



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: 29. Oktober 2013

Inhaltsverzeichnis

1 Einführung	1
1.1 Motivation	1
1.2 Zielsetzung	1
1.3 Gliederung der Arbeit	1
2 Grundlagen	3
2.1 NoSQL - Eine Einführung	3
2.1.1 Document Stores	3
2.1.2 Extensible Record Store	3
2.1.3 Key-Value-Store	4
2.1.4 Graph Datenbanken	4
2.1.5 NoSQL Theoretische Grundlagen	5
2.2 In Memory Datenbanken	6
2.3 Component Object Model	7
2.3.1 Architektur	8
2.3.2 COM-Client	8
2.3.3 COM-Server	8
2.3.4 COM-Schnittstelle	9
2.3.5 COM-Objekte	9
2.3.6 Interface Definition Language	9
3 Systemanalyse	11
3.1 CAS genesisWorld	11
3.1.1 Architektur	12
3.1.2 Präsentationsschicht & Logikschicht	12
3.1.3 Datenhaltungsschicht	13
3.2 Anforderungsanalyse	14
3.2.1 Funktionale Anforderungen	14
3.2.2 Nicht funktionale Anforderungen	15
3.3 Ermittlung relevanter Daten	15
4 Analyse ausgewählter Datenbanken	19
4.1 Datenbanken	19
4.1.1 CouchDB	19
4.1.2 MongoDB	20
4.1.3 Voldemort	20
4.1.4 Redis	21
4.1.5 HBase	21
4.1.6 Cassandra	21
4.1.7 VoltDB	22
4.1.8 H2	22
4.2 Gegenüberstellung	22

4.3 Auswahl einer Datenbank	24
5 Konzeption	27
5.1 Architektur	27
5.2 Technologien	29
5.3 Datenbank	30
5.3.1 Schema der Datenbank	30
5.3.2 Zugriffsstrukturen	32
5.4 Extract Transform Load Prozess	32
5.4.1 Extract	32
5.4.2 Transform	33
5.4.3 Load	34
5.5 Sicherstellung der Aktualität der Daten	34
5.6 Darstellungskonzepte	35
6 Umsetzung	37
6.1 Kompression	37
7 Ergebnisanalyse	39
7.1 Test Aufbau	39
7.2 Auswertung	39
7.3 Handlungsempfehlungen	39
8 Zusammenfassung	41
8.1 Ergebnis	41
8.2 Ausblick	41
Literaturverzeichnis	43

Abbildungsverzeichnis

2.1	Beispiel einer Spalten Familie	4
2.2	Beispiele von Knoten und Kanten in einer Graph Datenbank	5
2.3	Bestandteile einer COM-Architektur	8
3.1	Schematische Darstellung der potenziellen Verknüpfungen	11
3.2	Schematische Darstellung der Architektur von CAS genesisWorld	12
3.3	Beispiel zur Benachrichtigung von PlugIns anhand eines Ablaufs bei einem Update	13
3.4	Funktionsweise von RelationTable anhand eines Beispiels	14
3.5	Auszug aus dem Schema des MSSQL 2008	16
5.1	Architektur des Systems	28
5.2	Neue Datenbankschema	31
5.3	Sequenzdiagramm eines Updates	34
5.4	Entwürfe der Oberfläche	36
7.1	Abfragegeschwindigkeit Vergleich	40
7.2	Oberfläche des Systems	40

Tabellenverzeichnis

4.1	Gegenüberstellung von den Charakteristika der Datenbanken	23
6.1	Vergleich des Speicherplatzverbrauchs	37
6.2	Zeilenanzahl	38
7.1	Abfragegeschwindigkeit Vergleich	39

1. Einführung

...

1.1 Motivation

...

1.2 Zielsetzung

...

1.3 Gliederung der Arbeit

...

2. Grundlagen

2.1 NoSQL - Eine Einführung

NoSQL ist keine Datenbank, noch nicht einmal ein Datenbanktyp. Der Terminus NoSQL fasst lediglich Datenbank 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 sind unter gewissen Umständen wichtiger als Atomarität und Konsistenz. NoSQL Datenbanken lassen sich anhand Ihres Datenmodells unterscheiden. Nach [Vai13] ist eine Klassifizierung in folgenden 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, wird sie in einem Dokument abgelegt. Diese können als eine Datei auf dem Dateisystem betrachtet werden. Dieses Dokument kann alle möglichen Daten aufnehmen und muss dabei keinem Schema folgen. Obwohl die Daten schematisch 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 wenn dynamische Entitäten eingesetzt werden die z.B. eine hohe Anzahl von optionalen Feldern besitzen.

2.1.2 Extensible Record Store

Extensible Record Stores, auch Wide Column Stores und auf deutsch oft spaltenorientierte Datenbank genannt, speichern im Gegensatz zu den traditionellen relationalen Datenbanken, Daten mehrerer Einträge in Spalten anstatt in Zeilen. Jeder Eintrag einer Spalte besteht aus einem Namen, den Daten und einen 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 die Struktur tausenden oder sogar Millionen von Spalten beinhalten

weshalb sie auch Wide Columns Stores genannt werden. Verwandten Spalten werden in Spalten-Familien durch von einer Anwendung bereitgestellte Reihe von Schlüsseln identifiziert. Weiterhin muss in einer Spalten-Familie nicht jede Zeile aus den gleichen Spalten bestehen.

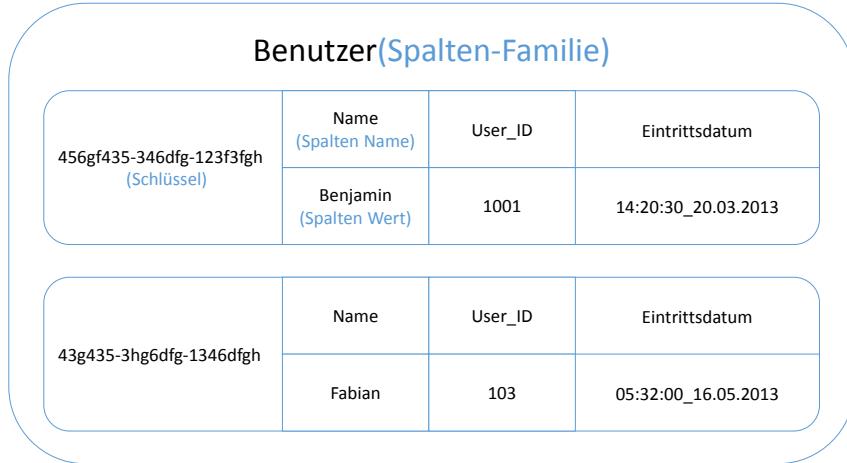


Abbildung 2.1: Beispiel einer Spalten Familie

Diese Architektur hat mehrere Vorteile. Meist weisen Werte in Spalten eine geringe Entropie auf was Sie geeignet für Kompressionsverfahren 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). Performance verschlechternd sind natürlich im Gegenzug Zugriffe die nicht mehr auf einzelnen Spalten erfolgen sondern über viele Spalten hinweg ausgeführt werden.

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 von den Daten in den Werten 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 die Komplexität solcher Datenbank gesteigert was jedoch nicht dazu führt das sie Join- oder Aggregat-Operatoren beherrschen.

2.1.4 Graph Datenbanken

Eine Graph Datenbank verwendet die Graphen Theorie 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 und jede Kante eine Beziehung oder Verbindung zwischen zwei Knoten, wie in Abbildung 2.2 zu sehen. 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 das sie statt Kanten, 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.

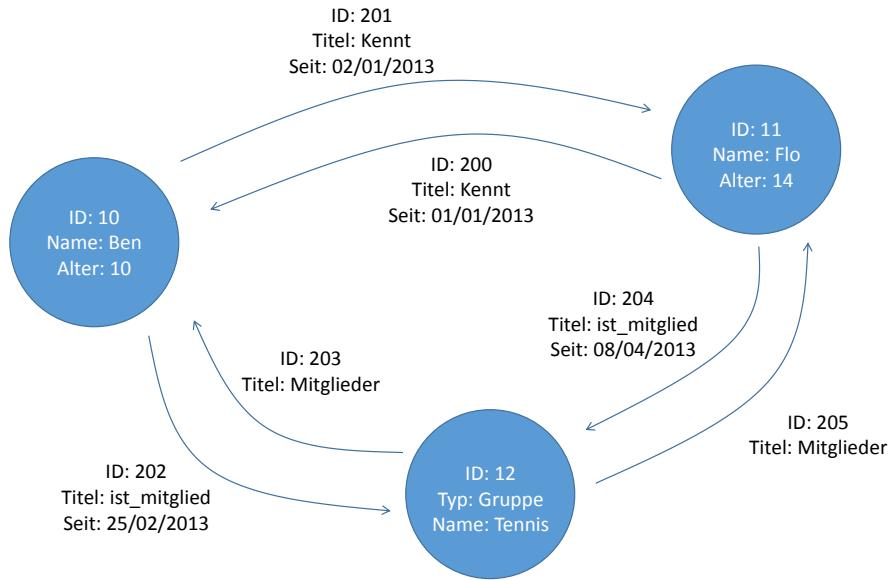


Abbildung 2.2: Beispiele von Knoten und Kanten in einer Graph Datenbank

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 Datenelement an mehr als einem Knoten gespeichert ist. Dies ist sehr nützlich, um Lese-Performance der Datenbank zu erhöhen, weil es einen Load Balancer erlaubt alle Lesevorgänge über viele Maschinen zu verteilen. Außerdem ist von Vorteil, dass es die Cluster robuster gegen den Ausfall einzelner Knoten macht.

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, was die Implikation hat dass die verteilten 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 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 dieselben Daten parallel zugreifen zu lassen ohne eine Beschädigung der Daten und die Möglichkeit von Deadlocks. Es ist

eine Alternative zu den Lock-basierte 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 Papier 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 benutzerdefinierte 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.

Vector Clocks (Vektor Uhren)

Vektor Uhren 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 Vektor Uhr welche aus Tupeln mit verschiedenen Zeitpunkten besteht. Jeder Zeitpunkt stellt einen Prozess der eine Modifikation an dem Datenelement vorgenommen hat dar. 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 Uhrenwerte 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 eine 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 Dr. Brewer erstmal in einem Symposium [Dr.00] 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 Partition tolerante verteilte Datenbank ist fehlertolerant gegen temporäre Verbindungsprobleme und ermöglicht es Partitionen über Knoten zu trennen. Ein System das tolerant partitioniert ist kann nur starke Konsistenz durch Verminderungen in seiner Verfügbarkeit erreichen. Grund dafür ist das es zuerst sicherstellen muss ob jeder Schreibvorgang abgeschlossen wurde bevor er eine Replikation durchführen kann. Jedoch kann es vorkommen das in einer verteilten Umgebung das nicht möglich ist, Aufgrund von Verbindungsfehlern oder anderen temporären Hardware Problemen.

2.2 In Memory Datenbanken

Eine In-Memory-Datenbank (IMDB) ist ein Datenbank-Management-System, das in erster Linie den Hauptspeicher als Medium für die Datenablage benutzt. 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 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 [Pla13].

Die hohe Performance dieser Systeme kann nicht durch simples verschieben der Daten in den Hauptspeicher erreicht werden. Bisherige Konzepte des Datenbankentwurfs müssen dabei 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. Wenn ein Geschäftsprozess in sich abgeschlossen ist werden die Daten nur noch aus Datenvorhaltungsgründen aufbewahrt. Eine Auslagerung solcher nicht benötigten Daten ermöglicht es zusätzlichen Speicherplatz Verbrauch zu reduzieren.

In-Memory Datenbanken die den relationalen Ansatz verfolgen besitzen zudem geänderte Query Optimizer. In herkömmlichen RDBMS sind Lese- und Schreiboperationen einer der wichtigsten Faktoren zur Bestimmung des optimalen Query Plans. 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. Snapshot 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.

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]. Er 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), ist es wichtig zu verstehen, dass es sich nicht um eine objektorientierte 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 Objekt Model 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 Maschinen Code möglich. Das führt dazu das COM des Öfteren als binärer Standard referenziert wird.

COM bietet die Möglichkeit auf viele Windows Funktionen zuzugreifen. Des weiteren ist COM die Basis für die OLE–Automation (Object Linking and Embedding) und ActiveX. Die Verwendung des COM-Standards bietet weiterhin 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.

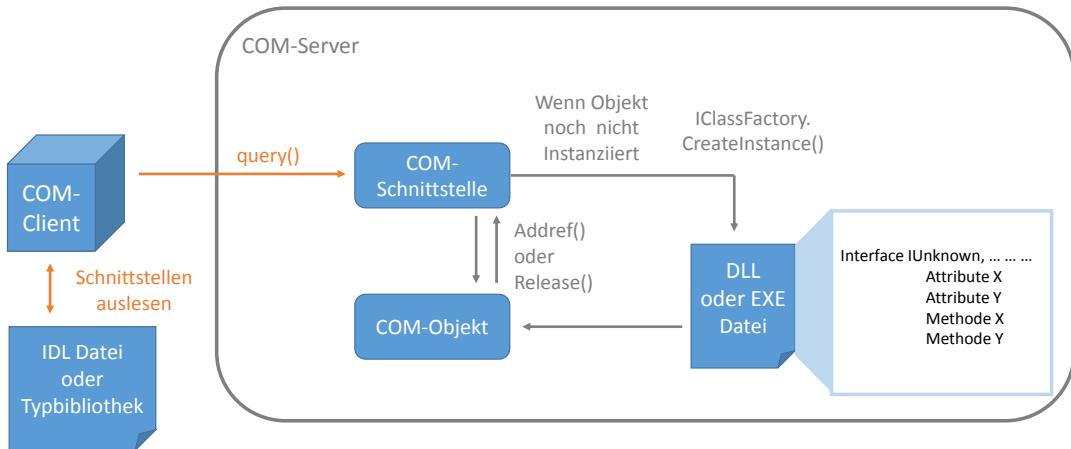


Abbildung 2.3: Bestandteile einer COM-Architektur

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 IDL (Interface Definition Language) zu Verfügung. Clients steht außerdem die Möglichkeit einer Query zur Verfügung, mithilfe derer festgestellt werden kann, ob ein Objekt das angefragte Interface unterstützt. Dabei wird lediglich die Query-Operation mit einer GUID als Parameter auf dem ausgewählten Objekt ausgeführt. Falls das Objekt das geforderte Interface unterstützt, liefert es den entsprechenden Pointer zum Interface zurück.

2.3.3 COM-Server

Ein COM-Server wird durch eine DLL oder ausführbare Datei realisiert, die eine COM-Komponente beinhaltet oder bereitstellt. Dabei unterscheidet man zwischen 3 Arten von COM-Servern. 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 zwischen Client und Server befindet. Dabei wird DCM (Distributed COM) eine spezielle Variante von COM verwendet. 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 zu einem Objekt zu manipulieren. Ein COM-Schnittstelle bezieht sich auf eine vordefinierte Gruppe von verwandten Funktionen, die eine Klasse implementiert, aber eine Schnittstelle muss nicht unbedingt alle Funktionen die die Klasse implementiert unterstützen. Eine Schnittstellenimplementierung wird mit einem Objekt verbunden, sobald eine Instanz des Objekts erzeugt wird 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 die das erledigt. 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 die der Lösung eines Problems dienen zu gruppieren. 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 Verhalten das die Entität umsetzen soll. 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 das selbst steuert. Wird ein Objekt Instanziert wird eine Referenzzähler hochgezählt. Dieser 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 (IDL) 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 dem Client und der Server-Programme. RPC Anwendungen benutzen die ACF-Datei um die Eigenschaften von Interfaces, die spezifisch für die Hardware und das Betriebssystem Operatoren sind zu beschreiben.

3. Systemanalyse

Zu Beginn der Arbeiten wird eine Systemanalyse zur Ermittlung des Ist- und Soll-Zustandes 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 steht der Kern der Systemanalyse, die Anforderungen, im Mittelpunkt. Aufbauend auf den Anforderungen werden in Abschnitt 3.3 die für die Umsetzung relevanten Daten ermittelt.

3.1 CAS genesisWorld

CAS genesisWorld ist ein Programm das die Organisation und Zusammenarbeit in Kundenbeziehungen und zwischen Kollegen steigert. 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. Dariüber hinaus lassen sich alle Daten beliebig miteinander verknüpfen. So werden zusätzliche Zusammenhänge deutlich und der Informationsgehalt erhöht sich. Ein Besprechungstermin lässt sich beispielsweise mit den Adressen der Teilnehmer und dem Dokument der Tagesordnung verknüpfen.

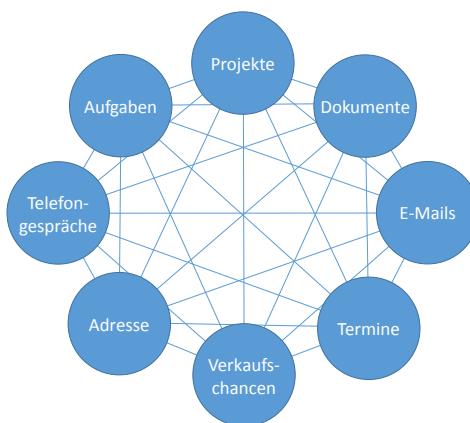


Abbildung 3.1: Schematische Darstellung der potenziellen Verknüpfungen

3.1.1 Architektur

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

- Die Präsentationsclients umfassen alle Dienste, die Benutzern die Informationen in Ansichten am Bildschirm zur Verfügung stellen.
- Der Applikationsserver umfasst alle Dienste, die die Geschäftslogik kapseln, Änderungen protokollieren, Benutzerrechte prüfen und die aufbereiteten Informationen den Präsentationsdiensten zur Verfügung stellen.
- Die Datenbankschicht umfasst alle Dienste, die zur Datenhaltung selbst notwendig sind.

3.1.2 Präsentationsschicht & Logikschicht

CAS genesisWorld Clients existieren in Form einer 32bit Windows Anwendung, sowie Mobile Clients in Android und iOS. Die Kommunikation der Mobile Clients findet über REST 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 für die Anmeldung und Rechteverwaltung sowie der DataService als zentraler Dienst für den Zugriff auf die CAS genesisWorld-Daten zur Verfügung. Die Schnittstelle des DataService wurde an Microsoft ADO angelehnt. Auf den Basisdiensten aufbauend existieren die Geschäftsdienste in Form der Schnittstellen der BusinessServices. Diese bieten spezielle Funktionen zu den jeweiligen Anwendungsbereichen.

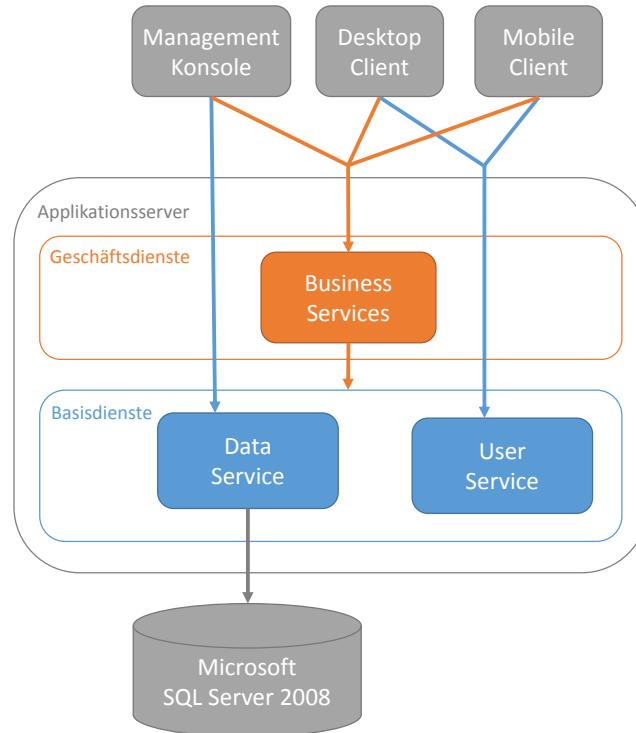


Abbildung 3.2: Schematische Darstellung der Architektur von CAS genesisWorld

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 Datensatz-Typen registrierten PlugIns. In Abbildung 3.3 wird anhand eines Beispiels der Vorgang dargestellt.

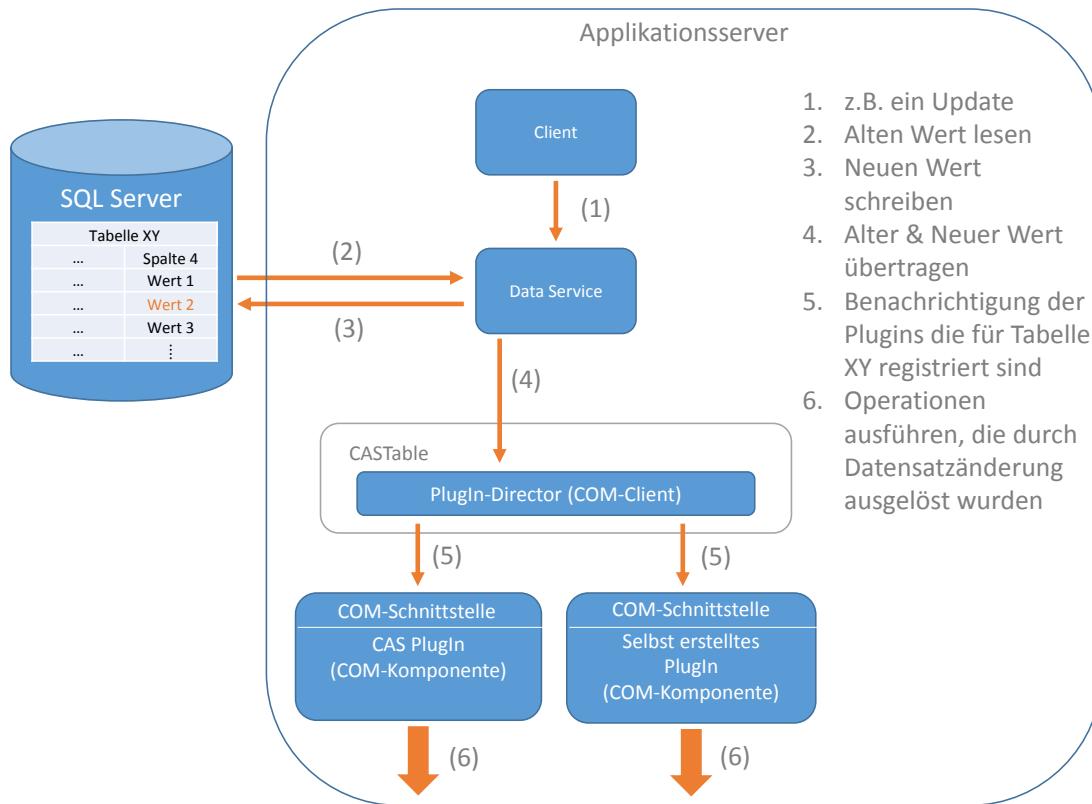


Abbildung 3.3: Beispiel zur Benachrichtigung von PlugIns anhand eines Ablaufs bei einem Update

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 damit die Möglichkeit, alte bzw. neue Werte von Feldern zu untersuchen und zu vergleichen und auf das Ergebnis zu reagieren.

Die Werte-Teilmenge 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 den SDK-PlugIns verwandte CAS PlugIns realisiert.

3.1.3 Datenhaltungsschicht

Die Datenhaltungsschicht wird durch einen Microsoft SQL Server 2008 (MSSQL) umgesetzt. Der SQL Server ist ein relationales Datenbankmanagementsystem (RDBMS) von

Microsoft, das für den Einsatz im Konzern-Umfeld konzipiert wurde. MSSQL verwendet T-SQL (Transact-SQL), eine Erweiterung von Sybase und Microsoft, die 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- / Fremdschlüsselbeziehungen hergestellt. In der CAS genesisWorld Datenbank werden nur Primärschlüssel verwendet. Die Beziehungen werden nicht wie sonst in mehreren Zwischentabellen realisiert sondern in einer einzigen Tabelle, die RelationTable. In Abbildung 3.4 ist ein Beispiel zu sehen.

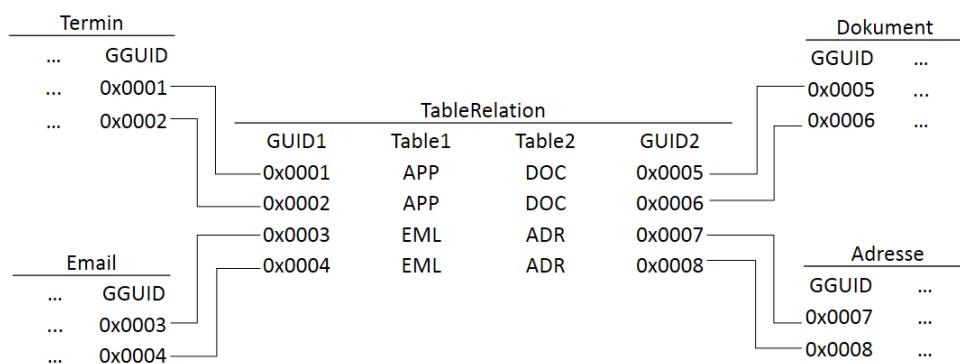


Abbildung 3.4: Funktionsweise von RelationTable anhand eines Beispiels

Die Spalten GUID1 und GUID2 beinhalten die jeweiligen Primärschlüssel der in Beziehung zu setzenden Tabellen. Als Zuordnungsmerkmal für die GGUID's zu Ihren Tabellen dienen die Spalten TableSign1 und TableSign2, deren Werte Kürzel der Tabellennamen sind. 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 ein 16 stelliger Binär-Wert. Alle Tabellen besitzen eine Spalte mit solchen Werten, was eine Datenintegrität in der gesamten Datenbank sicherstellt.

3.2 Anforderungsanalyse

Während der Anforderungsanalyse wird ermittelt welche Eigenschaften und Fähigkeiten das System zur Erreichung des Ziels benötigt. Wir unterscheiden bei der Einteilung der Anforderungen zwischen Funktionalen und nicht Funktionalen. Wie der Name bereits verrätet wird bei dem erst genannten die Funktionalität des zu erstellenden Systems beschrieben. Unter die Kategorie der nicht Funktionalen fallen alle anderen Anforderungen.

3.2.1 Funktionale Anforderungen

Folgende funktionalen Anforderungen wurden festgelegt:

- Ermittlung der Anzahl der Beziehungen zwischen zwei Personen
 - Ranking der Ergebnisse
 - Rückgabewert beinhaltet Gesamtwert der Abfrage, sowie die einzelnen Werten aus denen er sich zusammensetzt

- Begrenzen der Ergebnisse durch die Anzahl der zurückgelieferten Personen
- Möglichkeit zum eingrenzen der Ergebnisse auf einen Zeitraum
- Ein- und Ausblenden von Suchkriterien ohne eine neue Anfrage senden zu müssen
- Manuelles gewichten der Suchkriterien
- Gewichten der 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 Persönliche Kontakt Personen
 - vermindern um Personengruppen

3.2.2 Nicht funktionale Anforderungen

Folgende nicht funktionale Anforderungen wurden erhoben:

- In Memory Datenbanken in Betracht ziehen, bei Tauglichkeit einsetzen und testen
- Nur eine Rechner Instanz für Datenbank und Applikationsserver
- Antwortzeiten des Systems möglichst gering halten
- Beachtung der Portabilität, damit der entwickelte Prototyp einfach auf anderen Systemen installiert werden kann.
- Lose Kopplung zwischen Präsentationsschicht und Logikschicht
- Daten auf der Oberfläche graphisch aufbereitet darstellen; keine Konsolen Eingabe/Ausgabe

3.3 Ermittlung relevanter Daten

Die Datenbank der CAS Software AG umfasst 398 Tabellen, die zusammen wiederum 11.620 Spalten beinhalten. Aufgrund fehlender Dokumentation über die Umsetzung der darüberliegenden Anwendungsschicht und der Abwesenheit von 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 das bei der Bewertung von Beziehungen zwischen Personen, die Person selbst dabei immer den Ausgangspunkt der Suche darstellt. Aus Datenbanksicht bedeutet das zuerst eine Tupel der SysUser Tabelle mit Ihrer GGUID herangezogen wird. Die GGUID ist dabei der erste Wert nach dem in der gesamten Datenbank gesucht wird. Sobald alle Tabellen gefunden wurden die den Wert beinhalten werden deren GGUIDs für die weitere Suche benutzt. Die Menge der Suche wird dabei nach jedem Schritt um die Teilmenge der bereits gefundenen Tabellen verringert. Die Suche wird abgebrochen sobald die Suchmenge keine Werte mehr enthält oder keine Tabellen mehr mit den entsprechenden Werten gefunden werden konnten. Durch dieses Vorgehen erhält man am Ende alle Beziehungen aufbauend auf der Annahme das

die GGUID als Referenzierungswert benutzt wurde. In Abbildung 3.5 ist ein schon auf das Wesentliche Reduzierter Ausschnitt des Ergebnisses zu sehen.

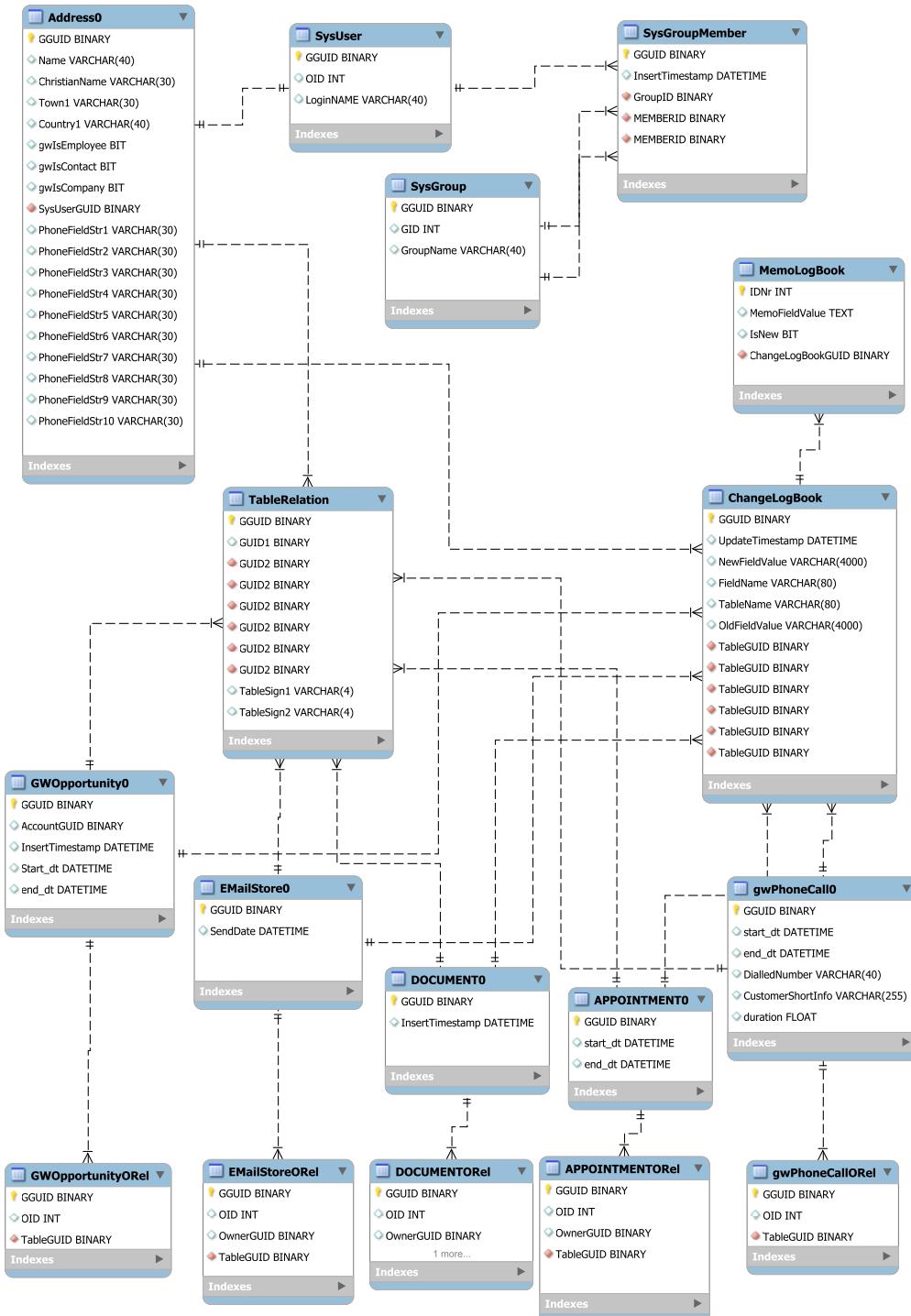


Abbildung 3.5: Auszug aus dem Schema des MSSQL 2008

Von den ursprünglich 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 wurden der Übersicht halber entfernt. Tabelle SysUser besitzt drei Spalten die von Bedeutung sind. Eine davon ist die GGUID die im folgenden nicht mehr 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 in der Oberfläche beim Anmelden in das neue System wieder verwendet werden.

Aufgrund der Anforderung an Filterung durch Gruppen wurden die Tabellen SysGroup-Member und SysGroup hinzugezogen. SysGroup besitzt eine GID was das gleiche für Gruppen ist wie die OID für Personen. Das Attribut GroupName wird für die Anzeige an der Oberfläche benötigt damit der Nutzer erkennen kann welche Gruppe er gerade ausgewählt hat. SysGroupMember stellt die Auflösungstabelle zwischen SysUser und SysGroup dar. Die Spalte GroupID beinhaltet Werte aus der SysGroup-GGUID Spalte. MemberID folgt dem gleichen Ansatz mit den SysUser-GGUID Werten. In Hinsicht auf die Logikschicht wird die Spalte InsertTimestamp für Veränderungen innerhalb der Gruppen benötigt.

Die Adress0 Tabelle wird für die restlichen 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. Name und ChristianName beinhalten den Vor- und Nachname der Person, welche bei der Zuordnung der Ergebnisse an der Benutzeroberfläche hilfreich sind. Vor ausgreifend ist zu sagen das nicht alle Telefongespräche über die dafür bestimmten Tabellen ermittelt werden können. Deswegen werden die Attribute PhoneFieldStr1-10 benötigt, mithilfe derer eine Zuordnung ermöglicht wird.

Die TableRelation enthält wie in Abschnitt 3.1.3 behandelt die Beziehungen der Datenbank. Auf das Szenario bezogen wird für die GUID1 die GGUID der SysUser benutzt. Mithilfe der GUID2 können die mit der Person verknüpften Termine, Verkaufschancen, Telefonate, Dokumente und Emails ermittelt werden. TableSign1 und TableSign2 werden zur Identifikation der jeweiligen Tabellen benötigt.

GWOportunity0 enthält alle Informationen zu 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 GWOportunity0. Bei EMailStore0 ist zu erwähnen das SendDate zur zeitlichen Einordnung verwendet werden kann. Das Attribut DialledNumber 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 das alle Beziehungen über die TableRelation bestimmt werden können. Dies trifft allerdings nicht ganz zu. Dort werden nur die an der Oberfläche manuell Verknüpften Tabellen aufbewahrt. Die restlichen Beziehungen können aus den ORel-Tabellen ermittelt werden. Jeder dieser Tabellen enthält eine OID bzw. GID zur Bestimmung der beteiligten Personen und/oder Gruppen.

Eine Betrachtung basierend auf Zeitspannen impliziert das Veränderungen über Daten über die Zeit. Um diese Änderungen zu erfassen wird die Tabelle ChangeLogBook benötigt. NewFieldValue enthält den neuen Wert eines Feldes. Wohingegen OldFieldValue den alten Wert des Feldes enthält. Die Spalte des geänderten Wertes ist inFieldName hinterlegt. Der Name der Tabelle ist der Spalt 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 die in der Tabelle MemoLogBook abgelegt. Dort wird der Datentyp Text verwendet der eine maximale Zeichenfolge von $2^{31}-1$ erlaubt.

4. Analyse ausgewählter Datenbanken

Ein Ziel der Bachelor Thesis ist es, hohe Performance in der Beantwortung der Benutzeranfragen zu erreichen. Die maßgebende Komponente ist in diesem Fall 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 aus der Literatur bekannte 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 aus der Literatur bekannte Datenbanken vorgestellt. Dabei wird insbesondere versucht, einen guten Überblick über die Charakteristika der einzelnen Datenbanken zu geben. Der dahinter stehende Gedanken ist, dass ein späterer Vergleich und Entschluss zur Nutzung besser nachvollziehbar sind.

4.1.1 CouchDB

CouchDB [Cou13] ist eine dokumentenorientierten 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. Indizes sind in CouchDB B-Bäume, so dass 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. Lesezugriffe können auf beliebigen Server stattfinden, wenn Aktualität keine Rolle spielt. Updates hingegen müssen an alle Server weitergegeben werden. CouchDB unterscheidet sich von anderen Systemen dadurch, dass es Eventual Consistency akzeptiert. Jeder Client erhält eine in sich selbst konsistente Ansicht der Datenbank. CouchDB implementiert Multi-Version Concurrency Control (MVCC) auf einzelne Dokumente mit einer Sequenz-ID die für jede Version eines Dokuments generiert wird. CouchDB benachrichtigt eine Anwendung, wenn jemand anderes das Dokument aktualisiert hat seit dem es zuletzt gefetched 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 hat einige Ähnlichkeiten mit CouchDB. Beide bietet Indizes auf Collections, sie sind lockless, und bieten einen Dokument-Abfragemechanismus. Es gibt jedoch wichtige Unterschiede:

- MongoDB unterstützt automatische Fragmentierung die Dokumenten 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, MongoDB hingegen nutzt atomare Operation auf Feldern

MongoDB speichert Daten in einem JSON-ähnlichen binären Format namens BSON. BSON unterstützt boolean, integer, float, Datum, String-und Binär-Typen. Client-Treiber verschlüsseln die lokalen Dokumentendatenstruktur 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 für höhere Leistung, jedoch können einige Updates bei einem Crash verloren gehen.

4.1.3 Voldemort

Projekt Voldemort [Vol13a] ist ein verteiltes Key-Value-Store (entwickelt von LinkedIn), der 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 values komplexere Datenstrukturen wie Maps oder Listen beinhalten. Voldemort stellt 4 verschiedene Operatoren zur Datenmanipulation bereit:

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

Eine Möglichkeit für Range Scans ist dabei 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 einem 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 keine Client-Seitiges Hashing oder feste Schlüsselräume wie in einigen NoSQL-Systeme gibt. Insbesondere stellt es lineare und modulare Skalierbarkeit sowie streng konsistenten Datenzugriff und automatisches und konfigurierbares Fragmentierung von Daten zu Verfügung. Auf Tabellen können in HBase über eine API zugegriffen werden. Sie können jedoch auch als Ein-und Ausgang Daten 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 Spalte 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 das 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 Sekunden Takt aus. Ein Commit-Log auf jedem Knoten fängt Schreibaktivität ab um Daten Haltbarkeit zu gewährleisten. Daten werden auch auf eine In-Memory Struktur geschrieben, die memtable. Anschließend in eine Datendatei auf der Festplatte geschrieben genannt SSTable , sobald die Speicherstruktur voll ist. Alle Schreibvorgänge werden automatisch aufgeteilt und in Cluster repliziert.

Cassandras Datenmodell basiert auf einem partitionierten Row-Store mit variabler Konsistenz. Zeilen werden in Tabellen organisiert, die erste Komponente des Primärschlüssels einer Tabelle ist der Partition Schlüssel. Innerhalb einer Partition werden Zeilen nach den verbliebenen Spalten des Primärschlüssels geclustert. Andere Spalten können getrennt von dem Primärschlüssel indiziert werden. Was Cassandra von HBase unterscheidet sind Spalten, die in einer verschachtelten Weise in Spalte Familien gruppiert werden können. Konsistenz Anforderungen, die zum Zeitpunkt der Abfrage angegeben werden können stellen auch ein Unterscheidungsmerkmal dar. Weiterhin ist Cassandra ein schreiborientiertes System während HBase entwickelt wurde um hohe Leistung für intensive Lese Workloads zu erzielen.

4.1.7 VoltDB

VoltDB [Vol13c] ist ein ACID-konformes relationales In-Memory-Datenbanksystem abgeleitet vom Forschungsprototyp H-Store [KKN⁺08]. Es basiert auf einer Shared-Nothing-Architektur und wurde entwickelt, um auf einem Cluster mit mehreren Knoten zu laufen. Dies wird erreicht indem man die Datenbank in getrennte Partitionen aufteilt bei dem jeder Knoten der Besitzer und Verantwortliche für die jeweiligen Partitionen ist. Durch Verwendung von Stored procedures 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, das 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 referenzielle Integrität, Transaktionen, Clustering, Datenkompression, Verschlüsselung und SSL. Die Datenbank kann im Embedded- oder Server-Modus betrieben werden.

4.2 Gegenüberstellung

Um eine übersichtliche Gegenüberstellung der Datenbanken zu erreichen wurde die Tabelle 4.1 angefertigt. Sie enthält vergleichbare Eigenschaften von Datenbanken auf die im folgenden eingegangen wird.

Die erste Eigenschaft ist das Erscheinungsjahr. Der Grund für die Aufnahme ist das Datenbanken die länger auf dem Markt sind meist eine höhere Reife aufweisen können. Bewährte Datenbanken haben bereits viele ihrer anfänglichen Probleme behoben was den Umgang mit Ihnen erleichtert. Natürlich sind ältere Systeme nicht frei von Fehlern jedoch existieren für diese viele Workarounds oder andere Lösungsansätze. Cassandra zum Beispiel hat neben zahlreichen Bugfixes über die Jahre CQL als Query Sprache, MapReduce Support, sekundäre Indizes, verbesserte Komprimierung und vieles weiteres erhalten. Es gibt 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 der Wirtschaftlichkeit sind Möglichkeiten der Manipulation des Source Code ein Pro für Open Source Produkte. Kommerzielle Lizenzen bieten hingegen eine höhere Zukunftssicherheit als Open Source Produkte. Sollten die Entwickler der Open Source Produkte keine Lust oder Zeit mehr haben kann die Entwicklung jederzeit eingestellt werden.

Unterstützte Programmiersprachen und Betriebssysteme sind Eigenschaften bei denen eine Betrachtung des Ist-Zustandes sinnvoll ist. Im Unternehmen sollte optimaler weise schon Erfahrung in Form von Wissen über die verwendeten 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. Durch Schemafreiheit gewinnt man nämlich an Flexibilität. Wohingegen man durch die Nutzung eines Schemas eine bessere Kontrolle der Daten im DBMS besitzt.

Tabelle 4.1: Gegenüberstellung von den Charakteristika der Datenbanken

Eigenschaft	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-Betriebssysteme	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
Datenschema	schemafrei	schemafrei	schemafrei	schemafrei	schemafrei	schemafrei	ja	ja
Typisierung	nein	ja	nein	ja	nein	nein	ja	ja
Sekundärindizes	nein	eingeschränkt	ja (über Views)	ja	nein	nein	ja	ja
SQL	nein	nein	nein	nein	nein	nein	ja	ja
APIs und andere Zugriffskonzepte	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
Unterstützte Programmiersprachen	C, C#, C++, Groovy, Java, PHP, Python, Scala	C#, C++, Java, Perl, PHP, 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
MapReduce	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
Transaktionskonzept	nein	nein	nein	nein	optimistisches Locking	nein	ACID	ACID
Nebenläufigkeit	ja	ja	ja	ja	ja	ja	ja	ja
Embeddable	nein	ja	ja	nein	nein	ja	ja	ja
In Memory fähig	nein	nein	nein	nein	ja	hybrid	ja	ja

Sekundärindizes sind gerade in Anbetracht der Forderung nach Performance eine interessante Fähigkeit von Datenbanken. 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 in der Regel 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.

Die in der Datenbank verwendete Schnittstelle zur Client-Server Kommunikation spielt

bei der Architektur des gesamt Systems eine Rolle. Wird der Applikationsserver z.B. auf dem gleichen Server betrieben wie die Datenbank, ist eine Schnittstelle die nur über das Netzwerk angesprochen werden kann nicht sehr sinnvoll. Wird die Datenbank hingegen über das Netzwerk von eventuell mehreren Servern angesprochen, so eignet sich der Einsatz von REST-Protokollen.

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 darüberliegende 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 Netzwerzkugriffe bei der Datenabfrage vermieden werden.

Die Eigenschaft In Memory spiegelt den Wunsch der CAS Software AG wieder. Sie stellt somit einen der wichtigsten Kriterien 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 entscheidender Bedeutung ob die entsprechende Datenbank den Anforderungen an das Gesamt System gerecht werden kann.

Dementsprechend ist in diesem Anwendungsfall die Performance einer Datenbank am bedeutsamsten. Die Verwendung des Hauptspeichers als Primärspeicher bedeutet einen theoretischen Geschwindigkeitsvorteil um den Faktor 100.000. In der Realität allerdings spielen bei der Abfragegeschwindigkeit viele verschiedene Faktoren eine Rolle. Trotzdem dürfte die Geschwindigkeitsvorteile 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 hier ersten nicht notwendig sondern auch 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 sind Funktion wie SUM werden nicht in der 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 muss ist Ihre mangelnde Komplexität. Sie sind besonders für Punkt-Abfragen geeignet.

Komplexe Anfragen sind nur durch deren Realisierung in der Logikschicht möglich. Was eine enormen Steigerung im Aufwand sich bringen würde. Daher entschied man sich auch gegen die Key-Value-Stores.

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 erhofft man sich hohe Geschwindigkeitsvorteile gegenüber herkömmlichen relationalen Systemen. 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 der groben Strukturierung eines Softwaresystems. Zur Gestaltung werden neben funktionalen und nicht funktionalen Anforderungen, technische und organisatorische Einflussfaktoren hinzugezogen. Für erste Überlegung werden alle Komponenten die zur Realisierung benötigt werden in Verbindung zueinander gesetzt. Weiterhin sind Technologien die zur Umsetzung von bestimmten Komponenten verwendet werden können festzulegen. Anschließend werden die Entwürfe der einzelnen Komponenten im Detail erstellt.

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 von nichts abhängig und kann jederzeit separat betrieben werden.

Die Vaadin Client-Side Engine verwaltet das Rendering der Oberfläche im Web-Browser durch den Einsatz verschiedener Client-Seitiger Widgets, die das Gegenstücke zu den serverseitigen Komponenten bilden. Es leitet Benutzerinteraktionen an die Server-Seite weiter und rendert anschließend die Änderungen für die Benutzeroberfläche. Die Kommunikation findet über asynchrone HTTP-oder HTTPS-Anfragen statt.

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

Interaktion mit dem Benutzer-Interface-Komponenten erzeugen Events, die zunächst auf der Client-Seite durch die 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 durch den Benutzer eingegeben Werte.

In der Server.war werden REST-Requests entgegen genommen. Anhand der mit übertragenen Filteroptionen werden die Bedingungen für die Datenbank Abfrage zusammengestellt. Anschließend wird eine Verbindung zur H2-Datenbank aufgebaut falls noch keine

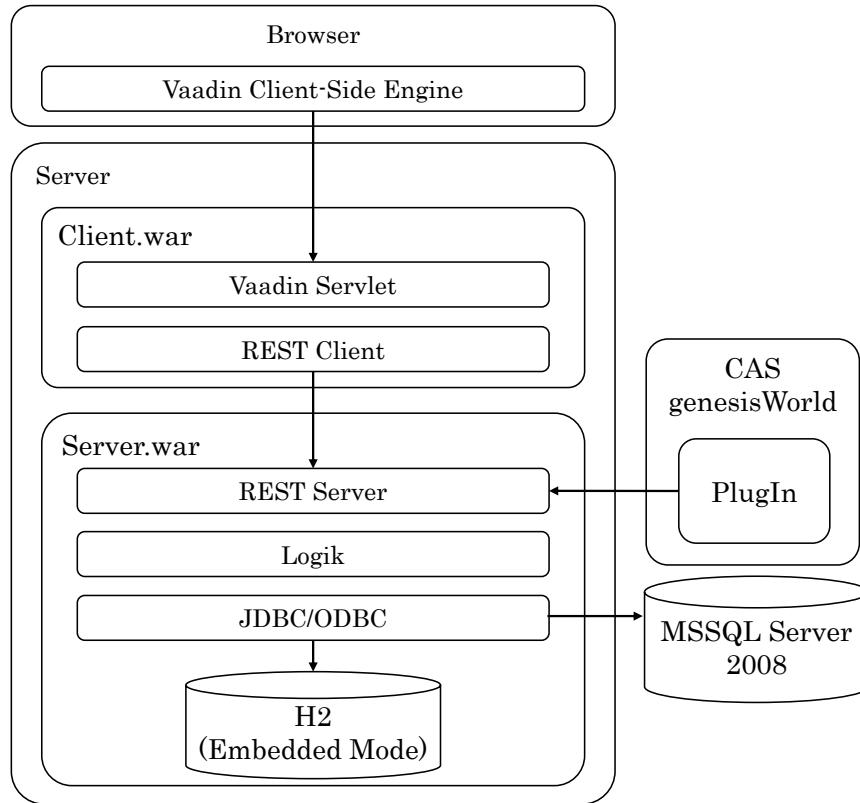


Abbildung 5.1: Architektur des Systems

besteht. 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 beinhaltet. Die Verwendung des REST-Protokolls zwischen der Client.war und Server.war stellt den nächsten Schritt der losen Kopplung dar. Einer der Vorteile ist das die Funktionen des Systems durch andere Clients abgegriffen werden können ohne Änderungen an bestehendem durchführen zu müssen. Beide WAR-Dateien werden in einem Apache Tomcat Webserver deployed und können über URLs angesprochen werden.

Die Logikkomponente in der Architektur stellt eine Zusammenfassung aller Funktionen des Anwendungskerns dar. Sie kümmert sich um die Generierung der Abfragen an die Datenbank. Welche eine dynamisch Generierung der Abfragen vornimmt, um durch keine unnötigen 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 stellt allerdings einen der wichtigsten Schritte für die Umsetzung dar.

Um nicht periodisch Extraktion und Transformation wiederholen zu müssen wird ein selbstgeschriebenes PlugIn im CAS genesisWorld Anwendungsserver eingesetzt. Die Grundaidee des PlugIns, ist Benachrichtigungen über Änderungen an unser System zu übermitteln. Dort wird Kontrolliert ob der Datensatz eine Relevanz für uns besitzt und besorgt

sich 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. OpenSCV wird für das Lesen und Schreiben von CSV-Dateien verwendet. JDBC wird zur Kommunikation zwischen Anwendungsserver und der Datenbank verwendet. Die H2-Datenbank stellt die Datenquelle des Systems dar. Im folgenden werden alle Bestandteile bis auf den H2 der bereits erläutert wurde, etwas näher beschrieben.

Vaadin

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

Die aktuelle Version Vaadin , wurde im Februar 2013 veröffentlicht. Der bekannteste Wechsel von Vaadin 6 war die Integration von GWT zu Vaadin, die eine bessere Unterstützung für die clientseitige Widget Entwicklung bedeutete 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 definieren. Jersey ist die Referenzimplementierung für diese Spezifikation. Jersey enthält im Grunde einen REST-Server und einen REST-Client. Auf der Serverseite verwendet Jersey ein Servlet die vordefinierten Klassen abtastet um REST-Ressourcen zu identifizieren. Über die web.xml Konfigurationsdatei werden von der Jersey-Distribution bereitgestellt Servlets registriert. Dieses Servlet analysiert die eingehenden HTTP-Anforderung und wählt die richtige Klasse und Methode für diese Anfrage aus. Diese Auswahl basiert auf Annotationen in der Klasse und Methoden. Weiterhin unterstützt JAX-RS die Erstellung von XML-und JSON über die Java Architektur für XML Binding (JAXB).

Apache Tomcat

Tomcat ist ein Open-Source-Web-Server entwickelt von der Apache Group. Apache Tomcat implementiert die Java Servlet und der Javaserver Pages (JSP) Spezifikationen von Sun Microsystems und stellt daher eine Referenzimplementierung dar. Er stellt weiterhin eine rein auf Java basierte HTTP Web Server 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. Opencsv ist eine sehr einfache CSV-Parser-Bibliothek für Java. Die Bibliothek kann zum erstellen, lesen und schreiben von CSV-Dateien benutzt werden. Der beste Teil der OpenCSV Parser ist, es nimmt eine CSV-Datei und mapt die Ergebnisse auf ein Java-Bean-Objekt.

JDBC

Die JDBC-API ermöglicht den programmgesteuerten Zugriff auf relationale Daten aus der Java Programmiersprache. 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 einem verteilten, heterogenen Umgebung interagieren.

5.3 Datenbank

...

5.3.1 Schema der Datenbank

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 vor der Entscheidung wie viele Information aus dem alten System in das Neue übernommen werden sollen. Um höchst mögliche Performance zu erreichen werden lediglich die für das Szenario benötigten Daten extrahiert. Allerdings entsteht durch nachträgliche Hinzunahme 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 über die Verbindungen. 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 eine Unterscheidung zwischen den Verbindungen zu erhalten wird der Typ in der Spalte DataTyp vermerkt. Die letzte Spalte endID beinhaltet die Personen zu denen die Verbindung letztendlich führt. 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 beinhaltet voraussichtlich Millionen von Werten. Es gibt jedoch nur 193 Länder auf der Welt. Das bedeutet das Wörter wie Deutschland sich sehr oft wiederholen. Die Spalte Country ist vom Datentyp Varchar welches pro Zeichen 2 Byte benötigt. Das wären beim Wort Deutschland 22 Byte. Legt man nun für die Spalte

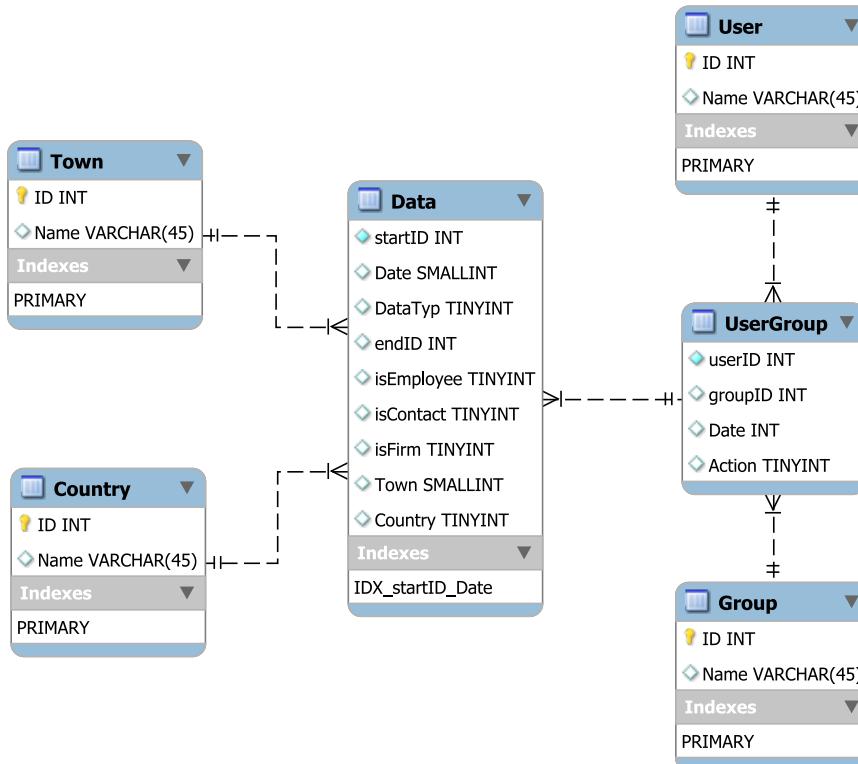


Abbildung 5.2: Neue Datenbankschema

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 jetzt nur noch die Zahlen anstatt des vollständig ausgeschrieben Wortes verwendet. Das würde beispielsweise mit dem Wort Deutschland eine Reduktion von 22 Byte auf 1 Byte bewirken. Die Reduzierung auf 1 Byte lassen 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. Stimmt oder stimmt nicht. Der Datentyp TINYINT reicht daher zur Abbildung des Zustandes völlig aus. Ein Feld vom Typ datetime benötigt 8 byte an Speicher. Um hier ebenfalls Einsparungen vorzunehmen wurde beschlossen das Datum als SMALLINT zu deklarieren. Dies ist möglich da nur das Tages-Format 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. Darauf aufbauend wird das Datum repräsentiert durch die Differenz in Tagen zum Nullpunkt in der Spalte Date abgelegt.

Möchte man nun die Abfrage eines Benutzers die eine Filterung anhand einer Stadt voraussieht beantworten, muss man zuerst an die ID der Stadt herankommen. Dabei können zwei verschiedene Ansätze verfolgt werden. Der erste Ansatz wäre einen Join zwischen Town und Data durchzuführen und direkt mit dem Namen der Stadt 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 zur besseren Performance führen. Dieses Vorgehen kann für die Stadt, das Land und die Gruppenzugehörigkeit angewendet werden.

Die Tabelle GroupDate unterscheidet sich von den anderen Auslagerungstabellen, den dort

sind noch weitere Details vermerkt. Diese ermöglichen es die Zusammenstellung von Gruppen über die Zeit nachzuvollziehen. Action legt fest ob die Tupel einen Eintritt einer Person oder einen Austritt darstellt. Die Spalte Date beinhaltet das Datum des Ereignisses. Mithilfe beider Attribute lassen sich Gruppenzusammenätzungen auf bestimmte Zeitpunkte bezogen rekonstruieren.

5.3.2 Zugriffsstrukturen

Indizes dienen der Beschleunigung von Suchen nach bestimmten Spaltenwerten. Ohne Indexe 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 Performance an. Zu Beachten ist jedoch das 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. Indiziert werden in unserem Schema die Spalte Name in der jeweiligen Tabelle, da der Client nur die jeweiligen Namen besitzt. Mithilfe der Namen werden die zugehörigen IDs ermittelt. Die Nutzung von Hash-Indizes bringt allerdings Limitierungen mit sich. Eine der wichtigsten ist das 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 Standard Index von H2 der ein B⁺-Baum verwendet. Dieser kann für die Spalte userID verwendet werden, der den ersten Wert einer Suche darstellt. Der B⁺-Baum eignet sich auch für die Data-Tabelle. 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. Geclusterete 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 wertschöpfende Geschäftsprozesse innerhalb eines Unternehmens. Sie sind demnach auf die Steuerung und Überwachung des Tagsgeschäftes 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 2008. 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 Betrachtung in Bezug auf Zeitspannen dar. Bei Analysen über die Zeit ist zu beachten das Daten sich im Laufe der Zeit verändern und dies

zu berücksichtigen ist. Die Tabelle Changelogbook bietet uns die Möglichkeit Veränderungen in den Datensätzen nach zu vollziehen. Eine solche Veränderungen über die Zeit ist in der Gruppenkonstellation zu finden, da Personen Gruppen verlassen oder beitreten können. Neben den Veränderungen über die Zeit sind Datensätze die über mehrere Tage gehen gesondert zu behandeln. Termine wie Tagungen und der gleichen erstrecken sich beispielsweise über mehrere Tage. In der MSSQL Datenbank werden diese Termine als ein Datensatz beachtet. In unserer Analyse stellt jede Tupel eine Verbindung dar. Jede Tupel besitzt einen Tag der für die Analyse herangezogen wird. Somit muss ein Datensatz der sich über mehrere Tage erstreckt in der H2-Datenbank in mehrere Teile aufgeteilt werden. Deswegen sollte im Ergebnis der SQL-Query vermerkt werden das dieser Termin mehrere Tage ging damit das in der Transformation berücksichtigt werden kann.

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 aus Bequemlichkeit geschoben. Das hat zur Folge das 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 Feld 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 das der Termin nach dem angegeben Zeitpunkt geschoben wurde. Werden Termine aus Gründen wie auch immer geschoben findet dies in der Regel vor dem Start des Termimes statt damit die Personen nicht unnötig zu dem Termin auftauchen. Die zweite ist das der neue Termin in der Zukunft liegt. Neben den beiden Bedingungen ist zu beachten das die Operation die auf die Datensätze ausgeführt wurde eine Update war.

Zur Zwischenspeicherung der Ergebnisse der Extraktion wird eine Ablage der in CSV-Dateien vorgenommen. Dieser Zwischenschritt dient der Nachvollziehbarkeit zwischen der Beschaffung der Daten und deren Transformation. Bei der Fehlersuche beispielsweise kann dies sehr hilfreich sein.

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 Zweckessig 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. Verbindungen in Form von E-Mails, Terminen usw. müssen auch in eine einheitliche Form gebracht werden. Spalten die gleiche Inhalte besitzen aber unterschiedlich bezeichnet sind müssen unter einem Terminus zusammengeführt werden.

Die in der Extraktion genannten Besonderheiten werden durch unterschiedliche Query-Abfragen ermittelt. Das führt zu vielen separaten Dateien. Diese sind zum Abschluss der Transformation zusammen zu führen. Das Ergebnis wird anschließend in einer CSV-Datei abgelegt, welche die Basis für das Befüllen der H2-Datenbank bildet.

5.4.3 Load

Beim Laden der Datensätze in den H2 kommt ein sogenannter bulk load-Prozess zum Einsatz. Dieser wird häufig beim laden von großen Datenmengen aus einer Datei eingesetzt. Er ermöglicht ein wesentlich schnelleres Befüllen der Datenbank bei großen Datenmengen im Gegensatz zu den INSERT-Operationen.

5.5 Sicherstellung der Aktualität der Daten

Systeme die auf den Datenbestand anderer Systeme aufbauen, können zwei verschiedenen Ansätze zur Sicherstellung ihrer Aktualität verfolgen. Unser nebenläufiges System bezeichnen wir als A und das System das den original Datenbestand darstellt als B. Einer der Ansätze ist die Intervall basierende Nachfrage über Veränderungen von A. Hierbei fragt A bei B in festgelegten Intervallen nach ob Daten verändert wurden. Die Wahl des Nachfrage-Intervalls stellt dabei eine der größten Schwierigkeiten dar. Ist der Intervall zu groß, sinkt die Aktualität des Datenbestandes. Ist er zu klein entsteht bei B ein starke Belastung. Der andere Ansatz ist das B Benachrichtigungen an A über Veränderungen sendet. Dadurch werden keine unnötigen Abläufe angestoßen, da nur im Falle von neuen Daten 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 eine Veränderung des Legacy Systems ausschließen.

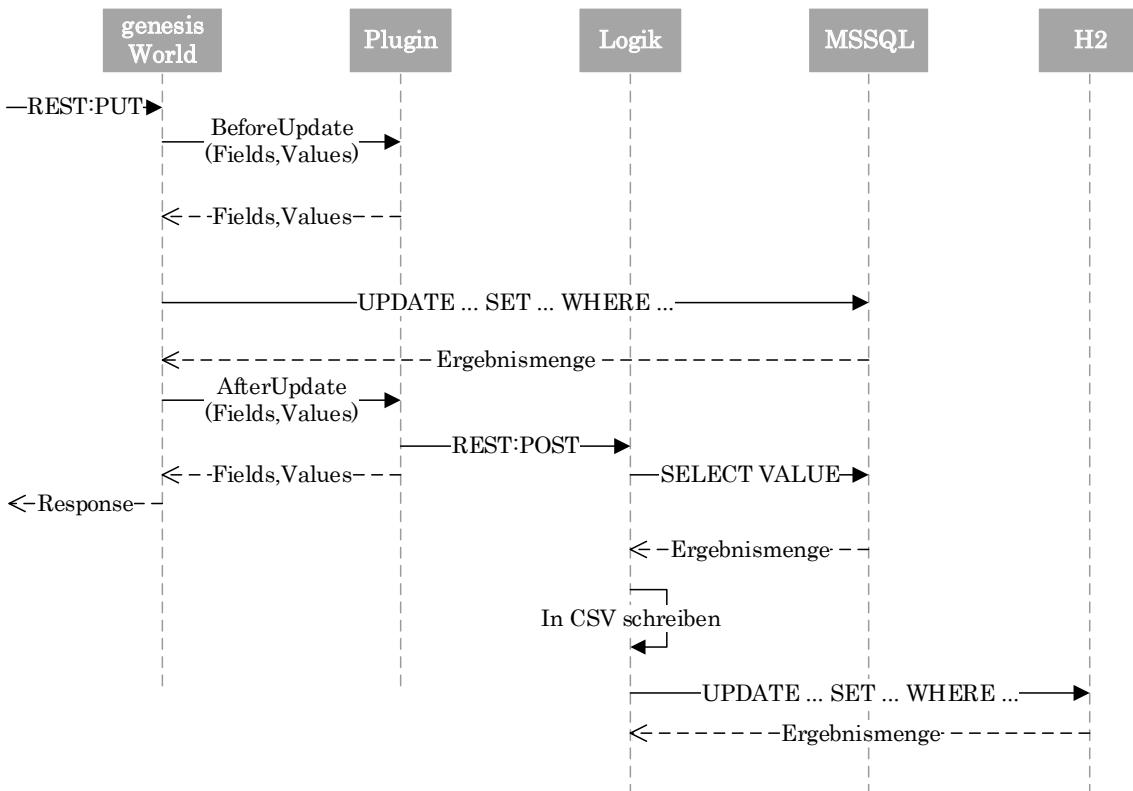


Abbildung 5.3: Sequenzdiagramm eines Updates

In CAS genesisWorld gibt es eine Möglichkeit den zweiten Ansatz umzusetzen. Die Idee dabei ist den Applikationsserver um ein sogenanntes Plugin zu erweitern das über Veränderungen in der MSSQL Datenbank benachrichtigt wird. Realisiert wird ein solches Plugin als COM-Objekt welches das Interface IGWSDKDataPlugIn implementiert. Das erstellte

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 den jeweiligen Datensatz-Typen registrierten PlugIns. Das Plugin selbst besitzt einen REST-Client der einen POST an die Logik sendet. Sie enthält die GGUID des Veränderten Datensatzes. Mithilfe dessen die Extraktion des betroffenen Datensatzes durchgeführt wird. Der neue/veränderte Datensatz wird zuerst in einer CSV zwischengespeichert und anschließend an die H2-Datenbank gesendet.

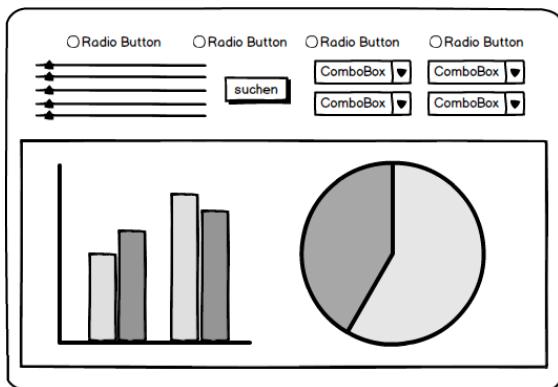
5.6 Darstellungskonzepte

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 einer Verbindung von interessiere sondern ihr Typ und ihre Häufigkeit zu einer bestimmten Person. Da keine Detailinformationen zur Verbindung 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.

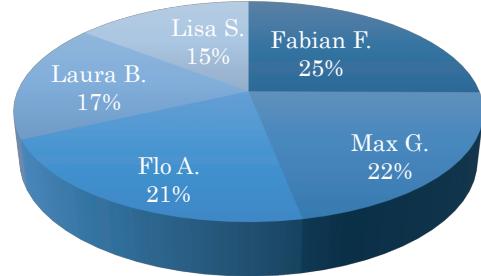
Nach dem Login findet eine Weiterleitung auf die eigentliche Seite statt. Dessen Aufbau ist in Abbildung 5.4 (a) zu sehen. Oben auf der Seite sind alle Regler, CheckBoxen und Eingabefelder zur Filterung der Ergebnismenge zu finden. Direkt darunter befindet sich das Diagramm das die Ergebnismenge einer Abfrage darstellt.

Diagrammtypen gibt es viele jedoch ergeben sich Einschränkungen durch das Framework welches zu Realisierung verwendet wird. Im Prinzip lässt sich jede Darstellung verwirklichen allerdings ist das Aufwand Nutzen Verhältnis zu berücksichtigen. In einer Vorauswahl wurden einige Typen ausgewählt die in Abbildung 5.4 (b)-(f) dargestellt sind.

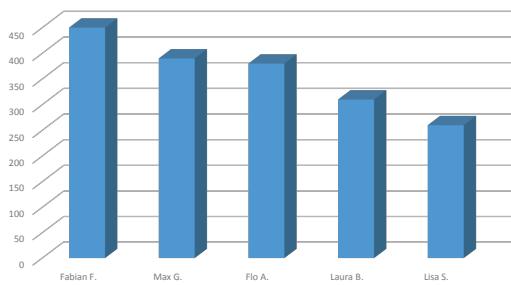
Netzdiagramme geben Eigenschaften verschiedener Systeme wieder. Sie eignen sich daher gut zur Darstellung von Ausprägungen. Für unsere Form der Daten ist diese Darstellung gänzlich ungeeignet. Mithilfe von Liniendiagrammen lassen sich Trends und Zeitreihen darstellen. Die Verwendung verschiedener Linien 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 in der Zeitreihe beinhaltet. Eine Tree Map dient der Abbildung von hierarchischen Daten. Dabei steht jede Fläche eines Rechtecks im proportionalen Zusammenhang zur Gesamtfläche. Die Beachtung von Größenverhältnissen stellt einen nützlichen Eigenschaft für unsere Daten dar. Eine Hierarchie innerhalb der Daten ist nicht im klassischen Sinn vorhanden. Vielmehr würde jedes Rechteck aus dem jeweiligen Anteilen der Verbindungstypen bestehen. Beispielsweise könnte die Person Ludwig Neer wiederum in Rechtecke unterteilt werden, die die Anzahl der E-Mails, Termine usw. enthalten. Das würde allerdings schnell zu einer schlechten Übersicht führen da viel zu viele Kacheln dargestellt werden müssen. 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 ein 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.



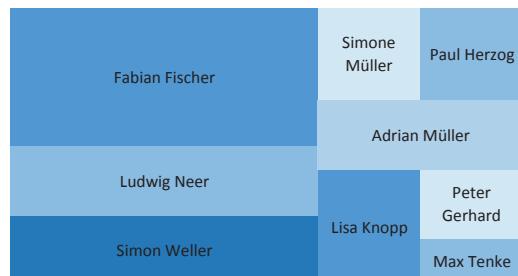
(a) Grober Entwurf des Aufbaus der Hauptseite



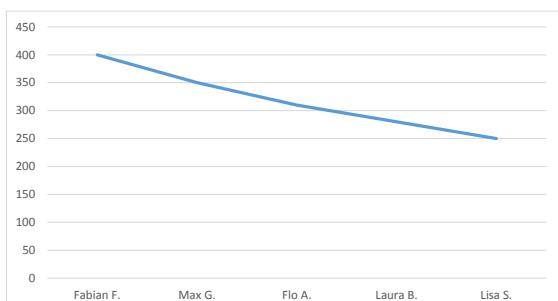
(b) Tortendiagramm



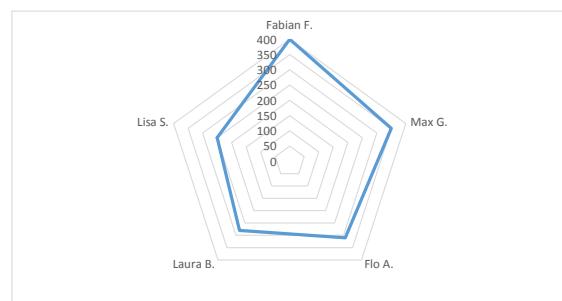
(c) Balkendiagramm



(d) Tree Map



(e) Liniendiagramm



(f) Netzdiagramm

Abbildung 5.4: Entwürfe der Oberfläche

6. Umsetzung

...

6.1 Kompression

Auf unsere Tabelle bezogen gibt es drei Spalten die für dieses Verfahren in Frage kommen. Zu einem die Spalte Datum, die 8 Byte pro Feld benötigt. Zum anderen Spalte Stadt mit 60 Byte pro Feld. Mit 80 Byte pro Feld stellt die Spalte Land das größte Einsparungspotenzial dar. Tabelle 6.1 zeigt das Ergebnis der Einsparungen. Zu beachten ist das bei der Spalte Datum kein Wörterbuch benötigt wird. Stattdessen wird ein selbstgewählter Nullpunkt festgelegt, der den Wert 0 besitzt. Sagen wir der Nullpunkt ist der 01.01.1990. Das bedeutet das der 10.02.1990 durch die Zahl 41 ersetzt werden würde.

Speicherplatzverbrauch ohne Wörterbücher

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 Wörterbücher

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 6.1: Vergleich des Speicherplatzverbrauchs

Name der Tabelle	Anzahl der relevanten Zeilen	Auflösung der GID
AppointmentORel	2.128.691	4.738.993
DocumentOREL	1.632.579	6.637.728
EMailStoreORel	1.287.419	2.728.035
gwPhoneCallORel	672.398	672.836
GWOportunityOREL	142.740	308.336
SysUser	3096	-
Adress0	3096	-
TableRelation	364.334	-
SysGroupMember	7662	
SysGroup	480	
Summe	6.242.495	

Tabelle 6.2: Zeilenanzahl

7. Ergebnisanalyse

...

7.1 Test Aufbau

...

7.2 Auswertung

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

Tabelle 7.1: Abfragegeschwindigkeit Vergleich

7.3 Handlungsempfehlungen

...

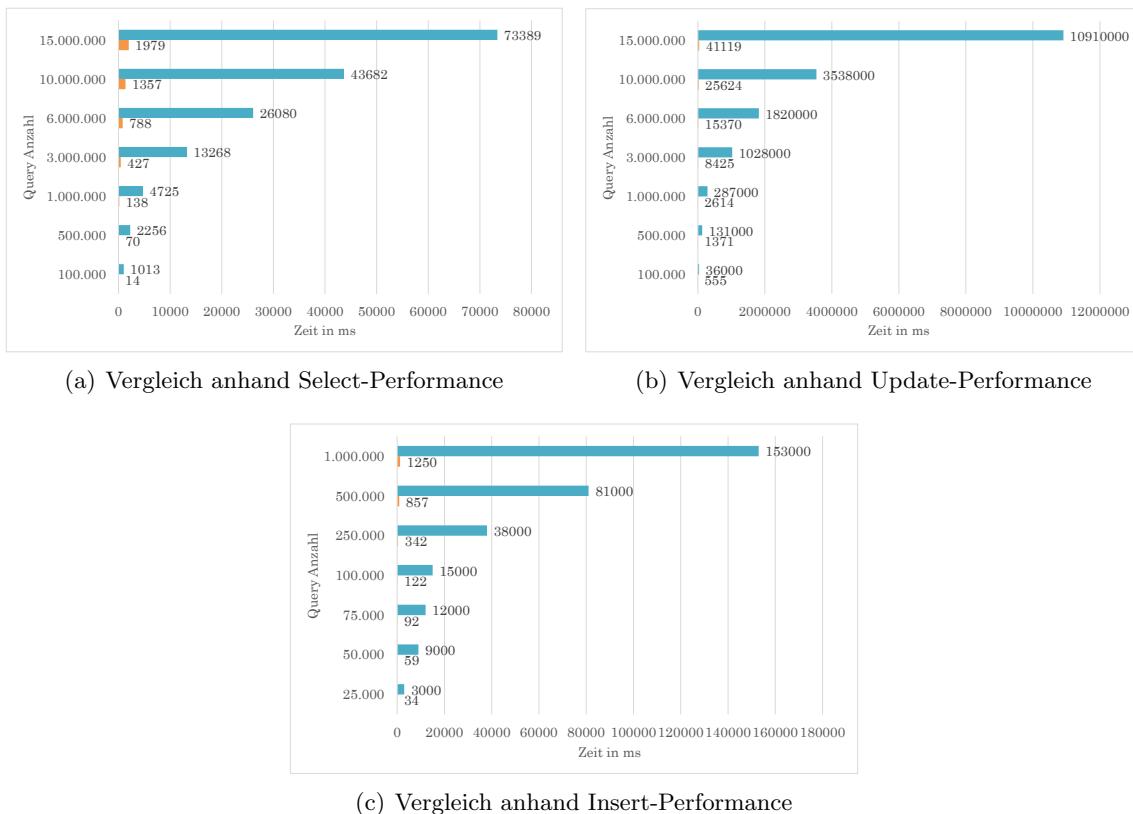


Abbildung 7.1: Abfragegeschwindigkeit Vergleich

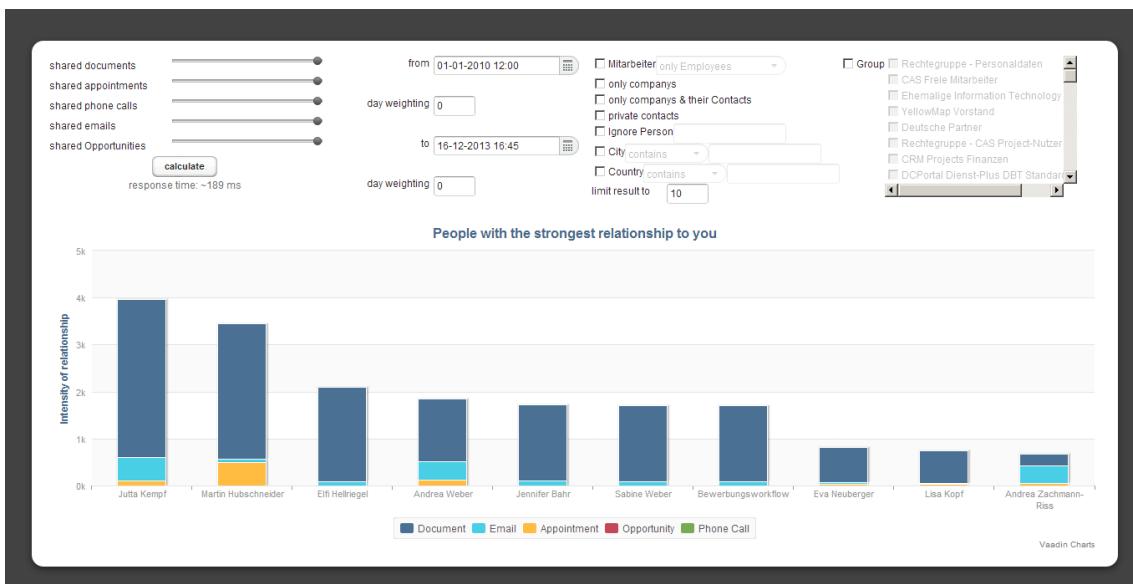


Abbildung 7.2: Oberfläche des Systems

8. Zusammenfassung

...

8.1 Ergebnis

...

8.2 Ausblick

...

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>.
- [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>.
- [Dr.00] A.Brewer Dr.Eric: *PODC keynote*. 2000. <http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf>, Online;accessed 27-November-2013.
- [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>.

- [HKJR10] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira und Benjamin Reed: *Zoo-Keeper: 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>.
- [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.
- [Pla13] 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>.
- [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. aufl. Auflage, 2011, ISBN 978-3-8266-9156-0; 3-8266-9156-3. http://deposit.d-nb.de/cgi-bin/dokserv?id=3872660&prov=M&dok_var=1&dok_ext=htm;http://d-nb.info/1014629934/04, Seiten : 176 - 182.
- [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].

