technische universität
dortmund

Masterarbeit

Konzeptionelle Entwicklung einer Datenbankabstraktionsschicht für Veränderungen am Datenmodell einer datengetriebenen Simulation

Maximilian Flockenhaus
Matrikelnummer: 143828
Studiengang: Logistik

ausgegeben am:
13.11.2018

eingereicht am:
24.04.2019

Prüfer: Univ.-Prof. Dr.-Ing Markus Rabe
Betreuer: Dipl.-Inf. Dominik Schmitt

Technische Universität Dortmund
Fakultät Maschinenbau

Inhaltsverzeichnis

1 Einleitung

Aufgrund sich ändernden wirtschaftlichen Aktivitäten steht die Logistik vor neuen Herausforderungen. Kürzere Produktlebenszyklen, höhere Innovationsgeschwindigkeiten ([Tha01]) und eine fortschreitende Globalisierung ([Bau01]) sind weitreichende Herausforderungen für die Wettbewerbsfähigkeit von Unternehmen im Bereich der Logistik. Ganzheitliche Logistiksysteme bieten eine Möglichkeit, Wettbewerbsvorteile auf- und auszubauen. Weltweit vernetzte Wirtschaftsstrukturen erfordern hochverfügbare Logistiksysteme, die den Material- und Informationsfluss in und zwischen Unternehmen sowie mit ihren Lieferanten und Kunden sicherstellen.
In vielen Unternehmen reicht die innerbetriebliche Optimierung nicht aus, um das langfristige Fortbestehen zu sichern. Große Potenziale bestehen dahingegen in der Optimierung der ganzheitlichen Logistikkette. Diese Potenziale können allerdings nur durch die Interaktion aller an einem Logistiknetzwerk Beteiligten vollumfänglich genutzt werden. ([Dob05], S. 32)

Die Analyse und Planung großer Logistiknetzwerke basiert zunehmend auf modellbasierten Methoden wie Simulation und Optimierung. ([BC09], S. 1) Dabei dient die Simulation als Entscheidungshilfe für die Gestaltung eines Logistiknetzwerkes hinsichtlich seiner Effizienz ([BC09], S. 242). Durch die Simulation werden Maßnahmen ermittelt, die das Logistiknetzwerk optimieren sollen. Die Auswirkungen der Maßnahmen werden wiederum mittels Simulation geprüft und anschließend bewertet. Unter Maßnahmen werden Veränderungen im Logistiknetzwerk beschrieben. Es wird zwischen kurzfristigen Maßnahmen, wie z.B. einer Bestandsänderung eines Artikels und langfristigen Maßnahmen, wie einer Umstrukturierung von Standorten unterschieden ([KKK12], S. 84). Bei der Komplexität eines Logistiknetzwerkes wird eine datengetriebene Simulation verwendet. Bei der datengetriebenen Simulation wird das Simulationsmodell auf der Grundlage externer Datenquellen, beispielsweise einer Datenbank, instanziiert. Ein Logistiknetzwerk kann somit durch ein Schema in der Datenbank dargestellt werden. Durch Veränderungen des Schemas und Einträgen in Relationen in der Datenbank können Maßnahmen am Netzwerk beschrieben werden ([RAS18]). Die Maßnahmen können mittels einer Modellierungssprache, wie z.B. SQL, modelliert und an die Datenbank weitergereicht werden. Wenn sich das Datenbankschema ändert, müssen die Maßnahmen entsprechend angepasst werden, da nicht jedes Datenbankmodell mit jeder Modellierungssprache kompatibel ist.

Ziel der Arbeit ist die konzeptionelle Entwicklung einer Datenbankabstraktionsschicht zur Kommunikation zwischen Maßnahmen und einer zugrundeliegenden Datenbank.
Es wird untersucht, wie ein generischer Ansatz einer Schnittstelle zwischen Maßnahmen und einer Datenbank aussehen kann, die ein Schema zur Abbildung eines Handelsnetzwerkes enthält. Durch den generischen Ansatz einer Datenbankabstraktionsschicht soll die Problematik der Kompatibilität umgangen werden. Es wird ein Konzept erarbeitet, welches Änderungen an dem Schema des Logistiknetzwerkes und Einträge in Relationen durch die Datenbankabstraktionsschicht an die zugrundeliegende Datenbank übermittelt.

Um eine Datenbankabstraktionsschicht zwischen Maßnahmen und einer Datenbank zu entwickeln, werden Grundlagen bezüglich Handelsnetzwerken, datengetriebener Simulation und

Datenbanken dargestellt. Durch die Definition eines Handelsnetzwerkes erfolgt eine Abgrenzung zu den bisher beschriebenen Logistiknetzwerken. Für die Beschreibung von Maßnahmen innerhalb des fertigen Konzeptes werden mögliche Maßnahmen im Bereich der Logistik- und Handelsnetzwerke untersucht. Durch die beschriebenen Grundlagen im Bereich der datengetriebenen Simulation und Datenbanken, soll zusammen mit den Maßnahmen innerhalb eines Handelsnetzwerkes, ein Rahmen für die Datenbankabstraktionsschicht entstehen. Basierend auf den aktuellen Entwicklungstendenzen einer Datenbankabstraktionsschicht wird ein Verfahren erarbeitet, das eine Schnittstelle zwischen Maßnahmen und Datenbanken sowie die Kommunikation des Schemas ermöglicht. Eine Datenbankabstraktionsschicht wird definiert und bereits bestehende Frameworks verglichen. Anschließend wird dieses Konzept anhand eines gegebenen Datenbankschemas zur Repräsentation eines Logistiknetzwerkes und einer gegebenen Datenbank anhand eines Prototyps evaluiert und getestet. Abschließend werden die Ergebnisse der vorliegenden Forschungsarbeit in einem Fazit zusammengefasst und ein Ausblick gegeben.

2 Handelsnetzwerk

Um Optimierungen an einem Handelsnetzwerk vornehmen zu können, muss der Begriff eines Handelsnetzwerkes zunächst definiert werden. In diesem Kapitel werden die Grundlagen für eine Definition hergeleitet. Grundlage der Definition bilden Begrifflichkeiten aus dem Kontext der Handelsnetze und Logistiknetzwerke sowie der klassischen Supply Chain. Zunächst werden die Begriffe eines Handelsnetzes und eines Logistiknetzwerkes erläutert. Anschließend erfolgt die Definition klassischer Supply Chains. Im Nachhinein werden diese Definitionen verwendet, um den Begriff eines Handelsnetzwerkes zu definieren.

Handelsnetz: Nach dem deutschen Duden wird ein Handelsnetz wie folgt definiert:

> *„Netz von Handelsbetrieben gleicher oder ähnlicher Art, die über eine Region verteilt sind“* ([Dud19])

Demzufolge wird ein Handelsnetz geografisch auf eine Region limitiert. In dieser Region befinden sich Handelsbetriebe, die entweder gleicher oder ähnlicher Art entstammen und ein Netz bilden.

Logistiknetzwerk: Ein Logistiknetzwerk beschreibt die Gesamtheit von in Verbindung stehenden logistischen Prozessen und deren Auswirkungen ([Rih08, S.55]). Ein Logistiknetzwerk wird von mehreren Unternehmen gebildet, darunter fallen z.B. Logistiksysteme von Spediteuren, Eisenbahnen, Schifffahrtslinien und Verkehrsbetrieben ([Gud10, S.549]). Damit verbindet das Netzwerk Standorte durch Informations-, Güter- und Finanzfluss ([Suc08, S.934]). Das Logistiknetzwerk wird von Güter-, Waren- und Personenströmen durchlaufen. Dabei werden sie von Informations- und Datenströmen gesteuert und kontrolliert (vgl. [Gud10, S.550]). Durch die hohe Anzahl verschiedener Ströme bilden Logistiknetzwerke eine hohe Komplexität und stellen Herausforderungen für Anwender dar ([RD15, S.1]). Eine Reduktion der Komplexität wird durch das Betrachten von Subsystemen des Logistiknetzwerkes erreicht. Damit wird die Auflösung des betrachteten Systems erhöht. ([VDI13, S.19]) Ein Logistiknetzwerk lässt sich in drei größere Bereiche aufteilen: Das innerbetriebliche Logistiknetzwerk einer Betriebsstätte, das außerbetriebliche Logistiknetzwerk einer Betriebsstätte und das unternehmensübergreifende Logistiknetzwerk ([Log13]).

Die Logistik kann als fließender Prozess verstanden werden. Die Planung, Information und damit zusammenhängenden Prozesse lassen sich in einem Modell darstellen. Das Modell zeichnet das Logistiksystem in seiner Grundstruktur ab. Ein solches System setzt sich aus Lieferpunkten (Quellen) und Empfangspunkten (Senken) zusammen, welche mittels Verbindungswegen (Kanten) miteinander verbunden werden([Gud12], S. 606). Dabei kann es unternehmensinterne als auch unternehmensübergreifende Strukturen umfassen. ([Dob05], S. 36) Abbildung 1 zeigt eine Darstellung eines Logistiknetzwerkes mit seinen Quellen, Kanten und Senken.

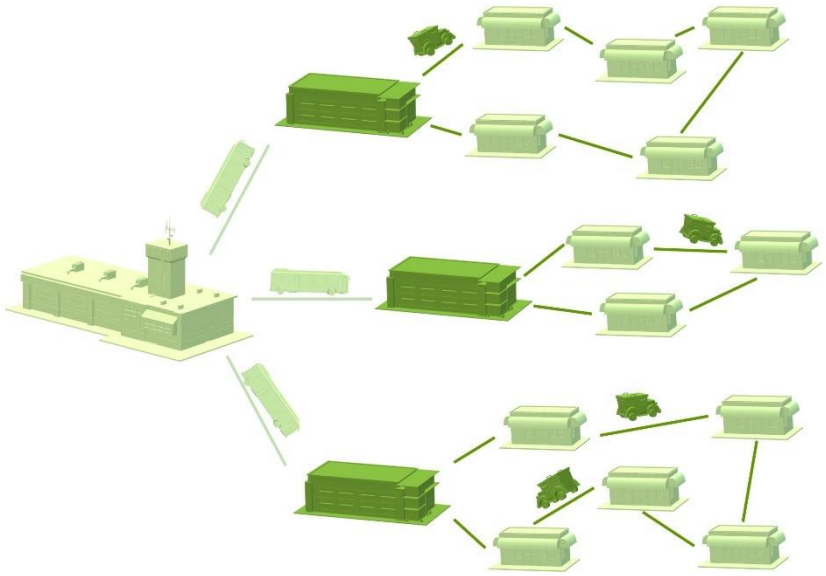


Abbildung 1: Darstellung eines Logistiknetzwerkes nach [Sch17]

Die Quellen, auch Knoten genannt, umfassen unternehmensinterne- oder unternehmensexterne Betriebsstandorte, in denen materielle Objekte hergestellt, be- und verarbeitet, gelagert sowie umgeschlagen werden ([Gud12], S. 606). Über die Verbindungswege werden die materiellen Objekte transportiert oder befördert. Dies resultiert aus der räumlich getrennten Lage von verschiedenen Knoten ([BK08], S. 5). Die Intensität der Verbindung zwischen den Knoten lässt sich über eine Gewichtung an den Kanten darstellen. Dabei kann es sich um eine Menge von transportierten materiellen Objekten zwischen zwei Knoten handeln ([BK08] S. 5). Der Ausgangsknoten bildet die Quelle und der zu beliefernde Knoten die Senke. Zwischen den verschiedenen Quellen und Senken werden Objekte, wie z.B. Handelswaren, Rohstoffe, Pakete, Halb- oder Fertigwaren oder Lebewesen transportiert.

Supply Chain: Eine Supply Chain (Lieferkette) beschreibt ein Netzwerk von Organisationen, die über Verbindungswege an verschiedenen Tätigkeiten und Prozessen der Wertschöpfung in Form von Dienstleistungen und Produkten für den Endkunden beteiligt sind. ([Chr98], S.15) Die Supply Chain bildet einen Schwerpunkt innerhalb der Kernprozesse eines Unternehmens und verläuft über Abteilungs- und Funktionsgrenzen wie Produktion, Einkauf, Logistik und Vertrieb hinweg. Innerhalb der Supply Chain steht die Zufriedenstellung des Kunden im Fokus. ([Bec08], S.48) In Abbildung 2 wird die Darstellung einer Supply Chain veranschaulicht.

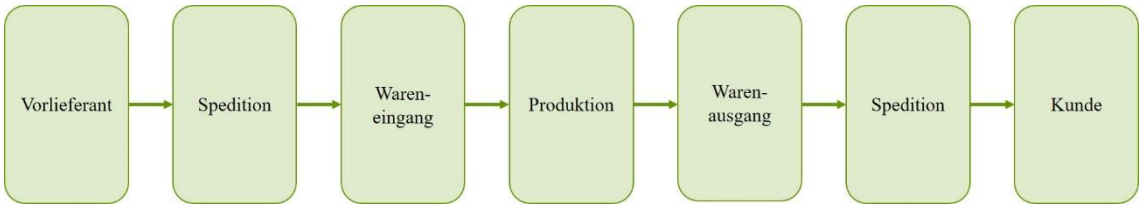


Abbildung 2: Darstellung einer Supply Chain nach [VK12], S. 25

Das Ziel der Supply Chain ist die ganzheitliche Koordination des Warenflusses im gesamten Netzwerk. Eine Betrachtung der einzelnen Teilnehmer steht im Hintergrund. Die Koordination des Warenflusses im gesamten Netzwerk wird auch Supply Chain Management genannt.

([VK12]) Um eine erfolgreiche Supply Chain zu erreichen, befasst sich das Supply Chain Management mit folgenden Teilzielen ([VK17], S.1):

- Orientierung am Nutzen des Endkunden
- Erhöhung der Kundenzufriedenheit durch Lieferung auf Abruf
- Reduzierung der Lagerbestände in der Supply Chain und damit verbundene Reduzierung der Kosten für die Lagerhaltung
- schnellere Anpassung an Marktveränderungen
- Erzeugung eines stetigen Warenflusses, einhergehend mit einer möglichen Vereinfachung der Steuerung
- Reduzierung der Auftragsabwicklungszeiten im Zeitwettbewerb
- Vermeidung von „Out-of-Stock"-Situation im Handelsbetrieb

Handelswaren: Nach Prof. Dr. Alexander Hennig, Professor für Handelsmanagement und Prof. Dr. Willy Schneider, Professor im Bereich BWL-Handel, werden Handelswaren wie folgt definiert:

„bewegliche Sachgüter, die in absatzfähigem Zustand bezogen und ohne Be- oder Verarbeitung (meist mit einem Aufschlag) wieder verkauft werden. Manipulationen wie Sortieren, Mischen, Abpacken, Markieren gelten dabei nicht als Be- oder Verarbeitung. Eine Einteilung ist nach einer Vielzahl von Merkmalen der Warentypologisierung *möglich."* ([HS18])

Nach den vorliegenden Definitionen lässt sich eine Definition für ein Handelsnetzwerk wie folgt ableiten:

„Ein Handelsnetzwerk ist ein Netzwerk, das auf eine Region limitiert ist, bestehend aus mehreren Organisationen, die über Verbindungswege an verschiedenen Tätigkeiten und Prozessen der Logistik beteiligt sind. Es bestehen mehrere Stufen im Netzwerk. Diese Stufen beinhalten Zentrallager, Regionallager und Auslieferungslager. In dem Handelsnetzwerk werden ausschließlich Handelsgüter gelagert, sortiert, gemischt, abgepackt, markiert oder geliefert."

Somit ähnelt die Definition eines Handelsnetzwerkes dem eines Logistiknetzwerkes. Ein Unterschied liegt in der Limitierung von Handelswaren. Innerhalb eines Handelsnetzwerkes können verschiedene Veränderungen stattfinden. Diese Veränderungen werden auch Maßnahmen genannt. Im folgenden Kapitel werden Maßnahmen im Handelsnetzwerk näher erläutert.

2.1. Maßnahmen im Handelsnetzwerk

Da nur wenig Literatur zu Handelsnetzwerken existiert, orientiert sich dieses Kapitel an zugrunde liegender Literatur, die mögliche Maßnahmen im Logistiknetzwerk, in der Supply Chain und in der Distributionslogistik beschreiben.
Maßnahmen lassen sich in kurzfristige (operative) und langfristige (strategische) Maßnahmen unterscheiden. Sowohl die operativen, als auch die strategischen Maßnahmen dienen der Optimierung des Netzwerkes.

Zu den operativen Maßnahmen zählen Produktivitätsverbesserungen im Lager- und Transportbereich, der Tourenplanung sowie Verbesserungen im Einkauf von Logistikdienstleistungen und Bestandsoptimierungen. ([KKK12], S. 84) Eine einfache operative Maßnahme könnte die Erhöhung des Lagerbestandes einer beliebigen Stock Keeping Unit (SKU) an einem Standort um einen beliebigen Wert beschreiben ([RAS18]). Eine SKU bezeichnet die kleinste Dispositionseinheit und ist mit der Artikelebene vergleichbar ([Ném10], S. 25). Die SKU ist mittels eines angebrachten Codes eindeutig identifizierbar. Der Code besteht aus einer Kombination von Buchstaben und Zahlen. ([Log16]) Operative Maßnahmen im Bereich des Bestandsmanagement können bei einer Reduktion des Bestandes von 10 % unter sonst gleichen Voraussetzungen den Gewinn eines Maschinenbaubetriebes um ca. 13 % steigern ([HHU11], S.116).

Die strategischen Maßnahmen umfassen eine weitestgehende Neugestaltung der Geschäftsprozesse. Bei der Neugestaltung wird das gesamte Netzwerk betrachtet. Darunter versteht sich die Abstufungen des Netzwerkes, die Festlegung der Standortanzahl und ihre geographische Lage, der Einzugsbereich der Lager sowie die Funktion der einzelnen Standorte im Netzwerk. ([KKK12], S. 84) Strategische Maßnahmen können bei Berücksichtigung von Flexibilität und Nachhaltigkeit im Gestaltungsprozess nicht nur die Anpassungsfähigkeit und ökologische Effizienz verbessern, sondern auch die logistische Leistung erhöhen. Ein weiterer Effekt ist das Einsparen von Kosten im Bereich der Logistik. ([BGW10], S. 49)

Mögliche strategische Maßnahmen nach Arnfried Nagel auf der Ebene der Netzwerkstruktur werden unter der Berücksichtigung der allgemeinen Ziele der Logistik in der Tabelle 1 dargestellt ([Nag11], S. 177).

Tabelle 1: Maßnahmen auf der Ebene der Netzwerkstruktur [Nag11]

	Maßnahme	Auswirkungen auf die klassischen Logistikziele				
		Kosten	Geschwindigkeit	Zuverlässigkeit	Flexibilität	Qualität
Gestaltungsbereich Netzwerkstruktur	Ökologisches Netzwerk-Redesign – Verringerung der Transportleistung unter Berücksichtigung der Transportauslastung durch Anpassung der Netzwerkstufigkeit	Situative Abhängigkeit – tendenziell sinkend durch höhere Auslastung und weniger Fahrtleistung	Situative Abhängigkeit – Reduzierung bei Direktbelieferung – Erhöhung bei zunehmender Stufigkeit	tendenziell geringe Auswirkung daher konstant	Höher bei Direktbelieferung durch erhöhte Reaktionsfähigkeit	nicht einschätzbar
	Regionalisierung von Beschaffungsstrukturen	Situative Abhängigkeit – sinkend durch geringeren Transport und Handling – höher durch Beschaffungspreise	tendenziell höher durch schnellere Lieferung	tendenziell höher durch Komplexitäts-Reduktion und Liefertreue	tendenziell höher durch räumliche Nähe	nicht einschätzbar
	Regionalisierung von Distributionsstrukturen	Situative Abhängigkeit –	tendenziell höher durch	tendenziell höher durch	tendenziell höher durch	nicht einschätzbar

		sinkend durch geringeren Transport und Handling – höher durch Produktionspreise	schnellere Lieferung	Komplexitäts-Reduktion und Nähe zum Abnehmer	räumliche Nähe	
	Umweltorientierte Standortentscheidungen – Reduzierung von Transport und Transportverlagerung	Situative Abhängigkeit – geringere Transportkosten durch reduzierte Fahrtleistung	tendenziell höher bei verbesserter Infrastruktur	tendenziell höher durch schnellere Lieferzeiten	tendenziell höher	nicht einschätzbar
	Einsatz von intermodalem Transport	Nicht eindeutig da höhere Handlingkosten aber geringere Transportkosten	tendenziell geringer	tendenziell höher durch Reduzierung von Stau	tendenziell geringer da reduzierte Netzwerkdichte	nicht einschätzbar
	Multi-User Logistikknoten – horizontale und vertikale Kooperationen	geringer durch Skaleneffekte	tendenziell höher durch mehr dezentrale Knoten im Netz	tendenziell höher durch schnellere Lieferzeiten und räumliche Nähe	tendenziell höher durch mehr Netzwerkknoten	nicht einschätzbar
	Konsolidierung von Warenbewegungen in horizontalen Kooperationen	geringer durch Skaleneffekte und höhere Transportauslastung	tendenziell geringer durch erhöhten Handlingsaufwand	tendenziell höher durch Risikoteilung	tendenziell geringer durch erhöhte Abhängigkeiten	nicht einschätzbar
	Einsatz von Umweltmanagementsystemen	tendenziell höhere Managementkosten aber Steigerung der Effizienz	Situative Abhängigkeit	Situative Abhängigkeit	Situative Abhängigkeit	nicht einschätzbar

Entscheidungsträger für Logistiknetzwerke stehen der komplexen Aufgabe gegenüber, ihre Netzwerke bei sich ändernden internen und externen Anforderungen zu managen. Daher müssen die Entscheidungsträger Maßnahmen identifizieren, um sich an den Zustand des Logistiknetzwerkes anzupassen und ihr Netzwerk zu optimieren. ([RAS18]) Eine Optimierung liegt z.B. in der Reduzierung von Lagerstandorten innerhalb Europas. Auf diese Weise können Lagerkosten gemindert werden. Demgegenüber stehen jedoch steigende Transportkosten für die Belieferung. Eine solche gegenläufige Kostenentwicklung, verbunden mit Bestands- und Auftragsabwicklungskosten, erfordert den Einsatz von strategischen Simulations-Tools. Nur eine genaue Betrachtung aller beteiligten Elemente führt zu einer langfristig richtigen Entscheidung. ([KKK12], S. 84) Für eine Bewertung der Entscheidung wird die Effizienz eines Netzwerkes mittels key performance indcator (KPI) repräsentiert. KPI‘s können z.B. die Kosten für ein Logistiknetzwerk oder das Servicelevel sein. ([BZ07], S. 17; [RCB14], S. 663) Ein größeres Logistiknetzwerk führt zu einer

wachsenden Anzahl von zu untersuchenden Maßnahmen, die bei der Optimierung des Netzwerkes berücksichtigt werden müssen ([RAS18]).

Die Analyse und Planung großer Logistiknetzwerke stützt sich zunehmend auf modellbasierte Methoden, wie die Simulation ([BC09], S. 1). Die Simulation soll als Entscheidungshilfe hinsichtlich der Optimierung des Netzwerkes dienen ([BC09], S. 242). Im Folgenden wird zunächst die Simulation im Allgemeinen erläutert. In einem zweiten Schritt wird der Fokus auf die Handels- bzw. Logistiknetzwerke gelegt.

2.2. Simulation eines Handelsnetzwerkes

Die Simulation beschreibt ein Vorgehen, bei dem ein möglichst realitätsnahes Abbild des Geschehens der Wirklichkeit erschaffen wird. Durch Abstraktion wird ein Modell generiert, an dem zielgerichtet experimentiert werden kann. Die resultierenden Ergebnisse der Experimente werden anschließend auf das reale Problem übertragen. Eine abstrakte Behandlung des Geschehens hat Vorteile, die im Bereich der Sicherheits- und Kostengründe liegen. ([Lac18])

Die Simulation selbst dient nicht als Optimierungsmethode. Sie ermöglicht es aber, gezielte Experimente durchzuführen, bei denen Entscheidungsvariablenwerte z.B. explizite Maßnahmen, festgestellt werden können. ([BHS92], S. 2) Abbildung 3 veranschaulicht das Vorgehen von Simulation und Optimierung.

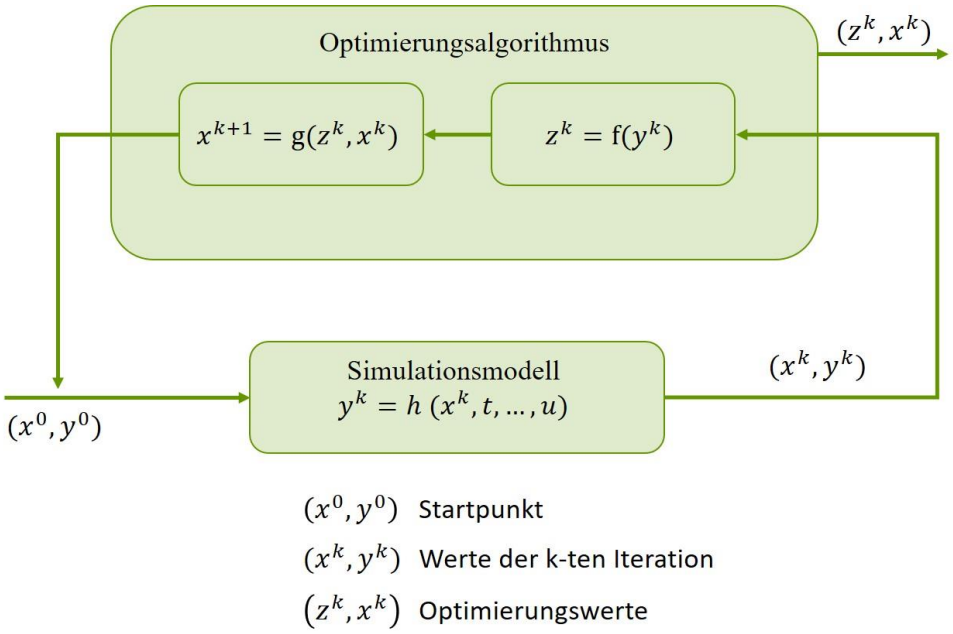


Abbildung 3: Vorgehen von Simulation und Optimierung nach [BHS92]

Diese Arbeit befasst sich mit der computergestützten Simulation. Bei der computergestützten Simulation führt die Modellierung zu einem als Software ablauffähigen Simulationsmodell ([GRSW17], S.22).

Im Rahmen einer Simulation von Produktion und Logistik definiert der Verein Deutscher Ingenieure (VDI) eine Simulation wie folgt:

„Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierbaren Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind“ ([VDI13], S. 3)

In der VDI-Richtlinie wird weiter erläutert, dass Prozesse über die Zeit entwickelt werden. Durch diesen Zusatz grenzt sich das Simulationsmodell von den rein statistischen Analysen ab, bei denen die Dynamik des Systems nicht explizit mit Hilfe eines ablauffähigen Modells repräsentiert wird. ([GRSW17], S. 23)

Die Dynamik entstammt der Physik und befasst sich mit der Lehre für Wirkungen von Kräften in Abhängigkeit von der Zeit ([RS07], S. 55). Wird ein System betrachtet, welches sich zeitlich verändert, wird von einem dynamischen System gesprochen. Ein Beispiel für ein dynamisch System in der Logistik ist ein Lagersystem. Abbildung 4 veranschaulicht einen sich verändernden Lagerbestand. Der Lagerbestand wird durch die Funktion F(t) dargestellt und ist abhängig von der Zeit t und der Anzahl von Ereignissen. Aus der vorliegenden Grafik können Ereignisse abgeleitet werden. So liegt bei der Steigung des Bestandes eine Anlieferung und bei einem abnehmenden Bestand eine Auslieferung vor. Das dynamische Verhalten des Leerens und Befüllens eines Bestandes kann mittels einer ereignisorientierten Simulation analysiert werden. Bei der ereignisorientierten Simulation ist nur das Ereignis relevant ([AF09], S. 311). Eine Betrachtung zwischen den Ereignissen wird nicht berücksichtigt.

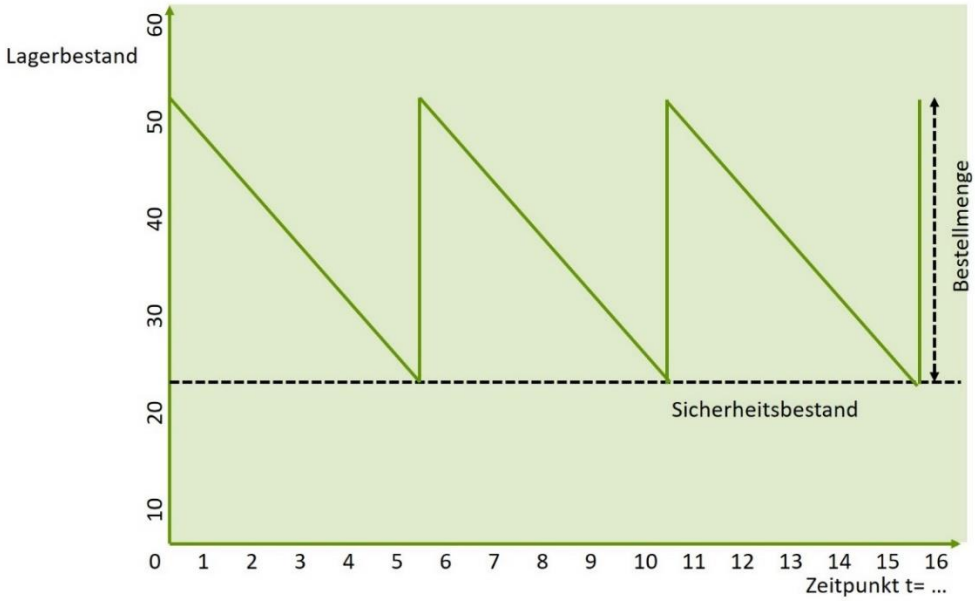


Abbildung 4: Veränderung des Lagerbestandes nach [Sch15]

Prinzipiell lassen sich zwei Simulationsmodelle hinsichtlich der Abbildung der Zeit unterscheiden. Zum einen die kontinuierliche Simulation, welche auf einer kontinuierlichen Zeitmenge basiert. Darüber hinaus wird auch die Zustandsmenge kontinuierlich abgebildet, so dass kontinuierliche Zeitfortschritte mit kontinuierlichen Zustandsmengen dargestellt werden. ([GRSW17], S. 24) Zum anderen die diskrete Simulation, die Zustandsänderungen zu diskreten Zeitpunkten auf Basis einer kontinuierlichen oder diskreten Zeitmenge betrachtet ([GRSW17], S. 24). Die VDI-Richtlinie 3633 definiert eine diskrete Simulation wie folgt:

„diskrete Simulation, bei der der Eintrittspunkt der Ereignisse durch die Ereignisse selbst festgelegt wird“ ([VDI13], S. 7)

So kann ein Ereignis auch eine Zustandsänderung bewirken, die aber keine Zeit in Anspruch nimmt ([VDI13], S. 7).

In Modellen der Produktion und Logistik, welche Stückgut (diskrete Transport- und Bearbeitungseinheiten) mit diskreten Zuständen berücksichtigen, werden die Zustände der Modellelemente nur zu bestimmten Zeitpunkten in Abhängigkeit vom Auftreten eines Ereignisses geändert. Prozesse innerhalb der Logistik lassen sich mittels zustandskontinuierlicher, aber zeitdiskreter Modelle abbilden. Eine kontinuierliche Simulation hat im Kontext des diskreten Materialflusses in Produktion und Logistik eine untergeordnete Bedeutung und wird im Folgenden nicht näher berücksichtigt. ([GRSW17], S. 25)

Für die ereignisdiskrete Simulation wird Software verwendet, die die zu simulierenden Ereignisse verwaltet. In Abbildung 5 wird ein Auszug von erhältlichen Softwarewerkzeugen dargestellt. Die Softwarewerkzeuge sind nach ihrem Spezialisierungsgrad angeordnet. Im oberen Bereich sind Werkzeuge angeordnet, die auf einen bestimmten Anwendungsbereich spezialisiert sind. Im unteren Bereich der Abbildung werden Werkzeuge dargestellt, die allgemeingültig sind. Dabei bilden Programmiersprachen wie C++ das allgemeinste Werkzeug zur Simulation. Simulationssprachen sind ebenfalls Programmiersprachen, die durch ihren Aufbau und spezielle Sprachelemente die Durchführung von Simulationen unterstützen.
Bekannte Simulationssprachen sind unter anderem Dynamo, GPSS, Simscript und Simula.

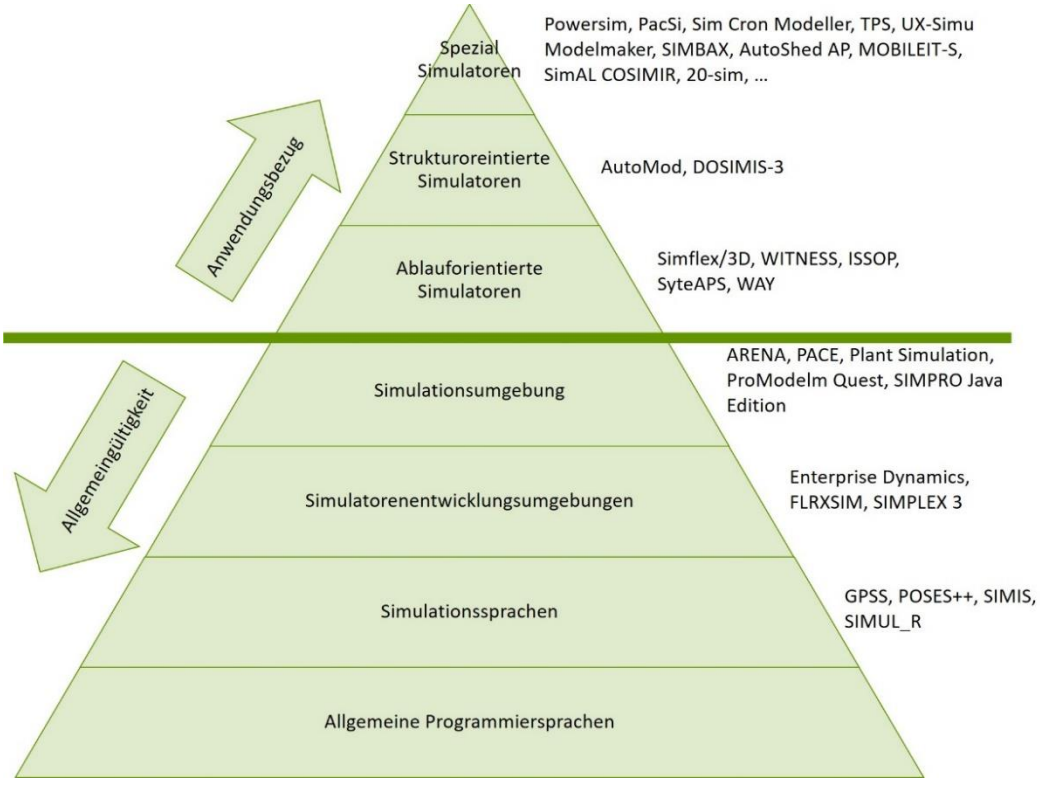


Abbildung 5: Softwarewerkzeuge für die Simulation nach [Ele12]

([LS18]) Eine Simulationsumgebung wie z.B. Plant Simulation oder SimChain bietet einen Kompromiss im Freiheitsgrad der Spezialisierung einer Simulation. So lassen sich neue Logistikassistenzsysteme in der Simulationsumgebung entwickeln. Logistische Assistenzsysteme sammeln Informationen aus unterschiedlichen Datenquellen und verdichten diese für eine konkrete Planungsaufgabe im Logistiknetzwerk. Sie sollen die Entscheidungsträger im Logistiknetzwerk unterstützen. ([Fra19]) Ein Beispiel für eine solche Entwicklung ist das Assistenzsystem von M. Rabe, M.

Ammouriova und D. Schmitt. In diesem Assistenzsystem wird ein modelliertes Logistiknetzwerk automatisch geändert. Unter diesen Veränderungen werden Maßnahmen angewendet. Die resultierenden Auswirkungen der veränderten Bedingungen lassen Rückschlüsse auf die Leistung des Netzwerkes zu. ([RAS18])

Simulationswerkzeuge können unter anderem auf bestehende Daten zurückgreifen. Diese Vorgehensweise wird datengetriebene Simulation genannt. Hierbei wird ein Simulationsmodell initialisiert und mit Daten versorgt. Für die Simulation benötigte Daten werden häufig in einer Datenbank gespeichert. Während der Simulation werden die Daten aus der Datenbank gelesen. Die Daten aus der Datenbank stellen Objekte mit ihren Eigenschaftsausprägungen dar. Auf diese Weise werden sämtliche Objekte programmgesteuert generiert. Das Simulationsmodell bleibt dabei unabhängig vom Inhalt der generierten Daten aus der Datenbank. Die datengetriebene Simulation bietet folgende Vorteile gegenüber einer konventionellen Simulation ([BHS92], S. 52):

- Flexible Anwendbarkeit des erstellten Simulationsmodells bei unterschiedlichen Gegebenheiten.
- Bei Änderungen werden die Daten in der Datenbank angepasst, nicht aber in dem Simulationsmodell. Bei einer ständigen Änderung der Daten kann eine quasi parallele Simulation erfolgen. Das heißt der Versuchsaufbau bleibt konstant und wird mit unterschiedlichen Datensätzen bedient, die jeweils andere Ergebnisse zeigen können.

Um eine datengetriebene Simulation durchführen zu können, müssen die verwendeten Daten in einer systematischen Form gespeichert werden. In den meisten Simulationsmodellen wird eine relationale Datenbank verwendet. ([BHS92], S. 53) Im anschließenden Kapitel werden Datenbanken näher erläutert und die verwendete relationale Datenbank, auf die Simulationsmodelle zugreifen, erklärt. Auf das Simulationsmodell sowie die Simulationsumgebung wird in dieser Arbeit nicht näher eingegangen, da das Simulationsmodell unabhängig vom Inhalt der generierten Daten aus der Datenbank ist.

3 Datenbanksystem

Daten sind nach einer Syntax zusammengefasste Zeichen, die mit entsprechender Bedeutung Informationen darstellen ([HGH11], S. 5). Dabei können die Daten verschiedene Datentypen haben, wie z.B. eine ganze Zahl (integer), ein Datum (date) oder eine alphanumerische Zeichenkette (char). Zudem können Daten nach ihrem Verwendungszweck in Stammdaten oder Bewegungsdaten unterschieden werden. Stammdaten, wie beispielsweise Personaldaten, ändern sich deutlich weniger häufig als z.B. Bewegungsdaten einer Lagerzubuchung. ([Vie12], S. 96)

Es besteht die Möglichkeit, Daten in einer Datenbank auf einem Computersystem zu speichern und zu verwalten. Inhaltlich zusammengehörige Daten können in Form von Datensätzen abgelegt werden. Jeder Datensatz beinhaltet eine feste Anzahl von Datenelementen und kann mit verschiedenen anderen Datensätzen in Beziehungen stehen.

Neben der Datenbank, die Daten speichert, existiert das Datenbankmanagementsystem, im Folgenden DBMS genannt, welches die Datenbank verwaltet und Zugriffsrechte definiert. Abbildung 6 zeigt die Datenbank und das DBMS, welche zusammen ein Datenbanksystem bilden. ([Vie12], S. 98) Einige Beispiele für eine DBMS Software sind Microsoft´s SQL Server, Corporation´s Oracle, Oracle´s MySQL und IBM´s DB2 ([CM14], S.23).

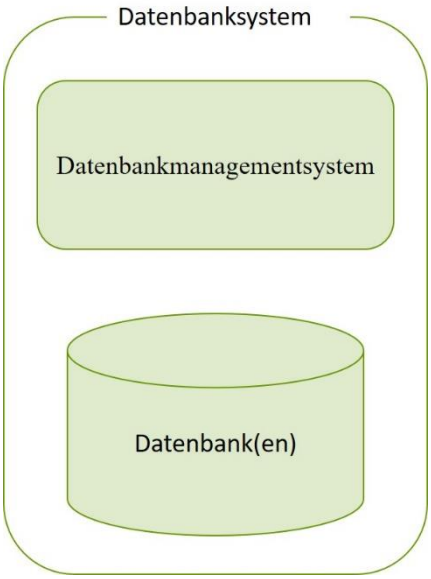


Abbildung 6: Zusammensetzung eines Datenbanksystems nach [Vie12]

Datenbanksysteme bieten mit ihrem DBMS im Gegenzug zu Computer-Dateien, die nicht in einem Datenbanksystem verwaltet werden, folgende Vorteile ([KE15], S. 21ff.):

Vermeidung von Redundanz und Inkonsistenz: Werden Daten in isolierten Dateien verwaltet, müssen dieselben Informationen oft redundant gespeichert werden, um verschiedenen Benutzern Zugriff zu gewähren. Bei Änderungen von Daten können Inkonsistenzen zwischen den verschiedenen Versionen entstehen. In einem globalen, integriertem DBMS kann diese Art von Redundanz vermieden werden. Dies geschieht indem das DBMS die Zugriffe auf die Datenbank verwaltet.

Verbesserte Zugriffsmöglichkeiten: DBMS ermöglichen einen Zugriff auf gut verwaltete Daten, wodurch eine schnelle Rückgabe von Informationen ermöglicht wird. Des Weiteren können in einem Datenbanksystem Informationen aus einer Datei mit anderen logisch verwandten Daten aus einer anderen Datei verknüpft werden. Die gesamten Informationen werden in einem homogenen integrierten DBMS modelliert, wodurch sich die Daten flexibel miteinander verknüpfen lassen.

Mehrbenutzerbetrieb: DBMS bieten eine Mehrbenutzerkontrolle an. Durch diese kann z.B. das Phänomen des „lost updates“ verhindert werden. Bei diesem Phänomen wird eine Datei gleichzeitig von zwei Benutzern geöffnet. Werden die Dateien nacheinander gespeichert, wird die erste gespeicherte Datei überschrieben und die bearbeiteten Daten des ersten Benutzers gehen verloren.

Vermeidung des Verlustes von Daten: Bei der Benutzung von isolierten Dateien kann es oft zu Verlusten von Daten kommen ([Sch00], S. 96). Dies kann z.B. durch das Überspeichern der Dateien von anderen Benutzern geschehen. Eine Wiederherstellung ist in den meisten Fällen nicht möglich oder sehr aufwendig. Eine Recoverykomponente des DBMS soll vor Datenverlust schützen. Das Recovery reicht vom Zurücksetzen einer einzigen Transaktion bis hin zum Erzeugen des letzten konsistenten Datenbankzustands aus einer Archivkopie. Eine Transaktion bildet eine logische Verarbeitungseinheit auf der Datenbank, welche vollständig ausgeführt werden muss, um Korrektheit sicherzustellen ([EN09], S.411). Zu einem bestimmten Zeitpunkt in der Datenbank gespeicherte Daten werden als Datenbankzustand bezeichnet ([EN09], S.42).

Keine Integritätsverletzung: Verschiedene Anwendungsgebiete erfordern über mehrere Informationseinheiten erstreckende Integritätsbedingungen. In einem DBMS wird die Konsistenzbedingung erzwungen, so dass eine Transaktion nur ausgeführt wird, wenn sie die Datenbasis in einen konsistenten Zustand überführt.

Bessere Datensicherheit: Umso mehr Benutzer einen Zugriff auf Daten haben, desto größer ist das Risiko für Sicherheitslücken. Durch ein zur Verfügung gestelltes Framework des DBMS lassen sich Zugriffsrechte verteilen, um eine bessere Datensicherheit zu gewährleisten.

Geringere Entwicklungskosten: Bei der Entwicklung neuer Anwendungsprogramme muss sich der Programmierer jedes Mal mit den oben beschriebenen Einschränkungen von isolierten Daten bzgl. der Dateiverwaltung auseinandersetzen. Ein DBMS bietet eine weitaus komfortablere Schnittstelle, die die Daten innerhalb einer Datenbank verwaltet und somit zu einer Reduktion von Entwicklungszeiten als auch Kosten führen kann.

Es gibt verschiedene Typen von Datenbanken, die unterschiedliche Eigenschaften aufweisen. Hierzu gehören unter anderem aktive Datenbanken und Cloud Datenbanken. Eine aktive Datenbank besitzt eine ereignisgesteuerte Architektur, welche auf Bedingungen außerhalb und innerhalb der Datenbank reagieren kann. Bei der ereignisgesteuerten Architektur rücken Ereignisse als zentrales Strukturierungskonzept in den Fokus der Softwarearchitektur ([BD10], S.vii). Innerhalb der aktiven Datenbank wird die Ereignissteuerung Trigger genannt. Prozeduren dienen dem Ausführen eines Skriptes und sind ebenfalls ausschlaggebend für den Typ einer aktiven Datenbank. ([WC96], S. 12) Eine Cloud Datenbank hat die Eigenschaft, dass Dateninhalte nicht auf einem Computersystem liegen, sondern auf mehrere Computersysteme verteilt werden. ([Cha14], S.8)

3.1. Datenbankmodelle

Das DBMS basiert auf dem Datenbankmodell. Zusammen bilden Sie die Infrastruktur für die Modellierung. Ein Datenbankmodell legt die Modellierungskonstruktion fest, mit der es möglich ist, ein computerisiertes Informationsabbild der realen Welt zu generieren. Es ermöglicht zum einen die Beschreibung der Datenobjekte, zum anderen die Festlegung der anwendbaren Operatoren und deren Wirkung. Für die Beschreibung der Datenobjekte und zur Festlegung der anwendbaren Operatoren und deren Wirkung existiert jeweils eine Teilsprache, aus denen das Datenmodell besteht. ([KE15], S. 25)

Die Datendefinitionssprache (engl. Data Definition Language), im Folgenden DDL genannt, wird zur Strukturbeschreibung der zu speichernden Datenobjekte verwendet ([Rit02], S.128). Hierbei werden gleichartige Datenobjekte durch ein gemeinsames Schema beschrieben. Ein Datenbankschema liegt vor, wenn von einer Strukturbeschreibung aller Datenobjekte eines betrachteten Anwendungsbereichs gesprochen wird. Ein Datenbankschema kann auch als Metadaten (Daten über Daten) verstanden werden. ([Wie09], S.249)

Die Datenmanipulationssprache (engl. Data Manipulation Language), im Folgenden DML genannt, hat den Bestandteil der Anfragesprache (engl. Query Language). Die DML kann zur Änderung von abgespeicherten Daten, zum Einfügen und zum Löschen von gespeicherten Daten genutzt werden. Zudem können Informationen in den gespeicherten Daten mittels der Anfragesprache gesucht werden. ([Rit02], S.128)

Aufbauend auf der DDL und DML existieren verschiedene Entwicklungen von Datenmodellen. Diese haben sich durch das Streben nach besserem Datenmanagement und durch kontinuierliche Weiterentwicklung ausgebildet. Hierbei wird versucht, die kritischen Mängel der vorherigen Modelle zu beheben und Lösungen für die sich ständig weiterentwickelnden Datenverwaltungsanforderungen bereitzustellen. In Tabelle 2 sind die verschiedenen Generationen von Datenmodellen abgebildet. Die ersten beiden Generationen werden oft als satzorientierte Datenmodelle zusammengefasst und haben heute fast nur noch eine historische Bedeutung. Das relationale Datenbanksystem ist heutzutage marktbeherrschend und bildet einen Bestandteil dieser Masterthesis. Die relationale Darstellung lässt sich als Tabelle vorstellen. Sie ist auf redundanzfreie physische Speicherung ausgelegt. Eine redundanzfrei physische Speicherung ist eine Speicherung von Informationen auf einem physischen Träger, bei dem es zu keiner Verdopplung von Informationen kommt. Das objektorientierte Datenmodell hat sich im Kontext des Internets und durch die Anforderung der Speicherung von verschiedenen Objekttypen beispielsweise Video, Bilder, Audio und Text entwickelt. Es gilt als zukunftsorientiertestes Modell. ([CM14], S.41)

Tabelle 2: Generationen von Datenbankmodellen [CM14]

Generation	Zeit	Datenmodell	Beispiel für DBMS	Kommentar
1.	1960s-1970s	Dateisystem	VMS/VSAM	Genutzt von IBM, verwaltet Datensätze, keine Beziehungen
2.	1970s	Horizontal und Netzwerk	IMS, ADABAS, IDS-II	Frühes Datenbanksystem, Navigationszugang

3.	Mid-1970s	Relational	DB2, Oracle, MS, SQL Server, MySQL	Konzeptionell Einfach, ER-Modell und supportet relationales Datenmodell
4.	Mid-1980s	Objekt-orientiert Objekt/relational	Versant, Objectivity/DB, DB2 UDB, Oracle 12c	Objekt/relational unterstützt Objekt Datentypen, Star Schema Unterstützung für Data Warehouses, Web Datenbanken werden allgemein
5.	Mid-1990s	XML Hybrid DBMS	DbXML, Tamino, DB2 UDB, Oracle 12c, MS SQL Server	Unstrukturierte Daten Unterstützung, unterstützt große Datensätze, Hybrid DBMS ergänzt Objekt front end zu relationalen Datenbanken
Aufkommende Modelle: NoSQL	Anfang 2000s bis heute	Schlüsselwert Speicher Spaltenorientiert	SimpleDB (Amazon), BigTable(Google), Cassandra (Apache), MongoDB, Riak	Verteilt, hoch skalierbar, hohe Performance, Fehlertolerant, Sehr Großer Speicher, API Verfügbar

Für die Skizzierung der verschiedenen Datenbankmodelle existieren mehrere mögliche Datenmodellierungen ([Sta05], S. VI):

- **objektorientierte Entwurfsmodelle, wie z.B. Unified Modeling Language:** Objektorientierte Entwurfsmodelle befassen sich damit, Softwareobjekte, ihre Verantwortlichkeiten und Kollaborationsbeziehungen zu definieren ([Lar05], S.45). Ein Beispiel ist die Unified Modeling Language (UML), welches eine visuelle Sprache zur Konstruktion, Spezifikation und Dokumentation der Artefakte von Systemen darstellt ([Lar05], S.46).
- **semantische Datenmodelle, wie z.B. Entity-Relationship-Modell:** Semantische Datenmodelle bilden Verfahren ab, die mögliche Beziehungen zwischen den Begriffen eines beliebigen Anwendungsbereiches vorgeben. Darüber hinaus wird die Vorgehensweise zu ihrer Analyse und Darstellung definiert, so dass das semantische Datenmodell auch eine Schnittstelle zu den Benutzern eines Systems bildet. ([Tot00], S.99)

Abbildung 7 zeigt einen Überblick der Datenmodellierung. Die Datenmodellierung beginnt mit einem Ausschnitt der realen Welt, die in ein konzeptuelles Schema umgesetzt wird. Anschließend wird das konzeptuelle Schema in ein Schema der oben beschriebenen Datenbankmodelle überführt.

Das semantische Datenmodell „Entity-Relationship-Modell“ wird in der Praxis oft zur Skizzierung von relationalen Datenbanken verwendet. Im Folgenden wird näher auf das Entity-Relationship-Modell eingegangen, welches anschließend in Kapitel 3.3 in ein relationales Schema überführt wird.

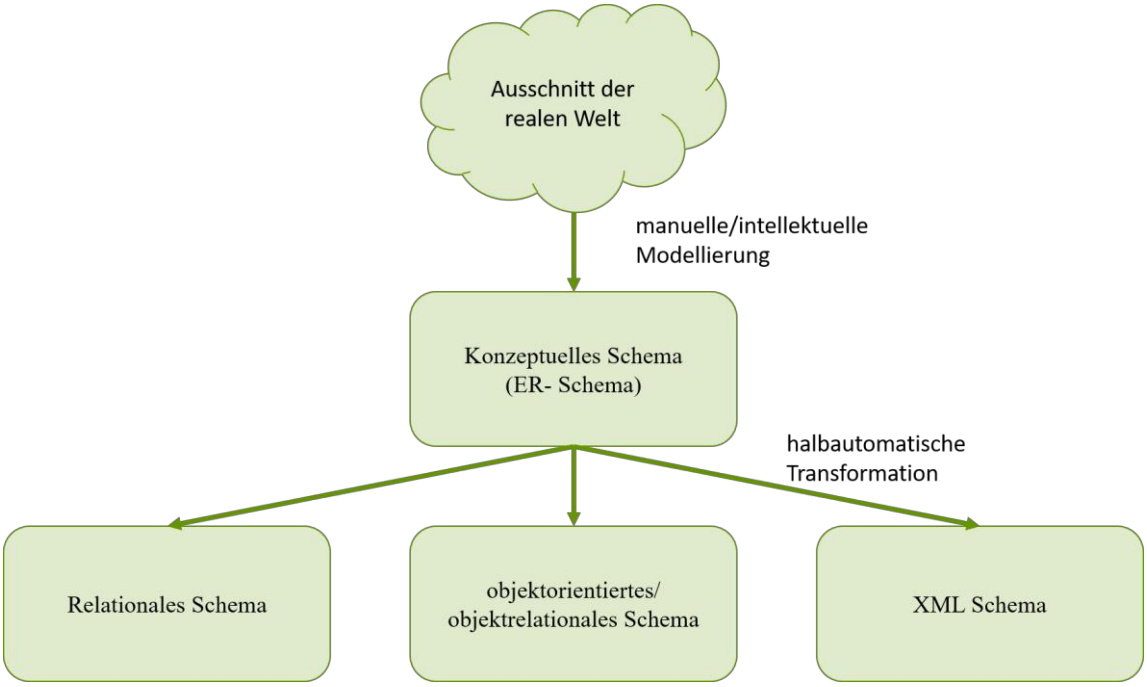


Abbildung 7: Überblick der Datenmodellierung nach [KE15], S. 27

3.2. Entity-Relationship-Modell

Das Entity-Relationship-Modell dient der Modellierung von Daten für Datenbankmodelle. Die grundlegenden Strukturen des Modells bilden die Gegenstände (Entitäten) und die Beziehungen (Relationship). Weitere Bestandteile bilden die Attribute und die Rollen. Entitäten lassen sich als unterscheidbare gedankliche oder physische Konzepte der zu modellierenden Welt beschreiben. Es werden ähnliche Gegenstände oder Gegenstandstypen abstrahiert und grafisch als Rechtecke dargestellt. Der Name des Gegenstandstypen wird innerhalb des Rechtecks angegeben. Die Beziehungen werden zwischen den Gegenstandstypen abstrahiert. Sie werden als Rauten mit Beschriftung dargestellt. Diese Rauten werden über ungerichtete Kanten mit den entsprechenden Gegenstandstypen verbunden. Attribute dienen der Charakterisierung von Gegenständen bzw. Beziehungen. Sie werden in Form von Kreisen oder Ovalen den Rechtecken oder Rauten durch verbundene Kanten zugeordnet. ([Gad17], S. 9)

Abbildung 8 zeigt ein Entity-Relationship-Modell für eine Hochschule. Die Gegenstandstypen sind in diesem Fall der Student, der Professor und die Vorlesung. Beziehungen sind in Form der Rauten dargestellt, also „hören“, „prüfen“ und „lesen“. Die Attribute „Vorname“, „Nachname“, „Semester“, etc., sind oval dargestellt und den Gegenstandstypen zugeordnet. Attribute, deren Werte den zugeordneten Gegenstand eindeutig innerhalb aller Gegenstandstypen identifiziert, werden Schlüssel genannt. Teilweise werden Schlüssel künstlich eingebaut, wie z.B. die „MatrNr“ oder die „VorlNr“. Besteht ein Schlüssel nur aus einem Attribut, wird er als einfacher Schlüssel, andernfalls als zusammengesetzter Schlüssel bezeichnet. Schlüssel werden im Modell unterstrichen oder mit doppeltem Oval bzw. Kreis gekennzeichnet.

Beziehungstypen lassen sich zudem hinsichtlich ihrer Ausprägung charakterisieren. Folgende Beziehungstypen zwischen zwei Gegenstandstypen E1und E2 existieren ([Gad17], S. 11):

- **1:1 Beziehung:** Diese Funktionalität beschreibt den Fall, dass jeder Gegenstand e1 aus E1 höchstens einem Gegenstand e2 aus E2 zugeordnet wird und umgekehrt jedem e2 aus E2 maximal ein Gegenstand e1 aus E1. Es kann dabei vorkommen, dass ein Gegenstand aus E1 keinem Partner aus E2 zugeordnet ist und andersherum.
- **1:N Beziehung/ N:1 Beziehung:** Diese Funktionalität beschreibt den Fall, dass jeder Gegenstand e1 aus E1 beliebig vielen (mehrere oder auch gar keine) Gegenständen aus E2 zugeordnet ist, aber jeder Gegenstand e2 aus E2 mit maximal einem Gegenstand aus E1 in Beziehung steht.
- **N:M Beziehung:** Diese Funktionalität beschreibt den Fall, in dem keinerlei Restriktionen gelten. Jeder Gegenstand aus E1 kann mit beliebig vielen Gegenständen aus E2 in Verbindung stehen und umgekehrt.

Der Typ der Beziehungen wird ebenfalls in das Entity-Relationship-Modell eingetragen (siehe Abbildung 8). Ein Professor prüft mehrere Studenten (1:N Beziehung). Insgesamt lässt sich so ein Modell der realen Welt als Vorlage für eine Datenbank entwickeln.

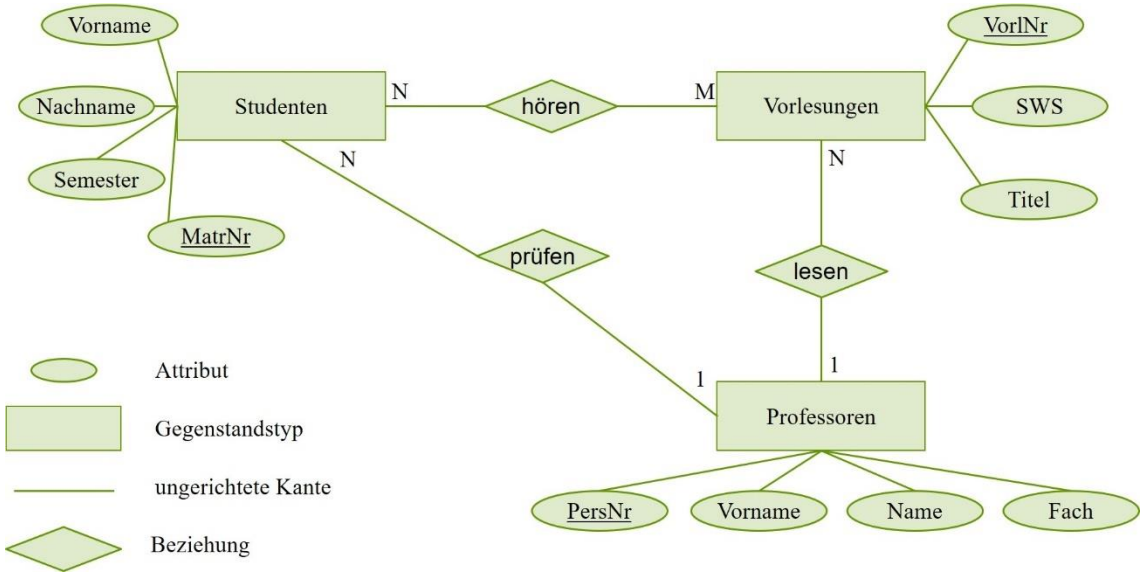


Abbildung 8: ER-Modell einer Hochschule

3.3. Das relationale Schema

Das relationale Schema besteht aus flachen Tabellen (Relationen), in denen die Zeilen den Datenobjekten entsprechen. Eine Zeile wird auch Tupel genannt. Die Spaltenüberschriften der Tabelle werden als Attribute oder Felder bezeichnet. Den Attributen wird jeweils ein Datentyp zugeordnet. Innerhalb einer Relation müssen Attribute eindeutig benannt sein. Unterschiedliche Relationen dürfen gleiche Attribute enthalten. Einzelne Felder in der Tabelle stellen die Ausprägungen der Attribute dar. ([Wie15], S. 17)

Die Umsetzung eines konzeptuellen Schemas, das als Entity-Relationship-Modell erstellt wurde, in ein relationales Schema, geschieht durch die Abbildung der Gegenstandstypen jeweils auf eine Relation. ([Wie15], S. 18) Das in Abbildung 8 gezeigte Entity-Relationship-Modell lässt

sich so in ein relationales Schema bringen (siehe Abbildung 9). Der vorher definierte Schlüssel lässt sich im relationalen Datenmodell zum „Primärschlüssel“ und „Fremdschlüssel“ spezifizieren. Jede Relation sollte einen Primärschlüssel besitzen, da dieser eine Identifikation der Daten ermöglicht. Der Fremdschlüssel ist ein Attribut einer Relation, der in einer anderen Relation einen Primärschlüssel darstellt. ([Gad17], S. 35)

MatrNr	Vorname	Name	Semester

VorlNr	SWS	Titel

PersNr	Vorname	Name	Fach

Abbildung 9: Relationales Schema einer Hochschule

Ein konzeptionelles Schema, das Datenredundanzen enthält, kann bei Änderungen der damit realisierten Datenbank zu Anomalien führen. Um Fehlverhalten zu vermeiden, wird das konzeptionelle Schema normalisiert. Ziel der Normalisierung ist es eine redundanzfreie Datenspeicherung zu erstellen. Dies wird durch die Zuordnung von Attributen zu Relationen erreicht, so dass innerhalb einer Relation keine Redundanz mehr auftritt. Zurzeit existieren sechs Normalformen ([Mei13], S. 36):

- 1. Normalform (1NF)
- 2. Normalform (2NF)
- 3. Normalform (3NF)
- Boyce-Codd-Normalform (BCNF)
- 4. Normalform (4NF)
- 5. Normalform (5NF)

Die ersten drei Normalformen stellen die Gebräuchlichsten und Wichtigsten dar. In der ersten Normalform werden die Attribute in einfache (atomare, skalare) Attribute überführt. Beispielsweise wird das Attribut „Maß eines Kastens“ in die Attribute Länge, Höhe und Breite aufgeteilt. Die Relation befindet sich in der ersten Normalform, wenn die Wertebereiche der Attribute der Relation atomar sind. Dabei kann die erste Normalform Redundanzen enthalten, welche mit den folgenden Normalformen aufgelöst werden. ([Gad17], S. 38)

Die zweite Normalform bedingt, dass sich die Relation in der ersten Normalform befindet und jedes nicht zum Primärschlüssel gehörende Attribut voll von diesem abhängig ist. Dies ist

nur bei zusammengesetzten Schlüsseln von Bedeutung, da eine Tabelle sich noch nicht in der zweiten Normalform befindet, wenn sie einen zusammengesetzten Primärschlüssel besitzt und ein Nichtschlüssel-Attribut nicht vom ganzen Primärschlüssel, sondern nur von einem Teilschlüssel abhängt. Die Spalten die von einem Schlüssel nicht vollständig funktional abhängig sind, werden in einer weiteren Relation ausgelagert. Der Teil des Schlüssels, von dem eine ausgelagerte Spalte funktional abhängig ist, wird zum Primärschlüssel der neuen Relation. ([Gad17], S. 38)

Die Relation befindet sich in der dritten Normalform, wenn sie sich in der zweiten Normalform befindet und zusätzlich kein Attribut, das nicht zum Primärschlüssel gehört, transitiv von diesem abhängt. Die Datenbank sollte aus Relationen in der dritten Normalform bestehen, um eigenständige Attribute korrekt zu identifizieren. Zusammengefasst bedeutet dies, dass die Attribute innerhalb einer Tabelle durch den Primärschlüssel vollständig determiniert sowie voneinander unabhängig sind. Abhängigkeiten zwischen Attributen sind in einer eigenen Tabelle zu isolieren. Dabei dürfen nur lokale oder globale Attribute vorhanden sein. Ein globales Attribut kommt mindestens in einer Relation im Primärschlüssel vor. Ein lokales Attribut kommt nur in einer Relation und nicht im Primärschlüssel vor. ([Gad17], S. 38)

Die Übertragung des normalisierten Schemas geschieht mit der in Kapitel 3.1 beschriebenen DDL. Nach der Übertragung des Schemas kann die DML genutzt werden, um Änderungen und Anfragen durchzuführen. Im folgenden Kapitel wird die relationale Anfragesprache näher betrachtet.

3.4. Relationale Anfragesprachen

Die relationale Anfragesprache oder Query Language ist Teil der DML. Sie ermöglicht die Datenabfrage. Eine der bekanntesten und meistbenutzten Anfragesprachen ist SQL. SQL ist eine Sprache mit deklarativem Code. Damit unterscheidet sich SQL von anderen Anfragesprachen, wie IMS oder CODASYL. Bei ihnen wird der imperative Code verwendet. Die Befehle, die der Computer ausführt, folgen einer bestimmten Reihenfolge. Folgendes Beispiel soll die Unterschiede der Sprachtypen verdeutlichen: Aus einer Anzahl von Tieren sollen nur Löwen gezählt werden. Der Code in der Programmiersprache Python wird in Listing 1 beschrieben. ([Kle17], S. 42)

Listing 1: lion.py: Zählen von Löwen

```
def getlion():
  Lion=[]
  for animal in animals:
    if animal = „Lion“:
      lion.append(animal)
  return len(lion)
```

In der relationalen Algebra wird folgende Schreibweise verwendet:

lion = σ_{family} = „Lion“ (animals)
Das Sigma stellt den Auswahloperator dar, der nur die Tiere widergibt, die die Eigenschaft family = „Lion“ haben

SQL folgt der Struktur der relationalen Algebra:

Select * FROM animals WHERE family = „Lion“;

In einer imperativen Sprache wird dem Computer vorgegeben, welche Operationen er in welcher Reihenfolge durchführen soll. Eine deklarative Sprache wie SQL oder die relationale Algebra verhält sich anders. Es wird angegeben, welche Daten benötigt werden und welche Konditionen zutreffen sollen. Gegebenenfalls wird eine Sortierung oder Gruppierung der Daten vorgenommen. Die Art und Weise, wie dies durchgeführt werden soll, wird nicht vorgegeben. Die Vorgehensweise wird von dem DBMS übernommen. ([Kle17], S. 43)

Es existieren verschiedene relationale DBMS, die verschiedene Anfragesprachen oder verschiedene Versionen nutzen. Seit 1986 existiert die erste SQL-Norm, die sich kontinuierlich erweitert hat. Nicht alle Datenbankhersteller sind auf dem aktuellsten Stand der Norm, so dass Unterschiede in der Sprache existieren. ([DD98], S. 29) Die marktführenden Produkte lauten: Oracle, IBM DB2, Microsoft SQL Server und PostgreSQL ([Dbe19]).

Neben der Funktion als Query Language, dient SQL auch zur Erstellung und Veränderung von Datenbankschemen. Eine neue Relation oder Tabelle wird mit dem Befehl „create table“ eingeleitet([W3S19]):

create table animals
(animal ID **integer not null**,
family **varchar**(15) **not null,**
class **varchar**(10));

Nach dem Namen der Tabelle „animals“ folgen in runden Klammern die Attribute mit ihrem Datentyp. Relationale Datenbanken basieren auf drei fundamentalen Datentypen als Attributs-Domänen: Zeichenketten, Zahlen und Datumstyp. Für jeden dieser Datentypen existieren unterschiedliche, historisch geprägte Varianten. Nach der Typangabe kann zusätzlich die Einschränkung „not null“ folgen. Durch diese Einschränkung wird vorgegeben, dass jedes eingetragene Tupel an dieser Stelle einen festen Wert besitzen muss.

3.5. Integration von SQL in Hostsprachen

Viele Anwendungen und Programme erfordern die Einbettung von SQL in eine Hostsprache. Die Hostsprache bildet eine übergeordnete Programmiersprache. Dadurch kann z.B. eine benutzerfreundliche Umgebung in Form einer grafischen Oberfläche (engl. GUI) erstellt oder eine Turing-Vollständigkeit (universelle Programmierbarkeit) erreicht werden. SQL Statements werden dabei in die Abfolge der regulären Hostsprache eingebettet. Ein Precompiler wandelt die SQL Statements in den normalen Code der Hostsprache um, der dann mit dessen Compiler übersetzt wird. ([KE15], S. 144) Der umgewandelte Code enthält Aufrufe der Schnittstelle zur Datenbank. Eine Verlinkung dieser Schnittstelle als Bibliothek in das fertige Programm muss erfolgen. Dabei ist die Bibliothek abhängig von der verwendeten Datenbank und wird in den meisten Fällen durch den Hersteller bereitgestellt.

Die Integration von SQL in Hostsprachen bringt diverse Nachteile mit sich. Impedance Mismatch stellt dabei einen der größten Nachteile dar ([Kle17], S. 29). Das Impedance Mismatch liegt in der Möglichkeit der Mengenverarbeitung Die meisten traditionellen Programmiersprachen haben keine Möglichkeit der Mengenverarbeitung. Die Tupel der Datensätze werden iterativ verarbeitet. SQL hingegen arbeitet mengenorientiert. Um dem entgegen zu wirken, existiert das Cursor Konzept. Mit dem Cursor ist eine iterative Verarbeitung möglich. Der Cursor zeigt dabei auf das zu bearbeitende Tupel. Bei komplexeren Anwendungen entstehen durch das vielfache Schließen und Öffnen eines Cursors Performance Einbrüche. ([KE15], S. 146) Abbildung 10 veranschaulicht den Ablauf graphisch.

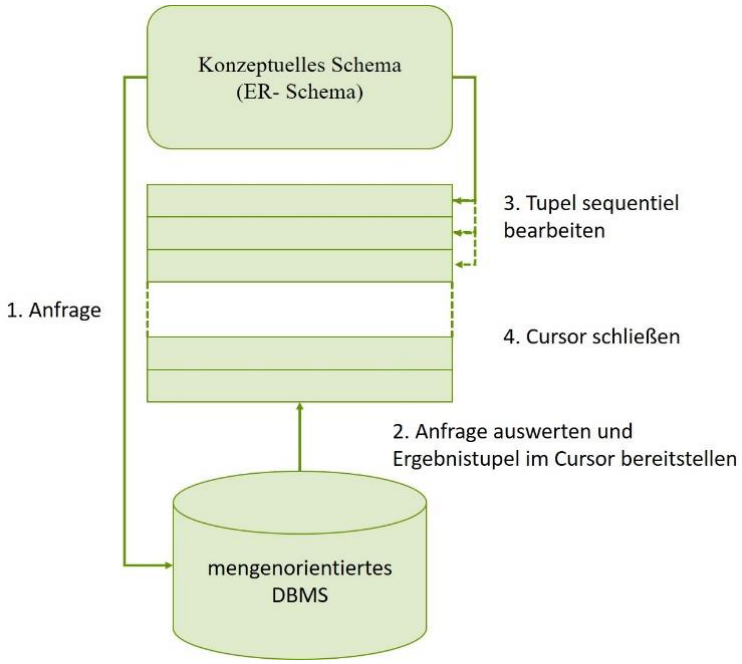


Abbildung 10: Veranschaulichung einer Cursor-Schnittstelle nach [KE15], S. 147

Weitere Nachteile existieren bei einer objektorientierten Sprache als Hostsprache und der Verwendung von relationalen Datenbanken. Es wird von einem objectrelational impedance mismatch gesprochen. Die unterschiedlich genutzten Paradigmen stellen eine Problematik dar. Um den Zustand von Objekten und das Modell der objektorientierten Applikation in einer Datenbank persistent speichern zu können, müssen folgende Komplikationen betrachtet werden([Amb97], [Bow15]):

Speichern der Struktur einer Klasse: In einem objektorientierten Modell besitzt ein Objekt eine Klasse. Dabei definiert die Klasse die Eigenschaften und Methoden des Objektes. Zusätzlich kann sich die Klasse in einer Klassenhierarchie befinden. Das relationale Schema besitzt hingegen keine Klassen. Folglich kann es nicht analog abgebildet werden.

Speichern des Zustandes eines Objektes: Methoden überführen Objektinstanzen in mehrere Zustände. Der aktuelle Zustand des Objektes muss im relationalen Schema dargestellt werden. Es muss analysiert werden, wie der Zustand zu identifizieren ist und in wie weit die Objektinstanz in der Datenbank gepflegt werden muss, um dies zu erreichen.

Speichern von Beziehungen zwischen Objekten: In einem objektorientierten Modell besteht die Möglichkeit des Datenaustauschs für Objekte. Objekte können alleine bestehen, mit anderen Objekten gekoppelt (Assoziation) sein, aus weiteren Objekten zusammengesetzt (Komposition)

sein oder selbst Objekte (Aggregation) beinhalten. Die vorhandene Objektbeziehung muss im persistenten Datenspeicher festgehalten werden.
Speichern von Klassenhierarchien (Vererbung): Klassen stehen in einer hierarchischen Beziehung zueinander. Sie folgen dem Prinzip der Vererbung. In einem relationalen Schema ist eine Modellierung der Vererbung allerdings nicht vorgesehen.
Abfrage von Objekten: In einem relationalen DBMS lassen sich komplexe Abfragen effizient bearbeiten. Eine Abfrage bezieht sich meist auf eine Menge von Objekten mit besonderen Eigenschaften. Im Rahmen der objektorientierten Modelle geschieht der Zugriff nicht über Mengen, sondern über die Navigation von Objektbeziehungen. Das Bilden konkreter Mengen von Objekten oder einer Gesamtansicht über alle Objekte stellt sich als äußerst schwierig dar, da alle Pfade von Beziehungen analysiert werden müssen.

In der Literatur werden zwei Lösungsvorschläge für die genannten Problemstellungen behandelt. Zum einen der komplette Austausch der Datenbank zum Typ einer objektorientierten Datenbank, zum anderen die objektrationale Abbildung ([IBNW09]). Auf den ersten Lösungsvorschlag wird in dieser Arbeit nicht weiter eingegangen, da davon ausgegangen wird, dass die zu verwendende Applikation auf relationalen Datenbanken aufbaut.

3.6. Objektrelationale Abbildung

Bei der Betrachtung der zu lösenden Problematik des „objectrelational impedance mismatch“, wird deutlich, dass die Hauptproblematik darin besteht, ein Element aus dem objektorientierten Paradigma in ein relationales Schema abzubilden. Es müssen ein oder mehrere Elemente auf der Seite des relationalen Models gefunden werden, die das Element des objektorientierten Modells abbilden können. Dieser Vorgang zwischen den Modellen wird auch Mapping genannt. Eine Lösung, die mehrere Mappings definiert und relationale Elemente in objektorientierte Elemente überführt und umgekehrt, wird objektrelationaler Mapper (ORM) genannt. ([SW12], S. 229) Folgende Strategien bewähren sich zur Lösung der Problematik:

Speichern des Zustands eines Objektes: In den meisten Mappings wird eine Klasse als Relation (Tabelle) im Datenbankschema dargestellt. Dabei ist der Name der Klasse gleich dem Namen der Relation. Die Attribute der Relation sind die Attribute der Instanz der Klasse.

Indem die Ausprägungen aller Attribute des Objektes in die Relation eingefügt werden, lässt sich der Zustand eines Objektes dauerhaft speichern. Ein Mapping überführt dabei den Typen des Objektattributes in den Typen der Tabellenspalte. Die Instanz des Objektes wird in Form eines Tupels in der Klassenrelation repräsentiert. ([Fow03], S. 191)

Die Tupel in der Relation müssen eindeutig unterscheidbar sein, damit die Informationen, welche sich in den Tupeln befinden, nicht vertauscht werden. Die eindeutige Unterscheidbarkeit wird dadurch erzeugt, dass der Klasse weitere Eigenschaften hinzugefügt werden. Diese Eigenschaften werden auch shadow information genannt. Diese shadow information wird häufig als numerischer Schlüssel dargestellt. Dieser Schlüssel zählt automatisch für jedes neu angelegte Tupel hoch, wodurch eine Unterscheidung jeder Instanz einer Klasse ermöglicht wird, auch wenn die Attribute übereinstimmen. Dieses Vorgehen wird Eindeutigkeit der Objektidentität genannt. ([Amb03])

Speichern von Klassenhierarchien: Das Speichern von Klassenhierarchien im relationalen Modell ist sehr komplex. Es existieren verschiedene Lösungen, die eine Darstellung der Vererbung ermöglichen.

- **Horizontales Mapping (Class Table Inheritance):** Für jede Klasse der Hierarchie existiert eine eigene Tabelle. In jeder Tabelle werden die Attribute der eigenen Klasse 1 und die Attribute der Klassen, von denen Klasse 1 geerbt hat, als Spalten eingetragen. Alle Attribute der übergeordneten Elternklasse werden somit auf die Kindklasse dupliziert. Die Beziehungen zwischen den beiden Klassen werden nicht abgebildet. Diese Information ist nur im Bereich des objektorientierten Modells bekannt. Das horizontale Mapping ist unidirektional und es gibt keine Möglichkeit ein ausschließlich relationales Modell in ein objektorientiertes Modell zu transformieren. Zum Laden einer Klasse aus der Hierarchie, muss genau eine Tabelle abgefragt werden. Wird jedoch die Struktur einer Klasse geändert, welche sich in der Hierarchie auf einer unteren Ebene befindet, müssen die Änderungen in allen anderen Kindklassen ebenfalls erfolgen. Dieser Vorgang ist aufwendig und fehleranfällig. ([Fow03], S. 285)
- **Vertikales Mapping (Concrete Table Inheritance):** Genau wie beim Horizontalen Mapping besitzt auch beim Vertikalen Mapping jede Klasse der Hierarchie eine eigene Tabelle. Die Attribute der Elternklasse werden nicht in die Kindklasse dupliziert. Stattdessen erhält jede Relation der Kindklasse einen Fremdschlüssel, der auf den Schlüssel der Relation der Elternklasse verweist (mehrere bei Mehrfachvererbung). Dies hat zur Folge, dass ein Objekt der Kindklasse K, welches von der Elternklasse E abgeleitet ist, nur aus der Datenbank geladen wird, wenn das entsprechende Tupel aus der Relation K und das dazu passende Tupel der Relation E geladen wird. Es ist möglich, dass die Elternklasse von einer weiteren Klasse abgeleitet ist. Je nachdem, wie groß die Hierarchie ist, wird der gespeicherte Zustand eines Objektes auf mehr als eine Tabelle verteilt. Für eine sehr große Hierarchie bedeutet dies, dass für das Laden eines Objektes, eine große Anzahl von Tabellen abgefragt werden muss. Da es keine Duplikate von Attributen gibt, stellt das Ändern einer Elternklasse kein Problem dar. Jede Klasse beherbergt seine Attribute in seiner Relation. ([Fow03], S. 293)
- **Filter-Mapping (Single Table Inheritance):** Beim Filter-Mapping wird nur eine Tabelle genutzt, um die gesamte Hierarchie abzubilden. Alle Instanzen der Klasse werden in dieser Tabelle dargestellt. Somit befinden sich ebenfalls alle Attribute der gesamten Hierarchie in der Tabelle. Beim Einfügen eines Tupels der Klasse 1 werden nicht alle Spalten der Klasse 2 benötigt. Es kommt vor, dass ein Tupel leere Attribute besitzt. Um die Zugehörigkeit des leeren Tupels zu einer Klasse zu bestimmen, werden zusätzlich shadow information gespeichert, also ein Object Identifier. In den meisten Fällen geschieht dies durch eine ergänzende Spalte in der Tabelle. Angegeben werden in dieser Tabelle die Namen der Klassen, denen das Tupel zugehörig ist. Diese ergänzende Spalte wird auch Filter (Discriminator) genannt und gibt dem Mapping seinen Namen. Es ist nicht möglich die Integrität der Daten im relationalen Schema zu gewährleisten, da jedes Feld (bei disjunkten Klassenattributen) leer sein darf. Es ist möglich, Attribute in der Datenbank zu hinterlegen, die nicht Teil der Klasse oder des Tupels sind. Bei einer

Abfrage an die Datenbank muss die Abfrage mit dem Kriterium des Filters erweitert werden, damit diese Abfrage performant bleibt. ([Fow03], S. 278)

- **Generelles Mapping (Metadata Mapping):** Beim Generellen Mapping existiert keine eigene Relation für jede Klasse der Hierarchie. Stattdessen wird die Struktur einer Klasse in einem generellen Schema dargestellt. Zur Struktur gehört unter anderem der Name der Klasse, die Attribute und der Name der Elternklasse (x). Eine Relation des Schemas beinhaltet z.B. die Namen aller Klassen und die Referenz auf eine Elternklasse. So existieren ausschließlich verschiedene Relationen mit Attributen, Werten, Instanzen oder Beziehungen. Abbildung 11 zeigt ein Schema für das generelle Mapping. ([Fow03], S. 306)

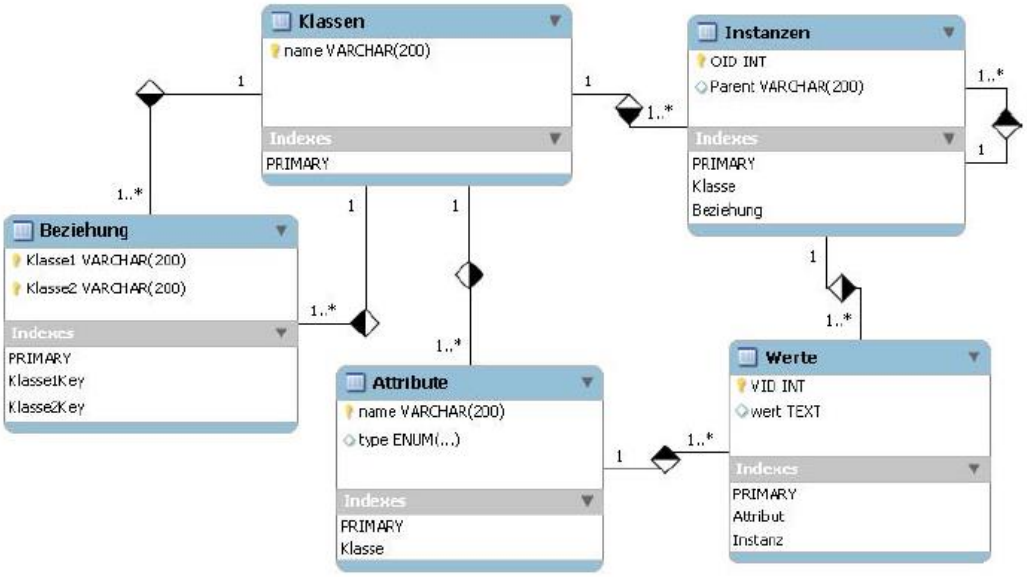


Abbildung 11: Beispielhaftes Schema eines generellen Mapping [Sch13]

Speichern von Beziehungen zwischen Objekten: Das Speichern von Beziehungen zwischen Objekten lässt sich wie beim relationalen Modell mittels Schlüsseln lösen. Je nach Beziehungstyp (1:1, 1:N oder N:M) werden Fremdschlüssel der Relation B in der Relation A angelegt, falls A und B in Beziehung stehen. Bei einem Beziehungstyp N:M wird eine zusätzliche Beziehungsrelation benötigt. Es werden jeweils der Fremdschlüssel von Relation A und Relation B in der Beziehungsrelation gespeichert. ([New06])

Laden von Objekten: Eine mögliche Lösung für das Laden von Objekten, besteht im Laden eines einzelnen Objekts mit einer festgelegten Objekt ID aus der Datenbank, um die dazugehörige Instanz in der objektorientierten Applikation zu erstellen. Bei diesem Vorgang muss die Datenbank nach dem Tupel mit der Objekt ID angefragt werden. Anschließend werden die Werte des Tupels in die Attribute des Objektes überführt. Dieses Vorgehen impliziert Performanceschwierigkeiten bei einer hohen Anzahl von zu ladenden Objekten, da für eine Menge von N Objekten N Abfragen an die Datenbank geschickt werden. Jedes Objekt verwaltet seine eigene Anfrage an die Datenbank. Bei relationalen Abfragen werden die Objekte in großen Mengen zurückgegeben, so dass ein Performanceeinbruch hervorgerufen wird.

Ein weiterer Aspekt der beim Laden von Objekten Berücksichtigung finden muss, ist die Art und Weise, wie die Anfragen an das bestehende System gestellt werden. Folgende Möglichkeiten stehen zur Verfügung ([Sch13], S. 19):

1. Alle Anfragen an die Datenbank werden ausschließlich mit SQL geschrieben. Es muss sichergestellt werden, dass alle Informationen für jedes Objekt vollständig sind.
2. Es wird eine eigene Anfragesprache entwickelt. Die eigens definierte Sprache sollte den Umgang mit Objekten erleichtern und anschließend in reines SQL umwandelbar sein.
3. Die Anfragen werden mittels eines Anfrageobjekts mit einem objektorientierten Application Programming Interface isoliert. Das Application Programming Interface bildet eine zusätzliche Softwarebibliothek. Später wird das Anfrageobjekt in reines SQL umgewandelt.

Es existiert keine einheitliche Präferenz, welche Lösung die Beste ist. Bei der reinen SQL Verwendung (1), wird die Applikation dauerhaft an ein Datenbankschema gebunden. Das Design des Schemas wird nicht mehr durch den objektorientierten Mapper definiert, sondern von der Anfrage, in Form von SQL. Die Schnittstelle zur Datenbank wird nicht abstrahiert. Das heißt es besteht eine Abhängigkeit von dem verwendeten DMBS. Bei einem Wechsel des DMBS muss auch die Applikation aufwendig geändert werden. ([Sch13], S. 19)

Option zwei ist sehr komplex, da eine eigene Abfragesprache entwickelt wird. Durch die Anforderung eines austauschbaren Datenbanksystems wird der Fokus dieser Arbeit auf dem Lösungsverfahren einem objektorientierten Application Programming Interface (3) liegen.

4 Application Programming Interface

Ein Application Programming Interface (API) bietet eine Abstraktion für ein Problem. Es gibt an, wie ein Programm mit Softwarekomponenten interagieren soll, die eine Lösung implementieren. Die API selbst wird normalerweise als Softwarebibliothek gespeichert, so dass sie von mehreren Anwendungen verwendet werden kann. Im Wesentlichen definieren API‘s wiederverwendbare Module einer bestimmten Funktionalität, die in Endbenutzeranwendungen integriert werden sollen. Dabei können die Module einzelne Funktionen oder mehrere hunderte von Klassen haben.

Eine moderne Applikation baut in vielen Fällen auf mehreren API auf, welche wiederum auf weitere API‘s zurückgreifen. Nicht die Implementierungsdetails, sondern die Funktionalität einer Komponente ist entscheidend für die gesamte Applikation. ([Chr16])

4.1. Erläuterung eines Application Programming Interfaces

Warum eine API in der eigens entwickelten Applikation verwendet werden sollte, lässt sich unter zwei Gesichtspunkten betrachten. Die API kann selbst entwickelt werden oder es kann eine bestehende API verwendet werden. Beide Varianten werden im Folgenden berücksichtigt. Durch das robuste Programmieren erhöht eine API im Allgemeinen die Stabilität eines Programmes. Bei der richtigen Gestaltung der API entstehen folgende Vorteile:

- **Verborgene Implementierung:** Durch eine verborgene Implementierung wird Flexibilität gewährleistet. Änderungen des Codes lassen sich leichter programmieren, ohne dass der End User etwas davon mitbekommt. Allerdings sollte die Kompatibilität der verschiedenen Versionen einer API, gegeben durch Updates, berücksichtigt werden.
- **erhöhte Lebensdauer:** API´s ermöglichen eine Entwicklung weg vom Spaghetti Code hin zu Modulen. Einzelne Module lassen sich besser warten und haben dadurch eine erhöhte Langlebigkeit.
- **Förderung der Modularisierung:** Eine API dient dazu, eine bestimmte Aufgabe oder einen bestimmten Anwendungsfall zu adressieren. API‘s bilden eine modulare Gruppierung von Funktionen mit einem bestimmten Fokus. Diese Gruppierungen handeln unabhängig von anderen Modulen oder Programmen.
- **Reduzierung von doppeltem Code:** In einem Spaghetti Code kann ein doppelter Code existieren. Dieser lässt sich meist in einer API zusammenfassen. Das heißt der Code muss nur einmal in der API aktualisiert werden, anstatt an allen Stellen im gesamten Code.
- **Entfernung von fest codierten Annahmen:** In vielen Programmen werden fest codierte Variablen übergeben. Mit Hilfe einer API können fest codierte Annahmen übergangen werden und ein abstrakter Ansatz entsteht.
- **Einfache Änderungen:** Durch die Modularisierung und die verborgene Implementierung lässt sich der Code leicht ändern.
- **leichtere Optimierung:** Da die API unabhängig von der Applikation besteht, lässt sich die Performance im Modul leichter optimieren. So lassen sich z.B. isolierte Performance Tests durchführen.

Einer der größten Vorteile besteht in der Wiederverwendbarkeit eines bereits existierenden Codes. In den Anfängen der Softwareentwicklung musste der Code für jede Anwendung neu geschrieben werden. Durch nun existierende kommerzielle und Open Source Bibliotheken ist es möglich, den Code der bestehenden Bibliotheken zu verwenden. Es gibt sehr viele API‘s, die frei zugänglich sind und in die eigene Applikation integriert werden können. Diese frei zugänglichen API‘s wurden stetig verfeinert und durch mehrere Entwickler getestet. Ein Nachteil der Verwendung bereits bestehender API‘s besteht darin, eine allgemeinere Schnittstelle zu erhalten als benötigt, die mit einer längeren Ladezeit einhergeht. ([Red11], S. 6ff)

Diese Entwicklung in der Softwareentwicklungsstrategie ist unter anderem das Ergebnis der Kräfte der Globalisierung ([Fri10]). Folgende Beispiele veranschaulichen, dass in einigen Fällen darauf verzichtet werden sollte, eine bestehende API zu verwenden und eine eigene Lösung zu entwickeln.

- **Lizenzierung:** Eine API kann alle benötigten Funktionalitäten beinhalten, aber nicht die passende Lizenzierung haben. Wird beispielsweise eine Open Source API genutzt, welche unter der Lizenz GNU General Public License (GPL) steht muss bei der Verwendung der API auch die entwickelte Software unter dieser Lizenz veröffentlicht werden. Dies impliziert eine vollständige Offenlegung des Quellcodes. Andere Lizenzen hingegen sind mit Kosten verbunden. ([Red11], S. 11)
- **Mangel an Einsicht:** Bei der Verwendung eines kommerziellen API kann es sein, dass der Quellcode nicht einzusehen ist. Entsteht bei der Programmierung ein Bug ist keine selbstständige Fehlerbehebung möglich. ([Red11], S. 11)
- **Mangel an Dokumentation:** Zu einigen API‘s existiert keine oder eine geringe Dokumentation. In diesem Fall ist es schwierig herauszufinden, was in der API implementiert ist. ([Red11], S. 11)

4.2. Datenbankabstraktionsschicht

Eine Datenbankabstraktionsschicht bildet eine Schnittstelle zwischen einer Applikation und einem Datenbanksystem wie SQL Server, Oracle, DB2, PostgreSQL oder SQLite. Jedes dieser Systeme bietet traditionell ihre eigene, auf ihr Produkt zugeschnittene Schnittstelle an. Damit ist es dem Programmierer möglich den Code für die Datenbankschnittstelle zu implementieren, welche er unterstützen möchte. ([AC15], S. 346)

Datenbankabstraktionsschichten reduzieren den Arbeitsaufwand, indem sie dem Entwickler eine konsistente API bereitstellen und die Datenbankmerkmale hinter dieser Schnittstelle soweit wie möglich ausblenden. Dies bedeutet, dass die Datenbankabstraktionsschicht nicht an ein bestimmtes Datenbanksystem gekoppelt ist. In zahlreichen Programmiersprachen gibt es viele Abstraktionsschichten mit unterschiedlichen Schnittstellen. ([Van09], S. 129)

Die Datenbankabstraktionsschicht vereinheitlicht den Zugriff auf Datenbanken, indem sie dem Anwendungsentwickler eine einzige Programmierschnittstelle zur Verfügung stellt. Da diese Schnittstelle nicht an eine bestimmte Abfragesprache (Teilmenge) gebunden ist und nur eine Schicht implementieren muss, um ihr Ziel zu erreichen, bietet sie Geschwindigkeit und Flexibilität. Da alle SQL-Dialekte einander ähnlich sind, können Anwendungsentwickler alle Sprachfeatures verwenden und möglicherweise konfigurierbare Elemente für

datenbankspezifische Fälle bereitstellen, wie z. B. Benutzer-IDs und Berechtigungsnachweise. Bei einer Schicht können dieselben Abfragen und Anweisungen auf einer Vielzahl von Datenbankprodukten mit einem geringen Aufwand ausgeführt werden.

Die häufigste Verwendung von Datenbankabstraktionsschichten findet sich in objektorientierten Programmiersprachen. In einer objektorientierten Sprache wie C++, Python oder Java kann eine Datenbank durch ein Objekt dargestellt werden, dessen Methoden und Member (oder deren Entsprechung in anderen Programmiersprachen) verschiedene Funktionalitäten der Datenbank darstellen. Eine Datenbankabstraktionsschicht bietet folgende Vorteile ([Wik19]):

- **Entwicklungszeitraum:** Entwickler können die API der Abstraktionsschicht unterstützen, so dass bei einer größeren Menge an verschiedenen Datenbanken, Entwicklungszeit gespart werden kann.
- **Vereinfachtes Deployment:** Mit Hilfe einer Abstraktionsschicht ist das Deployment beim Ersetzen oder Hinzufügen einer neuen Datenbank deutlich leichter.
- **Zukunftssicher:** Mit dem Aufkommen neuer Datenbanktechnologien müssen sich Softwareentwickler nicht an neue Schnittstellen anpassen.
- **Entwicklungstests:** Durch die Abstraktionsschicht lässt sich eine Produktionsdatenbank auf Desktop-Ebene im Arbeitsspeicher z.B. SQLite duplizieren und testen.
- **Erweiterte Datenbankfunktionen:** Die Datenbankabstraktionsschicht ermöglicht die Etablierung zusätzlicher Funktionen, die eine Datenbank normalerweise nicht hat. So ermöglicht die Schnittstelle DBvolution eine Standardabweichungsfunktion, die in einigen Datenbanksystemen zuvor nicht verfügbar war.

Allerdings ist eine Datenbankabstraktionsschicht auch mit gewissen Nachteilen behaftet:

- **Geschwindigkeit**: Jede Abstraktionsebene verringert die Gesamtgeschwindigkeit in Abhängigkeit davon, wie viel zusätzlicher Code ausgeführt werden muss. Je mehr eine Datenbankschicht von der nativen Datenbankschnittstelle abstrahiert und versucht, Features zu emulieren, die nicht in allen Datenbank-Backends vorhanden sind, desto langsamer ist die Gesamtleistung. Dies gilt insbesondere für Datenbankabstraktionsschichten, die versuchen, die Abfragesprache zu vereinheitlichen.
- **Abhängigkeit:** Die verwendete Schnittstelle kann veralten oder nicht mehr unterstützt werden, so dass eine unbegrenzte, dauerhafte Nutzung nicht garantiert ist.
- **Maskierte Operationen:** In vielen Fällen ist bei der Verwendung einer Datenbankabstraktionsschicht ein Debugging auf Ebene der Datenbank sehr schwer oder gar nicht möglich. Die Abstraktion der Datenbank erschwert den Zugriff. Viele Operationen werden nicht unterstützt. Dieses Problem nimmt mit der Größe und dem Umfang der Datenbank zu.

Die Funktionsweise einer Datenbankabstraktionsschicht wird in Abbildung 12 dargestellt.

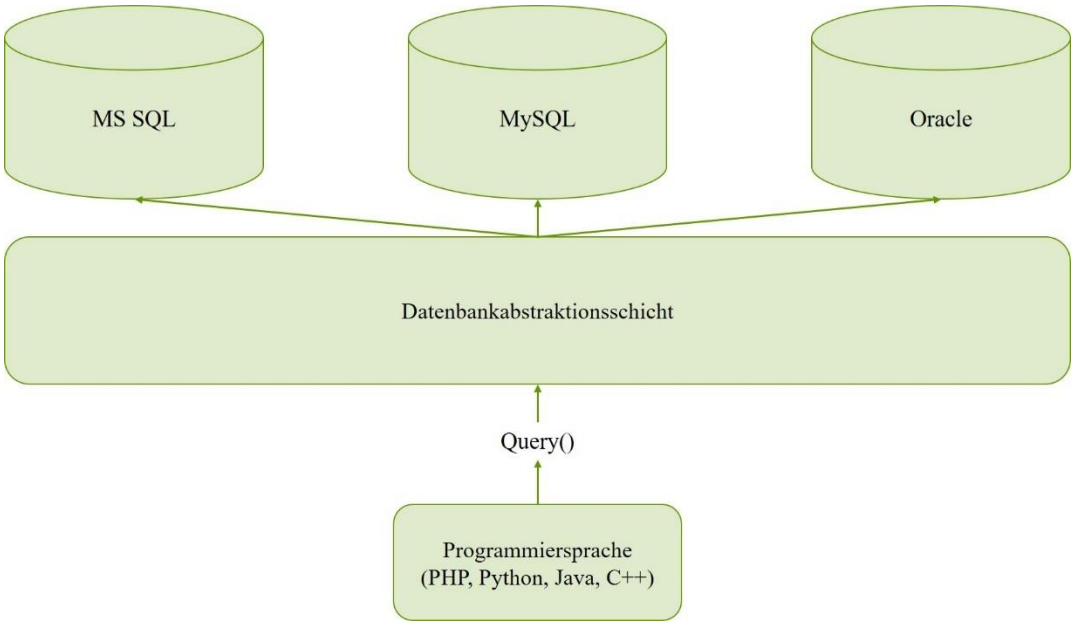


Abbildung 12: Funktionsweise einer Datenbankabstraktionsschicht nach [Tof15]

4.3. Frameworks

Ein Framework bildet ein Programmiergerüst zur Erstellung von Softwareanwendungen. Das Framework bietet einen Rahmen des Programmes, so dass die Architektur der Software bereits definiert ist. Das Framework ermöglicht dabei unteranderem den Zugriff auf bestehende API´s. ([Tec13])

Zusammenfassend lässt sich festhalten, dass eine Datenbankabstraktionsschicht, die in einer objektorientierten Sprache implementiert ist, aus einem objektrelationalen Mapper und einer Schnittstelle, die Anfragen an eine bestehende Datenbank übermittelt, besteht. Mittels der objektorientierten Sprache lässt sich eine grafische Oberfläche entwickeln. Es existieren mehrere Frameworks, welche das Erstellen einer Datenbankabstraktionsschicht ermöglichen. In Tabelle 3 sind eine Vielzahl von Frameworks abgebildet die ein objektrelationales Mapping ermöglichen und eine Schnittstelle zur Datenbank anbieten.

Tabelle 3: Frameworks zur Bildung einer Datenbankabstraktionsschicht [Wik19b]

Software	Plattform	Availability	License	Version
SubSonic	.NET 2.0	Open source	New BSD	3.0 / July 2009
Dapper	.NET 4.0	Open source	Apache License 2.0	1.8 NuGet
ECO	.NET 4.0	Commercial		ECO 6 Final (2011-04-18[2])
EntitySpaces	.NET 4.0	Open source	Modified BSD License	2012.1.0930.0 / October 4, 2012
Microsoft ADO.NET Entity Framework	.NET 4.5	Part of .NET 4.5	Apache License 2.0[4]	v6.0 (2014)
nHibernate	.NET 4.5	Open source	GNU Lesser General Public License	4.0 (2014-08-17[5])

Entity Framework Core	.net core	Open Source	Apache License 2.0	2.0
WebORB Integration Server	.NET, Java, PHP	Commercial & Open source	WebORB for .NET and WebORB for Java = Proprietary License, WebORB for PHP = Mozilla Public License	WebORB for .NET v.4.2 (Oct 2010), WebORB for Java v.4.0 (Sep 2010), WebORB for PHP (Sep 2008)
MyBatis/iBATIS	Cross-platform	Open source	Apache License 2.0	
ODB	Cross-platformC++	Dual-licensed	GNU General Public License and Proprietary License	2.4.0 / February 11, 2015
Apache Cayenne	Java Virtual Machine	Open source	Apache License 2.0	3.0.2 / July 21, 2011
DataNucleus	Java Virtual Machine	Open source	Apache License 2	4.1.0.RELEASE / May 19, 2015
EclipseLink	Java Virtual Machine	Open source	Eclipse Public License Version 1.0 (EPL) and Eclipse Distribution License Version 1.0 (EDL)	2.4.2 / July 4, 2013
Hibernate	Java Virtual Machine	Open source	GNU Lesser General Public License	4.2.5 / August 28, 2013
jOOQ	Java Virtual Machine	Open source	Apache License 2.0 and Proprietary License	3.2.0 / October 9, 2013
TopLink	Java Virtual Machine	Commercial	Oracle License	10g
DBIx::Class	Perl	Open source	Artistic License 1.0 & GPL	0.082820 / March 20, 2015[1]
Doctrine	PHP	Open source	MIT	2.4/April 8, 2014
RedBeanPHP	PHP	Open source	BSD License	4/April 1, 2014
Skipper	PHP	Commercial	Proprietary software	3.0
Django	Python	Open source	BSD licenses	2.1 (1 August 2018)
SQLAlchemy	Python	Open source	MIT License	1.2.14 / November 10, 2018
SQLObject	Python	Open source	LGPL	3.7.1 / February 2, 2019
Storm	Python	Open source	LGPL 2.1	0.20 / June 28, 2013
Her	Ruby	Open source	MIT License	0.8.6 / March 14, 2017[3]

Im Folgenden sollen zwei Frameworks näher betrachtet und verglichen werden. Zum einen das in Java programmierte Hibernate, zum anderen das in Python geschriebene SQLAlchemy.

Hibernate: Hibernate ist ein Java-Framework, dass die Entwicklung von Java-Anwendungen für die Interaktion mit der Datenbank vereinfacht. Es ist ein Open Source ORM Framework, welches sich als Alternative zu kommerziellen Produkten durchgesetzt hat. Abbildung 13 veranschaulicht

die Funktion von Hibernate. In Java programmierte Objekte werden in eine relationale Datenbank gemappt. ([Tut15])

Bei Betrachtung der allgemeinen Ansicht der Hibernate-Anwendungsarchitektur fällt auf, das Hibernate aus einer mehrschichtigen Architektur besteht. Die Schichten sollen den Benutzer in seiner Arbeit unterstützen, ohne dass Kenntnisse der zugrundeliegenden API‘s bestehen müssen.

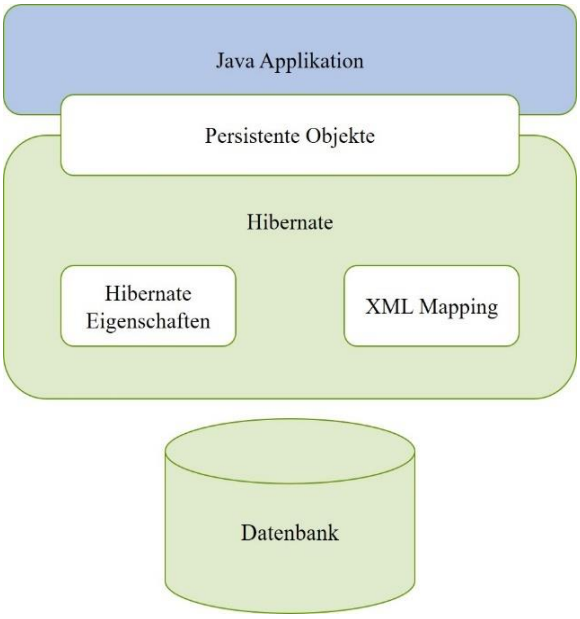


Abbildung 13: Allgemeine Architektur von Hibernate nach [Tut15]

Eine detailliertere Sicht auf die Architektur (siehe Abbildung 14) zeigt die Bestandteile von Hibernate. Das Hibernate Framework benutzt drei bestehende API‘s, um Anfragen an die zugrundeliegende Datenbank zu stellen: JDBC (Java Database Connectivity), JTA (Java Transaction API) und JNDI (Java Naming Directory Interface). Durch diese drei API‘s ist es Hibernate möglich, mit einer Vielzahl von relationalen Datenbanken zu interagieren: HSQL Database Engine, DB2/NT, MySQL, PostgreSQL, FrontBase, Oracle, Microsoft SQL Server Database, Sybase SQL Server und Informix Dynamic Server. Durch die Vielzahl an unterstützen Datenbanksystemen ist die Bildung eines generischen Ansatzes zwischen einer Anwendung und mehreren Datenbanken möglich. Innerhalb des Hibernate Frameworks liegen mehrere Klassen-Objekte, welche der Benutzer aufrufen kann ([Tut15]):

Configuration Object: Das Configutarion Object ist das erste Objekt, welches in einer Hibernate Applikation erstellt wird. Das Objekt repräsentiert eine Konfigurations- oder Eigenschafts-Datei, welche von Hibernate gefordert wird. Innerhalb des Configuration Objects werden zum einen die Database Connection definiert. Zum anderen das Class Mapping Setup gebildet. Hibernate ermöglicht es dem Nutzer zwischen drei verschiedenen Mappings zu wählen:

- Table per class hierarchy (Filter-Mapping)
- Table per subclass (horizontal Mapping)
- Table per concrete class (vertikales Mapping)

Zu jeder Klasse, die dauerhaft gespeichert werden soll, wird eine XML-Steuerungsdatei angelegt, die das ausgewählte Mapping spezifiziert. Das Anlegen der XML-Dateien kann auch automatisch durch Tools erfolgen, indem der Java-Quellcode ausgelesen wird. Durch die zur Verfügung

stehenden Mappings von Hibernate ist es möglich, dem Anwendungsfall entsprechend, ein optimales Mapping zu wählen. Dabei ist es ebenfalls möglich, verschiedene Mappings in einer Applikation zu kombinieren.

Session Factory Object: Das Configuration Object wird verwendet, um ein Session Factory Object zu erstellen, das wiederum Hibernate für die Anwendung mithilfe der bereitgestellten Konfigurationsdatei konfiguriert und die Instanziierung eines Session-Objekts ermöglicht. Die SessionFactory ist ein sicheres Threadobjekt und wird von allen Threads einer Anwendung verwendet. Ein Thread ist ein leichtgewichtiger Prozess, der vom Benutzer gesteuert wird. Das Session Factory Object wird während des Startens der Anwendung erstellt und zur späteren Verwendung aufbewahrt. Pro Datenbank wird ein Session Factory Object mit einer separaten Konfigurationsdatei benötigt.

Session Object: Eine Session wird verwendet, um eine physische Verbindung mit einer Datenbank herzustellen. Das Session-Objekt ist so konzipiert, dass es jedes Mal instanziiert werden kann, wenn eine Interaktion mit der Datenbank erforderlich ist. Persistente Objekte werden gespeichert und über ein Session Objekt abgerufen. Eine Session sollte nicht zu lange geöffnet werden, da der Prozess über einen längeren Zeitraum nicht stabil ist.

Transaction Object: Eine Transaktion stellt eine Arbeitseinheit mit der Datenbank dar. Die meisten relationalen Datenbanksysteme unterstützen die Transaktionsfunktionalität. Transaktionen im Ruhezustand werden von einem zugrundeliegenden Transaktionsmanager und einer Transaktion von JDBC oder JTA verarbeitet.

Das „Transactions Object“ ist ein optionales Objekt, welches in einer Hibernate-Applikation nicht verwendet werden muss.

Query Object: Query Objects verwenden SQL- oder HQL-Abfragesprachen (Hibernate Query Language), um Daten aus der Datenbank abzurufen und Objekte zu erstellen. Eine Anfrageinstanz wird verwendet, um Anfrageparameter zu binden, die Anzahl der von der Anfrage zurückgegebenen Ergebnisse zu begrenzen und die Anfrage schließlich auszuführen.

HQL ist eine objektorientierte Anfragesprache, die SQL ähnelt. Anstatt Tabellen und Spalten zu bearbeiten, arbeitet HQL mit persistenten Objekten und deren Eigenschaften. HQL-Anfragen werden von Hibernate in herkömmliche SQL-Anfragen übersetzt, die wiederum Aktionen in der Datenbank ausführen. SQL-Anweisungen können direkt mit Hibernate unter Verwendung von reinem SQL verwendet werden. Unter Berücksichtigung von Datenbankportabilitätsproblemen wird von der reinen Verwendung von SQL abgeraten. Da wie in Kapitel 3.6 erwähnt, die Applikation sich an einem Datenbankschema bindet.

Criteria Object: „Criteria Objects“ werden zum Erstellen und Ausführen objektorientierter Kriterien und Abfragen zum Abrufen von Objekten verwendet.

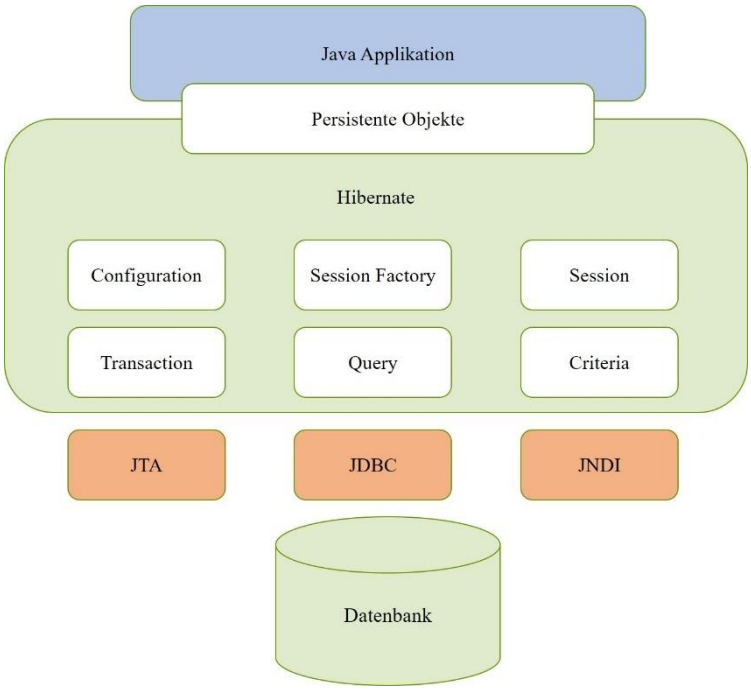


Abbildung 14: Detaillierte Ansicht von Hibernate nach ([Tut15])

SQLAlchemy: SQLAlchemy ist ein Open-Source-SQL-Toolkit und ORM-Framework, welches in der Programmiersprache Python entwickelt wurde. Nach eigenen Angaben des Entwicklers Michael Bayer löst das Framework die Problematik des „object-relational impedance mismatch“ mit einem modernen Ansatz. So basiert der häufigste Kritikpunkt von ORMs auf der Annahme, dass die Verwendung eines ORMs dazu dient, eine relationale Datenbank zu "verbergen", die Aufgaben zu übernehmen, eine Interaktion mit der Datenbank zu erstellen und diese auf ein Implementierungsdetail zu reduzieren. Der Ansatz der Verschleierung beinhaltet, dass die Möglichkeit relationale Strukturen zu entwerfen und abzufragen, nicht dem Entwickler zur Verfügung steht, sondern stattdessen von einer undurchsichtigen Bibliothek verwaltet wird. Nach Bayer bilden allerdings die relationalen Strukturen und funktionsfähigen SQL-Abfragen den Kern des Anwendungsdesigns. Wie diese Strukturen in Anfragen entworfen, organisiert und manipuliert werden sollen, hängt nicht nur davon ab, welche Daten gewünscht werden, sondern auch von der Informationsstruktur. Wenn dieses Dienstprogramm verborgen ist, bietet die Verwendung einer relationalen Datenbank keinen Mehrwert. ([Bay05])

Von den Benutzern von SQLAlchemy wird gefordert, die relationale Form ihrer Daten zu berücksichtigen. Bei einer Verbergung der relationalen Form entsteht ein begrenzter Nutzen einer relationalen Datenbank, wodurch alle klassischen Probleme der object-relational impedance mismatch hervorgerufen werden.

Die Zuordnung eines Objektmodells zu einem Schema und das Verwalten über SQL-Anfragen ist eine häufige Aufgabe. Die Automatisierung dieser Aufgaben durch Tools ermöglicht die Entwicklung einer Anwendung, die übersichtlicher, leistungsfähiger und effizienter ist. So kann die Anfrage, die für die manuelle Entwicklung dieser Vorgänge erforderlich wäre, deutlich schneller ausgeführt werden. ([Bay05])

Zu diesem Zweck bezeichnet SQLAlchemy sich selbst als Toolkit, welches den Entwickler als Konstrukteur aller relationalen Strukturen und Verbindungen zwischen den Strukturen und der Anwendung sieht und nicht als passiven Konsumenten von Entscheidungen, die von einer

Bibliothek getroffen werden. Durch das Aufzeigen von relationalen Konzepten ermutigt SQLAlchemy den Entwickler, eine benutzerdefinierte, jedoch vollständig automatisierte Interaktionsschicht zwischen der Anwendung und der relationalen Datenbank anzupassen. Die Innovation von SQLAlchemy besteht darin, einen hohen Automatisierungsgrad zu ermöglichen, ohne dabei die Kontrolle über die relationale Datenbank zu beeinträchtigen. ([Bay05])

SQLAlchemy möchte jede Ebene der Datenbankinteraktion als umfassende API verfügbar machen. Diese Aufgabe wird in die zwei Hauptkategorien „Core“ und „ORM“ unterteilt. Der Core umfasst die Interaktion mit der Python Database API (DBAPI), das Rendern von textuellen SQL-Anweisungen, die von der Datenbank verstanden werden und die Schemaverwaltung. Diese Funktionen werden alle als öffentliche API‘s dargestellt. Der ORM ist eine spezifische Bibliothek, die auf dem Core aufgebaut ist. Die mit SQLAlchemy bereitgestellte ORM ist nur eine von mehreren möglichen Objektabstraktionsschichten, die auf dem Core erstellt werden können. Dabei können ORM und Core unabhängig voneinander genutzt werden. Abbildung 15 veranschaulicht die Architektur von SQLAlchemy.

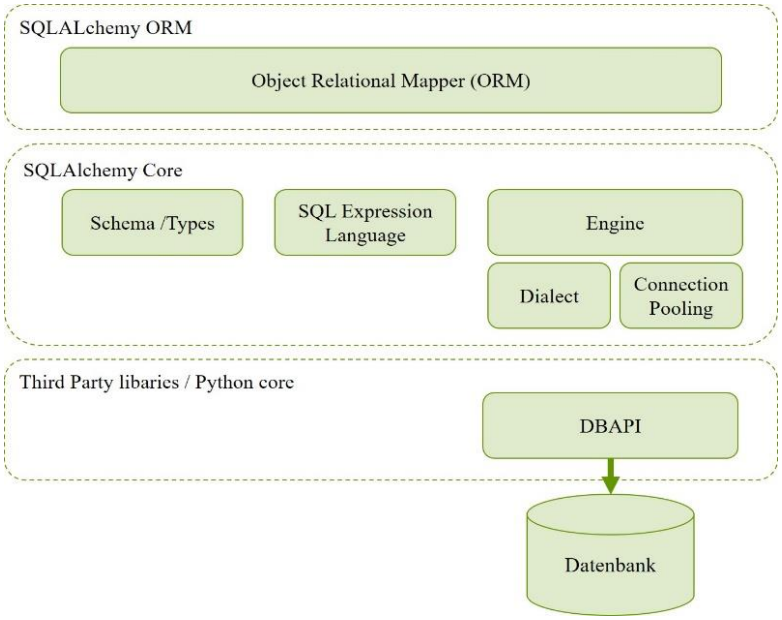


Abbildung 15: Architektur von SQLAlchemy nach [Bay05]

Die Trennung von ORM und Core hat sowohl Vor- als auch Nachteile. Der in SQLAlchemy vorhandene explizite Core führt dazu, dass der ORM von der Datenbank abgebildete Klassenattribute mit einer, als Tabelle bezeichneten, Struktur verknüpft und nicht direkt mit den in der Datenbank ausgedrückten Spaltennamen als String. Des Weiteren führt der Core dazu eine SELECT-Anfrage zu erzeugen, die eine Struktur namens „select“ verwendet, anstatt Objektattribute direkt in einer String-Anweisung zusammenzusetzen. Der Core ermöglicht Ergebniszeilen durch eine Fassade namens „ResultProxy“ zu empfangen. Diese ordnet die Auswahl transparent jeder Ergebniszeile zu, anstatt Daten direkt von einem Datenbankcursor an ein benutzerdefiniertes Objekt zu übertragen. Elemente des Cores sind in einer sehr einfachen ORM-zentrierten Anwendung möglicherweise nicht sichtbar. Da der Core jedoch sorgfältig in den ORM integriert ist, um einen Übergang zwischen ORM und Core-Konstrukten zu ermöglichen, kann eine komplexere ORM-zentrierte Anwendung eine oder zwei Ebenen nach unten verschoben werden, um die Datenbank je nach Situation abzustimmen. Mit der Entwicklung von SQLAlchemy ist die Core-API im

regulären Gebrauch weniger explizit geworden, da der ORM immer komplexere und umfassendere Muster bietet. ([Bay05])

Ein Nachteil der Architektur und Trennung von ORM und Core liegt darin, dass Anweisungen mehrere Schritte durchlaufen müssen. Dies führt zu einer verlangsamten Laufzeit. Der Hauptgrund für die langsame Laufzeit sind die vielen einzelnen Funktionsaufrufe. Herkömmliche Maßnahmen zur Verbesserung der Laufzeit liegen in der Verkürzung von Aufrufen durch Neuanordnung und Inlining sowie das Ersetzen von leistungskritischen Bereichen durch C Code. Beim Inlining wird der Code, der aufzurufenden Funktion an Stelle des Aufrufs kopiert ([Sta09]). Die Programmiersprache C arbeitet näher an der Hardware als Python und bietet somit eine schnellere Laufzeit. Um die Leistung zu verbessern, greift SQLAlchemy sowohl auf das Inlining als auch auf das Ersetzen mit C Code zurück. Mit Hilfe und Entwicklung des schnellen Compilers PyPy für Python konnte die Laufzeit noch weiter reduziert werden. PyPy ist ein Compiler der in Sonderfällen sogar schneller arbeitet als C oder Java ([Fij11]). Hierdurch konnten die Leistungsprobleme behoben werden, ohne den bestehenden Code durch C zu ersetzen. ([Bay05])

Database API und Core: SQLALchemy benutzt die zum Python Core gehörende „PEP 249 -- Python Database API Specification v2.0“ ([Lem17]). Diese API wird genutzt, um auf der untersten Ebene eine Verbindung zur Datenbank herzustellen. Durch ein im Core definiertes Dialektsystem lassen sich verschiedene Datenbanksysteme ansprechen. Innerhalb des Dialektsystems liegen mehrere Dialektklassen. Diese ermöglichen eine Kommunikation mit folgenden Datenbanksystemen: PostgreSQL, MySQL, SQLite, Oracle, Microsoft SQL Server, Firebird und Sybase. Durch das Dialektsystem lassen sich weitere externe Dialekte etablieren. Darunter fallen zum Beispiel ([Bay05a]):

- ibm_db_sa - driver for IBM DB2 and Informix.
- PyHive - driver for Apache Hive and Presto.
- sqlalchemy-redshift - driver for Amazon Redshift, adapts the existing PostgreSQL/psycopg2 driver.
- sqlalchemy-drill - driver for Apache Drill.
- sqlalchemy-hana - driver for SAP Hana.
- sqlalchemy_exasol - driver for EXASolution.
- sqlalchemy-sqlany - driver for SAP Sybase SQL Anywhere, developed by SAP.
- sqlalchemy-monetdb - driver for MonetDB.
- snowflake-sqlalchemy - driver for Snowflake.
- sqlalchemy-tds - driver for MS-SQL, on top of python-tds.
- crate - driver for CrateDB.

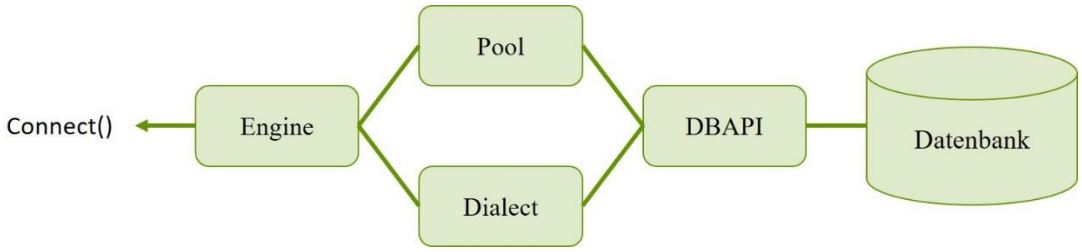


Abbildung 16: Verbindung zur Datenbank in SQLAlchemy nach [Bay05a]

Abbildung 16 zeigt die Interaktion des Cores mit der DBAPI. Die DBAPI ist eine API, welche eine Verbindung zur Datenbank herstellen kann. Durch den Pool wird die Adresse der zugrundeliegenden Datenbank bestimmt, welche an die DBAPI weitergeleitet wird. Innerhalb des Pools lassen sich mehrere Datenbanken anbinden. Insgesamt bilden der Pool und das Dialektsystem die Engine. Sobald die Engine erstellt wird, kann sie entweder direkt für die Interaktion mit der Datenbank verwendet werden oder an ein Session-Objekt übergeben werden, um mit dem ORM zu arbeiten. Eine Konfiguration einer Engine wird in Listing 2 dargestellt.

Listing 2: engine.py: Erstellung einer Engine [Bay05a]

```
from sqlalchemy import create_engine

engine = create_engine(
'postgresql://scott:tiger@localhost:5432/mydatabase')
```

In diesem Beispiel bildet „postgresql“ den Dialekt der Datenbank. Die Datenbank selbst liegt in dem Pfad „localhost:5432/mydatabase“ und ist zugänglich für den Benutzer „scott“ mit dem Passwort „tiger“. ([Bay05a])

Wenn die Datenbankkonnektivität eingerichtet ist, besteht der nächste Schritt darin, die Erstellung und Bearbeitung von Backend-agnostischen SQL-Anweisungen bereitzustellen. Um dies zu erreichen, muss zunächst definiert werden, wie sich auf die in einer Datenbank vorhandenen Tabellen und Spalten bezogen wird - das sogenannte "Schema". Tabellen und Spalten zeigen, wie Daten organisiert sind. Die meisten SQL-Anweisungen bestehen aus Ausdrücken und Befehlen, die auf diese Strukturen verweisen. ([Bay05])

Eine ORM- oder Datenzugriffsschicht muss programmgesteuert auf die SQL-Sprache zugreifen. An der Basis befindet sich ein programmatisches System zur Beschreibung von Tabellen und Spalten. Hier bietet SQLAlchemy die erste Aufteilung von Core und ORM, indem es die Table- und Column-Konstrukte anbietet, die die Struktur der Datenbank unabhängig von der Modellklassendefinition eines Benutzers beschreiben. Der Grund für die Unterscheidung der Schemadefinition von der objektrelationalen Abbildung ist, dass das relationale Schema eindeutig in Bezug auf die relationale Datenbank entworfen werden kann, einschließlich potenzieller plattformspezifischer Details, ohne von objektrelationalen Konzepten durcheinander gebracht zu werden. Von der ORM-Komponente unabhängig zu sein, bedeutet auch, dass das Schemabeschreibungssystem für alle anderen objektrelationalen Systeme, die auf dem Core basieren, ebenso nützlich ist. ([Bay05])

Das Tabellen- und Spaltenmodell fällt in den Bereich der sogenannten Metadaten und bietet ein Sammlungsobjekt namens MetaData zur Darstellung einer Sammlung von Table-Objekten. Die Struktur leitet sich hauptsächlich von Martin Fowlers Beschreibung von "Metadata Mapping" in Patterns der Enterprise Application Architecture ab. ([Bay05])

Innerhalb des Cores liegt die SQL Expression Language. Dieses Objekt dient der Generierung von SQL. Python-Objekte und -Ausdrücke werden verwendet, um eine Baumstruktur zu konstruieren, die die Python-Operatoren so verwendet, dass Operatoren SQL-Anweisungsverhalten erhalten können. Bei diesem Ansatz repräsentieren Python-Objekte lexikalische Teile eines SQL-Ausdrucks. Methoden für diese Objekte sowie überladene Operatoren generieren daraus neue lexikalische Konstrukte. ([Bay05])

SQL-Ausdrücke innerhalb einer Programmiersprache müssen mittels Precompiler übersetzt werden (siehe Kapitel 3.5). SQLAlchemy benutzt für dieses Vorgehen eine Klasse namens Compiled, welche SQL-Ausdrücke mittels Methoden übersetzt. In Abbildung 17 wird eine Aufteilung des Ausdrucks in Methoden überführt.

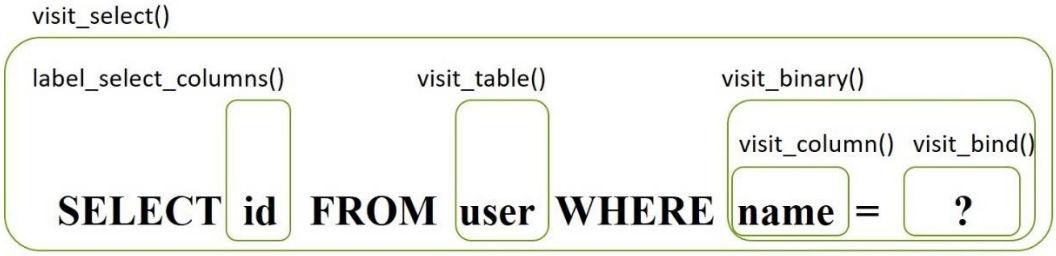


Abbildung 17: Aufrufhierachie von Statements beim Compilieren nach [Bay05]

ORM: Der ORM folgt dem bekannten „Metadata Mapping“, das in dem Buch „Patterns of Enterprise Architecture“ beschrieben wird. Insgesamt basiert der SQLAlchemy-ORM auf den von Fowler beschriebenen Vorgehensweisen. Zudem wurde der ORM von SQLAlchemy stark von dem zuvor beschriebenen Hibernate-Framework beeinflusst. SQLAlchemy erlaubt es zwei Variationen des Metadata Mapping zu benutzen. Zum einen existiert das „classical mapping“, zum anderen das „declarative mapping“. ([Bay05a])

Bei dem „classical mapping“ wird eine objektrelationale Datenzuordnung auf eine vorhandene Benutzerklasse angewendet. Diese Form betrachtet ein Tabellen-Objekt und eine benutzerdefinierte Klasse als zwei individuell definierte Entitäten, die über eine als Mapper bezeichnete, Funktion zusammengefügt werden. Nachdem der Mapper auf eine benutzerdefinierte Klasse angewendet wurde, übernimmt die Klasse neue Attribute, die den Spalten in der Tabelle entsprechen. Die Erstellung einer Klasse wird in Listing 3 dargestellt.

Listing 3: mapping.py: Erstellung eines Klasse [Bay05a]

```
class User(object):
  pass

mapper(User, user_table)
# now User has an ".id" attribute
User.id
```

Der Mapper kann der Klasse auch andere Arten von Attributen hinzufügen, darunter Attribute, die auf andere Arten von Objekten verweisen oder beliebigen SQL-Ausdrücken entsprechen. ([Bay05])

Neben dem „classical mapping“ hat sich das „declarative mapping“ entwickelt, welches auf einem Konfigurationssystem aufbaut. Dieses Konfigurationssystem ähnelt anderen Klassendeklarationssystemen, die von anderen objektbezogenen Tools verwendet werden.
In diesem System definiert der Endbenutzer explizit Attribute innerhalb der Klassendefinition, wobei jedes Attribut in der Klasse, das zugeordnet werden soll, dargestellt wird. Das Tabellen-Objekt wird in den meisten Fällen weder explizit erwähnt noch wird die Mapper-Funktion angegeben. Nur die Klasse, die Column-Objekte und andere ORM-bezogene Attribute werden benannt([Bay05a]). Das Hinzufügen von Attributen veranschaulicht Listing 4.

Listing 4: mapping.py: Hinzufügen von Attributen [Bay05a]

```
class User(Base):
    __tablename__ = 'user'
    id = Column(Integer, primary_key=True)
```

Bei diesem Verfahren wird nur die Art und Weise des „classical mappings“ reorganisiert, so dass das „declarative mapping“ auf das „classical mapping“ aufbaut. Das Verfahren des „declarative mapping“ ist für Erweiterungen auf Klassenebene zugänglicher als ein klassisches Mapping. Unabhängig davon, ob ein klassisches oder ein deklaratives Mapping verwendet wird, übernimmt eine zugeordnete Klasse neue Verhaltensweisen, mit denen SQL-Konstrukte anhand ihrer Attribute ausgedrückt werden können. ([Bay05])

Query Objekt und Session Objekt: SQLAlchemy initiiert das gesamte Verhalten beim Laden von Objekten über ein Objekt namens Query. Der Basisstatus Query enthält die Entitäten, wie die Liste der zugeordneten Klassen und/ oder einzelnen SQL-Ausdrücke, die abgefragt werden sollen. Die Query enthält außerdem einen Verweis auf die Session, die die Konnektivität zu einer oder mehreren Datenbanken darstellt sowie einen Cache mit Daten, welche in Bezug auf Transaktionen auf diesen Verbindungen gesammelt werden.

Das Session Objekt ermöglicht den allgemeinen Zugriff für den Endbenutzer auf den ORM, das heißt, das Speichern und Laden von Daten. Die Session bietet den Ausgangspunkt für Abfragen und Persistenz Operationen für eine gegebene Datenbankverbindung. Dabei dient die Session nicht nur als Datenbankkonnektivität, sondern auch als aktive Referenz für die Menge aller zugeordneten Entitäten, die relativ zu dieser Sitzung im Speicher vorhanden sind. Auf diese Weise implementiert die Sitzung eine Fassade für die von Fowler identifizierte „identity map“ and „unit of work patterns“. Die „identity map“ verwaltet eine datenbankidentitätseindeutige Zuordnung aller Objekte für eine bestimmte Sitzung, wodurch die Probleme vermieden werden, die durch doppelte Identitäten verursacht werden. Die „unit of work patterns“ baut auf der „identity map“ auf, um ein System zur Automatisierung des Prozesses bereitzustellen, bei dem alle Statusänderungen in der Datenbank auf möglichst effektive Weise beibehalten werden. Teil der „unit of work“ ist die Flush Methode, die innerhalb des Session Objektes liegt. Die Aufgabe der Methode besteht darin, den gesamten ausstehenden Status in einer bestimmten Session in die

Datenbank zu verschieben, um die von der Sitzung verwalteten, neuen, fehlerhaften und gelöschten Sammlungen zu leeren. Nach Abschluss des Vorgangs stimmen der In-Memory-Status der Sitzung und der Inhalt der aktuellen Transaktion überein. Der In-Memory-Status beschreibt den Status im Arbeitsspeicher. ([Bay05])

Vergleich von Hibernate und SQLAlchemy

Grundsätzlich unterscheiden sich die beiden Frameworks in der Programmiersprache. Hibernate ist in Java geschrieben und SQLAlchemy in Python. Beide Programmiersprachen bieten Vor- als auch Nachteile. Diese ergeben sich unter anderem im Bereich der Performance, der Syntax, der Dynamik und der Portabilität ([Edu19]). Beide Programmiersprachen tauchen in den Indexen von TIOBE, PYPL und RedMonk in den Top 10 der beliebtesten Programmiersprachen 2018 auf ([Moh18]). Da die zugrundeliegende Architektur beider Frameworks auch in einer anderen Programmiersprache entwickelt werden kann, wird hier nicht weiter auf die spezifischen Vor- und Nachteile der Programmiersprachen eingegangen.

Beim Betrachten der allgemeinen Architektur fällt auf, dass Hibernate aus einer Schicht besteht die mehrere Objekte beinhaltet. SQLAlchemy hingegen besteht aus zwei Schichten, die getrennt voneinander dem Benutzer zur Verfügung stehen. Hibernate hat das Ziel, das zugrundeliegende relationale Konstrukt dem Benutzer zu verschleiern, so dass er keinen Bezug zu einem relationalen System mehr hat. SQLAlchemy hingegen versucht die Rolle des Entwicklers als Konstrukteur aller relationalen Strukturen und Verbindungen zwischen diesen Strukturen und der Anwendung zu betonen.

Beide Frameworks benutzen 3rd Party API‘s, um eine Verbindung mit der oder den zugrundeliegende/n Datenbank/en herzustellen. Hibernate greift auf insgesamt drei verschiedene API‘s zurück (siehe Abbildung 14), um eine große Anzahl an Dialekten von verschiedenen Datenbanksystemen abzudecken und das Binden von Objekten zu Namen zu ermöglichen.
SQLAlchemy verwendet ausschließlich eine API, um eine Verbindung mit der Datenbank herzustellen. Das Hinzufügen von neuen Dialekten ist in beiden Frameworks möglich. Bei Versionsanpassungen kann jedoch die Wartung und Sicherstellung der Kompatibilität bei drei verschiedenen 3rd Party API‘s aufwendiger sein als bei einer.

SQLAlchemy oreintiert sich beim Mapping an dem von Martin Fowlers in „Patterns of Enterprise Application Architecture“ beschriebenen Metadata Mapping und bietet zum einen das classical mapping und zum anderen das declarative mapping an. Das declarative mapping ist durch eine Weiterentwicklung des classical mapping entstanden und bietet eine Erweiterungsmöglichkeit auf Klassenebene. Das Mapping erfolgt innerhalb von Python-Objekten. Hibernate hingegen ermöglicht die Verwendung von drei verschiedenen Mappings. Diese lassen sich auch untereinander kombinieren. Durch die verschiedenen Mappings lässt sich situationsbedingt die Performance steigern. Die Mappings werden in XML-Steuerungsdateien gespeichert. Da XML ein textbasiertes Format beinhaltet, kann es bei größeren Mappings größeren Speicherbedarf verursachen. Durch den erhöhten Speicherbedarf existiert eine höhere Übertragungsmenge. XML Formate dagegen sind sehr kompatibel und lassen sich in anderen Anwendungen verwenden.

In Bezug auf das Query Objekt hat Hibernate eine eigene objektorientierte Anfrage-Sprache entwickelt. Die Hibernate Query Language arbeitet nicht mit Tabellen und Spalten wie SQL, sondern mit persistenten Objekten und deren Eigenschaften. HQL-Abfragen werden von

Hibernate in herkömmliche SQL-Abfragen übersetzt, die wiederum Aktionen für die Datenbank ausführen. Eine Verwendung von reinem SQL ist ebenfalls möglich. SQLAlchemy hat dieses Konzept übernommen und ermöglicht, über isolierte Anfrageobjekte Querys zu stellen, die in reines SQL umgewandelt werden. Wie auch Hibernate ermöglicht SQLAlchemy das Verwenden von reinem SQL. Allerdings rät Hibernate davon ab, reines SQL zu verwenden. SQLAlchemy bietet durch den Ansatz des Cores eine bessere Unterstützung von reinem SQL. Für die Übertragung der Anfrageobjekte in die Datenbank benutzen beide Frameworks einen temporären Speicher namens „Session“. Die Session ermöglicht Insert-, Update- und Delete-Befehle.

Insgesamt ähneln sich Hibernate und SQLAlchemy stark in ihrer Architektur und Umsetzung. Beide Architekturen wurden nach dem, in Martin Fowlers Buch „Patterns of Enterprise Application Architecture“ beschrieben Kapitel „Mapping to Relational Databases“, konzipiert. In Tabelle 4 werden nochmals die Unterschiede und Gemeinsamkeiten von Hibernate und SQLAlchemy zusammengefasst.

Tabelle 4: Vergleich von Hibernate und SQLAlchemy

	Hibernate	Java
Programmiersprache	Java	Python
Philosophie	Verbergung der relationalen Strukturen	Betonung der relationalen Struktur
Aufbau	Eine Schicht	Zwei unabhängige Schichten
Verwendung von 3rd Party API	3 API: JTA, JDBC, JNDI	1 API: DBAPI
Art des Daten Mapping	• Table per class hierarchy (Filter-Mapping) • Table per subclass (vertikales Mapping) • Table per concrete class (horizontales Mapping)	Metadata Mapping: classical, declarative
Form des Daten Mappings	XML-Steuerungsdatei	Python-Objekte
Query Objekte	Hibernate Query Language, SQL	Anfrageobjekte werden isoliert. Später wird das Anfrageobjekt in reines SQL umgewandelt; SQL
Übertragung von INSERT, UPDATE, DELETE	Temporärer Speicher in Form einer Session	Temporärer Speicher in Form einer Session

Beide Frameworks sind in ihrer Aufgabe etablierte Produkte und erfüllen ihren Zweck der Erstellung eines ORM und einer Datenbankabstraktionsschicht. Bei einem Vergleich auf stackshare, einer Umfrageplattform, ergab sich, dass 486 Unternehmen Hibernate nutzen und 136 Unternehmen SQLAlchemy. Bei den auf Stack Overflow geposteten Fragestellungen zeigt sich eine deutliche Tendenz zu Hibernate mit 76.500 Fragen zu 13.100 Fragestellung bei SQLAlchemy. Betrachtet man die Redditpoints und GitHub Stars, liegt SQLAlchemy vorne. ([Sta19])
Dies könnte daran liegen, dass die Nutzung von Python als Programmiersprache in den letzten Jahren zugenommen hat. Durch die einfache Syntax ist Python leicht zu erlernen und findet

Anwendung im Bereich des Machine Learnings und der Datenanalyse. Abbildung 18 zeigt die steigende Nutzung von Python. Durch die steigende Verwendung von Python erweitern sich die zur Verfügung stehenden Bibliotheken. So stehen 172,198 Projekte Anfang 2019 zur Verfügung ([PSF19]). Im Vergleich der in Tabelle 5 dargestellten Bibliotheken anderer Programmiersprachen bietet Python deutlich mehr Bibliotheken an. ([Rob17])

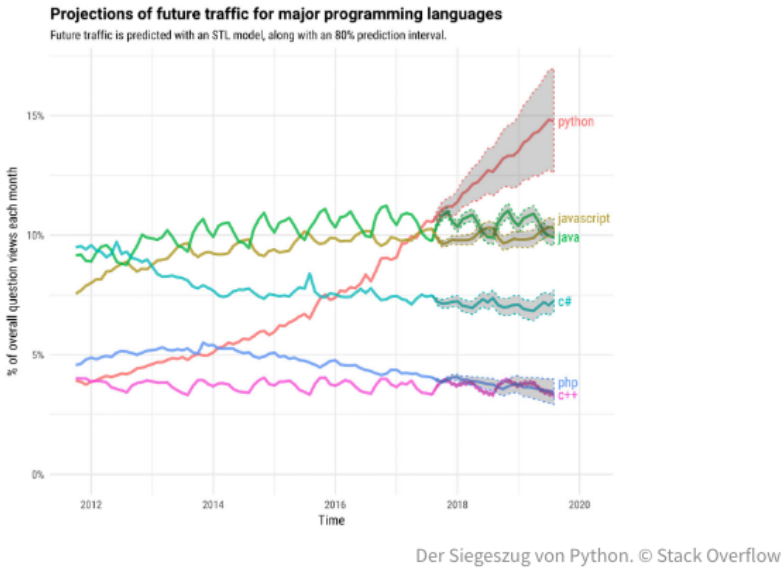


Abbildung 18: Wachstum von Python [Rob17]

Tabelle 5: Übersicht von Programmbibliotheken [Wik19a]

Sprache	Teile/Pakete	Header/Klassen	Funktionen/Methoden/Konstruktoren
C (C89+Amendments)	1	18	142
C (C99)	1	24	482
C++	1	32 + 18 (C89)	
Java 2 (JDK 1.2)	62	1.287	≈ 18.000
Java 6	202	3.850	21.881
.Net 1.0	41	3.581	35.470
.Net 1.1	43	3.818	37.556
.Net 2.0	51	7.419	74.607
.Net 3.0	80	10.639	102.613
.Net 3.5	98	11.417	109.657

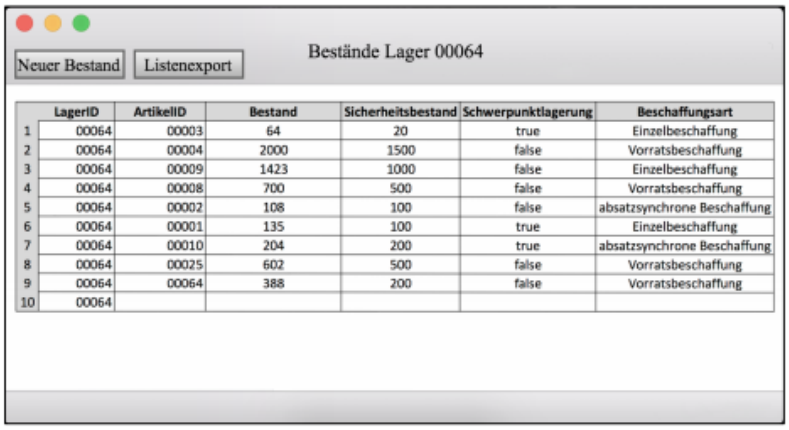
5 Konzeptionelle Entwicklung einer Datenbankabstraktionsschicht

Für die Entwicklung eines Konzeptes einer Datenbankabstraktionsschicht zur Kommunikation zwischen Maßnahmen und einer zugrundeliegenden Datenbank wird im Folgenden ein technisch möglicher Aufbau, der in Kapitel 2.1 beschriebenen Maßnahmen näher untersucht.

Für die technische Umsetzung der Maßnahmen in einem Handelsnetzwerk orientiert sich diese Arbeit an dem in „Improving the performance of logistics assistance system for materials trading networks by grouping similar actions“ von Markus Rabe et al. beschriebenen „action type designer“. Zudem wird die Masterthesis „Entwicklung einer Methodik zur grafischen Darstellung von Maßnahmen in Werkstoffhandelsnetzwerken“ von Erkut Baydar unter der Betreuung von Markus Rabe und Dominik Schmitt herangezogen, um ein Verständnis für eine grafische Darstellung von Maßnahmen zu erlangen.

Rabe et al. beschreiben den „action type designer“ als eine textuelle oder grafische Oberfläche, die es ermöglicht, einem Anwender benutzergenerierte Maßnahmen für ein Logistiknetzwerk zu erzeugen ([RAS18]). Generierte Maßnahmen lassen sich in einem Aktionstypverzeichnis speichern. Das Modellieren der Maßnahmen wird durch vordefinierte Konstrukte einer domänenspezifischen Modellierungssprache ermöglicht. Die Modellierungssprache beschränkt sich auf die Beschreibung von Maßnahmen in logistischen Netzwerken des Materialhandels. Daher kann die Komplexität der Modellierung von Maßnahmen im Vergleich zur Implementierung der Maßnahmen auf das Datenmodell direkt, z.B. durch das Schreiben von spezifischem SQL-Code, verringert werden. ([RAS18])

Eine beispielhafte Betrachtung der in Kapitel 2.1 vorgestellten Maßnahme der Bestandsänderung eines Artikels, lässt sich im Aktionstypverzeichnis speichern. Eine grafische Oberfläche zur Darstellung der Maßnahme einer Bestandsänderung zeigt Abbildung 19, welche von Erkut Baydar erarbeitet wurde.



	LagerID	ArtikelID	Bestand	Sicherheitsbestand	Schwerpunktlagerung	Beschaffungsart
1	00064	00003	64	20	true	Einzelbeschaffung
2	00064	00004	2000	1500	false	Vorratsbeschaffung
3	00064	00009	1423	1000	false	Einzelbeschaffung
4	00064	00008	700	500	false	Vorratsbeschaffung
5	00064	00002	108	100	false	absatzsynchrone Beschaffung
6	00064	00001	135	100	true	Einzelbeschaffung
7	00064	00010	204	200	true	absatzsynchrone Beschaffung
8	00064	00025	602	500	false	Vorratsbeschaffung
9	00064	00064	388	200	false	Vorratsbeschaffung
10	00064					

Abbildung 19: Grafische Oberfläche für eine Maßnahme [Bay17]

Anhand des beschriebenen „action type designer“ und der grafischen Umsetzung einer Maßnahme lässt sich ein technischer Aufbau für Maßnahmen, welche an eine zugrundeliegende Datenbank übermittelt werden sollen, entwickeln. Abbildung 20 veranschaulicht die Architektur. Da der „action type designer“ eine grafische Oberfläche anbietet wird als Hostsprache eine ob-

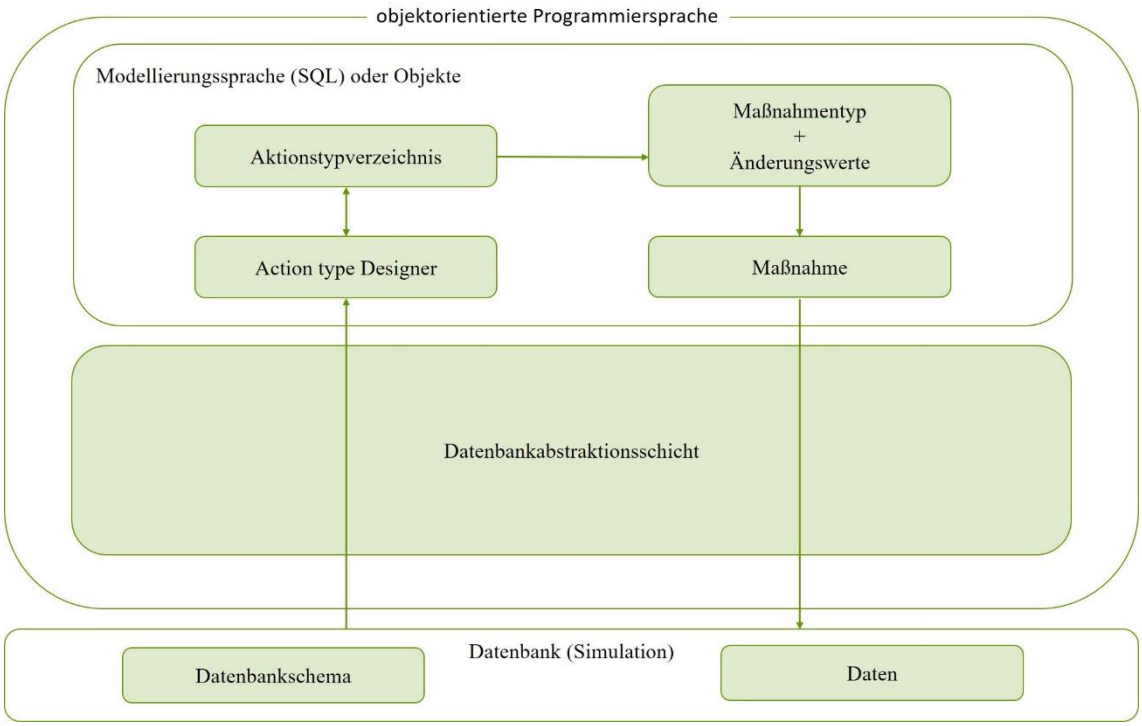


Abbildung 20: Architektur einer Applikation, die Maßnahmen an eine Datenbank weiterleitet in Anlehnung an [Sch17]

jektorientierte Programmiersprache verwendet (siehe Kapitel 3.5). Innerhalb der objektorientierten Programmiersprache wird eine Modellierungssprache (z.B. SQL) integriert, um das Modellieren von Maßnahmen zu ermöglichen. Da in einer objektorientierten Programmiersprache entwickelt wird, ist es auch möglich, Maßnahmen als Objekte darzustellen. Diese Möglichkeit der Darstellung von Maßnahmen orientiert sich an den entwickelten Frameworks SQLAlchemy und Hibernate, welche es ermöglichen, mit Objekten oder SQL zu arbeiten. Im „action type designer“ modellierte Maßnahmen werden im Aktionstypverzeichnis gespeichert. Aus dem Aktionstypverzeichnis können fertige Maßnahmentypen geladen werden. Zu den verschiedenen Maßnahmentypen lassen sich Änderungswerte, z.B. Bestand eines Artikels, eintragen. Die fertige Maßnahme, bestehend aus Maßnahmentyp und Änderungswert, wird durch die Datenbankabstraktionsschicht an die zugrundeliegende Datenbank weitergeleitet. Die Datenbank besteht aus einem Datenbankschema und Daten.

Um die genaue Funktionsweise der Weiterleitung von Maßnahmen durch die Datenbankabstraktionsschicht an die zugrundeliegende Datenbank zu bestimmen, werden zunächst die Anforderungen an die Datenbankabstraktionsschicht festgelegt.

5.1. Anforderungen an die Datenbankabstraktionsschicht

Grundsätzlich soll die Datenbankabstraktionsschicht eine Abstraktion einer Datenbank darstellen. Eine Beschreibung eines Handelsnetzwerkes durch Entitäten und Attributen soll ermöglicht

werden, unabhängig vom Aufbau des Handelsnetzwerkes und der Datenbank in der die Entitäten und Attribute gespeichert sind. Zusätzlich soll ein intuitiver Zugriff auf die Entitäten und deren Attribute ermöglicht werden. Im „action type designer“ modellierte Maßnahmen sollen gegen das unternehmensspezifische Schema modellierbar sein. Abgeleitet aus den vorherigen Kapiteln und den beschriebenen Anforderungen lässt sich die Architektur der Datenbankabstraktionsschicht für eine Veränderung einer datengetriebenen Simulation näher spezifizieren.

5.2. Architektur der Datenbankabstraktionsschicht

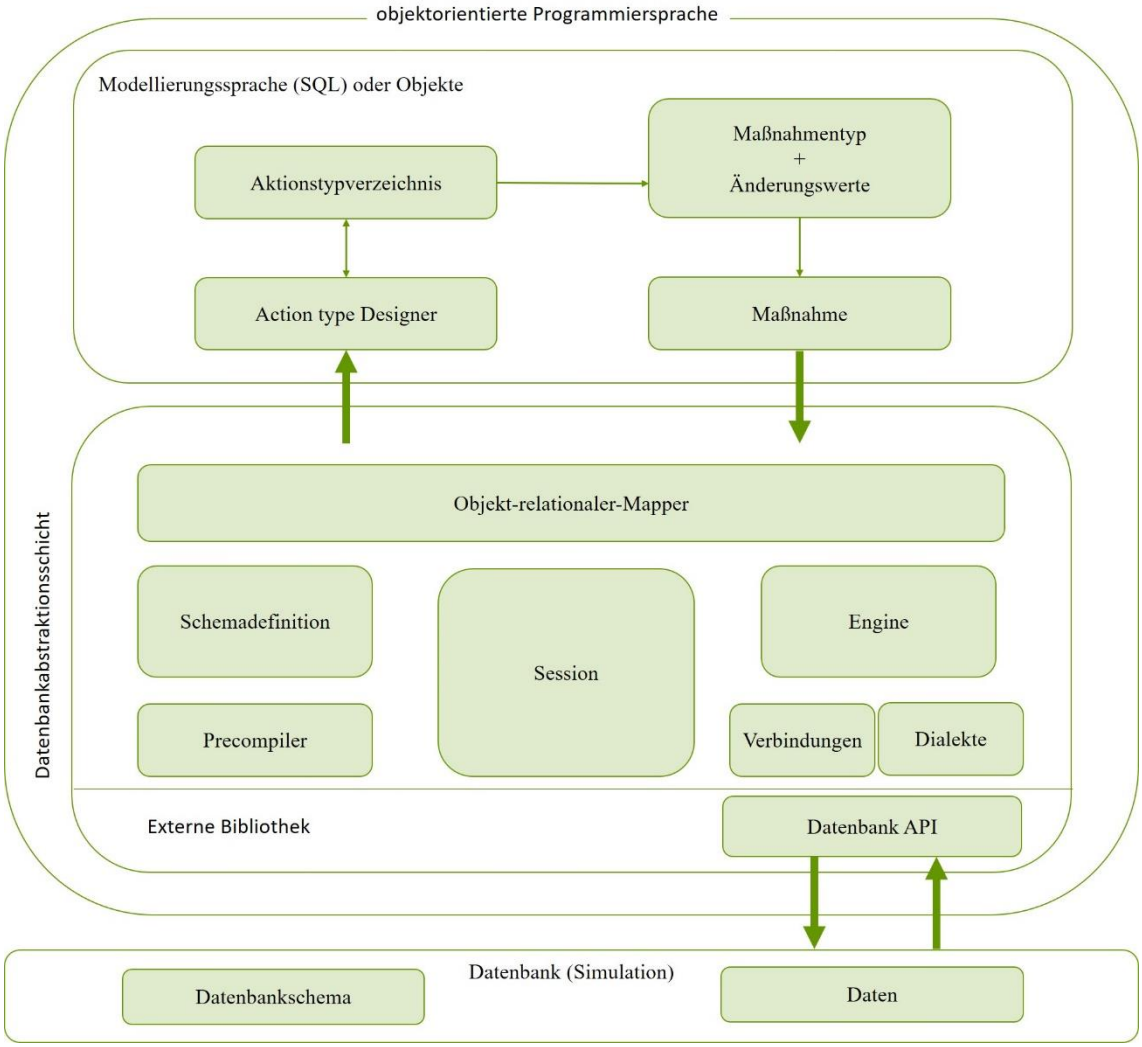


Abbildung 21: Darstellung einer konzeptionellen Architektur der Datenbankabstraktionsschicht in Anlehnung an [Tut15], [Bay05], [Sch17]

Abbildung 21 zeigt die entwickelte konzeptionelle Architektur der Datenbankabstraktionsschicht. Dabei orientiert sich die Datenbankabstraktionsschicht stark an den in 4.3 beschriebenen Frameworks von SQLAlchemy und Hibernate. Die Datenbankabstraktionsschicht nutzt eine externe Bibliothek „Datenbank API“, um eine Verbindung zu der zugrundeliegenden Datenbank herzustellen. Die Verwaltung der „Verbindungen“ und „Dialekte“ geschieht innerhalb der Abstraktionsschicht. Zusammen bilden die Dialekte und Verbindungen eine „Engine“. Durch die Verwaltung der Verbindungen und Dialekte innerhalb der Datenbankabstraktionsschicht ist eine Verwaltung einfacher, als durch mehrere externen Bibliotheken (siehe Kapitel 4.1). Durch die „Session“

wird eine „unit of Work“ repräsentiert. Sie bildet einen temporären Speicher, in dem Objekte und Methoden bearbeitet und ausgeführt werden können. Der „Precompiler“ dient zur Überführung von SQL Statements in Methoden. Die „Schemadefinition“ bildet ein programmatisches System zur Beschreibung von Tabellen und Spalten. Insgesamt bilden die Schemadefinition, der Precompiler, die Session und die Engine mit ihren Verbindungen und Dialekten einen Kern, abgeleitet aus dem Framework SQLAlchemy. Aufbauend auf dem Kern liegt der objektrelationale Mapper. Das Konzept orientiert sich an Hibernate hinsichtlich der Möglichkeit, verschiedene Arten des Mappings zur Verfügung zu stellen. Durch die verschiedenen Möglichkeiten des Mappings lassen sich die unterschiedlichen Vorteile der Mappings dem Kontext anpassen. So werden z.B. bei einer sehr tiefen Klassenhierarchie, unter Verwendung des horizontalen Mappings, zu viele joins benötigt, um Objekte laden zu können. Ein join ermöglicht das Verknüpfen von Spalten mehrerer Tabellen, die gemeinsame Werte besitzen. Bei einer sehr großen Basisklasse, die Kindsklassen mit wenig Attributen besitzt, kann das Filter-Mapping sehr effektiv sein. Hingegen wäre in diesem Fall das vertikale Mapping nachteilig, da es sehr viele kleine Tabellen erzeugen würde. Durch die vielen kleinen Tabellen würde eine längere Laufzeit bei Aufrufen entstehen. Durch die Möglichkeit der Auswahl zwischen unterschiedlichen Mappings, können die Vorteile des Mappings dem Kontext angepasst werden. ([Sch13])

Maßnahmen, die in der übergeordneten Anwendung, entweder in einer Modellierungssprache oder in Objekten definiert werden, können an die Datenbankabstraktionsschicht übergeben werden. Dort werden die Maßnahmen entweder erst in der Session bearbeitet und dann an die Datenbank weitergeleitet, oder direkt über die Engine an die Datenbank weitergeleitet.

6 Konzeptionelle Entwicklung einer Methode für Änderungen an dem Datenbankschema eines Handelsnetzwerkes

Für eine Methode zur Verwaltung und für Änderungen an einem Datenbankschema einer Datenbank lassen sich in der Literatur verschiedene Namen finden, die sich allerdings in ihrer grundsätzlichen Idee nicht unterscheiden. Namen für die Methodik sind beispielsweise „Schema Change Management“, „Datenbankmigration“, „Schemamigration“, „Management Of Schema Evolution“. Die grundlegende Idee für die Verwaltung und für Änderungen an einem Datenbankschema beinhaltet eine Versionierung des Datenbankschemas. Bei einer Versionierung des Datenbankschemas kann festgestellt werden, auf welchem Stand sich das Schema befindet oder ob Schema-Änderungen existieren, die noch eingepflegt werden müssen. Es existieren zwei Möglichkeiten eine Versionierung für ein Datenbankschema vorzunehmen. Bei der vertikalen Vorgehensweise werden Migrationsdateien erstellt, die Schema-Änderungen beschreiben. Die horizontale Vorgehensweise hält das gesamte Schema der verschiedenen Versionen vor. ([THM12]; [ALP91]; [Mih16])

In der vertikalen Versionierung eines Datenbankschemas wird pro Version des Schemas eine Migration erstellt. Die Migration beinhaltet die Änderungen an dem Schema, die für diese Version vorgesehen sind. Zusätzlich besteht eine Möglichkeit die Änderungen dieser Version wieder rückgängig zu machen. Die Änderungen können durch eine DDL wie z.B. SQL durchgeführt werden. Durch das Ausführen der DDL, die innerhalb der Migrationen gespeichert ist, lässt sich jede Version des Datenbankschemas in die Datenbank übertragen.

Bei der horizontalen Versionierung eines Datenbankschemas wird pro Version des Datenbankschemas das gesamte Schema gespeichert. Dabei wird, wie bei der vertikalen Versionierung das Schema in Form von DDL Statements repräsentiert. ([THM12])

Da nicht jedes Datenbanksystem mit der gleichen DDL arbeitet, lässt sich nicht jede Migration in jede Datenbank übertragen. Um dies zu erreichen, muss eine Datenbankabstraktionsschicht implementiert werden. Wird eine Schemaänderung der Migration über eine Abstraktionsschicht ausgeführt, lässt sich die Schemaänderung unabhängig vom Datenbanksystem durchführen.

Die vorgestellten Frameworks Hibernate und SQLAlchemy bieten eine Bibliothek für die Versionierung des Datenbankschemas an. In Hibernate heißt diese Bibliothek „HBM2DDL“. Sie ermöglicht die Übertragung der zu mappenden Klassen, die in dem ORM erstellt wurden, in einem Schema in die Datenbank. Dabei wird durch einen Vergleich des aktuellen Stands des Schemas in der Datenbank mit dem aktuellen Stand des im ORM durch Klassen definierten Schemas ein Skript erzeugt, welches die Unterschiede darstellt. Es handelt sich daher um eine vertikale Versionierung. Beim Ausführen des Skriptes, genannt Update, wird das Datenbankschema so geändert, dass es mit dem definierten Schema im ORM identisch ist. Hierzu werden die, in Kapitel 4.3 beschriebenen Methoden verwendet. Die Bibliothek HBM2DDL umfasst dabei keine Speicherung der Altversionen, so dass nur die aktuelle Version besteht. Hibernate weist darauf hin, bei größeren Projekten externe Versionierungs-Tools zu verwenden, wie z.B. FlywayDB

([MEB19]). FlywayDB funktioniert wie HBM2DDL nach der Methodik der vertikalen Versionierung, bietet aber zusätzlich eine flyway_schema_history an, welche alle Versionen des Schemas speichert ([Box19]). So bietet auch der Entwickler von SQLAlchemy eine Bibliothek namens „Alembic“ für die Migration von Datenbankschemen an. Diese Bibliothek funktioniert ebenfalls nach der Methodik der vertikalen Versionierung. Anders als bei Hibernate werden bei Alembics alle Versionen des Schemas gespeichert. Wie auch bei Hibernate bietet Alembics die Möglichkeit sich mit dem ORM zu verbinden, so dass eine automatische Version der Migration durch einen Vergleich von im ORM gespeicherten Objekten, die ein Schema bilden und dem aktuellen Datenbankschema erstellt wird. Die Änderungen des Datenbankschemas auf die gewünschte Version erfolgen durch die im Core etablierte Engine, die die verschiedenen Verbindungen und Dialekte managen kann. Da Alembic für die Datenbankmigration im Prototyp verwendet wird, wird auf die Methoden von Alembic näher eingegangen. ([Bay10])

Init: Mit dem „init“ Befehl von Alembic wird eine Migrationsumgebung geschaffen, die sich an die speziellen Anforderungen der Anwendung anpassen lässt. Die Migrationsumgebung ist ein Verzeichnis, das aus mehreren Scripten besteht. Dabei wird die Migrationsumgebung nur einmal erzeugt und anschließend mit dem Quellcode der Anwendung zusammen verwaltet. Die Migrationsumgebung wird in Abbildung 22 dargestellt.

```
yourproject/
    alembic/
        env.py
        README
        script.py.mako
        versions/
            3512b954651e_add_account.py
            2b1ae634e5cd_add_order_id.py
            3adcc9a56557_rename_username_field.py
```

Abbildung 22: Migrationsumgebung von Alembic [Bay10]

In der Migrationsumgebung, die innerhalb des Projektes erzeugt wird, liegt die env.py Datei. Diese Datei ist ein Python Skript, das bei jeder Migration ausgeführt wird. Das Skript stellt eine Verbindung mit der in SQLAlchemy definierten Engine her. Sie dient als Quelle der Datenbankkonnektivität. Das Skript kann so angepasst werden, dass mehrere Engines bearbeitet werden können, benutzerdefinierte Argumente an die Migrationsumgebung übergeben werden und anwendungsspezifische Bibliotheken und Modelle geladen und verfügbar gemacht werden können. Eine weitere Datei, die innerhalb der Migrationsumgebung liegt, ist die script.py.mako Datei. Diese Datei ist ein Mako Template, welches zur Erzeugung von Migrationen, die innerhalb des Ordners Versionen liegen, genutzt wird. Das Mako Template ist skriptfähig, so dass die Struktur jeder Migrationsdatei gesteuert werden kann, einschließlich Standardimporten sowie Änderungen in der Struktur der Funktionen „upgrade ()“ und „downgrade ()“. Die Migrationen, die sich im Ordner Versionen befinden, werden zur Identifizierung mit einem Globally Unique Identifier versehen. Anders als bei anderen Migrations-Tools, die eine aufsteigende Versionierung verfolgen, lässt sich durch einen Globally Unique Identifier nachträglich eine Version dazwischen spielen. ([Bay10])

Migration: Migrationen stellen die Versionen des Datenbankschemas dar. Eine Migrationsdatei wird mit dem Befehl „revision“ oder „migrate“ erzeugt. Die Datei enthält Header-Informationen,

Bezeichner für die aktuelle und eine Downgrade-Version, einen Import grundlegender Alembic-Direktiven sowie die leeren Upgrade- und Downgrade-Funktionen. Die Funktionen Upgrade und Downgrade müssen mit Anweisungen gefüllt werden, die eine Reihe von Änderungen an dem Datenbankschema vornehmen. Durch die Downgrade-Versions-Bezeichnung lässt sich die Reihenfolge der Migrationen festlegen. Die in der Migrationsdatei verfügbare Upgrade-Funktion, lässt sich mit SQL oder mit Klassen, die in der Abstraktionsschicht definiert sind, befüllen. Analog zur Upgrade-Funktion werden in der Downgrade-Funktion die Schemaänderungen rückgängig gestaltet. Alembic bietet die Möglichkeit durch das Scannen, der im ORM definierten Klassen, die Upgrade- sowie Downgrade-Funktion automatisiert zu befüllen. ([Bay10]) Listing 5 zeigt die Erstellung einer Migration.

Listing 5: 1975ea83b712_create_account_table.py: Erstellen einer Migration [Bay10]

```
"""create account table

Revision ID: 1975ea83b712
Revises:
Create Date: 2011-11-08 11:40:27.089406

"""

# revision identifiers, used by Alembic.
revision = '1975ea83b712'
down_revision = None
branch_labels = None

from alembic import op
import sqlalchemy as sa

def upgrade():
    pass

def downgrade():
    pass
```

6.1. Anforderungen einer Methode für Änderungen an dem Datenbankschema

Für eine Integration der oben beschriebenen Methodik für Änderungen an dem Datenbankschema müssen zunächst die Anforderungen festgelegt werden. Änderungen am Datenbankschema sollten unabhängig von dem verwendeten Datenbanksystem durchführbar sein. HBM2DDL und Alembic bieten durch Nutzung der in dem ORM erstellten Klassen eine Abstraktion der Datenbank, wodurch diese Anforderung erfüllt ist. Die Methodik der Versionierung bietet zudem die Möglichkeit der horizontalen und vertikalen Speicherung des Schemas. Alembic, HBM2DDL, sowie FlywayDB sind etablierte Schemamigrations-Tools, die eine vertikale Speicherung nutzen.

Bei der Betrachtung der Anwendung „Maßnahmen eines Handelsnetzwerkes an eine zugrundeliegende Datenbank weiterleiten“ wird eine Kombination aus horizontaler und vertikaler Speicherung präferiert. Kleinere Änderungen, wie Beziehungen oder das Ändern von Tabellennamen lassen sich durch eine vertikale Speicherung des Schemas durchführen. Große Änderungen, die einen kompletten Austausch des Schemas erfordern, können horizontal gespeichert werden.

6.2. Architektur einer Methode für Änderungen an dem Datenbankschema eines Handelsnetzwerkes

Die Architektur einer Methode für Änderungen an dem Datenbankschema eines Handelsnetzwerkes wird in Abbildung 23 dargestellt. Die Methode der Schema-Migration besteht aus einer Environment-Datei, in der die Verbindungen der Engine zur Datenbankabstraktionsschicht gespeichert werden. Durch die Verbindung der Schema-Migration mit der Engine lässt sich sowohl das definierte Schema in dem ORM als auch das vorliegende Schema in der Datenbank auslesen. Durch eine Template-Datei lassen sich Versionen in einem Versionen-Ordner generieren. Das Template besteht aus Informationen über die Version des Schemas sowie einer Upgrade- und Downgrade-Funktion. Innerhalb des Versionen Ordners lassen sich horizontale und vertikale Versionen speichern. Die Architektur orientiert sich insgesamt an bereits bestehenden Tools wie Alembic und FlywayDB.

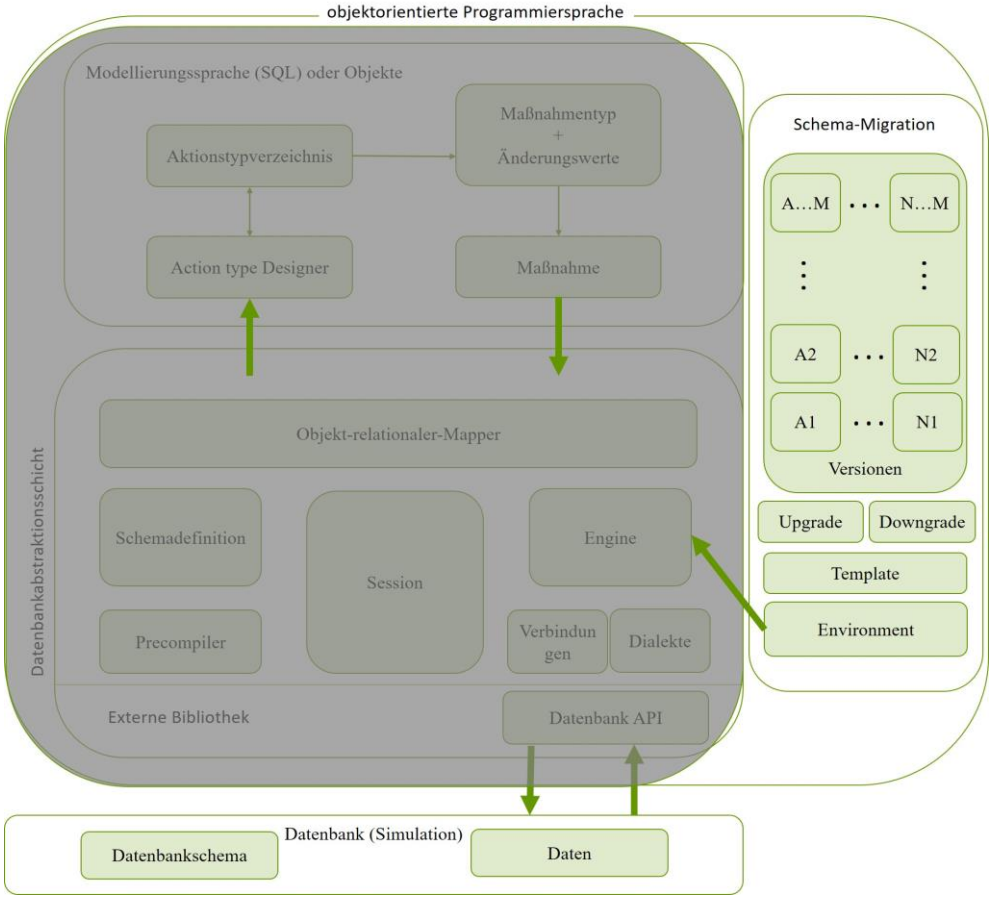


Abbildung 23: Architektur einer Schema-Migration

7 Beispielhafte Entwicklung eines Prototyps mit einem Datenbankschema für ein Handelsnetzwerk

Für die Umsetzung eines Prototyps der in Kapitel 5.2 konzeptionell entwickelten Datenbankabstraktionsschicht wird die Programmiersprache Python verwendet. Da sich das erarbeitete Konzept an dem Framework SQLAlchemy orientiert, wird dieses für den Prototyp verwendet. Für eine Veranschaulichung von Maßnahmen in einer grafischen Oberfläche wird die Datenbankabstraktionsschicht in ein Webframework integriert, das eine Darstellung über HTML ermöglicht. Für eine Änderung an dem Schema wird das auf SQLAlchemy aufbauende Versionierungs-Tool Alembic verwendet. Die in dieser Arbeit vorkommenden Code Passagen sind nicht vollständig und zeigen nur den für die Datenbankabstraktionsschicht relevanten Code. Eine Reproduzierung eines funktionalen Prototyps ist möglich durch den auf GITHUB (https://github.com/papp0/Master_ITPL) bereitgestellten Code.

7.1. Entwicklung eines Datenbankschemas für ein Handelsnetzwerk

Zunächst wird ein Datenbankschema für ein Handelsnetzwerk erstellt, an dem Maßnahmen durchgeführt werden können. Um das Datenbankschema überschaubar zu halten, besteht das im Prototyp verwendete Datenbankschema für ein Handelsnetzwerk aus Standorten genannt „Sites“ und Handelswaren genannt „SKU“. Beide Gegenstandstypen verfügen über mehrere Eigenschaften. Das in Kapitel 3.3 beschriebene ER-Modell wird angewendet, um das Datenmodell zu skizzieren (siehe Abbildung 24).

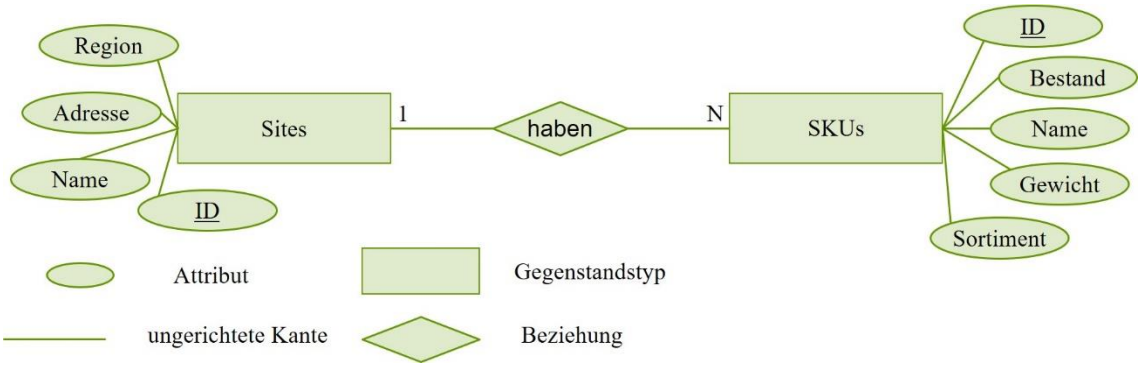


Abbildung 24: ER-Modell eines Handelsnetzwerkes nach [RAS18]

Eine Überführung des ER-Modells in ein relationales Schema ergibt zwei Relationen. Diese Relationen werden in Abbildung 25 dargestellt.

ID	Name	Adresse	Region

ID	Name	Sortiment	Gewicht	Bestand

Abbildung 25: Relationales Schema eines Handelsnetzwerkes

7.2. Etablierung einer Datenbank

Um das relationale Schema zugänglich zu machen, wird eine relationale Datenbank benötigt. Für den Prototyp wird in diesem Fall eine SQLite Datenbank verwendet. Eine SQLite Datenbank wird in einer einzigen Datei gespeichert und benötigt keinen Server ([Gri18], S.37). In Kapitel 8.2 der Evaluation wird die SQLite Datenbank ausgetauscht, um die Anforderung der Abstraktion der Datenbank unabhängig vom Typ des Datenbanksystems zu testen. Um eine Verbindung mit der Datenbank herzustellen wird die Engine verwendet, die aus den Verbindungen und dem Dialekt besteht. Die Engine wird in einer config Datei beschrieben. Die config Datei wird in Listing 6 dargestellt.

Listing 6: config.py: Engine Konfiguration

```
import os
basedir = os.path.abspath(os.path.dirname(__file__))
class Config(object):
    # ...
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or \
        'sqlite:///' + os.path.join(basedir, 'app.db')
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

Die SQLALCHEMY_DATABASE_URI bildet die Engine. Dabei wird die Verbindung zur ‚DATABASE_URL' hergestellt. Ist keine ‚DATABASE_URL' definiert, wird eine Verbindung zu einer SQLite Datenbank hergestellt, welche sich in dem Ordner befindet, in dem das Projekt gespeichert wird.

7.3. Integration des Datenbankschemas

Die Daten, die in der Datenbank gespeichert werden sollen, werden durch eine Sammlung von Klassen dargestellt. Die ORM-Schicht in SQLAlchemy führt die erforderlichen Übersetzungen aus, um aus den Objekten, die aus Klassen erstellt wurden, Zeilen in den entsprechenden Datenbanktabellen abzubilden. Die Klassen werden in einer models.py Datei beschrieben, die in Listing 7 dargestellt ist.

Listing 7: models.py: Datenbankmodell

```
from app import db
...
class Site(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), index=True, unique=True)
    region = db.Column(db.String(64))
    adresse = db.Column(db.String(64))
    skus = db.relationship('SKU', backref='standort', \
                           lazy='dynamic')
    def __repr__(self):
        return "<Site(name='%s', region='%s', adresse='%s')>" % (
            self.name, self.region, self.adresse)

class SKU(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64))
    sortiment = db.Column(db.String(64))
    gewicht = db.Column(db.Float)
    site_name = db.Column(db.Integer, db.ForeignKey('site.name'))
    bestand = db.Column(db.Integer)
    def __repr__(self):
        return "<SKU(name='%s', sortiment='%s', gewicht='%s')>" %(self.name, self.sortiment, self.gewicht)
```

Die Klasse Site bildet eine Tabelle mit dem Namen „site“ ab. Die Tabelle hat die Attribute „id“, „name“, „region“, adresse und bildet eine 1:N Beziehung mit den SKUs. Dabei werden den Attributen bestimmte Datentypen zugeordnet, die in der Datenbank realisiert werden. Die Methode „__repr__“ zeigt Python wie ein Objekt aus der Klasse dargestellt werden soll.

Die in models.py beschriebenen Klassen bilden das Datenbankschema des Handelsnetzwerkes, welches in die Datenbank übertragen werden soll. Da in der endgültigen Anwendung verschiedene Versionen eines Datenbankschemas in einer Datenbank gespeichert werden sollen, wird das Versionierungs-Tool Alembic verwendet. Durch die in dem Tool verfügbare Methode „migrate“ wird eine Version des Datenbankschemas erstellt. Die migrate Methode erkennt Änderungen an den in models.py beschriebenen Klassen und vergleicht diese mit dem aktuellen Datenbankschema in der Datenbank. Diese Änderungen werden in eine Update Funktion geschrieben. Durch die Update Funktion werden Änderungen der Unterschiede zwischen dem

beschriebenen Datenbankschema in models.py und dem Datenbankschema in der Datenbank ausgeführt. Jede erstellte Version des Datenbankschemas kann wiederhergestellt werden. Eine Version des Datenbankschemas für die oben beschriebenen Klassen wird in Listing 8 dargestellt.

Listing 8: 2a08323fbca9_sku_und_site.py: Version des Datenbankschemas

```
"""SKU und Site
Revision ID: 2a08323fbca9
Revises: e2e0c49780f2
Create Date: 2019-03-19 11:26:28.590513
"""
from alembic import op
import sqlalchemy as sa
# revision identifiers, used by Alembic.
revision = '2a08323fbca9'
down_revision = 'e2e0c49780f2'
branch_labels = None
depends_on = None
def upgrade():
    # ### commands auto generated by Alembic - please adjust! ###
    op.create_table('site',
    sa.Column('id', sa.Integer(), nullable=False),
    sa.Column('name', sa.String(length=64), nullable=True),
    sa.Column('region', sa.String(length=64), nullable=True),
    sa.Column('adresse', sa.String(length=64), nullable=True),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_index(op.f('ix_site_name'), 'site', ['name'], unique=True)
    op.create_table('SKU',
    sa.Column('id', sa.Integer(), nullable=False),
    sa.Column('name', sa.String(length=64), nullable=True),
    sa.Column('sortiment', sa.String(length=64), nullable=True),
    sa.Column('gewicht', sa.Float(), nullable=True),
    sa.Column('site_name', sa.Integer(), nullable=True),
    sa.Column('bestand', sa.Integer(), nullable=True),
    sa.ForeignKeyConstraint(['site_name'], ['site.name'], ),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_index(op.f('ix_SKU_name'), 'SKU', ['name'], unique=True)
    # ### end Alembic commands ###
def downgrade():
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_index(op.f('ix_SKU_name'), table_name='SKU')
    op.drop_table('SKU')
    op.drop_index(op.f('ix_site_name'), table_name='site')
    op.drop_table('site')
```

7.4. Integration der Maßnahmen

In dem Prototyp sind zwei Versionen von Maßnahmen integriert. Zum einen eine Version, die zunächst Maßnahmen in einem Python-Objekt speichert und diese dann in SQL umwandelt, zum anderen eine Version, die reines SQL verwendet. In der ersten Version sind die Maßnahmen zur Erstellung, Änderung, und Löschen von Standorten verfügbar. Zudem ist die Maßnahme verfügbar ein SKU einem bestimmten Standort zuzufügen. Wird der Standort gelöscht, werden auch alle zugeordneten SKUs gelöscht. Die vordefinierten Maßnahmen repräsentieren ein Aktionstypverzeichnis. Abbildung 26 und 27 zeigen die im Prototyp verwendete grafische Oberfläche für die oben beschriebenen Maßnahmen.

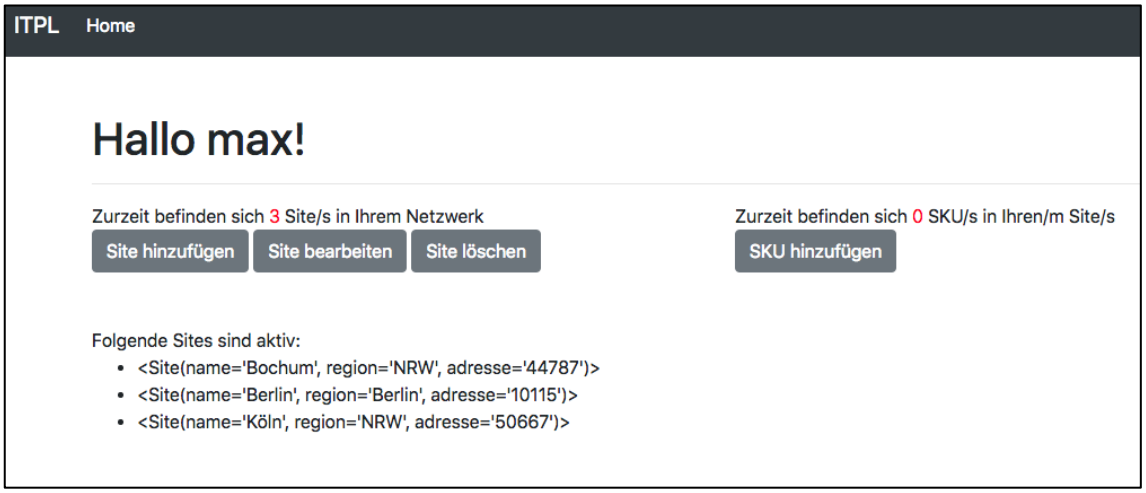


Abbildung 27: GUI 1 Übersicht des Prototyps

Abbildung 26: GUI 2 Modellierte Maßnahmen des Prototyps

Für ein besseres Verständnis der Umsetzung von Maßnahmen wird die Maßnahme „Site bearbeiten“ näher betrachtet. Die Eingabeformen „Name“, „Region“ und „Adresse“ werden durch eine Python Klasse erzeugt. Die Form des Namens wird als Dropdownmenü dargestellt und zeigt alle sich bisher im Netzwerk befindenden Sites an. Die im Netzwerk vorhandenen Sites werden per

Query, die sich in der Datenbankabstraktionsschicht befindet, ausgelesen. Die Klasse und die Query für die Form werden in Listing 9 dargestellt.

Listing 9: forms.py: Klasse für SiteupdateForm

```
...
def querySites():
    return Site.query

class SiteupdateForm(FlaskForm):
    name = QuerySelectField(query_factory=querySites)
    region = StringField('Region', description="NRW")
    adresse = StringField('Adresse', description="44135")
    submit = SubmitField('Bearbeiten')
```

Um die Site zu bearbeiten, werden die in die Form eingetragenen Daten in ein Python-Objekt überführt, welches wiederum im Nachgang als SQL an die Datenbank übertragen wird. Dieser Vorgang wird in Listing 10 dargestellt.

Listing 10: routes.py: Datenbankeintrag Site bearbeiten

```
...
def updateSite():
    form = SiteupdateForm()
    if form.validate_on_submit():
        x = form.name.data
        site = Site.query.filter_by(name=x.name).first()
        site.adresse = form.adresse.data
        site.region = form.region.data
        db.session.commit()
    return render_template('updateSite.html', form=form)
```

Die Maßnahmen zum Erstellen und Löschen von Sites und dem Erstellen von SKU funktionieren ähnlich der oben beschriebenen Maßnahme des Bearbeitens von Sites. Bei der Maßnahme „Löschen“ werden, alle einer Site zugeordneten SKUs, ebenfalls gelöscht. Diese Option lässt sich ausschalten. Sie wird im Framework SQLAlchemy „Cascade“ genannt.

In der zweiten Version von Maßnahmen werden Maßnahmen in Form von SQL direkt an die Datenbank weitergeleitet. Die grafische Oberfläche für diese Version einer Maßnahme wird in Abbildung 28 dargestellt.

Abbildung 28: GUI 3 SQL definierte Maßnahmen des Prototyps

Die Form, die in Abbildung 28 dargestellt ist, wird in forms.py beschrieben. Sie ähnelt der Klasse SiteupdateForm, welche in Listing 9 beschrieben wurde. Das in die Form eingetragene SQL Statement wird beim Bestätigen direkt an die Datenbank gesendet. Dieser Vorgang wird in Listing 11 dargestellt.

Listing 11: routes.py: Direkte SQL Statements

```
def SQL():
    form = SQLForm()
    if form.submit.data and form.validate_on_submit():
        db.engine.execute(text(form.sql.data))
    return render_template('SQL.html', form=form)
```

8 Evaluation des Konzeptes durch Test des Prototypen

Die Evaluation des Konzeptes findet durch den Test des Prototypen statt. Hierzu werden vier Tests beschrieben und durchgeführt, die die in Kapitel 5.1 und 6.1 beschriebenen Anforderungen prüfen.

8.1. Test 1: Intuitiver Zugriff auf die Datenbank

Test Nr. 1 befasst sich mit der Anforderung des intuitiven Zugriffs auf die in der Datenbank gespeicherten Entitäten und Attribute über die Datenbankabstraktionsschicht. Dieser Test lässt sich in einer Python Umgebung durchführen, die in Listing 12 dargestellt ist.

Listing 12: tests.py: Testen des Zugriffs auf die Datenbank durch die Datenbankabstraktionschicht

```
import unittest
from app import app, db
from app.models import Site, SKU
class SiteSkuCase(unittest.TestCase):
    def setUp(self):
        app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite://'
        db.create_all()
    def tearDown(self):
        db.session.remove()
        db.drop_all()
    def test_sites_skus(self):
        # Erstelle 4 Sites
        s1 = Site(name='Dortmund', region='NRW', adresse='44135')
        s2 = Site(name='Münster', region='NRW', adresse='48143')
        s3 = Site(name='Berlin', region='Berlin', adresse='10115')
        s4 = Site(name='Hamburg', region='Hamburg', adresse='20095')
        db.session.add_all([s1, s2, s3, s4])
        # Erstelle 4 SKU
        sku1 = SKU(name='nike', sortiment='schuhe', gewicht='0.5')
        sku2 = SKU(name='adidas', sortiment='schuhe', gewicht='0.5')
        sku3 = SKU(name='puma', sortiment='schuhe', gewicht='0.5')
        sku4 = SKU(name='erima', sortiment='schuhe', gewicht='0.5')
        db.session.add_all([sku1, sku2, sku3, sku4])
        db.session.commit()
        # Ordne jeweils einer Site ein SKU zu
        sku1.standort = s1
        sku2.standort = s2
```

```
            sku3.standort = s3
            sku4.standort = s4
            db.session.commit()
            # check the followed posts of each user
            f1 = Site.query.all()
            f2 = SKU.query.all()
            f3 = Site.query.get(1)
            f3.skus.all()
if __name__ == '__main__':
    unittest.main(verbosity=2)
```

Beim Durchführen des Tests wird zunächst eine im Arbeitsspeicher liegende SQlite Datenbank erzeugt, in der die vorher definierten Klassen, der Site und der SKU, gemappt werden. Nun werden vier beispielhafte Sites erzeugt, die mittels der Datenbankabstraktionsschicht an die im Arbeitsspeicher liegende SQlite Datenbank weitergeleitet werden. Anschließend werden vier SKUs erzeugt, die jeweils einer Site zugeordnet werden. Nach der Erstellung und Zuordnung von Sites und SKUs werden die Einträge in der Datenbank, die durch die Datenbankabstraktionsschicht erfolgten, mittels query überprüft. Das Ergebnis des Tests zeigt, dass ein intuitiver Zugriff auf Entitäten und Attribute über die Datenbankabstraktionsschicht erfolgen kann und somit die Anforderung erfüllt ist.

8.2. Test 2: Abstraktion des Datenbanksystems

Test Nr. 2 soll die Abstraktion testen, indem ein Austausch des Datenbanksystems erfolgt. Das zugrundeliegende Datenbanksystem soll von SQlite auf PostgresSQL geändert werden. Für diesen Tausch wird ausschließlich die Datenbank URL und der zu verwendende Dialekt benötigt. Diese werden in der Engine, die Teil der Datenbankabstraktionsschicht ist, eingetragen. Der Austausch wird in der config.py (siehe Listing 13) beschrieben. Durch das Austauschen der URL und des Dialektes verbindet sich die Datenbankabstraktionsschicht mit einem PostgresSQL Datenbanksystem und das in models.py erstellte Schema kann übertragen werden.

Listing 13: config.py: URL und Dialekt Austausch der Engine

```
class Config(object):
    # ...
    SQLALCHEMY_DATABASE_URI = 'postgresql://localhost/postgres'
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

Das Ergebnis des zweiten Tests zeigt, dass eine Abstraktion der Datenbank durch die Datenbankabstraktionsschicht ermöglicht wird. Durch den Aufbau der Engine lassen sich verschiedene URL und Dialekte von unterschiedlichen Datenbanken eintragen.

8.3. Test 3: Prüfung ob Maßnahmen gegen ein spezifisches Datenmodell modellierbar sind

Test Nr. 3 prüft, ob modellierte Maßnahmen gegen das unternehmensspezifische Datenmodell modellierbar sind. Dieser Test findet wie Test Nr. 1 in einer Python Umgebung statt. Dabei wird eine in SQL modellierte Maßnahme über die Datenbankabstraktionsschicht an die Datenbank übermittelt. Der Test ist in tests.py dargestellt (siehe Listing 14). Der Test ist bestanden, wenn die auszuführende Maßnahme im unternehmensspezifischen Datenmodel ausgeführt wird. Die Ergebnisse des Tests zeigen, dass die Anforderungen an die Datenbankabstraktionsschicht erfüllt sind.

Listing 14: tests.py: Testen von modellierten Maßnahmen

```
...
class SiteSkuCase(unittest.TestCase):
    ...
def test_core(self):
    db.engine.execute(
text("INSERT INTO site (name, region, adresse) VALUES ('Dortmund', 'NRW', '44135');"))
```

8.4. Test 4: Änderungen des Schemas

Test Nr. 4 beinhaltet Änderungen an dem verwendeten Schema für ein Handelsnetzwerk. Da Änderungen am Schema über ein Versionierungs-Tool erfolgen, werden bei Test Nr. 4 zwei verschiedene Versionen eines Schemas erstellt und dann in die Datenbank übertragen. Der Test verläuft bei der Durchführung positiv, so dass die unterschiedlichen Versionen des Schemas in die Datenbank übertragen werden. Bei der Durchführung traten Aspekte auf, die weiter betrachtet werden sollten. Zum einen werden bei der Verwendung des „Auto Generating“ nicht alle Änderungen an dem Schema erkannt, so dass die Erstellung einer Version manuell nachbearbeitet werden muss. Zum anderen wird durch das Versionierungs-Tool nur das Schema in der Datenbank geändert. In einer fertigen Applikation, die Maßnahmen an die zugrundeliegende Datenbank übermittelt, sollten die Versionen des Schemas der Datenbank und dem in der Datenbankabstraktionsschicht definierten Schema übereinstimmen. Um dies zu erreichen, könnte ein übergeordnetes Versionierungs-Tool, wie z.B. Git verwendet werden.

Insgesamt wurden alle vier Tests positiv abgeschlossen, so dass die Anforderungen an die Datenbankabstraktionsschicht erfüllt sind.

9 Fazit

Bisher wurden die Schemata von Handelsnetzwerken an eine datengetriebene Simulation gebunden. Durch die, in dieser Arbeit entwickelte Datenbankabstraktionsschicht soll die zugrundeliegende Datenbank, die von der Simulationssoftware genutzt wird, abstrahiert werden. Durch die Abstraktion der Datenbank soll die Kompatibilität zwischen unterschiedlichen Datenbankmodellen gewährleistet werden, so dass Schemaänderungen impliziert durch Maßnahmen angewendet werden können.

In Kapitel 2 wurde das Handelsnetzwerk und seine Simulation untersucht. Die Definition eines Handelsnetzwerkes wurde über die Ableitung aus dem Logistiknetzwerk, dem Handelsnetz und der Supply Chain bestimmt. Zusätzlich wurden Maßnahmen innerhalb eines Handelsnetzwerkes differenziert. Bei der Betrachtung der Simulation im Fokus auf Handels- und Logistiknetzwerke wurde herausgestellt, dass sich im Bereich der Logistik zustandskontinuierliche, zeitdiskrete Modelle abbilden lassen. Bei der datengetriebenen Simulation lassen sich die diskreten Modelle mittels eines Schemas innerhalb einer relationalen Datenbank darstellen. Änderungen der Daten finden in der datengetriebenen Simulation statt, im Simulationsmodell hingegen nicht. Bei einer ständigen Änderung der Daten kann eine quasi parallele Simulation erfolgen.

Für ein besseres Verständnis der relationalen Datenbanken, die in der diskreten datengetriebenen Simulation für Handels- und Logistiknetzwerke verwendet werden, wurde in Kapitel 3 auf Datenbanksysteme eingegangen. Den Fokus bildet dabei das relationale Schema. Um Schemaänderungen über grafisch dargestellte Maßnahmen durchzuführen, wurde eine Integration von SQL in eine objektorientierte Programmiersprache betrachtet. Bei der Integration von SQL in eine objektorientierte Programmiersprache tritt das Problem des „objectrelational impedance mismatch“ auf. Die Problematik des „objectrelational impedance mismatch“ entsteht durch die unterschiedlich genutzten Paradigmen des Relationalen und des Objektorientierten. Mögliche Lösungen für die Problemstellung sind ein Austausch der relationalen Datenbank in eine objektorientierte Datenbank oder das Vorgehen einer objektrelationalen Abbildung. Bedingt durch die diskrete Simulation und die Abbildung eines Modells im relationalen Schema wurde der Lösungsansatz der Objektrelationalen Abbildung differenziert betrachtet. Unter Mapping wird die Überführung von objektorientierten Elementen in relationale Elemente und umgekehrt verstanden. Nach der Überführung der Elemente müssen die Anfragen an die Datenbank weitergeleitet werden. Die Anfragen werden mittels eines Anfrageobjekts einer objektorientierten Application Programming Interface isoliert. Später wird das Anfrageobjekt in reines SQL umgewandelt.

In Kapitel 4 wird der Hintergrund von API’s erläutert und speziell auf die Datenbankabstraktionsschicht eingegangen. Die Datenbankabstraktionsschicht bildet eine Schnittstelle, welche die unterschiedlichen Datenbankmerkmale weitgehend unberücksichtigt lässt. Dies bedeutet, dass die Datenbankabstraktionsschicht nicht an ein bestimmtes Datenbanksystem gekoppelt ist und somit Kompatibilität durch Abstraktion erzeugt. Zur Erstellung einer Datenbankabstraktionsschicht existieren verschiedene Frameworks. Zwei der meist verwendeten Frameworks wurden in Kapitel 4 vorgestellt und miteinander verglichen. Der Vergleich der beiden Frameworks Hibernate und SQLAlchemy zeigte, dass sich die beiden Frameworks in ihrer Architektur und

Umsetzung sehr ähneln. Beide Architekturen wurden nach dem, in Martin Fowlers Buch „Patterns of Enterprise Application Architecture“ beschrieben Kapitel „Mapping to Relational Databases“, konzipiert.

In Kapitel 5 wurden die Anforderungen an die Datenbankabstraktionsschicht definiert, welche die Maßnahmen eines Handelsnetzwerkes an eine zugrundeliegende Datenbank übermitteln soll. Mit diesen Anforderungen wurde ein Konzept der Architektur einer Datenbankabstraktionsschicht entwickelt. Dieses Konzept orientiert sich an den beiden Frameworks Hibernate und SQLAlchemy.

Eine Methode zur Verwaltung und Änderungen an einem Datenbankschema eines Handelsnetzwerkes wurde in Kapitel 6 untersucht. Hierzu wurde ein Konzept entwickelt, das sich in das vorherige Konzept der Datenbankabstraktionsschicht integrieren lässt. Dieses Konzept bietet sowohl eine horizontale als auch eine vertikale Versionierung des Schemas.

Anhand der entwickelten Konzepte konnte ein Prototyp programmiert werden, der grafisch dargestellte Maßnahmen über die Datenbankabstraktionsschicht an eine zugrundeliegende relationale Datenbank übermittelt. Für die Umsetzung des Prototypen wurden verschiedene Frameworks verwendet.

Für die Evaluation der entwickelten Konzepte wurde der Prototyp in Kapitel 8 getestet. Die Ergebnisse der Tests ergaben, dass die Anforderungen an die Konzepte überwiegend erfüllt wurden. Nur bei der Versionierung des Schemas besteht ein Nachteil in der manuellen Nacharbeit bei der automatischen Erstellung einer Version. Die Versionen des Schemas der Datenbank und des Schemas der Datenbankabstraktionsschicht sollten übereinstimmen. Dies kann durch ein übergeordnetes Versionierungs-Tool erfolgen.

Innerhalb dieser Masterthesis wurde eruiert, dass bereits Frameworks zur Erstellung einer Datenbankabstraktionsschicht existieren. Das Konzept der bestehenden Frameworks zur Bildung einer Datenbankabstraktionsschicht lässt sich unabhängig von der endgültigen Anwendung integrieren. Z.B. ist es indifferent, ob ein Datenbankschema für ein Handelsnetzwerk oder für eine Schule verwendet wird.

Der Austausch der relationalen Datenbank in eine objektorientierte Datenbank, als Lösung des „objectrelational impedance mismatch“, stellt einen potenziellen Ausgangspunkt für eine weitergehende zukünftige Forschungsarbeit dar. Hier könnte untersucht werden, in wie weit eine kontinuierliche Simulation in der Industrie 4.0 sinnvoll ist. Durch das Internet der Dinge lassen sich unterschiedliche Daten generieren. Wird das mooresche Gesetz mit einbezogen, lassen sich diese Daten mit kontinuierlich leistungsstärkeren Computern analysieren. Die unterschiedlichen Daten (Audio, Bild, Text, Video) könnten durch ihre Datenmodellierung nicht mehr in ein relationales Schema gebracht werden, wodurch ein objektorientiertes Schema Anwendung finden könnte. Die Anwendung eines objektorientierten Schemas anstelle eines relationalen Schemas würde die Anpassung der Simulationswerkzeuge implizieren.

Literaturverzeichnis

[AC15] Tim Ambler, Nicholas Cloud: *JavaScript Frameworks for Modern Web Dev.* Apress, 2015

[AF09] Arnold, Dieter; Furmans, Kai: *Materialfluss in Logistiksystemen*. Springer-Verlag 2009.

[ALP91] Andany, José; Léonard, Michel; Palisser, Carole: Management of Schema Evolution in Databases. University of Geneva, 1991.

[Amb97] Ambler, Scott W.: *Mapping Objekcts to relational Databases*. Ambysoft Inc. http://jeffsutherland.com/objwld98/mappingobjects.pdf Stand Januar 2019

[Amb03] Ambler, Scott W.: *Mapping Objects to Relational Databases: O/R Mapping In Detail*. http://agiledata.org/essays/mappingObjects.html Stand Januar 2019

[Bau01] Baumgarten, Helmut: *Logistik im E-Zeitalter: die Welt der globalen Logistiknetzwerke*. Frankfurt am Main in Frankfurter Allg. Buch, 2001.

[Bay05] Bayer, Michael: *SQLAlchemy.* The Architecture of Open Source Applications. http://aosabook.org/en/sqlalchemy.html Stand März 2019.

[Bay05a] Bayer, Michael: *SQLAlchemy Documentation.* SQLAlchemy, 2005. https://docs.sqlalchemy.org/en/latest Stand März 2019

[Bay10] Bayer, Michael: *Alembic´s documentation*. Bayer, Michael, 2010. https://alembic.sqlalchemy.org/en/latest/index.html Stand März 2019

[Bay17] Baydar Erkut: *Entwicklung einer Methodik zur grafischen Darstellung von Maßnahmen in Werkstoffhandelsnetzwerken*. Technische Universität Dortmund, 2017.

[BC09] Peter Buchholz, Uwe Clausen: *Große Netze der Logistik: Die Ergebnisse des Sonderforschungsbereichs 559*. Springer-Verlag Berlin Heidelberg, 2009. ISBN – 978-3-540-71048-6

[BD10] Bruns, Ralf; Dunkel, Jürgen: *Event-Driven Architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse*. Springer-Verlag, 2010.

[Bec08] Becker, Thorsten: *Prozesse in Produktion und Supply Chain optimieren*. Springer-Verlag Berlin Heidelberg, 2008

[BGW10] Brauer, Kati; Groß, Wendelin; Wolff, Stefan: *Flexibilität und Nachhaltigkeit – Neue Herausforderungen im Supply Chain Design*. In Corinna Engelhardt-Nowitzki, Olaf Nowitzki, Helmut Zsifkovits: Supply Chain Network Management: Gestaltungskonzepte und Stand der praktischen Anwendung. Springer-Verlag, 2010.

[BHS92] Biethahn, J.; Hummeltenberg, W.; Schmidt, B.: *Simulation als betriebliche Entscheidungshilfe*. Springer-Verlag Berlin Heidelberg, 1992

[BK08] Becker, J.; Knackstedt, R. (Hrsg.): *Wertschöpfungsnetzwerke: Konzepte für das Netzwerkmanagement und Potenziale aktueller Informationstechnologien.* Heidelberg: Physica-Verl, 2008

[Bow15 Bowers, David: *Exposing the Myth: Object-Relational Impedance Mismatch is a Wicked Problem.* The Open University United Kingdom 2015. http://oro.open.ac.uk/43318/1/download.php_articlei%3Ddbkda_2015_2_10_50020 Stand Januar 2019.

[Box19] Boxfuse GmbH: *How Flyway works*. Boxfuse GmbH, 2019. https://flywaydb.org/getstarted/how Stand März 2019

[BZ07] Brandimarte,Paolo; Zotteri, Giulio: *Introduction to Distribution Logistics.* John Wiley & Sons, 2007.

[Cha14] Chao, Lee*: Cloud Database Development and Management*. Taylor and Francis Group, 2014.

[Chr98] Christopher, Martin: *Logistics and Supply Chain Management. Strategies for Reducing Cost and Improving Service.* 2nd Ed., London, 1998.

[Chr16] Christensson, Per. *"API Definition."* TechTerms. (June 20, 2016). Stand Februar 2019. https://techterms.com/definition/api.

[CM14] Coronel, Carlos; Morris, Steven: *Database Systems: Design, Implementation, and Management.* Cengage Learning, 2015. ISBN – 978-1-305-62748-2

[Dbe19] DB- Engines Ranking: *https://db-engines.com/de/ranking* Stand Januar 2019.

[DD98] Date, Chris J.; Darwen Hugh: *SQL - der Standard: SQL/92 mit den Erweiterungen CLI und PSM.* Pearson Deutschland GmbH, 1998.

[Dob05] Giorgi Doborjginidze: *Analyse der Entwicklung intermodaler Logistik-Netzwerke in mittel- und osteuropäischen Ländern*. Kölner Wissentschaftsverlag, 2005

[Dud19] Duden: *Handelsnetz*. https://www.duden.de/node/706200/revisions/1726907/view Stand 06.03.2019

[Edu19] Educba: *Differences Between Java vs Python.* Educba, 2019. https://www.educba.com/java-vs-python/ Stand März 2019.

[Ele12] Eley,Michael: *Simulation in der Logistik: Einführung in der Erstellung ereignisdiskreter Modelle unter Verwendung des Werkzeuges „Plant Simulation*. Berlin, Heidelberg: Springer Berlin Heidelber, 2012.

[EN09] Ramez, Elmasri; Sham, Navathe: *Grundlagen von Datenbanksystemen.* Pearson Deutschland GmbH, 2009.

[Fij11] Fijal: *PyPy faster than C on a carefully crafted example.* PyPy Status Blog, 2011. https://morepypy.blogspot.com/2011/02/pypy-faster-than-c-on-carefully-crafted.html Stand März 2019

[Fow03] Fowler, Martin: *Patterns für Enterprise-Application-Architekturen.* mitp Verlags GmbH & Co. KG, 2003.

[Fra19] Frauenhofer-Institut für Materialfluss und Logistik IML: *Logistische Assistenzsysteme*. https://www.iml.fraunhofer.de/de/abteilungen/b1/informationslogistik_und_assistenzsysteme/produkte/assistenzsysteme.html Stand März 2019

[Fri10] Firedmann, Thomas L.*: Was zu tun ist: eine Agenda für das 21. Jahrhundert*. Suhrkamp, 2010.

[Gad17] Gadatsch, Andreas: *Datenmodellierung für Einsteiger: Einführung in die Entity-Relationship-Modellierung und das Relationenmodell.* Springer Fachmedien Wiesbaden GmbH, 2017. ISBN – 978-3-658-19068-2

[Gud10] Gudehus, T.: *Logistik, 4. Auflage*. Berlin Heidelberg: Springer-Verlag, 2010

[Gud12] Gudehus, T.: *Netzwerke, Systeme und Lieferketten (Logistik)* (Studienausg. der 4., aktualisierten Aufl.). Berlin: Springer Vieweg, 2012.

[Gri18] Grinberg, Miguel*: The New and Improved Flask Mega- Tutorial*. Grinberg, Miguel, 2018.

[GRSW17] Gutenschwager, Kai; Rabe, Markus; Spieckermann, Sven; Wenzel, Sigrid: *Simulation in Produktion und Logistik: Grundlagen und Anwendung*. Springer-Verlag GmbH Deutschland 2017

[HGH11] Knut,Hildebrand; Marcus, Gebauer; Holger, Hinrichs; Michael Mielke: *Daten- und Informationsqualität: Auf dem Weg zur Information Excellence.* Vieweg + Teubner Verlag | Springer Fachmedien Wiesbaden, 2011. ISBN – 978-3-8348-1453-1.

[HHU11] Heiserich, Otto-Ernst; Helbig, Klaus; Ullmann, Werner: Logistik: Eine praxisorientierte Einführung.Gabler Verlag | Springer Fachmedien Wiesbaden GmbH, Wiesbaden, 2011

[HS18] Henning, Alexander; Schneider, Willy: *Handelswaren.* https://wirtschaftslexikon.gabler.de/definition/handelswaren-33567/version-257089 Stand März 2019

[IBNW09] Christopher Ireland, David Bowers, Michael Newton, Kevin Waugh*: A Classification of Object-Relational Impedance Mismatch*. The Open University, Milton Keynes, UK, 2009.

[KE15] Kemper, Alfons; Eikler, André: *Datenbanksysteme: Eine Einführung*. Walter de Gruyter GmbH, Berlin/Boston, 2015. ISBN – 978-3-11-044375-2

[KKK12] Klaus, Peter; Krieger, Winfried; Krupp, Michael: *Gabler Lexikon Logistik:Management logistischer Netzwerke und Flüsse*. Springer Verlag, 2012.

[Kle17] Kleppmann, Martin: *Designing Data-Intensive Applications*. O´Reilly Media, 2017.

[Lac18] Lackes, Richard: Simulation. https://wirtschaftslexikon.gabler.de/definition/simulation- 43833/version-267158 Stand März 2019.

[Lar05] Larmann, Craig: *UML 2 und Patterns angewendet - objektorientierte Softwareentwicklung*. Mitp Verlags GmbH & Co. KG, 2005.

[Lem17] Lemburg, Marc-André: *PEP 249 -- Python Database API Specification v2.0*. Python, 2017. https://www.python.org/dev/peps/pep-0249/ Abruf März 2019.

[Log13] Logistik Know How: *Logistiknetzwerke: Übersicht*. https://logistikknowhow.com/materialfluss-und-transport/logistiknetzwerk-uebersicht/) Stand März 2019.

[Log16] Logistik Know How: *Stock Keeping Unit (SKU)*. https://logistikknowhow.com/aktuelle-themen/stock-keeping-unit/ Stand März 2019.

[LS18] Lackes, Richard; Siepermann, Markus: *Simulationssprache*. https://wirtschaftslexikon.gabler.de/definition/simulationssprache-43720/version-267046 Stand März 2019

[Mei13] Meier, Andreas: *Relationale Datenbanken: Leitfaden für die Praxis*. Springer Verlag, 2013.

[Nag11] Nagel, Arnfried: *Logistik im Kontext der Nachhaltigkeit: ökologische Nachhaltigkeit als Zielgröße bei der Gestaltung logistischer Netzwerke.* Univerlagtuberlin, 2011.

[Ném10] Németh, Zsuzsa: *Internationale Sortimentsgestaltung und -steuerung auf Basis von Category Management*. Peter Lang GmbH, Internationaler Verlag der Wissenschaften, 2010.

[New06] Neward, Ted: *The Vietnam of Computerscience*. https://web.archive.org/web/20180122225729/http://blogs.tedneward.com/post/the- vietnam-of-computer-science/ Stand Januar 2019.

[MEB19] Mihalcea, Vlad; Ebersole, Steve; Boriero, Andrea; et al.: *Hibernate ORM 5.3.9.Final User Guide*. Hibernate, 2019. docs.jboss.org/hibernate/orm/current/userguide/html_single/Hibernate_User_Guide.html Stand März 2019

[Mih16] Mihalcea, Vlad*: Schema Migration with Hibernate and FlywayDB*. Sitepoint, 2016. https://www.sitepoint.com/schema-migration-hibernate-flywaydb/ Stand März 2019.

[Moh18] Mohilo, Dominik: *Top 10 der Programmiersprachen: Pythons Siegeszug in den aktuellen Rankings*. Jaxenter, 2018. https://jaxenter.de/top-10-redmonk-pypl-tiobe-vergleich-python-73850 Stand März 2019.

[PSF19] Python Software Foundation: *Python Package Index*. Python Software Foundation, 2019. https://pypi.org/ Stand März 2019.

[RAS18] Rabe, Markus; Ammouriova, Majsa; Schmitt, Dominik: *Improving the performance of logistics assistance system for materials trading networks by grouping similar actions.* Proceedings of the 2018 Winter Simulation Conference.

[RCB14] Rushton, Alan; Croucher, Phil; Baker, Peter: *The Handbook of Logistics and Distribution Management:Understanding the Supply Chain.* Kogan Page Publishers, 2014.

[RD15] Rabe, Markus; Dross, F.: *A Reinforcement Learning Approach for a Decision Support System for Logitics Networks*. In: Yilmaz, L. et.al.: Winter Simulation Conference. Piscataway, NJ, USA: IEEE, 2015.

[Red11] Reddy, Martin: *API Design for C++*. Elsevier, 2011.

[Rih08] Riha, I.: *Entwicklung einer Methode für Cost Benefit Sharing in Logistiknetzwerken.* Dissertation Technische Universität Dortmund, 2008

[Rit02] Ritchie, Collin: Relational Database Principles. Cengage Learning EMEA, 2002.

[Rob17] Robinson, David: *The Incredible Growth of Python*. Stackoverflow Blog, 2017. https://stackoverflow.blog/2017/09/06/incredible-growth-python/ Stand: März 2019.

[RS07] Richard, Hans Albert; Sander, Manuela: *Technische Mechanik. Dynamik: Grundlagen - effektiv und anwendungsnah*. Springer-Verlag, 2007

[Sch00] Schröder, Frank: *Ein datenbankbasiertes Anwendersystem zur Berechnung von Gittermast-Fahrzeugkranen.* Herbert Utz Verlag, 2000.

[Sch13] Scheit, Phillip: Analyse und Lösungen für den Object-relational Impedance-Mismatch. Grin Verlag, 2013. ISBN- 9783656467816

[Sch15] Schneck (Hrsg.), Lexikon der Betriebswirtschaft, 9. Auflage, München 2015 www.finanzen.net/wirtschaftslexikon/Bestand/9

[Sch17] Schmitt, Dominik: *Methode zur Beschreibung und Umsetzung von Veränderungen in SM für Handelsnetzwerke*. Thyssenkrupp Materials Services, 2017.

[Sta05] L. Staud, Josef: *Datenmodellierung und Datenbankentwurf: Ein Vergleich aktueller Methoden*. Springer Berlin Heidelberg, 2005. ISBN – 3540205772

[Sta09] stackoverflow: *What is inlining?* Stackoverflow, 2019. https://stackoverflow.com/questions/1546694/what-is-inlining Stand März 2019.

[Sta19] stackshare: *Hibernate vs. SQLAlchemy*. Stackshare, 2019. https://stackshare.io/stackups/hibernate-vs-sqlalchemy Stand März 2019.

[Suc08] Sucky, E.: Netzwerkmanagement. In: Arnold, D., Kuhn, A., Furmans, K. et.al.: *Handbuch Logistik*. Berlin Heidelberg: Springer-Verlag, 2008.

[SW12] Skulschus, Marco; Wiederstein, Marcus: Oracle PL, SQL: Objekte und objektrelationale Techniken. Comelio Medien, 2012

[Tec13] TechTerms: *Framework.* TechTerm, 2013. https://techterms.com/definition/framework Stand März 2019

[Tha01] Thaler, Klaus: *Supply-Chain-Management: Prozessoptimierung in der logistischen Kette*. Köln: Fortis-Verl. FH, 1999. ISBN – 9783933430533.

[THM12] Technische Hochschule Mittelhessen: *Datenbankmigration*. Technische Hochschule Mittelhessen, 2012. https://wiki.thm.de/Datenbankmigration Stand März 2019.

[Tof15] Tofunmi O., Paul: *A Brief Explanation of Database abstraction Layer: Writing database queries for the future.* A Medium Corporation, 2015. https://medium.com/@paultofunmi/a-brief-explanation-of-database-abstraction-layer-writing-database-queries-for-the-future-7a1c84c9a45d Stand: März 2019.

[Tot00] Totok, Andreas: *Modellierung von OLAP- und Data-Warehouse-Systemen*. Dr. Andreas Totok, 2000

[Tut15] Tutortialspoint: Hibernate java persistence framework. Tutorials Point, 2015. https://www.tutorialspoint.com/hibernate/ Stand März 2019.

[Van09] VanDyk, John K.: Das Drupal Entwicklerhandbuch. Addison-Wesley Verlag, 2009. ISBN – 9783-8273-2798-7

[VDI13] VDI-Gesellschaft Produktion und Logistik, Fachbereich Fabrikplanung und -betrieb: *VDI- Richtlinie 3633 Blatt 1: Simulation von Logistik-, Materialfluss- und Produktionssystemen- Begriffe*. Berlin: Beuth Verlag, 2013.

[Vie12] Viehweg, Iris: *Einführung in die Wirtschaftsinformatik*. Wiesbaden: Springer Gabler, 2012.

[VK12] Vahrenkamp, Richard; Kotzab, Herbert: *Logistik: Management und Strategien: Management und Strategien*. Oldenbourg Wissenschaftsverlag; Auflage: 7, 2012.

[VK17] Vahrenkamp, Richard; Kotzab, Herbert: *Logistikwissen kompakt.* Walter de Gruyter GmbH & Co KG, 2017.

[W3S19] W3school: *SQL CREATE TABLE Statement*. https://www.w3schools.com/sql/sql_create_table.asp Stand Januar 2019

[WC96] Widom, Jennifer; Ceri, Stefano: *Active Database Systems: Triggers and Rules f or Advanced Database Processing*. Morgan Kaufmann Publishers, 1996.

[Wie09] Wieken, John-Harry: *SQL: Einstieg für Anspruchsvolle.* Pearson Deutschland GmbH, 2009

[Wie15] Wiese, Lena: *Advanced Data Management For SQL, NoSQL, Cloud and Distributed Databases*. De Gruyter, 2015. ISBN - 978-3-11-044141-3

[Wik19] Wikipedia: *Database abstraction layer.* https://en.wikipedia.org/wiki/Database_abstraction_layer Stand Februar 2019.

[Wik19a] Wikipedia: *Programmbibliothek.* https://de.wikipedia.org/wiki/Programmbibliothek Stand März 2019.

[Wik19b] Wikipedia: *Comparison of object-relational mapping software.* https://en.wikipedia.org/wiki/Comparison_of_object-relational_mapping_software Stand März 2019.

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
ER-Modell	Entity-Relationship-Modell
DBMS	Datenbankmanagementsystem
DDL	Data Definition Language
DML	Data Manipulation Language
GUI	Graphical User Interface
KPI	key performance indcator
NF	Normalform
ORM	Objektrelationaler Mapper
SKU	Stock Keeping Unit
SQL	Structured Query Language
UML	Unified Modeling Language
VDI	Verein Deutscher Ingenieure

Eidesstattliche Versicherung
(Affidavit)

Flockenhaus, Maximilian — 143828

Name, Vorname (Last name, first name) — Matrikelnr. (Enrollment number)

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem folgenden Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present Bachelor's/Master's* thesis with the following title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution.

Titel der Bachelor-/Masterarbeit*:
(Title of the Bachelor's/ Master's* thesis):

Konzeptionelle Entwicklung einer Datenbankabstraktionsschicht für Veränderungen am Datenmodell einer datengetriebenen Simulation

*Nichtzutreffendes bitte streichen
(Please choose the appropriate)

Ort, Datum (Place, date) — Unterschrift (Signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to €50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, section 63, subsection 5 of the North Rhine-Westphalia Higher Education Act (*Hochschulgesetz*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:**

Ort, Datum (Place, date) — Unterschrift (Signature)

**Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.