegt. Darauf folgend wird jeweils eine Datenbank aus den vier Kernklassen von NoSQL-Datenbanken untereinander verglichen.

3.1.1 Bestimmung der Vergleichskriterien

Durch die umfangreiche Literaturrecherche in Kapitel 2.3.3 lassen sich alle Kategorien der verschiedenen Autoren in der Tabelle 1-5 zusammenfassend

darstellen und nach dem Mehrheitsprinzip (von mindestens drei Autoren erwähnt) die für diese Arbeit relevanten Vergleichskriterien festlegen.

Auswahl der Vergleichskriterien					
Kriterien	Edlich et al. (2010)	Gupta et al. (2017)	Tudorica et al. (2017)	Jing et al. (2011)	Kalid et al. (2017)
Daten	x	x		x	
Transaktion	x	x	x		
Leistung	x	x	x	x	x
Abfragen	x	x		x	
Architektur	x	x			
Replikation	x	x	x		
Schema		x			
Partition		x			x
Sicherheit	x	x			
Lizenz	x	x	x		x
Persistenz			x	x	
Skalierung		x			x
CAP	x			x	x
Support	x			x	

Tabelle 3-1: Auflistung von Vergleichskriterien aus unterschiedlichen Studien zur Auswahl von NoSQL-Datenbanken

Somit werden für diese Arbeit sieben relevante Vergleichskriterien festgelegt:

- Daten
- Transaktion
- Leistung
- Abfragen
- Replikation
- Lizenzierung
- CAP

Das erste Vergleichskriterium ‚Daten' fasst die Datenart, Datenmenge, Komplexität der Daten und die Intensität der Datenbeziehungen zusammen. Wie in Kapitel 2.1.3 (Transaktionsdaten in der Supply Chain) beschrieben, handelt es sich um Transaktionsstrukturdaten, -aktivitätsdaten und -kontrolldaten, die eine hohe Verknüpfungsintensität untereinander aufweisen. Da keine Datentypen selbst definiert werden müssen, weisen Transaktionsdaten eine geringe Datenkomplexität auf und infolge von Big Data nimmt die Datenmenge immer weiter zu. Dieses Vergleichskriterium ist von wesentlicher Bedeutung in dieser Arbeit und findet besondere Berücksichtigung bei der Gewichtung in der Vegleichsmatrix.

Das zweite Vergleichskriterium ‚Transaktion' bildet das Konsistenzmodell der NoSQL-Datenbank, BASE oder ACID, wie in Kapitel 2.3.1 (Einführung in NoSQL-Datenbanken) beschrieben ist, ab. Zwar sind die meisten NoSQL-Systeme Eventually Consistent, also BASE, einige bieten jedoch auch ACID an.

Das dritte Vergleichskriterium ‚Leistung' beschreibt die relative Leistung einer NoSQL-Datenbank, die mit vordefinierten Anwendungsszenarien durch variable Workloads ermittelt wird. Näher ist dies in Kapitel 2.3.3 (Vergleichskriterien zur Auswahl von NoSQL-Datenbanken) durch die Benchmarking-Instrumente YCSB und XGDBench beschrieben. Da sich nur drei der vier Kernklassen untereinander vergleichen lassen, findet dieses Vergleichskriterium die geringste Berücksichtigung in der Vergleichsmatrix.

Das vierte Vergleichskriterium ‚Abfragen' befasst sich mit der Form der Abfrage, also der Query-API, womit die Abfragesprache der NoSQL-Anwendung gemeint ist. Dieses Kriterium ist wichtig, da es eine Vielzahl von Abfragesprachen gibt und der Anwender gegebenenfalls diese erst erlernen müsste. Es lassen sich aber auch beispielsweise komplexe Abfrage durch die Abfragesprache Cypher, die extra für Neo4j entwickelt wurde, realisieren.

Das fünfte Vergleichskriterium ‚Replikation' beschreibt die Konfigurierung der Anzahl der Replikate im System. Weil NoSQL-Datenbanken eine verteilte Architektur aufweisen und auf dem Scale-Out-Prinzip basieren, das eine Erweiterung des Systems durch Hinzufügen neuer Rechnerressourcen, wie es in der Cloud-Umgebung üblich ist, ermöglicht. Somit ist dieses Kriterium für die horizontale Skalierung des Systems entscheidend.

Das sechste Vergleichskriterium ‚Lizenzierung' beschreibt das Recht der wirtschaftlichen Nutzbarkeit einer NoSQL-Datenbank. Zwar haben die meisten NoSQL-Systeme eine freie Softwarelizenzierung (Open Source), sie stellt jedoch nur limitierte Funktionen zu Verfügung und ist für die kommerzielle Nutzung nur bedingt

geeignet ist. Zusätzliche Angebote wie Supportleistungen oder die Nutzung größerer RAM-Speicher werden nur kostenpflichtig angeboten.

Das siebte Vergleichskriterium ‚CAP' beschreibt die CAP-Abwägung des NoSQL-Systems, also die Einordnung der NoSQL-Datenbank in den CA-, CP- oder AP-Bereich des CAP-Theorems. Dies ist in Kapitel 2.3.1 (Einführung in NoSQL-Datenbanken) beschrieben, bildet ein Kernkriterium bei der Auswahl einer NoSQL-Datenbank und findet bei der Gewichtung besondere Berücksichtigung.

Zusätzlich kann ein Fragebogen dem Unternehmen bei der Auswahl einer NoSQL-Datenbank helfen. Damit kann man weitere Kategorien, wie z.B. die Unterstützungsleistung des Supports, die Sicherheit der Anwendung, durch Rollen- und Benutzermanagement, oder auch Unternehmensrestriktionen, wie die Vertragsbindungen an Hersteller, zum Vergleich der NoSQL-Anwendungen mit einfließen lassen. So lassen sich die NoSQL-Systeme noch präziserer und bedarfsgerechter analysieren, um die jeweils passende Datenbank zum spezifischen Anwendungsgebiet finden zu können.

3.1.2 Vergleich der NoSQL-Datenbanken

Zum Vergleich werden die vier populärsten NoSQL-Datenbanken der vier Kernklassen nach dem Ranking von DB-Engines ausgewählt. Die Popularität eines Systems wird mithilfe von Parametern, wie beispielsweise der Häufigkeit von Nennungen auf Websites oder der Häufigkeit von technischen Diskussionen über das System, aber auch dem allgemeinen Interesse an dem System, durch Messungen über Google Trends, ermittelt. Der Popularitätswert wird durch Normierung und Mittelung der einzelnen Parametern ermittelt. [Dbe18] Auf dieser Grundlage werden die folgenden NoSQL-Datenbanken der vier Kernklassen zum einzelnen Vergleich herangezogen:

- Spaltenorientierte Datenbank: Cassandra
- Dokumentenorientierte Datenbank: MongoDB
- Schlüssel-Wert Datenbank: Redis
- Graphenorientierte Datenbank: Neo4j

Nicht nur das Ranking von DB-Engines sieht diese NoSQL-Systeme als führend in ihren Kernklassen an, auch aktuelle internationale Studien und Konferenz bestätigen dies. [Gup17, S.293ff; Tan16, S.105ff] Wie zuvor erläutert ist jeweils eine NoSQL-Datenbank aus den vier Kernklassen ausgewählt und wird nach den in Kapitel 3.1.1 (Bestimmung der Vergleichskriterien) festgelegten Kriterien untereinander verglichen. Im Folgenden wird die Vergleichsmatrix zur Auswahl einer NoSQL-Datenbank (Tabelle 1-5) aufgeführt:

Vergleichsmatrix zur Auswahl einer NoSQL-Datenbank									
NoSQL-Datenbank		**Cassandra**		**MongoDB**		**Redis**		**Neo4j**	
Kernklasse		spaltenorientiert		dokumentenorientiert		Schlüssel-Wert		graphorientiert	
Vergleichs-kriterium	Gewicht in %								
Daten	25	Möglichkeit der Darstellung von semi-strukturierten Daten in Spaltenform	**3**	Möglichkeit der Darstellung der Daten im JSON-Format	**2**	Möglichkeit der Darstellung von einfachen Daten in Schlüssel-Wert-Paaren	**1**	Möglichkeit der Darstellung von stark zusammen-hängenden Daten mittels Graphen Theorie	**4**
Transaktion	15	Unterstützung von Atomizität und Isolation innerhalb einzelner Operationen	**1**	Atomare Operationen in einzelnen Dokumenten möglich	**1**	Atomare Ausführung von Befehlsblöcken und Skripten, optimistisches Sperren	**1**	ACID	**2**
Leistung	5	100k Daten in DB laden: 13,5 sek. Ausführung versch. Workloads: 5,4 sek.	**1**	100k Daten in DB laden: 10,3 sek. Ausführung versch. Workloads: 1,8 sek.	**2**	100k Daten in DB laden: 7,9 sek. Ausführung versch. Workloads: 1,1 sek.	**3**	x	**0**
Abfragen	15	Proprietäres Protokoll: Cassandra Query Language - CQL (SQL ähnliche Abfragesprache)	**1**	Proprietäres Protokoll basierend auf JSON	**1**	Proprietäres Protokoll: RESP - REdis Serialization Protocol	**1**	Cypher query language Java, Ruby API Python, Jython API REST API TinkerPop 3	**2**
Replikation	10	frei wählbare Replikationsfaktoren	**2**	Master-Slave Replikation	**1**	Master-Slave Replikation Multi-Master Replikation (nur mit zusätzl. Paket)	**1**	Cusual Clustering durch Raft protocol	**1**
Lizenz	10	Open Source (Apache) Kosten bei kommerzieller Nutzung	**1**	Open Source (AGPL) Kosten bei kommerzieller Nutzung	**1**	Open Source (BSD) Kosten bei kommerzieller Nutzung	**1**	Open Source (GPL) Kosten bei kommerzieller Nutzung	**1**
CAP	20	AP Eventual Consistency Immediate Consistency (individuell einstellbar bei jedem Schreibvorg.)	**2**	CP Eventual Consistency Immediate Consistency (individuell einstellbar bei jedem Schreibvorg.)	**2**	CP Eventual Consistency	**1**	Eventual Consistency konfigurierbar Immediate Consistency im Standalone-Modus	**2**
Summe	100	1,8		1,5		1,1		2,2	

Tabelle 3-2: Vergleichsmatrix zur Auswahl einer NoSQL-Datenbank

In Tabelle 3-2 (Vergleichsmatrix von NoSQL-Datenbanken) werden die einzelnen NoSQL-Systeme durch die Vergabe von Werten zwischen eins und vier (1 = gering, 4 = hoch) bewertet und somit untereinander verglichen. Dabei werden in der letzten Zeile der Tabelle alle Werte gewichtet und kumuliert zusammengefasst.

Wie in Kapitel 3.1.1 (Bestimmung der Vergleichskriterien) erwähnt, ist das Vergleichskriterium *‚Daten'* für diese Arbeit ein Schwerpunkt und findet deshalb bei der Gewichtung mit 25% besondere Rücksicht. Auch das Vergleichskriterium *‚CAP'* wird als Kernkriterium (Kapitel 2.3.1 Einführung in NoSQL-Datenbanken) bei der Auswahl von NoSQL-Datenbanken angesehen und hat daher 20% Gewichtung.

Die Vergleichskriterien *‚Transaktion'* und *‚Abfragen'* sind bei der Auswahl einer Datenbank aufgrund diverser Faktoren, die in Kapitel 2.3.3 (Vergleichskriterien zur Auswahl von NoSQL-Datenbanken) und Kapitel 3.1.1 (Bestimmung der Vergleichskriterien) erläutert wurden, von hoher Bedeutung und werden deswegen mit 15% gewichtet.

Die Vergleichskriterien ‚Replikation' und ‚Lizenz' haben vergleichsweise eine geringe Bedeutung bei dem Auswahlverfahren und werden folglich mit 10% gewichtet. Wie in Kapitel 3.1.1 (Festlegung der Vergleichskriterien) erläutert, dienen sie zum einen dem Verfahren der Skalierbarkeit des Systems und zum anderen dem wirtschaftlichem Nutzen.

Das Vergleichskriterium *‚Leistung'* hat mit 5% die geringste Gewichtung in der Vergleichsmatrix, da, wie in Kapitel 2.3.3 (Vergleichskriterien zur Auswahl von NoSQL-Datenbanken) erklärt, sich nur drei der vier Kernklassen hinsichtlich dieses Kriteriums untereinander vergleichen lassen und somit kein valider Vergleich aller vier Datenbanken erfolgen kann.

Nach der Analyse der einzelnen NoSQL-Systeme durch die Vergleichsmatrix (Tabelle 3-2) ist die graphenorientierte Datenbank Neo4j zur bedarfsgerechten Darstellung von Supply-Chain-Daten am besten geeignet.
Nun werden anhand ausgewählter Beispiele die Vergabe von einzelnen Werten erläutert. Im ersten Vergleichskriterium *‚Daten'* wird Neo4j mit dem höchstem Wert vier bewertet, weil graphenorientierte NoSQL-Datenbanken besonders stark miteinander in Beziehung stehende Daten darstellen können. Besonders bei Prozessen der Supply Chain, wo eine hohe Dichte von Lieferantenstrukturen vorherrscht, weisen die entstehenden Transaktionsdaten starke Verbindungen untereinander auf. Des Weiteren unterstützen graphenorientierte Datenbanken komplexe Datenstrukturen und unterliegen hierbei keinen Beschränkungen, was aufgrund ihrer Schemafreiheit ermöglicht wird. Zusätzlich ist hierdurch die Integrierung neuer Daten besonders einfach, was bei stetigem Wandel von

Systemen und Geschäftsprozessen unerlässlich ist. Besonders ist auch die einfache Datenmodellierung hervorzuheben, die durch einfache graphische Darstellungen mit Pfeilen und Elementen visualisiert werden kann. Im zweiten Vergleichskriterium *‚Transaktion‘* wird Neo4j am besten bewertet, da es nach Beendigung einer Transaktionen die Daten im konsistenten Zustand hinterlässt, was dem Konsistenzmodell ACID entspricht. Gegenüber sensiblen Finanzdaten müssen Transaktionsdaten nicht unbedingt strikt konsistent gehalten werden, jedoch bringt es Vorteile mit sich, wie die Einfachheit von ACID Transaktionen und einer verlässlichen Datenbasis. Zusätzlich wird im letzten Vergleichskriterium *‚CAP‘* deutlich, dass es Neo4j möglich ist, die Konfigurierung von Eventual Consistency, zu unterstützen. Auch hier wird Neo4j mit zwei bewertet. Im fünften Vergleichskriterium ‚Abfragen‘ bietet Neo4j eine eigene, SQL ähnliche, Abfragesprache Cypher an und weist außerdem die meisten Schnittstellen (APIs) zu anderen Sprachen auf.
Somit ist die Datenbank hier wieder am besten bewertet und ist im Vergleich zu den drei anderen NoSQL-Systemen mit dem gewichteten und kumulierten Wert von 2,2 über 19% besser bewertet als die zweitplatzierte spaltenorientierte Datenbank Cassandra (1,8). Gegenüber der dokumentenorientierten Datenbank MongoDB (1,5) schneidet Neo4j mit über 32% besser ab und die Schlüssel-Wert-Datenbank Redis (1,1) ist dieser Vergleichsmatrix letztplazierte Datenbank.
Zusammenfassend ist zu sagen, dass die graphenorientierte NoSQL-Datenbank Neo4j optimalerweise für die Darstellung von Supply-Chain-Daten, im speziellem Transaktionsdaten, geeignet ist. Sie bietet eine breite Community mit zahlreichem Support an und war mit ihrer ersten Version 2010 marktreif, womit sie First Mover und somit Pionierleistungen im Bereich von Graphen-Datenbanken geleistet hat. Dies spiegelt sich vor allem in der einfachen Handhabung sowie der übersichtlichen Darstellung von Information in Neo4j wider.

3.2 Neo4j NoSQL-Datenbank

In diesem Kapitel wird die ausgewählte graphenorientierte NoSQL-Datenbank Neo4j zur Darstellung eines beispielhaften Transaktionsdatensatzes verwendet. Ziel ist es, eine Supply Chain vom Rohstofflieferanten zum Endkunden mit den jeweiligen Beziehungen untereinander mithilfe der graphischen Oberfläche von Neo4j darzustellen.
Zu Anfang wird die Abfragesprache Cypher und ihre gängige Abfragesyntax vorgestellt. Hier nach wird ein graphisches Datenmodell der Supply Chain erstellt, wo die jeweiligen Knoten, wie ‚*Lieferant*', mit den zugehörigen Attributen, wie ‚*Kosten*', über die Kanten ‚*Liefert*' verknüpft werden. Anschließend wird das Datenmodell über Cypher in Neo4j geladen und über verschiedene Abfragen, wie beispielsweise der Abfrage nach dem kürzesten Pfad zwischen variablen Knoten, werden nochmals die Vorteile von Neo4j verdeutlicht.

3.2.1 Exemplarische Darstellung

Zu Anfang ist zu sagen, dass die Neo4j Inc. zwei Möglichkeiten zur Verwendung der Datenbank anbieten, zum einen die Desktop-Version und zum anderen eine Browser-Version ‚Neo4j Sandbox Online'. In dieser Arbeit wird ‚Neo4j Desktop Version 1.1.9' verwendet. Der grundlegende Aufbau der Abfragesprache Cypher (CQL) ähnelt der bekannten, in relationalen Datenbanken verwendeten, Abfragesprache SQL. Hier ist der grundsätzliche Aufbau über die Befehle:

- SELECT
- FROM
- WHERE

möglich. In Cypher geschieht dies über die grundlegenden Befehle:

- MATCH
- WHERE
- RETURN

Zusätzlich können über den Befehl CREATE Knoten mit einer beliebigen Anzahl an Attributen erstellt werden. Zwei Knoten werden mit folgendem Befehl über die Kante ‚*LIEFERT*' miteinander verbunden `(Knoten1) -[:LIEFERT]-> (Knoten2)`.

In Abbildung 3-1 (Graphisches Datenmodell einer beispielhaften Supply Chain) wird der Aufbau der Supply Chain mit den jeweiligen Knoten, den dazugehörigen Attributen, dargestellt und über Kanten miteinander verbunden. Jeder Knoten hat das ihn eindeutig identifizierende Attribut *‚id‘* und bekommt ein Attribut *‚name‘* vergeben. Zusätzlich hat jeder Knoten einen Ort, der über Attribute *‚lat’* (latitude) und *‚lon’* (longitude), Längen- und Breitengrade, angegeben wird. Bis auf den Knoten *‚Kunde‘* haben alle anderen Knoten zusätzlich die Attribute *‚cost‘*, *‚time‘* und *‚co2‘* - welche die Kosten des jeweiligen Knotens, die Zeit im jeweiligen Knoten und den dazugehörigen CO2-Ausstoß beschreiben. Alle Werte bis auf *‚id‘* werden zufällig vergeben.
Wie auf Seite 14 in Abbildung 2-5 (Übersicht der Kategorien von Supply-Chain-Daten) beschrieben, werden Supply-Chain-Daten hinsichtlich Transaktionsdaten abgegrenzt und in drei Kategorien aufgeteilt. Um in dieser beispielhaften Supply Chain Transaktionsdaten zu verwenden, werden jedem Teilnehmer der Supply Chain die Attribute *‚lat’ und ‚lon’* zugeordnet. Sie gehören zu den Transaktionsstrukturdaten, da sie die Lieferadresse des jeweiligen Teilnehmers eindeutig beschreiben. Das Attribut *‚cost‘* gehört zu den Transaktionsaktivitätsdaten und gibt den Preis wider. Das Attribut *‚time‘* gehört zu den Transaktionskontrolldaten und gibt den Zeitstempel wider.

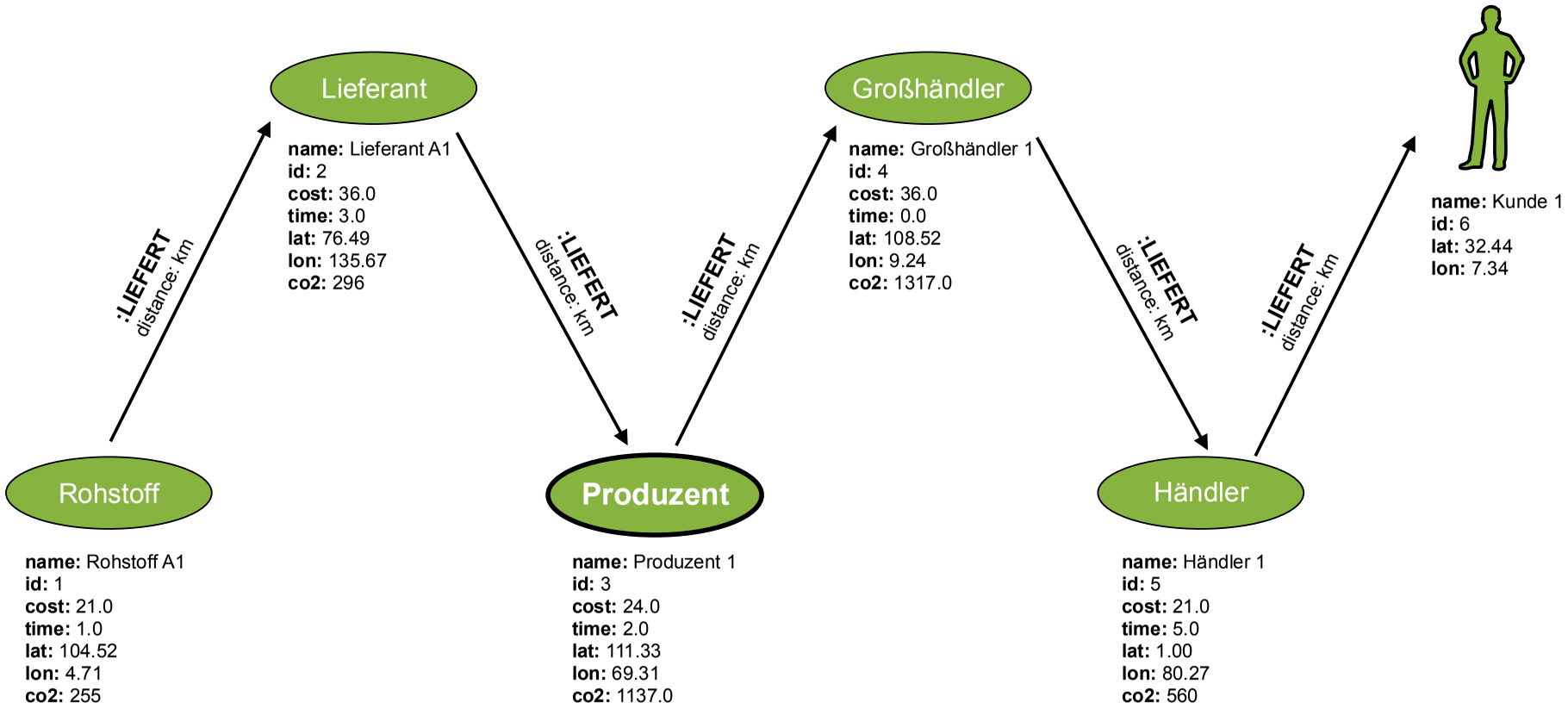

Abbildung 3-1: Graphisches Datenmodell einer beispielhaften Supply Chain

Nun werden die einzelnen Knoten und Kanten in Cypher über den Befehl CREATE erzeugt und in Neo4j geladen. Die Syntax sieht wie folgt aus:

- Variable : Parameter
- (Klasse)
- {Attribut}
- (Bedingung) | (Inhalt)

Um im späteren Verlauf die Anzahl der Knoten beliebig skalieren zu können wird mit dem Befehl FOREACH wird eine Bedingung erzeugt die durch den Befehl ‚range' ein Spectrum zwischen selbst definierten Grenzen erzeugt. Durch den Parameter r wird hinter jeden Namen des Knotens ein Wert editiert und so mehrere Knoten in dem zuvor ausgewählten Spektrum erzeugt.

```
// Supply Chain (Added 30 labels, created 30 nodes, set 144 properties) //
FOREACH (r IN range(1,5)|
CREATE (:rohlieferanta{ name:"Rohstoff A" + r, cost: round(exp(rand()*3)+20),
co2: round(exp(rand()*8)+250), lat: tan(rand())*100, lon: tan(rand())*100, time: round(rand()*5)}))

FOREACH (r IN range(1,3)|
CREATE (:rohlieferantb{ name:"Rohstoff B" + r, cost: round(exp(rand()*3)+20), co2:
round(exp(rand()*8)+250), lat: tan(rand())*100, lon: tan(rand())*100, time:
round(rand()*5)}))

FOREACH (r IN range(1,2)|
CREATE (:lieferanta { name:"Lieferant A" + r, cost: round(exp(rand()*3)+20), co2:
round(exp(rand()*8)+250), lat: tan(rand())*100, lon: tan(rand())*100, time:
round(rand()*5)}))

FOREACH (r IN range(1,3)|
CREATE (:lieferantb { name:"Lieferant B" + r, cost: round(exp(rand()*3)+20), co2:
round(exp(rand()*8)+250), time: round(rand()*5)}))

FOREACH (r IN range(1,1)|
CREATE (:produzent { name: "Produzent" + r, lat: tan(rand())*100, lon: tan(rand())*100,
co2: round(exp(rand()*8)+250), cost: round(exp(rand()*3)+20), time: round(rand()*5) }))

FOREACH (r IN range(1,2)|
CREATE (:grosshandler { name: "Grosshändler" + r, lat: tan(rand())*100, lon: tan(rand())*100,
co2: round(exp(rand()*8)+250), cost: round(exp(rand()*3)+20), time: round(rand()*5)}))

FOREACH (r IN range(1,4)|
CREATE (:handler { name:"Händler" + r, cost: round(exp(rand()*3)+20), co2:
round(exp(rand()*8)+250), lat: tan(rand())*100, lon: tan(rand())*100, time:
round(rand()*5)}))

FOREACH (r IN range(1,10)|
CREATE (:kunde { name:"Kunde" + r, lat: tan(rand())*100, lon: tan(rand())*100}))
```

Abbildung 3-2: Cypher Code zur Erstellung der Supply Chain in Neo4j

Den Attributen *cost, co2, lat, lon* und *time* der Knoten werden verschiedene, zufällig berechneten Werte zugeordnet, mit dem Befehl ‚round' gerundet und dem entsprechende Knoten hinzugefügt. Somit erzeugt Neo4j 30 Knoten mit 144 Attributen.

Nun werden mit den Befehlen MATCH und CREATE UNIQUE die Beziehungen zwischen den Knoten untereinander modelliert. Der Befehlszusatz UNIQUE sorgt dafür, dass es keine Duplikate in den Beziehungen gibt. Mit UNIQUE entstehen in diesem Beispiel 74 Beziehungen, wobei ohne UNIQUE die Anzahl der Beziehungen auf 15520 Beziehungen ansteigt. Dies kommt daher, dass alle möglichen Pfade von einem Startknoten A zu einem Zielknoten B dargestellt werden.

```
36  // Relationships (74 relationships) //
37  MATCH (la:lieferanta), (p:produzent), (gh:grosshandler), (h:handler), (k:kunde)
38  CREATE UNIQUE (la)-[:LIEFERT]->(p)-[:LIEFERT]->(gh)-[:LIEFERT]->(h)-[:LIEFERT]->(k)
39  WITH p, la
40  MATCH (lb:lieferantb) CREATE UNIQUE (lb)-[:LIEFERT]->(p)
41  WITH lb, la
42  MATCH (ra:rohlieferanta), (rb:rohlieferantb)
43  CREATE UNIQUE (ra)-[:LIEFERT]->(la)
44  CREATE UNIQUE (rb)-[:LIEFERT]->(lb)
```

Abbildung 3-3: Cypher Code zur Erstellung der Beziehungen

Durch den Befehl MATCH und RETURN wird die Supply Chain in Neo4j dargestellt und in Abbildung 3-5 (Visualisierung der gesamten Supply Chain in Neo4j) visualisiert.

```
46  // Visualization //
47  MATCH (a)-[r]-()
48  RETURN a, r
```

Abbildung 3-4: Cypher Code zur Visualisierung der Supply Chain

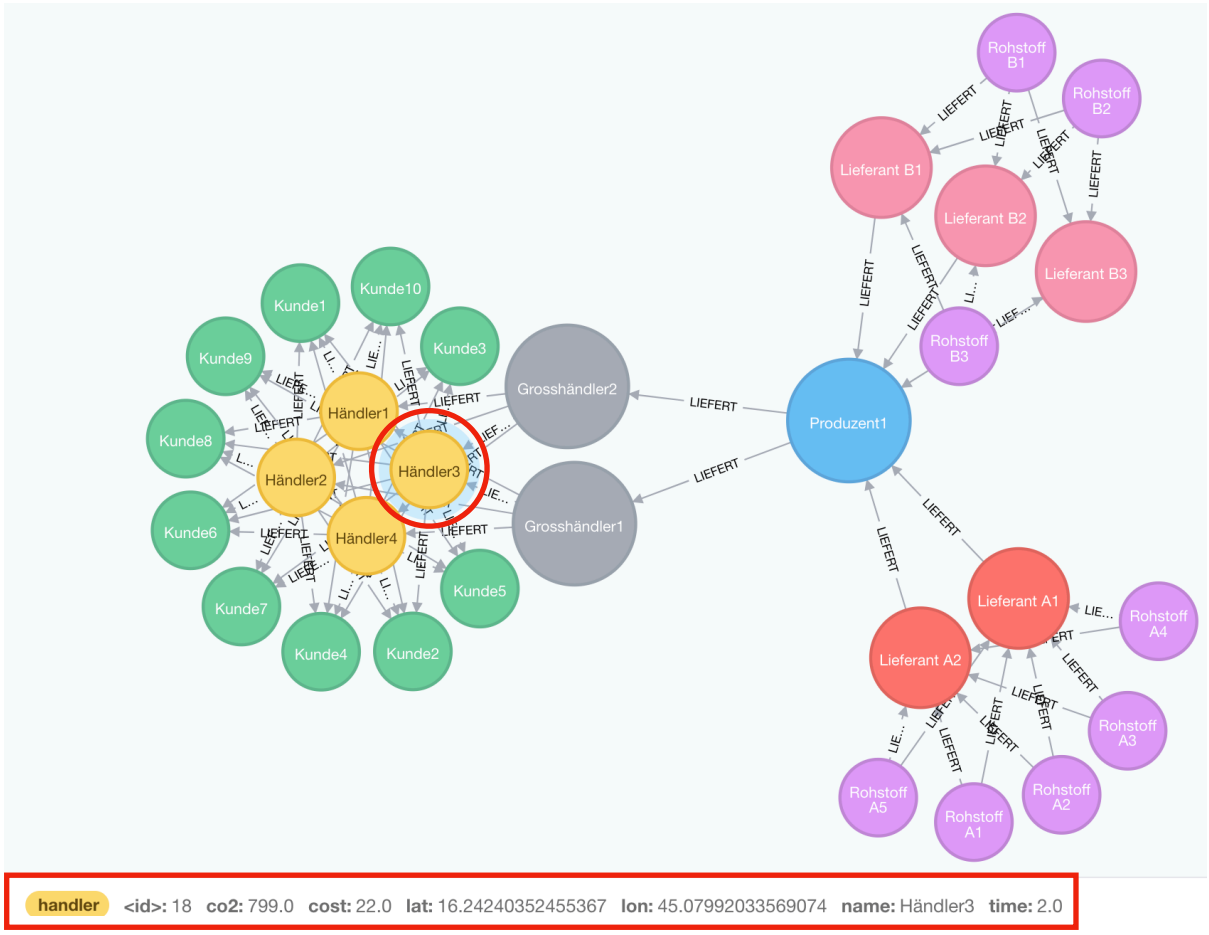

Abbildung 3-5: Visualisierung der gesamten Supply Chain in Neo4j

Durch Bewegen des Mauszeigers auf einen Knoten, hier beispielsweise ‚Händler 3', werden unten die einzelnen zugeordneten Attribute und die *‚id'*, die diesen Knoten eindeutig identifiziert, aufgeführt.

Die automatische Anordnung der so ausgegeben graphische Darstellung der Supply Chain hat den Vorteil, dass beispielsweise bei einer hohen Anzahl an Rohstofflieferanten und Lieferanten ihre Beziehungsintesität visuell sichtbar gemacht wird. Nachteilig ist, dass die einzelnen Knoten von Neo4j nicht der logischen Supply-Chain-Anordnung der Teilnehmer, vom Rohstofflieferanten hin zum Kunden, entspricht. Nur durch eine manuelle Anordnung lassen sich die einzelnen Knoten fixieren, um so die Supply Chain visuell übersichtlich dazustellen. In Abbildung 3-6 (Logische Anordnung der einzelnen Teilnehmer der Supply Chain in Neo4j) wird dies per ‚drag & drop' realisiert.

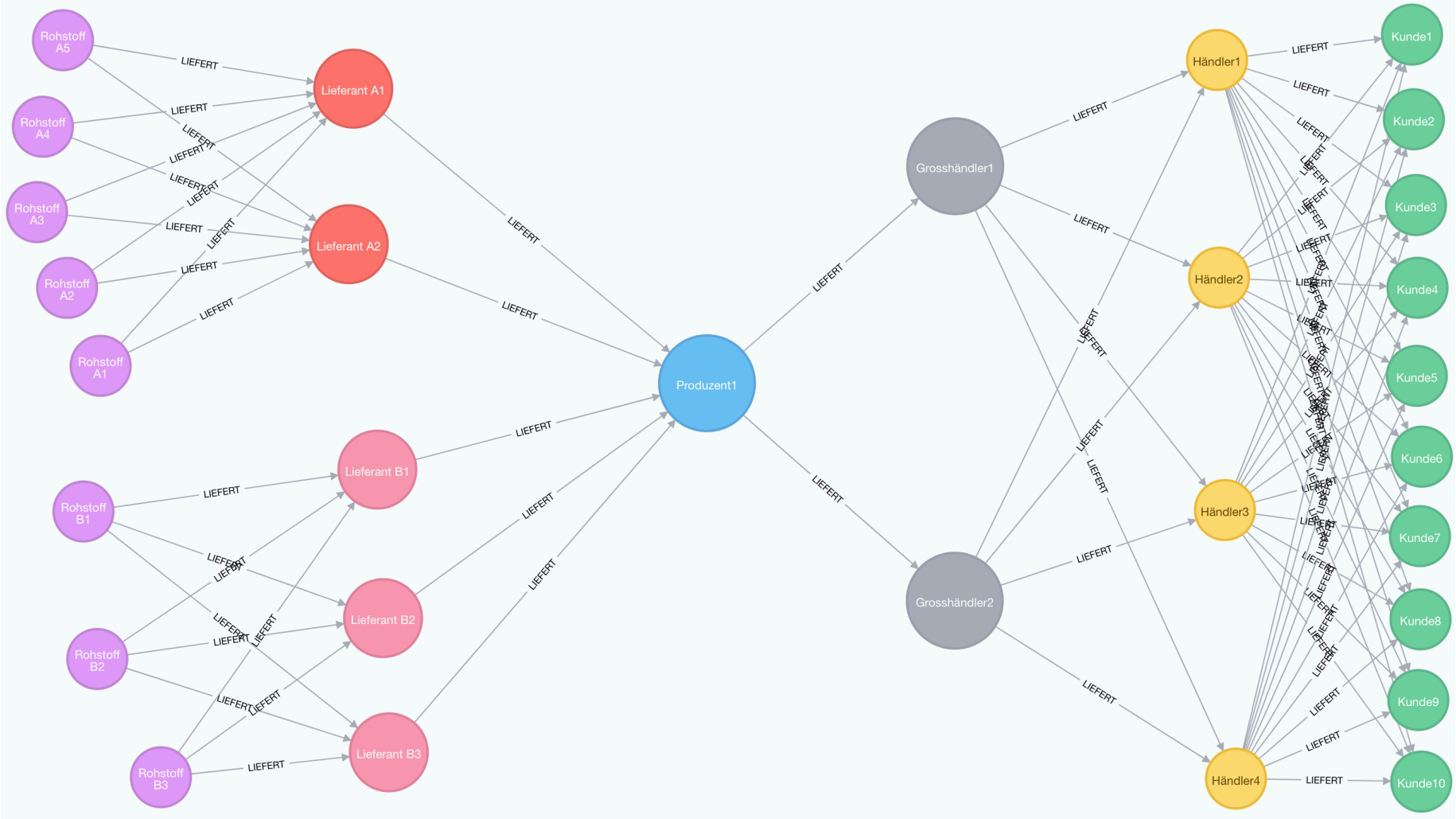

Abbildung 3-6: Logische Anordnung der einzelnen Teilnehmer der Supply Chain in Neo4j

Hiernach wird Distanz zwischen zwei Knoten berechnet und der jeweiligen Kante zugeordnet.

```
// Distance //
MATCH (a)-[r]->(b)
WITH r, a, b, 2 * 6371 * asin(sqrt(haversin(radians(toInt(a.lat) - toInt(b.lat))) +
cos(radians(a.lat))* cos(radians(b.lat))* haversin(radians(a.lon - b.lon))))
AS dist SET r.km = round(dist)
```

Abbildung 3-7: Cypher Code zur Berechnung der Distanz zwischen zwei Knoten und Kantenzuordnung

Somit ist die Supply Chain mit Transaktionsdaten und den jeweiligen Restriktionen vollständig in Neo4j modelliert. Nun werden folgende Abfragen erstellt:

- Klassisches Transportproblem: besten Großhändler finden, der die kürzesten Distanzen zwischen den einzelnen Knoten hat,
- schnellsten Pfad (über die Zeit) vom Rohstofflieferanten zum Händler,
- günstigsten Pfad (über die Kosten) vom Rohstofflieferanten zum Händler,
- umweltfreundlichsten Pfad (über den CO2-Ausstoß) vom Rohstofflieferanten zum Händler,
- Zusammengefügte Abfragen: schnellster und günstigster Pfad (über die Zeit und Kosten) vom Rohstofflieferanten zum Händler (geordnet nach Zeit),
- Zusammengefügte Abfragen: günstigster und kürzester Pfad (über die Kosten und Distanz) vom Rohstofflieferanten zum Händler (geordnet nach Zeit).

Um den besten Großhändler zu finden, werden zum einen die Distanzen vom Produzent zum Großhändler (r1) und vom Großhändler zum Händler (r2) aufsummiert, gespeichert als *‚Total_Distance‘,* und zum anderen wird der Mittelwert gebildet und als *‚Average_Distance’* gespeichert. Hiernach werden über den Befehl RETURN die Großhändler, geordnet nach der gesamten Distanz wiedergegeben. Der Quellcode in Cypher sieht wie folgt aus:

```
// Find shortest //
MATCH (p:produzent)-[r1]->(w)-[r2]->(h:handler)
WITH DISTINCT(substring(w.name, 12)) AS Num,
 avg(r1.km + r2.km) AS Average_Distance,
 sum(r1.km + r2.km) AS Total_Distance
RETURN "Grosshandler" + Num AS grosshandler, Total_Distance, round(Average_Distance)
ORDER BY Total_Distance
```

Abbildung 3-8: Cypher Code zum Transportproblem

Als Ergebnis gib Neo4j eine Liste der besten Großhändler aus, diese lässt sich als CSV-Datei lokal speichern. Abbildung 3-9 (Neo4j Ausgabe des besten Großhändlers) zeigt die nach der gesamten Distanz geordnete Ausgabe:

grosshandler	Total_Distance	round(Average_Distance)
"Grosshandler1"	34102.0	8526.0
"Grosshandler2"	47742.0	11936.0

Abbildung 3-9: Neo4j Ausgabe des besten Großhändlers

Als nächstes wird eine Abfrage zur Findung des schnellsten Pfades entlang der gesamten Supply Chain erstellt:

```
64   // Find fastest //
65   MATCH chain=(rl:rohlieferanta)-[r*]->(h:handler)
66   WITH reduce(wait = 0, s IN nodes(chain)| wait + s.time) AS waitTime, chain
67   WITH extract(n IN nodes(chain)| n.name) AS SupplyChain, waitTime
68   ORDER BY waitTime
69   RETURN SupplyChain, waitTime
```

Abbildung 3-10: Cypher Code zum schnellsten Pfad über Rohstofflieferant A

Zu Anfang wird über den Befehl MATCH eine Supply Chain vom Rohstofflieferanten A zum Händler definiert und als *‚chain‘* gespeichert. Hiernach werden über den Befehl WITH mehrere Abfragen verbunden. Über die Funktionen ‚reduce’ wird die Wartezeit *‚waitTime‘* und über die Funktion *‚extract‘* wird die Supply Chain *‚SupplyChain‘* definiert. Zum Schluss der Abfrage wird nach der Wartezeit geordnet und eine Liste mit den einzelnen Pfaden der Supply Chain und den dazugehörigen Zeiten ausgegeben. In Abbildung 3-11 ist ein Ausschnitt der gesamten Liste, die aus insgesamt 80 möglichen Pfaden besteht, dargestellt.

SupplyChain	waitTime
["Rohstoff A1", "Lieferant A1", "Produzent1", "Grosshändler1", "Händler2"]	6.0
["Rohstoff A1", "Lieferant A1", "Produzent1", "Grosshändler2", "Händler2"]	6.0
["Rohstoff A3", "Lieferant A1", "Produzent1", "Grosshändler1", "Händler2"]	7.0
["Rohstoff A3", "Lieferant A1", "Produzent1", "Grosshändler2", "Händler2"]	7.0
["Rohstoff A1", "Lieferant A1", "Produzent1", "Grosshändler1", "Händler1"]	8.0
["Rohstoff A1", "Lieferant A1", "Produzent1", "Grosshändler2", "Händler1"]	8.0
["Rohstoff A1", "Lieferant A2", "Produzent1", "Grosshändler1", "Händler2"]	8.0
["Rohstoff A1", "Lieferant A2", "Produzent1", "Grosshändler2", "Händler2"]	8.0
["Rohstoff A3", "Lieferant A1", "Produzent1", "Grosshändler1", "Händler1"]	9.0
["Rohstoff A3", "Lieferant A1", "Produzent1", "Grosshändler2", "Händler1"]	9.0

Abbildung 3-11: Neo4j Ausgabe zum schnellsten Pfad über Rohstofflieferant A

Durch Änderung der Abfrage in der ersten Zeile zu `chain=(rl:rohlieferantb)` wird der schnellste Pfad über den Rohstofflieferanten B ausgegeben:

SupplyChain	waitTime
["Rohstoff B1", "Lieferant B1", "Produzent1", "Grosshändler1", "Händler2"]	7.0
["Rohstoff B2", "Lieferant B1", "Produzent1", "Grosshändler1", "Händler2"]	8.0
["Rohstoff B3", "Lieferant B1", "Produzent1", "Grosshändler1", "Händler2"]	8.0
["Rohstoff B1", "Lieferant B2", "Produzent1", "Grosshändler1", "Händler2"]	9.0
["Rohstoff B1", "Lieferant B1", "Produzent1", "Grosshändler1", "Händler3"]	9.0
["Rohstoff B1", "Lieferant B1", "Produzent1", "Grosshändler2", "Händler2"]	9.0
["Rohstoff B1", "Lieferant B3", "Produzent1", "Grosshändler1", "Händler2"]	10.0
["Rohstoff B1", "Lieferant B1", "Produzent1", "Grosshändler1", "Händler1"]	10.0

Abbildung 3-12: Neo4j Ausgabe zum schnellsten Pfad über Rohstofflieferant B

Somit ist der Pfad der Supply Chain über den Rohstofflieferanten A1, zum Lieferanten A1, über den Produzenten, zum Großhändler 1 und schließlich zum Händler 1 mit einer gesamt Zeit von 6.0 am schnellsten und für den Produzent zu präferieren.

Nun soll der günstigste Pfad der Supply Chain gefunden und ausgegeben werden. Dies geschieht analog zum Quellcode des schnellsten Pfades:

```
71   // Find cheapest //
72   MATCH chain=(rl:rohlieferanta)-[r*]->(h:handler)
73   WITH reduce(costs = 0, s IN nodes(chain)| costs + s.cost) AS totalCosts, chain
74   WITH extract(n IN nodes(chain)| n.name) AS SupplyChain, totalCosts
75   RETURN SupplyChain, totalCosts
76   ORDER BY totalCosts
```

Abbildung 3-13: Cypher Code zum günstigsten Pfad über Rohstofflieferant A

Neo4j gibt eine Liste mit einzelnen Pfaden der Supply Chain über den Rohstofflieferanten A als Ergebnis aus:

SupplyChain	totalCosts
["Rohstoff A5", "Lieferant A1", "Produzent1", "Grosshändler1", "Händler3"]	114.0
["Rohstoff A5", "Lieferant A2", "Produzent1", "Grosshändler1", "Händler3"]	114.0
["Rohstoff A5", "Lieferant A1", "Produzent1", "Grosshändler1", "Händler1"]	115.0
["Rohstoff A5", "Lieferant A2", "Produzent1", "Grosshändler1", "Händler1"]	115.0
["Rohstoff A5", "Lieferant A1", "Produzent1", "Grosshändler1", "Händler2"]	115.0
["Rohstoff A5", "Lieferant A2", "Produzent1", "Grosshändler1", "Händler2"]	115.0
["Rohstoff A1", "Lieferant A1", "Produzent1", "Grosshändler1", "Händler3"]	117.0
["Rohstoff A1", "Lieferant A2", "Produzent1", "Grosshändler1", "Händler3"]	117.0

Abbildung 3-14: Neo4j Ausgabe zum günstigsten Pfad über Rohstofflieferant A

SupplyChain	co2
["Rohstoff B3", "Lieferant B1", "Produzent1", "Grosshändler1", "Händler2"]	1940.0
["Rohstoff B3", "Lieferant B1", "Produzent1", "Grosshändler1", "Händler4"]	1996.0
["Rohstoff B3", "Lieferant B3", "Produzent1", "Grosshändler1", "Händler2"]	2234.0
["Rohstoff B3", "Lieferant B3", "Produzent1", "Grosshändler1", "Händler4"]	2290.0
["Rohstoff B1", "Lieferant B1", "Produzent1", "Grosshändler1", "Händler2"]	2325.0

Abbildung 3-18: Neo4j Ausgabe zum umweltfreundlichsten Pfad über Rohstofflieferant B

Analog lässt sich der Quellcode für die Pfade vom Rohstofflieferanten B umschreiben und als Ergebnis folgende Liste von Neo4j ausgeben:

SupplyChain	totalCosts
["Rohstoff B1", "Lieferant B2", "Produzent1", "Grosshändler2", "Händler3"]	111.0
["Rohstoff B2", "Lieferant B2", "Produzent1", "Grosshändler2", "Händler3"]	111.0
["Rohstoff B1", "Lieferant B2", "Produzent1", "Grosshändler2", "Händler2"]	113.0
["Rohstoff B1", "Lieferant B3", "Produzent1", "Grosshändler2", "Händler3"]	113.0
["Rohstoff B2", "Lieferant B2", "Produzent1", "Grosshändler2", "Händler2"]	113.0

Abbildung 3-15: Neo4j Ausgabe zum günstigsten Pfad über Rohstofflieferant B

Somit ist der Pfad der Supply Chain über den Rohstofflieferanten B1, zum Lieferanten B2, über den Produzenten, zum Großhändler 2 und schließlich zum Händler 3 mit Kosten von 111.0 am günstigsten und für den Produzent zu präferieren.

Zusätzlich soll der umweltfreundlichste Pfad der Supply Chain ermittelt werden. Hierzu wird folgende Abfrage in Cypher erstellt:

```
78   // Find co2-lowest //
79   MATCH chain=(rl:rohlieferanta)-[r*]->(h:handler)
80   WITH reduce(co2 = 0, s IN nodes(chain)| co2 + s.co2) AS co2, chain
81   WITH extract(n IN nodes(chain)| n.name) AS SupplyChain, co2
82   RETURN SupplyChain, co2
83   ORDER BY co2
```

Abbildung 3-16: Cypher Code zum umweltfreundlichsten Pfad über Rohstofflieferant A

Als Ergebnis liefert Neo4j eine Liste der Pfade der Supply Chain, mit aufsummierten CO2-Ausstößen jedes einzelnen Teilnehmers und gibt den Pfad mit der geringsten Umweltbelastung aus.

SupplyChain	co2
["Rohstoff A2", "Lieferant A2", "Produzent1", "Grosshändler1", "Händler2"]	2589.0
["Rohstoff A5", "Lieferant A2", "Produzent1", "Grosshändler1", "Händler2"]	2606.0
["Rohstoff A2", "Lieferant A2", "Produzent1", "Grosshändler1", "Händler4"]	2645.0
["Rohstoff A5", "Lieferant A2", "Produzent1", "Grosshändler1", "Händler4"]	2662.0
["Rohstoff A1", "Lieferant A2", "Produzent1", "Grosshändler1", "Händler2"]	2702.0

Abbildung 3-17: Neo4j Ausgabe zum umweltfreundlichsten Pfad über Rohstofflieferant A

Der umweltfreundlichste Pfad über den Rohstofflieferanten B ist in Abbildung 3-18 dargestellt.
Somit ist der Pfad über den Rohstofflieferanten B3, zum Lieferanten B1, über den Produzenten, zum Großhändler 1 und schließlich zum Händler 2 mit einem CO2-Ausstoß von 1940.0 am umweltfreundlichsten.

Nun wird eine zusammengefügte Abfrage nach dem schnellsten und gleichzeitig günstigsten Pfad der Supply Chain ausgehend vom Rohstofflieferanten A in Cypher erstellt:

```
81  // Fastest + Cheapest //
82  MATCH chain=(rl:rohlieferanta)-[r*]->(h:handler)
83  WITH reduce(wait = 0, s IN nodes(chain)| wait + s.time) AS waitTime, chain
84  WITH reduce(costs = 0, s IN nodes(chain)| costs + s.cost) AS totalCosts, waitTime, chain
85  WITH extract(n IN nodes(chain)| n.name) AS SupplyChain, waitTime, totalCosts
86  RETURN SupplyChain, waitTime, totalCosts
87  ORDER BY waitTime, totalCosts
```

Abbildung 3-19: Cypher Code zum schnellsten und günstigsten Pfad über Rohstofflieferant A

Als Ergebnis wird von Neo4j eine Liste mit folgendem Pfad, wobei die Schnelligkeit präferiert wird und die Kosten sekundär sind, ausgegeben.

SupplyChain	waitTime	totalCosts
["Rohstoff A1", "Lieferant A2", "Produzent1", "Grosshändler1", "Händler2"]	7.0	120.0
["Rohstoff A5", "Lieferant A2", "Produzent1", "Grosshändler1", "Händler2"]	7.0	122.0
["Rohstoff A1", "Lieferant A1", "Produzent1", "Grosshändler1", "Händler2"]	7.0	129.0
["Rohstoff A5", "Lieferant A1", "Produzent1", "Grosshändler1", "Händler2"]	7.0	131.0
["Rohstoff A1", "Lieferant A2", "Produzent1", "Grosshändler2", "Händler2"]	9.0	116.0

Abbildung 3-20: Neo4j Ausgabe zum schnellsten und günstigsten Pfad über Rohstofflieferant A

SupplyChain	waitTime	totalCosts
["Rohstoff B1", "Lieferant B1", "Produzent1", "Grosshändler1", "Händler2"]	7.0	125.0
["Rohstoff B2", "Lieferant B1", "Produzent1", "Grosshändler1", "Händler2"]	8.0	125.0
["Rohstoff B3", "Lieferant B1", "Produzent1", "Grosshändler1", "Händler2"]	8.0	131.0
["Rohstoff B1", "Lieferant B2", "Produzent1", "Grosshändler1", "Händler2"]	9.0	117.0
["Rohstoff B1", "Lieferant B1", "Produzent1", "Grosshändler2", "Händler2"]	9.0	121.0

Abbildung 3-24: Neo4j Ausgabe zum günstigsten und kürzesten Pfad über Rohstofflieferant B

Somit ist der Pfad über den Rohstofflieferanten A1, zum Lieferanten A2, über den Produzenten, zum Großhändler 1 und schließlich zum Händler 2 gleich schnell, mit

einer Zeit von 7.0, wie vom Rohstofflieferanten B ausgehend. Die Kosten liegen bei 120.0 und sind somit um 5.0 günstiger als die Kosten des Pfades ausgehend von Rohstofflieferant B.
Jetzt wird eine zusammengefügte Abfrage nach dem kostengünstigsten und gleichzeitig kürzesten Pfad der Supply Chain ausgehend vom Rohstofflieferanten A in Cypher erstellt. Analog zum vorherigen Beispiel lautet der Quellcode wie folgt:

```
114  // Cheaptest + Shortest //
115  MATCH chain=(rl:rohlieferanta)-[r*]->(k:kunde)
116  WITH reduce(costs = 0, s IN nodes(chain)| costs + s.cost) AS totalCosts, chain
117  WITH reduce(dist = 0, s IN relationships(chain)| dist + s.km) AS distance, totalCosts, chain
118  WITH extract(n IN nodes(chain)| n.name) AS SupplyChain, distance, totalCosts
119  RETURN SupplyChain, totalCosts, distance
120  ORDER BY totalCosts, distance
```

Abbildung 3-22: Cypher Code zum günstigsten und kürzesten Pfad über Rohstofflieferant A

Als Ergebnis liefert Neo4j folgende Liste der Pfade, wobei die Kosten vor den Distanzen präferiert werden:

SupplyChain	totalCosts	distance
["Rohstoff A3", "Lieferant A1", "Produzent1", "Grosshändler1", "Händler2", "Kunde3"]	133.0	33342.0
["Rohstoff A3", "Lieferant A1", "Produzent1", "Grosshändler1", "Händler2", "Kunde6"]	133.0	33722.0
["Rohstoff A3", "Lieferant A1", "Produzent1", "Grosshändler1", "Händler2", "Kunde7"]	134.0	31053.0
["Rohstoff A3", "Lieferant A1", "Produzent1", "Grosshändler1", "Händler3", "Kunde3"]	134.0	31958.0
["Rohstoff A3", "Lieferant A1", "Produzent1", "Grosshändler1", "Händler1", "Kunde3"]	134.0	33448.0

Abbildung 3-23: Neo4j Ausgabe zum günstigsten und kürzesten Pfad über Rohstofflieferant A

Analog ist die Ausgabe des Pfades über den Rohstofflieferanten B wie folgt:

SupplyChain	totalCosts	distance
["Rohstoff B1", "Lieferant B2", "Produzent1", "Grosshändler1", "Händler2", "Kunde3"]	131.0	36790.0
["Rohstoff B1", "Lieferant B2", "Produzent1", "Grosshändler1", "Händler2", "Kunde6"]	131.0	37170.0
["Rohstoff B3", "Lieferant B2", "Produzent1", "Grosshändler1", "Händler2", "Kunde3"]	132.0	28887.0
["Rohstoff B3", "Lieferant B2", "Produzent1", "Grosshändler1", "Händler2", "Kunde6"]	132.0	29267.0
["Rohstoff B1", "Lieferant B2", "Produzent1", "Grosshändler1", "Händler2", "Kunde7"]	132.0	34501.0

Abbildung 3-25: Cypher Code zur skalierten Supply Chain

Zusammenfassend lässt sich sagen, dass der Pfad über den Rohstofflieferanten B1, zum Lieferanten B2, über den Produzenten, zum Großhändler 1, zum Händler 2 und

schließlich zum Kunden 3 mit Gesamtkosten von 131.0 am günstigsten und kürzesten ist.

Darüber hinaus wird nun die Darstellung der Supply Chain mit einer Vielzahl an Teilnehmern demonstriert. Dazu wird im Quellcode das Spektrum der einzelnen Teilnehmer erhöht. Bei einer Erhöhung auf über hundert Teilnehmer liefert Neo4j die Fehlermeldung *‚out of memory‘,* was auf die Restriktion von 1GB RAM-Speicher schließen lässt. Selbst nach Erhöhung auf 2GB, 3GB oder 4GB wird die selbe Fehlermeldung ausgegeben. Aus diesem Grund wird die Supply Chain ohne den Pfad über Rohstofflieferant B demonstriert. Der Cypher Quellcode lautet wir folgt:

```
// Supply Chain //
FOREACH (r IN range(1,30)|
CREATE (:rohlieferanta{ name:"Rohstoff A" + r, cost: round(exp(rand()*3)+20),
co2: round(exp(rand()*8)+250), lat: tan(rand())*100, lon: tan(rand())*100, time:
round(rand()*5)}))

//FOREACH (r IN range(1,30)|
//CREATE (:rohlieferantb{ name:"Rohstoff B" + r, cost: round(exp(rand()*3)+20), co2:
//round(exp(rand()*8)+250), lat: tan(rand())*100, lon: tan(rand())*100, time:
//round(rand()*5)}))

FOREACH (r IN range(1,10)|
CREATE (:lieferanta { name:"Lieferant A" + r, cost: round(exp(rand()*3)+20), co2:
round(exp(rand()*8)+250), lat: tan(rand())*100, lon: tan(rand())*100, time:
round(rand()*5)}))

//FOREACH (r IN range(1,10)|
//CREATE (:lieferantb { name:"Lieferant B" + r, cost: round(exp(rand()*3)+20), co2:
//round(exp(rand()*8)+250), time: round(rand()*5)}))

FOREACH (r IN range(1,2)|
CREATE (:produzent { name: "Produzent" + r, lat: tan(rand())*100, lon: tan(rand())*100,
co2: round(exp(rand()*8)+250), cost: round(exp(rand()*3)+20), time: round(rand()*5) }))

FOREACH (r IN range(1,2)|
CREATE (:grosshandler { name: "Grosshändler" + r, lat: tan(rand())*100, lon:
tan(rand())*100,
co2: round(exp(rand()*8)+250), cost: round(exp(rand()*3)+20), time: round(rand()*5)}))

FOREACH (r IN range(1,4)|
CREATE (:handler { name:"Händler" + r, cost: round(exp(rand()*3)+20), co2:
round(exp(rand()*8)+250), lat: tan(rand())*100, lon: tan(rand())*100, time:
round(rand()*5)}))

FOREACH (r IN range(1,20)|
CREATE (:kunde { name:"Kunde" + r, cost: round(exp(rand()*3)+20), co2:
round(exp(rand()*8)+250), lat: tan(rand())*100, lon: tan(rand())*100, time:
round(rand()*5)}))
```

So liefert Neo4j eine Darstellungsform der beispielhaften Supply Chain mit 68 Teilnehmern, die untereinander 412 Beziehungen aufweisen.

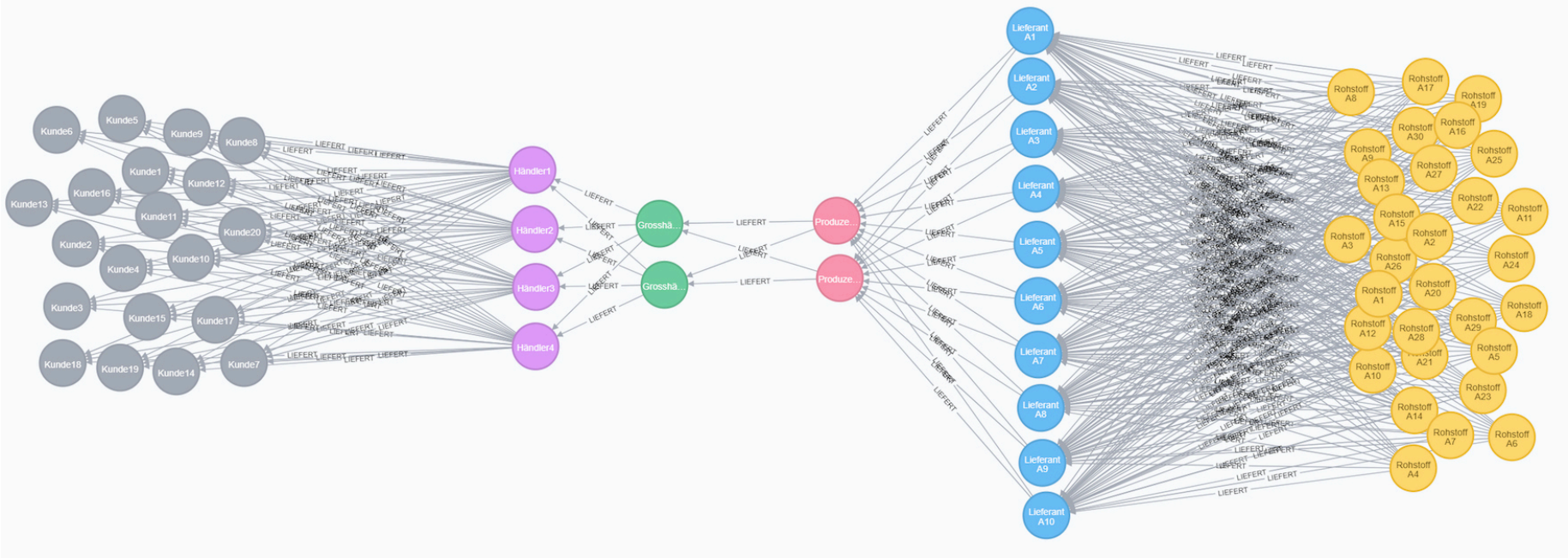

Abbildung 3-26: Neo4j Ausgabe zur skalierten Supply Chain

Nachteilig an dieser Darstellung ist, dass bei einer drastischen Erhöhung der Anzahl an Teilnehmern die Kanten nicht mehr leserlich wiedergegeben werden können. Zusätzlich ist bei mehr als sieben verschiedenartigen Teilnehmern der Supply Chain die farbliche Trennung nicht mehr zu realisieren, da nur sieben Farbgebungen existieren. Die klaren Vorteile sind die visualisierten Ballungszentren zwischen Händler und Kunden. Hier erkennt man die immense Dichte der Verknüpfungen.

3.2.2 Fazit

Zusammenfassend lässt sich sagen, dass die graphenorientierte NoSQL-Datenbank Neo4j eine bedarfsgerechte graphische Oberfläche zur Darstellung von Supply-Chain-Daten anbietet. Da die Modellierung und Speicherung der Daten über Knoten und Kanten erfolgt, hat Neo4j den klaren Vorteil der einfachen und übersichtlichen Datenmodellierung, die beispielsweise direkt am Whiteboard geschehen kann. Vor allem Verknüpfungsintensive Datenmengen lassen sich nicht nur anschaulich, sondern vor allem benutzerfreundlich, darstellen.
Zusätzlich lassen sich große Datenmengen importieren und beliebige Abfragen erstellen. Die Ausgabe erfolgt über den Neo4j-Browser, wo einzelne Knoten durch farbige Kreise anschaulich dargestellt werden. Die Anordnung der Knoten erfolgt automatisch und durch die übersichtliche Visualisierung der Kanten werden die Beziehungsintensitäten zwischen den einzelnen Knoten verdeutlicht. Beispielsweise lassen sich so leicht Ballungszentren visualisieren und auf anhieb auffindbar machen.
Nicht nur die übersichtliche grafische Darstellung und einfache Abfrageform, sind klare Vorteile der graphenorientierten NoSQL-Datenbank Neo4j, auch die Möglichkeit, über die eigens für Neo4j entwickelte native Abfragesprache Cypher, benutzerfreundlich und somit einfache, aber auch komplexe Abfragen erstellen zu können. Die Ausgabe der Ergebnisse erfolgt über Listen, die sich nicht nur exportieren, sondern auch einzelne Ergebnisse grafisch darstellen lassen. Aufgrund der Ähnlichkeit von Cypher zur SQL-Abfragesprache, bedeutet dies für Entwickler nur kurze Lernzeiten. Vor allem zeichnet sich Neo4j durch eine Vielzahl von Sprachanbindungen, wie beispielsweise Java-APIs, aus.
Des Weiteren ist die Abfragegeschwindigkeit von Neo4j deutlich schneller, als die von relationalen Systemen. Dies ist Möglich, da die Geschwindigkeit der Abfrage nicht mehr von der Gesamtmenge der Daten abhängt, sondern von der konkreten Anzahl der Beziehungen zwischen den einzelnen Daten. So realisiert Neo4j auch komplexe Abfragen in Echtzeit.

4. Zusammenfassung und Ausblick

In dieser Bachelorarbeit wurde ein Beitrag erarbeitet, der Unternehmen bei der Auswahl von Speicherungs- und Aufbereitungmöglichkeiten großer Datenmengen durch NoSQL-Datenbanken unterstützt.

Als erstes Ergebnis dieser Bachelorarbeit konnten die Grenzen von relationalen Datenbanken aufgezeigt und die Potenziale von NoSQL-Datenbanken zur Verarbeitung umfangreicher Datenmengen aufgeführt werden.

Darüber hinaus wurden auf Basis einer intensiven Literaturrecherche Vergleichskriterien im Kontext von Transaktionsdaten zur Auswahl einer NoSQL-Datenbank identifiziert. Neben den sieben ausgewählten Vergleichskriterien wurde auf eine Möglichkeit des Fragenkataloges zur feineren Auswahl einer NoSQL-Datenbank für Unternehmen hingewiesen. Hierauf aufbauend wurden einzelne Datenbanken der vier Kernklassen von NoSQL-Systemen untereinander verglichen und so die zweckdienlichste NoSQL-Datenbank zur bedarfsgerechten Darstellung von Supply-Chain-Daten gefunden.

Im Rahmen dieser Arbeit wurde die praktische Verwertbarkeit der ausgewählten graphenorientierten NoSQL-Datenbank Neo4j anhand der exemplarischen Darstellung einer beispielhaften Supply Chain mit eingeschlossenen Transaktionsdaten demonstriert. Zudem wurde Neo4j auf Grundlage der erstellten Abfragen und Darstellungsformen kritisch diskutiert.

Zusammenfassend lässt sich sagen, dass NoSQL-Datenbanken nicht den Anspruch haben das bestehende Umfeld von relationalen Datenbanken in Unternehmen zu ersetzen, sondern vielmehr die spezifischen Vorteile für bestimmte Anwendungen anzubieten. Für die stetig ansteigenden Datenmengen eigenen sich mehrere unterschiedliche Datenbanken in Unternehmen besser, um das volle Potenzial von Big Data auszunutzen. Die Kombination von bestehenden Systemen und graphenorientierten Datenbanken eröffnen neue Möglichkeiten der besseren Auswertung und Nutzung von großen Datenmengen. Dieser Trend der Kombination wird polyglotte Persistenz genannt und soll für weitere wissenschaftliche Arbeiten als Ausgangspunkt dienen, um verschiedene relationale und NoSQL-Technologien sinnvoll zu kombinieren und so Unternehmen eine noch zweckdienlichere Nutzung ihrer Informationen anzubieten.

5. Literaturverzeichnis

[Arn08] Arndt, H.; Supply Chain Management: Optimierung logistischer Prozesse. 4. Auflage, Wiesbaden: Gabler Verlag: 2008

[Cat11] Cattel, R: Scalable SQL and NoSQL data stores, ACM SIGMOND Record, Vol. 39, No. 4, pp. 12-27, New York, NY, USA: 05.2011, URL: http://portal.acm.org/citation.cfm?doid=1978915.1978919, letzter Zugriff: 24.06.2018

[Coo10] Cooper, B., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R.: Benchmarking cloud serving systems with YCSB. In Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10). ACM, New York, NY, USA: 2010, pp. 143-154. URL: http://portal.acm.org/citation.cfm?doid=1807128.1807152, letzter Zugriff: 13.07.2018

[Dav18] Davoudian, A; Chen, L; Liu, M: A Survey on NoSQL Stores, ACM Computing Surveys, Vol. 51, No. 2, Article 40, pp. 1-43, New York: 06.2018, http://dl.acm.org/citation.cfm?doid=3186333.3158661, letzter Zugriff: 18.08.2018

[Dbe18] solid IT GmbH; URL: https://db-engines.com/de/ranking, letzter Zugriff: 27.08.2018

[Edl10] Edlich, S.; Friedland A.; Hampe, J., Brauer, B.: NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken. 1. Auflage, München: Hansen Verlag: 2010

[Edl18] Prof. Dr. Stefan Edlich, Stubenrauchstr 9c, 14167 Berlin, URL: http://nosql-databases.org/, letzter Zugriff: 17.07.2018

[Gup17] Gupta, A.; Tyagi, S.; Panwar, N.; Sachdeva, S.; Saxena, U.: NoSQL databases: Critical analysis and comparison. International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN), IEEE, pp. 293-299, Gurgaon, India: 10.2017, URL: http://ieeexplore.ieee.org/document/8284494/, letzter Zugriff: 20.08.2018

[Hei11] Heiserich, O; Helbig, K; Ullmann, W: Informations- und Kommunikationssysteme der Logistik. Wiesbaden: Gabler Verlag, 2011

[Haa18] Haas, A.: Intelligence Systeme im Logistik- und Supply Chain Management: Entwicklung eines Metamodells für einen weiterführenden Managementansatz. 1. Auflage, Wiesbaden: Springer Fachmedien Wiesbaden GmbH: 2018

[Hau14] Hausladen, I: IT-gestützte Logistik: Systeme, Prozesse und Anwendungen. 2. Auflage, Wiesbaden: Springer Gabler: 2014

[Jin11] Jing, H; Haihong, E; Guan, L; Jian, D: Survey on NoSQL database, 6th International Conference on Pervasive Computing and Applications (ICPCA), IEEE, pp. 363-366, South Africa: 2011, URL: http://ieeexplore.ieee.org/document/6106531/, letzter Zugriff: 13.08.2018

[Kem15] Kemper, A; Wickler, A: Datenbanksysteme: Eine Einführung. 10. Auflage, Berlin Boston: De Gruyter Verlag: 2015

[Kuh02] Kuhn, A; Hellingrath B.: Supply Chain Management: Optimierte Zusammenarbeit in der Wertschöpfungskette. Berlin Heidelberg Springer Verlag: 2002

[Law01] Lawrenz, O.; Hildebrand, K.; Nenninger, M.; Hillek, T.: Supply Chain Management: Konzepte, Erfahrungsberichte und Strategien auf dem Weg zu digitalen Wertschöpfungsnetzen. 2. Auflage, Braunschweig: Vieweg Verlag, 2001

[Mei16] Meier, A; Kaufmann, M: SQL- & NoSQL-Datenbanken. 8. Auflage, Berlin Heidelberg: Springer Verlag, 2016

[Pok13] Pokorny, J: NoSQL databases: a step to database scalability in web environment, International Journal of Web Information Systems, 2013, Vol. 9 Issue:1, pp.69-82, URL: https://doi.org/10.1108/17440081311316398, letzter Zugriff: 30.07.2018

[Sch09] Schemm, W: Zwischenbetriebliches Stammdatenmanagement: Lösung für die Datensynchronisation zwischen Handel und Konsumgüterindustrie. Berlin Heidelberg: Springer Verlag, 2009

[Sch16] Schulte, C: Logistik: Wege zur Optimierung der Supply Chain. 7 Auflage, München: Verlag Franz Vahlen GmBh, 2016

[Sch12] Schuh, G; Stich, V: Produktionsplanung und -steuerung 1: Grundlagen der PPS. 4. Auflage, Berlin Heidelberg: Springer Vieweg, 2012

[Sch10] Schwarzer, B; Krcmar, H: Wirtschaftsinformatik: Grundlagen betrieblicher Informationssysteme. 4. Auflage, Stuttgart: Schäffer-Peoschel Verlag, 2010

[Tan16] Tang, E; Fan, Y: Performance Comparison between Five NoSQL Databases. 7th International Conference on Cloud Computing and Big Data (CCBD), pp. 105-109 ,Macau (China): 2016, URL: http://ieeexplore.ieee.org/document/7979888/, letzter Zugriff: 04.09.2018

[Zie15] Ziegler, J: Systematische Untersuchung von möglichen Datenkategorien in Supply Chains. Technische Universität Dortmund, 2015