



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

Customer-Relationship-Management 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. Dessen Datenbestand reicht von Adressen und Kontaktmöglichkeiten, über Angebote mit Bewertung der Realisierungschancen, bis hin zu kompletten Kundenhistorien. Mitarbeiter erhalten durch die strukturierte Ablage von Informationen ein System mithilfe dessen sie im täglichen Kundendialog unterstützt werden. Jedoch ist es mit Hilfe von CAS genesisWorld nicht möglich, Mitarbeiter mit analytisch gewonnenen Informationen zu versorgen. Beispielsweise wäre eine Abfrage von Beziehungen zwischen Personen im vorliegenden System sehr langsam und komplex. Denn dafür müssten enorme Mengen an verschiedenen Daten zur Beantwortung der Abfrage zusammengetragen werden. Gerade um Wettbewerbsvorteile zu erlangen sind Funktionen die über das Anbieten von operativen Daten hinausgehen von Bedeutung.

In der vorliegenden Arbeit wird aus diesem Grund ein System zur Bewertung von Beziehungen zwischen Personen aus einem CRM-System entwickelt. Hierbei werden Prozesse und Abläufe 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 Datenbanken auf ihre Verwendbarkeit evaluiert. Des weiteren werden Strukturen und Konzepte erarbeitet, wie die Daten übernommen, abgelegt und wieder abgerufen werden können. Schlussendlich werden die Ergebnisse anhand fachlicher und technischer Anforderungen bewertet.

Inhaltsverzeichnis

1 Einführung	1
1.1 Motivation	1
1.2 Zielsetzung	2
1.3 Gliederung der Arbeit	2
2 Grundlagen	5
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 NoSQL 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
3 Systemanalyse	13
3.1 CAS genesisWorld	13
3.1.1 Architektur	14
3.1.2 Präsentationsschicht & Logikschicht	14
3.1.3 Datenhaltungsschicht	15
3.2 Anforderungsanalyse	16
3.2.1 Funktionale Anforderungen	17
3.2.2 Nichtfunktionale Anforderungen	17
3.3 Ermittlung relevanter Daten	18
4 Analyse ausgewählter Datenbanken	21
4.1 Datenbanken	21
4.1.1 CouchDB	21
4.1.2 MongoDB	22
4.1.3 Voldemort	22
4.1.4 Redis	23
4.1.5 HBase	23
4.1.6 Cassandra	23
4.1.7 VoltDB	24
4.1.8 H2	24
4.2 Gegenüberstellung	24

4.3 Auswahl einer Datenbank	26
5 Konzeption	29
5.1 Architektur	29
5.2 Technologien	31
5.3 Datenbankdesign	32
5.3.1 Konzeptionelles Design	32
5.3.2 Zugriffsstrukturen	34
5.4 Extract Transform Load Prozess	35
5.4.1 Extract	35
5.4.2 Transform	36
5.4.3 Load	36
5.5 Synchronisation des Datenbestandes	36
5.6 Darstellungskonzepte	37
6 Umsetzung	41
6.1 Aufbau der Server.war	41
6.2 Aufbau der Client.war	43
6.3 Erzeugung der Abfrage	45
6.4 ETL Prozess	46
6.5 Aktualisierung des Datenbestandes	48
6.6 Oberfläche	49
7 Fazit und Ausblick	53
7.1 Zusammenfassung	53
7.2 Bewertung der Ergebnisse	53
7.3 Ausblick	55
Literaturverzeichnis	57

Abbildungsverzeichnis

2.1	Beispiel einer Spalten-Familie	6
2.2	Objekte in einer Graphdatenbank	7
2.3	Bestandteile einer COM-Architektur	10
3.1	Verknüpfungen in CAS genesisWorld	13
3.2	Schematische Darstellung der Architektur von CAS genesisWorld	14
3.3	Beispiel zur Benachrichtigung von Plugins anhand eines Ablaufs bei einem Update	15
3.4	Funktionsweise von RelationTable anhand eines Beispiels	16
3.5	Auszug aus dem Schema des MSSQL 2008	19
5.1	Konzeptionelle Darstellung der Architektur	30
5.2	Neues Datenbankschema	33
5.3	Sequenzdiagramm für einen neuen Datensatz	37
5.4	Entwürfe für die Oberfläche	38
6.1	Server Klassendiagramm	42
6.2	Client Klassendiagramm	44
6.3	Gewichtung der Zeit	46
6.4	Ausschnitt einer CSV-Datei nach der Extraktion	47
6.5	Klassendiagramm Plugin	48
6.6	Anmeldefenster	49
6.7	Hauptseite der Anwendung	50
7.1	Abfragegeschwindigkeit Vergleich	55

Tabellenverzeichnis

4.1	Gegenüberstellung der Datenbankeigenschaften	25
5.1	Vergleich des Speicherplatzverbrauchs	34
7.1	Abfragegeschwindigkeit Vergleich	54

1. Einführung

1.1 Motivation

Produkte weisen eine stetig steigenden 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 Differenzierung vom Wettbewerb. 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. In den Prozessen entstehen viele Daten bzw. Informationen die in den CRM-Systemen abgelegt werden. Welche Daten dies sind hängt von der jeweiligen Zielsetzung des CRM-Systems ab. Fundamentale Daten wie die Adressen und Kontaktdata der Kunden, sowie komplettete Kundenhistorien (Telefonate, Meetings, E-Mails) sind jedoch in den meisten Systemen vorhanden.

Mit den vorhandenen Informationen können Mitarbeiter im täglichen Kundendialog unterstützt werden. Ein beispielhaftes Szenario wäre dafür ein eingehender Anruf eines Kunden, welcher über eine Absenderkennung des Systems erkannt wird und weitere Informationen zum Kunden bereitstellt. Solche unterstützende Funktionen basieren auf der Nutzung von operativen Daten. Um Mitarbeiter in ihren Tätigkeiten noch besser zu unterstützen sind Informationen nicht nur abzurufen, sondern auch auf Erkenntnisse hin auszuwerten. Dazu können Daten völlig unabhängig von den operativen Geschäftsprozessen, in neue, logische Zusammenhänge gesetzt werden. Versucht man dies auf dem Datenbestand der operativen Geschäftsprozesse durchzuführen, trifft man schnell auf zahlreiche Probleme. Eines der größten Probleme ist die Steigerung des Aufwands in der Ermittlung von Informationen, da Daten für die Beantwortung der neuen Fragestellungen suboptimalen vorliegen. Ein solches Problem ist auf die Form der Datenhaltung zurückzuführen. Möchte man beispielsweise die Beziehungen zwischen Personen analysieren, müssen zuerst Daten identifiziert werden, die dies ermöglichen. Sind die Daten gefunden, werden zusätzlich noch komplexe Abfragen benötigt, um die gewünschten Ergebnisse zu erzeugen. Auf dem Markt gibt es Produkte die sich der Lösung dieser Probleme annehmen. Sie sind allerdings auf die Beantwortung allgemeiner Fragestellungen ausgelegt. Für kleinere Unternehmen oder spezielle Anforderungen bietet sich daher eigene Entwicklungen an. Die vorliegende Arbeit setzt an diesem Punkt an und zeigt die Entwicklung eines Systems zur Beantwortung folgender Fragestellung: Wie lässt sich ein System zur Bewertung von Beziehungen, zwischen Personen aus einem CRM-System umsetzen?

1.2 Zielsetzung

Anknüpfend an die zuvor aufgeworfene Frage wird im Rahmen der Arbeit ein analytisches Informationssystem entwickelt. Es soll eine Bewertung von Beziehungen zwischen Personen in einem CRM-Systems ermöglichen. Neben der Funktionalität des Systems, soll eine hohe Abfragegeschwindigkeit (unter 1 Sek.) erreicht werden. Das zu entwickelnde System soll zwar auf dem Datenbestand des CRM-Systems basieren, allerdings trotzdem unabhängig davon funktionieren. Aufgrund dessen soll ein neues System entwickelt werden. Damit erhofft man sich Altlasten des bestehenden Systems zu umgehen und bessere Resultate zu erzielen.

Dabei sollen Entwicklungen der letzten Jahre, wie NoSQL- und In-Memory-Datenbanken untersucht werden. Zur Auswahl einer geeigneten Datenbanken sollen deren Eigenschaften untersucht werden. Neben einer Datenbank sind Technologien für die Kommunikation und Anwendungslogik festzulegen. Weiterhin sollen alle relevante Daten ermittelt werden, die zur Erfüllung der Anforderungen notwendig sind. Überdies soll ein ETL-Prozess entworfen werden, mithilfe dessen eine Übertragung der Daten zwischen den Datenbanken möglich wird. Außerdem sollen die Funktionen des Anwendungsservers über Schnittstellen ansprechbar sein. Zur Gewährleistung der Aktualität von Daten sollen Lösungswege zu dessen Sicherstellung erarbeitet werden. Um die Ergebnisse der Anwendungslogik für den Nutzer grafisch aufzubereiten, soll auch ein Client implementiert werden. Die Oberfläche des Clients sollte 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 vermittelt. Zuerst wird auf den Begriff NoSQL aus dem Bereich der Datenbanken eingegangen. Dabei werden die unterschiedlichen Typen von NoSQL-Datenbanken erklärt. Nachdem ein Überblick über die Ausprägungen von NoSQL Datenbanken gewonnen wurde, werden die einschlägige Begriffe im Bereich NoSQL erläutert. Die Begriffe werden im Voraus behandelt, da sie in der Evaluation von Datenbank auftauchen. Neben NoSQL hat der Terminus In-Memory-Datenbank in den letzten Jahren an Interesse gewonnen. Daher wird ein kurzer Einblick in die Thematik gegeben. Neben den Datenbanken wird das Component Object Model erläutert. Grundlagen in diesem Bereich verschaffen einen Einblick in die Technologie des CRM-Systems, welche für spätere Betrachtungen benötigt werden.

Analyse In Kapitel 3 wird die Architektur, sowie einzelne relevante Bestandteile des vorhandenen CRM-Systems untersucht. Außerdem werden in dem Kapitel die Anforderungen an das neue System erhoben. Überdies wird das umzusetzende Szenario näher beschrieben. Weiterhin werden die im neuen System benötigten Teile des Datenbestandes ermittelt.

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 die Gegenüberstellung festgelegt, anhand derer ein Vergleich durchgeführt wird. Anschließend wird unter Beachtung der Anforderungen eine Datenbank ausgewählt und dargelegt, warum sich gegen die Anderen entschieden wurde.

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

Umsetzung In Kapitel 6 wird auf die Umsetzung der Planungen eingegangen. Dabei wird auf einer Abstraktionsebene beschrieben, wie die Implementierungen arbeiten. Es wird bewusst auf den Einsatz von Quelltexten verzichtet, um eine besseres Verständnis über die Logik und die Struktur zu gewinnen.

Ergebnis Die letztendlich abschließende Betrachtung fasst die Ergebnisse der vergangenen 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. Weiterhin schließt diese Arbeit mit einem Ausblick auf das weitere Vorgehen.

2. Grundlagen

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

2.1 NoSQL - Eine Einführung

NoSQL ist ein Begriff der weder für eine bestimmte Datenbank, noch für einen bestimmten Datenbanktyp verwendet wird. Der Terminus NoSQL fasst lediglich Datenbanken zusammen, die nicht dem Modell der relationalen Algebra folgen. Eines haben sie jedoch gemeinsam: Die treibende Idee, die zu Ihrer Entwicklung geführt hat. Die Entstehung der NoSQL Datenbanken ist auf die schlechte horizontale Skalierbarkeit von relationalen Datenbanken zurückzuführen. Verfügbarkeit und Skalierbarkeit 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 dabei 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. Obwohl die Daten schemalos sind, 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 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, weshalb sie auch Wide Columns Stores genannt werden. 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.

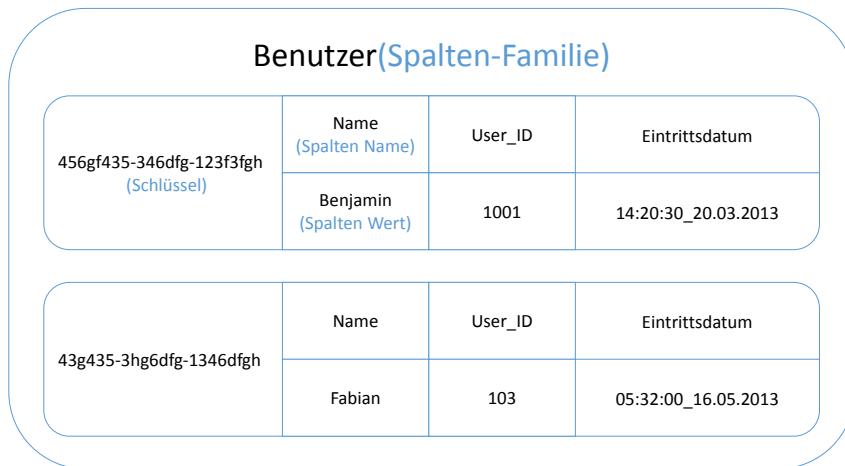


Abbildung 2.1: Beispiel einer Spalten-Familie

Diese Architektur hat mehrere Vorteile. Meist weisen Werte in Spalten eine geringe Entropie auf, was sie besonders für Kompressionsverfahren geeignet macht. Die Verarbeitung der Anfragen wird ebenfalls beschleunigt, da keine unnötigen Informationen gelesen werden. Dies trifft in der Regel für Lese- und Schreibprozesse zu, wenn es um eine einzelne Spalte geht (in der Regel ein disk-seek). Die Geschwindigkeit nimmt beim Zugriff jedoch mit steigender Anzahl von Spalten ab.

2.1.3 Key-Value-Store

Key-Value-Stores sind die einfachsten NoSQL-Datenbanken. Sie ermöglichen die Speicherung von Werten mit einem Schlüssel. Im Gegensatz zu relationalen Datenbanken, haben Key-Value-Stores keine Kenntnis über die Daten der Werte und sind daher schemafrei. Grundsätzlich trifft das auf die meisten Umsetzungen von Key-Value-Stores zu. Redis und andere Umsetzungen sind allerdings in der Lage, strukturierte Daten abzulegen und Ihre Felder zu indexieren. Dadurch wird eine Steigerung der Komplexität solcher Datenbanken erreicht, was jedoch nicht dazu führt, dass sie Verbund- oder Aggregat-Operatoren beherrschen.

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. Graph-Datenbanken 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 NoSQL Theoretische Grundlagen

Im Nachfolgenden werden Begriffe und Verfahren erläutert, die durch die NoSQL Bewegung geprägt wurden.

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

Fragmentierung Fragmentierung in der Datenbank ist der Zustand, bei dem die Daten in mehrere Fragmente aufgeteilt wurden. 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.

Eventuelle Konsistenz Später in diesem Kapitel wird das CAP-Theorem eingeführt, welches besagt, dass verteilte Datenbanken entweder stark konsistent oder verfügbar sein können. Folglich können die meisten NoSQL Datenbanken nur eventuelle Konsistenz bieten, eine abgeschwächte Art der starken Konsistenz. 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, wobei jeder Prozess zuerst eine exklusive Sperre auf einem Datenelement anfordern muss, bevor es gelesen oder aktualisiert werden kann. Zu diesem Zweck werden intern verschiedene Versionen eines Objektes gehalten.

MapReduce MapReduce ist ein von Google entwickeltes Programmiermodell für verteilte Berechnungen und ist 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. Es kann verwendet werden, um Aufgaben auf großen Datenmengen durchzuführen, die zu groß für eine einzelne Maschine zu handhaben wären.

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 gegen temporäre Verbindungsprobleme und ermöglicht es, Partitionen über Knoten zu trennen. 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. Jedoch kann es vorkommen, dass dies in einer verteilten Umgebung nicht möglich ist. Ursachen dafür können Verbindungsfehler oder andern temporäre Hardwareprobleme sein.

2.2 In-Memory-Datenbanken

Eine In-Memory-Datenbank (IMDB) ist ein Datenbankmanagementsystem, dass in erster Linie den Hauptspeicher als Medium für die Datenablage verwendet. Eine IMDB wird auch als Hauptspeicher-Datenbank (MMDB) oder Echtzeit-Datenbank (RTDB) bezeichnet. IMDBs sind schneller als die Festplatten optimierte Datenbanken, denn sie führen weniger CPU-Befehle beim Lesen und Schreiben aus und ihre internen Optimierungsalgorithmen sind viel einfacher gestaltet. Einsatz finden sie vor allem in Anwendungen, bei

denen Reaktionszeit von entscheidender Bedeutung ist. Mehrkernprozessoren, 64-bit Architekturen und gesunkene RAM Preise stellen die treibenden Faktoren in der Entwicklung solcher Systeme dar [Pla13a].

Die hohe Performance dieser Systeme kann nicht nur durch verlagern der Daten in den Hauptspeicher erreicht werden. Vielmehr müssen bisherige Konzepte im Datenbankentwurf neu überdacht werden. In herkömmlichen Datenbanken ist der Speicherverbrauch kein relevanter Faktor. In IMDBs hingegen ist der Einsatz von Speicherplatz sparenden 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 geringen 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 Datenvorhaltungsgründen aufbewahrt. Die zur Datenaufbewahrung benötigte Hauptspeicherkapazität, kann durch solche Regeln stark reduziert werden.

In-Memory-Datenbanken die den relationalen Ansatz verfolgen besitzen zudem geänderte Abfrageoptimierer. In herkömmlichen RDBMS sind Lese- und Schreiboperationen eine der wichtigsten Faktoren zur Bestimmung des optimalen Abfrageplans. Sie spielen jedoch eine stark untergeordnete Rolle in IMDB. Im Gegenzug nimmt die Reduktion von CPU-Zyklen einen höheren Stellenwert ein.

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 Veränderungen an Daten zwischen Snapshots nicht verloren gehen, werden sie in Log Dateien zwischengespeichert. Zusammen mit den Snapshots dienen sie als Grundlage für die Datenwiederherstellung.

An dieser Stelle schließt die Einführung im Bereich der Datenbanken. Im folgenden wird auf das Component Object Model eingegangen, um eine Grundlage für die spätere Betrachtung der CAS genesisWorld Komponenten zu bilden.

2.3 Component Object Model

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 objekt-orientierte Sprache, sondern um einen Standard handelt. Er definiert nicht die Sprache, Struktur oder Implementierungsdetails. Jeder dieser Entscheidungen werden 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 Rechner laufen. Die Umsetzung in verschiedenen Sprachen ist durch die Umsetzung der Kommunikation in binären Maschinencode möglich. Das führt dazu, dass COM des öfteren als binärer Standard referenziert wird.

COM bietet die Möglichkeit auf viele der Windows-Funktionen direkt zuzugreifen. Des weiteren ist COM die Basis für die OLE-Automation¹(Object Linking and Embedding) und ActiveX². Die Verwendung des COM-Standards bietet folgende Vorteile:

- Sprachunabhängig
- Versionsunabhängig
- Plattformunabhängig
- Objektorientiert
- Ortsunabhängig
- Automatisiert

2.3.1 Architektur

COM basiert auf dem Client/Server-Prinzip. 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: Bestandteile einer COM-Architektur

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 werden über Typbibliotheken veröffentlicht oder stehen in Form von Beschreibungen in der Interface Definition Language(IDL) zu Verfügung. Einem Client steht außerdem die Möglichkeit einer Abfrage zur Verfügung, mithilfe er feststellen kann ob ein Objekt das angefragte Interface unterstützt. Dabei wird lediglich die Abfrageoperation mit einer Globally Unique Identifier (GUID) als Parameter auf dem ausgewähltem Objekt ausgeführt. Falls das Objekt das geforderte Interface unterstützt, liefert es den entsprechenden Pointer zur Methode zurück.

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

²ActiveX 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

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, mit in den Prozess der Anwendung (COM-Client) geladen 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 eingesetzt wird, sobald ein Netzwerk sich zwischen Client und Server befindet. Dabei wird DCOM (Distributed COM) verwendet, die eine spezielle Variante von COM darstellt. Unterscheiden tut sich DCOM lediglich durch den Einsatz eines vollständigen RPC-Protokolls, bei dem ein Protokollstack vorgeschaltet wird.

2.3.4 COM-Schnittstelle

COM ist eine Technologie die es Objekten ermöglicht über Prozess- und Rechnergrenzen hinweg so einfach wie in einem einzigen Prozess zu interagieren. COM 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, die eine Klasse implementiert. Eine Schnittstelle allerdings muss nicht unbedingt alle Funktionen unterstützten die eine Klasse implementiert. Eine Schnittstellenimplementierung wird mit einem Objekt verbunden, sobald eine Instanz des Objekts erzeugt wurde und die Implementierung die Dienste des Objekts bereitstellt. Zum Beispiel definiert ein hypothetisches Interface namens ISquare, eine Methode A. Diese Methode A soll das Quadrat einer Zahl zurückliefern. Ein Programmierer verwendet vielleicht integer als Datentyp und ein anderer Double. Auch das Quadrat könnte durch Multiplizieren zweier Zahlen berechnen werden oder durch rufen einer Funktion. Das alles spielt für den Client keine Rolle, den der Verweis des Pointers im Speicher den er letztendlich benutzt, ist durch das Interface definiert und ändert sich nicht.

Eine typische Vorgehensweise für die Entwicklung von Interfaces ist es Funktionalitäten und Daten in logische Mengen zu gruppieren, die der Lösung eines Problems dienen. 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. Dieser Prozess wird Interface basiertes Programmieren genannt. Zuletzt wird eine COM-Anwendung als eine Framework oder eine Hierarchie aller COM-Objekte umgesetzt.

2.3.5 COM-Objekte

Ein COM-Objekt bietet Funktionen des COM-Servers über ein Interface an. Durch die Implementierung *IClassFactory.CreateInstance()* kann eine Instanzierung 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 Instanzierung eines Objektes wird eine Referenzzählung hochgezählt. Dieser wird durch rufen 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 den Client und Server-Programmen. RPC Anwendungen benutzen die ACF-Datei, um die Eigenschaften von Interfaces, die spezifisch für die Hardware und Betriebssystem-Operatoren sind, zu beschreiben.

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. Im Rahmen der Analyse ist die Kontextabgrenzung eines der wichtigsten Bestandteile. Dabei wird eine Abgrenzung zwischen dem Umfang und der Umgebung des Systems vorgenommen. Zuerst wird in Abschnitt 3.1 eine Ist-Analyse durchgeführt. In Abschnitt 3.2 wird auf die Anforderungen, die an das System gestellt werden eingegangen. Aufbauend auf den Anforderungen werden in Abschnitt 3.3, die für die Umsetzung relevanten Daten 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, sowie als mobile Version 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-Applikationsservers, 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. Als Basisdienste stehen der UserService und der DataService zu Verfügung. Für die Anmeldung und Rechteverwaltung ist der UserService zustän-dig. Der DataService hingegen, als zentraler Dienst für den Zugriff auf die CAS genesis-World Daten. 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

Server-SDK-Plugins 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 CAS genesisWorld registriert. Der Server delegiert bei einer Datenoperation den Aufruf an die für den jeweiligen Datensatztypen registrierten Plugins. In Abbildung 3.3 ist ein Beispiel des Vorgangs dargestellt.



Abbildung 3.3: Beispiel zur Benachrichtigung von Plugins anhand eines Ablaufs bei einem Update

Im Allgemeinen stehen in den COM-Schnittstellen der Plugins, jeweils alle Felder eines Datensatz-Typen zur Verfügung, sowie die individuelle Teilmenge der Felder mit neuen Werten. 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 Werteteilmenge des aktuell verarbeiteten Datensatzes kann verändert, d.h. erweitert oder reduziert werden und die Werte selber sind änderbar. 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.

3.1.3 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, der mehrere Funktionen zum SQL-Standard hinzufügt [Cor13]. Weiterhin unterstützt MSSQL standardisierte Datenbankschnittstellen, wie Open Database Connectivity (ODBC) und Java Database Connectivity (JDBC).

In den meisten relationalen Datenbanken werden Beziehungen über Primär- und Fremdschlüssel abgebildet. In der CAS genesisWorld Datenbank werden nur Primärschlüssel eingesetzt. Die Beziehungen werden nicht wie sonst in mehreren Zwischentabellen realisiert, sondern in einer einzigen Tabelle, die *TableRelation*. Abbildung 3.4 zeigt eine beispielhafte, schematische Darstellung der *TableRelation*.

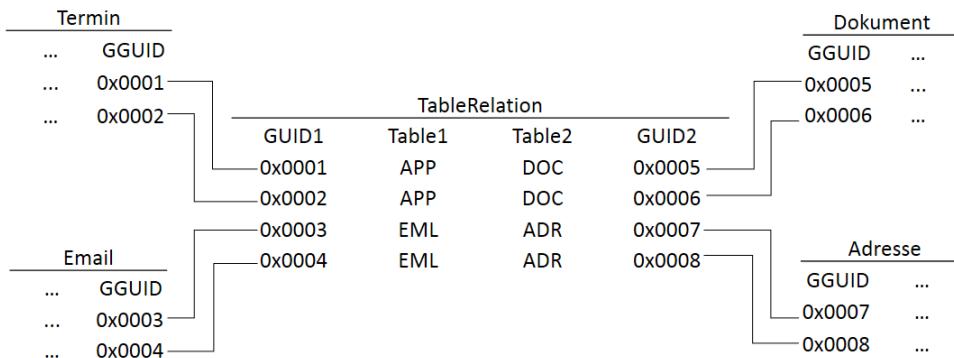


Abbildung 3.4: Funktionsweise von RelationTable anhand eines Beispiels

Die Spalten *GUID1* und *GUID2* beinhalten die jeweiligen Primärschlüssel der in Beziehung zu setzenden Tabellen. Als Zuordnungsmerkmal für die *GGUIDs* zu ihren Tabellen dienen die Spalten *TableSign1* und *TableSign2*, deren Inhalt aus Kürzeln der Tabellennamen besteht. Möglich wird die Verknüpfung verschiedener Tabellen durch einen besonderen Primärschlüssel. Dieser wird für jede neue Zeile generiert und ist in der gesamten Datenbank eindeutig. Er wird als Genesis Global Unique Identifier (*GGUID*) bezeichnet und ist eine global eindeutige Zahl bestehend aus 128 Bit. Jede Tabellen besitzen eine Spalte mit *GGUIDs*, welche eine Datenintegrität in der gesamten Datenbank sicherstellt.

3.2 Anforderungsanalyse

Während der Anforderungsanalyse wird ermittelt welche Eigenschaften und Fähigkeiten das System zur Erreichung der Ziele benötigt. Wir unterscheiden bei der Einteilung der Anforderungen zwischen Funktionalen und Nichtfunktionalen. Beim erst genannten wird die Funktionalität des zu erstellenden Systems beschrieben, wohingegen alle anderen Anforderungen unter letzteres fallen.

Bevor wir auf die funktionalen und nichtfunktionalen Anforderungen eingehen, wird das umzusetzende Szenario näher beschrieben. Mit dem zu entwickelndem System soll eine Bewertung der Beziehung zwischen Personen aus CAS genesisWorld ermöglicht werden. Indessen soll ermittelt werden, welche Personen die ausgeprägteste Beziehung zu einer vorher bestimmten Person besitzen. Die Bewertung der Ausprägung basiert dabei auf der Häufigkeit des Kontakts mit der Person. Die Häufigkeit des Kontakts wird dabei anhand von fünf verschiedenen Merkmalen ermittelt. Zu einem wird der E-Mail-Verkehr unter den Personen für die Betrachtung herangezogen. Überdies werden die Telefonate unter den Personen miteinbezogen. Außerdem spielen nachvollziehbare Treffen (Termine) zwischen

den Personen eine Rolle. Unter Personen geteilte Dokumente werden auch als Merkmal festgesetzt. Das letzte Merkmal stellt die Verkaufschance gegenüber einem Kunden dar. Wie ausgeprägt letztendlich die Beziehung zu einer anderen Person ist, wird anhand der Anzahl solcher Merkmale ermittelt. Auf die fünf Merkmale wird im weiteren Verlauf der Arbeit nur noch mit dem Begriff "Verbindungsmerkmale" verwiesen. Weiterhin ist die Betrachtung nicht auf die gesamte Dauer des Kontakts vorgesehen, sondern auf festgelegte Zeitspannen. Beispielsweise sollte es möglich sein, die Ergebnisse auf den Zeitraum vom 01.02 bis 10.08.2013 einzuschränken. Zusätzlich sollten weitere Eingrenzungen möglich sein, die im folgenden Abschnitt beschrieben werden.

3.2.1 Funktionale Anforderungen

Folgende funktionale Anforderungen wurden erhoben:

- Ermittlung der Anzahl von Verbindungsmerkmalen ausgehend von einer Person, zu allen anderen Personen im CRM-System
- Ranking der Personen basierend auf der Summe von Verbindungsmerkmalen, die von der Ausgangsperson ausgehen
- Abfrageergebnis soll die Summe der gesamten Verbindungsmerkmale zu den jeweiligen Person enthalten, sowie die Summe der einzelnen Verbindungsmerkmale
- Begrenzung des Ergebnisses hinsichtlich der Anzahl an Personen
- Möglichkeit zum eingrenzen des Ergebnisses auf einen festen Zeitraum
- Ein- und Ausblenden von Suchkriterien, ohne eine neue Anfrage senden zu müssen
- Gewichtung der Suchkriterien durch den Nutzer
- Gewichtung von Zeitspannen durch den Nutzer
- Filterung der Ergebnismenge durch:
 - Ausschließen von Personen oder Eingrenzen auf Personen
 - Städte und/oder Länder der Personen
 - Verringern auf Personen, die einem Unternehmen zugeordnet sind
 - Beschränken auf Kontakt Personen von Unternehmen
 - Begrenzen auf Kontakt Personen die keinem Unternehmen angehören
 - Vermindern um Personengruppen

3.2.2 Nichtfunktionale Anforderungen

Folgende nicht nichtfunktionale Anforderungen wurden erhoben:

- Eine Rechnerinstanz für Datenbankserver und Applikationsserver
- Sehr hohe Verarbeitungsgeschwindigkeit (unter einer Sekunde)
- Lose Kopplung (zwischen Logik und Darstellung)
- Portabilität
- Graphische Darstellung des Ergebnisses
- keine zusätzlichen Kosten

3.3 Ermittlung relevanter Daten

Die Datenbank der CAS Software AG umfasst 398 Tabellen, die zusammen wiederum 11.620 Spalten beinhalten. Aufgrund einer fehlenden Dokumentation über die Umsetzung der Anwendungsschicht und der Abwesenheit von fehlenden, definierten Beziehungen innerhalb der Datenbank wurde ein eigenes Verfahren zur Ermittlung der Beziehungen entwickelt.

Für den Ausgangspunkt der Suche wurde eine Tabelle namens *SysUser* verwendet. Sie beinhaltet jeden Benutzer des Systems. Ihre Eignung beruht auf der Annahme, dass bei der Bewertung von Beziehungen zwischen Personen, die Person selbst dabei immer den Ausgangspunkt der Suche darstellt. Aus Datenbanksicht bedeutet dies, dass zuerst eine Tupel der *SysUser* Tabelle mit ihrer *GGUID* herangezogen wird. Die *GGUID* ist dabei der erste Wert nachdem in der gesamten Datenbank gesucht wird. Sobald alle Tabellen gefunden wurden, die den Wert beinhalten, werden deren *GGUIDs* für die weitere Suche verwendet. Die Menge der Suche wird nach jedem Schritt, um die Teilmenge der bereits gefundenen Tabellen verringert. Die Suche wird abgebrochen, sobald die Suchmenge keine Werte mehr aufweist oder keine Tabellen mehr mit den entsprechenden Werten gefunden wurden. Durch dieses Vorgehen erhält man am Ende alle Beziehungen, aufbauend auf der Annahme, dass die *GGUID* als Wert für die Referenzierung genutzt wurde. In Abbildung 3.5 ist ein schon auf das Wesentliche reduzierter Ausschnitt des Ergebnisses zu sehen.

Von den ursprünglichen 398 Tabellen sind nur noch 17 übrig geblieben, auf die im weiteren Verlauf eingegangen wird. Alle Tabellen enthalten in der ursprünglichen Form wesentlich mehr Spalten und wurden der Übersicht halber entfernt. Tabelle *SysUser* besitzt drei Spalten die von Bedeutung sind. Eine davon ist die *GGUID*, die im folgenden nicht weiter erwähnt wird, da sie jede Tabelle enthält. Die *OID* wird für jeden Nutzer einmalig vergeben und wird in anderen Tabellen als Zuordnungsmerkmal verwendet. *LoginName* ist wie der Name schon sagt, der Benutzername des Nutzers und kann beim anmelden auf der Oberfläche verwendet werden.

Aufgrund der Anforderung von Filterung der Ergebnisse durch Gruppen, wurden die Tabellen *SysGroupMember* und *SysGroup* hinzugezogen. *SysGroup* besitzt eine *GID*, die sich bei Gruppen genauso verhält, wie die *OID* bei Personen. Das Attribut *GroupName*, wird für die Anzeige an der Oberfläche benötigt. Mithilfe dessen der Nutzer nachvollziehen kann, welche Gruppe er gerade ausgewählt hat. *SysGroupMember* stellt die Auflösungstabelle zwischen *SysUser* und *SysGroup* dar. Die Spalte *GroupID* beinhaltet Werte aus der *GGUID* Spalte der *SysGroup* Tabelle. *MemberID* folgt dem selben Ansatz, allerdings werden die *GGUIDs* der *SysUser* Tabelle verwendet. Hinsichtlich der Logikschicht wird die Spalte *InsertTimestamp* für Veränderungen innerhalb der Gruppen benötigt.

Die *Address0* Tabelle wird für die Filter Anforderungen benötigt. *Town1* und *Country1* geben die Stadt sowie das Land an, in der die Person ansässig ist. Zur Ermittlung, ob eine Person eine Kontaktperson, Mitarbeiter oder eine Firma repräsentiert, werden die Attribute *gwIsContact*, *gwIsEmployee* und *gwIsCompany* benötigt. *ChristianName* und *Name* beinhalten den Vor- und Nachname der Person, welche für die Zuordnung der Ergebnisse an der Benutzeroberfläche hilfreich sind. Im voraus ist zu sagen, dass nicht alle Telefongespräche über die dafür bestimmten Tabellen ermittelt werden können. Deswegen werden die Attribute *PhoneFieldStr1-10* benötigt, mithilfe derer eine Zuordnung möglich ist.

Die *TableRelation* enthält, wie in Abschnitt 3.1.3 behandelt, die Beziehungen der Datenbank. Beispielsweise kann für die *GUID1* die *GGUID* der *SysUser* verwendet werden. Mithilfe der *GUID2* können die mit der Person verknüpften Termine, Verkaufschancen, Telefonate, Dokumente und E-Mails anschließend bestimmt werden. *TableSign1* und *TableSign2* können zur Identifikation der jeweiligen Tabellen verwendet werden.



Abbildung 3.5: Auszug aus dem Schema des MSSQL 2008

GWOpportunity0 enthält alle Informationen über die jeweiligen Verkaufschancen. Das Attribut *InsertTimestamp* wird zur Feststellung des Erzeugungszeitpunktes benötigt. *Start_dt* und *end_dt* legen den Zeitraum der Verkaufschance fest. Der Besitzer einer Verkaufschance wird über die *AccountGUID* bestimmt. Mit *EMailStore0*, *Document0* *Appointment0* und *gwPhoneCall0* verhält es sich wie mit *GWOpportunity0*. Bei *EMailStore0* ist zu erwähnen, dass *SendDate* zur zeitlichen Einordnung verwendet werden kann. Das Attribut *Dialed-Number* der Tabelle *gwPhoneCall0* wird zum Vergleich mit der in der Adresse hinterlegten Telefonnummer benötigt. Zur Überprüfung ob das Telefonat über einen Tag hinausging,

wird das Attribut *duration* herangezogen.

Bis jetzt galt die Annahme, dass alle Beziehungen über die *TableRelation* bestimmt werden können. Dies trifft allerdings nicht für alle Beziehungen zu. Dort werden nur, die an der Oberfläche manuell verknüpften Objekte aufbewahrt. Die restlichen Beziehungen können aus den ORel-Tabellen ermittelt werden. Jeder dieser Tabellen enthält eine *OID* bzw. *GID*, die zur Bestimmung der beteiligten Personen und/oder Gruppen dienen.

Eine Betrachtung basierend auf Zeitspannen, impliziert Veränderungen der Daten über die Zeit gesehen. Um diese Änderungen zu erfassen wird die Tabelle *ChangeLogBook* benötigt. *NewFieldValue* enthält den neuen Wert eines Feldes. Wohingegen *OldFieldValue* den alten Wert des Feldes repräsentiert. Die Spalte des geänderten Wertes ist in *FieldName* hinterlegt. Der Name der Tabelle ist der Spalte *TableName* zu entnehmen. Die Referenzierung auf eine Tupel wird in der Spalte *TableGUID* vorgenommen. Aus Gründen des Speicherplatzverbrauchs werden nur varchar Datentypen bei Zeichenfolgen verwendet. Varchar ist jedoch auf 4000 Zeichen limitiert. Falls Zeichenfolgen diese Grenze überschreiten, werden diese in der Tabelle *MemoLogBook* abgelegt. Dort wird der Datentyp Text verwendet, der eine maximale Zeichenfolglänge von $2^{31} - 1$ (2.147.483.647) erlaubt.

4. Analyse ausgewählter Datenbanken

Ein Ziel der Arbeit ist hohe Geschwindigkeiten in der Beantwortung der Benutzeranfragen zu erreichen. Die maßgebende Komponente in diesem Fall ist die Datenbank. Sie führt die zeitintensiven Ermittlungen, Berechnungen und Filterungen 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 Datenbanken ausgewählt wurde.

4.1 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, 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 hat Feldwerte, die Datentypen (Text, numerisch oder boolean) oder Datenstrukturen (ein Dokument oder Liste) beinhalten. Abfragen werden mit views zum Filtern der Dokumente ausgeführt. In CouchDB werden für Indizes B-Bäume verwendet, sodass die Ergebnisse sortiert und Wertebereich-Anfragen ausgeführt werden können. Abfragen können parallel über mehrere Knoten mit einem MapReduce Mechanismus verteilt werden. CouchDB erreicht Skalierbarkeit durch asynchrone Replikation, nicht durch Fragmentierung. Lesezugegriffe können auf beliebigen Server stattfinden, wenn Aktualität keine Rolle spielt. Updates hingegen müssen an alle Server weitergegeben werden. CouchDB unterscheidet sich von anderen Systemen durch die Akzeptanz von eventueller Konsistenz. CouchDB implementiert MVCC auf einzelne Dokumente, mithilfe einer Sequenz-ID, die für jede Version eines Dokuments generiert wird. CouchDB benachrichtigt eine Anwendung, wenn jemand anderes das Dokument aktualisiert hat, seitdem es zuletzt auf der Datenbank abgelegt wurde. Die Anwendung kann dann versuchen, die Updates zu kombinieren oder das Update zu wiederholen, um die Daten zu überschreiben. CouchDB erfüllt damit im lokalen Einsatz

die ACID-Eigenschaften. Jede Transaktion ist eine in sich abgeschlossene Operation, die entweder ganz oder gar nicht ausgeführt wird. Es treten keine Seiteneffekte zwischen den Anfragen auf. Außerdem wird die Datenbank immer in einem konsistenten Zustand hinterlassen.

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, sind lockless, und bieten einen Abfragemechanismus für Dokumente. Es gibt jedoch wichtige Unterschiede:

- MongoDB unterstützt automatische Fragmentierung, die Dokumente über Server verteilt.
- Dynamische Abfragen mit automatischer Verwendung von Indizes werden von MongoDB unterstützt. In CouchDB werden, durch das Schreiben von map-reduce-views, Daten indiziert und gesucht.
- CouchDB nutzt MVCC bei Dokumenten, wohingegen MongoDB atomare Operation auf Feldern nutzt

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 verschlüsseln die lokalen Dokumentdatenstrukturen in das BSON Format und senden es 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 über einen benutzerdefinierten Schlüssel 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), welche ein hoch skalierbares Speicher-System zur Verfügung stellt. Voldemort repliziert sich durch automatisches partitionieren und anschließendes verteilen der Daten auf multiple 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. Jedoch 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. Der Parameter Version, im DELETE-Operator, dient der Unterscheidung der Datensätze und ist auf das Verfahren zur Gewährleistung der Konsistenz zurückzuführen. Zur Gewährleistung der eventuellen Konsistenz werden Timestamps und die Vector Clock Technik eingesetzt. Neben der eventuellen Konsistenz 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. Entweder durch Momentaufnahmen der Daten und anschließendes ablegen auf der Festplatte, in regelmäßigen Abständen oder durch aufzeichnen eines Logs mit allen ausgeführten Operationen. 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 locking um.

4.1.5 HBase

HBase ist eine verteiltes, Open Source Column Store Datenbanksystem, welches auf Google's BigTable basiert [CDG⁺06]. HBase läuft auf Apache Hadoop und Apache ZooKeeper [HKJR10] und verwendet das Hadoop Distributed Filesystem (HDFS) [SKRC10], um Störung-Toleranz und Replikation zu bieten. Zeilen Operationen sind in HBase atomar, mit Sperren auf Zeilenebene und Transaktionen. Partitionierung und Verteilung sind transparent, da es kein clientseitiges Hashing oder feste Schlüsselräume wie in einigen NoSQL-Systemen gibt. 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 API zugegriffen werden. Sie können jedoch auch als Ein- und Ausgangsdaten für MapReduce-Jobs in Hadoop verwendet werden. 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 Schlüsselspalte indiziert und in Spalten-Familien gruppiert.

4.1.6 Cassandra

Apache Cassandra ist eine verteilte Column-Store Datenbank die von Facebook entwickelt wurde [LM10]. Sie ist eine Mischung aus Amazon Dynamo und Google BigTable, wodurch sie des öfteren als Hybrid zwischen Key-Value-Store und Column Store bezeichnet wird. 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 auftreten können und auch wirklich auftreten. Cassandra behandelt das Problem von Fehlern durch Verwendung eines Peer-to-Peer-System, in dem alle Knoten gleich sind und die Daten von allen Knoten des Clusters verteilt werden. Jeder Knoten tauscht Informationen über das Cluster im Sekundentakt aus. Ein Commit-Log auf jedem Knoten fängt Schreibaktivität ab, um Datenhaltbarkeit zu gewährleisten. Daten werden auch auf eine In-Memory Struktur geschrieben, die memtable. Sobald die Speicherstruktur voll ist, werden die Daten in eine Datei auf die Festplatte geschrieben, auch SSTable genannt. Alle Schreibvorgänge werden automatisch aufgeteilt und auf mehrere Cluster repliziert.

Cassandras Datenmodell basiert auf einem partitionierten Row-Store mit eventueller Konsistenz. Zeilen werden in Tabellen organisiert, wobei die erste Komponente des Primär-schlüssels einer Tabelle der Partition-Schlüssel ist. Innerhalb einer Partition werden Zeilen

nach den verbliebenen Spalten des Primärschlüssels geclustert. 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.

Ein weiteres Unterscheidungsmerkmal stellt die Möglichkeit zur Angabe der Konsistenz Anforderung dar, die zum Zeitpunkt der Abfrage angebar ist. Weiterhin ist Cassandra ein schreiborientiertes System, während HBase entwickelt wurde, um hohe Leistung für intensive Leseaufgaben zu erzielen.

4.1.7 VoltDB

VoltDB [Vol13c] ist ein ACID-konformes, relationales In-Memory-Datenbanksystem, abgeleitet vom Forschungsprototyp H-Store [KKN⁺08]. Da VoltDB auf dem Ansatz der relationalen Algebra beruht 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. Durch Verwendung von gespeicherten Prozeduren als Transaktionseinheit werden Round-Trip-Messages zwischen SQL-Anfragen verhindert. Die Anfragen werden seriell in einem einzigen Thread ausgeführt, sodass kein locking and latching mehr notwendig ist [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. Die Datenbank kann im Embedded- oder Server-Modus betrieben werden.

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.

Als erste Eigenschaft wurde das Erscheinungsjahr festgelegt. Es ermöglicht Rückschlüsse auf die Ausgereiftheit einer Datenbank zu schließen. Ältere Datenbanken haben bereits viele ihrer anfänglichen Fehler beseitigt, was sie für den Einsatz in produktiven Umgebungen favorisiert. Natürlich sind ältere Systeme nicht gänzlich frei von Fehlern, allerdings existieren für sie meist Workarounds und Lösungsansätze. Cassandra zum Beispiel, erhielt über die Jahre neben zahlreichen Bugfixes, CQL als Query Sprache, MapReduce Support, sekundäre Indizes, verbesserte Komprimierung und vieles mehr. Allerdings gibt es keine feste Regel, die besagt wann 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.

Eine wichtige Rolle spielt jedoch die Lizenz unter die Datenbank vertrieben wird. Für Unternehmen ist die Wirtschaftlichkeit eines Systems von großer Bedeutung. Deshalb bieten

Open Source Produkte mit ihren geringen Anschaffungskosten, trotz des eingeschränkteren Supports, einen hohen Anreiz. Neben Wirtschaftlichkeit, ist Anpassbarkeit von Quelltext ein Argument für Open Source Produkte. Kommerzielle Lizenzen bieten hingegen eine höhere Zukunftssicherheit als Open Source Produkte, da letzteres meist von wenigen Privatpersonen entwickelt wird.

Unterstützte Programmiersprachen und Betriebssysteme sind Eigenschaften, bei denen eine Betrachtung des Ist-Zustandes sinnvoll ist. Im Unternehmen sollte idealerweise schon Erfahrung in den betrachteten Technologien vorhanden sein. Externe Mitarbeiter, sowie Schulungen sind teuer und müssen bei der Wahl, einer für das Unternehmen unbekannte Technologie, berücksichtigt werden.

Die Frage nach ein Schema stellt sich bei einer Betrachtung der Datenstruktur. Wenn sich die Struktur der abzulegenden Daten häufig ändert oder keine einheitliche Struktur unter den Daten zu erkennen ist, sind Schema freie Datenbanken von Vorteil. Den sie bietet ein hohes Maß an Flexibilität. Wohingegen man durch die Nutzung eines Schemas eine bessere Kontrolle über die Daten gewinnt.

Tabelle 4.1: Gegenüberstellung der Datenbankeigenschaften

Eigen-schaft	HBase	Cassandra	CouchDB	MongoDB	Redis	Voldemort	VoltDB	H2
Release-Datum	2008	2008	2005	2009	2009	2009	2010	2004
Datenbankmodell	Wide Column	Wide Column	Document	Document	Key-Value	Key-Value	Relational DBMS	Relational DBMS
Lizenz	Open Source	Open Source	Open Source	Open Source	Open Source	Open Source	Kommerziell	Open Source
Server-Betriebs-systeme	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	plattformunabhängig
Daten-schema	schemafrei	schemafrei	schemafrei	schemafrei	schemafrei	schemafrei	ja	ja
Typisie-rung	nein	ja	nein	ja	nein	nein	ja	ja
Sekun-därindi-zes	nein	eingeschränkt	ja (über Views)	ja	nein	nein	ja	ja
SQL	nein	nein	nein	nein	nein	nein	ja	ja
APIs und andere Zugriffs-konzepte	Java API, RESTful HTTP API, Thrift	Proprietäres Protokoll (CQL)	RESTful HTTP/JSON API	Proprietäres Protokoll basierend auf JSON	Proprietäres Protokoll	Proprietäres Protokoll	Java API, RESTful HTTP/JSON API, JDBC	Java API, ODBC, JDBC
Unter-stützte Pro-gram-mier-sprachen	C, C#, C++, Groovy, Java, PHP, Python, Scala	C#, C++, Java, Perl, JavaScript, Python, Ruby, +5	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, +8	C#, C++, Java, PHP, Python	C#, C++, Java, PHP, Python
MapRe-duce	ja	ja	ja	ja	nein	nein	nein	nein
Konsistenzkonzept	Immediate Consistency	Eventual Consistency, Immediate Consistency	Eventual Consistency	Eventual Consistency, Immediate Consistency	Eventual Consistency	Strict Consistency, Eventual Consistency	Integritätsbedingungen	Integritätsbedingungen
Transak-tionskon-zept	nein	nein	nein	nein	optimistisches Locking	nein	ACID	ACID
Neben-läufigkeit	ja	ja	ja	ja	ja	ja	ja	ja
Embed-dable	nein	ja	ja	nein	nein	ja	ja	ja
In Memory-fähig	nein	nein	nein	nein	ja	hybrid	ja	ja

Sekundärindizes können Lesegeschwindigkeiten steigern, weshalb sie eine interessante Da-

tenbankenfunktion darstellen. Sie erlauben Indizes auf einem oder mehreren Schlüsseln oder Nicht-Schlüsselattributen, was die Effizienz einer Suche steigern kann. Einige NoSQL Datenbank unterstützen solche Indizes, wohingegen relationale Datenbanksysteme die Definition beliebiger Sekundärindizes erlauben.

Typisierungen soll zum Ausdruck bringen, ob vordefinierte Datentypen wie Float oder Date in der Datenbank vorhanden sind. Ein Vorteil in der Verwendung von Datentypen ist eine Vorabkontrolle der Daten, sodass nur Daten mit den entsprechenden Eigenschaften verwendet werden. Zum Nachteil kann die mangelnde Flexibilität ausgelegt werden. Der Nutzer muss 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 proprietäre Protokolle oder standardisierte Protokolle handelt. Proprietäre Protokolle weisen meist eine höhere Einarbeitungszeit für die Mitarbeiter auf. Bei Arbeiten mit Standardtechnologien kann meist auf vorhandenem Wissen aufgebaut werden, was die Einarbeitungszeit verkürzt.

Die Entscheidung ob eine gewisse Stärke der Konsistenz ausreichend ist, wird durch den Anwendungsfall bestimmt. In manchen Anwendungen ist es schlichtweg egal, ob Daten redundant sind oder nicht. Die nächst höhere Anwendungsschicht, muss bei Inkonsistenz damit rechnen, sonst kann es zu schwerwiegenden Fehlern kommen. Beim Transaktionskonzept verhält es sich wie bei der Konsistenz, es hängt vom Anwendungsfall ab. Nebenläufigkeit gibt lediglich an, ob gleichzeitig ausgeführte Datenmanipulation, durch die Datenbank unterstützt wird.

Ob eine Datenbank im Embedded Modus betrieben werden kann ist von Bedeutung, wenn eine Integration in die Anwendung gewünscht ist. Dadurch können z.B. Verzögerungen durch Netzwerkzugriffe bei der Datenabfrage vermieden werden.

Die Eigenschaft In-Memory spiegelt den Wunsch der CAS Software AG wieder. Sie stellt somit ein wichtigstes Kriterium für die Auswahl der Datenbank dar.

4.3 Auswahl einer Datenbank

Jede Datenbank hat seine eigenen Stärken und Schwächen. Bei der Wahl der passenden Datenbank, ist nicht entscheiden, welche Datenbank im Vergleich zur anderen die Beste ist. Vielmehr ist von Bedeutung, ob die entsprechende Datenbank, den Anforderungen an das Gesamtsystem gerecht wird.

Dementsprechend ist in diesem Anwendungsfall die Abfragegeschwindigkeit einer Datenbank am bedeutsamsten. Die Verwendung des Hauptspeichers als Primärspeicher bedeutet einen theoretischen Geschwindigkeitsvorteil um den Faktor 50.000 [Pla13b]. In der Realität allerdings, spielen bei der Abfragegeschwindigkeit viele verschiedene Faktoren eine Rolle. Trotzdem dürfte der Geschwindigkeitsvorteil enorm gegenüber traditionellen Systemen sein. Fünf der Neun Datenbanken bieten diese Möglichkeit nicht. Deswegen ist zu klären ob diese Datenbanken andere Charakteristiken aufweisen können, um diesen Nachteil auszugleichen.

Cassandra und HBase ermöglichen hohe Performance durch horizontale Skalierung. Horizontale Skalierung ist vor allem bei hoher Last sinnvoll. Die Kunden der CAS Software AG sind alles mittelständische Unternehmen, welche nicht an die Nutzerzahlen von Facebook und Google herankommen. Daher sind keine Zugriffe im Millionen Bereich zu erwarten. Horizontale Skalierung ist dementsprechend nicht notwendig, sowie durch die Limitierung auf einen Rechner nicht möglich. Es ist zu erwarten das Cassandra und HBase auf einzelnen Servern nicht an die Performance von In Memory fähigen Datenbanken herankommen. Dies führte zu einer Entscheidung gegen die beiden Vertreter der Wide Column Stores.

Die Document Datenbanken sind zwar auch horizontal skalierbar, kommen jedoch nicht an die Performance von beiden zuvor genannten Datenbanken heran. Ihre Stärke liegt in Ihrer Schema freien Datenhaltung, die an dieser Stelle von geringem Wert ist, da die Daten eine feste Struktur haben. 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 schneller sein sollten, als Hauptspeicher basierte Datenbanken.

Die Key-Value-Stores ermöglichen mit Ihrer Form der Datenhaltung und der In Memory Fähigkeit, hohe Zugriffsgeschwindigkeiten. Was Ihnen jedoch zum Nachteil ausgelegt werden kann, ist ihre mangelnde Komplexität. Weiterhin sind sie auf Punkt-Abfragen ausgelegt. Komplexe Anfragen sind nur durch eine Realisierung in der Logikschicht möglich. Welche eine enormen Steigerung des Aufwands bedeutet. Daher wurde sich auch gegen die Key-Value-Stores entschieden.

VoltDB ist von den Eigenschaften her ein optimaler Kandidat, allerdings nicht Open Source. Die Datenbank konnte dadurch nicht verwendet werden. H2 hingegen ist Open Source und bietet Optionen zum vorhalten der Tabellen im Hauptspeicher. Davon werden sich hohe Geschwindigkeitsvorteile gegenüber herkömmlichen relationalen Systemen erhofft. Durch den Ansatz der relationalen Algebra, ist das Arbeiten mit SQL möglich. Das birgt Vorteile, da auf bereits bekanntem Wissen aufgebaut werden kann.

5. Konzeption

Ein Konzept dient in der Softwarearchitektur zur Bildung eines abstrakten Systemmodells als Basis für die Umsetzung. Zur Gestaltung werden technische Details weggelassen und stattdessen allgemeingültige Begriffe und ihre Zusammenhänge definiert. Weiterhin wird ein Grundverständnis durch definieren von Strukturen und Konzepten gebildet. Zu Beginn der Überlegung werden Systemgrenzen festgelegt und beschrieben was Teil des System ist. Überdies 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. Abschließend wird auf die Entwürfe der einzelnen Komponenten näher eingegangen.

5.1 Architektur

Zuerst wird ein erster Überblick über den groben Aufbau des Systems gegeben. In Abbildung 5.1 können die Zusammenhänge des abzubildenden Software-Systems betrachtet werden. Bei einer Betrachtung in der 3-Schichten-Architektur stellt der Browser und die Client.war die Darstellungsschicht dar. Die Fachkonzeptschicht ist in der Server.war umgesetzt. Die Datenbank befindet sich zwar auch in der Server.war, allerdings ist sie trotzdem unabhängig und kann jederzeit separat betrieben werden.

Die Vaadin Client-Side-Engine verwaltet das Rendering der Oberfläche im Web-Browser, durch den Einsatz verschiedener clientseitiger Widgets, die das Gegenstücke zu den serverseitigen 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.

Serverseitig arbeitet die Vaadin-Anwendungen auf der Java-Servlet-API. Das Vaadin-Servlet oder genauer die Klasse *VaadinServlet* ist für die Delegation verschiedenen Clients zuständig. Sie empfängt Anfragen und legt mithilfe von Cookies fest welche Benutzersitzung, zu welchem Client gehört.

Interaktionen mit dem Benutzer-Interface-Komponenten erzeugen Events, die zunächst auf der Clientseite durch Widgets verarbeitet werden. Nachfolgend werden die Events durch den HTTP-Server, das Vaadin-Servlet und durch die Komponenten der Benutzeroberfläche geleitet, bis sie zu den in der Anwendung definierten Event-Listenern gelangen. In den Listenern wird mithilfe des REST-Clients ein POST-Requests an die Logik gesendet. Dieser enthält alle in der Oberfläche definierten Parameter.



Abbildung 5.1: Konzeptionelle Darstellung der Architektur

In der Server.war werden REST-Requests entgegen genommen. Anhand der mitübertragenen Filteroptionen, werden die Bedingungen für die Datenbankabfrage zusammengestellt. Anschließend wird eine Verbindung zur H2-Datenbank aufgebaut. Das Ergebnis der Abfrage wird in das JSON-Format überführt und zurück an die Client.war geschickt. Dort angekommen werden die Daten an die Chart-Komponenten übergeben, was ein Neuaufbau der Komponente bewirkt.

Eine der Anforderung ist die unabhängige Umsetzung von Client und Server. Der erste Schritt zur Umsetzung der geforderten losen Kopplung zwischen Darstellung und Logik, wird durch die Aufsplittung in zwei verschiedene Anwendungen erreicht. Die Client.war beinhaltet die Klassen und Objekte der Darstellung. Wohingegen die Server.war alle Elemente zur Umsetzung der Logik enthält. Die Verwendung des REST-Protokolls zwischen der Client.war und Server.war stellt den nächsten Schritt der losen Kopplung dar. Einer der Vorteile ist, dass Funktionen des Systems durch andere Clients genutzt werden können, ohne Änderungen am Server durchführen zu müssen. Beide WAR-Dateien werden in einem Apache-Tomcat-Webserver deployed und können über die dementsprechende URL angesprochen werden.

Die Logikkomponente in der Architektur stellt eine Zusammenfassung aller Funktionen des Anwendungskerns dar. Sie kümmert sich um die Generierung der Abfragen, welche an die Datenbank gestellt werden. Dabei erfolgt eine dynamische Generierung der Abfragen, um nicht durch unnötige Bedingungen die Abfragegeschwindigkeit zu verringern. Abfragen werden mithilfe der Java Database Connectivity(JDBC) an die Datenbank gestellt. Neben der Generierung der Abfragen enthält die Logikkomponente Funktionen zum Extrahieren und Transformieren der Daten, aus der alten Datenbank. Der ETL-Prozess wird nur einmalig ausgeführt, allerdings stellt er einen wichtigen Schritt für die Umsetzung dar.

Um nicht periodisch Extraktion und Transformation wiederholen zu müssen, wird ein

selbstgeschriebenes Plugin im CAS genesisWorld-Anwendungsserver eingesetzt. Die Grundidee des Plugins ist Benachrichtigungen über Änderungen an unser System zu übermitteln. Dort findet eine Kontrolle statt, die den Datensatz auf Relevanz prüft. Ist dies der Fall, besorgt sich die Anwendungslogik anhand der zuvor übermittelten GGUID alle benötigten Daten.

5.2 Technologien

Als einer 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 Tomcat7 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 dient der Kommunikation zwischen Anwendungsserver und der Datenbank. Die H2-Datenbank stellt die Datenquelle des Systems dar. Im folgenden werden alle Bestandteile bis auf den H2 der bereits erläutert wurde, näher beschrieben.

Vaadin Vaadin ist ein Open-Source, Java basiertes Framework für den Aufbau von modernen Web-Anwendungen. Der Kerngedanke des Frameworks ist, dass die gesamte Anwendungslogik in der Serverseite einer Anwendung ausgeführt wird, während die Clientseite nur für das Senden der Benutzeraktionen an den Server und verantwortlich für die Reaktion auf die Antworten ist. Da es auf GWT basiert, kann sowohl der Client- und Server-Code in reinem Java geschrieben werden.

Die aktuelle Version von Vaadin, wurde im Februar 2013 veröffentlicht. Die folgenreichste Änderung von Vaadin6 war die Integration von GWT zu Vaadin, die eine bessere Unterstützung für die clientseitige Widget-Entwicklung bedeutet und sogar die Möglichkeit zu Erstellung von offline Vaadin-Anwendungen mit sich bringt.

Neben Open-Source, ist die im Unternehmen vorhandene Erfahrung ein Grund für die Wahl des Frameworks. Ausschlaggebend war jedoch VaadinCharts, welches ein Erweiterung von Vaadin darstellt. Es basiert auf Highcharts, einem JavaScript-Packet, welches eine umfangreiche Sammlung an Funktionen zur Darstellung von Diagrammen besitzt.

Jersey Jersey RESTful Web Services ist ein Open-Source-Framework zur Entwicklung von RESTful Web Services in Java, die Unterstützung für JAX-RS-APIs bietet und die JAX-RS (JSR 311 und JSR 339)-Referenzimplementierung darstellt. JAX-RS-Annotationen werden verwendet um die REST Relevanz von Java-Klassen zu definieren. Jersey ist dabei die Referenzimplementierung dieser Spezifikation. Jersey enthält im Grunde einen REST-Server und einen REST-Client. Auf der Serverseite verwendet Jersey ein Servlet, das vordefinierten Klassen abtastet 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-Anforderungen und wählen die richtige Klasse und Methode für die Anfragen aus. Diese Auswahl basiert auf Annotationen in den Klasse und Methoden. Weiterhin unterstützt JAX-RS die Erstellung von XML-und JSON, über die Java Architektur für XML Binding (JAXB).

Apache Tomcat7 Tomcat ist ein Open-Source Webserver, entwickelt von der Apache Group. Der Apache Tomcat implementiert die Java-Servlet und die Javaserver-Pages(JSP) Spezifikationen von Sun Microsystems und ist daher ebenfalls eine Referenzimplementierung. Er stellt weiterhin eine rein auf Java basierende HTTP-Webserver Umgebung dar. Apache Tomcat enthält Tools für Konfiguration und Management, kann aber auch durch die Bearbeitung von XML-Dateien konfiguriert werden.

opencsv Da Java das Parsen von CSV-Dateien nativ nicht unterstützt, müssen wir auf Drittanbieter-Bibliothek zurückgreifen. Mit opencsv erhalten wir eine sehr einfache CSV-Parser-Bibliothek für Java. Die Bibliothek kann zum erstellen, lesen und schreiben von CSV-Dateien benutzt werden. Die beste Fähigkeit des opencsv-Parsers ist das Mapping von Ergebnissen auf Java-Bean-Objekte.

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

5.3 Datenbankdesign

Das Datenbankdesign stellt einen wichtigen Abschnitt der Konzeption dar. Festlegungen im Bereich des Datenmodells werden in dieser Phase getroffen. Sie entscheiden ob Anforderungen und Erwartungen erfüllt werden können. In dieser Phase sind die Charakteristika der Daten zu untersuchen und das Datenmodell entsprechend nach ihnen auszulegen.

5.3.1 Konzeptionelles Design

Normalisierung dient der Organisation von Feldern und Tabellen einer relationalen Datenbank, um Redundanz und Abhängigkeit zu minimieren. Die Kehrseite hingegen ist eine Steigerung des Aufwands, um die benötigten Daten wiederzugewinnen. Normalisierung bietet die Möglichkeit einen Austausch zwischen Performance und Stabilität des Datenbankmodells vorzunehmen. In unserem Fall stellt ersteres absolute Priorität dar. Daher wird versucht die Normalisierung so gering wie möglich zu halten.

Die erste Überlegung hinsichtlich des Schemas ist, welche Daten für die Beantwortung der Abfragen benötigt werden. Der Datenbankdesigner steht bei analytischen System immer wieder vor der Entscheidung, wie viele Information aus dem alten System in das Neue übernommen werden sollten. Um höchst mögliche Performance zu erreichen werden lediglich die für das Szenario benötigten Daten extrahiert. Allerdings entsteht durch nachträgliches hinzunehmen von Funktionen ein erhöhter Aufwand für Änderungen am Schema und des ETL-Prozesses. Abbildung 5.2 zeigt das für die Datenbank neu entworfene Schema.

Die Idee hinter dem Schema ist die Verwendung einer einzelnen Tabelle zur Aufbewahrung der Informationen, der Verbindungsmerkmale. Diese Tabelle ermöglicht es ausgehend von einem Benutzer, alle Verbindungen zu anderen Personen zu finden. Im Grunde genommen sind vier Spalten dafür ausreichend. Die erste Spalte *startID* beinhaltet die Person, von der die Suche ausgeht. Eine Zuordnung der Tupel zu einem Datum erfolgt über die Spalte *Date*. Um Verbindungsmerkmale zu unterscheiden werden Zahlen von 1 bis 5, für die jeweiligen Verbindungsmerkmale, in der Spalte *DataTyp* verwendet. Die letzte Spalte *endID*, beinhaltet die Personen zu denen die Verbindungsmerkmale letztendlich führen. Anderen Spalten wie z.B. *Town* oder *Country* dienen lediglich der Filterung der Ergebnisse.

Um mit den geringeren Speicherkapazitäten die uns zur Verfügung stehen zurechtzukommen, wird auf das Problem der Datenredundanz eingegangen. Durch Normalisierung lässt sich Datenredundanz zwar nicht verringern, allerdings kann man sie in kontrollierbare Bahnen lenken. Im neuen Schema wurden solche Maßnahmen auf die Spalte *Town* und *Country* angewendet. Beide Spalten werden voraussichtlich Millionen von Werten beinhalten. Es gibt jedoch nur 193 Länder auf der Welt. Das bedeutet, dass Wörter wie Deutschland sich sehr oft wiederholen werden. Die Spalte *Country* ist vom Datentyp *Varchar*, welches pro Zeichen 2 Byte benötigt. Das wären beim Wort Deutschland 22 Byte. Legt man



Abbildung 5.2: Neues Datenbankschema

nun für die Spalte *Country* eine neue Tabelle an, wird in dieser jedes Land nur einmal vermerkt. Jedes Land bekommt einen Schlüssel in Form einer Zahl. In der eigentlichen Tabelle *Data* werden nur noch die Zahlen, anstatt den vollständig ausgeschrieben Wörtern verwendet. Das würde beispielsweise bei dem Wort "Deutschland" eine Reduktion von 22 Byte auf 1 Byte bewirken. Die Reduzierung auf 1 Byte lässt sich auf das `tinyint`-Format zurückführen. Das gleiche gilt für die Spalte *Town*. Bei ihr wird allerdings der Datentyp `smallint` verwendet, mithilfe dessen ein Zahlenbereich von -32768 bis 32767 abgebildet werden kann. Die Spalten *isEmployee*, *isContact* und *isFirm* können nur zwei verschiedene Zustände darstellen. Trifft zu oder trifft nicht zu. Der Datentyp `bool` reicht daher zur Abbildung der zweiseitigen Zustände aus. 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 da nur der Tag innerhalb des Datums von Interesse ist. Dazu wird ein frei gewählter Nullpunkt festgelegt. In unserem Fall wurde der 01.01.1990 als Nullpunkt gewählt, da keine älteren Daten existieren, die Relevanz besitzen. Darauf aufbauend wird das Datum, durch die Differenz in Tagen zum Nullpunkt, in der Spalte *Date* abgelegt. Die Hochrechnung der Tabelle 5.1 zeigt, dass durch die Normalisierung der Speicherplatzverbrauch um bis zu $\frac{1}{6}$ gesenkt werden kann.

Möchte man nun die Abfrage eines Benutzers die eine Filterung anhand einer Stadt voraus sieht beantworten, muss man zuerst an die *ID* der Stadt herankommen. Dabei können zwei verschiedene Ansätze verfolgt werden. Der erste Ansatz wäre ein Join zwischen *Town* und *Data*, um direkt mit dem Namen der Stadt zu arbeiten. Diese Variante dürfte aufgrund des Kreuzproduktes von Millionen von Zeilen nicht sehr performant sein. Eine andere Möglichkeit ist, eine separate Abfrage an die Datenbank zu stellen, in der die *ID* zum Namen ermittelt wird. Mithilfe der *ID* kann dann ohne einen Join die Ergebnismenge ermittelt werden. Dieser Ansatz dürfte vor allem durch die Abwesenheit von Netzwerkzugriffen zu höheren Abfragegeschwindigkeiten führen. Dieses Vorgehen kann für die Stadt, das Land

und die Gruppenzugehörigkeit angewendet werden.

Speicherplatzverbrauch ohne Normalisierung

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

Speicherplatzverbrauch mit Normalisierung

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

Tabelle 5.1: Vergleich des Speicherplatzverbrauchs

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 Zusammenstellung von Gruppen über die Zeit nachzuvollziehen. Action legt fest ob die Tupel einen Eintritt einer Person oder einen Austritt darstellen. Die Spalte Date beinhaltet das Datum des Ereignisses. Mithilfe beider Attribute lassen sich Gruppenzusammensetzung auf bestimmte Zeitpunkte bezogen rekonstruieren.

5.3.2 Zugriffsstrukturen

Indizes dienen der Beschleunigung von Suchen nach bestimmten Spaltenwerten. 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 sind die Kosten dafür. Daher bietet der Einsatz sich gerade in Anbetracht nach der Forderung von hoher Abfragegeschwindigkeit an. Zu beachten ist jedoch, dass jeder Index einen Zuwachs des Speicherplatzverbrauchs mit sich bringt. Zur Indexierung der Tabellen *Town*, *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. Indizierungen sollen in unserem Schema über die Spalten mit der Bezeichnung *Name* in den jeweiligen Tabellen vorgenommen werden, da der Client mit den Namen anstatt mit den IDs arbeitet. Mithilfe der Namen werden deshalb die zugehörigen IDs ermittelt. Die Nutzung von Hash-Indizes bringt allerdings Limitierungen mit sich. Eine der wichtigsten ist, dass sie nur für Vergleiche(“=”) verwendet werden können. Somit werden keine Wertebereich-Abfragen(“<” oder “>”) unterstützt. Es gibt allerdings noch andere Nachteile [SSH11], auf die aber in dieser Arbeit nicht näher eingegangen wird. Für die Tabelle *UserGroup* eignet sich der B^+ -Baum Standard-Index von H2. Dieser kann für die Spalte *userID* verwendet werden, der den ersten Wert einer Suche darstellt. Der B^+ -Baum-Index eignet sich auch für die Tabelle *Data*. Hier ist außerdem die Verwendung eines Mehr-Attribut-Indexes vorgesehen. Der Vorteil eines Mehr-Attribut-Indexes ist, dass bei einer Punkt-Abfrage über alle Zugriffsattributwerte nur ein Indexzugriff erfolgen muss. Indexiert werden in unserem

Fall die Spalte *startID* und *Date*. Beide Spalten sind sortiert und bieten sich somit für die Verwendung eines geclusterten Index an. Geclusterte Indizes sind in der gleichen Form sortiert wie die interne Relation. Ein geclusterter Index unterstützt Bereichsanfragen sehr gut, was bei der Beschränkung auf Zeitspannen von Vorteil sein dürfte.

5.4 Extract Transform Load 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. Somit sind die Daten in ihren Begrifflichkeiten häufig nicht vergleichbar und ihrer Bewertung sowie Konsolidierung unterschiedlich. Um die Daten dennoch für analytische Zwecke einzusetzen, ist eine Überführung in eine geeigneter Struktur von Vorteil. Eine solche Überführung wird in der Literatur als Extract-Transform-Load(ETL)-Prozess bezeichnet [ESHB11].

5.4.1 Extract

Zunächst dient die Extraktion primär der Beschaffung von Daten, aus dem MSSQL Server. Überdies können durch den Prozess Daten bereits reduziert, zusammengeführt und ersetzt werden. Für eine zutreffende Formulierung der Abfragen, müssen Besonderheiten in die Ermittlung der Daten beachtet werden. Eine vollständige und korrekte Datenmenge stellt die Grundlage jeder guten Analyse dar.

Die erste Besonderheit stellt die Analyse über Zeiträume hinweg dar. Es gilt dabei die Veränderungen der Daten über die Zeit zu berücksichtigen. Die Tabelle *Changelogbook* ermöglicht es Veränderungen in den Datensätzen nachzuvollziehen. Eine solche nachvollziehbare Veränderung ist in der Gruppenzusammensetzung zu finden, aufgrund von Abgängen und Zugängen von Personen. Neben den Datensätze die über die Zeit verändert wurden, existieren Datensätze die sich über längere Zeiträume erstrecken. Termine wie Tagungen beispielsweise, erstrecken sich über mehrere Tage. In der MSSQL-Datenbank werden diese Termine in einer Tupel aufbewahrt. Bei unserer Analyse hingegen stellt jede Tupel eine Verbindung zu einem bestimmten Zeitpunkt in Tagen dar. Somit muss ein Datensatz der sich über mehrere Tage erstreckt, in der H2-Datenbank durch mehrere Tupeln repräsentiert werden. Aufgrund dessen muss im Ergebnis der SQL-Abfrage die Anzahl der Tage vermerkt werden. In späteren Transformationen kann mithilfe dieser Angaben die entsprechende Anzahl an Tupeln erzeugt werden.

Eine weitere Besonderheit ergibt sich durch ein nicht im System vorgesehenes Verhalten der Benutzer, welche die Auswertung der Daten erschwert. CAS genesisWorld ermöglicht es Termine zu schieben. 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 statt gefunden haben, in der Datenbank nicht mehr existieren. Um trotzdem diese Termine zu berücksichtigen wurde folgendes Konzept erarbeitet. Dem *Changelogbook* lässt sich entnehmen ob die Felder *start_dt* und *end_dt* verändert wurden. Zur Feststellung ob ein Termin stattgefunden hat und anschließend geschoben wurde, müssen zwei Bedienungen erfüllt sein. Die erste ist der Zeitpunkt der Schiebung, die nach dem Termin liegen muss. Wird ein Termin aus anderen Gründen geschoben, findet dies in der Regel vor dem Start des Termimes statt, damit die Personen nicht unnötig zum Termin erscheint. Die zweite Bedingung ist, dass der neue Termin in der Zukunft liegen muss. Neben den beiden Bedingungen ist zu beachten, ob die Operation auf den Datensätzen ein Update war. Nur dann ist der Datensatz von Relevanz für die Abfrage.

Die Ergebnisse sämtlicher Extraktionen sollen in CSV-Dateien abgespeichert werden. Damit werden unter anderem Fehlersuchen vereinfacht. Weiterhin wird die Belastung des

Hauptspeichers verringert, da nicht alle Ergebnisse bis zum Schluss des ETL-Prozesses in der Java-Laufzeitumgebung aufbewahrt werden müssen.

5.4.2 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 Null Werten und zu 4% aus Leeren Feldern. Daten die beispielsweise Null-Werte 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 jedoch vernachlässigt werden, da sie zweckmäßig abdingbar sind.

Der nächste Schritt wäre die Harmonisierung der Daten. Unter anderem besitzen die Telefonnummern kein einheitliches Format. Sie wurde manuell von Sachbearbeitern eingetragen. Das erfordert ein zusammenführen aller Nummern in ein einheitliches Format, welches einen automatischen Vergleich ermöglicht. Die Verbindungsmerkmale müssen ebenfalls in eine einheitliche Form gebracht werden. Spalten die gleiche Inhalte besitzen aber unterschiedlich bezeichnet sind, müssen unter einer Bezeichnung zusammengeführt werden.

Die in der Extraktion genannten Besonderheiten werden durch unterschiedliche Datenbankabfragen ermittelt. Dies führt zu vielen separaten Dateien. Zur Nutzung der Daten sind sie zum Abschluss der Transformation zusammenzuführen. Das Ergebnis wird anschließend in einer CSV-Datei gespeichert, welche die Basis für das Befüllen der H2-Datenbank bildet.

5.4.3 Load

Beim Laden der Datensätze in die H2-Datenbank kommt ein sogenannter "bulk load" zum Einsatz. Dieser wird häufig zum laden von großen Datenmengen aus einer Datei in eine Datenbank eingesetzt. Er ermöglicht ein wesentlich schnelleres Befüllen der Datenbank bei großen Datenmengen, als wie üblicherweise mit INSERT-Operatoren.

5.5 Synchronisation des Datenbestandes

Systeme die auf dem Datenbestand anderer Systeme aufbauen, können zwei verschiedenen 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 ein 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. Zwar wird die Aktualität der Daten gewährleistet jedoch büßt A an Entscheidungsfreiheit ein. A kann nicht mehr selbst entscheiden wann aktualisiert werden soll. Der zweite Ansatz ist zwar effizienter jedoch nicht immer umsetzbar. Das kann technische oder unternehmenspolitische Gründe haben, die notwendige Veränderung am Legacy-Systems ausschließen.

In CAS genesisWorld gibt es eine Möglichkeit den zweiten Ansatz umzusetzen. Die Idee dabei ist den Applikationsserver um ein sogenanntes Plugin zu erweitern, welches über Veränderungen in den Datensätzen benachrichtigt wird. Ein solches Plugin kann als COM-Objekt mithilfe des Interfaces *IGWSDKDataPlugIn* realisiert werden. Das resultierende

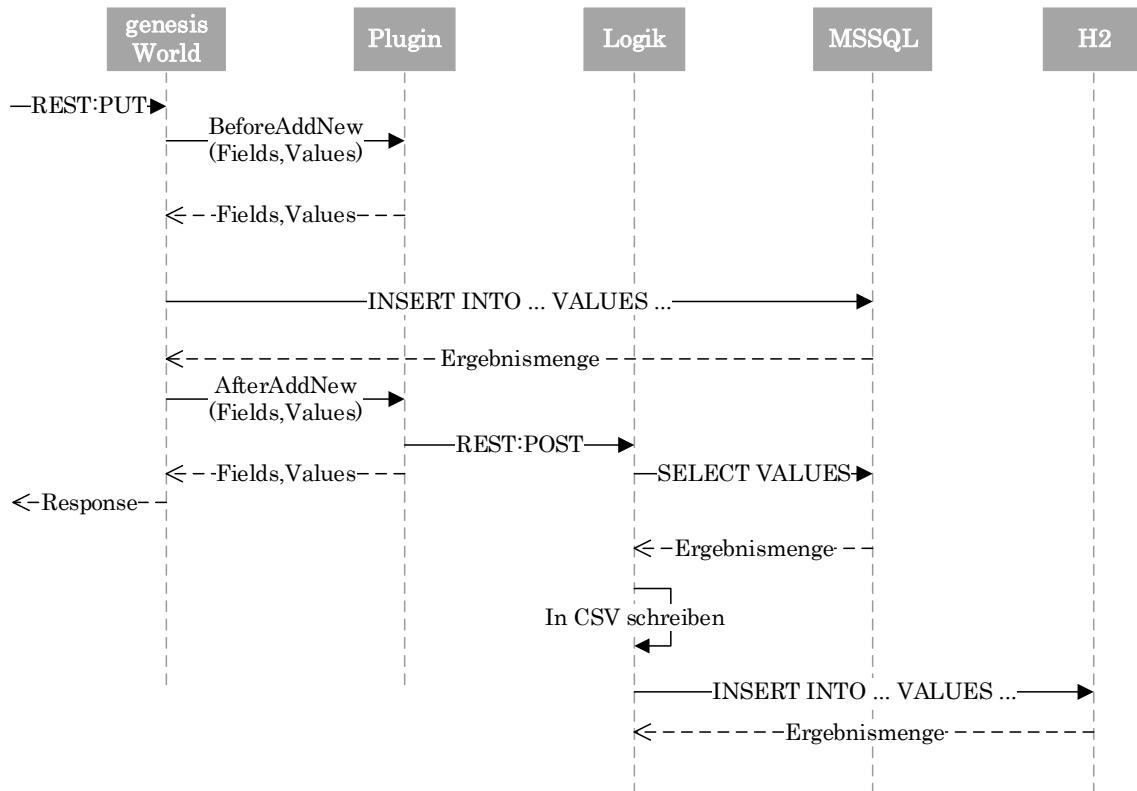


Abbildung 5.3: Sequenzdiagramm für einen neuen Datensatz

COM-Objekt wird im Server von CAS genesisWorld registriert. Der Server delegiert, wie in Abbildung 5.3 zu sehen, bei einer Datenoperation den Aufruf an die für die jeweiligen Tabellen registrierten Plugins. Das Plugin selbst soll einen REST-Client besitzt, der einen POST an die Logik sendet. Er enthält die *GGUID* des Veränderten Datensatzes. Mithilfe dessen die Extraktion des betroffenen Datensatzes angestoßen werden soll. Geplant ist neue Datensätze zuerst in einer CSV-Datei zwischenspeichern und anschließend in die H2-Datenbank einzufügen. Aktualisierungen der H2-Datenbank können auf der momentanen Datenbasis nur durch Erfassung neuer Datensätze aus dem MSSQL Server umgesetzt werden. Um auch Updates zu berücksichtigen müsste zu jeder Tupel die entsprechende *GGUID* vorhanden sein. Ohne die *GGUID* ist eine Zuordnung der Datensätze zwischen den Datenbanken nicht möglich. In diesem Fall wurde entschieden, dass dies kein Problem darstellt und es ausreichend ist die neuen Datensätze zu erfassen.

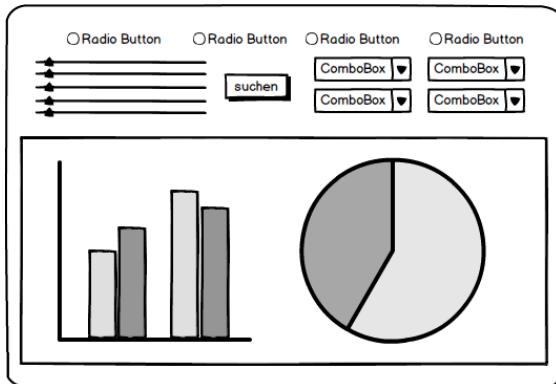
5.6 Darstellungskonzepte

Bei der Konzeption einer Darstellung ist die Grad der Granularität von Informationen ein wichtiger Leitfaktor, zur Bestimmung des Aufbaus. In unserem Fall ist nicht die Eigenschaft eines Verbindungsmerkmals von interessiere, sondern ihr Typ und ihre Häufigkeit zu einer bestimmten Person. Da keine Detailinformationen zum Verbindungsmerkmal vorhanden sind, kann jeder Benutzer frei wählen von welcher Person ausgehend die Analyse stattfinden soll. Für die Oberfläche bedeutet dies einen Einstiegspunkt in Form eines Fensters, in dem der jeweilige Benutzername von dem die Suche ausgehen soll, eingegeben wird. Zusätzlich soll die IP-Adresse und Portnummer des Server angebbar sein, falls sich dieser auf einem anderen Rechner befindet.

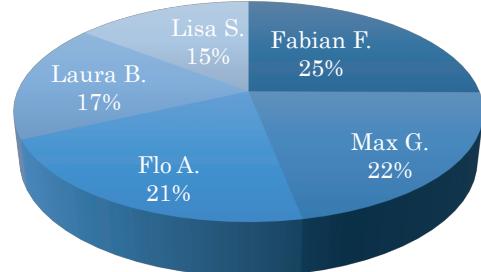
Nach der Anmeldung findet eine Weiterleitung auf die eigentliche Seite statt. Dessen Aufbau ist in Abbildung 5.4 (a) zu sehen. Im oberen Bereich auf der Seite sind alle Regler,

CheckBoxen und Eingabefelder zur Filterung der Ergebnismenge zu finden. Direkt darunter befindet sich ein Diagramm, welches die Ergebnismenge einer Abfrage visualisieren soll.

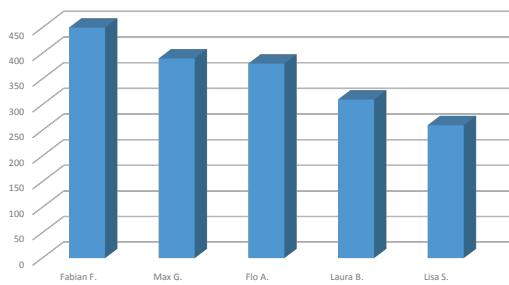
Diagrammtypen gibt es viele allerdings ergeben sich Einschränkungen durch die Verwendung eines Frameworks. Im Prinzip lässt sich jede Darstellung verwirklichen, allerdings ist das Aufwand-Nutzen-Verhältnis zu berücksichtigen. In einer Vorauswahl wurden einige umsetzbare Typen ausgewählt die in Abbildung 5.4 (b)-(f) dargestellt sind.



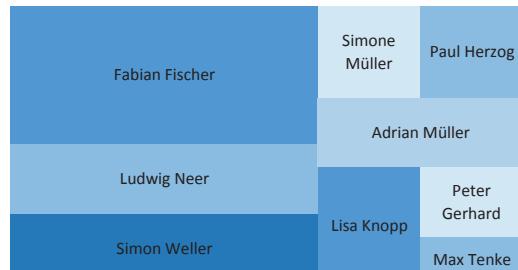
(a) Grober Entwurf des Aufbaus der Hauptseite



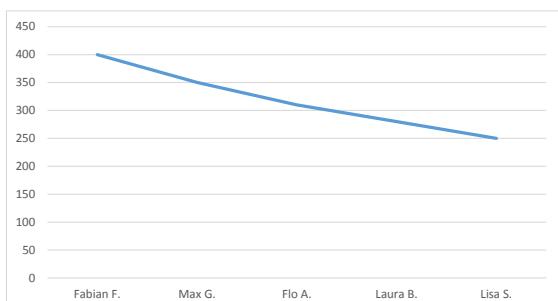
(b) Tortendiagramm



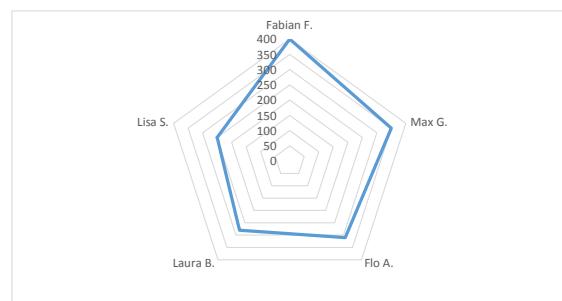
(c) Balkendiagramm



(d) Tree Map



(e) Liniendiagramm



(f) Netzdiagramm

Abbildung 5.4: Entwürfe für die Oberfläche

Netzdiagramme geben Eigenschaften verschiedener Systeme wieder. Sie eignen sich daher gut zur Darstellung von Ausprägungen. Für unsere Form der Daten ist diese Darstellung gänzlich ungeeignet, da mit Mengen gearbeitet wird.

Mithilfe von Liniendiagrammen lassen sich Trends und Zeitreihen darstellen. Die Verwendung verschiedener Linien ermöglicht zudem, die Darstellung mehrerer Trends. Die Benutzung dieses Diagramms macht keinen Sinn, da die Ergebnismenge sich nicht auf verschiedene Zeitpunkte bezieht, sondern die Summe der Werte aus einer Zeitreihe beinhalteten soll.

Bei einer Tree Map steht jede Fläche eines Rechtecks im proportionalen Zusammenhang zur Gesamtfläche. Die Beachtung von Größenverhältnissen stellt einen nützlichen Eigenschaft für unsere Daten dar. In unserem Fall würde jedes Rechteck aus dem jeweiligen Anteilen der Verbindungsmerkmale bestehen oder mithilfe eines Drilldowns¹ die Verbindungsmerkmale aufzeigen. Beispielweise könnte die Person Ludwig Neer wiederum in Rechtecke unterteilt werden, mit der jeweiligen Anzahl der verschiedenen Verbindungsmerkmale. Das würde allerdings schnell zu einer schlechten Übersicht führen, da zu viele Kacheln zu einer schlechten Übersicht führen. Wird in einer Tree Map die Drilldown-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 ermöglichen eine Betrachtung der Gesamtheit zu ihren Einzelstücken, da der Kreis ein geschlossenes System darstellt. Allerdings müssen alle Teile sich auf die gleiche Basis beziehen. Es eignet sich hervorragend zur Darstellung von Verhältnissen. Möchte man jedoch noch die Zusammenstellung eines einzelnen Stückes noch einmal aufteilen, ist eine zweite Ansicht nötig.

Am besten dürfte sich ein Balkendiagramm eignen. Reihenfolgen beispielsweise lasse sich durch die resultierenden Stufen sehr gut darstellen. Balken selbst lassen sich außerdem in einzelne Teile aufspalten, ohne die Übersichtlichkeit zu verringern. Gegenüber dem Kreisdiagramm kann es zwar keine Betrachtung des Gesamten liefern, allerdings ist das in diesem Anwendungsfall auch nicht nötig.

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

6. Umsetzung

In diesem Kapitel wird auf die konkrete Umsetzung der Konzepte eingegangen. Die Komponenten des Systems selbst wurden aus architektonischer Sicht, wie in der Konzeption beschrieben umgesetzt. Daher wird vielmehr auf die genaue Umsetzung der Funktionen und Prozesse eingegangen. Anhand von Klassendiagrammen wird in den ersten beiden Kapiteln, der Aufbau der Komponenten erläutert. Weiterhin wird der ETL-Prozess, sowie die Erzeugung der Abfrage separat betrachtet. Überdies wird in dem darauf folgenden Abschnitt der genaue Ablauf der Daten-Aktualisierung beschrieben. Abschließend wird der Aufbau der Oberfläche mit den damit verbundenen Designentscheidungen erläutert.

6.1 Aufbau der Server.war

Die Web-Archive-Datei beinhaltet ein dynamisches Webprojekt aus dem Eclipse Web Tools Platform(WTP) Projekt. Aufgrund der typischen Struktur des Webprojektes, wird im folgenden auf deren Klassen eingegangen. Abbildung 6.1 zeigt das Klassendiagramm der Server-WAR-Datei. 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 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

Mit der Klasse `JerseyServer` wird der REST-Server umgesetzt. 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 können Übergabeparameter vom Typ `UriInfo` und/oder `HttpHeaders` besitzen, die Abrufe von Metadaten der REST-Requests ermöglichen.

Neben den Methoden zur Beantwortung von REST-Requests, enthält die Klasse alle Objekte zur Durchführung des ETL-Prozesses. Die Klasse `ConnectorJDBC` besitzt ein Attribut namens `con`, welches den Verbindungsauflauf zum MSSQL-Server, mithilfe von JDBC ermöglicht. Zur Extraktion der Daten aus dem MSSQL-Server wird ein Objekt der Klasse `QueryBuilder` verwendet. Wie in der Abbildung zu sehen werden für die verschiedenen Tabellen, des neuen Schemas, eigene Methoden zur Verfügung gestellt. Methoden welche die Übergabewerte `table`, `date` und `n` besitzen, werden für die verschiedenen Verbindungsmerkmale benötigt. Mithilfe des Parameters `table` wird der Name der Tabelle in der MSSQL Datenbank übergeben. Der Parameter `date` gibt das Feld an, was für die Ermittlung des Datums verwendet werden soll. Um den Typ eines Verbindungsmerkmals zwischen Personen festzuhalten wird der Parameter `n` verwendet, der eine Zahl zwischen eins und fünf beinhaltet. `QueryBuilder` verwendet ein Objekt vom Typ `CSV-Builder`, um die Ergebnisse 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 CSV-Dateien. Weiterhin werden die durch die Extraktionen gewonnenen CSV-Dateien mithilfe eines `CSVReaderWriter` Objekts ausgelesen. Nach der Bearbeitung durch die Methoden der `Transform` Klasse, werden die Daten wieder in CSV-Dateien abgelegt. `CSVBuilder` besitzt Methoden die zusätzliche Parameter zum schreiben aufweisen, die modifizierte Schreiboperationen erlauben. Wohingegen `CSVReaderWriter` mithilfe der Methode `writeDataToCSV()`, sowie den Parametern `path` und `data` allgemeine Schreiboperationen durchführt.

Mithilfe der Klasse `Load` wird die Datenbank befüllt. Sie kann wie zuvor erwähnt von einem `Database` Objekt verwendet werden oder durch ein `JerseyServer` Objekt. Beim `JerseyServer` werden mit der Methode `load()` die Methoden der `Load` Klasse aufgerufen. In der Klasse `Database` werden sie im Konstruktor selbst aufgerufen.

Die Logik Klasse enthält alle Attribute und Methoden zur Beantwortung von Anfragen, durch den Nutzer. Zur Generierung der Bedingungen einer SQL-Abfrage werden separate Methoden verwendet. Die jeweiligen Methoden werden nur gerufen, sobald die entsprechende Bedingung, in der vom Nutzer erhaltenen JSON Datei, vorhanden ist. Angesteuert wird die Generierung einer Abfrage durch die Methode `buildQuery()`.

6.2 Aufbau der Client.war

Einstiegspunkt in der Client.war ist die Klasse `CasAnalyticUI`. Sie ist von der 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 (oder Tab) oder einen Teil einer HTML-Seite, wo eine Vaadin-Anwendung eingebettet ist darstellen. Nachdem eine `UI` von der Anwendung erstellt wurde, wird diese mit der Methode `init(VaadinRequest)` initialisiert. Zur Übersicht werden die Komponenten der Darstellung in die Klasse `RootUI` ausgelagert.

`RootUI` wird dabei von der `CasAnalyticUI` instanziert. Die Klasse `RootUI` beinhaltet das Anmeldefenster, sowie die 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.

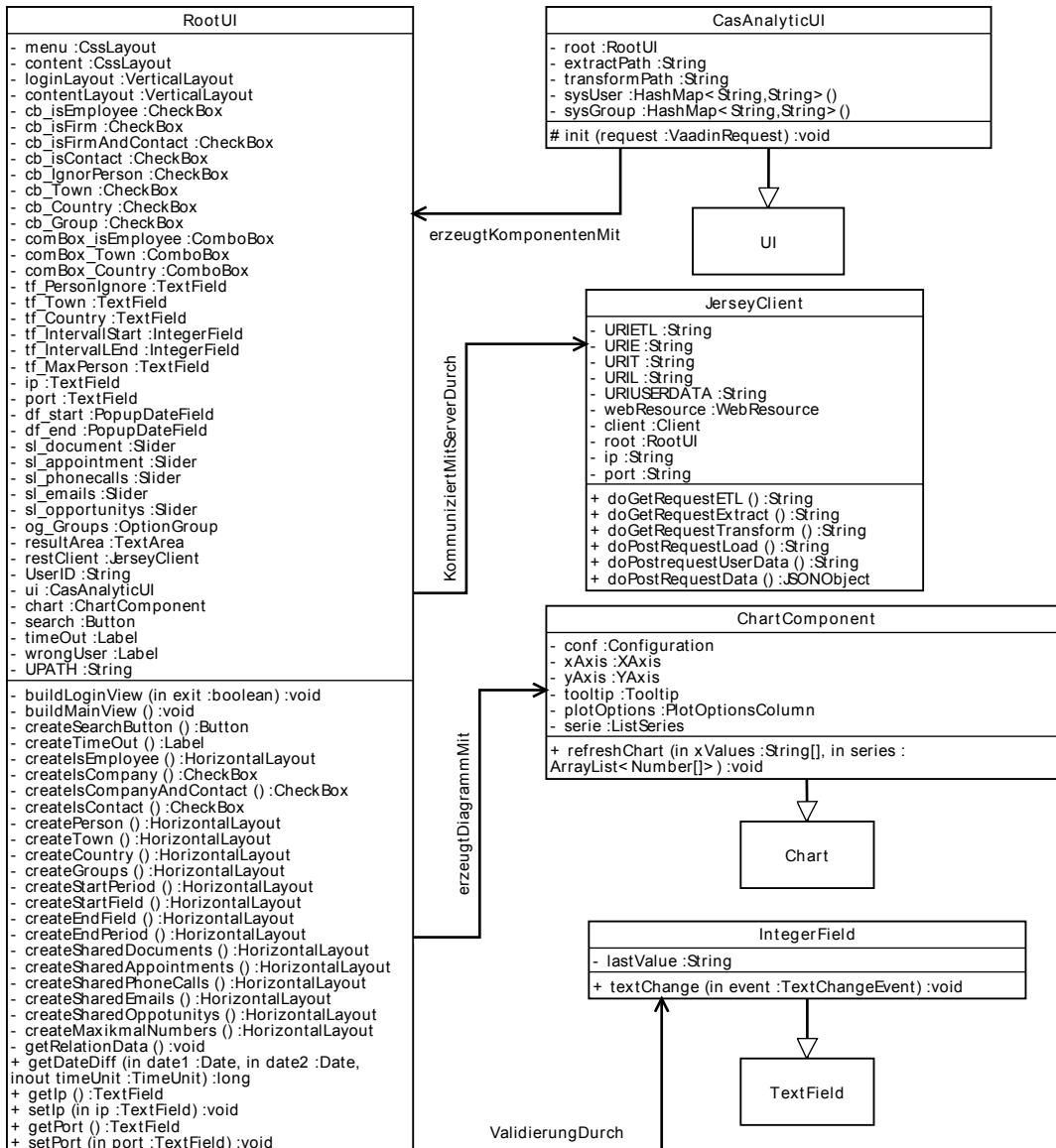


Abbildung 6.2: Client Klassendiagramm

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 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 jedes Vaadin-Objekt, welches eine Element an der Oberfläche darstellt, eine separate Methode zur Erzeugung verwendet. Änderungen am Aussehen oder an der Funktionalität der jeweiligen Vaadin-Objekte, werden nur innerhalb der entsprechenden

Methode vorgenommen.

An der Oberfläche gibt es Felder 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 Erzeugung der Abfrage

Um Abfragen durch den Benutzer möglichst performant zu beantworten erfolgt die Erzeugung dynamisch. Das bedeutet die SQL-Abfrage an die Datenbank je nach Anforderung anders aufgebaut sein kann. Die Basisstruktur ändert sich allerdings nicht. Diese besteht aus der Bildung von Summen, basierend auf verschiedenen Verbindungsmerkmalen. Zu Berücksichtigen ist das jede Abfrage einen Anfang- und Endzeitpunkt enthält, die zu beachten sind. Nachdem festgestellt wurde wie viele Verbindungsmerkmale von den jeweiligen Typen zu einer Person verlaufen, wird zusätzlich die Gesamtsumme der Verbindungsmerkmale zu einer Person gebildet. Die Summe wird zur Sortierung der Ergebnisse verwendet. Bei der Sortierung wird absteigend vorgegangen, um die Personen mit den meisten Verbindungsmerkmale zu der von der Suche ausgehend Person zu ermitteln. Das Ergebnis wird wiederum auf eine durch den Benutzer festgelegte Anzahl reduziert.

Zu diesen Kernfunktionalitäten können durch Nutzerangaben weitere Funktionalitäten hinzukommen. Eine von ihnen stellt die Gewichtung von Zeitpunkten dar. Der Ansatz zur Umsetzung der Gewichtung der Zeit, wird anhand der Abbildung 6.3 erläutert. Die Abbildung zeigt ein Koordinatensystem, mit den Gewichtung von einzelnen Zeitpunkten. Die x-Achse stellt den zeitlichen Verlauf dar. Während die y-Achse die Gewichtung darstellt. t_{start} markiert den Startzeitpunkt, wohingegen t_{end} den Endzeitpunkt angibt. Mithilfe von t_1 und t_2 werden Zeitspannen festgelegt, die differenziert zu gewichten sind. Um nun die Tage zu gewichten wurde der lineare Ansatz gewählt. Das bedeutet zwischen t_{start} und t_1 steigt die Relevanz stetig, bis sie die 100 Prozent erreicht hat. Ab dort besitzen alle Tage eine 100 prozentige Relevanz, außer es ist ein Wert für t_2 angegeben worden. Falls ja wird ab t_2 jeder Tag linear fallend bewertet.

Zur Berechnung der Gewichtung zu einem bestimmten Tag, werden die folgenden zwei Variablen verwenden:

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

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

Dabei wird wie folgt vorgegangen. t_{start} markiert den Ausgangszeitpunkt in Tagen. Aufsteigend zählend wird jeder Tag bis t_1 mit der Variabel f_1 multipliziert. Für den Zeitpunkt t_2 wird die Summe der Tage in der Zeitspanne zwischen t_2 und t_{end} verwendet. Die Summe wird mit jedem weiteren Tag um 1 vermindert, bis zum Zeitpunkt t_{end} , der 0 darstellt. Bei jedem Schritt wird der Tag mit der Variablen f_2 multipliziert.

Neben der Gewichtung der Zeit lassen sich die jeweiligen Verbindungsmerkmale unterschiedlich gewichten. Bei einer Abweichung von 100 Prozent wird die Summe des jeweiligen Verbindungsmerkmals, um die durch den Nutzer bestimmten Prozentsatz verringert.

Die restlichen Parameter dienen der Filterungen und werden je nach Nutzer-Eingabe, zu den Bedingungen der SQL-Abfrage hinzugenommen. Unter ihnen gibt es eine Bedingung,



Abbildung 6.3: Gewichtung der Zeit

die zuvor ermittelt werden muss und daher näher beschrieben wird. Dabei handelt es sich um den Ausschluss von Gruppen aus der Ergebnismenge, da ihre Zusammensetzung über die Zeit variabel ist. Dazu wird die Tabelle *UserGroup* verwendet. Dabei wird wie folgendermaßen vorgegangen. Zuerst wird die Ergebnismenge auf die ausgewählten Gruppen reduziert. Anschließend wird für jede Gruppe ein eigener Container erzeugt, der alle Personen der Gruppe enthält. Liegt nun der Zeitpunkt des Feldes *Date* nach dem Anfangszeitpunkt der Abfrage und die Spalte *Action* enthält eine 1, wird der Container um diese Person reduziert. Enthält die Spalte *Action* eine 0, wird die Person zum Container hinzugefügt. Mit einer 1 in der Spalte *Action* wird auf den Austritt einer Person aus der Gruppe, zum angegebenen Zeitpunkt aus der Spalte *Date*, verwiesen. Eine 0 bedeutet, dass die Person zu dem angegebenen Datum hinzukam. Dadurch wird die Zusammensetzung zum Anfangszeitpunkt wiederhergestellt. Mithilfe der Personen aus den Container, wird nun die SQL-Abfrage um weitere Bedingungen erweitert.

Zu beachten ist natürlich das innerhalb des Zeitraums Gruppenveränderungen stattgefunden haben können, diese jedoch nicht zu berücksichtigen sind. Es kann jeweils nur ein bestimmter Zeitpunkt für die Betrachtung berücksichtigt werden. In diesem Fall wurde der Anfangszeitpunkt gewählt.

6.4 ETL Prozess

Um an die notwendigen Daten zu gelangen werden zuerst die Informationen aus der alten Datenbank extrahiert. Dazu wird ein Verbund gebildet, der Tupeln aus den betroffenen Tabellen verschmelzen lässt. Die manuellen erstellten Verbindungen in der alten Datenbank werden mithilfe der Tabelle *TableRelation* festgehalten. Zur Gewinnung dieser Verbindungen wird als erstes eine SQL-Abfrage definiert, die für jede der Tabellen *gwOpportunity*, *gwPhoneCall0*, *Document0*, *EmailStore0* und *Appointment0* separat ausgeführt wird.

Mithilfe des ersten Verbundes zwischen *SysUser* und Adresse wird die Adresse einer Person ermittelt. Anschließend werden durch einen Verbund der *TableRelation* und *SysUser* alle Tabellen ermittelt, die mit der Person eine Verbindung besitzen. Der nächste Verbund wird zwischen *TableRealtion* und einer der fünf zuvor genannten Tabellen gebildet. Um festzustellen welche Personen beispielsweise mit einem Dokument arbeiten, 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 erhalten.

Die durch den Verbund gewonnen Informationen werden weiterhin auf drei relevante Werte verringert. Zu einem die *OID* des *SysUser*, von dem die Suche ausgeht. Zum anderen das Datum, welches durch ein Verbindungsmerkmal ermittelt wird. Weiterhin wird die zweite *OID*, die durch den Verbund mit einer zweiten *SysUser* Tabelle gewonnen wurde, behalten. Zum Schluss wird manuell eine vierte Information beigefügt, die besagt welchem Objekt die Tupel entstammt.

Für den Sonderfall das ein Datum über mehrere Tage geht, wird eine fünfte Spalte hinzugefügt, welche den Zeitraum in Tagen beinhaltet. Zur Beschaffung der geschobenen Termine wird genau wie in der Konzeption beschrieben verfahren.

Zur Ermittlung direkter Verbindungen zwischen Personen wird lediglich ein Verbund aus zwei ORel-Tabellen eines Objektes gebildet. Dieser Verbund beinhaltet bereits die *OIDs* der beiden Personen. Zur Ermittlung des Datums wird noch ein Verbund mit der Tabelle des Objektes gebildet. Die negativen *OID* Werte werden genauso wie oben beschrieben aufgelöst.

Jede Antwort einer SQL-Abfrage zur Gewinnung von Daten, wird in einer SCV-Datei direkt auf dem Tomcat Server abgelegt. Diese CSV-Dateien stellen die Grundlage der Transformation dar. Jede dieser Dateien beinhaltet Verbindungen zwischen Personen, in der Form wie sie in Abbildung 6.4 zu sehen ist.

```

1703 "22", "5238", "3", "42", "1", "1", "1", "Karlsruhe", "Deutschland"
1704 "22", "5238", "3", "42", "1", "1", "1", "Karlsruhe", "Deutschland"
1705 "22", "5238", "3", "11", "1", "1", "1", "Karlsruhe", "Deutschland"
1706 "22", "5238", "3", "123", "1", "1", "1", "Karlsruhe", "Deutschland"
1707 "22", "null", "3", "123", "1", "1", "1", "Karlsruhe", "Deutschland"
1708 "22", "5238", "3", "110", "1", "1", "1", "Karlsruhe", "Deutschland", "3"
1709 "22", "5238", "3", "20", "", ""
1710 "22", "5238", "3", "20", null, null, null, null, null

```

Abbildung 6.4: Ausschnitt einer CSV-Datei nach der Extraktion

Alle CSV-Dateien werden auf die in Abbildung 6.4 zu sehenden Ungereimtheiten untersucht. Dabei wird in Zeilen, wo die letzten fünf Werte fehlen, die Adresse über die *OID* (die vierte Zahl) ergänzt. Bei null Werten wird überprüft, ob wirklich keine Adresse vorhanden ist, falls doch werden diese ebenfalls ergänzt. Wenn wie in Zeile 1707 zu sehen, eine null Anstatt einem Datum steht, wird die Zeile entfernt. Falls wie in Zeile 1708 ein zusätzlicher zehnter Wert vorhanden ist, bedeutet das die Verbindung über mehrere Tage geht. Die Zeile bleibt bestehen allerdings wird der letzte Wert entfernt. Die Zahl wird jedoch noch verwendet, um die weiteren Zeilen mit aufsteigendem Datum zu erzeugen. Nach der Beseitigung von Anomalien werden noch die Städte und Länder durch ihre jeweilige *IDs* aus der Tabelle *Town* und *Country* ersetzt.

Die veränderten Daten werden wieder in CSV-Dateien abgelegt. Diese werden gleich bezeichnet, besitzen allerdings noch den Zusatz „_transf“, der sie als transformiert kennzeichnet. Diese Dateien werden anschließend in einer CSV-Datei zusammengeführt. Bei der Zusammenführung sind zum ersten mal alle Daten gleichzeitig in der Anwendung, weshalb an dieser Stelle alle Duplikate beseitigt werden. Weiterhin werden die Zeilen sortiert. Dabei wird mit zwei Kriterien sortiert. Das erste Kriterium ist der Erste Wert, die *OID* der Person von dem die Suche ausgeht. Wenn hierbei gleiche Werte verglichen werden, wird

das zweite Feld(Datum) herangezogen. Nachdem alle Zeilen sortiert und von Duplikaten bereinigt sind, werden alle Zeilen in einer CSV-Datei abgelegt.

Diese Datei wird bei jedem Start des Tomcats dazu verwendet, um einen Bulk-Load für die H2-Datenbank zu initialisieren. Nach der Befüllung der Datenbank mit Daten werden abschließend die Indizes erzeugt.

6.5 Aktualisierung des Datenbestandes

Wie zuvor in Abschnitt 5.5 behandelt, wird die Aktualisierung unseres Datenbestandes von CAS genesisWorld angestoßen. Zur Implementierung, ist die Definition einer COM-Komponente notwendig. Dazu wird eine DLL-Datei angelegt. Diese muss Namenskonventionen einhalten. Dateien die aufgenommen werden sollen, müssen mit dem Prefix *pGSA-xExtCustomServerDataPlugin* beginnen. Die DLL-Datei ist in der *RegisterSDKDataPlugins.xml* hinterlegt, damit der Anwendungsserver beim Start unser Plugin findet. Weiterhin ist in der XML-Datei eine Tabelle paarweise mit einer DLL angegeben. Dadurch wird ein Plugin auf eine Datenbanktabelle registriert und bekommt alle betreffenden Änderungen mit.

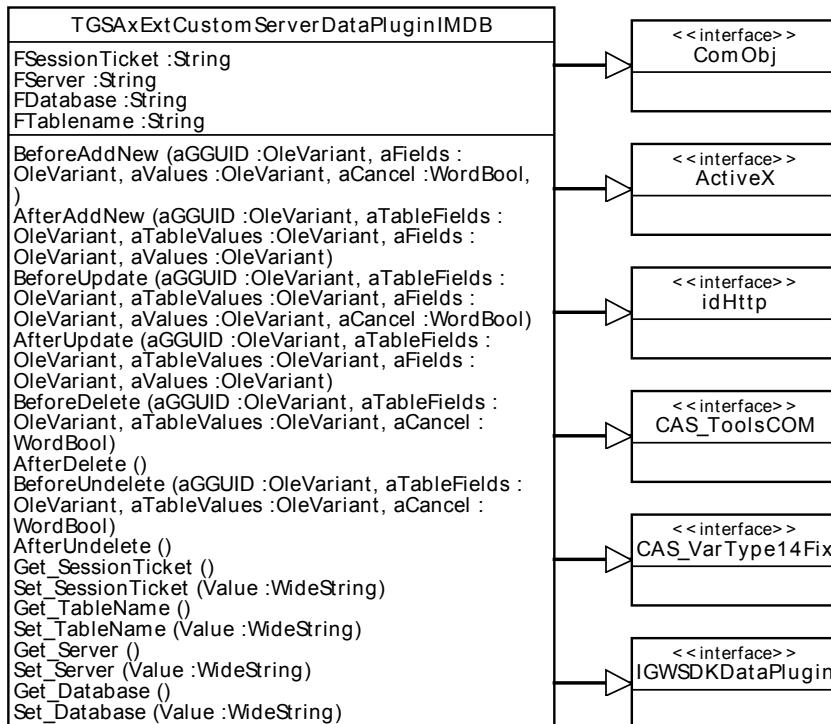


Abbildung 6.5: Klassendiagramm Plugin

Die Programmbibliothek selbst ist in Delphi realisiert. Abbildung 6.5 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. Die eigentliche Kernfunktionalität, das Abfangen der geänderten Daten, wird durch die Funktionen der *IGWSKDDataPlugin* Schnittstelle implementiert.

Die Funktionen der Abbildung 6.5 gehören von der Funktion *BeforeAddNew()* bis *AfterUndelete()* zur *IGWSDKDataPlugin* Schnittstelle. Es sind zwar alle Funktionen in der DLL implementiert, allerdings werden nur die Funktionen die mit *After* beginnen auch mit Logik hinterlegt. Für unser System reicht es nämlich aus, über Änderungen im Nachhinein benachrichtigt zu werden. Diese Funktionen enthalten alle die gleiche Logik und unterscheiden sich lediglich in den Übergabeparameter.

Die Logik innerhalb der Funktionen sieht wie folgt aus. Eine Veränderungen der Datensätze wird vom Nutzer angestoßen. Das Plugin wird nach den Änderungen aufgerufen. Die entsprechende Funktion erhält die *GGUID* der Tupel, den Namen der Spalte, sowie die veränderten Werte. Anschließend wird überprüft, ob die Änderungen für unser System von Relevanz ist. Dafür wird überprüft, ob die betroffene Spalte zu den in der ETL-Phase betroffenen Spalten gehört. Falls ja wird die *aGGUID* in einen String konvertiert. Weiterhin werden die Werte einer *idHttp* Variable gesetzt. Dazu werden Werte wie die URI oder HTTP-Header zugewiesen. Sobald alle Daten in der *idHttp* gesetzt sind, wird ein POST-Request an unser System übermittelt.

Der POST-Request enthält lediglich die *GGUID* und die Art der Operation, die auf den Daten ausgeführt wurde. Bei einem Update beispielsweise wird ein Header mit einem Key-Value übermittelt. In unserem Server wird lediglich überprüft, ob der geändert Wert bereits existiert. Falls ja wird er gelöscht und neu ermittelt. Falls nein, nur letzteres.

6.6 Oberfläche

In diesem Abschnitt wird die Umsetzung auf der Darstellungsebene erörtert. Den Einstiegspunkt für Nutzer stellt das in Abbildung 6.6 zu sehende Anmeldefenster dar. Der Hintergrund der Webseite ist in einem dunklen grau gestaltet, um einen Kontrast zum weißen Hintergrund der Bedienelemente zu bewirken. Zur Identifikation des Systems mit der Firma ist dessen Logo im linken Teil dargestellt. Im rechten Teil des Fensters existieren drei Eingabefelder. Zuerst ein Feld zur Eingabe der IP-Adresse des Server. 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.

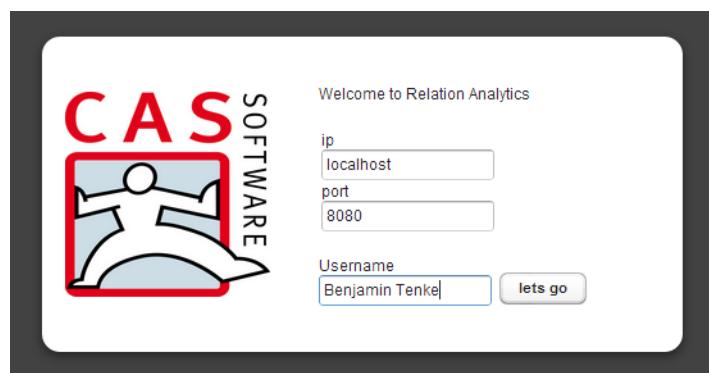


Abbildung 6.6: Anmeldefenster

Das Hauptfenster wurde vom Aufbau, wie in Abschnitt 5.6 beschrieben umgesetzt. Im oberen Bereich findet sich eine Leiste, die an Produkte wie Microsoft Office erinnert. Dies schafft ein vertrautes Gefühl mit der Oberfläche, da auf bekanntem aufgebaut werden kann. Die Leiste ist in vier Bereiche gegliedert. Der erste Bereich, ganz links, dient der Gewichtung der Verbindungsmerkmale 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 sowie kleine Wertebereich führte zu der Entscheidung. Zum stellen der Anfrage wird ein einfacher Button eingesetzt. Direkt unter dem Button befindet sich Text, der zur Ausgabe der benötigten Zeit vorgesehen ist.

Der zweite Bereich in der Mitte, dient zeitlichen Anpassungen. Das erste und dritte Feld können für Veränderung 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 6.3. Das obere Feld ist für t_1 . Hier kann die Zeitspanne zwischen t_{start} und t_1 in Tagen festgelegt werden. Für t_{ende} und t_2 verhehlt es sich mit dem unteren Feld ebenso.

Bis auf die Gruppenfilterung sind im dritten Bereich alle restlichen Filtermöglichkeiten vorhanden. Mithilfe von Checkboxen kann der Nutzer festlegen, welche Filterungen 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 Nutzer diese bestimmen.

Der Bereich ganz rechts in der Leiste, ist für den Ausschluss von Gruppen vorgesehen. Hier 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 der Namen durch den Nutzer. Das hat auch den Vorteil, dass der Benutzer Gruppen sieht die er eventuell nicht exakt beim Namen kennt.

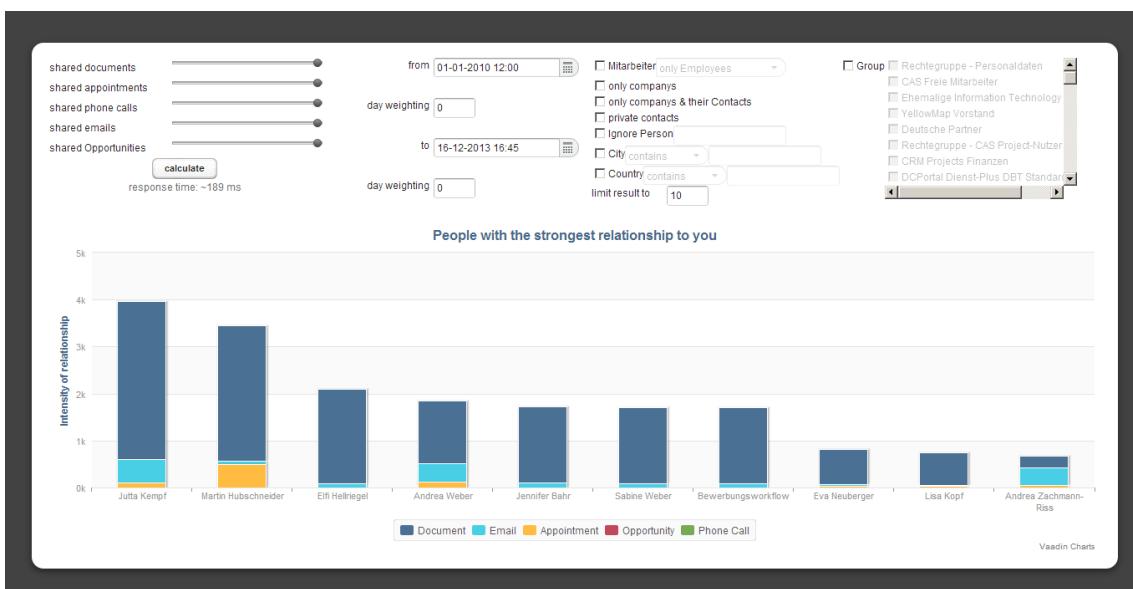


Abbildung 6.7: Hauptseite der Anwendung

Den zentralen Bereich des Fensters stellt das Diagramm dar. Die Balken selbst sind in fünf verschiedene Elemente unterteilt. Jedes Element wird dabei, durch eine andere Farbe dargestellt. Die fünf Elemente sind die verschiedenen Verbindungsmerkmale. Die Zuordnung

der Farbe zu dem jeweiligen Merkmal, wird über eine Legende im unteren Bereich des Fensters umgesetzt. Eine Besonderheit ist, dass durch einen Klick auf eine der Farben, das jeweilige Merkmal von der Darstellung ausgeschlossen wird. Beispielsweise kann der Nutzer auf die blaue Farbe neben dem Dokument klicken, was einen Neuaufbau des Diagramms, ohne dieses Merkmal bewirkt. Durch den Ausschluss wird allerdings keine neue Abfrage gesendet. Das bedeutet die Reihenfolge in der Personen angezeigt werden bleibt gleich, sowie die Datenbasis. Mit einem wiederholten Klick lässt sich der original Zustand wiederherstellen. Zusätzlich zu der Y-Achse, die eine Gesamtpunktzahl aufzeigt, kann der jeweilige Anteil eines Merkmals betrachtet werden. Dies geschieht durch einfaches platzieren des Mauszeigers, auf dem jeweiligen Bereich des Balkens. Dadurch öffnet sich ein Tooltip, welches die Anzahl der Punkte im Verhältnis zur Gesamtpunktzahl zeigt.

Die Anfragestellung erfolgt in der Regel mit dem dafür vorgesehenen Button. Die Regler und das Datum lösen jedoch bei Veränderungen automatisch eine neue Abfrage aus. Dies soll die hohe Antwortgeschwindigkeit des Systems untermauern und eine bessere Nutzererfahrung schaffen. Das Datum, sowie die Gewichtung wurden dafür ausgewählt, da diese die am meist benutzen Konfigurationsmöglichkeiten darstellen.

7. Fazit und Ausblick

In diesem abschließenden Kapitel werden die Ergebnisse dieser 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 analytisches System, basierend auf den CAS genesisWorld Daten entwickelt. Hierzu wurden zuerst alle relevanten, operativen Komponenten für die Entwicklung ermittelt. Anschließend wurde eine Szenario festgelegt, in dessen die Personen ausgeprägtesten Beziehung, zu einer ausgewählten Person ermittelt werden sollte. Mit dieser Frage im Hinterkopf, wurde die Datenbank nach den passenden Datensätzen, zur Beantwortung der Aufgabenstellung durchsucht. Nachdem klar war, welche Daten zu übernehmen sind, 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 durch ihre im Hauptspeicher gehaltenen Tabellen überzeugen konnte.

Aufbauend auf der bereits getätigten Wahl der Datenbank, wurden Konzepte zur Umsetzung des Systems entwickelt. Bei der Konzeption wurde deduktiv vorgegangen. Zuerst wurde die Architektur festgelegt. Aufbauend auf ihr wurden die einzelnen Komponenten detailliert geplant. Bei der Planung wurde nicht versucht ein universell einsetzbares System zu entwickeln, sondern vielmehr eine domänenspezifische Lösung für das Szenario zu erreichen. Nachdem alle Technologien, sowie Vorgehensweisen festgelegt wurden, ging man auf die Umsetzungen ein. Indessen eine Beschreibung der Funktionsweisen der einzelnen Komponenten durchgeführt wurde. Neben der Funktionsweise wurden die Interaktion unter den Komponenten dargelegt. Schließlich ging man noch auf die Umsetzung der Darstellung ein und den damit verbundenen Designentscheidungen ein.

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 die eher eine Rahmenbedingung darstellt, konnte eingehalten werden. Zur Umsetzung der Lösung wurde ein einziger Server verwendet. Weiterhin wurde eine lose Kopplung erreicht. Dies 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 noch dessen Einsatz auf verschiedenen Servern erlaubt.

Einer der wichtigsten Anforderungen ist die geringe Abfragegeschwindigkeit. Tabelle 7.1 zeigt, dass man dieser Forderung gerecht wird. Ebenfalls deutlich zu erkennen, ist die Auswirkung des geänderten Schemas. Der Sprung von 98.000 ms auf 350 ms, ist durch die Reduktion der Abfragekomplexität zu erklären. Im neuen Schema existieren weniger Tabellen und Tupeln. Außerdem ist im neuen Schema kein Verbund in der Datenbankabfrage mehr notwendig, der sehr rechenintensiv sein kann. Allerdings ist zu beachten, dass solche Maßnahmen, wie sie im Schemadesign ergriffen wurden weitreichende Folgen haben. Eine davon ist eine sehr schlechte Erweiterbarkeit des Schemas. Im momentanen Schema können lediglich Spalten hinzugezogen werden, dessen Inhalt in allen Verbindungsmerkmalen vorhanden ist. Außerdem würde für jede Spalte weitere 18 Mio. Werte hinzukommen. Die Hinzunahme von merkmalspezifischen Attributen führt ebenfalls zu hohen Änderungsaufwänden. Die Konsequenz die daraus gezogen werden müsste, wäre eine starke Normalisierung des Schemas, die die *data* Tabelle in mehrere Tabellen aufteilen würde. Dies würde wiederum den Einsatz von Verbundoperatoren erfordern. Dadurch würde die Abfragezeit wieder steigen. Allerdings wären wesentlich weniger Verbundoperatoren nötig als im alten Schema. Aufgrund dessen ist trotzdem mit einer deutlichen geringeren Abfragegeschwindigkeit 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

Bei Änderungen die eine Steigerung der Komplexität mit sich bringen, stellt sich eine Frage. Ist die Datenbank nur aufgrund des geänderten Schemas deutlich schneller? Um dieser Frage nachzugehen wurden einige Tests durchgeführt. Abbildung 7.1 zeigt die Ergebnisse dieser Testreihen. Alle Test wurden auf einem Rechner durchgeführt. Dieser simulierte den Zugriff von 100 gleichzeitigen Benutzern, mithilfe von Multithreading. Die in den Diagrammen angegebene Zeit bezieht sich somit auf die Ausführung aller 100 Abfragen. Jeder simulierte Benutzer führt somit, 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. Der Vergleich anhand von Insert-Anweisungen wird in (c) gezeigt.

Die Ergebnisse der Tests zeigen, dass die H2-Datenbank bei den durchgeführten 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 Geschwindigkeit, im Gegensatz zu herkömmlichen Datenbanken, gewinnt. Diese Geschwindigkeit wird

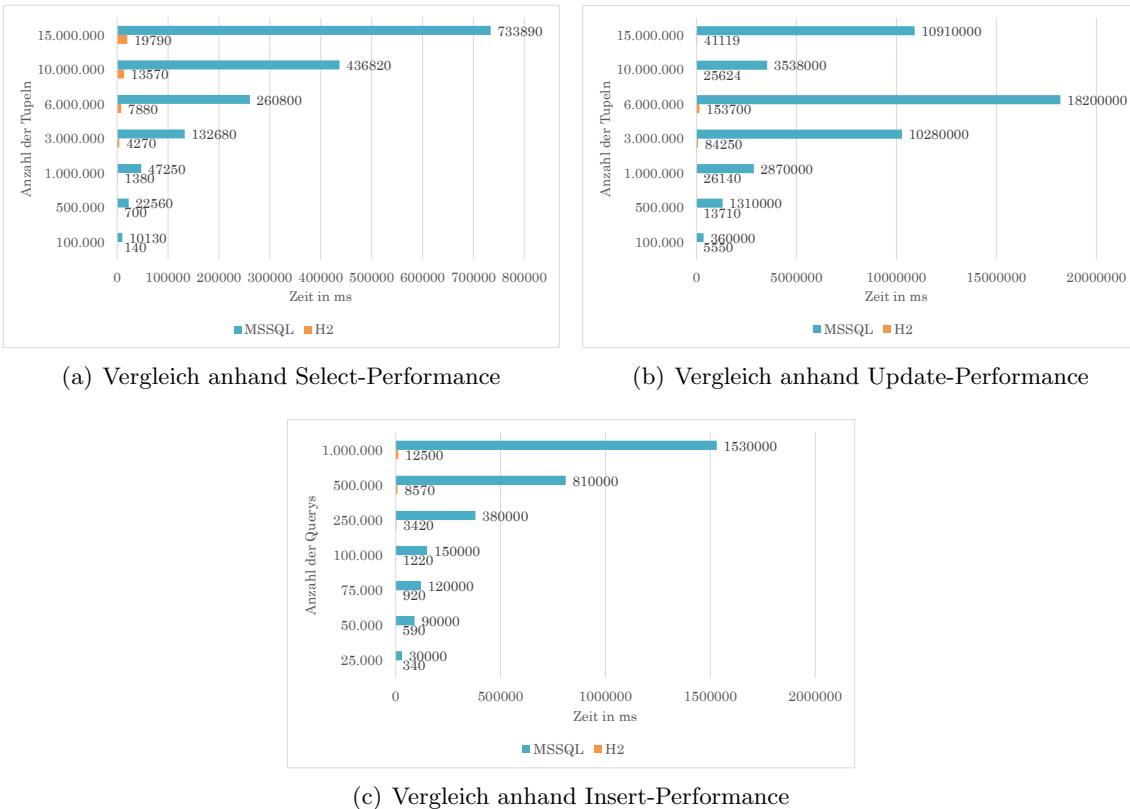


Abbildung 7.1: Abfragegeschwindigkeit Vergleich

zum Teil durch den Verzicht auf Persistenz erlangt. Zum anderen durch die Nutzung des Hauptspeichers als Speichermedium. Im vorliegenden System, welches fast nur Leseoperationen durchführt, stellt die mangelnde Persistenz allerdings kein großes Defizit dar. Die Nutzung des Hauptspeicher 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 eine performante Lösung bereit, die Untersuchung und Bewertung von Beziehungen zwischen Personen ermöglicht.

Die Bewertung der Beziehungen beruht derzeit lediglich auf der Anzahl von Verbindungsmerkmalen. Dabei wird nur die Häufigkeit gewertet. Um die Ausprägung einer Beziehung noch genauer feststellen zu können, werden zusätzliche Regeln benötigt. Diese Regeln müssten auf psychologischen Erkenntnissen und Erfahrungswerten aufbauen. Durch Regeln könnte die Aussagekraft von Ergebnissen weiter steigen. Beispielsweise ist die Kommunikation durch Termine, Telefonate und E-Mail Verkehr, kein Indikator für Vertrauen oder dergleichen. Dokumente in die man anderen Personen Einsicht gewährt und nicht öffentlich sind, setzen eine engere Zusammenarbeit oder Vertrauen voraus. Dies sollte somit stärker gewichtet sein als beispielsweise ein Telefonat. Neben den festen Regeln sind Vorschläge für die Benutzer über verschiedenen Gewichtung der Verbindungsmerkmale sinnvoll. Dabei kann der Nutzer zwischen verschiedenen Vorgaben wählen, die basierend auf Erkenntnissen und Erfahrungen beruhen. Jeder dieser Vorschläge verkörpert verschiedene Charakteristiken. Beispielsweise könnte eine solche Vorgabe wie folgt aussehen. Telefonate, E-Mail und Termine könnten viel stärker gewichtet werden, falls Interesse besteht zu erfahren

mit welchen Personen am meisten direkter Kontakt besteht. Diese Einstellungen zur differenzierten Gewichtung zwischen den Verbindungsmerkmalen, könnten als vordefinierte Regeln angeboten werden.

Weiterhin kann durch den Einsatz des Systems, der Vertrieb eines Unternehmens unterstützt werden. Eines der denkbaren Szenarien setzt eine Erweiterung des Systems voraus. Mithilfe von zusätzlichen Informationen über den Wert der jeweiligen Person für die Firma, könnten Vergleiche angestellt werden. Diese Vergleiche könnten anhand von Abgleichen der Position des Kunden im Ranking der Beziehung und dem Ranking anhand des Wertes eines Kunden erfolgen. So könnte Unstimmigkeiten in Aufwand und Nutzen entdeckt werden. Außerdem könnte jeder Vertriebsmitarbeiter mithilfe des Systems eine individuelle Unterstützung erfahren. Hierbei könnte das Ranking, der Personen mit den stärksten Beziehungen zum Vertriebsmitarbeiter herangezogen werden. Dadurch können Vertriebsmitarbeiter überprüfen ob sie dem jeweiligen Kunden genug Zeit widmen oder anderen zu wenig. Auch könnte das Ranking durch eine sehr simple Manipulation so geändert werden, dass die Personen angezeigt werden mit denen man am wenigsten zu tun hat. Was wiederum eine Überprüfung auf mangelhafte Kundenpflege ermöglicht. Überdies könnten Möglichkeiten zum Vergleich mehrerer Personen realisiert werden. Der Vergleich könnte dabei unter Personen aus einer Gruppe oder aus einer durch den Nutzer zusammengestellten Menge erfolgen. Mithilfe eines Vergleichs unter Vertriebsmitarbeitern 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 einen Kunden konzentrieren. Eine weitere weiterführende Möglichkeit, wäre die Hinzunahme der Darstellung von einzelnen Beziehungen über die Zeit. Liniendiagramme wären dabei eine geeignete Form der Visualisierung, da sich mit ihnen zeitliche Abläufe gut darstellen lassen. Mithilfe des momentanen Datenbestandes ist dies möglich, es muss lediglich eine Abfrage und die Darstellung dazu entwickelt 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⁺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>.

- [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>.
- [GR11] John Gantz und David Reinsel: *Extracting Value from Chaos*. 0, 2011. <http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf>, [Online;accessed 09-January-2014].
- [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⁺08] Robert Kallman, Hideaki Kimura, Jonathan Natkins, 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.
- [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, 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.
- [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. 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, 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, [Online;accessed 14-November-2013].
- [Vol13c] VoltDB: *Technical Overview*. 2013. http://voltdb.com/downloads/datasheets_collateral/technical_overview.pdf, [Online;accessed 14-November-2013].

