



Hochschule Karlsruhe  
Technik und Wirtschaft  
UNIVERSITY OF APPLIED SCIENCES

# Entwicklung eines Systems zur Bewertung von Beziehungen zwischen Personen aus einer CRM-Lösung

Bachelor Thesis  
von

**Benjamin Tenke**

An der Fakultät für Wirtschaftsinformatik  
Matrikel-Nr: 33227

Erstgutachter:

Prof. Dr. Thomas Morgenstern

Zweitgutachter:

Prof. Dr. Andreas Schmidt

Betreuernder Mitarbeiter:

Michal Dvorak

Zweiter betreuender Mitarbeiter:

Ludwig Neer

Entwurf vom: 28. Dezember 2013

---

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

**Karlsruhe, DATE**

.....  
**(Benjamin Tenke)**

# Zusammenfassung

Das Kundenbeziehungsmanagement stellt heutzutage eine enorme Relevanz für Unternehmen dar. Der stetige Wettbewerb in dem sich Unternehmen befinden, zwingt sie verstärkt auf kundenorientierte Strategien zu setzen. Die CAS Software AG bietet mit CAS genesisWorld ein Produkt zur systematischen Gestaltung der Kundenbeziehungsprozesse an. Der Datenbestand der CAS Software AG reicht von Adressen mit Kontaktmöglichkeiten, über Angebote mit Bewertung der Realisierungschancen, bis hin zu kompletten Kundenhistorien. Mitarbeiter erhalten durch die strukturierte Ablage von Informationen ein System mit dem sie im täglichen Kundendialog unterstützt werden. Mithilfe von CAS genesisWorld ist es möglich Mitarbeiter mit analytisch gewonnenen Informationen zu versorgen. Allerdings ist dies sehr langsam und komplex, weil enorme Mengen an Daten zur Beantwortung der Abfrage zusammengeführt werden.

Um diese Probleme zu umgehen wird in der vorliegenden Arbeit ein eigenständiges System entwickelt. Es soll eine performante Bewertung von Beziehungen zwischen Personen aus einem CRM-System ermöglichen. Hierbei werden Modelle und Prozesse zur Umsetzung eines solchen Vorhabens vorgestellt. Überdies wird das bestehende CRM-System untersucht, Anforderungen an das neue System erhoben und relevante Daten identifiziert. Aufbauend auf den gewonnenen Informationen werden verschiedene Datenbanken auf ihre Verwendbarkeit evaluiert. Des Weiteren werden Konzepte erarbeitet, wie die Daten übernommen, abgelegt und wieder abgerufen werden können. Zum Schluss werden die Ergebnisse anhand fachlicher und technischer Anforderungen bewertet.



# Inhaltsverzeichnis

<b>1 Einführung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Zielsetzung . . . . .	2
1.3 Gliederung der Arbeit . . . . .	2
<b>2 Grundlagen</b>	<b>5</b>
2.1 NoSQL - Eine Einführung . . . . .	5
2.1.1 Document Stores . . . . .	5
2.1.2 Extensible Record Store . . . . .	6
2.1.3 Key-Value-Store . . . . .	6
2.1.4 Graphdatenbank . . . . .	6
2.1.5 Theoretische Grundlagen . . . . .	7
2.2 In-Memory-Datenbanken . . . . .	8
2.3 Component Object Model . . . . .	9
2.3.1 Architektur . . . . .	10
2.3.2 COM-Client . . . . .	10
2.3.3 COM-Server . . . . .	11
2.3.4 COM-Schnittstelle . . . . .	11
2.3.5 COM-Objekte . . . . .	11
2.3.6 Interface Definition Language . . . . .	11
<b>3 Systemanalyse</b>	<b>13</b>
3.1 CAS genesisWorld . . . . .	13
3.1.1 Architektur . . . . .	14
3.1.2 Präsentationsschicht & Logikschicht . . . . .	14
3.1.3 Datenhaltungsschicht . . . . .	15
3.1.4 Server-SDK-Plugin . . . . .	15
3.2 Anforderungsanalyse . . . . .	16
3.2.1 Funktionale Anforderungen . . . . .	17
3.2.2 Nichtfunktionale Anforderungen . . . . .	17
3.3 Ermittlung relevanter Daten . . . . .	18
3.4 Charakteristika der Datenbankabfrage . . . . .	20
<b>4 Bestimmung einer Datenbank</b>	<b>21</b>
4.1 Einzelbetrachtung von Datenbanken . . . . .	21
4.1.1 CouchDB . . . . .	21
4.1.2 MongoDB . . . . .	22
4.1.3 Voldemort . . . . .	22
4.1.4 Redis . . . . .	22
4.1.5 HBase . . . . .	23
4.1.6 Cassandra . . . . .	23
4.1.7 VoltDB . . . . .	24

4.1.8	H2 . . . . .	25
4.1.9	Neo4J . . . . .	25
4.2	Gegenüberstellung . . . . .	25
4.3	Auswahl einer Datenbank . . . . .	28
<b>5</b>	<b>Konzeption</b>	<b>31</b>
5.1	Architektur . . . . .	31
5.2	Datenbankdesign . . . . .	33
5.2.1	Konzeptionelles Modell . . . . .	33
5.2.2	Datenbankschema . . . . .	34
5.2.3	Zugriffsstrukturen . . . . .	36
5.3	ETL-Prozess . . . . .	37
5.3.1	Extract . . . . .	37
5.3.2	Transform . . . . .	38
5.3.3	Load . . . . .	39
5.4	Entwurf der Oberfläche . . . . .	39
5.5	Technologien . . . . .	41
<b>6</b>	<b>Umsetzung</b>	<b>43</b>
6.1	Server-Webprojekt . . . . .	43
6.2	Client-Webprojekt . . . . .	45
6.3	Aufbau der H2-Datenbankabfrage . . . . .	46
6.4	ETL-Prozess . . . . .	49
6.5	Aktualisierung des Datenbestandes . . . . .	51
6.6	Oberfläche . . . . .	52
<b>7</b>	<b>Fazit und Ausblick</b>	<b>55</b>
7.1	Zusammenfassung . . . . .	55
7.2	Bewertung der Ergebnisse . . . . .	55
7.3	Ausblick . . . . .	57
<b>Literaturverzeichnis</b>		<b>59</b>

# Abbildungsverzeichnis

2.1	Beispiel einer Spalten-Familie . . . . .	6
2.2	Objekte in einer Graphdatenbank . . . . .	7
2.3	Das Konzept von COM . . . . .	10
3.1	Verknüpfungen in CAS genesisWorld . . . . .	13
3.2	Schematische Darstellung der Architektur von CAS genesisWorld . . . . .	14
3.3	ER-Modell . . . . .	15
3.4	Beispiel zur Benachrichtigung von Plugins anhand eines Ablaufs bei einem Update . . . . .	16
3.5	Auszug aus dem Schema der MSSQL-Datenbank . . . . .	18
4.1	Komponenten des Systems und der Umwelt . . . . .	23
4.2	Schematische Darstellung des Cassandra Datenmodells . . . . .	24
4.3	Datenmodell anhand eines Social-Network-Beispiels . . . . .	25
5.1	Komponenten des Systems und der Umwelt . . . . .	32
5.2	ER-Modell als Darstellungsform des Konzeptionelles Modell . . . . .	34
5.3	Neues Datenbankschema . . . . .	35
5.4	ETL-Prozess . . . . .	38
5.5	Entwurf der Oberfläche . . . . .	40
5.6	Entwürfe für die Oberfläche . . . . .	40
6.1	Server Klassendiagramm . . . . .	44
6.2	Client Klassendiagramm . . . . .	46
6.3	Ergebnisse der verschachtelten SQL-Abfrage . . . . .	47
6.4	Gewichtung der Zeit . . . . .	48
6.5	Vorgehensweise bei der Extraktion . . . . .	49
6.6	Ausschnitt einer CSV-Datei nach der Extraktion . . . . .	50
6.7	Klassendiagramm Plugin . . . . .	51
6.8	Sequenzdiagramm für einen neuen Datensatz . . . . .	52
6.9	Anmeldefenster . . . . .	53
6.10	Hauptseite der Anwendung . . . . .	54
7.1	Vergleich von Antwortzeiten . . . . .	57



# **Tabellenverzeichnis**

4.1	Gegenüberstellung der Datenbankeigenschaften . . . . .	27
5.1	Vergleich des Speicherplatzverbrauchs . . . . .	36
7.1	Abfragegeschwindigkeit Vergleich . . . . .	56



# 1. Einführung

## 1.1 Motivation

Produkte weisen eine stetig steigende Homogenität und eine damit verbundene Austauschbarkeit auf, wodurch es Unternehmen immer schwerer fällt sich über Produkte am Markt zu differenzieren. Dadurch werden Kunden- und Serviceorientierung besonders interessant für die Wettbewerbsdifferenzierung. Durch eine höherwertige und individuelle Kundenbearbeitung können für Unternehmen Wettbewerbsvorteile entstehen. Sämtliche Prozesse und Abläufe innerhalb eines Unternehmens die darauf abzielen werden unter dem Begriff Customer Relationship Management (CRM) zusammengefasst. Um CRM-Aktivitäten gezielt auf die Prozesse auszurichten müssen diese zuerst erkannt werden [WH04]. Weiterhin beschreibt CRM ein strategisches Konzept, das die Gewinnung und Bindung von Kunden durch den Einsatz von CRM-Software fördern soll [WH04]. Aus technologischer Sicht ist hiermit der Aufbau und die Nutzung einer Kundendatenbank gemeint. Welche Daten sie beinhaltet, hängt von der jeweiligen Zielsetzung des CRM-Systems ab. Fundamentale Daten wie die Adressen und Kontaktdaten der Kunden, sowie komplette Kundenhistorien (Telefonate, Meetings, E-Mails) sind allerdings in vielen CRM-Systemen vorhanden. Die Literatur teilt das CRM in folgende drei Bereiche auf: kommunikatives CRM, operatives CRM und analytisches CRM. Während das operative und kommunikative CRM den direkten Kontakt und die Steuerung der Kommunikationskanäle unterstützen, ist das analytische CRM für die Erhebung und Auswertung der Kundendaten zuständig [Hel13]. Infolgedessen unterscheiden sich nicht nur die Funktionen der Bereiche, sondern auch der Kontext in dem Daten betrachtet werden. Auswertungen beispielsweise setzen Daten völlig unabhängig von den operativen Geschäftsprozessen, in neue, logische Zusammenhänge. In der Regel gilt es diese separat von den operativen Daten aufzubewahren. An diesem Punkt setzt die vorliegende Arbeit an.

Die CAS Software AG besitzt mit CAS genesisWorld ein Produkt welches den kommunikativen und operativen Bereich des CRM abdeckt. Eine neue Überlegung des Unternehmens ist, Beziehungen von Personen untereinander zu untersuchen und ihre Ausprägung zu identifizieren. Innerhalb der Firma allerdings existiert keine Datenbank die eine optimale Form der Datenhaltung für solche Analysen bietet. Infolgedessen wurde in der vorliegenden Arbeit eine Lösung für die vorherige Überlegung erarbeitet.

## 1.2 Zielsetzung

Im Rahmen der Arbeit soll eine Lösung entwickelt werden mit der die Ausprägung von Beziehung zwischen den Personen aus CAS genesisWorld bewertet werden kann. Außerdem soll eine zufrieden stellende Antwortzeit (< 1s) erreicht werden. Das zu entwickelnde System soll einerseits auf dem Datenbestand von CAS genesisWorld basieren, andererseits aber auch unabhängig davon funktionieren. Aus technischer Sicht soll eine neue Datenbank und ein neuer Anwendungsserver eingesetzt werden, um Altlasten des bestehenden Systems zu umgehen und geringe Antwortzeiten zu erzielen.

Für die Auswahl einer Datenbank sollen technische Neuerungen der letzten Jahre, wie NoSQL- und In-Memory-Datenbanken, berücksichtigt werden. Dabei sollen Eigenschaften der Datenbanken betrachtet und verglichen werden. Zusätzlich sind Technologien für die Kommunikation und Anwendungslogik festzulegen. Weiterhin sind relevante Daten für das neue System aus der CAS genesisWorld Datenbank zu ermitteln. Überdies soll ein Prozess entworfen werden, um die Daten aus CAS genesisWorld zu extrahieren, transformieren und in die neue Datenbank einzufügen. Außerdem sollen die Funktionen des neuen Anwendungsservers über Schnittstellen ansprechbar sein. Um die Daten des neuen Systems aktuell zu halten sollen entsprechende Lösungswege zur Synchronisation erarbeitet werden. Weiterhin sind die Abfrageergebnisse für den Benutzer grafisch aufzubereiten. Die dazu entwickelte Oberfläche soll möglichst übersichtlich und einfach zu handhaben sein.

## 1.3 Gliederung der Arbeit

Die weiteren Arbeiten untergliedern sich in folgende Abschnitte:

**Grundlagen** In Kapitel 2 werden Grundlagen zum besseren Verständnis der Arbeit vermittelt. Zuerst wird auf den Begriff NoSQL aus dem Bereich der Datenbanken eingegangen. Dabei werden die unterschiedlichen Typen von NoSQL-Datenbanken vorgestellt. Nachdem ein Überblick über die Ausprägungen von NoSQL-Datenbanken gegeben wurde, werden die einschlägigen Begriffe im Bereich NoSQL erläutert. Die Begriffe werden im Voraus behandelt, da sie in der Evaluation von Datenbank auftauchen. Neben NoSQL gewann in den letzten Jahren der Terminus In-Memory an Aufmerksamkeit. Daher wird ein kurzer Einblick in die Thematik gegeben. Des Weiteren wird das Component Object Model erläutert. Die Grundlagen in dieser Technologie verschaffen einen Einblick in die technische Basis von CAS genesisWorld, welche für spätere Betrachtungen benötigt werden.

**Analyse** In Kapitel 3 wird die Architektur, sowie einzelne relevante Bestandteile von CAS genesisWorld untersucht. Weiterhin werden die für die Umsetzung benötigten Daten aus der CAS genesisWorld Datenbank ermittelt, die Anforderungen an das neue System erhoben und das umzusetzende Szenario näher beschrieben.

**Evaluation** Die Untersuchung, Gegenüberstellung und Auswahl einer geeigneten Datenbank wird im Kapitel 4 behandelt. Bei der Untersuchung der Datenbanken werden ihre Eigenschaften, sowie Stärken und Schwächen näher beschrieben. Weiterhin werden Eigenschaften für den Vergleich der Datenbanken festgelegt. Anschließend wird unter Beachtung der Anforderungen eine Datenbank ausgewählt.

**Konzeption** In der Konzeption wird die Architektur des neuen Systems entworfen. Weiterhin werden in Kapitel 5 Strukturen und Konzepte zur Definition eines Systemmodells entworfen. Darauf aufbauend werden die einzelnen Komponenten des Modells ausgearbeitet und die zur Umsetzung benötigten Technologien erläutert.

**Umsetzung** In Kapitel 6 wird auf die Umsetzung der Planungen eingegangen. Dabei wird auf abstrakte Weise beschrieben, wie die Implementierung arbeitet. Es wird bewusst auf den Einsatz von Quelltext verzichtet, um die Struktur und die Abläufe innerhalb der Komponenten in den Vordergrund zu stellen.

**Ergebnis** Die abschließende Betrachtung fasst die Ergebnisse der Arbeitsschritte in Kapitel 7 zusammen. Dabei wird weniger auf die konkreten Bestandteile eingegangen, sondern vielmehr auf die Charakteristika des neuen Systems. Das Vorgehen bei der Beschreibung wird durch die zuvor erhobenen Anforderungen geleitet. Zum Schluss schließt die Arbeit mit einem Ausblick auf weiterführende Gedanken.



## 2. Grundlagen

Das Kapitel Grundlagen geht zu Beginn auf den Begriff NoSQL ein und stellt verschiedene NoSQL-Implementierungen vor. Dabei wird unter anderem auf grundlegende Begriffe aus dem NoSQL Umfeld eingegangen. Anschließend werden Eigenschaften und Unterscheidungsmerkmale von In-Memory-Datenbanken behandelt. Abschließend soll ein Einblick in das Component Object Model (COM) gegeben werden. Dabei wird die allgemeine Funktionsweise dargelegt und wichtige Komponenten des Standards erläutert.

### 2.1 NoSQL - Eine Einführung

Der Terminus NoSQL bezeichnet Datenbanken die nicht dem Ansatz der relationalen Algebra folgen. Ihre Entstehung ist auf die schlechte horizontale Skalierbarkeit von relationalen Datenbanken zurückzuführen. Verfügbarkeit und Skalierbarkeit sind unter gewissen Umständen wichtiger als Atomarität und Konsistenz. Dieser Umstand führte neben der Entwicklung von NoSQL-Datenbanken zur Entstehung von Datenbanken die unter dem Terminus NewSQL zusammengefasst werden. Sie verfolgen einen anderen Ansatz als NoSQL-Datenbanken und werden im Rahmen dieser Arbeit nicht näher betrachtet, weshalb weiterhin auf [Sto11] verwiesen wird. Weiterhin lassen sich NoSQL-Datenbanken anhand ihres Datenmodells unterscheiden. Nach [Vai13] ist eine Klassifizierung in folgende Kategorien möglich:

#### 2.1.1 Document Stores

Document Stores koppeln komplexe Datenstrukturen (Dokumente) mit einem eindeutigen Schlüssel. Der Datenzugriff findet in der Regel über das HTTP-Protokoll mit REST-API oder über das Apache Thrift-Protokoll statt [ASK07]. In Document Stores gibt es außerdem kein Schema. Statt jeden Datensatz in einer Zeile bestehend aus Spalten zu speichern, werden sie in einem Dokument abgelegt. Diese können als eine Datei auf dem Dateisystem betrachtet werden. Solche Dokumente können alle möglichen Daten aufnehmen und müssen dabei keinem Schema folgen. Trotz der Schemafreiheit sind sie nicht frei von formellen Restriktionen. Die meisten der verfügbaren Datenbanken unter dieser Kategorie benutzen XML, JSON, BSON oder YAML. Document Stores eignen sich für den Einsatz von dynamischen Entitäten, die unregelmäßige Strukturen besitzen.

### 2.1.2 Extensible Record Store

Extensible Record Stores, auch Wide Column Stores genannt, speichern Daten mehrerer Einträge in Spalten anstatt in Zeilen. Jeder Eintrag einer Spalte besteht aus einem Namen, den Daten und einem Zeitstempel.

In Extensible Record Stores werden sogenannte Spalten-Familien zur Gruppierung ähnlicher oder verwandter Inhalte verwendet. In Abbildung 2.1 ist eine solche Spalten-Familie zu sehen. Spalten-Familien besitzen keine logische Struktur und geben somit kein Schema vor. Weiterhin können sie Millionen von Spalten beinhalten. Verwandte Spalten werden in Spalten-Familien durch eine von der Anwendung bereitgestellte Reihe von Schlüsseln identifiziert. Weiterhin muss in einer Spalten-Familie nicht jede Zeile aus den gleichen Spalten bestehen.

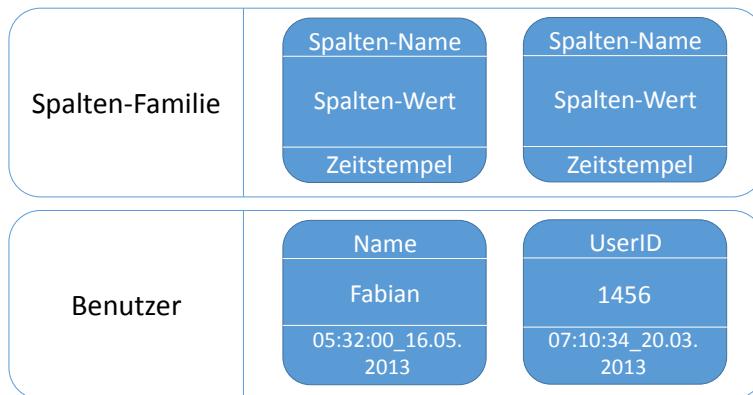


Abbildung 2.1: Beispiel einer Spalten-Familie

### 2.1.3 Key-Value-Store

Grundsätzlich verwendet der Key-Value-Store eine einfache Form der Datenspeicherung. Ein bestimmter Schlüssel referenziert auf einen Wert, der eine willkürliche Zeichenkette sein kann. In einigen Umsetzungen können die Werte außer Strings auch Listen, Sets oder auch Hashes beinhalten. Der Zugriff auf die Werte erfolgt über einen eindeutigen Schlüssel, d.h. jeder Schlüssel repräsentiert ein eindeutig identifizierbares Objekt. Im Gegensatz zu relationalen Datenbanken haben Key-Value-Stores keine Kenntnis über das Datenmodell und sind daher schemafrei. Sie setzen sich zum Ziel skalierbar und fehlertolerant zu sein. Zu den Einsatzorten zählen Web-Applikationen mit vielen aber einfachen Daten.

### 2.1.4 Graphdatenbank

Eine Graphdatenbank verwendet die Graphentheorie zur Abbildung und Abfrage von Beziehungen [RWE13]. Im Grunde besteht eine solche Datenbank aus einer Menge von Knoten und Kanten. Jeder Knoten repräsentiert dabei eine Entität, wohingegen Kanten Beziehungen oder Verbindung zwischen zwei Knoten darstellen. Abbildung 2.2 verdeutlicht dies in einem Beispiel. Knoten definieren sich durch einen sogenannten "unique identifier", sowie durch die Anzahl abgehenden und/oder eingehenden Kanten und einer Menge von Attributen. Kanten werden wie Knoten definiert, nur dass diese, anstatt Knoten, einen Start- und End-Knoten besitzen. Graphdatenbanken eignen sich gut für die Analyse von Verbindungen, weshalb sie oft zur Datengewinnung im Social Media Umfeld genutzt werden.

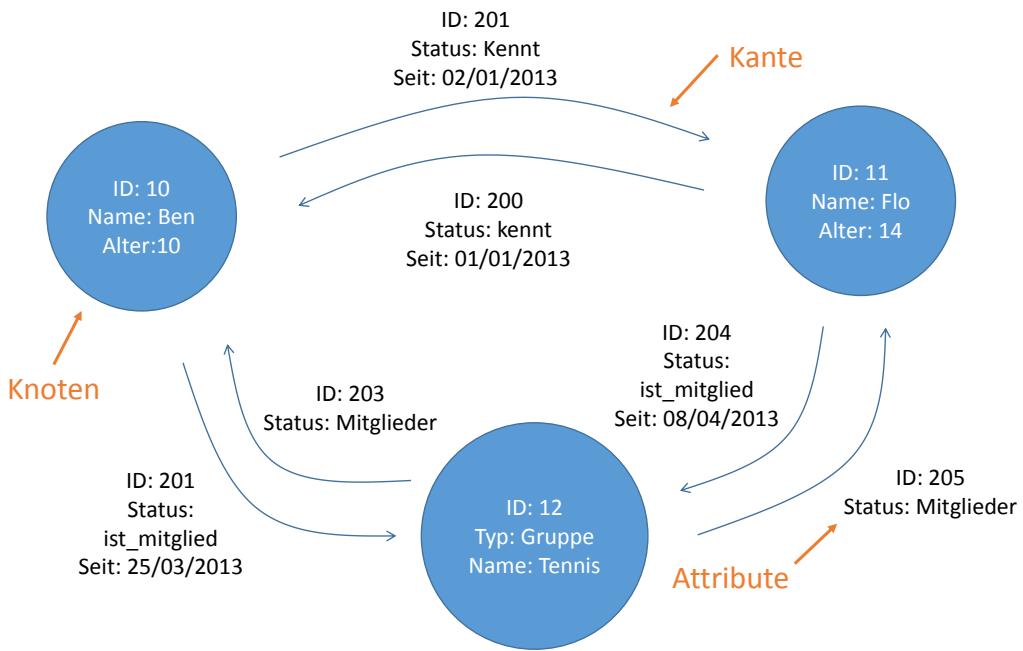


Abbildung 2.2: Objekte in einer Graphdatenbank

### 2.1.5 Theoretische Grundlagen

Im Nachfolgenden werden die durch die NoSQL Bewegung geprägten Begriffe und Konzepte erläutert.

**Replikation** Replikation im Falle von verteilten Datenbanken bedeutet, dass ein Daten-element auf mehr als einem Knoten (Computer) gespeichert ist. Dies ist sehr nützlich, um Leseleistungen der Datenbanken und deren Ausfallsicherheit zu erhöhen. Ermöglicht wird dies durch einen Load-Balancer, der Lesevorgänge über viele Maschinen verteilt.

**Fragmentierung** Fragmentierung in der Datenbank bedeutet eine Verteilung des Datenbestandes auf verschiedene Fragmente. Diese können dann über viele Knoten verteilt werden. Die Datenpartitionierung kann beispielsweise mit einer konsistenten Hash-Funktion erfolgen, die auf dem Primärschlüssel der Datenelemente angewendet wird, um das zugehörige Fragment zu bestimmen.

**Eventual-Consistency** Später in diesem Kapitel wird das CAP-Theorem eingeführt, welches besagt, dass verteilte Datenbanken entweder stark konsistent oder verfügbar sein können. Da die meisten NoSQL-Datenbanken Verfügbarkeit priorisieren, wird in diesen Datenbanken das Konzept Eventual-Consistency eingesetzt. Es stellt eine abgeschwächte Art der starken Konsistenz dar. Starke Konsistenz bedeutet, dass alle mit der Datenbank verbundenen Prozesse immer die gleiche Version der Daten sehen. Eventuelle Konsistenz ist schwächer und garantiert nicht, dass jeder Prozess die selbe Version sieht.

**Multiversion Concurrency Control (MVCC)** MVCC ist eine effiziente Methode, mehrere Prozesse auf die selben Daten parallel zugreifen zu lassen, ohne eine Beschädigung der Daten und Deadlocks zu riskieren. Es ist eine Alternative zu den Lock-basierten Ansätzen,

bei der jeder Prozess zuerst eine exklusive Sperre auf einem Datenelement anfordern muss, bevor er ihn lesen oder aktualisieren kann. Zu diesem Zweck werden intern verschiedene Versionen eines Objektes gehalten.

**MapReduce** MapReduce ist ein von Google entwickeltes Programmiermodell für verteilte Berechnungen und wurde zuerst in einem Artikel von Dean und Ghemawat [DG08] beschrieben. Anwendungen, die mit dem MapReduce-Framework geschrieben werden, können automatisch auf mehreren Computern verteilt werden, ohne dass der Entwickler einen benutzerdefinierten Code für die Synchronisation und Parallelisierung schreiben muss. MapReduce wird in Fällen verwendet in denen einzelne Maschinen zu lange bräuchten um die gegebene Aufgabe zu bewältigen.

**Vektoruhren** Vektoruhren basieren auf der Arbeit von Lamport [Lam78] und werden von vielen Datenbanken verwendet, um festzustellen, ob ein Datenelement durch konkurrierende Prozesse verändert wurde. Jedes Datenelement besitzt eine Vektoruhr, welche aus Tupeln mit verschiedenen Zeitpunkten besteht. Jeder Zeitpunkt stellt einen Prozess dar, der eine Modifikation an dem Datenelement vorgenommen hat. Jede Uhr beginnt bei Null und wird durch seinen Prozess bei jedem Schreibvorgang erhöht. Um den eigenen Wert der Uhr zu erhöhen, verwendet der Schreibprozess das Maximum aller Werte der Uhren im Vektor und erhöht sie um eins. Wenn zwei Versionen eines Elements zusammengeführt werden, können die Vektoruhren benutzt werden, um Konflikte zu erkennen. Wenn mehr als ein Wert einer Uhr differenziert, muss ein Konflikt vorhanden sein. Wenn es keinen Konflikt gibt, kann die aktuelle Version durch den Vergleich der Maxima der Uhren ermittelt werden.

**Das CAP-Theorem** Das CAP-Theorem wurde von Brewer erstmals in einem Symposium [Bre00] über den Trade-Off in verteilten Systemen eingeführt und wurde später von Gilbert und Lynch [GL02] formalisiert. Es besagt, dass in einem verteilten Datenspeichersystem nur zwei Merkmale aus Verfügbarkeit, Konsistenz und Partitionstoleranz garantiert werden können. Verfügbarkeit bedeutet in diesem Fall, dass die Clients in einem bestimmten Zeitraum immer Daten lesen und schreiben können. Eine partitionierte, verteilte Datenbank ist fehlertolerant gegenüber temporären Verbindungsproblemen und arbeitet auch bei Verlust von Nachrichten, einzelner Netzknoten oder Partition des Netzes weiter. Ein System das tolerant partitioniert ist, kann nur eine starke Konsistenz durch Verminderungen in seiner Verfügbarkeit erreichen. Grund dafür ist, dass es zuerst sicherstellen muss, ob jeder Schreibvorgang abgeschlossen wurde, bevor er eine Replikation durchführen kann. Allerdings kann es vorkommen, dass dies in einer verteilten Umgebung nicht möglich ist. Ursachen dafür können Verbindungsfehler oder andern temporäre Hardwareprobleme sein. Konsistenz bedeutet im Falle des CAP-Theorems, dass alle Knoten zur selben Zeit dieselben Daten sehen.

## 2.2 In-Memory-Datenbanken

Eine In-Memory-Datenbank (IMDB) ist ein Datenbankmanagementsystem, das in erster Linie den Hauptspeicher als Medium für die Datenablage verwendet. Eine IMDB wird auch als Main-Memory-Database (MMDB) oder Real-Time-Database (RTDB) bezeichnet. IMDBs sind schneller als die auf Festplatten zugreifenden Datenbanken, da der Hauptspeicher wesentlich niedrigere Zugriffszeiten aufweist. Außerdem führen sie weniger CPU-Befehle beim Lesen und Schreiben aus und ihre internen Optimierungsalgorithmen sind viel

einfacher gestaltet. Einsatz finden sie vor allem in Anwendungen in denen kurze Reaktionszeiten von entscheidender Bedeutung sind. Mehrkernprozessoren, 64-bit-Architekturen und gesunkene RAM-Preise stellen die treibenden Faktoren in der Entwicklung solcher Systeme dar [Pla13a].

Die hohe Performance dieser Systeme resultiert nicht nur durch die Datenhaltung im Hauptspeicher. Vielmehr müssen bisherige Konzepte im Datenbankentwurf neu überdacht werden. Beispielsweise besitzen IMDB, die den relationalen Ansatz verfolgen, geänderte Abfrageoptimierer. In herkömmlichen RDBMS sind Lese- und Schreiboperationen eine der wichtigsten Faktoren zur Bestimmung des optimalen Abfrageplans. In IMDB spielen sie allerdings eine stark untergeordnete Rolle. Im Gegenzug nimmt die Reduktion von CPU-Zyklen einen höheren Stellenwert ein.

In herkömmlichen Datenbanken ist der Speicherverbrauch kein relevanter Faktor. In IMDB hingegen ist der Einsatz von speicherplatzsparenden Maßnahmen eine Notwendigkeit. Dictionary Encoding, Run-Length Encoding oder Cluster Encoding sind nur einige Techniken zur Reduktion des Speicherplatzverbrauches. Solche Techniken bieten sich vor allem in spaltenorientierten Systemen aufgrund der eher geringeren Entropie innerhalb der Spalten an [AMF06]. Neben den Optimierungsansätzen in der Datenhaltung können Regeln formuliert werden, um nicht mehr verwendete Daten zu erkennen. Dabei kann z.B. zwischen aktiven Daten (Daten von nicht abgeschlossenen Geschäftsprozessen) und passiven Daten (Daten von abgeschlossenen Geschäftsprozessen) unterschieden werden [LLS13]. Wenn ein Geschäftsprozess in sich abgeschlossen ist, werden die Daten nur noch aus Datenvorhaltsgründen aufbewahrt. Die zur Datenaufbewahrung benötigte Hauptspeicherkapazität kann durch solche Regeln stark reduziert werden.

In traditionellen Datenbanken stellt das Wiederherstellen aufgrund des nicht flüchtigen Speichers kein Problem dar. IMDB müssen dagegen für den Fall eines Systemausfalls Snapshot-Dateien anlegen. Diese werden zur Wiederherstellung des Datenbestandes benötigt. Snapshots sind Abbilder des aktuellen Datenbestandes. Um Rücksicht auf die Performance zu nehmen, werden die Snapshots entweder in Intervallen oder zu festgelegten Ereignissen erzeugt. Damit die Veränderungen an Daten zwischen Snapshots nicht verloren gehen, werden sie in Log Dateien zwischengespeichert. Zusammen mit den Snapshots bilden sie die Grundlage für die Datenwiederherstellung.

An dieser Stelle endet der Abschnitt über Datenbanken. Im Folgenden wird auf das Component Object Model eingegangen.

## 2.3 Component Object Model

Das Component Object Model (COM) ist ein binärer Schnittstellenstandard für Software-Komponenten, der von Microsoft im Jahr 1993 eingeführt wurde [Loo01]. Es wird verwendet um Interprozesskommunikation und dynamische Objekterstellung in einer Vielzahl von Programmiersprachen zu ermöglichen. Um zu verstehen was COM ist (und damit alle COM-basierten Technologien), muss einem klar sein, dass es sich nicht um eine objektorientierte Sprache, sondern um einen Standard zur Beschreibung von Objektmodellen handelt. Er definiert keine Sprache, Struktur oder Implementierungsdetails. Die jeweilige Umsetzung wird dem Programmierer überlassen. Es spezifiziert lediglich ein Objektmodell und die Anforderungen an die Kommunikationen zwischen COM-Objekten und anderen Objekten. Es spielt dabei keine Rolle, ob Objekte sich im gleichen oder in unterschiedlichen Prozessen befinden. Sie können sogar auf unterschiedlichen Rechnern laufen. Die Implementierung in verschiedenen Sprachen ist durch das Überführen der Kommunikation in binären Maschinencode möglich. Das führt dazu, dass COM des öfteren als binärer Standard referenziert wird.

COM bietet die Möglichkeit auf viele Windows-Funktionen direkt zuzugreifen. Des Weiteren ist COM die Basis für die OLE-Automation<sup>1</sup>(Object Linking and Embedding) und ActiveX<sup>2</sup>. Die Verwendung des COM-Standards bietet folgende Vorteile:

- Sprachunabhängigkeit
- Versionsunabhängigkeit
- Plattformunabhängigkeit
- Objektorientierung
- Ortsunabhängigkeit
- Automatisierung

### 2.3.1 Architektur

Wie in Abbildung 2.3 zu sehen, erzeugt ein COM-Client eine COM-Komponente in einem so genannten COM-Server und nutzt die Funktionalität des Objektes über COM-Schnittstellen.

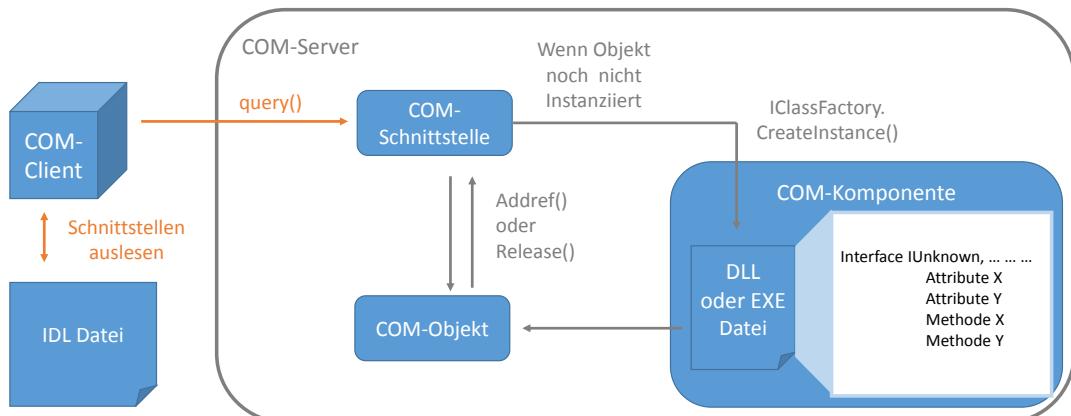


Abbildung 2.3: Das Konzept von COM

### 2.3.2 COM-Client

Der COM-Client stellt den Benutzer einer COM-Komponente dar. Die Nutzung der COM-Komponenten erfolgt über sogenannte Interfaces. Interfaces liegen in Form von Beschreibungen in der Interface Definition Language (IDL) vor. Einem Client steht außerdem die Möglichkeit zur Verfügung, abzufragen ob ein Objekt das angefragte Interface unterstützt. Dabei wird lediglich eine Abfrage an das ausgewählte Objekt gestellt, die eine Globally Unique Identifier (GUID)<sup>3</sup> als Übergabeparameter besitzt. Falls das Objekt das geforderte Interface unterstützt, liefert es den entsprechenden Pointer zur Methode zurück.

<sup>1</sup>OLE ist ein dynamisches Datenaustauschverfahren zur dynamischen Verknüpfung von Objekten auf der Desktop-Ebene. Dadurch können Daten von OLE-fähigen Anwendungen untereinander verknüpft werden

<sup>2</sup>ActiveX bezeichnet ein Softwarekomponenten-Modell. Es ermöglicht den Zugriff auf Datenbanken sowie weiteren Anwendungen und Programmierungen. Im Internet-Explorer beispielsweise wird mithilfe von ActiveX der MediaPlayer zum öffnen von Multimedia-Dateien aufgerufen

<sup>3</sup>Die GUID ist eine global eindeutige Zahl. In COM wird sie zur Identifikation von Schnittstellen verwendet.

### 2.3.3 COM-Server

Ein COM-Server wird durch eine DLL oder ausführbare Datei realisiert, die eine COM-Komponente beinhaltet oder bereitstellt. Dabei wird zwischen 3 Arten von COM-Servern unterschieden. Die erste Variante ist der In-Process-Server, der sich dadurch auszeichnet, dass er beim Instanziieren einer COM-Komponente in den Prozess der Anwendung (COM-Client) übertragen wird. Der Local-Server hingegen tritt in Form eines ausführbaren Programmes auf, der COM-Komponenten implementiert. Dieser wird gestartet sobald ein COM-Client die COM-Komponente des Servers instanziert. Die Kommunikation erfolgt über ein RPC-Protokoll. Die dritte Variante ist der Remote-Server, der verwendet wird sobald COM über ein Rechnernetz kommunizieren soll. Dabei wird DCOM (Distributed COM) verwendet, die eine spezielle Variante von COM darstellt.

### 2.3.4 COM-Schnittstelle

Die COM-Schnittstellen ermöglicht dies durch die Angabe eines einzigen Weges (Schnittstelle), um die Daten eines Objektes zu verändern. Eine COM-Schnittstelle bezieht sich auf eine vordefinierte Gruppe von verwandten Funktionen aus einer Klasse. Eine Schnittstelle allerdings muss nicht unbedingt alle Funktionen unterstützen die eine Klasse implementiert. Eine Schnittstellenimplementierung wird mit einem Objekt verbunden, sobald eine Instanz des Objektes erzeugt wurde und die Implementierung die Dienste des Objektes bereitstellt.

Eine typische Vorgehensweise für die Entwicklung von Interfaces ist es Funktionalitäten und Daten die der Lösung eines Problems dienen in einem Interface zusammenzufassen. Ein Interface spiegelt dabei ein Verhalten innerhalb einer Problemdomäne wieder. Im Anschluss werden COM-Klassen durch Entwickeln verschiedener Objekttypen gebildet. Objekttypen repräsentieren Entitäten, die verschiedene Kombinationen von Interfaces benutzen, basierend auf dem gewünschten Verhalten der Entität.

### 2.3.5 COM-Objekte

Ein COM-Objekt bietet Funktionen des COM-Servers über ein Interface an. Durch die Implementierung *IClassFactory.CreateInstance()* kann eine Instanziierung im COM-Server vorgenommen werden. Zurückgeliefert wird dann eine Instanz der Klasse. COM-Objekte müssen nicht wieder freigegeben werden, da der COM-Server dies selbst steuert. Bei der Instanziierung eines Objektes wird ein Referenzzähler hochgezählt. Dieser wird durch den Aufruf von *Release()* wieder dekrementiert. Solange der Zähler ungleich 0 ist, bleibt das Objekt erhalten.

### 2.3.6 Interface Definition Language

Die Syntax der Microsoft Interface Definition Language (MIDL) basiert auf der Syntax der Programmiersprache C. Das MIDL-Design gibt zwei verschiedene Dateien vor: die Interface Definition Language (IDL)-Datei und die Anwendungskonfigurationsdatei (ACF). Die IDL-Datei enthält eine Beschreibung der Schnittstelle zwischen Client- und Serveranwendung.



## 3. Systemanalyse

Zu Beginn der Arbeiten wird eine Systemanalyse zur Ermittlung des Ist- und Sollzustandes durchgeführt. Nach [Rup13] versteht man darunter das Beschreiben der vorhandenen und zukünftigen Systeme. Zuerst wird in Abschnitt 3.1 eine Ist-Analyse durchgeführt. In Abschnitt 3.2 wird auf die, an das System gestellt, Anforderungen eingegangen. Aufbauend auf den Anforderungen werden in Abschnitt 3.3 die relevanten Daten für die Umsetzung ermittelt.

### 3.1 CAS genesisWorld

CAS genesisWorld ist eine Software, die Organisation und Zusammenarbeit in Kundenbeziehungen und zwischen Kollegen steigern soll. Alle Informationen bzw. Daten werden in CAS genesisWorld zentral gespeichert und sind so für alle verfügbar. Welche Daten ein Anwender sieht, hängt von seinen Rechten und Einstellungen ab. Die Daten, d.h. Termine, Aufgaben, Adressen, Dokumente usw. werden in CAS genesisWorld von den Nutzern gepflegt und aktuell gehalten. Darüber hinaus lassen sich wie in Abbildung 3.1 dargestellt, alle Daten beliebig miteinander verknüpfen. So werden zusätzliche Zusammenhänge deutlich und der Informationsgehalt steigt. Ein Besprechungstermin lässt sich beispielsweise mit den Adressen der Teilnehmer und dem Dokument der Tagesordnung verknüpfen.

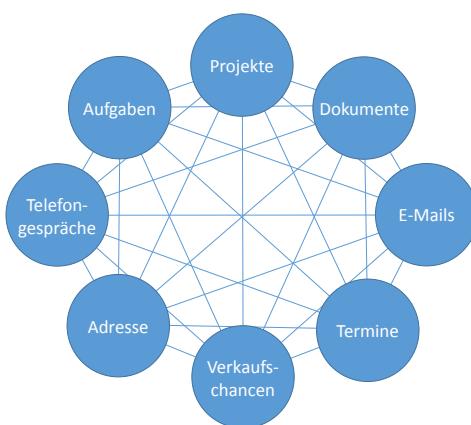


Abbildung 3.1: Verknüpfungen in CAS genesisWorld

### 3.1.1 Architektur

Die N-Tier-Architektur von CAS genesisWorld lässt sich in drei wesentliche Bereiche gliedern:

- Die Präsentationsclients umfassen alle Dienste, die Informationen in Bildschirman-sichten den Benutzern zur Verfügung stellen.
- Der Applikationsserver umfasst alle Dienste, um die Geschäftslogik zu kapseln, Än-derungen zu protokollieren, Benutzerrechte zu prüfen und die aufbereiteten Informa-tionen den Präsentationsdiensten zur Verfügung zu stellen.
- Die Datenbankschicht umfasst alle Dienste die zur Datenhaltung selbst notwendig sind.

### 3.1.2 Präsentationsschicht & Logikschicht

Der CAS genesisWorld Client existiert in Form einer Windowsanwendung, als mobile Ver-sion in Android, Windows Phone, BlackBerry OS und iOS. Die Kommunikation der Clients mit CAS genesisWorld findet über das REST-Protokoll statt [CSA13].

Die Funktionalität des CAS genesisWorld Anwendungsservers wurde in Form von COM-Objekten implementiert. Damit stehen dessen Dienste auch Dritten zur Verfügung, die dadurch mit eigenen Applikationen die Informationen von CAS genesisWorld präsentieren oder weiterverarbeiten können. Eine erster Überblick der CAS genesisWorld Komponenten ist in Abbildung 3.2 zu sehen. Als Basisdienste stehen der UserService und der DataService zu Verfügung. Für die Anmeldung und Rechteverwaltung ist der UserService zuständig. Der DataService ist als zentraler Dienst für den Zugriff auf die CAS genesisWorld Daten verantwortlich. Die Schnittstelle des DataService wurde an Microsoft ADO angelehnt. Auf den Basisdiensten aufbauend existieren die Geschäftsdiene, in Form der Schnittstellen der BusinessServices. Diese bieten spezielle Funktionen zu den jeweiligen Anwendungsbe-reichen.



Abbildung 3.2: Schematische Darstellung der Architektur von CAS genesisWorld

### 3.1.3 Datenhaltungsschicht

Die Datenhaltungsschicht enthält einen Microsoft SQL Server 2008 (MSSQL). Der SQL Server ist ein relationales Datenbankmanagementsystem (RDBMS) von Microsoft, dass für den Einsatz im Konzernumfeld konzipiert wurde. MSSQL verwendet T-SQL (Transact-SQL), eine Erweiterungen von Sybase und Microsoft, die den SQL-Standard um prozedurale Sprachelemente erweitert [Cor13]. Weiterhin unterstützt MSSQL standardisierte Datenbankschnittstellen, wie Open Database Connectivity (ODBC) und Java Database Connectivity (JDBC).

Beim Schema der Datenbank wird auf eine Besonderheit eingegangen. In relationalen Datenbanken werden Beziehungen über Primär- und Fremdschlüssel abgebildet. Dies ist auch in der MSSQL-Datenbank der Fall, jedoch mit einer Besonderheit. In der MSSQL-Datenbank werden nur Primärschlüssel deklariert. Es werden Fremdschlüssel verwendet allerdings sind sie nicht als solche deklariert.

Ein Grund dafür ist die Art wie die Funktionalität in der Datenbank umgesetzt wurde, die eine Verknüpfung sämtlicher CRM-Objekte ermöglicht. Abbildung 3.3 zeigt das ER-Modell für die Funktionalität. Bei 398 Tabellen existieren durch die Funktionalität sehr viele M:N-Beziehungen. Anstatt für jede M:N-Beziehung eine separate Tabelle anzulegen wurde lediglich eine Tabelle verwendet. Sie besitzt vier relevante Spalten. In zwei von ihnen werden die Fremdschlüssel der in Beziehung zu setzenden Tabellen aufbewahrt. Damit die nächst höhere Schicht die Fremdschlüssel den entsprechenden Tabellen zuordnen kann werden in den anderen beiden Spalten die Zuordnungskürzel hinterlegt. Jede Tabelle besitzt sein eigenes eindeutiges Kürzel. Die Nutzung einer Auflösungstabelle ist nur möglich weil die beiden Spalten in denen die Fremdschlüssel aufbewahrt werden nicht als solche deklariert sind.

Weiterführende Erläuterungen zum relevanten Teil des Schemas werden in Abschnitt 3.3 behandelt.

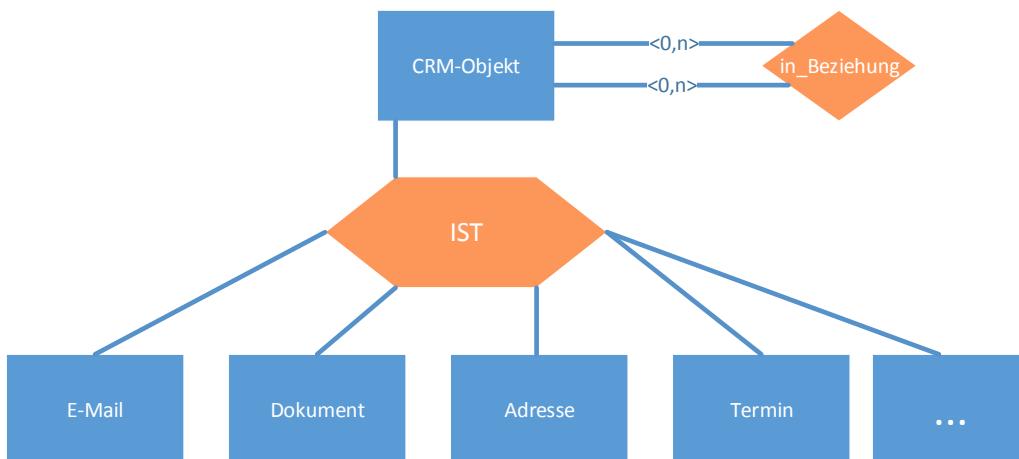


Abbildung 3.3: ER-Modell

Die Spalten *GUID1* und *GUID2* beinhalten die jeweiligen Primärschlüssel der in Beziehung zu setzenden Tabellen. Mithilfe der Spalten *TableSign1* und *TableSign2* können die *GGUIDs* den Tabellen, aus denen sie entstammen, zugeordnet werden. Die *GGUID* ist in der gesamten Datenbank eindeutig und dient als Primärschlüssel für jede Tupel in der Datenbank.

### 3.1.4 Server-SDK-Plugin

Die Server-SDK-Plugins bieten die Möglichkeit die Datenverarbeitung, um eine eigene Logik zu erweitern oder zu modifizieren.

Realisiert werden die Plugins als COM-Objekte, die ein Plugin-Interface namens *IGWSDK-DataPlugIn* implementieren. Das erstellte COM-Objekt wird im Server von CASgenesis-World registriert. Der Server delegiert bei einer Datenoperation den Aufruf an die für den jeweiligen Datensatztypen registrierten Plugins. In Abbildung 3.4 ist ein Beispiel des Vorgangs dargestellt. Die CASTable ist für die Delegation der Datenmanipulation-Anweisungen zuständig. Sie empfängt Anweisungen vom DataService und führt diese auf dem MSSQL-Server aus. Außerdem besitzt sie mit dem Plugin-Direktor eine Komponente mit den Plugins über Änderungen in den Datensätzen informiert werden. Wie in der Abbildung zu sehen werden zuerst die fest integrierten Plugins wie das CAS-Address-Plugin benachrichtigt. Anschließend werden die von Dritten erstellten Plugins informiert.

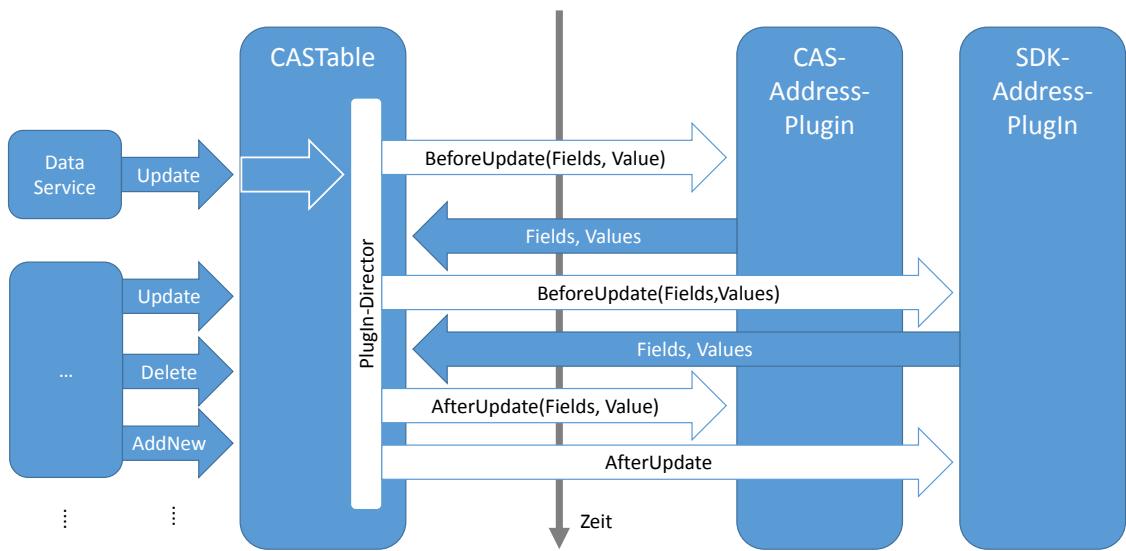


Abbildung 3.4: Beispiel zur Benachrichtigung von Plugins anhand eines Ablaufs bei einem Update

Im Allgemeinen stehen in den COM-Schnittstellen der Plugins, jeweils alle Felder eines Datensatztypen zur Verfügung, sowie die neuen Werte der Felder. In den Plugins besteht somit die Möglichkeit, alte bzw. neue Werte von Feldern zu untersuchen und zu vergleichen und auf das Ergebnis zu reagieren.

Die Werte des aktuell verarbeiteten Datensatzes können verändert, d.h. erweitert oder reduziert werden. Darüber hinausgehend sind auch automatisierte Aktionen realisierbar, die weitere Datensätze betreffen. So könnten z.B. abhängig von den Eingangswerten einer neu angelegten Adresse, neue Aufgaben angelegt und mit Inhalt versehen werden. Einige automatische Datenoperationen von CAS genesisWorld werden über CAS-Plugins realisiert, die mit den SDK-Plugins verwandt sind.

## 3.2 Anforderungsanalyse

Während der Anforderungsanalyse wird ermittelt, welche Eigenschaften und Fähigkeiten das System zur Erreichung der Ziele benötigt. Bei der Einteilung der Anforderungen wird zwischen Funktionalen und Nichtfunktionalen unterschieden. Bei ersterem wird die Funktionalität des zu erstellenden Systems beschrieben, wohingegen die Randbedingungen und Qualitätsanforderungen unter letzteres fallen.

### 3.2.1 Funktionale Anforderungen

Der Kernfunktionalität des Systems ist die Bewertung von Beziehungen zwischen Personen aus einem CRM-System. Die Bewertung soll auf sogenannten CRM-Objekten basieren. Jedes Objekt repräsentiert ein anderes Element des CRM-Systems. Die Adresse einer Person ist ein solches CRM-Objekt, allerdings kann sie nur einer Person zugeordnet werden. In der Umsetzung der Funktionalität spielen Objekte, die mehrere Personen betreffen, eine Rolle. Um eine Beziehung bewerten zu können werden Objekte herangezogen, die einen Austausch von Informationen unter Personen wiederspiegeln.

Das erste Objekt, welches dem Kriterium gerecht wird, ist der Termin, während dessen eine Zusammenkunft bestimmter Personen stattfindet. Das Dokument stellt das nächste Objekt dar. Dessen Eignung beruht auf der Möglichkeit, anderen Personen Zugriffsrechte und somit Einsicht auf Dokumente zu gewähren. Zur Beachtung des Informationsaustausches durch verbale sowie schriftliche Kommunikation, werden die E-Mail und das Telefonat einbezogen. Das letzte Objekt ist die Verkaufschance. Sie repräsentiert eine Aussicht auf einen potentiellen Abschluss eines Geschäfts. Durch sie können Vertriebsmitarbeiter ihre Leads<sup>1</sup> strukturiert und organisiert ablegen.

Der Wert einer Beziehung soll anhand der Anzahl von Objekten, in denen beide Personen vorkommen, ermittelt werden. Ein Beispiel für eine solche Anzahl von Objekten sind alle Termine, an denen beide Personen teilgenommen haben. Summiert mit der Anzahl der anderen Objekte ergibt sich die Wertigkeit der Beziehung.

Weiterhin soll der Benutzer die Bewertung aller Beziehungen zu einer Person ermitteln können. Die Beziehungen sind in einer absteigenden Reihenfolge nach ihren Werten zu präsentiert. Das Ergebnis soll außerdem durch den Benutzer auf eine von ihm festgelegte Menge an Beziehungen eingrenzbar sein. Neben der Anzahl von Beziehungen, soll auch der für die Bewertung betrachtete Zeitraum verändert werden können. Es soll auch möglich sein, den Zeitraum unterschiedlich zu gewichten. Dazu sind zwei Zeitspannen festzulegen. Die eine beginnt am Anfang des Zeitraums und endet zu einem angegebenen Zeitpunkt. Die andere beginnt zu einem angegebenen Zeitpunkt und reicht bis zum Ende des Zeitraums. Beide Zeitpunkte sollen durch den Benutzer angegeben werden können. Nicht nur der Zeitraum, sondern auch die CRM-Objekte sollen gewichtbar sein. Hierzu soll dem Benutzer die Möglichkeit offen stehen, einzelne CRM-Objekte unterschiedlich zu gewichten. Überdies soll er entscheiden können, welche Personen, Gruppen, Städte, Kontakte, Firmenkontakte, Mitarbeiter und Länder aus der Bewertung ausgeschlossen werden.

### 3.2.2 Nichtfunktionale Anforderungen

Folgende nichtfunktionale Anforderungen wurden erhoben:

- Falls die Möglichkeit besteht nur eine Rechnerinstanz für den Datenbankserver und Applikationsserver verwenden.
- Das System soll sehr kurze Antwortzeiten (< 1s) in der Beantwortung von Benutzerabfragen aufweisen.
- In der Implementierung soll eine möglichst lose Kopplung zwischen der Anwendungslogik und den Komponenten der Darstellung erreicht werden.
- Das System soll eine hohe Portabilität besitzen, damit eine einfache Inbetriebnahme auf anderen Rechnern möglich ist.
- Die Ergebnisse der Bewertung sind dem Nutzer graphisch aufzubereiten.

---

<sup>1</sup>Stellt eine erfolgreiche Kontaktanbahnung eines Produkt- oder Dienstleistungsanbieters zu einem potenziellen Interessenten dar

- Der Einsatz von Open-Source-Produkten ist gegenüber den von kommerziellen Produkten vorzuziehen.

### 3.3 Ermittlung relevanter Daten

In diesem Abschnitt werden die relevanten Datensätze aus der MSSQL-Datenbank erörtert. Dazu werden die Tabellen aus Abbildung 3.5 näher beschrieben. Sie werden zum Realisieren der funktionalen Anforderungen benötigt.

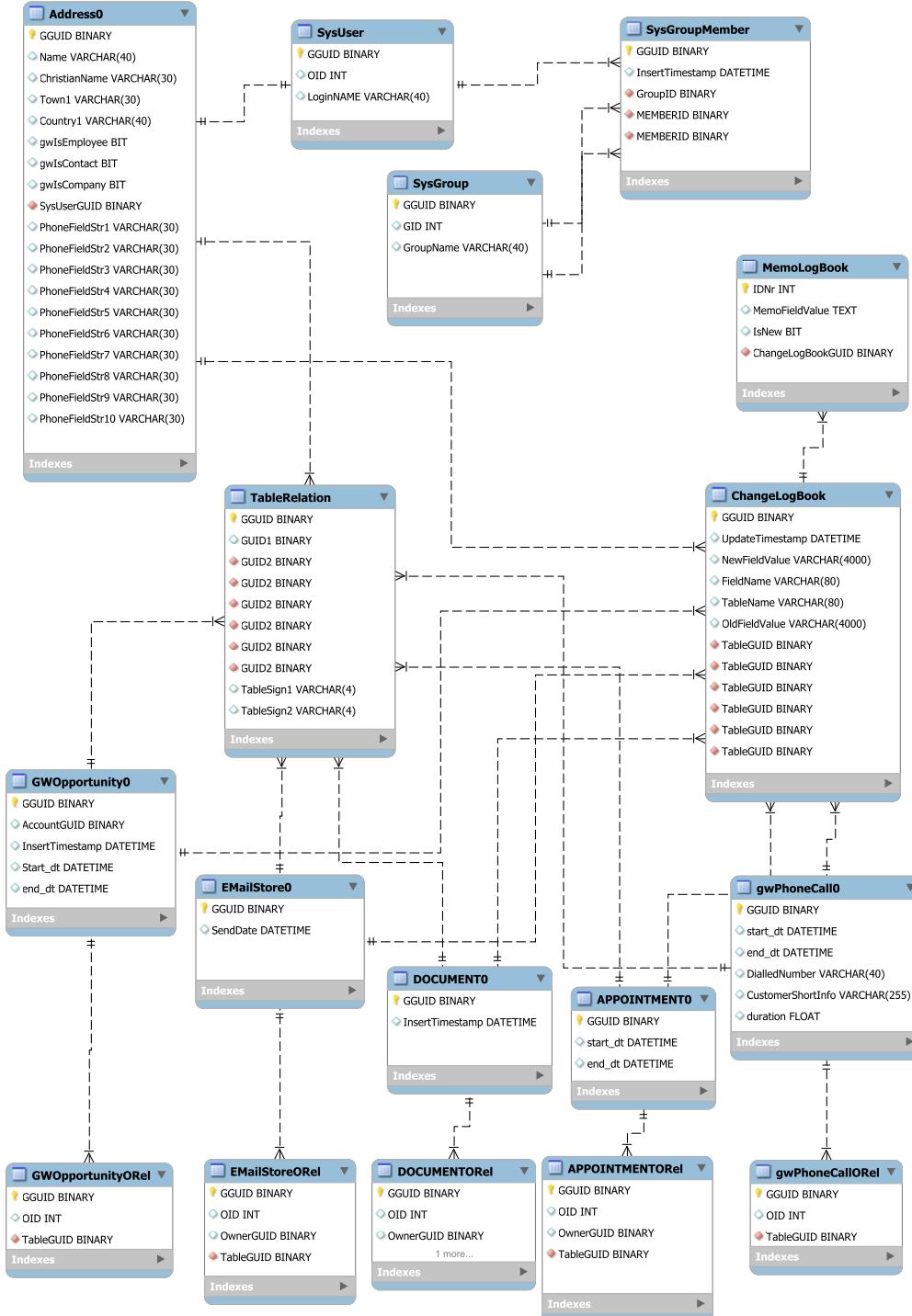


Abbildung 3.5: Auszug aus dem Schema der MSSQL-Datenbank

Die erste Tabelle *SysUser* beinhaltet Informationen über Personen. Für die Umsetzung des Systems werden drei Attribute der Tabelle benötigt. Zum einen den Primärschlüssel der Tabelle, die *GGUID*. Sie wird des Weiteren nicht mehr erwähnt, da sie in jeder Tabelle den Primärschlüssel darstellt. Das Attribut *OID* wird in anderen Tabellen als Fremdschlüssel verwendet und wird somit zur Identifikation der jeweiligen Person benötigt. Das Attribut *LoginName* repräsentiert den Benutzernamen der Personen und wird im neuen System weiterverwendet.

Um Personen, die bestimmten Gruppen angehören, aus der Bewertung von Beziehungen auszuschließen werden die Tabellen *SysGroupMember* und *SysGroup* benötigt. Die erste der beiden Tabellen wird zur Realisierung der M:N-Beziehung zwischen *SysUser* und *SysGroup* verwendet. Sie beinhaltet neben den Fremdschlüsseln der andern Tabellen einen Zeitstempel. Dieser ermöglicht es nachzuvollziehen wann eine Person in eine Gruppe eingetreten ist. Der Zeitstempel wird zur Rekonstruktion von früheren Gruppenzusammensetzungen benötigt. Dadurch können die Personen ermittelt werden die zu einem Zeitpunkt in der Vergangenheit einer Gruppe angehörten oder auch nicht. Die Tabelle *SysGroup* enthält weiterhin die Attribute *GroupName* und *GID*. Ersteres beinhaltet den Namen einer Gruppe. Das andere Attribut wird benötigt, da es in anderen Tabellen als Fremdschlüssel verwendet wird.

Weiterhin können Personen aus bestimmten Ländern oder Städten in der Bewertung nicht beachtet werden. Zu dessen Umsetzung wird die Tabelle *Address0* benötigt. Sie enthält die gesamte Anschrift der jeweiligen Personen allerdings werden nur die Attribute *Town1* und *Country1* verwendet. Neben der Anschrift wird in dieser Tabelle vermerkt ob eine Person ein Mitarbeiter, Firmenkontakt oder eine private Kontaktperson ist. Um diese Personen ausschließen zu können werden die Attribute *gwIsEmployee*, *gwIsCompany* und *gwIsContact* benötigt. Weiterhin lässt sich mit der Tabelle feststellen welche Telefonnummern die entsprechende Person besitzt. Sie werden benötigt um die Telefongespräche zuordnen zu können. Die Attribute *PhoneFieldStr1* bis *PhoneFieldStr10* werden zur Aufbewahrung dieser Telefonnummern verwendet.

Die Tabelle *TableRelation* realisiert, wie in Abschnitt 3.1.3 behandelt, die M:N-Beziehungen zwischen allen Tabellen die CRM-Objekte repräsentieren. Sie wird benötigt um die CRM-Objekte einer Person zu ermitteln. Beispielsweise könnten alle von der Person erstellten Termine mit der *GGUID* der Person ermittelt werden. Dafür müssen alle Tupeln ermittelt werden dessen *GUID1* mit der *GGUID* der Person übereinstimmen. Infolgedessen sind mithilfe der Spalte *GUID2* alle Fremdschlüssel der von der Person erstellten CRM-Objekte ersichtlich. Um nur die Termine zu erhalten sind ausschließlich Tupeln mit dem Kürzel "APP" in der Spalte *TableSign2* zu berücksichtigen.

Alle Informationen zu den Verkaufschancen finden sich in der Tabelle *GWOportunity0*. Das Attribut *InsertTimestamp* wird zum Ermitteln des Erzeugungszeitpunktes benötigt. Die Attribute *start\_dt* und *end\_dt* geben den Zeitraum der Verkaufschance fest. Der betroffene Kunde der Verkaufschance kann über das Attribut *AccountGUID* ermittelt werden. Die Tabellen *EmailStore0*, *Document*, *Appointment0* und *gwPhoneCall0* enthalten die Informationen der anderen vier CRM-Objekte. Eine Besonderheit in der Tabelle *EmailStore0* ist die Verwendung von *SendDate* zur zeitlichen Einordnung einer E-Mail. Die Tabelle *gwPhoneCall0* weist eine andere Besonderheit auf. Sie besitzt ein Attribut namens *DialedNumber*, welches zur Ermittlung des Gesprächspartners verwendet wird.

Die Bewertung findet innerhalb von Zeiträumen statt. Dadurch müssen die Veränderungen von Daten über die Zeit hinweg beachtet werden. In der Datenbank werden sämtliche Aktualisierungen der Datensätzen in der Tabelle *ChangeLogBook* aufbewahrt. Sobald ein Feld in der Datenbank überschrieben wurde, wird eine neue Tupel in der *ChangeLogBook* angelegt. In der Spalte *UpdateTimestamp* wird der Zeitpunkt der Änderung erfasst. Die

Spalte *NewFieldValue* enthält den neuen Wert des geänderten Datensatzes. In der Spalte *OldFieldValue* wird der alte Wert abgelegt. Mithilfe der Spalte *TableName* ist nachvollziehbar in welcher Tabelle sich das geänderte Feld befindet und in *FieldName* ist die Bezeichnung der betroffenen Spalte festgehalten. Um die Zeile in der die Änderung stattfand festzuhalten, wird die *GGUID* der Tupel in der Spalte *TableGUID* aufbewahrt.

Einige Aktualisierungen in der Datenbank sind so umfangreich, dass die Änderungen die maximale Zeichenlänge des *NewFieldValue* oder *OldFieldValue* Attributs überschreitet. Damit diese Änderungen nicht verloren gehen wird die Tabelle *MemoLogBook* eingesetzt. Sie besitzt ein Attribut namens *MemoFieldFeld* vom Datentyp Text. Er ermöglicht es Zeichenfolgen in einer maximalen Länge von 536.870.911 Zeichen zu speichern. Dadurch können die Informationen abgespeichert werden, die zu groß für die Varchar-Felder aus der Tabelle *ChangeLogBook* sind.

### 3.4 Charakteristika der Datenbankabfrage

Nachdem klar geworden ist was die Anforderungen sind und mit welchen Daten gearbeitet wird, können potenzielle Charakteristika der Datenbankabfrage ermittelt werden. Die Erkenntnisse bieten eine Hilfestellung bei der Auswahl einer Datenbank.

Der Nutzer selbst führt keine Aktualisierung der Datensätze durch. Das bedeutet es werden auch nahezu keine Schreibvorgänge stattfinden. Die wenigen Schreibvorgänge werden durch den CAS genesisWorld Anwendungsserver angestoßen. Somit ist die erste Erkenntnis, dass hauptsächlich Lesevorgänge stattfinden und die Datenbank dementsprechend darauf ausgelegt sein sollte. Weiterhin müssen laut den Anforderungen Summen gebildet werden. Datenbanken sollten daher mit Datentypen arbeiten und Berechnung über die Abfragesprache durchführen können. Da CRM-Objekte unterschiedliche gewichtet werden, muss die Datenbank auch Produkte bilden können. Weiterhin besitzt der Nutzer viele Möglichkeiten das Ergebnis einer Abfrage zu filtern, wodurch eine Abfrage viele Bedienungen enthalten kann.

## 4. Bestimmung einer Datenbank

Ein Ziel der Arbeit ist es kurze Antwortzeiten in der Beantwortung von Benutzeranfragen zu erreichen. Die maßgebende Komponente in diesem Fall ist die Datenbank. Sie führt die zeitintensiven Berechnungen des Gesamtsystems durch. Um Anhaltspunkte für mögliche Kandidaten zu bekommen, sollen im Folgenden eine Reihe bekannter Datenbanken vorgestellt und gegenübergestellt werden. Bei der Zusammenstellung wurde darauf geachtet, dass ein möglichst weites Spektrum unterschiedlicher Datenbanksysteme ausgewählt wurde.

### 4.1 Einzelbetrachtung von Datenbanken

Im Folgenden werden nun einige Datenbanken vorgestellt. Dabei wird insbesondere versucht einen guten Überblick über die Charakteristika der einzelnen Datenbanken zu geben. Der dahinter stehende Gedanken ist, dass der Vergleich und die Auswahl von einer Datenbank dadurch besser nachvollziehbar werden.

#### 4.1.1 CouchDB

CouchDB [Cou13] ist eine dokumentorientierte Datenbank, die seit Anfang 2008 unter der Apache-Lizenz verbreitet wird. In CouchDB werden die Daten in Collections anstatt in Tabellen abgelegt. Collections bestehen aus einer Sammlung von unabhängigen Dokumenten. Jedes Dokument verwaltet seine eigenen Daten in einem freien Schema. Ein Dokument in CouchDB kann Werte mit festen Datentypen (Text, numerisch oder boolean) oder Datenstrukturen (ein Dokument oder Liste) besitzen. In CouchDB werden für Indizes B-Bäume verwendet, sodass die Ergebnisse sortiert aufbewahrt und Wertebereich-abfragen ausgeführt werden können. Abfragen können parallel über mehrere Rechner mit einem MapReduce Mechanismus verteilt ausgeführt werden. CouchDB erreicht Skalierbarkeit durch asynchrone Replikation<sup>1</sup> und nicht durch eine Fragmentierung der Daten. Lesezugriffe können auf beliebigen Server stattfinden, wenn Aktualität keine Rolle spielt. Updates hingegen müssen an alle Server weitergegeben werden.

CouchDB kann wie andere NoSQL-Datenbanksysteme eine Eventual-Consistency der Daten bieten. Es implementiert MVCC auf einzelne Dokumente, mithilfe einer Sequenz-ID, die für jede Version eines Dokuments generiert wird. Eine Anwendung wird von CouchDB

---

<sup>1</sup>Wenn zwischen der Bearbeitung der Daten und der Replizierung eine Latenz liegt, spricht man von Asynchronität. Die Daten sind nur zu dem Zeitpunkt der Replikation synchron.

benachrichtigt wenn jemand anderes das Dokument aktualisiert hat, seitdem es zuletzt auf der Datenbank abgelegt wurde. Die Anwendung kann dann versuchen die Updates zu kombinieren (merge) oder das Update zu wiederholen, um die Daten zu überschreiben. CouchDB erfüllt außerdem im lokalen Einsatz die ACID-Eigenschaften, da jede Transaktion eine in sich abgeschlossene Operation, die entweder ganz oder gar nicht ausgeführt wird ist. Es treten keine Seiteneffekte zwischen den Anfragen auf.

#### 4.1.2 MongoDB

MongoDB ist ein in C++ geschriebener, Open Source Document Store [CD10]. Es besitzt einige Ähnlichkeiten mit CouchDB. Beide bieten Indizes auf Collections und arbeiten mit einer optimistischen Synchronisation.

MongoDB speichert Daten in einem JSON-ähnlichen, binären Format namens BSON. BSON unterstützt boolean, integer, float, Datum, String- und Binär-Typen. Die Treiber der Clients wandeln die lokalen Dokumentdatenstrukturen in das BSON Format und senden diese an den MongoDB-Server. Weiterhin unterstützt MongoDB die GridFS-Spezifikation für große binär Dateien, wie z.B. Filme oder Bilder. MongoDB unterstützt Master-Slave-Replikation mit automatischem Failover und Recovery. Replikation (und Wiederherstellung) basieren auf dem Prinzip der Fragmentierung. Collections werden dabei automatisch fragmentiert. Die Replikation ist asynchron umgesetzt um höhere Leistung zu erzielen, jedoch können Updates dadurch bei einem Crash verloren gehen.

#### 4.1.3 Voldemort

Projekt Voldemort [Vol13a] ist eine verteilte Key-Value-Store Datenbank (entwickelt von LinkedIn), welches ein hoch skalierbares Speicher-System zur Verfügung stellt. Voldemort repliziert sich durch automatisches Partitionieren und anschließendes Verteilen der Daten auf mehrere Server. Jeder Server stellt einen unabhängigen Knoten im System dar, der für die Verwaltung seiner Daten verantwortlich ist. Dadurch existiert kein Single Point of Failure im Cluster. Ein solches Daten Model erlaubt eine Cluster Expansion, ohne eine Neuverteilung der Daten vornehmen zu müssen. In Voldemort können verschiedene Storage Systeme, wie BerkeleyDB oder MySQL eingesetzt werden.

Für die Ablage der Daten werden in Voldemort sogenannte Stores verwendet. Unterstützt werden lediglich Key-Value Ablagen. Allerdings können die Werte auch komplexe Datenstrukturen wie Maps oder Listen beinhalten. Voldemort stellt für die Datenmanipulation vier verschiedene Operatoren zur Verfügung:

- PUT (Key, Value)
- GET (Key)
- MULTI-GET (Keys)
- DELETE (Key, Version)

Eine Möglichkeit für Bereichsabfragen ist nicht vorhanden. Zur Gewährleistung der Eventual-Consistency werden Timestamps und Vektoruhren eingesetzt. Neben der Eventual-Consistency bietet Voldemort einen Betrieb mit starker Konsistenz an.

#### 4.1.4 Redis

Redis [Seg13] ist ein In-Memory-, Key-Value-Store mit einer Option für Persistenz. Redis Datenmodell unterstützt Strings, Hashes, Listen, Mengen und sortierte Mengen. Obwohl Redis für In-Memory-Daten entworfen wurde, kann je nach Anwendungsfall ein (semi-)

persistenter Bestand angelegt werden. Dieser wird durch die regelmäßige Ablage des Datenbestandes auf die Festplatte erreicht. Überdies werden durch Aufzeichnen eines Logs alle ausgeführten Operationen protokolliert, damit der Originalzustand bei Bedarf wiederhergestellt werden kann. Weiterhin kann Redis mit einer Master-Slave-Architektur repliziert werden. Genau wie andere Key-Value-Stores implementiert Redis insert, delete und lookup Operatoren. Weiterhin setzt Redis atomare Updates durch den Einsatz von Sperren um, die einen exklusiven Zugriff eines Prozesses auf eine Ressource realisieren.

#### 4.1.5 HBase

HBase ist eine verteiltes, Open Source Wide Column Store Datenbanksystem, welches auf Googles BigTable basiert [CDG<sup>+</sup>06]. HBase verwendet Apache Hadoop und Apache ZooKeeper [HKJR10] und das Hadoop Distributed Filesystem (HDFS) [SKRC10], um Skalierbarkeit und Replikation zu bieten. Zeilenoperationen sind in HBase atomar, mit Sperren auf Zeilenebene und Transaktionen. HBase wurde entwickelt um hohe Leistung für intensive Leseaufgaben zu erzielen.

Insbesondere stellt es lineare und modulare Skalierbarkeit, sowie streng konsistenten Datenzugriff und automatische, konfigurierbare Fragmentierung von Daten zu Verfügung. Auf Tabellen kann in HBase über eine Java-, Avro<sup>2</sup>- oder Thrift-API zugegriffen werden. In Abbildung 4.1 ist ein Beispiel zur Art der Datenhaltung in HBase dargestellt. Anwendungen speichern in HBase Daten in Tabellen, die aus Zeilen und Spalten-Familien bestehen. Spalten-Familien beinhalten wiederum Spalten. Darüber hinaus kann jede Zeile einen anderen Satz von Spalten beinhalten. Alle Spalten sind mit einem vom Benutzer bereitgestellten Schlüssel Spalte indiziert und in Spalten-Familien gruppiert.

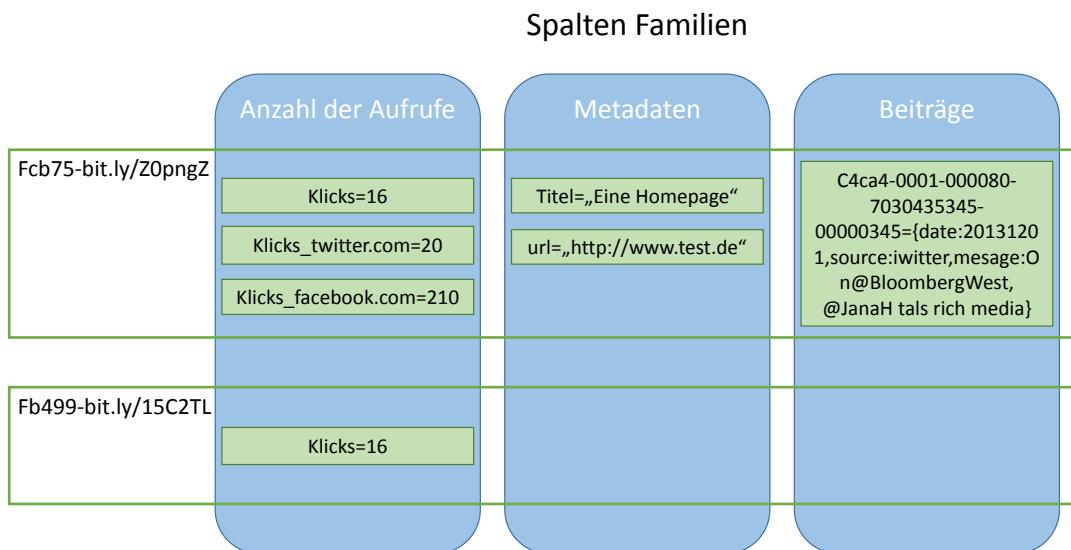


Abbildung 4.1: Komponenten des Systems und der Umwelt

#### 4.1.6 Cassandra

Apache Cassandra ist ein Wide Column Store der von Facebook entwickelt wurde [LM10]. Es ist eine Mischung aus Amazon Dynamo und Google BigTable, wodurch es des öfteren

<sup>2</sup>Avro ist ein Remote Procedure Call- und Serialisierungs-Framework, das als Teil von Apaches Hadoop-Projekt entwickelt worden ist.

als Hybrid zwischen Key-Value-Store und Column Store bezeichnet wird. Der Verweis auf Amazon Dynamo beruht auf der Verwendung dessen Replikationsmechanismus der leicht weiterentwickelt wurde. Außerdem nutzt Cassandra die Datenstruktur von BigTable. Cassandra wurde entwickelt, um große Daten-Workloads über mehrere Knoten, ohne Single Point of Failure zu behandeln. Die Architektur ist von der Annahme geprägt, dass System- und Hardware-Fehler jederzeit auftreten können. Cassandra behandelt das Problem von Fehlern durch Verwendung eines Peer-to-Peer-Systems, in dem alle Knoten gleichberechtigt sind und die Daten über alle Knoten des Clusters verteilt werden. Jeder Knoten tauscht Informationen über das Cluster im Sekundentakt aus. In den Commit-Log werden zuallererst die Schreibvorgänge protokolliert bevor die Daten auf eine In-Memory Struktur (memtable) geschrieben werden. Sobald die Speicherstruktur voll ist werden die Daten in eine Datei (SSTable) auf der Festplatte abgelegt. Die durch Schreibvorgänge veränderten Daten werden automatisch aufgeteilt und auf mehrere Cluster repliziert. Weiterhin wurde Cassandra entwickelt um hohe Leistungen in schreibintensiven Aufgaben zu erzielen.

Cassandras Datenmodell basiert auf einem partitionierten Row-Store mit Eventual-Consistency. Zeilen werden in Tabellen organisiert, wobei die erste Komponente des Primärschlüssels einer Tabelle der Partitionschlüssel ist. Andere Spalten können getrennt vom Primärschlüssel indiziert werden. Was Cassandra von HBase unterscheidet sind ihre Spalten, die in einer verschachtelten Weise in Spalten-Familien gruppiert werden können. In der Abbildung 4.2 sind die Verschachtlungen dargestellt.

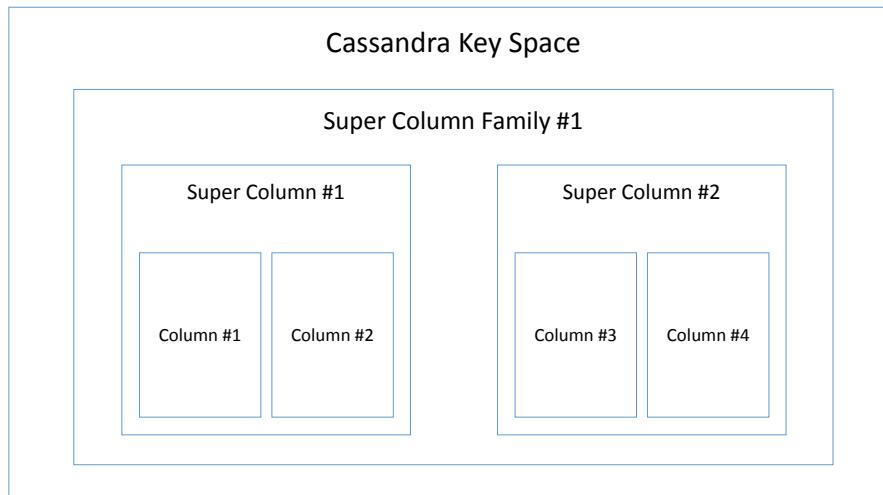


Abbildung 4.2: Schematische Darstellung des Cassandra Datenmodells

#### 4.1.7 VoltDB

VoltDB [Vol13c] ist ein ACID-konformes, relationales In-Memory-Datenbanksystem, abgeleitet vom Forschungsprototyp H-Store [KKN<sup>+</sup>08]. Da VoltDB auf dem Ansatz der relationalen Algebra beruht und auf verteilten Knoten betrieben werden kann, zählt es zu den NewSQL-Datenbanken. Es basiert auf einer Shared-Nothing-Architektur und wurde entwickelt, um auf einem Cluster mit mehreren Knoten zu laufen. Erreicht wird dies indem die Datenbank in getrennte Partitionen aufgeteilt wird, bei dem jeder Knoten Besitzer und Verantwortlicher für die jeweiligen Partitionen ist. Die Anfragen in VoltDB werden seriell in einem einzigen Thread ausgeführt, sodass keine Sperren und Riegel innerhalb der Transaktionen mehr notwendig sind [Vol13b]. Die Daten werden im Arbeitsspeicher gehalten, was eine Ausführung ohne Netzwerkzugriff und I/O-Vorgänge ermöglicht, falls die Daten nur auf einem Knoten liegen.

#### 4.1.8 H2

H2 ist ein in Java geschriebenes relationales Datenbanksystem, dass im Jahre 2004 von Thomas Müller veröffentlicht wurde. Es wird unter der Eclipse Public License verbreitet und ist damit Open Source. H2 bietet neben den festplattenbasierten Tabellen, auch eine In-Memory Variante an. Tabellen können dabei dauerhaft oder temporär sein. Weiterhin beherrscht H2 referentielle Integrität, Transaktionen, Clustering, Datenkompression, Verschlüsselung und SSL [Mü13]. Die Datenbank kann im Embedded- oder Server-Modus betrieben werden.

#### 4.1.9 Neo4J

Neo4j ist eine in Java implementierte Open-Source-Graphdatenbank. Neo4j wird des öfteren als eine eingebettete, Disk-basierte, ACID-transaktionale Datenbank-Engine bezeichnet [RWE13]. Es speichert strukturiert Daten in Graphen anstatt in Tabellen. Für den Zugriff auf die Daten bietet Neo4J die Abfragesprache Cypher an. Cypher ist eine deklarative Graphabfragesprache, die es erlaubt ausdrucksstarke und effiziente Abfragen und Aktualisierung des Graphen vorzunehmen. Überdies unterstützt Neo4J noch andere Zugriffskonzepte, die im Abschnitt 4.2 erwähnt werden.

Im Neo4J Datenmodell wird alles durch Knoten, Beziehungen und Eigenschaften umgesetzt. Eine Beziehung verbindet zwei Knoten, hat einen klar definierten, verbindlichen Typ und kann wahlweise gerichtet sein. Eigenschaften sind Schlüsselwertpaare, die an Knoten und Beziehungen gebunden sind. In Abbildung 4.3 ist ein Beispiel modelliert.

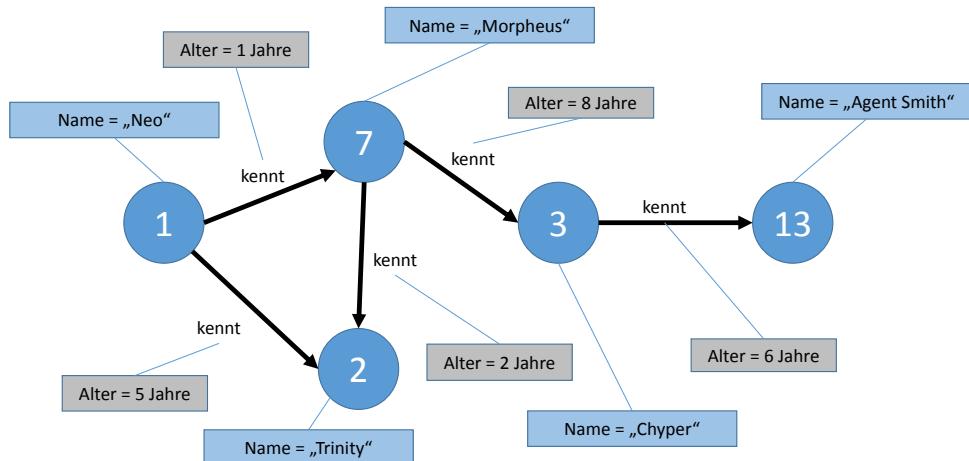


Abbildung 4.3: Datenmodell anhand eines Social-Network-Beispiels

In der Abbildung besitzt jeder Knoten eine ID (Zahl) und jede Beziehung einen Typ. Weiterhin existiert nur ein einziger Typ im Beispiel, mit der Bezeichnung "kennt". Das Beispiel besitzt Knoten mit der Eigenschaft (Name) und die Beziehungen verfügen über Eigenschaften zur Beschreibung wie lange sich die Personen bereits kennen.

Typische Anwendungsbereiche von Neo4J sind beispielsweise Semantic Web, LinkedData, GIS, Genomanalysen und Modellierung von sozialen Netzwerken.

## 4.2 Gegenüberstellung

Zur übersichtlichen Gegenüberstellung der Datenbanken wird die Tabelle 4.1 herangezogen. Sie enthält vergleichbare Eigenschaften von Datenbanken, auf die im Folgenden eingegangen wird.

Die erste Eigenschaft ist das Erscheinungsjahr einer Datenbank, welches bereits Rückschlüsse auf die Reife des Datenbanksystems in verschiedenen Aspekten ermöglicht. Ältere Datenbanken haben bereits viele ihrer anfänglichen Fehler beseitigt, was sie für den Einsatz in produktiven Umgebungen geeigneter machen. Natürlich sind ältere Systeme nicht gänzlich frei von Fehlern, allerdings existieren für viele Probleme entsprechende Lösungsansätze. Cassandra zum Beispiel, erhielt über die Jahre neben zahlreichen Bugfixes, MapReduce Support, sekundäre Indizes, verbesserte Komprimierung, eine eigene Query Sprache (CQL). Es gibt allerdings keine Regel die besagt zu welchem Zeitpunkt ein System die Reife für den produktiven Einsatz erreicht hat. Es spielen natürlich auch andere Faktoren bei der Bestimmung der Ausgereiftheit eine Rolle, wie z.B. die Größe des Unternehmens oder Teams das hinter der Datenbank steht. Die Eigenschaft hat weniger den Zweck eines Kriteriums, sondern eher eines Indikators.

Tabelle 4.1: Gegenüberstellung der Datenbankeigenschaften

Eigen-schaft	HBase	Cassandra	CouchDB	MongoDB	Redis	Voldemort	VoltDB	H2	Neo4j
Release Datum	2008	2008	2005	2009	2009	2009	2010	2004	2007
Daten-bank-modell	Wide Column	Wide Column	Document	Document	Key-Value	Key-Value	Relatio-nal DBMS	Relatio-nal DBMS	Graphdatenbank
Lizenz	Open Source	Open Source	Open Source	Open Source	Open Source	Open Source	Kommer-zialiell	Open Source	Open Source
Server Be-triebs-sys-te-me	Linux, Unix, Windows	BSD, Linux, OS X, Windows	Android, BSD, Linux, OS X, Solaris, Windows	Linux, OS X, Solaris, Windows	BSD, Linux, OS X, Windows	Linux, Unix, Windows	Linux, OS X	platt-formu-nab-hängig	Linux, OS X, Windows
Daten-sche-ma	schemafrei	schemaf-rei	schemafrei	schemafrei	schemafrei	schemaf-rei	ja	ja	schemafrei
Typi-sierung	nein	ja	nein	ja	nein	nein	ja	ja	ja
Sekun-därin-dizes	nein	einge-schränkt	ja (über Views)	ja	nein	nein	ja	ja	ja
SQL	nein	nein	nein	nein	nein	nein	ja	ja	nein
APIs und andere Zu-griffs-konzepte	Java API, RESTful HTTP API, Thrift	Proprietä-res Protokoll (CQL)	RESTful HTTP/JSON API	Proprietäres Protokoll basierend auf JSON	Proprietäres Protokoll	Proprietä-res Protokoll	Java API, RESTful HTTP/J-SON API, JDBC	Java API, ODBC, JDBC	Cypher query language, Java API, RESTful HTTP API
Unter-stützte Pro-grammier-sprach-en	C, C#, C++, Groovy, Java, PHP, Python, Scala	C#, C++, Java, Perl, PHP, Python, Ruby, +9	C, C#, Java, JavaScript, Perl, PHP, PL/SQL, Python, Ruby, +9	C#, C++, Java, JavaScript, Perl, PHP, Python, Ruby, +4	C#, C++, Java, JavaScript, Perl, PHP, Python, Ruby, +12	C#, C++, Java, Perl, PHP, Python, Ruby, +12	C#, C++, Java, PHP, Python	C#, C++, Java, PHP, Python	.Net, Clojure, Go, Groovy, Java, JavaScript, Perl, PHP, Python, Ruby, Scala
Ma-pRedu-ce	ja	ja	ja	ja	nein	nein	nein	nein	nein
Konsi-tenz-konzept	Immediate Consistency	Eventual Consistency, Immediate Consistency	Eventual Consistency	Eventual Consistency, Immediate Consistency	Eventual Consistency	Strict Consistency, Eventual Consistency	Integri-tätsbedin-gungen	Integri-tätsbedin-gungen	Eventual Consistency
Trans-ak-tionskon-zept	nein	nein	nein	nein	optimisti-sches Locking	nein	ACID	ACID	ACID
Neben-läufig-keit	ja	ja	ja	ja	ja	ja	ja	ja	ja
Embed-dable	nein	ja	ja	nein	nein	ja	ja	ja	ja
In Me-mory fähig	nein	nein	nein	nein	ja	hybrid	ja	ja	nein

Ein wichtiger Faktor ist die Lizenz unter der die Datenbank vertrieben wird. Für Unternehmen ist die Wirtschaftlichkeit eines Produktes von immenser Bedeutung. Deshalb bieten Open Source Produkte mit ihren geringen Anschaffungskosten einen hohen Anreiz. Bei der Verwendung von Open Source ist allerdings mit einem schwächeren Support als bei kommerziellen Produkten zu rechnen. Ein weiteres Argument zur Nutzung von Open Source ist die Möglichkeit einen Einblick in den Quelltext zu erhalten und diesen gegebenenfalls auch zu bearbeiten. Kommerzielle Lizenzen bieten hingegen eine höhere Zukunftssicherheit als Open Source Produkte, da Unternehmen in ihrer Arbeit beständiger sind als Privatpersonen.

Die Betrachtung von unterstützten Programmiersprachen und Betriebssysteme ist zum Feststellen der Kompatibilität mit vorhandenen Systemen notwendig. In Unternehmen sollte idealerweise schon Erfahrung in den potenziellen Technologien vorhanden sein. Denn externe Mitarbeiter, sowie Schulungen sind teuer und sollten bei der Wahl einer Technologie oder Produktes berücksichtigt werden.

Datenbanken können ein festes Schema besitzen oder schemafrei sein. Welche der beiden Möglichkeiten für die Lösung einer Problemstellung geeigneter ist kann anhand der Struktur der abzulegenden Daten bestimmt werden. Wenn sich die Struktur der abzulegenden Daten häufig ändert oder keine einheitliche Struktur unter den Daten zu erkennen ist, sind schemafreie Datenbanken von Vorteil. Denn sie bieten ein hohes Maß an Flexibilität, wohingegen durch die Nutzung eines Schemas eine bessere Kontrolle über die Daten entsteht.

Sekundärindizes können Lesegeschwindigkeiten steigern, weshalb sie zum Erreichen von kurzen Antwortzeiten eine interessante Datenbankenfunktion darstellen. Sie erlauben Indizes auf einem oder mehreren Schlüsseln oder Nicht-Schlüsselattributen, wodurch die Effizienz einer Suche gesteigert werden kann. Einige NoSQL-Datenbanken unterstützen solche Indizes, wohingegen relationale Datenbanksysteme die Definition beliebiger Sekundärindizes gestatten.

Die Vermeidung von Laufzeitfehlern ist ein Ziel der Typisierungen. Typisierte Datenbanken schränken den Wertebereich von Variablen ein. Ein Vorteil der dadurch entsteht ist eine Vorabkontrolle der Daten, sodass nur Daten mit den entsprechenden Eigenschaften verwendet werden. Ein Nachteil ist die mangelnde Flexibilität. Der Entwickler muss sich wie beim Schema zwischen Flexibilität und Kontrolle entscheiden.

Das in der Datenbank verwendete Zugriffskonzept spielt bei der Architektur des gesamten Systems eine Rolle. Zu einem ist zu unterscheiden ob es sich um ein proprietäres Protokoll oder ein standardisiertes Protokoll handelt. Proprietäre Protokolle weisen meist eine höhere Einarbeitungszeit für die Mitarbeiter auf. Bei der Arbeit mit Standardtechnologien kann auf vorhandenem Wissen aufgebaut werden, was die Einarbeitungszeit verkürzt.

Eine Entscheidung hinsichtlich der Stärke von Konsistenz wird durch den Anwendungsfall bestimmt. Wenn die Redundanz von Daten kein Problem im Anwendungsfall darstellt ist keine hohe Konsistenz notwendig. Allerdings sollte die nächst höhere Anwendungsschicht bei Inkonsistenz damit rechnen, sonst kann es zu schwerwiegenden Fehlern kommen.

Das Transaktionskonzept (Nebenläufigkeit) gibt an, ob gleichzeitig ausgeführte Datenmanipulationen durch die Datenbank unterstützt werden. Genau wie bei der Konsistenz ist der Anwendungsfall für die Wahl des Transaktionskonzeptes ausschlaggebend.

Datenbanken die im Embedded-Modus betrieben werden können sehr einfach in Anwendungen integriert werden. Ein Vorteil dabei ist das es zu keinen Verzögerungen durch Netzwerkzugriffe kommt.

Die CAS Software AG wünscht sich hohe Ausführungsgeschwindigkeiten, wodurch die In-Memory-Fähigkeit einer Datenbank von entscheidender Bedeutung sein kann.

### 4.3 Auswahl einer Datenbank

Jede Datenbank hat ihre eigenen Stärken und Schwächen. Bei der Wahl einer geeigneten Datenbank ist nicht entscheidend welche Datenbank für allgemeine Aufgaben die optimale ist. Es ist wichtiger, dass die entsprechende Datenbank die Charakteristika die zur Erreichung des Ziels benötigt werden besitzt.

In dieser Arbeit steht eine geringe Antwortzeit als Eigenschaft der Datenbank im Vordergrund. Die Verwendung des Hauptspeichers als Speichersystem bedeutet einen theoretischen Geschwindigkeitsvorteil um circa 50.000 [Pla13b]. Das Speichersystem alleine ist zwar nicht aussagekräftig genug um die Entscheidung nur aus diesem Grund zu treffen. Es wird allerdings angenommen das simple Abfragen, die hauptsächlich Daten lesen, um einiges schneller sind als in festplattenbasierten Datenbanken. Fünf Datenbanken bieten diese Eigenschaft nicht. Deswegen ist zu untersuchen, ob diese Datenbanken andere Charakteristiken aufweisen können, die den Nachteil ausgleichen.

Cassandra und HBase ermöglichen hohe Performance durch horizontale Skalierung. Horizontale Skalierung ist vor allem bei hohem Datenaufkommen sinnvoll. Die Kunden der CAS Software AG sind alles mittelständische Unternehmen, welche nicht an die Nutzerzahlen von Facebook und Google herankommen. Daher sind keine täglichen Zugriffszahlen im Millionenbereich zu erwarten. Horizontale Skalierung ist dementsprechend nicht notwendig, sowie durch die Limitierung auf einen Rechner nicht möglich. Außerdem ist zu erwarten das Cassandra und HBase auf einzelnen Servern nicht an die Performance von In-Memory fähigen Datenbanken herankommen. Aufgrund dessen wurde sich gegen die beiden Vertreter der Wide Column Stores entschieden.

Die Document Store Datenbanken sind zwar auch horizontal skalierbar, jedoch kann wie bereits geschildert dieser Vorteil nicht ausgenutzt werden. Ihre Stärke liegt in ihrer Schema freien Datenhaltung, die an dieser Stelle von geringem Wert ist, da die verwendeten Daten eine feste Struktur besitzen. Außerdem werden Funktion wie *SUM()* nicht in der Datenbank eigenen API mitgeliefert, was sie für analytische Aufgaben bedingt brauchbar macht. Letztendlich können CouchDB und MongoDB keine Argumente liefern, weshalb sie für unser Szenario geeignet sind. Infolgedessen entschied man sich gegen sie.

Die Key-Value-Stores ermöglichen niedrige Zugriffszeiten mit ihrer In-Memory Datenhaltung. Was ihnen zum Nachteil ausgelegt werden kann, ist ihre mangelnde Komplexität. Damit ist gemeint, dass man auf komplexe Suchalgorithmen oder Indizes verzichten muss, da der Zugriff auf einen Datensatz lediglich über dessen Schlüssel erfolgt. Weiterhin sind sie auf Punkt-Abfragen ausgelegt. Komplexe Abfragen können daher nur in der Logikschicht realisiert werden. Diese würde zu einer nicht vorhersehbaren Steigerung im Aufwand bei der Umsetzung der Datenbankzugriffsschicht führen. Viele Funktionen müssten vom Entwickler selbst implementiert werden. Daher wurde sich auch gegen die Key-Value-Stores entschieden.

Neo4J wird bereits in vielen Bereichen eingesetzt und hat sich als robustes Produkt erwie sen. Weiterhin besitzt es Anbindungen an eine Reihe von Sprachen als auch Integrations möglichkeiten in vorhandene Frameworks wie Spring, Ruby on Rails, PHP, etc. Die Open-Source Lizenz ist natürlich auch ein Pluspunkt. Gerade in Anbetracht unseres Szenarios sind Graphen auf den ersten Blick eine sehr nützliche Form um Beziehungen zu ermitteln. Es muss allerdings beachtet werden, dass keine komplexen Verknüpfungen über viele Personen hinweg zu ermittelt sind. Es wären immer nur sehr wenige Kanten zu betrachten. Die Geschwindigkeitsvorteile von Graphdatenbanken lassen sich allerdings nur bei komplexen und weitreichenden Ermittlungen ausnutzen. Die Verwendung der Datenbank ist zwar eine Möglichkeit, jedoch kann sie ihre Stärken in dem für die Arbeit betrachtetem Szenario nicht richtig ausnutzen. Überdies sind kurze Antwortzeiten sehr wichtig und Neo4J bietet

keine Funktionen zum Betrieb im Hauptspeicher. Es wäre lediglich mithilfe der Manipulation des Caches möglich einen In-Memory artigen Betrieb der Datenbank zu erreichen. Welche Seiteneffekt dabei auftreten ist nicht klar sowie die Probleme die dadurch entstehen könnten. Aufgrund dessen das keine komplexen oder weitreichenden Verknüpfungen zwischen vielen Personen betrachtet werden und sie Nachteile in der Latenz gegenüber In-Memory-Datenbanken haben, konnte Neo4J nicht überzeugen.

VoltDB ist von den Eigenschaften her ein optimaler Kandidat, allerdings nicht Open Source. Aufgrund dessen wurde sich gegen VoltDB entschieden.

Die H2-Datenbank hingegen ist Open Source und bietet Optionen zum Vorhalten der Daten im Hauptspeicher. Davon werden sich hohe Geschwindigkeitsvorteile gegenüber herkömmlichen relationalen Datenbanksystemen erhofft. Durch den Ansatz der relationalen Algebra in der Datenbank ist die Arbeit mit SQL möglich. Das birgt Vorteile, da auf bereits bekanntem Wissen aufgebaut werden kann. Die H2-Datenbank wurde ausgewählt, da sie mit der Kombination von neuen Ansätzen wie der Nutzung des Hauptspeichers und alt bewährten Prinzipien aus der relationalen Algebra eine schlüssige Lösung ergibt.



## 5. Konzeption

Ein Konzept dient in der Softwarearchitektur der Konstruktion eines abstrakten Systemmodells. Zur Gestaltung werden technische Details weggelassen und stattdessen allgemeingültige Begriffe und ihre Zusammenhänge festgelegt. Weiterhin wird ein Grundverständnis durch Definieren von Strukturen und Konzepten gebildet. Darüber hinaus werden Schnittstellen definiert, die Wechselwirkungen zwischen den Komponenten beschreiben. Weiterhin werden im Zuge der Überlegungen Technologien ausgewählt, die zur Umsetzung der verschiedenen Komponenten verwendet werden.

### 5.1 Architektur

Zuerst wird ein erster Überblick über den geplanten Aufbau des Systems gegeben. Abbildung 5.1 zeigt die Komponenten des Systems und der Umwelt. Der Tomcat mit seinen Webcontainern bildet die Basis des Systems. Wie auf der Abbildung zu sehen sind zwei verschiedene Projekte vorgesehen. Zum einen ein Client-Webprojekt und zum anderen ein Server-Webprojekt. Das Client-Webprojekt soll die Klassen und Methoden zur Umsetzung der Darstellung beinhalten. Die Anwendungslogik sowie die Datenbank sind im Server-Webprojekt vorgesehen. Der Grund für die Aufteilung in zwei verschiedene Webprojekte ist die Anforderung der losen Kopplung zwischen Client und Server. Beide Webprojekte werden in Form von Web-Archive-Dateien in einem Apache-Tomcat-Websserver deployed und können über die entsprechende URL angesprochen werden. Weiterhin soll mithilfe eines selbstgeschriebenen Plugins für den CAS genesisWorld Anwendungsserver die Synchronisation zwischen den Systemen umgesetzt werden. Der Browser, CAS genesisWorld und der MSSQL-Server stellen die Umwelt des Systems dar. Im Folgenden wird auf jede Komponente der Architektur eingegangen.

**Vaadin Client-Side-Engine** Vaadin ist ein auf Java basiertes Webanwendungs-Framework [Gro14]. Es bietet eine serverseitige Architektur, wodurch ein Großteil der Programmlogik auf dem Server läuft. Auf der Clientseite baut Vaadin auf dem Ajax-Framework Google Web Toolkit auf und stellt dazu die Client-Side-Engine bereit. Sie verwaltet das Rendering der Oberfläche im Web-Browser. Dies geschieht durch den Einsatz verschiedener clientseitiger Widgets, die das Gegenstück zu den serverseitigen Vaadin-Komponenten bilden. Es leitet Benutzerinteraktionen an die Serverseite weiter und rendert anschließend die Änderungen für die Benutzeroberfläche. Die Kommunikation findet über asynchrone HTTP-oder HTTPS-Anfragen statt. Weiterhin wird die Komponente durch Vaadin automatisch erzeugt und wird daher für die Umsetzung als gegeben betrachtet.

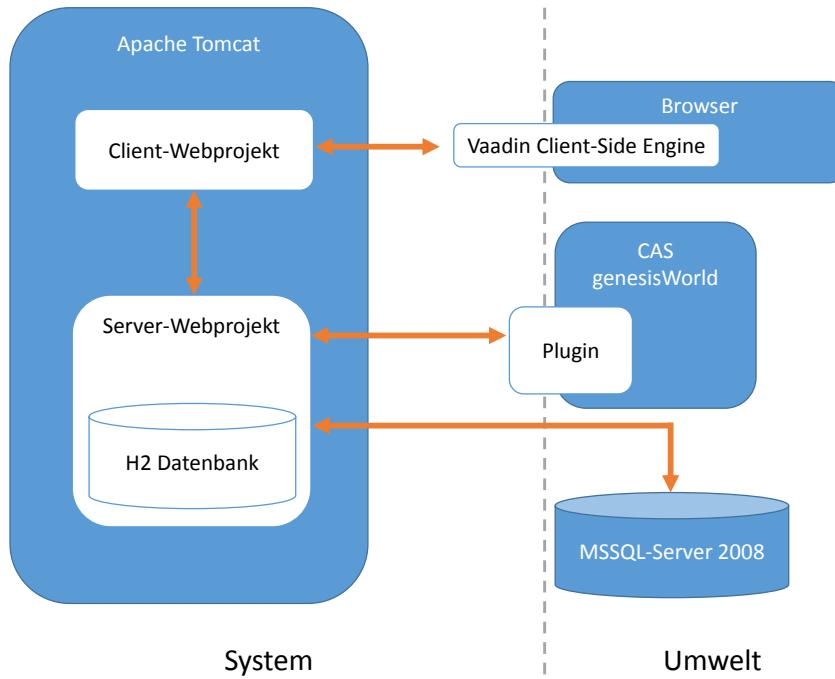


Abbildung 5.1: Komponenten des Systems und der Umwelt

**Client-Webprojekt** Die Oberfläche des Systems wird durch ein Vaadin-Projekt realisiert. Mit dessen Hilfe werden die Bedien- und Darstellungselemente der Anwendung definiert. Sie ist außerdem für die Interaktion mit dem Benutzer zuständig. Die Delegation von verschiedenen Clients beispielsweise wird von einem Vaadin-Servlet übernommen. Dazu zählt das Empfangen von Anfragen und deren Zuordnung zu einer Sitzung des jeweiligen Benutzers. Die Elemente der Oberfläche selbst werden in Java implementiert. Mithilfe der Java-Klassen wird zur Laufzeit eine Javascript basierte Homepage erzeugt. Die Übersetzung von Java auf Javascript übernimmt Vaadin.

Das Client-Webprojekt wird im ständigen Informationsaustausch mit dem Server-Webprojekt stehen. Die Kommunikation zwischen den beiden Komponenten soll über das REST-Protokoll stattfinden. Dazu implementiert das Client-Webprojekt einen REST-Client. Die Verwendung des REST-Protokolls zwischen dem Client-Webprojekt und Server-Webprojekt stellt überdies ein weiteres Element der losen Kopplung dar.

Ein Prozess in dem die einzelnen Bestandteile Verwendung finden, kann wie folgt aussehen: Die Interaktionen der Benutzer mit der Oberfläche wird Events erzeugen, die zunächst auf der Clientseite durch Widgets verarbeitet werden. Nachfolgend werden die Events durch den HTTP-Server an das Vaadin-Servlet übergeben. Dieser leitet die Events an die entsprechenden Vaadin-Objekte weiter, bis sie zu den in der Anwendung definierten Event-Listenern gelangen. In den Listenern werden anschließend die REST-Clients aufgerufen. Mit Hilfe der REST-Clients werden die Eingaben der Nutzer an das Server-Webprojekt übermittelt.

**Server-Webprojekt** Die eigentliche Lösung der Problemstellung soll im Server-Webprojekt des Softwaresystems implementiert werden. Es soll vollständig auf Java basieren. In ihr werden sich Klassen und Methoden befinden die eine Ermittlung der Informationen aus der H2-Datenbank ermöglichen. Weiterhin soll ein REST-Server implementiert werden, der die Benutzereingaben entgegennimmt. Der REST-Server ist außerdem für die Kommunikation

mit dem Plugin zuständig.

Weiterhin wird das Projekt die ETL-Prozessschritte zum Überführen der Daten aus der MSSQL-Datenbank in die H2-Datenbank implementieren. Die genauen Prozessschritte werden in Abschnitt 5.3 behandelt.

**H2-Datenbank** Die H2-Datenbank wird als ein Bestandteil des Server-Webprojektes eingesetzt. Dies wird durch den Betrieb der H2-Datenbank im Embedded-Modus erreicht. Dadurch werden Verzögerungen vermieden, die bei der Kommunikationen über ein Netzwerk entstehen können. Um möglichst kurze Antwortzeiten zu erreichen wird die In-Memory-Variante der Tabellen verwendet.

**CAS genesisWorld Plugin** Systeme die auf dem Datenbestand anderer Systeme aufbauen können zwei verschiedene Ansätze zur Sicherstellung ihrer Aktualität verfolgen. Unser nebenläufiges System bezeichnen wir als A und den CAS genesisWorld Anwendungsserver als B. Einer der Ansätze ist die Intervall basierte Nachfrage über Veränderungen von A. Hierbei fragt A bei B zu festgelegten Zeitpunkten nach, ob Daten verändert wurden. Die Definition eines optimalen Intervalls stellt eine der größten Schwierigkeiten dar. Ist der Intervall zu groß, sinkt die Aktualität des Datenbestandes. Ist er zu klein, entsteht eine starke Belastung für B. Der andere Ansatz ist A über Veränderungen an den Datensätzen von B zu informieren. Dadurch werden keine unnötigen Abläufe angestoßen, da nur im Falle einer Manipulation eines Datensatzes Prozesse in Bewegung gesetzt werden. Der zweite Ansatz ist zwar effizienter, allerdings nicht immer umsetzbar. Das kann technische oder unternehmenspolitische Gründe haben, die notwendige Veränderung am Legacy-System ausschließen.

In CAS genesisWorld gibt es die Möglichkeit den zweiten Ansatz umzusetzen. Die Idee dabei ist den CAS genesisWorld Anwendungsserver um ein Plugin zu erweitern, welches über Veränderungen in den Datensätzen benachrichtigt wird. Das Plugin soll über einen REST-Client die *GGUID* des betroffenen Datensatzes an das Server-Webprojekt senden. Dort soll eine Kontrolle stattfinden, die den Datensatz auf Relevanz überprüft. Wird eine Relevanz festgestellt besorgt sich das Server-Webprojekt, anhand der zuvor übermittelten GGUID, alle benötigten Daten.

## 5.2 Datenbankdesign

Das Datenbankdesign stellt einen wichtigen Abschnitt in der Konzeption dar. An dieser Stelle werden Festlegungen im Bereich des Datenmodells getroffen. Sie entscheiden, ob Anforderungen und Erwartungen erfüllt werden können. Weiterhin werden die Charakteristika der aufzubewahrenden Daten untersucht und das Datenmodell entsprechend nach ihnen gestaltet.

### 5.2.1 Konzeptionelles Modell

Zunächst wird mithilfe eines konzeptionellen Modells eine Übersicht der zu verwaltenden Daten aufgezeigt. Das Modell bietet durch einen hohen Abstraktionsgrad einen idealen Einstieg in das geplante Vorhaben. In Abbildung 5.2 ist das Modell in Form eines ERD-Diagramms dargestellt.

Im Mittelpunkt stehen in diesem Fall die Personen. Diese besitzen jeweils eine eigene Adresse mit ihren Kontaktdataen. Personen können außerdem zu Gruppen gehören, müssen aber nicht zwangsläufig einer zugeordnet werden. Neben der Adresse erzeugen Personen ihre eigenen CRM-Objekte. Diese können sie mit anderen Personen teilen, wodurch ein

CRM-Objekt zu mehreren Personen in Beziehung treten würde. CRM-Objekte wiederum können in verschiedenen Ausprägungen z.B. als E-Mail oder Dokument auftreten. Die Ausprägungen besitzen verschiedene Attribute welche es ermöglichen das CRM-Objekt einem Zeitpunkt zuzuordnen.

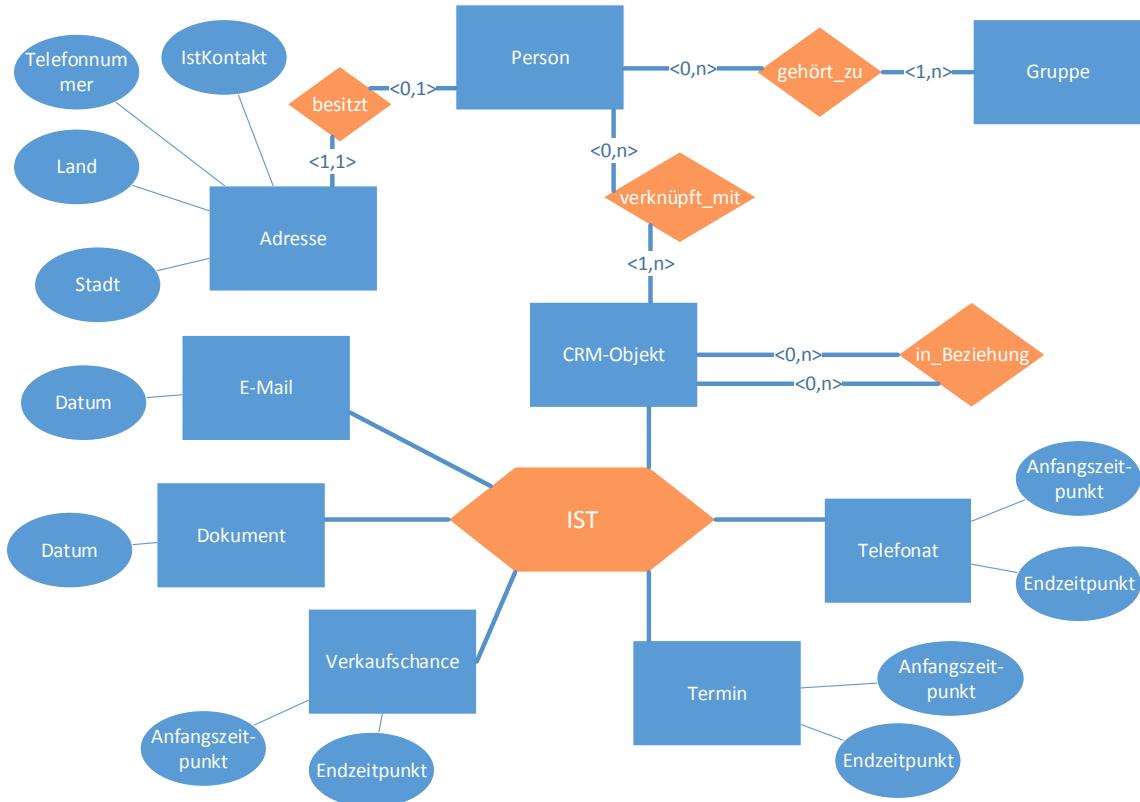


Abbildung 5.2: ER-Modell als Darstellungsform des Konzeptionelles Modell

### 5.2.2 Datenbankschema

Zunächst wird auf das geplante Schema eingegangen welches in der H2-Datenbank eingesetzt wird. Indessen werden die Überlegungen und Entscheidungen die zur Entstehung des Schemas geführt haben erläutert.

Zuerst werden Überlegungen hinsichtlich der Normalisierung angestellt. Die Normalisierung bietet dem Designer die Möglichkeit einen Austausch zwischen Performance und Stabilität vorzunehmen [Gei11]. Im neuen System stellt ersteres absolute Priorität dar. Aus diesem Grund hat eine Normalisierung keine besonders hohe Priorität.

Abbildung 5.3 zeigt das für die Datenbank neu entworfene Schema.

Die zugrundeliegende Idee ist alle Informationen über die Beziehung zwischen den Personen und den CRM-Objekten in der *Data*-Tabelle aufzubewahren. Dadurch liegt ein wesentlich geringer normalisiertes Schema vor als in der MSSQL-Datenbank. Jede Tupel repräsentiert dabei ein CRM-Objekt, das zwei Personen in Beziehung zueinander setzt. Außerdem soll jede Tupel einem Tag zugeordnet werden.

Die erste und vierte Spalte *startID* beinhalten jeweils die ID der Personen. Eine Zuordnung der Tupel zu einem Datum erfolgt über die Spalte *Date*. Um CRM-Objekte zu unterscheiden, werden Zahlen von eins bis fünf in der Spalte *DataTyp* für die jeweiligen Ausprägungen des CRM-Objektes vergeben. Alle anderen Spalten, wie *Town* oder *Country* beinhalten Werte, anhand denen die Ergebnismenge verringert werden kann.

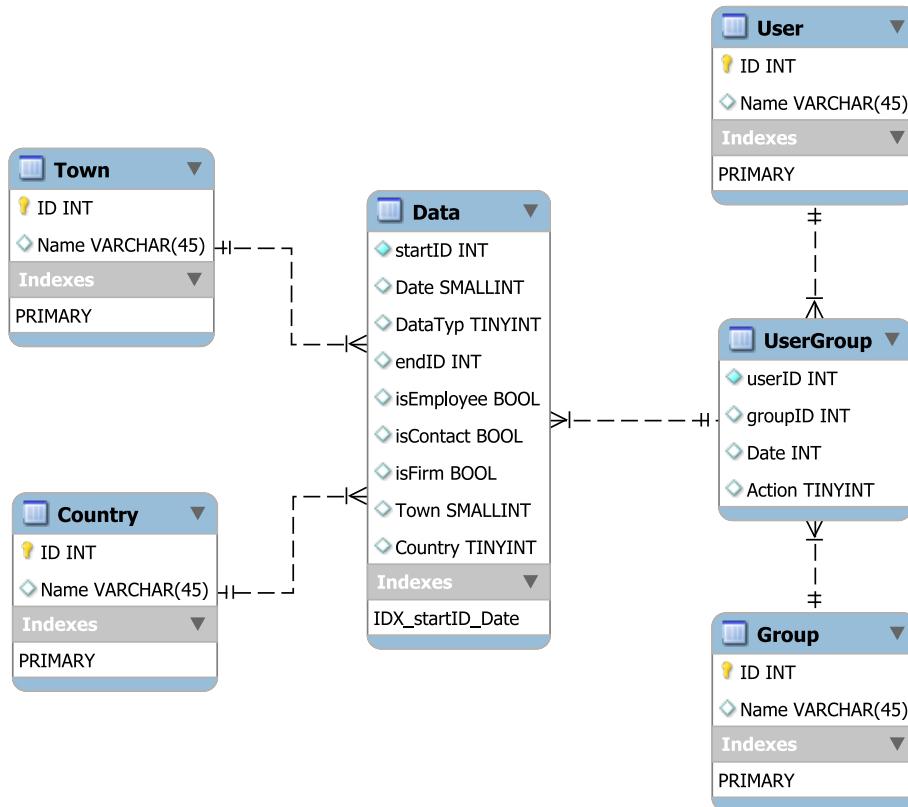


Abbildung 5.3: Neues Datenbankschema

Um mit den geringeren Kapazitäten des Hauptspeichers zurechtzukommen, wird auf das Problem der Datenredundanz eingegangen. Durch Normalisierung lässt sich Datenredundanz zwar verringern, allerdings steigt dadurch der Aufwand zur Wiedergewinnung der Daten. Im neuen Schema werden solche Maßnahmen auf die Spalte *Town* und *Country* angewendet.

Die Tabelle *Data* wird voraussichtlich Millionen von Tupeln besitzen, weshalb Möglichkeiten in der Normalisierung in Betracht gezogen werden. Beispielsweise werden sich die Ländernamen sehr oft wiederholen. Der Datentyp Varchar benötigt pro Zeichen 2 Byte an Speicherplatz. Aufgrund der stetigen Wiederholung von gleichen Wörtern ist die Verwendung von Varchar an dieser Stelle ungeeignet. Angesichts dessen sollen die Ländernamen in einer eigenen Tabelle aufbewahrt werden.

Die Normalisierung würde zum Beispiel bei dem Wort "Deutschland" eine Reduktion von 22 Byte auf 1 Byte bewirken. Die Reduzierung auf 1 Byte entsteht durch die Verwendung des Datentyps tinyint. Diese Schritte zur Normalisierung werden ebenfalls für die Spalte *Town* angewandt. Bei ihr wird allerdings der Datentyp smallint verwendet, da dessen Zahlenbereich von -32768 bis 32767 reicht. Damit lassen sich alle Städte aus der MSSQL-Datenbank abdecken.

Die Spalten *isEmployee*, *isContact* und *isFirm* können nur zwei verschiedene Zustände darstellen. Trifft zu oder trifft nicht zu. Der Datentyp bool ist daher ausreichend zur Abbildung der Zustände. Ein Feld vom Datentyp datetime benötigt 8 byte an Speicher. Um hier ebenfalls Einsparungen vorzunehmen, wurde beschlossen das Datum als smallint zu deklarieren. Dies ist möglich, weil nur der Tag des Datums von Interesse ist. Dazu wird ein frei gewählter Zeitpunkt Null festgelegt. In unserem Fall wurde der 01.01.1990 als Zeitpunkt gewählt, da keine älteren Daten existieren, die eine Relevanz besitzen. Der Wert eines Datums wird durch die Anzahl der Tage seit dem Zeitpunkt Null ermittelt. Ein Bei-

spiel wäre der 05.01.1990 der in der Spalte *Date* als eine 4 vermerkt werden würde. Die Hochrechnung der Tabelle 5.1 zeigt, dass durch die Normalisierung der Speicherplatzverbrauch auf  $\frac{1}{6}$  gesenkt werden kann.

#### Speicherplatzverbrauch ohne Normalisierung

Zeitpunkt(timestamp)	8 byte	x	18.000.000	=	$\sim 137$ MB
Stadt(varchar)	16 byte	x	18.000.000	=	$\sim 343$ MB
Land(varchar)	20 byte	x	18.000.000	=	$\sim 274$ MB
				Summe	$\sim 754$ MB

#### Speicherplatzverbrauch mit Normalisierung

Zeitpunkt(smallint)	2 byte	x	18.000.000	=	$\sim 34$ MB
Stadt(integer)	4 byte	x	18.000.000	=	$\sim 72$ MB
Stadt(varchar + tinyint)	16 byte	x	21.000	=	$\sim 0,32$ MB
Land(tinyint)	1 byte	x	18.000.000	=	$\sim 17$ MB
Land(varchar + tinyint)	20 byte	x	218	=	$\sim 0,004$ MB
				Summe	$\sim 123$ MB

Tabelle 5.1: Vergleich des Speicherplatzverbrauchs

Durch die Normalisierung kann zwar Speicherplatz gespart werden doch nun muss überlegt werden wie die Informationen wiederbeschafft werden sollten. Wird eine Benutzerabfrage gestellt die eine Filterung anhand einer Stadt voraussieht, wird zuerst die *ID* der Stadt benötigt. Dabei können zwei verschiedene Ansätze verfolgt werden. Der erste Ansatz wäre ein Verbund zwischen *Town* und *Data*, um direkt mit dem Namen der Stadt zu arbeiten. Eine andere Möglichkeit wäre eine separate Abfrage an die Datenbank zu stellen, in der die *ID* zum Namen ermittelt wird. Mithilfe der *ID* kann dann ohne einen Verbund die Ergebnismenge ermittelt werden. Dieser Ansatz dürfte zu geringeren Antwortzeiten führen, da keine Netzwerkverzögerungen existieren. Überdies findet der Zugriff mittels geeigneter Indizierung und auf In-Memory-Tabellen statt, was die Antwortzeit ebenfalls verkürzt. Diese Vorgehen können für die Stadt, das Land und die Gruppenzugehörigkeit angewendet werden.

Die Tabelle *GroupDate* unterscheidet sich von den anderen Tabellen wie *Town* oder *Country*, da in dieser noch weitere Details vermerkt sind. Diese ermöglichen es die Personenkonstellation innerhalb der Gruppen über die Zeit hinweg nachzuvollziehen. In der Spalte *Action* wird festgelegt, ob die Tupel einen Eintritt oder einen Austritt einer Person darstellt. Die Spalte *Date* beinhaltet das Datum des Ereignisses. Mithilfe beider Attribute lassen sich Gruppenzusammensetzung auf bestimmte Zeitpunkte bezogen rekonstruieren.

### 5.2.3 Zugriffsstrukturen

Zum Beschleunigen der Zugriffe auf die Datensätze der H2-Datenbank wird auf die beabsichtigten Zugriffsverfahren eingegangen.

Indizes werden zur Beschleunigung von Suchen nach bestimmten Spaltenwerten eingesetzt. Ohne Indizes müsste die H2-Datenbank beim ersten Datensatz beginnen und dann die gesamte Tabelle durchgehen, um eine Abfrage zu beantworten. Je größer die Tabelle ist, desto höher ist der Aufwand dafür. Der Einsatz von Indizes ist in Anbetracht der Zielsetzung von

kurzen Antwortzeiten ein unerlässliches Hilfsmittel. Jeder Index bedeutet allerdings einen Zuwachs im Speicherplatzverbrauch, was bei In-Memory-Datenbanken zu berücksichtigen ist. Deshalb werden nur absolut notwendige Indizes erzeugt. Außerdem führen Indizes zu einem erhöhten Aufwand bei Aktualisierungen, da neben den Datensätzen auch ihre Indizes aktualisiert werden müssen. Zur Indexierung der Tabellen *Town*, *UserGroup*, *Country*, *User* und *Group* eignen sich Hash-Indizes. Sie bieten einen extrem schnellen Zugriff auf die Daten. Diese Schnelligkeit ergibt sich aus der Verwendung von Berechnungsvorschriften zur Ermittlung der Position des gesuchten Wertes [SSH11]. Indizierungen sollen im vorliegenden Schema über die Spalten mit der Bezeichnung *Name* vorgenommen werden, da der Client mit dem Namen anstatt mit der ID arbeitet. Mithilfe des Namens wird anschließend die zugehörige ID ermittelt. Die Nutzung von Hash-Indizes bringt allerdings Limitierungen mit sich. Eine der wichtigsten ist, dass sie nur für Vergleiche ("=") verwendbar sind. Somit werden keine Wertebereichabfragen ("<" oder ">") unterstützt.

Für die Tabelle *Data* eignet sich der B<sup>+</sup>-Baum-Index. Er ermöglicht eine effizientere Ausführung der Grundoperationen wie Suchen, Einfügen und Löschen. Die Spalte *startID* und *Date* eignen sich am besten für die Indexierung. Das erste Attribut auf das bei der Suche zugegriffen wird ist die *startID*, da sie die Personen beinhaltet von der aus die Beziehungen ermittelt und bewertet werden. Der zweite Zugriffswert (*Date*) eignet sich aufgrund der Tatsache, dass immer bestimmte Zeitspannen betrachtet werden. Sind die Werte der Spalte sortiert, können durch Bereichsabfragen sehr schnell alle Tupeln zwischen dem Start- und Endzeitpunkt ermittelt werden. Dabei handelt es sich um einen Mehr-Attribut-Index, da wir zwei Attribute verwenden. Der Vorteil eines Mehr-Attribut-Index ist, dass bei einer Punkt-Abfrage über alle Zugriffsattributwerte nur ein Indexzugriff erfolgen muss.

Beide Spalten sind sortiert und bieten sich somit für die Verwendung von einem geclusterten Index an. Ein geclusterter Index ist in der gleichen Form sortiert wie die interne Relation. Dadurch bietet er eine sehr gute Unterstützung für Bereichsabfragen.

## 5.3 ETL-Prozess

Daten der operativen Systeme unterstützen die wertschöpfenden Geschäftsprozesse innerhalb eines Unternehmens. Sie sind demnach auf die Steuerung und Überwachung des Tagesgeschäfts ausgerichtet und daher transaktionsbezogen. Die Daten sind aufgrund ihres eigentlichen Verwendungszwecks in ihren Begrifflichkeiten häufig nicht vergleichbar und ihrer Bewertung sowie Konsolidierung unterschiedlich. Um die Daten dennoch für unsere Bewertung einzusetzen, ist eine Überführung in eine geeignete Struktur von Vorteil. Eine solche Überführung wird in der Literatur als Extract-Transform-Load (ETL)-Prozess bezeichnet [ESHB11].

Abbildung 5.4 zeigt das erarbeitete Konzept zur Umsetzung eines solchen ETL-Prozesses. In den nächsten drei Abschnitten werden die ETL-Schritte näher erläutert.

### 5.3.1 Extract

Zunächst dient die Extraktion primär der Beschaffung von Daten aus der MSSQL-Datenbank. Überdies können durch den Prozess Daten bereits reduziert, zusammengeführt und ersetzt werden. Für eine zutreffende Formulierung der Abfragen müssen Besonderheiten im Ermitteln der Daten beachtet werden. Eine solche Besonderheit wäre beispielsweise zwei Spalten mit unterschiedlichen Formaten bei dem Datum.

In der Extraktion sollen SQL-Abfragen formuliert werden mit denen die Daten aus der MSSQL-Datenbank in die H2-Datenbank überführt werden können. Dabei gilt es die Abfragen so zu formulieren, dass die Struktur des Abfrageergebnisses der Tabelle des neuen Schemas entspricht.

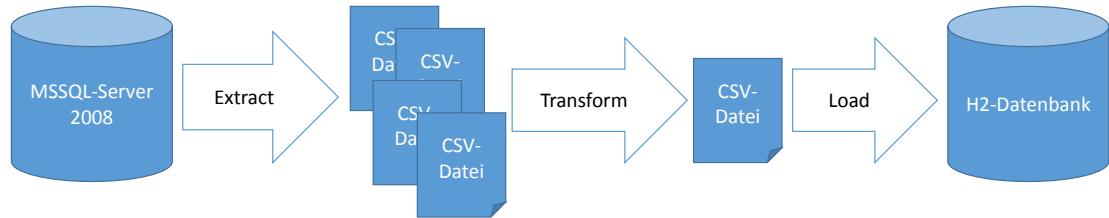


Abbildung 5.4: ETL-Prozess

Weiterhin sind Besonderheiten in der Extraktion zu beachten. Es existieren Datensätze in der MSSQL-Datenbank die sich über längere Zeiträume erstrecken. Beispielsweise erstrecken sich Termine wie Tagungen über mehrere Tage. In der MSSQL-Datenbank werden diese Termine in einer Tupel aufbewahrt. Bei unserer Analyse hingegen stellt jede Tupel eine Verknüpfung zwischen Personen zu einem bestimmten Tag dar. Somit muss ein Datensatz der sich über mehrere Tage erstreckt, in der H2-Datenbank durch mehrere Tupeln repräsentiert werden. Aufgrund dessen wird im Ergebnis der SQL-Abfrage die Zeitspanne eines CRM-Objektes in Tagen vermerkt. In späteren Transformationen kann mithilfe dieser Angabe die entsprechende Anzahl an Tupeln erzeugt werden.

Eine weitere Besonderheit ergibt sich durch eine Funktionalität von CAS genesisWorld, welche es ermöglicht Termine auf andere Zeitpunkte zu verschieben. Diese Funktion wird von manchen Nutzern missbraucht. Anstatt für einen ähnlichen Termin einen neuen Eintrag anzulegen, wird ein alter Termin aus Bequemlichkeit geschoben. Das hat zur Folge, dass Termine die tatsächlich stattgefunden haben, in der Datenbank nicht mehr existieren. Um trotzdem diese Termine zu berücksichtigen wurde folgendes Konzept erarbeitet.

Dem *Changelogbook* lassen sich Veränderungen von Feldern entnehmen. Um Schiebungen zu erkennen werden die Änderungen in den Spalten *start\_dt* und *end\_dt* benötigt. Zur Feststellung ob ein Termin stattgefunden hat und anschließend geschoben wurde, müssen drei Bedienungen erfüllt sein. Die erste ist der Zeitpunkt der Schiebung, der nach dem Termin liegen muss. Wird ein Termin aus anderen Gründen geschoben findet dies in der Regel vor dem Start des Termins statt, damit die Personen nicht unnötig zum Termin erscheinen. Die zweite Bedingung ist, dass der neue Termin in der Zukunft liegen muss. Neben den beiden zuvor genannten Bedingungen muss die Operation auf den Datensätzen ein Update gewesen sein. Nur dann ist der Datensatz von Relevanz für die Ermittlung der geschobenen Termine.

Die Ergebnisse sämtlicher Extraktionen werden in CSV-Dateien abgespeichert. Damit werden unter anderem nachvollziehbare Zwischenergebnisse erzeugt, die z.B. bei Fehlersuchen hilfreich sind. Weiterhin wird die Belastung für das Systems verringert, da nicht alle Ergebnisse bis zum Ende der Extraktion in der Java-Laufzeitumgebung aufbewahrt werden müssen.

### 5.3.2 Transform

Zu Beginn der Transformation werden Filterungen durchgeführt. Unter der Filterung von operativen Daten versteht man eine Bereinigung syntaktischer oder inhaltlicher Defekte, der zu übernehmenden Daten. Die MSSQL-Datenbank besteht zu 37% aus Nullwerten und zu 4% aus Feldern mit ausschließlich Leerzeichen. Daten die beispielsweise Nullwerte enthalten und für die Ermittlung des Datums benötigt werden, sind für die Analyse nicht zu gebrauchen. Sie können daher im Laufe des Prozesses aus den Daten entfernt werden. Bei

den anderen Filteroperationen können Nullwerte vernachlässigt werden, da sie zweckmäßig abdingbar sind.

Der nächste Schritt ist die Harmonisierung der Daten. Unter anderem besitzen die Telefonnummern kein einheitliches Format. Sie wurde manuell von Sachbearbeitern eingetragen. Zur Lösung des Problems werden aller Nummern in ein einheitliches Format gebracht, welches einen automatischen Vergleich ermöglicht. Dies geschieht durch vordefinierte Regeln. Die verschiedenen Variationen werden durch händische Untersuchungen ermittelt. Basierend auf den Variationen werden die Regeln zur Transformation festgelegt. Die verschiedenen Ausprägungen von CRM-Objekten müssen ebenfalls in eine einheitliche Struktur gebracht werden. Sie besitzen alle die benötigten Informationen, allerdings werden diese unter unterschiedlichen Bezeichnungen und eventuell in einem anderen Datenformat aufbewahrt.

Die in der Extraktion genannten Besonderheiten werden durch unterschiedliche Datenbankabfragen ermittelt. Dies führt zu vielen separaten Dateien. Zur Nutzung der Daten sind diese zum Abschluss der Transformation zu sortieren und zusammenzuführen. Das Ergebnis wird anschließend in einer CSV-Datei gespeichert, welche die Basis zum Einspielen der Daten in die H2-Datenbank bildet. Der Grund für das Sortieren und Zusammenzuführen der Dateien liegt in der Wiederverwendung der Datei in anderen Projekten, worauf in der Arbeit allerdings nicht näher eingegangen wird.

### 5.3.3 Load

Beim Laden der Datensätze in die H2-Datenbank soll ein sogenannter "bulk load" verwendet werden. Dieser wird häufig zum Laden von großen Datenmengen aus einer Datei in eine Datenbank eingesetzt. Er ermöglicht ein wesentlich schnelleres Einspielen von großen Datenmengen in die Datenbank, gegenüber der Verwendung von INSERT-Operatoren.

## 5.4 Entwurf der Oberfläche

Beim Entwurf einer Oberfläche ist der Detaillierungsgrad von Informationen ein wichtiger Leitfaden für die Gestaltung. In unserem Fall ist nicht die Eigenschaft eines CRM-Objekts von Interesse, sondern der Typ und dessen Häufigkeit von Beziehungen zu bestimmten Personen. Da keine vertraulichen und sensiblen Daten über die Personen oder die CRM-Objekte aufbewahrt werden, kann jeder Benutzer frei wählen von welcher Person die Bewertung ausgehen soll. Für die Oberfläche bedeutet dies einen Einstiegspunkt in Form eines Fensters in dem der Benutzername einer Person, von dem die Suche ausgehen soll, eingegeben wird. Zusätzlich werden Felder zum Eintragen der IP-Adresse und Portnummer des Server-Webprojektes angezeigt.

Nach der Anmeldung wird eine neue Seite aufgebaut, in der das Ergebnis der Analyse zu sehen ist. Dessen Aufbau ist in Abbildung 5.5 zu sehen. Im oberen Bereich auf der Seite sind alle Regler, CheckBoxen und Eingabefelder zur Filterung der Ergebnismenge zu finden. Direkt darunter befindet sich ein Diagramm, welches das Ergebnis der Abfrage visualisieren soll.

Bei der Wahl eines Diagrammtyps ergeben sich durch das verwendete Framework allerdings Einschränkungen. Im Grunde lässt sich jede Darstellung umsetzen, jedoch ist das Aufwand-Nutzen-Verhältnis zu berücksichtigen. Beispielsweise wäre in diesem Fall die Entwicklung eines eigenen Diagramms mit einem Aufwand verbunden der den geringen Mehrwert nicht rechtfertigen würde. Daher werden nur mit dem Framework umsetzbare Diagrammtypen betrachtet. In einer Vorauswahl wurden einige Typen ausgewählt, die in Abbildung 5.6 dargestellt sind.

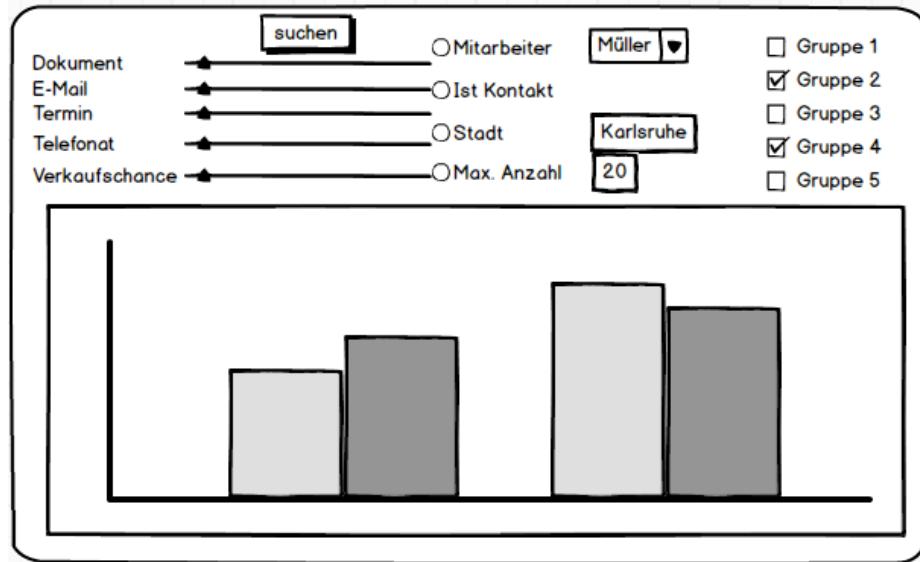


Abbildung 5.5: Entwurf der Oberfläche

Netzdiagramme (a) geben Eigenschaften verschiedener Systeme wieder. Sie eignen sich daher gut zur Darstellung von Ausprägungen. Für die vorliegenden Daten ist diese Darstellung gänzlich ungeeignet, da Mengen betrachtet werden.

Mithilfe von Liniendiagrammen (b) lassen sich Trends und Zeitreihen darstellen. Die Verwendung verschiedener Linien ermöglicht zudem die Darstellung mehrerer Trends. Die Benutzung dieses Diagramms wäre nicht sinnvoll, da die Ergebnismenge sich nicht auf verschiedene Zeitpunkte bezieht, sondern die Summe der Werte aus einer Zeitreihe beinhaltet.

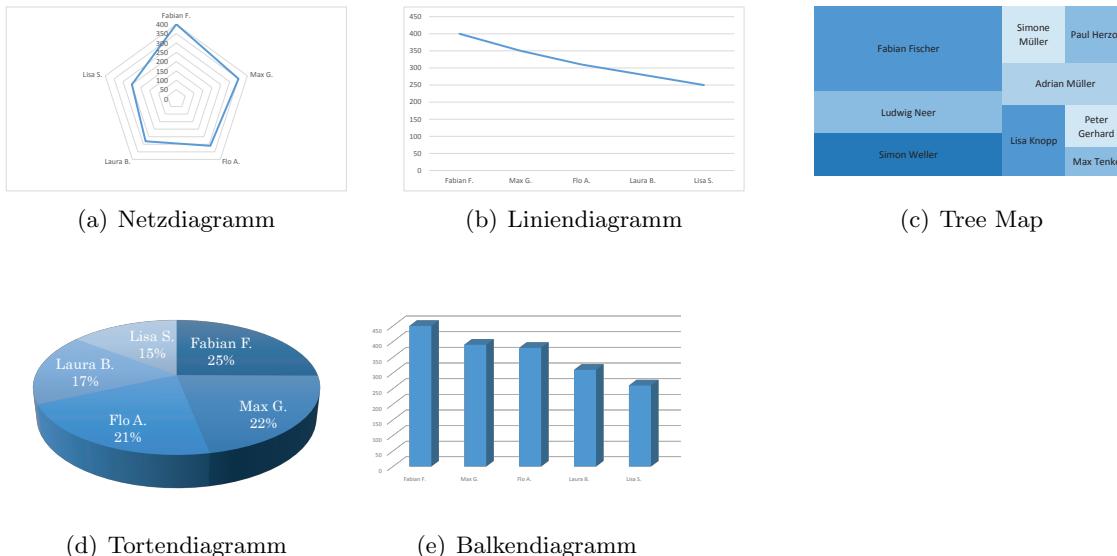


Abbildung 5.6: Entwürfe für die Oberfläche

Bei einer Tree Map (c) steht jede Fläche eines Rechtecks im proportionalen Zusammenhang zur Gesamtfläche. Die Beachtung von Größenverhältnissen stellt eine nützliche Eigenschaft für unsere Daten dar. In unserem Fall würde jedes Rechteck die Summe der CRM-Objekte einer Person darstellen. Diese könnten wiederum in weitere Rechtecke un-

terteilt werden oder mithilfe eines Drilldowns<sup>1</sup> die Anteile der unterschiedlichen CRM-Objekte aufzeigen. Eine weitere Unterteilung der Rechtecke würde allerdings aufgrund zu vieler Kacheln schnell zu einer schlechten Übersicht führen. Wird in einer Tree Map die Driltdown-Navigation gewählt, ist die Übersicht aller Informationen auf einen Blick nicht mehr gegeben. Aufgrund der Nachteile in der jeweiligen Variation wurde sich gegen den Einsatz einer Tree Map entschieden.

Kreisdiagramme (d) ermöglichen eine Betrachtung der Gesamtheit zu ihren Einzelstücken, da der Kreis ein geschlossenes System darstellt. Allerdings müssen sich alle Kreissegmente auf die gleiche Basis beziehen. Es eignet sich hervorragend zur Darstellung von Verhältnissen. Wird nun eine weitere Unterteilung der Teilwerte benötigt, geht die Übersicht verloren. Um das zu vermeiden wird die unterteilte Teilmenge häufig in separaten Ansichten dargestellt. Allerdings steigt dadurch der Aufwand für den Nutzer in der Bedienung des Systems.

Das Balkendiagramm (e) ist für die Darstellung der Daten am geeignetsten. Reihenfolgen lassen sich beispielsweise durch die resultierenden Stufen sehr gut darstellen. Balken selbst lassen sich außerdem in einzelne Teile aufspalten, ohne die Übersichtlichkeit zu verringern. Gegenüber dem Kreisdiagramm bietet es zwar eine schlechtere Betrachtung des Gesamten zu den Einzelstücken, allerdings ist das in diesem Anwendungsfall auch nicht problematisch. Es ist vielmehr von Bedeutung die Ergebnisse in Form eines Rankings darzustellen.

## 5.5 Technologien

Als eine der am meist verbreitetsten Programmiersprachen, stellt Java die Grundlage aller verwendeten Technologien dar. Zur Darstellung der Inhalte für den Client wird Vaadin verwendet. Der Apache Tomcat-Server nimmt die Rolle des Anwendungsservers ein. Die Kommunikation auf Basis von RESTful Web Services wird mithilfe von Jersey realisiert. Weiterhin wird opencsv für das Lesen und Schreiben von CSV-Dateien verwendet. JDBC wird zur Kommunikation zwischen dem Anwendungsserver und der Datenbank. Die H2-Datenbank stellt die Datenquelle des Systems dar. Im Folgenden werden alle Bestandteile, bis auf die bereits erläuterte H2-Datenbank, näher beschrieben.

**Vaadin 7** Vaadin ist ein Open-Source-Framework für den Aufbau von modernen Web-Anwendungen. Es verwendet ein rein serverseitiges, eventbasiertes Modell und ermöglicht eine Anwendungsentwicklung ohne direkte Verwendung von HTML und JavaScript-Code. Das Framework ermöglicht es, die gesamte Anwendungslogik auf der Serverseite einer Anwendung auszuführen, während die Clientseite nur für das Senden der Benutzeraktionen an den Server und für die Reaktion auf die Antworten verantwortlich ist. Da es auf GWT basiert, kann sowohl der Client- als auch der Server-Code in reinem Java geschrieben werden.

Die aktuelle Version von Vaadin wurde im Februar 2013 veröffentlicht. Die folgenreichste Änderung von Vaadin 6 auf Vaadin 7 ist eine komplette Neuentwicklung der Clientseite die weniger auf GWT aufbaut. Das ermöglicht eine bessere Unterstützung für die clientseitige Widget-Entwicklung und bietet sogar die Möglichkeit Offline-Anwendungen zu erstellen.

Die im Unternehmen vorhandene Erfahrung sowie die Open-Source Lizenz sind die treibenden Gründe für die Wahl von Vaadin. Allerdings war VaadinCharts, eine Erweiterung für Vaadin, für die Auswahl ausschlaggebend. Es basiert auf Highcharts, einem JavaScript-Paket. Highcharts zeichnet sich durch eine umfangreiche Sammlung an Funktionen zur Darstellung von Diagrammen aus.

---

<sup>1</sup> Als Driltdown wird im Allgemeinen die Navigation in hierarchischen Daten bezeichnet. Auf Oberflächen bezogen wird damit die Darstellung von Detailinformationen durch einen Klick auf Darstellungselemente ausgedrückt.

**Jersey** Jersey ist ein Open-Source-Framework zur Entwicklung von RESTful Web Services in Java, welches eine Unterstützung für JAX-RS-APIs bietet und die JAX-RS (JSR 311 und JSR 339)-Referenzimplementierung darstellt. JAX-RS-Annotationen werden verwendet um die Entwicklung und das Deployment von Webservice-Clients und Service-Endpunkten zu vereinfachen. Jersey enthält einen REST-Server und einen REST-Client. Auf der Serverseite verwendet Jersey ein Servlet zum Abfragen von vordefinierten Klassen, um REST-Ressourcen zu identifizieren. Über die web.xml Konfigurationsdatei werden die von der Jersey-Distribution bereitgestellten Servlets registriert. Diese Servlets analysieren die eingehenden HTTP-Nachrichten und wählen die entsprechende Klasse und Methode für die Anfragen aus. Diese Auswahl basiert auf Annotationen in diesen Klassen und Methoden. Weiterhin unterstützt JAX-RS die Erstellung von XML und JSON mithilfe der Java Architektur für XML Binding (JAXB).

**Apache Tomcat7** Tomcat ist ein Open-Source-Webserver, der von der Apache Group entwickelt wurde. Der Apache Tomcat implementiert die Java Servlets-, sowie die JavaServer Pages-Spezifikation von Sun Microsystems und stellt folglich eine Referenzimplementierung dar. Er stellt weiterhin eine rein auf Java-basierende HTTP-Webserverumgebung dar. Weiterhin kann der Tomcat über eine Oberfläche sowie durch Bearbeiten von XML-Dateien konfiguriert werden. Außerdem kann er als Behälter für Web-Anwendungen verwendet werden.

**opencsv** Da Java das Parsen von CSV-Dateien nativ nicht unterstützt, wird auf eine Drittanbieterbibliothek zurückgegriffen. Diese heißt opencsv und ist eine sehr einfache CSV-Parser-Bibliothek für Java. Die Bibliothek kann zum Erstellen, Lesen und Schreiben von CSV-Dateien verwendet werden. Die wichtigste Fähigkeit des opencsv-Parsers ist das Mapping von CSV-Daten auf Java-Bean-Objekte und umgekehrt.

**JDBC** Die JDBC-API ermöglicht den programmgesteuerten Zugriff auf relationale Daten, direkt aus der Java Programmiersprache heraus. Durch die Verwendung der JDBC-API können Java-Anwendungen SQL-Anweisungen ausführen, Ergebnisse abrufen und die Veränderungen auf die Datenquelle zurückschreiben. Die JDBC-API kann auch mit mehreren Datenquellen in einer verteilten, heterogenen Umgebung interagieren.

# 6. Umsetzung

In diesem Kapitel wird auf die Implementierung der Konzepte eingegangen. Die Architektur wurde wie in der Konzeption beschrieben umgesetzt. Daher wird vielmehr auf die genauen Funktionsweisen und Prozesse eingegangen. Anhand von Klassendiagrammen wird in den ersten beiden Kapiteln die Struktur und Funktionsweise der Webprojekte erläutert. Weiterhin wird der ETL-Prozess und die Abfrageerzeugung detailliert beschrieben. Der genaue Ablauf in der Aktualisierung wird in dem darauf folgenden Abschnitt behandelt. Abschließend wird der Aufbau der Oberfläche mit den damit verbundenen Designentscheidungen erläutert.

## 6.1 Server-Webprojekt

Es handelt sich um ein dynamisches, Java-basiertes Eclipse-Webproject, dessen Struktur und Einstellungen den Standardvorgaben aus der Erzeugung entsprechen. Deshalb wird direkt auf die Klassen eingegangen. Abbildung 6.1 zeigt das Klassendiagramm des Server-Webprojekt. Das Diagramm dient als Basis für die nachfolgenden Erläuterungen.

Die H2-Datenbank wird im Embedded-Modus betrieben, was eine Instanziierung der Datenbank zur Laufzeit notwendig macht. Die Instanziierung erfolgt in der Klasse *Database*. Das Attribut *dataSource* stellt die H2-Datenbank in Form eines Objektes dar. Eine Verbindung zur Datenbank wird mithilfe der Methode *getConnection()* aufgebaut. Diese Verbindung wird permanent offen gehalten, solang der Tomcat-Server läuft. Dazu wird die Verbindung dem Attribut *con* zugewiesen, welches von allen Methoden verwendet wird, die eine Verbindung zur Datenbank aufbauen wollen. Um die Datenbank mit der Web-Anwendungen zu starten, ist die Verwendung eines Servlets nötig. Dazu benutzen wir die Klasse *EntryPoint*, die das Interface *HttpServlet* implementiert. Um das Servlet direkt beim Start des Tomcats aufzurufen, sind in der *web.xml* folgende Zeilen eingetragen:

```
1 <servlet>
2   <servlet-name>H2</servlet-name>
3   <servlet-class>de.cas.db.EntryPoint</servlet-class>
4   <load-on-startup>1</load-on-startup>
5 </servlet>
```

Die 1 im Element *<load-on-startup>* bewirkt den Aufruf der Methode *init()* die eine Instanziierung der Klasse *Database* vornimmt. Zur Erzeugung des Schemas wird eine separate Klasse namens *SchemaBuilder* eingesetzt. In ihr werden sämtliche SQL-Anweisungen

zur Generierung des Schemas aufbewahrt und können über die Methode `createSchema()` ausgeführt werden.

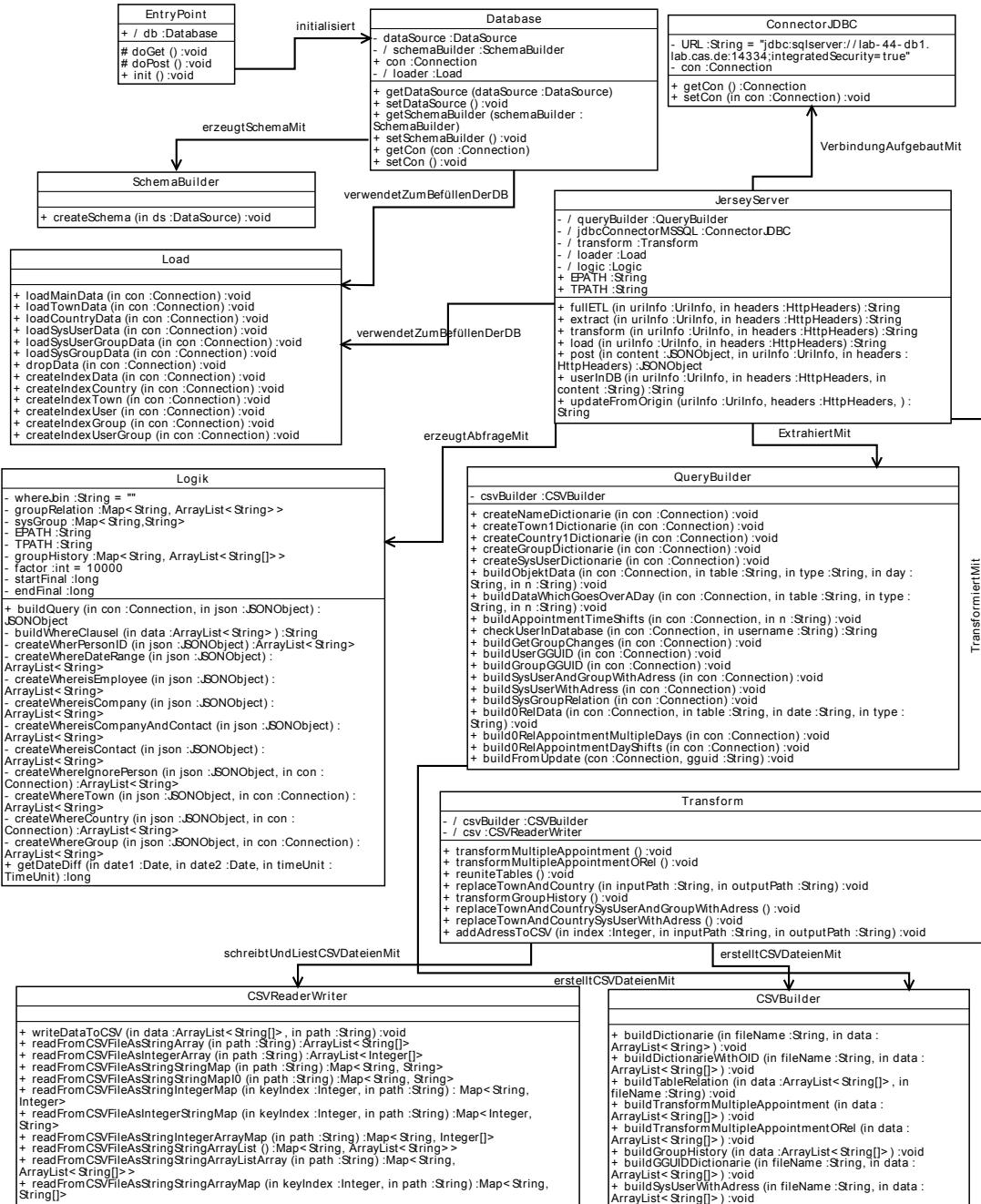


Abbildung 6.1: Server Klassendiagramm

In der Klasse `JerseyServer` ist ein REST-Server implementiert. Sie besitzt Methoden die mit den entsprechenden Annotationen, wie `@GET` oder `@POST`, die REST-Requests entgegen nehmen. Mit der Annotation `@Path` wird die URL angegeben, unter der die Methode angesprochen werden kann. Diese Methoden besitzen Übergabeparameter vom Typ `UriInfo` und `HttpHeaders` besitzen, mithilfe denen die Metadaten der REST-Requests abgegriffen werden.

Neben den Methoden zur Verarbeitung von REST-Requests, enthält die Klasse alle Objekte zur Durchführung des ETL-Prozesses. Das Objekt `ConnectorJDBC` besitzt ein Attribut namens `con`, welches den Verbindungsauaufbau zum MSSQL-Server, mithilfe von JDBC er-

möglichst. Zur Extraktion der Daten aus der MSSQL-Datenbank wird ein Objekt der Klasse *QueryBuilder* verwendet. Wie in der Abbildung zu sehen wird für jede SQL-Anweisung eine eigene Methode mit den entsprechenden Konfigurationen verwendet. Methoden welche die Übergabewerte *table*, *date* und *n* besitzen, werden zum Beschaffen der CRM-Objekte eingesetzt. Mithilfe des Parameters *table* wird der betroffene Tabellenname aus der MSSQL-Datenbank übergeben. Der Parameter *date* gibt das Attribut an, was für die Ermittlung des Datums verwendet werden soll. Um den Typ eines CRM-Objektes zwischen Personen festzuhalten wird der Parameter *n* verwendet, der eine Zahl zwischen eins und fünf beinhaltet. *QueryBuilder* verwendet ein Objekt der Klasse *CSV-Builder*, um die Ergebnisse der SQL-Abfragen in Dateien festzuhalten. Den Methoden wird als Übergabeparameter ein Dateiname sowie die zu speichernden Informationen übergeben.

Die Klasse *Transform* enthält Attribute und Methoden zur Bearbeitung der in der Extraktion erzeugten CSV-Dateien. Der Inhalt der Dateien wird mithilfe eines *CSVReaderWriter* Objekts ausgelesen und in die Java-Laufzeitumgebung geladen. Nach der Bearbeitung durch die Methoden der *Transform* Klasse, werden die Daten wieder in CSV-Dateien zurückgeschrieben. *CSVBuilder* besitzt Methoden die zusätzliche Parameter zum Schreiben aufweisen, die Anpassungen an den Schreiboperationen erlauben. Wohingegen *CSVReaderWriter* mithilfe der Methode *writeDataToCSV()*, sowie den Parametern *path* und *data* für allgemeine Schreiboperationen verwendet wird.

Mithilfe der Klasse *Load* wird die Datenbank befüllt. Sie wird von den *Database* und *JerseyServer* Objekten verwendet. In der Klasse *JerseyServer* werden mit der Methode *load()* die Methoden der Klasse *Load* aufgerufen. In der Klasse *Database* werden sie im Konstruktor selbst aufgerufen, um die Datenbank beim starten des Anwendungsserver automatisch zu befüllen.

Die Klasse *Logik* beinhaltet Attribute und Methoden zum Beantworten von Benutzerabfragen. Für jede Bedingungen der SQL-Abfrage an die H2-Datenbank wird eine separate Methode verwendet. Die jeweiligen Methoden werden nur gerufen, sobald der entsprechende Eintrag in der vom Benutzer übermittelten JSON-Datei vorhanden ist. Generiert werden die Abfragen an die H2-Datenbank durch die Methode *buildQuery()*.

## 6.2 Client-Webprojekt

Der Einstiegspunkt des Webprojekts ist die Klasse *CasAnalyticUI*. Sie ist von der Vaadin-Klasse *UI* abgeleitet. Die *UI* ist die oberste Komponente jeder Komponentenhierarchie in Vaadin. Es gibt eine Benutzeroberfläche für jede Vaadin-Instanz in einem Browserfenster. Ein *UI*-Objekt kann entweder ein gesamtes Browserfenster (-Tab) oder einen Teil einer HTML-Seite darstellen, in der eine Vaadin-Anwendung eingebettet ist. Nachdem eine *UI* von der Anwendung erstellt wurde, wird diese mit der Methode *init(VaadinRequest)* initialisiert. Zur Übersicht sind die Komponenten der Darstellung in die Klasse *RootUI* ausgelagert.

Die *RootUI* wird in der *CasAnalyticUI* instanziert. Die Klasse *RootUI* beinhaltet Objekte die Vaadin-Komponenten zur Darstellung des Anmeldefensters sowie der Hauptansicht. Mithilfe der Methode *buildLoginView()* werden die Komponenten des Anmeldefensters zur *UI*-Komponente hinzugefügt. Nach der Erzeugung der Komponenten, wird eine *JerseyClient* Klasse instanziert. Diese wird verwendet sobald der Nutzer IP, Port und einen Namen eingegeben hat und auf anmelden klickt. Anschließend wird die Methode *doPostRequestUserData()* gerufen, um zu überprüfen ob der Nutzer im System vorhanden ist.

Falls ja, werden durch die Methode *MainView()* alle bisherigen Komponenten der *UI* entfernt und durch Komponenten des Hauptfensters ersetzt. Das *JerseyClient* Objekt wird direkt im Anschluss verwendet, um mithilfe der Methode *doPostRequestData()* einen REST-Request an den Server zu senden. Dieser liefert das Ergebnis der SQL-Abfrage in einem

JSON-Objekt zurück, mit dessen eine erste Erzeugung des Diagramms durchgeführt wird. Das Diagramm selbst besitzt eine eigene Klasse namens *ChartComponent*. Sie leitet sich von der Klasse *Chart* ab, die Teil der VaadinChart-Bibliothek ist. Mithilfe der Methode *refreshChart()* wird das Diagramm bei Benutzerabfragen aktualisiert. Dazu werden ihr die Namen der Personen für die x-Achse übergeben sowie die neuen Balkenwerte. Weiterhin wird für jede Vaadin-Komponente eine separate Methode zur Konfiguration verwendet. Änderungen am Aussehen oder an der Funktionalität der jeweiligen Vaadin-Komponenten, werden nur innerhalb der entsprechenden Methode vorgenommen.

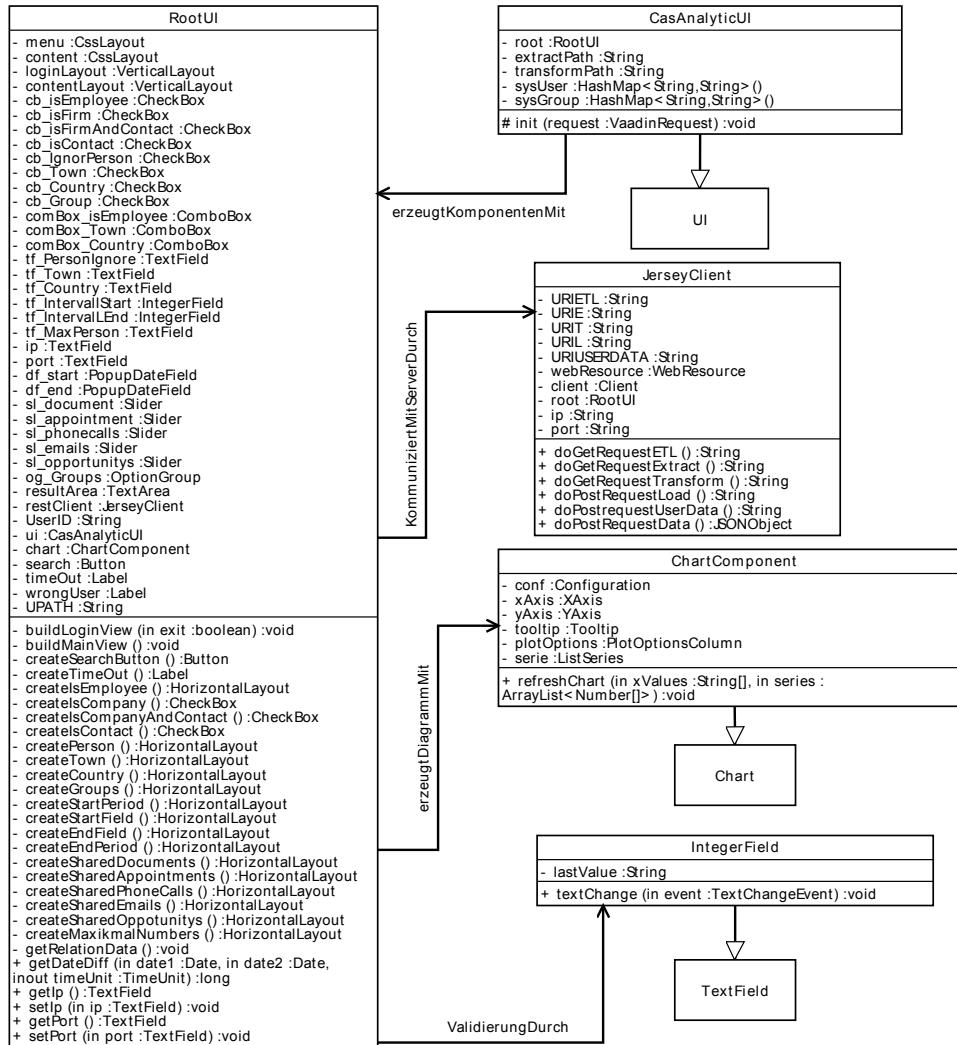


Abbildung 6.2: Client Klassendiagramm

An der Oberfläche gibt es Eingabefelder die nur Zahlen erwarten. Eingaben die nicht numerisch sind werden durch den Einsatz der Klasse *IntegerField* verhindert. Diese erweitert die Klasse *TextField*. Sie besitzt einen Event-Listener, der jede Eingabe des Benutzers abfängt. Gibt der Nutzer nicht numerische Zeichen ein werden diese direkt wieder entfernt. Dadurch werden Falscheingaben durch den Nutzer ausgeschlossen.

### 6.3 Aufbau der H2-Datenbankabfrage

Die Abfrage an die H2-Datenbank ist geschachtelt aufgebaut und unterteilt sich in eine innere und äußere Abfrage. In Abbildung 6.3 sind die Ergebnisse der inneren und äußeren

Abfrage abgebildet. Die Tabelle (A) zeigt einen aufs wesentliche reduzierten Ausschnitt der Datenbasis. In diesem Beispiel wird zur Bewertung von Beziehungen, die Person mit der *startID* 1 zum Betrachten ausgewählt. Mithilfe der inneren Abfrage wird zuerst die Anzahl der jeweiligen CRM-Objekte in Bezug auf die *endID* ermittelt. Das Ergebnis ist in der Tabelle (B) zu sehen. In ihr ist beispielsweise zu erkennen, dass die betrachtete Person 8 Dokumente mit einer Person teilt, welche die *endID* 3 besitzt. Die Tabelle (B) enthält allerdings noch nicht alle benötigten Informationen. Um an diese zu gelangen wird basierend auf der Tabelle (B) die äußere Abfrage gestellt. Dessen Ergebnis ist in der Tabelle (C) dargestellt. In ihr ist die Anzahl der verschiedenen CRM-Objekte in einer Tupel festgehalten. Weiterhin besitzt sie die Summe aller CRM-Objekte. Diese werden benötigt um eine Reihenfolge unter den Tupeln zu bilden. Diese Reihenfolge wird zum Ermitteln eines Rankings der Beziehungen benötigt.

Mit diesem Vorgehen werden noch nicht alle Funktionen abgedeckt, sie stellt allerdings die Basisfunktionalität dar. Diese kann durch Eingaben der Nutzer erweitert werden. Diese zusätzlichen Parameter die vom Benutzer übergeben werden, müssen in der SQL-Abfrage berücksichtigt werden. Um die SQL-Abfrage so schlank wie möglich zu halten werden die optionalen Konfigurationen nur bei Bedarf in die SQL-Abfrage aufgenommen. Ein Beispiel dafür ist die Gewichtung von Zeit, die eine höhere Komplexität der Abfrage bewirkt und nur bei einer Angabe durch den Benutzer in die SQL-Anweisung einfügt wird.

Abbildung 6.3: Ergebnisse der verschachtelten SQL-Abfrage

Das Verfahren zur Gewichtung der Zeit wird anhand der Abbildung 6.4 erläutert. Die Abbildung zeigt ein Koordinatensystem mit der Gewichtung von einzelnen Zeitpunkten. Die x-Achse stellt den zeitlichen Verlauf und die y-Achse die Gewichtung dar. Der Startzeitpunkt wird durch  $t_s$  markiert, wohingegen  $t_e$  den Endzeitpunkt angibt. Mithilfe von  $t_1$  und  $t_2$  werden die zu gewichtenden Zeitspannen festgelegt. Um nun die Zeitspannen anders zu gewichten wird eine lineare Abstufung der Tage vorgenommen. Für die Zeitspanne zwischen  $t_s$  und  $t_1$  bedeutet dies, dass der Wert eines Tages zunehmend steigt. Wird  $t_1$  erreicht, besitzt jeder Tag wieder eine Wertigkeit von 1. Bei  $t_2$  verhält es sich ähnlich. Mit jedem Tag ab  $t_2$  sinkt der Wert des Tages bis der Zeitpunkt  $t_e$  erreicht ist.

Um die Gewichtung eines bestimmten Tages zu berechnen werden die folgenden zwei Faktoren verwendet:

$$f_1 = \frac{1}{t_1 - t_s} \quad (6.1)$$

$$f_2 = \frac{1}{t_e - t_2} \quad (6.2)$$

Neben den Faktoren  $f_1$  und  $f_2$  werden Variablen zum Erfassen der schrittweisen Erhöhungen und Verringerungen von Tagen benutzt. Für den Zeitraum zwischen  $t_s$  und  $t_1$  wird die Variable  $v_1$  verwendet. Der andere Zeitraum arbeitet mit der Variable  $v_2$ . Die Variable  $v_1$  beginnt mit dem Wert 0 und erhöht sich mit jedem Tag um 1. Die Differenz zwischen  $t_2$  und  $t_e$  stellt den Wert von  $v_2$  dar. Dieser wird mit jedem Tag ab  $t_2$  um 1 verringert.

Mit den Faktoren und Variablen werden die Werte der Tage berechnet. Dazu wird der Faktor mit der Variabel multipliziert. Das Produkt bildet den Wert eines Tages. Dieser wird in der Datenbankabfrage verwendet, um Tupeln einen geringeren Wert zuzuweisen.

Die SQL-Abfrage wird dazu um eine weitere innere Abfrage erweitert, die das zuvor beschriebene Vorgehen umsetzt. Diese Abfrage wird zuerst gestellt, da nach der anderen inneren Abfrage kein Datum mehr vorhanden ist.

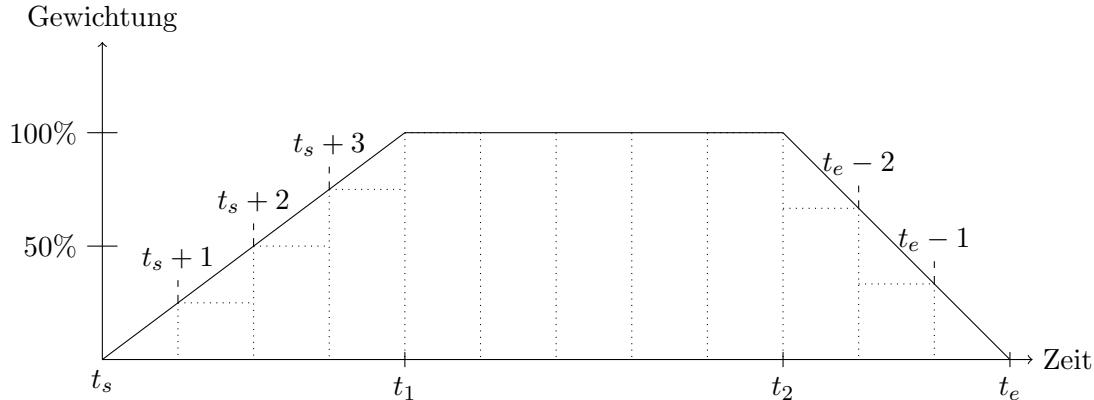


Abbildung 6.4: Gewichtung der Zeit

Neben der Zeit lassen sich die jeweiligen CRM-Objekte unterschiedlich gewichten. Bei einer Abweichung von 100 Prozent wird die Anzahl des jeweiligen CRM-Objektes um den entsprechenden Prozentsatz verringert. Diese Gewichtung findet in der inneren Abfrage statt. Dazu wird die Anzahl mit dem jeweiligen Prozentsatz verrechnet.

Die restlichen vom Benutzer festgelegten Parameter fügen weitere WHERE-Klauseln in die SQL-Anweisung ein. Ein Parameter muss zuvor ermittelt werden und wird daher näher beschrieben. Es handelt sich dabei um Gruppen, die aus der Ergebnismenge ausgeschlossen werden können. Ihre Konstellation in Bezug auf Personen ist im Laufe der Zeit variabel. Um dies zu berücksichtigen wird die Tabelle *UserGroup* verwendet. Dabei wird folgendermaßen vorgegangen: Zuerst wird für jede Gruppe ein eigenes Array in der Java-Laufzeitumgebung erzeugt. Das Array beinhaltet die IDs der Personen aus der jeweiligen Gruppe. Liegt nun der Zeitpunkt des Feldes *Date* nach dem Anfangszeitpunkt der Abfrage und die Spalte *Action* enthält eine 1, wird das Array um diese Person reduziert. Enthält sie eine 0, wird die Person zum Array hinzugefügt. Mit einer 1 in der Spalte *Action* wird der Austritt einer Person aus der Gruppe markiert. Eine 0 weist auf den Eintritt einer Person in die Gruppe hin. Dadurch wird die Struktur der Gruppe zum Anfangszeitpunkt wiederhergestellt. Mit Hilfe des Array werden anschließend die WHERE-Klauseln um weitere auszuschließende Personen erweitert.

Innerhalb von Zeiträumen können sich Gruppen auch verändern, jedoch kann dies nicht berücksichtigt werden. Es kann jeweils nur ein bestimmter Zeitpunkt betrachtet werden. In diesem Fall wurde der Anfangszeitpunkt  $t_s$  ausgewählt.

## 6.4 ETL-Prozess

Beginnend mit der Extraktion wird in der Abbildung 6.5 das Vorgehen in der Informationsbeschaffungsweg dargestellt. Jeder Pfeil stellt einen Verbund zwischen den jeweiligen Tabellen dar. Das Vorgehen in der Abbildung zeigt wie die Datensätze für die Tabelle *Data* ermittelt werden.

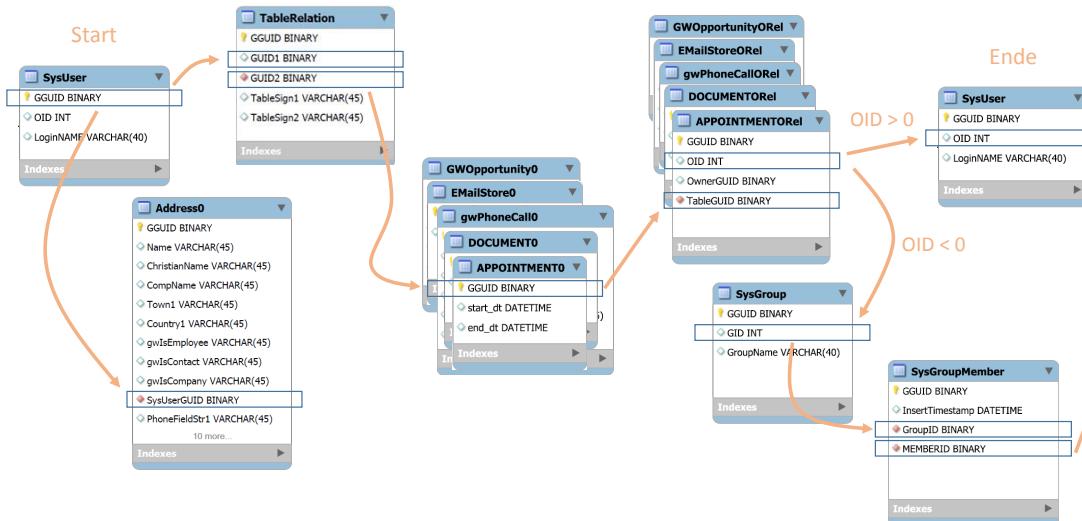


Abbildung 6.5: Vorgehensweise bei der Extraktion

Die erste Information entstammt der Tabelle *SysUser*. Mithilfe eines Verbundes zwischen den Tabellen *SysUser* und *Address0* werden die persönlichen Daten zu den Personen ermittelt. Anschließend werden durch einen Verbund zwischen *TableRelation* und *SysUser* alle Tabellen bestimmt, mit denen die Personen eine Verknüpfung besitzen. Der nächste Verbund wird zwischen *TableRelation* und den Tabellen *gwOpportunity*, *gwPhoneCall0*, *Document0*, *EmailStore0* und *Appointment0* gebildet. Dadurch werden alle CRM-Objekte der Personen ausgemacht. Um festzustellen mit welchen anderen Personen ein Dokument geteilt ist, wird ein weiterer Verbund mit der passenden ORel-Tabelle gebildet. In der ORel-Tabelle kann die *OID* positiv, sowie negativ sein. Bei einem negativen Wert stellt die *OID*, eine *GID* der Tabelle *SysGroup* dar. Zur Auflösung von Gruppen in einzelne Personen werden folgende Verbunde gebildet. Zuerst zwischen *SysGroup* und *SysGroupMember*, um alle Personen die zu einer Gruppe gehören zu erhalten. Anschließend zwischen *SysGroupMember* und *SysUser*, um die *OID* der Person zu bestimmen.

Die durch den Verbund gewonnenen Informationen werden weiterhin auf die 9 relevanten Werte des neuen Schemas verringert. Zu einem die *OID* des *SysUser*, von dem die Suche ausgeht. Zum anderen das Datum, welches durch das CRM-Objekt ermittelt wird. Weiterhin wird die zweite *OID* beibehalten, die durch den Verbund mit einer zweiten *SysUser* Tabelle gewonnen wird. Zum Schluss wird manuell eine vierte Information beigefügt, die besagt welchem CRM-Objekt die Tupel entstammt. Weiterhin werden die restlichen fünf Werte aus der Adresse übernommen.

Für den Sonderfall, dass ein Datum über mehrere Tage geht, wird eine zehnter Wert an das Ergebnis angehängt, welche den Zeitraum in Tagen beinhaltet. Zur Beschaffung der

geschobenen Termine wird die gleiche Abfrage wie zuvor gestellt, allerdings mit einem Verbund zwischen der Tabelle Appointment0 und *ChangeLogBook*. Dadurch kann zum Bestimmen der betroffenen Termine wie in Kapitel 5.3.1 beschrieben vorgegangen werden.

Jedes Ergebnis einer Datenbankabfrage wird in einer CSV-Datei auf dem Tomcat-Server gespeichert. Diese CSV-Dateien stellen die Grundlage der Transformation dar. Jede dieser Dateien beinhaltet Werte für die Tabelle *Data* aus der H2-Datenbank, in der Form wie sie in Abbildung 6.6 zu sehen ist.

Struktur der Datei	
"startID", "Date", "Typ", "EndID", "isEmployee", "isContact", "isFirm", "Town", "Country", "Dauer"	
1703	"10", "4211", "1", "14476", "false", "true", "false", "Karlsruhe", "Deutschland"
1704	"10", "4211", "1", "15260", "false", "false", "true", "Karlsruhe", "Deutschland"
1705	"10", "4211", "1", "16642", "true", "false", "false", "Karlsruhe", "Deutschland"
1706	"10", "4096", "1", "15916", "true", "false", "true", "Karlsruhe", "Deutschland"
1707	"10", "4096", "1", "12669", "false", "true", "false", "Karlsruhe", "Deutschland"
1708	"10", "4096", "1", "15836", "false", "true", "true", "Karlsruhe", "Deutschland", "3"
1709	"10", "4096", "1", "14462", "null", "null", "null", "null", "null"
1710	"10", "4096", "1", "15912", "null", "null", "null", "null", "null"

Abbildung 6.6: Ausschnitt einer CSV-Datei nach der Extraktion

Alle CSV-Dateien werden auf die in Abbildung 6.6 zu sehenden Anomalien untersucht. Dabei wird in Zeilen in denen die letzten fünf Werte fehlen (Zeile 1709), die Adresse über die zweite *OID* ergänzt. Bei Nullwerten wird überprüft ob wirklich keine Adresse vorhanden ist, falls doch wird die Adresse ergänzt. In Zeile 1707 ist zu sehen, dass ein Nullwert anstatt eines Datums vorkommen kann. Diese Zeilen werden aus den CSV-Dateien entfernt. Wenn wie in Zeile 1708 ein zusätzlicher Wert vorhanden ist, erstreckt sich die Dauer des CRM-Objektes über mehrere Tage. Die Zeile bleibt bestehen allerdings wird der letzte Wert entfernt. Die Zahl wird jedoch zwischengespeichert und zur Erzeugung der entsprechenden Anzahl von Tupeln wiederverwendet. Bei einer Dauer von drei Tagen würde die Zahl entfernt und zwei weitere Tupeln in die Datei eingefügt werden. Jede Tupel würde einen anderen Tag in der Zeitspanne des CRM-Objektes darstellen. Nach der Beseitigung von Anomalien werden noch die Städte und Länder durch ihre jeweilige *ID* aus der Tabelle *Town* und *Country* ersetzt.

Die überprüften Daten werden wieder in CSV-Dateien abgelegt. Diese besitzen den gleichen Namen, allerdings wird noch der Zusatz *\_transf* angehängt, der sie als transformiert kennzeichnet. Diese Dateien werden anschließend in der Java-Laufzeitumgebung zusammengeführt. Bei der Zusammenführung sind zum ersten Mal alle Daten gleichzeitig in der Anwendung vorhanden, weshalb an dieser Stelle alle Duplikate beseitigt werden. Überdies werden die Zeilen sortiert. Dabei wird mit zwei Kriterien verfahren. Das erste Kriterium ist die *OID* der Person von der die Suche ausgeht. Falls Werte sich gleichen wird das Datum zum Sortieren herangezogen. Nachdem alle Zeilen sortiert und von Duplikaten bereinigt sind, werden sie in einer CSV-Datei abgelegt.

Diese Datei wird bei jedem Start der Datenbank verwendet, um einen Bulk-Load für die

H2-Datenbank zu initialisieren. Nachdem Einspielen der Daten in die Datenbank, werden die Indizes der Datensätzen erzeugt.

## 6.5 Aktualisierung des Datenbestandes

Wie zuvor in Abschnitt 5.1 behandelt, wird die Aktualisierung unseres Datenbestandes von CAS genesisWorld angestoßen. Die Implementierung ist in Form einer COM-Komponente umgesetzt. Sie wird in einer DLL-Datei definiert. Diese muss Namenskonventionen einhalten. Es werden nur Dateien vom CAS genesisWorld Anwendungsserver erkannt die mit dem Prefix *pGSAxExtCustomServerDataPlugin* beginnen. Der Name der DLL-Datei ist in der *RegisterSDKDataPlugIns.xml* hinterlegt, damit der Anwendungsserver beim Start das Plugin findet. Weiterhin ist in der XML-Datei eine Tabelle immer paarweise mit einer DLL angegeben. Dadurch wird ein Plugin auf eine Datenbanktabelle registriert und bekommt alle betreffenden Änderungen mit.

Die Programmbibliothek selbst ist in Delphi geschrieben. Abbildung 6.7 zeigt die Struktur der DLL-Datei. Die Klasse selbst implementiert sechs verschiedene Schnittstellen. *ComObj* stellt Funktionen zur Erstellung und Bearbeitung von COM-Objekten zur Verfügung. Um Funktionalitäten von CAS genesisWorld vollständig zu nutzen, wird die *ActiveX* Schnittstelle benötigt. Wie bereits behandelt findet die Übertragung der Daten über das REST-Protokoll statt, wofür die *idHttp* Schnittstelle verwendet wird. Um Konvertierungen der vom Anwendungsserver erhaltenen Binärwerte vorzunehmen, werden die Funktionen der Schnittstellen *CAS\_ToolsCOM* und *CAS\_VarType14Fix* genutzt. Das Auffangen der geänderten Daten, welches die eigentliche Kernfunktionalität darstellt, wird durch die Funktionen der *IGWSDKDataPlugin* Schnittstelle implementiert.

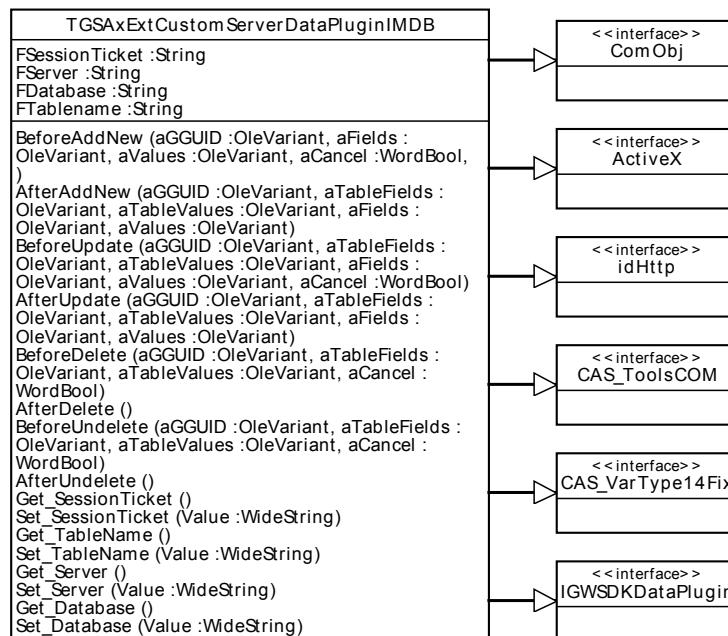


Abbildung 6.7: Klassendiagramm Plugin

Die Funktionen der Abbildung 6.7 gehören von *BeforeAddNew()* bis *AfterUndelete()* zur *IGWSDKDataPlugin* Schnittstelle. Es sind zwar alle Funktionen der Schnittstelle in der DLL-Datei implementiert, allerdings sind nur die mit "After" beginnen auch mit Logik hinterlegt. Für unser System reicht es aus, über Änderungen im Nachhinein benachrichtigt zu werden. Die Funktionen enthalten alle die gleiche Logik und unterscheiden sich lediglich in den Übergabeparameter.

Die Funktionsweise wird im Folgenden anhand der Kommunikationen beim Anlegen eines neuen Datensatzes durch einen CAS genesisWorld Client erläutert und in der Abbildung 6.8 dargestellt. Nachdem ein Benutzer einen neuen Datensatz angelegt hat wird das Plugin benachrichtigt. Die Funktion *BeforeAddNew()* enthält die *GGUID* der Tupel, den Namen der Spalte, sowie die neuen Werte des Datensatzes. In unserem Plugin führt das zu keiner Reaktion an dieser Stelle, da wir erst auf Änderungen im Nachhinein reagieren. Im Anschluss an die Benachrichtigung wird der neue Datensatz in die MSSQL-Datenbank eingefügt. Nachdem das passiert ist werden die registrierten Plugins wieder benachrichtigt.

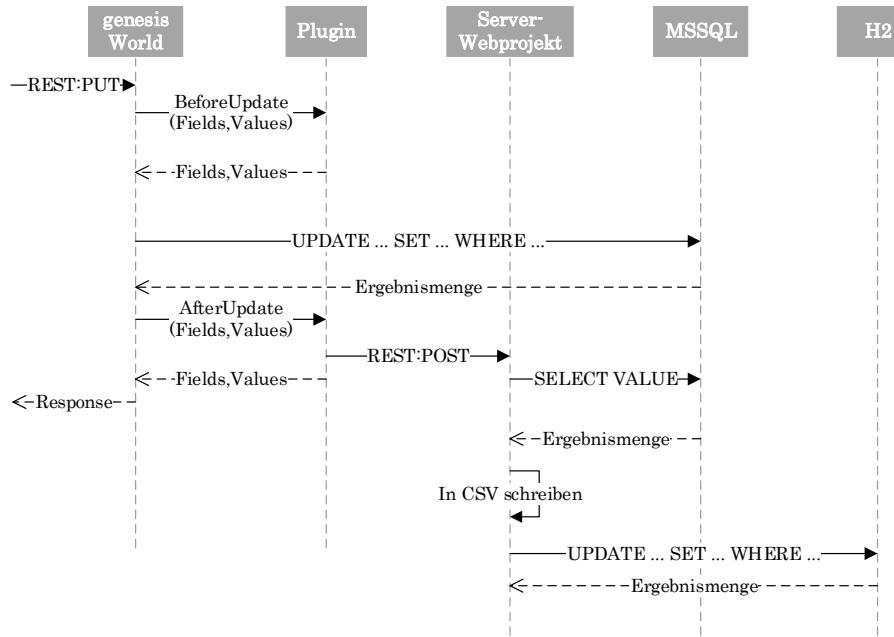


Abbildung 6.8: Sequenzdiagramm für einen neuen Datensatz

In dem Plugin ist die entsprechende Methode *AfterAddNew()* mit einer Logik hinterlegt. Diese prüft zuerst ob der Datensatz eine Relevanz für das System besitzt. Falls er sich als relevant herausstellt, wird die *aGGUID* in einen String konvertiert. Anschließend werden die Header-Werte der *idHttp* Variable gesetzt. Sie beinhalten Werte, wie die URI oder HTTP-Metadaten. Sobald alle Daten in der *idHttp* gesetzt sind, wird ein POST-Request an das Server-Webprojekt im Tomcat-Server übermittelt.

Der POST-Request enthält die *GGUID* und die Art der Operation, die auf den Daten ausgeführt wurde. Bei neuen Daten beispielsweise wird ein Header namens "newGGUID" mit der zuvor konvertierten *GGUID* als Wert gesetzt. Im Server-Webprojekt wird der neue Wert zuerst in eine CSV-Datei geschrieben und anschließend in die H2-Datenbank eingefügt.

## 6.6 Oberfläche

In diesem Abschnitt wird die Umsetzung der Darstellung erörtert. Den Einstiegspunkt für Benutzer stellt das in Abbildung 6.9 zu sehende Anmeldefenster dar. Der Hintergrund der Webseite ist in einem dunklen grau gestaltet, um einen Kontrast zum weißen Hintergrund der Bedienelemente zu schaffen. Zur Identifikation des Systems mit der Firma ist das Logo der CAS Software AG im linken Teil abgebildet. Im rechten Teil des Fensters existieren drei Eingabefelder. Zuerst ein Feld zur Eingabe der IP-Adresse des Servers. Die dazugehörige Portnummer wird im darauf folgenden Feld eingegeben. Das dritte Feld ist für den Namen des Nutzers vorgesehen, der den Ausgangspunkt der Analyse darstellt. Abschließend wird ganz klassisch ein Button zum Fortfahren auf der Webseite eingesetzt. Falls allerdings der

eingegeben Nutzernname nicht existiert, wird eine Warnmeldung direkt über dem zweiten Eingabefeld ausgegeben und auf der Anmeldeseite verblieben.

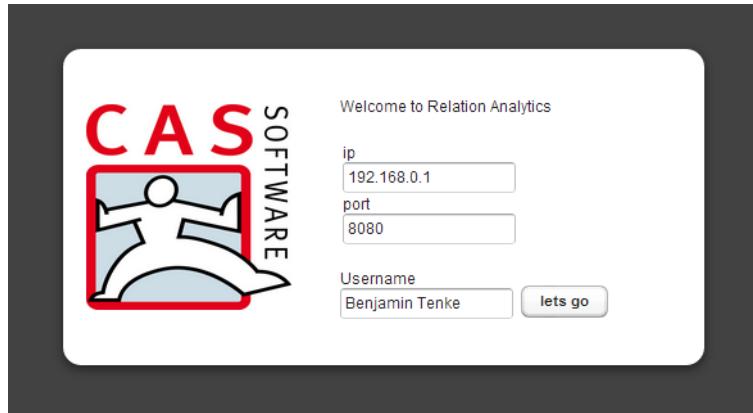


Abbildung 6.9: Anmeldefenster

Das Hauptfenster wurde vom Aufbau, wie in Abschnitt 5.4 beschrieben umgesetzt. Im oberen Bereich befindet sich eine Leiste, die anhand der Microsoft Richtlinien für Design entworfen wurde. Dies schafft ein vertrautes Gefühl mit der Oberfläche und schafft eine schnelle Akzeptanz bei den Nutzern. Die Leiste ist in vier Bereiche aufgeteilt. Der erste Bereich, ganz links, dient der Gewichtung der CRM-Objekte und dem anstoßen der Abfrage. Aufgrund der Gewichtung in Prozent, ist ein fester Wertebereich von 0 bis 100 vorgegeben. Textfelder eignen sich daher weniger, da sie beliebige Eingaben ermöglichen. Der Einsatz von Reglern bietet eine einfachere und selbsterklärende Form der Bedienung. Der begrenzte und kleine Wertebereich begünstigen den Einsatz der Regler. Zum Stellen der Anfrage wird ein einfacher Button eingesetzt. Direkt unter dem Button befindet sich ein Text, der die benötigte Zeit für die Verarbeitung der Abfrage anzeigen.

Der zweite Bereich dient zeitlichen Anpassungen. Das erste und dritte Feld können zum Verändern des Betrachtungszeitraums verwendet werden. Sie beinhalten den Anfangs- und Endzeitpunkt. Händische Eingaben weisen eine schlechte Bedienbarkeit auf, weswegen ein sogenannter "Datumspicker" eingesetzt wird. Dieser befindet sich direkt neben dem Textfeld und öffnet sich nach einem Klick auf das Symbol. Er stellt einen grafischen Kalender dar, aus dem durch klicken auf ein Tag das Datum bestimmt werden kann. Die Möglichkeit zur Eingabe durch direktes ändern des Textes bleibt allerdings weiterhin erhalten. Die anderen beiden Felder sind für die Gewichtung der Zeit vorgesehen. Diese Felder dienen zur Festlegung von  $t_1$  und  $t_2$ , aus der Abbildung 6.4. Das obere Feld ist für  $t_1$ . Hier kann die Zeitspanne zwischen  $t_s$  und  $t_1$  in Tagen festgelegt werden. Der Wert für die Zeitspanne zwischen  $t_e$  und  $t_2$  wird aus dem unteren Eingabefeld entnommen.

Bis auf den Ausschluss von Personen anhand ihrer Gruppen sind alle anderen Kriterien zur Filterungen der Personen im dritten Bereich vorhanden. Mithilfe der Checkboxen kann der Nutzer festlegen, welche Kriterien zum Ausschluss auf die Analyse angewendet werden sollten. Neben der Filterung durch bestimmte Personen, Länder, Städte usw. ist hier eine Begrenzung der Ergebnismenge umgesetzt. Im untersten Feld kann der Benutzer die Anzahl der angezeigten Personen bestimmen.

Der Bereich ganz rechts in der Leiste, ist für den Ausschluss von Gruppen vorgesehen. In ihr werden alle Gruppen im System mit einer Checkbox und einem Namen dargestellt. Dabei können beliebig viele Gruppen ausgewählt werden. Da die Anzahl der Gruppen überschaubar ist, entschied man sich alle anzuzeigen, anstatt einer händischen Eingabe durch den Nutzer. Dadurch können Benutzer Gruppen auswählen, die sie zuvor nicht beim Namen kannten.

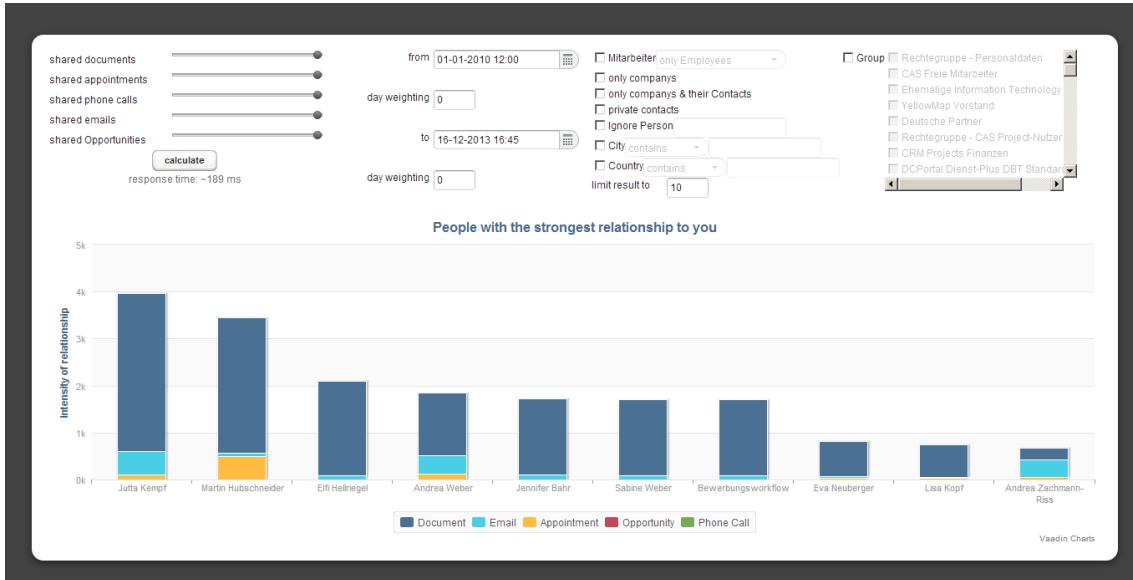


Abbildung 6.10: Hauptseite der Anwendung

Den zentralen Bereich des Fensters stellt das Diagramm dar. Die Balken selbst sind in fünf verschiedene Elemente unterteilt. Jedes Element wird durch eine andere Farbe dargestellt. Die fünf Elemente sind die verschiedenen CRM-Objekte. Die Zuordnung der Farbe zu dem jeweiligen Element, wird über eine Legende im unteren Bereich des Fensters umgesetzt. Eine Besonderheit ist, dass durch einen Klick auf eine der Farben, das jeweilige Element von der Darstellung ausgeschlossen wird. Beispielsweise kann der Nutzer auf die blaue Farbe klicken, was einen Neuaufbau des Diagramms ohne Dokumente bewirkt. Durch den Ausschluss wird allerdings keine neue Abfrage gesendet. Das bedeutet die Reihenfolge in der die Personen angezeigt werden und die Datenbasis bleibt gleich. Mit einem wiederholten Klick auf die entsprechende Farbe in der Legende lässt sich der Originalzustand wiederherstellen. Zusätzlich zu der y-Achse, die eine Gesamtpunktzahl aufzeigt, kann das Verhältnis von der Anzahl eines CRM-Objektes zur Summe betrachtet werden. Dies geschieht durch einfaches platzieren des Mauszeigers, auf dem jeweiligen Bereich des Balkens. Dadurch öffnet sich ein Tooltip mit den entsprechenden Informationen.

Die Ausführung der Anfrage erfolgt in der Regel mit dem dafür vorgesehenen Button. Die Regler und das Datum, jedoch lösen bei Veränderungen automatisch eine neue Abfrage aus. Dies soll die kurze Antwortzeit des Systems untermauern und eine bessere Nutzererfahrung schaffen. Das Datum sowie die Gewichtung wurden dazu ausgewählt, da sie die am meisten benutzen Konfigurationsmöglichkeiten darstellen.

# **7. Fazit und Ausblick**

In diesem abschließenden Kapitel werden die Ergebnisse der Arbeit in ihren wichtigsten Punkten zusammengefasst und anhand der Anforderungen aus Kapitel 3.2 bewertet. Anschließend wird ein Ausblick auf weiterführende Möglichkeiten, sowie zukünftige Verbesserungsmöglichkeiten gegeben.

## **7.1 Zusammenfassung**

Aus der Motivation heraus wurde in der vorliegenden Arbeit ein System zur Bewertung von Beziehungen entwickelt. Hierzu wurden zuerst alle relevanten Komponenten von CAS genesisWorld untersucht. Dabei wurden Tabellen und Spalten identifiziert, die zur Umsetzung des neuen Systems notwendig sind. Anschließend wurden die Anforderungen an das neue System erhoben. Mit dem Wissen über die zu übernehmenden Daten und den Anforderungen wurde eine passende Datenbank ausgewählt. Diese sollte den zuvor erhobenen Anforderungen gerecht werden. Dabei wurden NoSQL-Datenbanken hinsichtlich ihrer Eignung untersucht. Sie konnten in diesem Fall allerdings nicht überzeugen, somit entschied man sich für die H2-Datenbank. Die Entscheidung zugunsten der H2-Datenbank ist auf die im Hauptspeicher gehaltenen Tabellen zurückzuführen.

Aufbauend auf der zuvor ausgewählten Datenbank wurden Konzepte zur Umsetzung des Systems entwickelt. Bei der Konzeption wurde deduktiv vorgegangen. Zuerst wurde die Architektur definiert und anschließend die einzelnen Komponenten detailliert geplant. Bei der Planung wurde nicht versucht ein universell einsetzbares System zu entwickeln, sondern vielmehr eine domänen spezifische Lösung für das Szenario auszuarbeiten. Nachdem alle Technologien sowie Vorgehensweisen festgelegt wurden, ging man auf die Umsetzungen ein. Indessen eine Beschreibung der Funktionsweise einzelner Komponenten durchgeführt wurde. Neben der Funktionsweise wurde die Interaktion unter den Komponenten dargelegt. Schlussendlich wurden die fertige Oberfläche und die getroffenen Designentscheidungen aufgezeigt.

## **7.2 Bewertung der Ergebnisse**

Die funktionalen Anforderungen konnten alle umgesetzt werden und wurden bereits im vorherigen Kapitel erläutert. Im Folgenden wird somit auf die Erfüllung der nicht funktionalen Anforderungen eingegangen. Dies erfolgt anhand der Gegenüberstellung von Anforderungen und den Charakteristika des Systems.

Die erste Anforderung konnte durch den Einsatz eines einzigen Servers eingehalten werden. Weiterhin wurde eine lose Kopplung zwischen Server und Client erreicht. Diese spiegelt sich in den REST-Schnittstellen der jeweiligen Komponenten wieder. Überdies gibt es keine Abhängigkeiten zwischen den Klassen der Darstellung und den Klassen der Geschäftslogik. Ein gewisses Maß an Portabilität wurde vorausgesetzt, damit ein Verlagern des Systems auf andere Instanzen kein Problem darstellt. Dies wurde durch die Verwendung der Web-Archive-Dateien erreicht. Sie ermöglichen den Einsatz auf verschiedenen Tomcat Servern, was sie nicht nur portabel macht, sondern auch verschiedenen Servern einsetzbar macht.

Die wichtigste Anforderung ist eine kurze Antwortzeit des Systems. Tabelle 7.1 zeigt, dass dieser Forderung nachgekommen wird. Ebenfalls deutlich zu erkennen ist die Auswirkung des geänderten Schemas. Der Sprung von 98.000 ms auf 350 ms ist durch die Reduktion in der Abfragekomplexität zu erklären. Die Abfragen erfolgen über wesentlich weniger Tabellen und Spalten als zuvor. Außerdem wird im neuen Schema kein Verbund in der Datenbankabfrage mehr benötigt. Allerdings sind bei derartigen Maßnahmen, wie sie im Schemadesign ergriffen wurden, weitreichende Folgen zu beachten. Eine davon ist eine sehr schlechte Erweiterbarkeit des Schemas. Im momentanen Schema können lediglich Spalten hinzugezogen werden, dessen Inhalt in allen CRM-Objekten vorhanden ist. Außerdem würden für jede weitere Spalte zusätzliche 18 Mio. Werte entstehen. Die Hinzunahme von CRM-Objekt spezifischen Attributen würde ebenfalls zu hohen Änderungsaufwänden führen. Als eine Konsequenz müsste die *data* Tabelle in mehrere Tabellen aufgeteilt werden. Dies würde eine starke Normalisierung des Schemas bewirken und den Einsatz von Verbundoperatoren erfordern. Dadurch würde die Verarbeitungsgeschwindigkeit bei Lesezugriffen steigen. Allerdings wären dennoch wesentlich weniger Verbundoperatoren als im alten Schema nötig. Aufgrund dessen ist trotzdem mit einer deutlich kürzeren Antwortzeit als in CAS genesisWorld zu rechnen.

Versuchskomponente	Zeit in ms
MSSQL Datenbank & Altes Schema	98000
MSSQL Datenbank & Neues Schema	350
H2 Datenbank & Neues Schema	80

Tabelle 7.1: Abfragegeschwindigkeit Vergleich

Nachdem Änderungen betrachtet wurden, die eine Steigerung der Komplexität bewirken, stellt sich folgende Frage: Ist die Datenbank nur aufgrund des geänderten Schemas deutlich schneller? Um dieser Frage nachzugehen wurden Tests durchgeführt. Abbildung 7.1 zeigt die Ergebnisse dieser Testreihen. Alle Testläufe wurden mit einem Rechner durchgeführt. Dieser simulierte mithilfe von Multithreading den gleichzeitigen Zugriff von 100 Benutzern. Die in den Diagrammen angegebene Zeit bezieht sich somit auf die Ausführung aller 100 Abfragen. Jeder simulierte Benutzer führt die auf der y-Achse angegebene Anweisung aus. Beim obersten Balken in (a) sind es beispielsweise 15 Mio. SELECT-Anweisungen pro Benutzer. In (b) hingegen wird die Verarbeitungsgeschwindigkeit bei Updates verglichen. Ein Vergleich der Performance bei Insert-Anweisungen ist in (c) dargestellt.

Die Ergebnisse der Tests zeigen, dass die H2-Datenbank bei den durchgeföhrten Tests deutlich schneller als die MSSQL Datenbank ist. Der H2 ist bei SELECT-Anweisungen um den Faktor 37 schneller. Bei Update-Anweisungen sogar um den Faktor 117. Ebenso bei Insert-Anweisungen, die einen Unterschied um den Faktor 124 aufweisen. Daraus lässt sich ableiten, dass die H2-Datenbank durch ihre In-Memory-Tabellen deutlich an Performance gewinnt. Diese wird zum Teil durch den Verzicht auf Persistenz erlangt. Würde die Datenbank ihre Daten zur Sicherung auf die Festplatte schreiben, müsste bei Schreiboperationen mit Verschlechterungen in der Performance gerechnet werden. Im vorliegenden System, welches fast nur Leseoperationen durchführt, stellt die mangelnde Persistenz al-

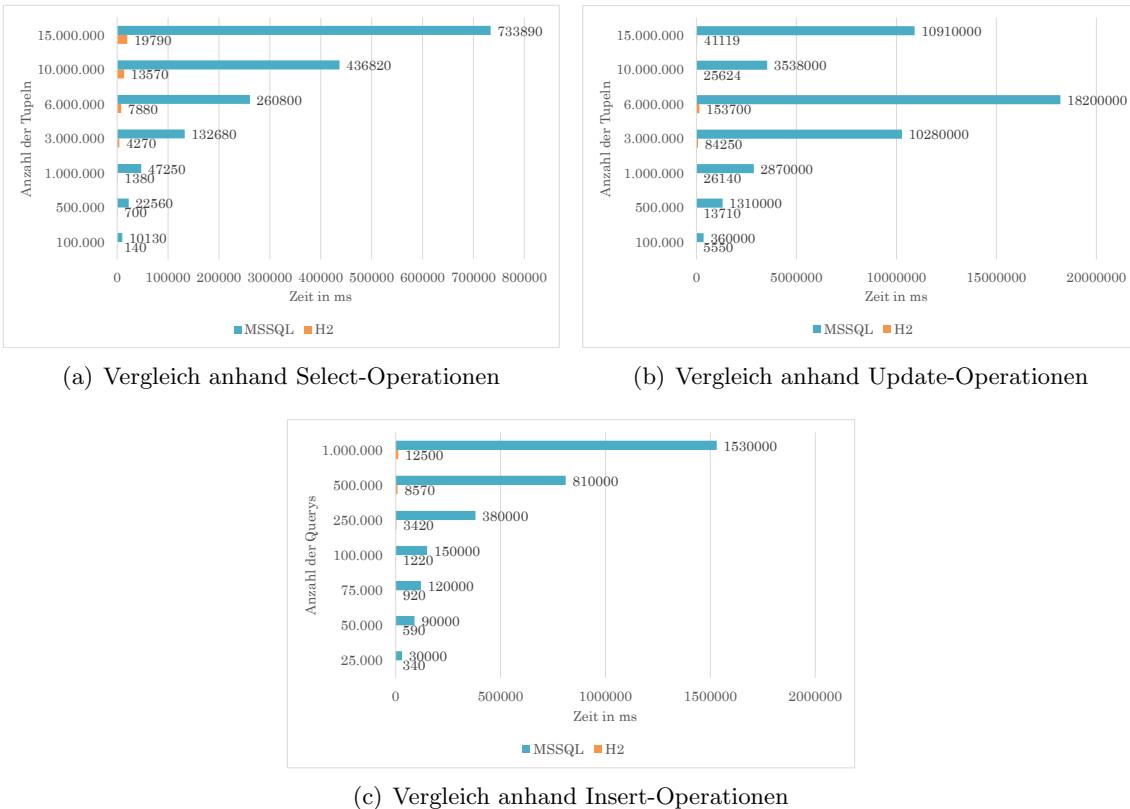


Abbildung 7.1: Vergleich von Antwortzeiten

lerdings kein großes Defizit dar. Ausschlaggebend für die Schnelligkeit ist allerdings die Nutzung des Hauptspeichers als Speichermedium. Dessen Gebrauch könnte allerdings in der Zukunft aufgrund der immer größer werdenden Datenmengen ein Problem darstellen.

### 7.3 Ausblick

Mit der Umsetzung des in der Arbeit beschriebenen Systems steht eine performante Lösung bereit, die eine Bewertung von Beziehungen zwischen Personen aus einem CRM-System ermöglicht.

Die Bewertung der Beziehungen beruht derzeit lediglich auf der Anzahl von CRM-Objekten. Dementsprechend wird nur die Häufigkeit gewertet. Um die Bewertung einer Beziehung genauer feststellen zu können, werden zusätzliche Regeln benötigt. Diese Regeln sollten auf psychologischen Erkenntnissen und Erfahrungswerten aufbauen. Durch Regeln ließe sich die Aussagekraft von Ergebnissen weiter steigern. Beispielsweise sind kommunikative Kontakte wie Telefonate oder E-Mail Verkehr, kein Indikator für Vertrauen. Die Einsicht in vertrauliche Dokumente setzt dagegen eine engere Zusammenarbeit bzw. Vertrauen voraus. Dies sollte somit stärker gewichtet werden.

Neben festen Regeln in der Anwendungslogik könnten Gewichtungsprofile für die Nutzer umgesetzt werden. Ein Profil stellt in diesem Fall eine Voreinstellung der Gewichtungen dar. Demnach würde jedes Profil eine andere Charakteristik in der Abfrage darstellen. Zu diesen Profilen sollte eine Beschreibung beiliegen, die dem Nutzer den Zweck der Gewichtung näher bringt. Dadurch könnten sinnvolle Anpassungen auch durch Mitarbeiter ohne entsprechendes Fachwissen über Beziehungen vorgenommen werden.

Weiterhin könnten durch Vertriebsmitarbeiter mithilfe des Systems individuell unterstützt werden. Dazu würden zusätzliche Informationen über den Wert eines Kunden benötigt.

Unter den Kunden müsste wie bei den Beziehungen ein Ranking aufgestellt werden. Nun könnten die Rankings auf Diskrepanzen verglichen werden. Auf diese Weise könnten zu große Unterschiede im betriebenen Aufwand und gewonnenen Nutzen entdeckt werden. Weiterhin könnten Rankings in umgekehrter Folge durchgeführt werden. Dadurch könnten Kundenbeziehungen auf mangelhafte Kundenpflege hin untersucht werden. Dazu müsste lediglich eine Anpassung an der SQL-Abfrage vorgenommen werden.

Überdies könnten Ergebnisse verschiedener Personen verglichen werden. Der Vergleich könnte dabei unter Personen aus einer Gruppe oder aus selbst erstellten Personenkonstellationen erfolgen. Auf Vertriebsmitarbeiter angewandt könnte überprüft werden, ob die Verteilung der Kunden auf einzelne Mitarbeiter effizient gestaltet ist. Beispielsweise läse sich damit feststellen, ob zu viele Mitarbeiter sich unwissentlich auf denselben Kunden konzentrieren.

Eine andere weiterführende Möglichkeit wären weitere Darstellungen, die Entwicklungen in Beziehungen über die Zeit hinweg zeigen. Liniendiagramme wären dabei eine geeignete Form der Visualisierung, da sich mit ihnen zeitliche Abläufe gut darstellen lassen. Der Datenbestand bietet die Möglichkeiten dies umzusetzen, allerdings müssen entsprechende Abfragen und Darstellungen implementiert werden.

# Literaturverzeichnis

- [AMF06] Daniel Abadi, Samuel Madden und Miguel Ferreira: *Integrating Compression and Execution in Column-oriented Database Systems*. In: *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, Seiten 671–682, New York, NY, USA, 2006. ACM, ISBN 1-59593-434-0. <http://doi.acm.org/10.1145/1142473.1142548>.
- [ASK07] Aditya Agarwal, Mark Slee und Marc Kwiatkowski: *Thrift: Scalable Cross-Language Services Implementation*. Technischer Bericht, Facebook, April 2007. <http://incubator.apache.org/thrift/static/thrift-20070401.pdf>.
- [Bre00] Dr. Eric Brewer: *PODC keynote*. 2000. <http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf>, Online;accessed 27-November-2013.
- [CD10] Kristina Chodorow und Michael Dirolf: *MongoDB - The Definitive Guide: Powerful and Scalable Data Storage*., Seiten 1–10, 16–17, 101–104, 127–129, 143–147. O'Reilly, 2010, ISBN 978-1-449-38156-1.
- [CDG<sup>+</sup>06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes und Robert E. Gruber: *Bigtable: A Distributed Storage System for Structured Data*. In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, Seiten 1–15, Berkeley, CA, USA, 2006. USENIX Association. <http://dl.acm.org/citation.cfm?id=1267308.1267323>.
- [Cor13] Janssen Cory: *SQL Server*. 2013. <http://www.techopedia.com/definition/1243/sql-server>, [Online;accessed 8-November-2013].
- [Cou13] CouchDB: *Technical Overview*. 2013. <http://docs.couchdb.org/en/latest/intro/overview.html>, Online;accessed 27-November-2013.
- [CSA13] CAS-Software-AG: *CAS Products WebServices SDK x5 documentation*. 2013. <https://partnerportal.cas.de/WebServicesSDK/pages/architecture/overview.html>, [Online;accessed 4-November-2013].
- [DG08] Jeffrey Dean und Sanjay Ghemawat: *MapReduce: Simplified Data Processing on Large Clusters*. Commun. ACM, 51(1):107–113, Januar 2008, ISSN 0001-0782. <http://doi.acm.org/10.1145/1327452.1327492>.
- [ESHB11] Shaker H. Ali El-Sappagh, Abdeltawab M. Ahmed Hendawi und Ali Hamed El Bastawissy: *A proposed model for data warehouse {ETL} processes*. Journal of King Saud University - Computer and Information Sciences, 23(2):91 – 104, 2011, ISSN 1319-1578. <http://www.sciencedirect.com/science/article/pii/S131915781100019X>.
- [Gei11] F. Geisler: *Datenbanken: Grundlagen und Design*. mitp, 2011, ISBN 9783826690884. <http://books.google.de/books?id=u7kbZYs0fdcC>, S. 177-180.

- [GL02] Seth Gilbert und Nancy Lynch: *Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services*. SIGACT News, 33(2):51–59, Juni 2002, ISSN 0163-5700. <http://doi.acm.org/10.1145/564585.564601>.
- [Gro14] Marko Groenroos: *Book of Vaadin*, 2014. <https://vaadin.com/download/book-of-vaadin/vaadin-7/pdf/book-of-vaadin.pdf>, [Online;accessed 20-Januar-2014].
- [Hel13] Stefan Helmke: *Effektives Customer Relationship Management : Instrumente - Einführungskonzepte - Organisation*, 2013, ISBN 978-3-8349-4176-3. <http://swbplus.bsz-bw.de/bsz375372644cov.htmhttp://dx.doi.org/10.1007/978-3-8349-4176-3>.
- [HKJR10] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira und Benjamin Reed: *ZooKeeper: Wait-free Coordination for Internet-scale Systems*. In: *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, Seiten 1–11, Berkeley, CA, USA, 2010. USENIX Association. <http://dl.acm.org/citation.cfm?id=1855840.1855851>.
- [KKN<sup>+</sup>08] Robert Kallman, Hideaki Kimura, Jonathan Atkins, Andrew Pavlo, Alexander Rasin, Stanley Zdonik, Evan P. C. Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, John Hugg und Daniel J. Abadi: *H-Store: a High-Performance, Distributed Main Memory Transaction Processing System*. Proc. VLDB Endow., 1(2):1496–1499, 2008, ISSN 2150-8097. <http://hstore.cs.brown.edu/papers/hstore-demo.pdf>.
- [Lam78] Leslie Lamport: *Time, Clocks, and the Ordering of Events in a Distributed System*. Commun. ACM, 21(7):558–565, Juli 1978, ISSN 0001-0782. <http://doi.acm.org/10.1145/359545.359563>.
- [LLS13] Justin J. Levandoski, Per Ake Larson und Radu Stoica: *Identifying hot and cold data in main-memory databases*. 2013 IEEE 29th International Conference on Data Engineering (ICDE), 0:26–37, 2013, ISSN 1063-6382.
- [LM10] Avinash Lakshman und Prashant Malik: *Cassandra: a decentralized structured storage system*. SIGOPS Oper. Syst. Rev., 44(2):1–5, April 2010, ISSN 0163-5980. <http://doi.acm.org/10.1145/1773912.1773922>.
- [Loo01] Peter Loos: *Go to COM : [das Objektmodell im Detail betrachtet; COM von Grund auf; beispielorientiert]*. Go-To-Reihe. Addison-Wesley, München [u.a.], 2001, ISBN 3-8273-1678-2.
- [Mü13] Thomas Müller: *H2 Tutorial*. 0, 2013. <http://www.h2database.com/html/tutorial.html>, [Online;accessed 06-Januar-2014].
- [Pla13a] Hasso Plattner: *A Course in In-Memory Data Management : The Inner Mechanics of In-Memory Databases*, 2013, ISBN 978-3-642-36524-9. <http://dx.doi.org/10.1007/978-3-642-36524-9>.
- [Pla13b] Hasso Plattner: *Lehrbuch In-Memory Data Management : Grundlagen der In-Memory-Technologie*. Springer Gabler, Wiesbaden, c2013, ISBN 978-3-658-03212-8; 3-658-03212-X. [http://deposit.d-nb.de/cgi-bin/dokserv?id=4452889&prov=M&dok\\_var=1&dok\\_ext=htm](http://deposit.d-nb.de/cgi-bin/dokserv?id=4452889&prov=M&dok_var=1&dok_ext=htm), 201309.
- [Rup13] Chris Rupp: *Systemanalyse kompakt*. Springer Vieweg, Berlin, 3. aufl. Auflage, 2013, ISBN 978-3-642-35445-8.

- [RWE13] Ian Robinson, Jim Webber und Emil Eifrem: *Graph databases : [compliments of Neo technology]*. O'Reilly, Beijing, 1. ed. Auflage, 2013, ISBN 978-1-449-35626-2; 1-449-35626-5. [http://deposit.d-nb.de/cgi-bin/dokserv?id=4300566&prov=M&dok\\_var=1&dok\\_ext=htm;http://swbplus.bsz-bw.de/bsz386976589cov.htm](http://deposit.d-nb.de/cgi-bin/dokserv?id=4300566&prov=M&dok_var=1&dok_ext=htm;http://swbplus.bsz-bw.de/bsz386976589cov.htm).
- [Seg13] Karl Seguin: *The Little Redis Book*. 2013. <http://openmymind.net/redis.pdf>, [Online;accessed 11-November-2013].
- [SKRC10] Konstantin Shvachko, Hairong Kuang, Sanjay Radia und Robert Chansler: *The Hadoop Distributed File System*. In: *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, Seiten 1–10, Washington, DC, USA, 2010. IEEE Computer Society, ISBN 978-1-4244-7152-2. <http://dx.doi.org/10.1109/MSST.2010.5496972>.
- [SSH11] Gunter Saake, Kai Uwe Sattler und Andreas Heuer: *Datenbanken : Implementierungstechniken*. mitp, Heidelberg, 3. aufl. Auflage, 2011, ISBN 978-3-8266-9156-0; 3-8266-9156-3. [http://deposit.d-nb.de/cgi-bin/dokserv?id=3872660&prov=M&dok\\_var=1&dok\\_ext=htm;http://d-nb.info/1014629934/04](http://deposit.d-nb.de/cgi-bin/dokserv?id=3872660&prov=M&dok_var=1&dok_ext=htm;http://d-nb.info/1014629934/04), Seiten : 176 - 182.
- [Sto11] Michael Stonebraker: *New SQL: An Alternative to NoSQL and Old SQL for New OLTP Apps*. 0, 2011. <http://cacm.acm.org/blogs/blog-cacm/109710-new-sql-an-alternative-to-nosql-and-old-sql-for-new-oltp-apps/fulltext>, [Online;accessed 23-November-2013].
- [Vai13] G. Vaish: *Getting Started with Nosql*, Seiten 25–49. Packt Publishing, Limited, 2013, ISBN 9781849694995.
- [Vol13a] Project Voldemort: *Voldemort a distributed database*. 2013. <http://www.project-voldemort.com/voldemort/>, [Online;accessed 13-November-2013].
- [Vol13b] VoltDB: *Application Brief*. 2013. [http://voltdb.com/downloads/app-briefs/voltdb\\_transactions.pdf](http://voltdb.com/downloads/app-briefs/voltdb_transactions.pdf), [Online;accessed 14-November-2013].
- [Vol13c] VoltDB: *Technical Overview*. 2013. [http://voltdb.com/downloads/datasheets\\_collateral/technical\\_overview.pdf](http://voltdb.com/downloads/datasheets_collateral/technical_overview.pdf), [Online;accessed 14-November-2013].
- [WH04] Klaus D. Wilde und Hajo Hippner: *Methodisches Vorgehen zur Einführung von CRM*. Springer Gabler, Wiesbaden, 2004, ISBN 978-3-409-12520-8. S. 15.

