Fachbereich Informatik und Medien

Bachelorarbeit

Evaluation von SQL- und NoSQL-Datenbanksystemen zur Speicherung medizinischer Messdaten

Vorgelegt von: Wolfram Weidner

am: 08.08.2012

zum

Erlangen des akademischen Grades

Bachelor of Science

(B.Sc.)

Erstbetreuer: Prof. Dr.-Ing. Susanne Busse

Zweitbetreuer: Prof. Dr.-Ing Thomas Preuß

Danksagung

Auf diesem Wege möchte ich Frau Prof. Dr. Susanne Busse für Ihre Betreuung und die Unterstützung bei meiner Arbeit danken. Ebenfalls möchte ich Herrn Prof. Dr. Thomas Preuß dafür danken, dass er sich bereit erklärt hat, das Zweitgutachten zu übernehmen.

Gleichzeitig danke ich Herrn Dipl.-Informatiker (FH) Steffen Lange M.Sc. für die angeregten Diskussionen zu meiner Arbeit sowie Frau Dipl.Informatiker (FH) Katja Orlowski M.Sc. für ihre Unterstützung bei der Dokumentation der komplexen Messgeräte im Fachbereich.

Insbesondere danke ich meiner Familie für die ständige Hilfe, so beim Korrekturlesen oder beim Abnehmen täglicher Arbeiten.

Zusammenfassung

Diese Bachelorarbeit befasst sich mit der Evaluierung einer SQL- und NoSQL-Datenbanklösung zur Speicherung der anfallenden medizinischen Messdaten im Fachbereich Informatik und Medien der Fachhochschule Brandenburg.

Im Studiengang Medizininformatik werden im Laufe des Studiums viele Versuche mit den unterschiedlichsten medizinischen Messgeräten, wie zum Beispiel EKG, EEG, EMG, Shimmer u.a. durchgeführt. Bei diesen Versuchen müssen teilweise sehr große Mengen an Messdaten gespeichert werden. Im weiteren Verlauf des Studiums werden diese Daten analysiert und ausgewertet. Da es noch kein Konzept zur effizienten Haltung der Daten in einem Datenbanksystem gibt und diese zurzeit in vielen verschiedenen Formaten im Dateisystem eines Servers zwischengespeichert werden, wurde in dieser Arbeit eine möglichst effiziente Datenbanklösung gesucht, implementiert und getestet.

Abstract

This bachelor thesis deals with the evaluation of SQL and NoSQL database solutions for storing the storage of medical data arising in computer science and media of Applied Sciences in Brandenburg.

In the medical informatics degree program many experiments with various medical instruments, such as ECG, EEG, EMG, Shimmer etc. are carried out during the studies. In these experiments sometimes a lot of measurement data have to be stored. In the further course of studies these data are analyzed and evaluated. At the moment there is no concept for efficient storing the data in a database system that means, these data are cached in many different formats in the file system of a server. The purpose of this thesis was to find, implement and test the most efficient database solution.

Inhaltsverzeichnis

Abbildungsverzeichnis **1**

Tabellenverzeichnis **2**

1 Einleitung **3**
1.1 Motivation . . . 3
1.2 Aufgabenstellung . . . 4
1.3 Aufbau der Arbeit . . . 5

2 Analyse des Arbeitsablaufes zur Speicherung der medizinischen Messdaten **6**
2.1 Einleitung . . . 6
2.2 Verwendete medizinische Messgeräte . . . 6
2.2.1 EKG . . . 6
2.2.2 EEG . . . 7
2.2.3 EMG . . . 8
2.2.4 EOG Eye Tracking . . . 9
2.2.5 Bewegungsanalyse mit der Kinect . . . 9
2.2.6 Bewegungsanalyse mit dem Shimmer . . . 10
2.2.7 Mikroskop . . . 10
2.2.8 Blutdruckmessgerät . . . 11
2.2.9 Ultraschallgerät . . . 11
2.2.10 Bewegungsanalyse mit dem MoCap . . . 12
2.2.11 ProComp Infiniti . . . 12
2.3 Ist-Zustand und Anforderungen . . . 12
2.4 Zusammenfassung . . . 16

3 SQL-Datenbanksysteme **17**
3.1 Einleitung . . . 17
3.2 Grundlagen von SQL-Datenbanksystemen . . . 17
3.2.1 Historischer Hintergrund von SQL-Datenbanksystemen . . . 17
3.2.2 Datenmodellierung mit dem Entity-Relationship-Modell . . . 19
3.2.3 Entitätentypen . . . 20
3.2.4 Beziehungen . . . 21
3.2.5 Integritätsbedingungen in Datenbanken . . . 24
3.2.6 Trigger . . . 25
3.2.7 Relationale Datenmodelle . . . 25
3.2.8 Datenmanipulation mit SQL . . . 28
3.3 Vor- und Nachteile von SQL-Datenbanksystemen . . . 30

3.4 Zusammenfassung 31

4 NoSQL-Datenbanksysteme 32
4.1 Einleitung 32
4.2 Historischer Hintergrund von NoSQL-Datenbanksystemen 33
4.3 Grundlagen von NoSQL-Datenbanksystemen 33
4.3.1 Map/ Reduce 34
4.3.2 CAP-Theroem 35
4.3.3 Wide Column Stores 36
4.3.4 Document Stores 39
4.4 Vor- und Nachteile von NoSQL-Datenbanksystemen 41
4.5 Zusammenfassung 42

5 Evaluation und Implementierung einer Datenbanklösung 43
5.1 Anforderungsanalyse 43
5.2 Konzeptioneller Entwurf 47
5.3 Wahl des Datenbankmanagementsystems 49
5.4 Logischer Entwurf 51
5.4.1 Relationale-Datenbankmanagementsysteme 51
5.4.2 NoSQL-Datenbankmanagementsysteme 51
5.5 Evaluation eines geeigneten relationalen Datenbankmanagementsystems . . 53
5.6 Evaluation eines geeigneten NoSQL-Datenbankmanagementsystems 57
5.7 Benchmark-Test der evaluierten Datenbanken 59
5.8 Benchmark-Test einer Alternativlösung 61
5.9 Zusammenfassung 63

6 Zusammenfassung und Ausblick 67
6.1 Zusammenfassung 67
6.2 Ausblick 69

7 Anhang 70

Literaturverzeichnis 88

Glossar 92

Abbildungsverzeichnis

2.1 EEG Headbox 8
2.2 EEG Versuch 8
2.3 Mikroskop Arbeitsplatz 11
2.4 Anwendungsfalldiagramm zum Versuchsablauf 13

3.1 Historischer Abriss einiger Datenbanksysteme 18
3.2 Entität Baum mit Attributwerten 20
3.3 Einige Grundkonzepte eines ER-Modells 21
3.4 Binäre Beziehungstypen 22
3.5 Ternäre Beziehungstypen 22
3.6 Kardinalität am Beispiel Student und Hochschule 23
3.7 1:1-Beziehung 23
3.8 1:n-Beziehung 24
3.9 n:m-Beziehung 24
3.10 Begriffliche Konventionen im relationalen Modell 26

4.1 Datenfluss des Map/Reduce-Verfahrens 34
4.2 CAP-Theorem 35

5.1 Entwickeltes ER-Modell 47
5.2 Relationales Schema der Datenbank 51

Tabellenverzeichnis

2.1 Aufbau der Headerdatei 7
2.2 Übersicht über Versuchsdaten 14

3.1 Tabelle Untersuchung 26
3.2 Tabelle Patient 26
3.3 Ergebnis beim Selektieren von Zeilen 27
3.4 Ergebnis der Projektion von Spalten 27
3.5 Ergebnis der Join Operation 28

4.1 Reihenorientierte Speicherung wie bei relationalen Modellen 36
4.2 Spaltenorientierte Speicherung wie bei Wide Column Stores 36
4.3 Fremdschlüssel in HBase 38

5.1 Gestellte Anforderungen an das Datenbanksystem 46
5.2 Data-Dictionary zum ER-Modell 48
5.3 Einige Unterschiede einzelner Datenbanken 49
5.4 Datenmodell für die Datenbank HBase 52
5.5 Überarbeitete Tabelle für den Versuch 54
5.6 Überarbeitete Tabelle für den Nachfolger eines Versuchs 54
5.7 Übersicht über die benötigte Zeit beim Einfügen und Auslesen von Daten . 61
5.8 Überarbeitung des Data-Dictionary 62
5.9 Übersicht über die benötigte Zeit beim Einfügen und Auslesen von Daten in MySQL 63
5.10 Erfüllte Anforderungen der DBMS 65
5.11 Entscheidungsmatrix zur Wahl eines Datenbanksystems 66

1 Einleitung

1.1 Motivation

Datenbanken werden seit Jahrzehnten in den unterschiedlichsten Bereichen erfolgreich eingesetzt. Besonders in der heutigen Zeit werden Datenbanken für die zunehmend technologisch werdende Informationsgesellschaft immer wichtiger. Das Verwalten unseres Geldes bei der Bank, das Einkaufen im Supermarkt oder das World Wide Web, um nur einige Beispiele zu nennen, wären ohne Datenbanksysteme nur sehr schwer zu realisieren.

Im Supermarkt werden Kassen- und Warensysteme eingesetzt, um die Arbeit der Angestellten zu erleichtern und den Bedürfnissen der Kunden bestmöglich entgegen zu kommen. Durch das Warensystem muss zum Beispiel sichergestellt werden, dass immer ausreichend Ware zum Verkauf zur Verfügung steht. Die Datenbank, in der die Daten für das Warensystem gespeichert werden, muss also Funktionalitäten bieten, die es in diesem Fall ermöglichen, automatisiert neue Ware zu bestellen, wenn ein gewisser Warenbestand unterschritten wird. Dies wird bei vielen Datenbanksystemen mittels Trigger umgesetzt. Ein Trigger ist eine Art kleines Programm, welches Datensätze überwacht und nach dem Eintreten eines definierten Ereignisses, hier das Unterschreiten einer bestimmten Warenmenge, eine Reaktion, wie in diesem Fall das Nachbestellen dieser Ware, auslöst.

Im World Wide Web kommen Datenbanksysteme in den unterschiedlichsten Bereichen, wie zum Beispiel Versandhäuser wie Amazon, Ebay u.a. oder im einfachsten Fall bei einem Gästebuch einer Webseite, zum Einsatz.

Im Zeitalter des Web 2.0 müssen immer größere Datenmengen, sogar schon im Petabyte-Bereich, verarbeitet werden. Auch bei bekannten Firmen wie Facebook, Google oder Twitter fallen immer größere Datenmengen an. Da NoSQL-Datenbanksysteme zur Verarbeitung solcher Datenmengen konzipiert wurden, weden sie von diesen Firmen immer öfter eingesetzt. (vgl. [EFH+11] S. 2) Diese Systeme können, unter anderem durch eine einfacher umzusetzende vertikale Skalierung, besser mit so großen Datenmengen umgehen.

Ein weiterer sehr wichtiger Bereich für den Einsatz von Datenbanksystemen ist die Medizin.

So werden zum Beispiel in diversen Arztpraxen Datenbanken zur Verwaltung von Patientenakten eingesetzt. In Krankenhäusern kommen sogenannte Krankenhausinformationssysteme (folgend als KIS bezeichnet) zum Einsatz, die teilweise sogar verschiedene räumlich getrennte Krankenhäuser miteinander verbinden. Die Funktionsweise und Bedeutung eines solchen KIS soll anhand des folgenden Beispiels kurz erläutert werden.

Ein Patient geht mit einem Einweisungsschein seines behandelnden Arztes in die Aufnahme des Krankenhauses. Dort wird als erstes ein Datensatz für ihn im KIS angelegt. Anfangs stehen nur persönliche Daten und der Grund des Krankenhausbesuches in dem angelegten Datensatz. Nachdem der Patient auf der entsprechenden Station angekommen ist, werden zur genauen Diagnose und Abstimmung von erforderlichen Behandlungen erst einmal diverse Untersuchungen durchgeführt. Alle Ergebnisse dieser Untersuchungen werden ebenfalls im KIS gespeichert. Muss der Patient zu weiteren Untersuchungen auf andere Stationen, um zum Beispiel Röntgenbilder, ein EKG, EEG, u.a. anfertigen zu lassen, so trägt der untersuchende Arzt der jeweiligen Station die Ergebnisse ebenfalls ins KIS ein. Ist der Patient wieder zurück auf seiner Station, braucht der behandelnde Arzt nur ins System zu schauen und kann dort alle Untersuchungsergebnisse und eventuell schon getätigte Diagnosen von Kollegen einsehen. Dadurch wird viel Zeit gespart und die Ärzte können dem Patienten schneller und effektiver helfen.

Dieses Szenario zeigt, wie wichtig und hilfreich Datenbanksysteme im medizinischen Bereich sind.

An der Fachhochschule Brandenburg kommt im Studiengang Medizininformatik bereits eine Vielzahl von medizinischen Geräten zum Einsatz. Hier gibt es jedoch noch kein einheitliches System, mit welchem die anfallenden Messdaten gespeichert werden können. In dieser Arbeit wird ein Datenbanksystem gesucht, mit welchem es möglich ist, alle anfallenden Daten zu speichern.

1.2 Aufgabenstellung

Ziel dieser Arbeit ist die Evaluierung eines geeigneten SQL- oder NoSQL-Datenbanksystems zur Speicherung medizinischer Messdaten im Studiengang Medizininformatik. Dafür muss die Struktur der einzelnen Messstationen in Bezug auf die Speicherung dieser Messdaten analysiert werden. Nach der Analyse soll geprüft werden, welche Datenbanksysteme die gestellten Anforderungen am besten erfüllen. Nachdem die Anwendung implementiert und getestet worden ist, soll abschließend das geeignetste Datenbanksystem ausgewählt werden.

1.3 Aufbau der Arbeit

Bei der Analyse des Arbeitsablaufes zur Speicherung der medizinischen Messdaten (Kapitel 2) wird der Aufbau und die Struktur der Messstationen genau analysiert. Hier werden die einzelnen medizinischen Messgeräte, wie zum Beispiel das EKG, EEG, EMG u.a. vorgestellt. Ziel dieser Analyse des "Ist-Zustandes" ist, die Komplexität der durchzuführenden Versuche und die Vielfalt der zu speichernden Daten zu dokumentieren.

Im Kapitel SQL-Datenbanksysteme (Kapitel 3) wird kurz auf die benötigten Grundlagen von SQL-Datenbanksystemen mit ihren Vor- und Nachteilen eingegangen.

Das Kapitel NoSQL-Datenbanksysteme (Kapitel 4) befasst sich mit den Grundlagen und den Vor- und Nachteilen von NoSQL-Datenbanken.

In dem Kapitel Evaluation und Implementierung einer Datenbanklösung (Kapitel 5) werden eine Anforderungsanalyse durchgeführt, ein ER-Modell entworfen und Datenbankmanagementsysteme (DBMS) im SQL- und NoSQL-Bereich ausgewählt. Anschließend wird ein Datenmodell für die ausgewählten DBMS entwickelt und implementiert. Für den Benchmark-Test wird ein SQL- und NoSQL-System ausgewählt, welches allen gestellten Anforderungen gerecht wird. Diese Datenbanken werden mit Testdaten befüllt und es wird geprüft, wie viel Zeit Standardoperationen, wie das Einfügen oder Auslesen von Daten, in Anspruch nehmen. Abschließend wird in diesem Kapitel die am besten geeignete Datenbank ausgewählt.

Das Schlusskapitel (Kapitel 6) fasst die gesamte Arbeit zusammen und bietet einen Ausblick auf mögliche Verbesserungen.

2 Analyse des Arbeitsablaufes zur Speicherung der medizinischen Messdaten

2.1 Einleitung

Im Studiengang Medizininformatik des Fachbereiches Informatik und Medien der Fachhochschule Brandenburg wird mit vielen unterschiedlichen medizinischen Geräten gearbeitet. Bei diesen Arbeiten müssen die Messergebnisse zur weiteren Analyse zwischengespeichert werden. Die verschiedenen Datentypen werden zur Zeit lokal auf den einzelnen Messplätzen und zentral auf einem NFS-Share gespeichert.

2.2 Verwendete medizinische Messgeräte

Um den Studenten in diesem Studiengang ein besonders breites Wissen und ein gewisses Grundverständnis der Problematik bei medizinischer Hard- und Software zu vermitteln, wurde ein breit gefächertes medizinisches Equipment im Fachbereich angeschafft. Bei den einzelnen Messstationen sind teilweise sehr unterschiedliche Abläufe zur Erhebung, Speicherung und Auswertung der anfallenden Daten nötig. Dieses Kapitel beschäftigt sich mit den unterschiedlichen Messstationen und den Abläufen zur Speicherung und Auswertung an den entsprechenden Messplätzen. Grundvoraussetzung, um eine effektive Datenbanklösung für das Problem zu finden, ist eine genaue Kenntnis der Arbeitsabläufe im Umgang mit den medizinischen Messdaten.

2.2.1 EKG

Bei der Elektrokardiografie wird eine abgeleitete Kurve der auftretenden Spannungen jedes einzelnen Herzschlages aufgezeichnet. Diese schwache Spannung beträgt circa $\frac{1}{1000}$Volt.
(vgl. [PB10b])

Bei der Durchführung eines Elektrokardiogramms liegt der Proband ruhig auf einer Liege oder läuft auf einem Laufband. Das erste wird als Ruhe-EKG und das zweite als Belastungs-EKG oder Ergometrie bezeichnet. Bei beiden Untersuchungen wird jeweils eine Messelektrode auf die Arme, Beine und je nach durchgeführter Methode eine bestimmte Anzahl von Elektroden auf die Brustwand in einer bestimmten Reihenfolge aufgebracht. (vgl. [Boe03]) In den Laboren des Fachbereichs Informatik und Medien wird die Untersuchung mit dem EKG nach Einthoven, Wilson und Goldberger, das heißt mit vier, sechs und zehn Ableitungen (Elektroden) durchgeführt. Um die vom Herzen ausgehende Spannung aufzeichnen zu können, wird zwischen den Messelektroden und der Haut ein Gel, welches elektrisch leitfähig ist, aufgebracht.

Die Untersuchung wird über einen bestimmten Zeitraum durchgeführt, bei dem die sogenannte EKG-Kurve aufgezeichnet wird. Die so entstandenen Rohdaten werden für eine spätere Auswertung lokal auf dem Rechner des EKG-Gerätes, im dat-, xml- oder wmv-Format, gespeichert. Bei der Datei im dat-Format handelt es sich um eine Binärdatei mit den eigentlichen Messdaten des EKG. Um diese Datei später auswerten zu können, wird über eine Software hierzu eine Headerdatei mit Metadaten erzeugt. Der Aufbau dieser Headerdatei ist in Tabelle 2.1 zu sehen. Bei der Datei im xml- und wmv-Format wird diese Headerdatei nicht benötigt.

Zusätzlich wird ein Netzlaufwerk auf einem Server eingebunden, auf welchem diese Dateien ebenfalls zwischengespeichert werden. Dies ist nötig, um gleichzeitig von anderen Arbeitsplätzen die entstandenen Daten auswerten zu können.

Zeilen Nummer	Zeile 1	Zeile 2 - n+1	weitere Zeilen
Bezeichnung	Dateiname	Dateiname	Kommentar
	Anzahl der Signale (n)	Anzahl der Bits je Wert	Diese Kommentarzeilen
	Abtastrate	Umrechnungsfaktor	beginnen mit einer “#”
	Anzahl der Signalpunkte	ADC Resolution	
	Kommentar	ADC Zero	
		Initial value	
		Skalierungsfaktor	
		Offset	
		Kommentar	

Tabelle 2.1: Aufbau der Headerdatei

2.2.2 EEG

Bei der Elektroenzephalografie werden die elektrischen Gehirnströme gemessen. Zur Messung dieser Gehirnströme werden Elektroden auf die Kopfhaut aufgebracht. Die Spannungsunter-

schiede zwischen jeweils zwei Elektroden werden durch die Headbox, die in Abbildung 2.1 zu sehen ist, verstärkt und als Hirnstromwelle aufgezeichnet. Bei der Hirnstromwelle ist die Frequenz, die in Hertz gemessen wird sowie die Wellenhöhe ein wichtiges Merkmal, welches zu einem späteren Zeitpunkt ausgewertet werden muss. Bei den Wellenformen wird zwischen Alphawellen, Betawellen, Thetawellen und Deltawellen unterschieden. Diese Wellenformen unterscheiden sich in ihrem Frequenzband. (vgl. [Sch10a])

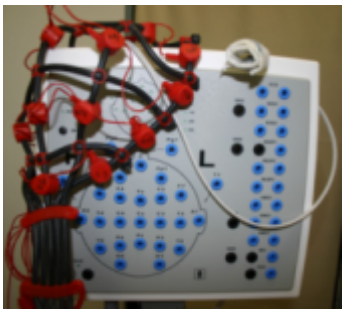

Abbildung 2.1: EEG Headbox

Eine Untersuchung, wie in Abbildung 2.2 zu sehen ist, kann zwischen 20 und 30 Minuten andauern. Die entstandenen Rohdaten werden wie bei der EKG-Untersuchung lokal auf dem EEG-Gerät und auf dem Server im Netz zur weiteren Auswertung gespeichert. Hier handelt es sich ebenfalls um ein dat-Format mit einer dazu generierten Headerdatei. Die Struktur der dat-Formate weicht bei den verschiedenen Untersuchungen voneinander ab. Der Aufbau der Headerdatei ist immer identisch zur Tabelle 2.1.

Abbildung 2.2: EEG Versuch

2.2.3 EMG

Bei der Elektromyografie handelt es sich um ein Verfahren, bei welchem die elektrischen Aktivitäten von Muskeln gemessen werden. Es gibt zwei Möglichkeiten, diese Muskelaktivitäten zu messen. Ein Verfahren ist das Nadel-EMG, bei der eine dünne Nadel-Elektrode direkt in den Muskel gestochen wird. Diese Methode wird jedoch in den Laboren des Fachbereiches

nicht durchgeführt. Das hier zur Anwendung kommende Verfahren ist das Oberflächen-EMG, bei dem aufklebbare Oberflächenelektroden zum Einsatz kommen. Bei der Untersuchung liegt der Proband wieder entspannt auf einer Liege und die Elektroden werden auf die Haut über dem zu messenden Muskel befestigt. (vgl. [Sch10b]) Der Proband muss jetzt diesen Muskel zuerst leicht und danach voll anspannen. Die elektrischen Aktivitäten des Muskels werden vom EMG-Gerät aufgezeichnet und lokal auf diesem abgespeichert. Auch diese Rohdaten werden zusätzlich auf dem vorgesehenen Server zur Auswertung abgelegt. Bei dem EMG werden diese Dateien im csv- und m-Format gespeichert. Bei dem Dateityp mit der Endung "m" handelt es sich um eine Datei, die mit der Hochsprache und Umgebung Matlab[1] erstellt wurde. Ein Beispiel zu einer solchen Datei ist im Anhang im Listing 7.17 zu sehen.

2.2.4 EOG Eye Tracking

Beim sogenannten Eye Tracking werden fokussierte Punkte (Fixationen), schnelle Augenbewegungen (Sakkaden) und Blickbewegungen (Regressionen) einer Person aufgezeichnet.

Dieses Verfahren wird beispielsweise in den Neurowissenschaften angewendet. (vgl. [Wik12a])

Auch bekannte Firmen wie Facebook oder Google forschen intensiv mit diesem Verfahren. So führte z.B. Google zwischen den Jahren 2005 und 2008 eine Studie mit 30 Teilnehmern durch. Dabei sollte das Suchverhältnis dieser Probanden bei der Verwendung der Bildsuchmaschine von Google getestet werden. Es wurde festgestellt, dass das Ergebnis eines Suchbegriffs in weniger als einer Sekunde abgescannt und unterbewusst eine Entscheidung getroffen wird. Dieses Verhalten ist zwar keine neue Erkenntnis, doch Google geht es bei diesen Untersuchungen darum, die Zeitspanne der Entscheidung näher zu analysieren. Das Ziel ist eine Art Interface-Optimierung der Google-Suchergebnisse. (vgl. [Neu])

Die Messergebnisse werden beim Eye tracking in einem eeg- oder ee_- Format lokal und auf dem Netzwerkserver abgespeichert.

2.2.5 Bewegungsanalyse mit der Kinect

Die Kinect[2] ist ein Produkt, welches unter anderem von der Firma Microsoft entwickelt wurde. Sie wurde ursprünglich zur controllerlosen Bedienung der XBox 360 entworfen. Bei der Kinect kommen ein CMOS-Sensor, welcher mit einer Abtastrate von 30 Hz arbeitet, sowie eine RGB-Kamera und vier Mikrofone zum Einsatz. Das Gerät sendet ein Muster von Infrarotpunkten

[1] http://www.mathworks.de/products/matlab/
[2] http://www.xbox.com/de-de/kinect

aus, welche einen bestimmten Bereich ausleuchten. Der CMOS-Sensor speichert die Daten der Infrarotpunkte und gleicht sie mit einem vorgegebenen Muster ab. Dieses intern gespeicherte Muster ist, in unserem Fall, ein Skelett-Modell mit 20 Gelenkpunkten. Bei diesem Vergleich kann mittels der Stereoskopie aus der Verzerrung der Muster eine Tiefeninformation gewonnen werden. (vgl. [TP11, Wik12b]) Mittels eines Matching-Algorithmus wird geprüft, ob es sich bei diesen Tiefeninformationen um einen Menschen handelt. Ist das Matching erfolgreich, so werden diese 20 Gelenkpunkte verfolgt. Auf diese Weise ist es möglich, die Bewegungsabläufe der 20 Gelenkpunkte genau aufzuzeichnen und in Form eines Tiefenbildes abzuspeichern. (vgl. [DIFKO12])

Bei diesem Verfahren werden die Dateien im csv-, dat- mit dazugehöriger Headerdatei, avi-, rgb-, rgba- und depth-Format gespeichert.

2.2.6 Bewegungsanalyse mit dem Shimmer

Bei dem Shimmer der Firma Shimmer Research[3] handelt es sich um einen mobilen Sensor, den es in verschiedenen Ausprägungen gibt. Die Sensoren bestehen aus einem Grundmodul, auf dem sich ein 3-Achsen Beschleunigungssensor befindet und unterschiedlichen anderen Sensoren, wie zum Beispiel einem 3-Achsen Gyroskop, einem 3-Achsen Magnetometer, et cetera. Bei einem Versuch mit einem Shimmer Sensor können Beschleunigung, Winkelgeschwindigkeit sowie biophysische Größen wie EKG und EMG gemessen werden. Der Shimmer wird, wie die Kinect, benutzt, um Bewegungsdaten aufzuzeichnen und auszuwerten. Die aufgezeichneten Rohdaten werden über Bluetooth an einen Rechner übermittelt und dort im dat-Format gespeichert. Auch hier wird wieder ein Header zu dieser Binärdatei erzeugt. (vgl. [DIFKO12])

2.2.7 Mikroskop

Bei einem Mikroskop handelt es sich um ein Linsensystem, das durch seinen Aufbau aus Objektiv und Okular (zwei verschiedenen Linsen) ein Objekt stark vergrößert darstellen kann. (vgl. [Sen03]) In das Mikroskop aus Abbildung 2.3 ist eine hochauflösende Kamera integriert, welche diese Bilder auf dem Rechner anzeigt. Dort werden diese dann lokal und auf dem Netzwerkserver als Bild im tif-Format gespeichert.

[3] http://www.shimmer-research.com/

Abbildung 2.3: Mikroskop Arbeitsplatz

2.2.8 Blutdruckmessgerät

Der Druck in den Arterien wird als Blutdruck bezeichnet. Bei der Blutdruckmessung wird das Herz- und Kreislaufsystem des Körpers überprüft. Das Blutdruckmessgerät besteht aus einem Stethoskop und einer Manschette, welche mit einem Manometer versehen ist. Die Manschette wird am Oberarm angebracht und aufgepumpt, bis die Arterie kein Blut mehr durchlässt. Das Stethoskop wird in der Armbeuge auf die Arterie aufgesetzt. Über das Manometer wird jetzt begonnen, langsam die Luft aus der Manschette abzulassen. Die durch das Stethoskop wahrgenommenen Geräusche bilden den oberen- oder auch systolischen Wert und den unteren- oder auch diastolischen Wert. (vgl. [PB10a])
Diese Daten werden im csv-Format abgespeichert.

2.2.9 Ultraschallgerät

Bei der Ultraschalluntersuchung auch Sonografie genannt, handelt es sich um ein bildgebendes Verfahren, bei dem verschiedene Körperregionen mit Hilfe von Ultraschallwellen dargestellt werden. Das Ultraschallgerät besteht aus einem Schallkopf, der als Sender und Empfänger für die Ultraschallwellen dient und einem kleinem Bildschirm. Bei der Ultraschalluntersuchung wird ein Gel, welches die elektrische Leitfähigkeit der Haut erhöht, auf den zu untersuchenden Bereich aufgebracht. Der Schallkopf des Ultraschallgerätes wird auf die entsprechende Körperstelle gepresst und sendet die Ultraschallwellen aus. Je nach Art des Gewebes werden diese Wellen entweder absorbiert oder reflektiert. Die reflektierten Wellen werden in elektrische Impulse umgewandelt, verstärkt und in einem zweidimensionalen Bild auf dem Bildschirm dargestellt. (vgl. [MK10])
Dieses aufgenommene Bildmaterial wird ebenfalls auf dem Ultraschallgerät und auf dem Server zwischengespeichert.

2.2.10 Bewegungsanalyse mit dem MoCap

Beim Motion Capture handelt es sich um eine Bewegungserfassung. Hierzu muss ein Kamerasystem, bestehend aus acht Kameras, aufgebaut werden. Zum System gehört noch ein Anzug mit integrierten Sensoren und ein Rechner, der die Daten verarbeitet. Die Testperson muss den Anzug anziehen und auf einer vorgegebenen Linie entlanglaufen. Das Kamerasystem nimmt dieses auf und übermittelt die Rohdaten an das Rechnersystem. So ist es möglich, das Gangmuster aufzuzeichnen und später auszuwerten.
Diese Daten werden im bvh-, c3d-, wmv- und fbx-Format gespeichert.

2.2.11 ProComp Infiniti

Mit der ProComp lassen sich mit verschiedenen Sensoren unterschiedliche Messungen durchführen.

So wird mit dem EKG-Flex/Pro-Sensor eine direkte elektroinduzierte Herzratenmessung durchgeführt. Der HR/BVP-Sensor ist ein Blutvolumen-Puls-Sensor, der die Herzfrequenz, Amplitude, Wellenform und die Herzratenvariabilität misst. Mit dem MyoScan-Pro EMG-Sensor lässt sich eine Oberflächen-Elektromyographie (Messung der Muskelaktivität) durchführen. Der SC-Flex/Pro ist ein Sensor, der zur Messung der Leitfähigkeit der Haut verwendet wird. Eine Messung der Oberflächentemperatur der Haut kann mit dem Temp-Flex/Pro-Sensor durchgeführt werden. Abschließend gibt es außerdem den Resp-Flex/Pro-Sensor, mit dem eine thorakale oder abdominale Messung der Atmung (Rate, Wellenform und Amplitude) durchgeführt werden kann.

Diese Sensoren werden an die ProComp Infiniti angeschlossen und übertragen die aufgenommenen Daten an einen Rechner, auf welchem sie im dat- oder wmv-Format gespeichert werden.

2.3 Ist-Zustand und Anforderungen

Wie im vorherigen Abschnitt festgestellt wurde, werden in diesem Bereich eine Vielzahl von Versuchen durchgeführt.

Im Anwendungsfalldiagramm in Abbildung 2.4 ist zu sehen, dass an einem Versuch und dessen Auswertung drei Personengruppen beteiligt sind.

Bei der Gruppe der Probanden handelt es sich in den meisten Fällen um Studierende, die aber gleichzeitig in der Rolle des Forschungspersonals Versuche auswerten. Je nach

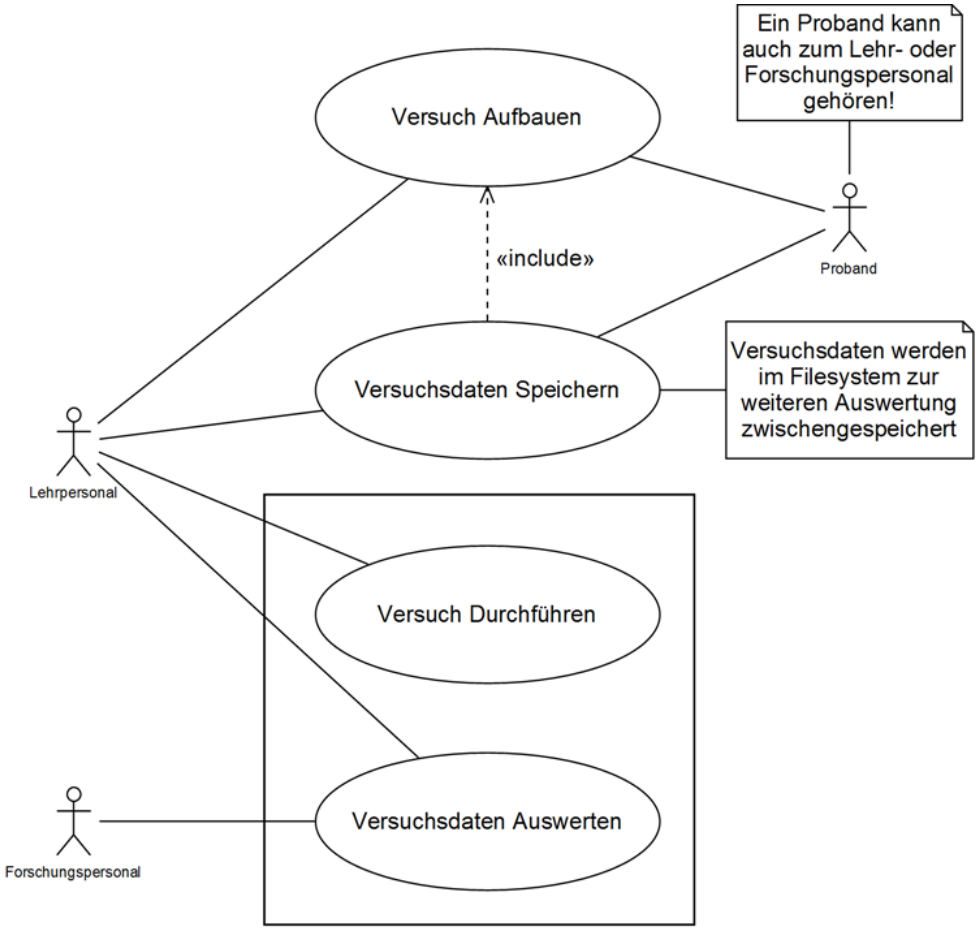

Abbildung 2.4: Anwendungsfalldiagramm zum Versuchsablauf

Versuchsart müssen zu dem jeweiligen Probanden unterschiedliche Daten gespeichert werden. So ist bei den Geh- und Hebeversuchen mit der Kinect, ProComp oder Schimmer die Bein- und Armlänge für die spätere Auswertung des Versuchs enorm wichtig. Für das EOG Eye Tracking ist das Alter und Geschlecht ein wichtiger Faktor. Diese Daten werden zur Zeit in einer Excel Tabelle gespeichert.

Die Gruppe Lehrpersonal besteht aus Professoren, wissenschaftlichen Mitarbeitern und studentischen Hilfskräften, die hauptsächlich im Bereich der Medizininformatik tätig sind. Diese Gruppe plant die Versuche, führt sie durch, wertet sie aus und speichert die Versuchsdaten zur weiteren Forschung auf dem Netzwerkserver ab. Diese Versuchsergebnisse mit den unterschiedlichen Dateiformaten:

dat-, csv-, xml-, wmv-, avi-, m-, eeg-, ee_-, tif-, bvh-, c3d-, depth-, rgb-, rgba- und fbx-Format

müssen in der Datenbank gespeichert werden. In Tabelle 2.2 ist eine Übersicht über einige Dateiformate und vor allem Dateigrößen von Versuchsergebnissen zu sehen.

Versuchsart	Dateityp	Erläuterung	Dateigröße
EKG	dat	Binärdatei	~100 MB
EMG	csv	Textdatei mit einfach strukturierten Daten	~180 MB
Kinect	avi	Audio/Video	~1,2 GB
	depth	Bilddatei mit Tiefeninformationen (depth Sensor)	~3,5 GB
	rgb	Bilddatei (rgb Sensor)	~13 GB
	rgba	Bilddatei mit Alpha Kanal	~1,5 GB
ProComp	dat	Binärdatei	~600 MB
Shimmer	dat	Binärdatei	~30 MB

Tabelle 2.2: Übersicht über Versuchsdaten

Bei diesen Beispielen handelt es sich um Versuchsdaten, die in diesem Format und in dieser Größe momentan auf dem Netzwerkserver vorhanden sind. Die maximale Größe eines Versuchsergebnisses darf 31 Gigabyte nicht überschreiten, deshalb wurden nur die größten Datensätze betrachtet. Diese Größenbeschränkung der Datenbank wird im Kapitel 5 näher erklärt. Da die Größe der Versuchsdaten von der Versuchslänge oder der Auflösung des Bildmaterials abhängt, ist es möglich, dass Versuche durchgeführt werden, bei denen noch größere Datenmengen entstehen.

Um eine 14 Gigabyte große Datei mit Versuchsdaten auf den Netzwerkserver zu kopieren, werden bei einer Netzwerkbandbreite von einem Gigabit pro Sekunde und einem 64-Bit Windows 7 Clientrechner ca. fünf Minuten benötigt. Dies sollte bei der Datenbanklösung, wenn möglich, nicht überschritten werden.

Zur Prognose des allgemeinen Datenaufkommens, z.B. pro Woche oder pro Monat, lässt sich schwer etwas sagen. Bedingt durch die unterschiedlichen Kurse (Module) und die unterschiedliche Anzahl von Studenten in den verschiedenen Semestern kann hier keine genaue Angabe gemacht werden. Einerseits gibt es Semester, in denen lediglich ein paar Megabyte an Daten anfallen, andererseits gibt es aber auch Semester, in denen mehrere Gigabyte an Daten produziert werden. Das Einzige, was zum Datenaufkommen gesagt werden kann, ist die Tatsache, dass die Versuche seit zwei Jahren durchgeführt werden und in dieser Zeit ca. 314 Gigabyte an Daten gespeichert wurden.

Zurzeit werden die Rohdaten durch das Lehrpersonal auf dem Netzwerkserver gespeichert. Dadurch ist es möglich, diese Daten später über das Netzwerk von beliebigen Rechnern auszuwerten. Die gespeicherten Daten sollen für einen Zeitraum von mindestens 10 Jahren auf dem Server abrufbar sein und können anschließend komprimiert in einem Archiv aufbewahrt werden.

Da auch Auswertungen an den medizinischen Messgeräten durchgeführt werden, ist es erforderlich, die originalen Dateiformate beizubehalten.

Die ausgewerteten Daten werden in einer Ordnerstruktur auf dem Netzwerkserver abgelegt. Um einen Verweis mit dem Rohdatensatz herstellen zu können, werden diese in Unterordnern abgespeichert. Dieses Verfahren ist unübersichtlich und bei vielen Datensätzen nicht zu empfehlen. Die originalen Rohdaten dürfen nicht verändert oder gelöscht werden, was bisher technisch nicht sichergestellt ist.

Um diese Daten besser auswerten zu können, wurde von Herrn Prof. Dr. H. Loose und Herrn M. Jahr ein Tool in der Sprache MatLab geschrieben. Der Konverter erzeugt aus den Versuchsdaten eine Headerdatei, wie in Tabelle 2.1 auf Seite 7 zu sehen, in der die Metainformationen zu den Versuchsdaten gespeichert werden. Dieser Konverter erzeugt zurzeit aus den Binär- und den EEG-Dateien des Eye Tracking Verfahrens eine Headerdatei, soll aber zu einem späteren Zeitpunkt noch erweitert werden, um so auch andere Datenformate zu verarbeiten.

Der Konverter soll später die Funktion einer Schnittstelle zur Datenbank einnehmen. D.h. er soll die Versuchsdaten und die dazugehörigen Metainformationen direkt in der Datenbank abspeichern oder die Rohdatensätze aus der Datenbank auslesen, um sie dann z.B. in MatLab oder auf den entsprechenden medizinischen Messgeräten auswerten zu können.

Bei der Gruppe des Forschungspersonals handelt es sich um Professoren, wissenschaftliche Mitarbeiter, studentische Hilfskräfte sowie Studenten. Dieser Personenkreis wertet die Versuchsdaten für die unterschiedlichsten Aufgaben aus. Bei den Hebeversuchen werden z.B. die Bewegungsmuster in Verbindung mit der Armlänge der Probanden analysiert. Bei dieser Analyse werden die Daten z.B. mit Daten von anderen Versuchen verglichen. Da sich die Hebebewegung von einem Probanden mit kurzen Armen stark von einem mit langen unterscheidet, müssen diese beim Vergleichen der Daten berücksichtigt werden. Bei den Gehversuchen werden die aufgenommenen Bewegungen der Gelenke in Verbindung mit der Beinlänge der Probanden ausgewertet. Die Auswertungen werden zu Lehrzwecken in verschiedenen Kursen und Modulen von Studenten durchgeführt. Diese Datensätze werden zu Forschungszwecken, für Promotionen und Master- oder Bachelorarbeiten verwendet.

Wichtig für die Evaluierung eines Datenbanksystems sind nur die Anwendungsfälle: Versuchsdaten speichern und Versuchsdaten auswerten, da sich nur diese Anwendungsfälle mit der Speicherung von Daten beschäftigen.

Abschließend bleibt zu sagen, dass die Kosten für die Umsetzung der Problemlösung so gering wie möglich sein müssen. Da dies kein offizielles Projekt ist, stehen keine Mittel zur Verfügung, um eventuelle Hard- oder Software zu beschaffen.

2.4 Zusammenfassung

Zur Analyse des momentanen Zustandes in den Laboren des Studienganges Medizininformatik lässt sich sagen, dass es sich um ein sehr breites Spektrum von eingesetzten medizinischen Messgeräten handelt. Es sind teilweise sehr unterschiedliche Messgeräte, bei denen nach der Durchführung eines Versuches unterschiedliche Ergebnisse zur weiteren Bewertung abgespeichert werden müssen. So kann es bei dem Mikroskop zu sehr großen Bilddateien im tif-Format kommen, während es bei dem EKG, EEG, EMG und Blutdruckmessgerät zu verhältnismäßig kleinen Dateien in einem dat- oder csv-Format kommt. Da die Versuchsdaten zu einem späteren Zeitpunkt immer wieder analysiert und ausgewertet werden müssen, muss zwingend zu den eigentlichen Daten ein Header mit Metainformationen zu dem jeweiligen Versuch gespeichert werden.

Außerdem ist zu beachten, dass die eigentlichen Rohdaten nicht verändert werden dürfen, aber die Ergebnisse der durchgeführten Analysen, im Zusammenhang mit den eigentlichen Rohdaten abgespeichert werden müssen. Dies muss beim Entwurf der Datenbank berücksichtigt werden. Als Schnittstelle zwischen der Datenbank und den eigentlichen Messgeräten ist die vorhandene Software, die zur Erstellung der Header-Dateien entwickelt wurde, vorgesehen. Sie übernimmt das Ablegen und das Auffinden der Daten im Datenbanksystem.

3 SQL-Datenbanksysteme

3.1 Einleitung

Um ein Grundverständnis für SQL-Datenbanksysteme zu schaffen, wird in diesem Kapitel auf die Grundlagen, die für diese Arbeit nötig sind, mit ihren allgemeinen Vor- und Nachteilen eingegangen. Durch die hohe Komplexität von Datenbanksystemen können diese nur kurz angesprochen werden. Für eine Vertiefung in diesem Bereich bieten sich z.B. die Fachbücher [EN02] oder [SsH08] an, welche sich ausführlich mit diesem Themengebiet auseinandersetzen.

3.2 Grundlagen von SQL-Datenbanksystemen

Dieser Abschnitt soll mit einem Zitat einer möglichen Definition einer Datenbank (auch Datenbanksystem) aus dem Informatikduden begonnen werden:

> "System zur Beschreibung, Speicherung und Wiedergewinnung von umfangreichen Datenmengen, die von mehreren Anwendungsprogrammen benutzt werden. Es besteht aus der Datenbasis, in der die Daten abgelegt werden, und den Verwaltungsprogrammen (Datenbanksoftware, Datenbankmanagementsystem), die die Daten entsprechend den vorgegebenen Beschreibungen abspeichern, auffinden oder weitere Operationen mit den Daten durchführen." ([CS01] S. 154)

3.2.1 Historischer Hintergrund von SQL-Datenbanksystemen

Nachdem der Begriff Datenbanksystem definiert wurde, wird kurz auf die historischen Hintergründe von SQL-Datenbanksystemen eingegangen.

In Abbildung 3.1 nach dem Buch [SsH08], ist eine kurze zeitliche Übersicht über verschiedenen Datenmodelle zu sehen. Auf die für diese Arbeit relevanten Modelle wird im Folgenden kurz eingegangen.

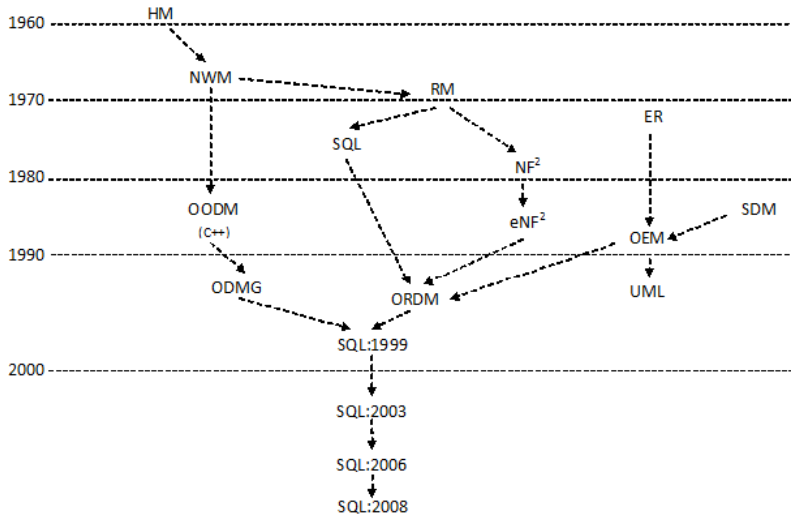

Abbildung 3.1: Historischer Abriss einiger Datenbanksysteme

In den 1960er Jahren entstand das erste Datenbanksystem auf Basis des **hierarchischen Datenmodells**. Der Begriff des Datenmodells wurde erstmalig bei den CODASYL-Konferenzen[4] eingeführt. Von dieser Konferenz wurde 1969 das **Netzwerkdatenmodell** veröffentlicht. Bei diesem Netzwerkdatenmodell werden die Daten durch verkettete Datensätze (Records), die ein Netz bilden, strukturiert. Als Beispiel hierfür ist das Flugreservierungssystem SABRE oder das System IDS von Charles Bachmann[5] zu nennen.

Als Grundlage für das **relationale Datenmodell** gelten die Arbeiten von Edgar F. Codd von 1970 und 1972. Für diese Arbeiten wurde er 1981 mit dem Turing-Award[6] ausgezeichnet. (vgl. [Adm12]) Bei diesem Datenmodell werden die Daten in Tabellen strukturiert. Tabellen sind sogenannte Tuppelmengen[7]. Die Beziehungen zwischen den einzelnen Tabellen werden durch Wertegleichheit ausgedrückt. Aufgrund der starken Verbreitung von relationalen Datenbanksystemen sowie deren gute Lösungsmöglichkeiten für den Anwendungsfall geht diese Arbeit im Bereich der SQL-Datenbanksysteme nur auf diese ein.

Das theoretische Relationenmodell wird durch das **SQL-Datenmodell** um einige weitere Aspekte wie Tabellen als Multimengen erweitert. Es werden hier eine Reihe von unterschiedlichen Wertebereichen, wie zum Beispiel lange Felder, Strings, Datumswerte und Nullwerte

[4] Conference on Data Systems Languages: Die erste Konferenz fand 1959 statt
[5] Charles Bachmann ist der Begründer der Netzwerkdatenbanken
[6] Der Turing-Award wurde nach Alan Turing benannt und wird jährlich von der Association for Computing Machinery (ACM) an Personen verliehen, die sich besonders um die Entwicklung der Informatik verdient gemacht haben. Er gilt als höchste Auszeichnung in der Informatik.
[7] In der Mathematik wird ein Tupel als endliche Liste definiert. In der Informatik steht der Begriff Tupel für eine geordnete Wertesammlung. In der relationalen Algebra, die hier gemeint ist, wird es als Synonym für einen Datensatz, dessen Werte Attribute genannt werden, verwendet. Das Tupel oder genauer ausgedrückt das n-Tupel bezeichnet eine Sammlung mit einer beliebigen Anzahl (n) von Attributen. (vgl. [Wik12d, Wik12c])

eingeführt. Durch die Anfragesprache wird jetzt gleichzeitig eine Gruppierung und Sortierung, die im Relationenmodell nicht vorgesehen ist, unterstützt.

Das abstrakte **Entity-Relationship-Modell** ist auf eine Arbeit von P. P. Chen[8] aus dem Jahr 1976 zurückzuführen. Im ER-Modell werden mittels Datensätze (Entitys), Beziehungen zwischen den Datensätzen (Relationships) und Attributen die Datenbestände modelliert. Dieses Modell wird oft beim Entwurf einer Datenbank eingesetzt und deshalb in den nächsten beiden Unterabschnitten genauer erläutert.
(vgl. [SsH08] S. 53 - 56, [FWBRB07] S.33 - 35 und **[Bus10]** S. 6 - 7)

Auf andere Datenmodelle wie z.B. semantische-, objektorientierte- oder objektrelationale Datenmodelle wird in dieser Arbeit aufgrund der fehlenden Relevanz nicht näher eingegangen. Der interessierte Leser sei auf Fachliteratur, wie z.B. das Buch [EN02] oder [SsH08], verwiesen.

3.2.2 Datenmodellierung mit dem Entity-Relationship-Modell

Der Datenbankentwurfsprozess ist ein sehr komplexes Themengebiet, welches in vier Phasen eingeteilt ist. Da es für den Entwurf und die Implementierung eines Datenbanksystems sehr wichtig ist, diese Phasen zu kennen und anwenden zu können, werden diese im folgenden Unterabschnitt näher erläutert.

Die erste Phase ist die **Erfassung und Analyse der Anforderungen**. In diesem Schritt müssen die Anforderungen der künftigen Datenbanknutzer an die Datenbank herausgefunden werden. Ein sehr wichtiges und entscheidendes Kriterium ist der Umfang der Daten, die im späteren Datenbanksystem gespeichert werden sollen. Das Ergebnis dieser Phase ist eine Art Katalog, ähnlich dem Pflichtenheft bei der Softwareentwicklung, in dem alle wichtigen und umzusetzenden Anforderungen, insbesondere die **funktionalen**, aufgelistet sind. Funktionale Anforderungen sind Operationen von Benutzern (oder Transaktionen), die auf der Datenbank ausgeführt werden sollen.

In der nächsten Phase wird mit Hilfe eines konzeptionellen Datenmodells ein **konzeptionelles Schema** für die Datenbank erstellt. Dieser Schritt kann auch als **konzeptioneller Entwurf** bezeichnet werden. Diese Phase ist eine kompakte Beschreibung der Anforderungen und beinhaltet genaue Beschreibungen der Entitätentypen, Beziehungen und Einschränkungen. Das konzeptionelle Schema kann als Referenz herangezogen werden, um sicherzustellen, dass alle Anforderungen erfüllt werden.

[8] Peter Chen, chinesisch: Chen Pin-Shan (Chen, Peter Pin-Shan) ist ein taiwanesischer Informatiker der 1976 die ER-Modellierung für die Datenanalyse entwickelte

Als dritte Phase kann die eigentliche Implementierung der Datenbank gezählt werden. In diesem Schritt, der auch als **logischer Entwurf** bezeichnet wird, wird das konzeptionelle Schema vom konzeptionellen Datenmodell in ein Implementierungsdatenmodell transformiert. Das Ergebnis dieser Phase ist ein Datenbankschema.

Die letzte Phase ist der **physische Entwurf**, in der Dateiorganisationen, Zugriffspfade und Speicherstrukturen festgelegt werden. (vgl. [EN02] S. 63 - 66)

3.2.3 Entitätentypen

Ein Entity-Relationship-Modell beschreibt Daten als Entitäten, Beziehungen und Attribute. Das Basisobjekt eines ER-Modells ist die Entität. Allgemein wird sie in der Fachliteratur mit E_1, E_2,... bezeichnet.

Die Entität ist ein Objekt, das entweder physisch existiert, wie zum Beispiel ein Baum oder eine Person, oder konzeptionell existiert, wie beispielsweise eine Abteilung oder ein Projekt. Eine Entität hat Attribute, allgemein als A bezeichnet, die sie näher beschreibt. Bei der Entität Baum könnten diese Eigenschaften der Typ, die Höhe, der Umfang und das Alter sein. Jedes Attribut besitzt, wie in Abbildung 3.2 zu sehen, einen zugehörigen Wert. Der Wertebereich kann optional zusätzlich mit einem D in der Form $A : D$ angegeben werden. (vgl. [SsH08] S. 60 - 64)

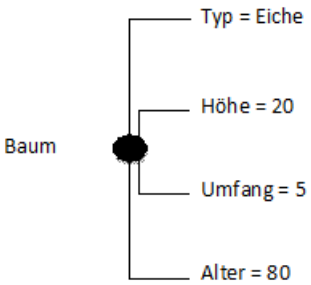

Abbildung 3.2: Entität Baum mit Attributwerten

Bei einem ER-Modell gibt es einfache (atomare) oder zusammengesetzte, einwertige oder mehrwertige und gespeicherte oder abgeleitete Attributtypen. Zusammengesetzte Attribute lassen sich weiter zerlegen. Ein Beispiel hierfür ist das Attribut Name. Es lässt sich somit in die Attribute Vorname und Nachname, die jetzt als einfache oder atomare Attribute bezeichnet werden, zerlegen. Bei den mehrwertigen Attributen kann, wie der Name schon sagt, das Attribut mehrere unterschiedliche Werte annehmen. Ein Beispiel ist die Entität

Person mit dem Attribut Anzahl der Kinder. Dieses Attribut kann mehrere Werte, wie zum Beispiel null, eins, zwei oder drei einnehmen. Bei den gespeicherten Attributen handelt es sich um gespeicherte Werte in der Datenbank wie das Geburtsdatum einer Person. Abgeleitete Attribute sind nicht direkt in der Datenbank gespeichert, lassen sich aber durch Beziehungen von anderen gespeicherten Attributen ableiten. So kann man aus dem gespeicherten Attribut aktuelles Datum und Geburtsdatum das Alter schlussfolgern. Das Alter ist also das abgeleitete Attribut.

Weiterhin gibt es in ER-Modellen sogenannte Schlüsselattribute. Diese werden im Allgemeinen mit S abgekürzt. Der Wert eines solchen Attributs ist eindeutig und unterscheidet sich in jeder einzelnen Entität der Sammlung. Anhand des Schlüsselattributs lässt sich also jede Entität eindeutig bestimmen. (vgl. [EN02] S. 68 - 70)

Schlüsselattribute werden im ER-Modell oder in der Textform entweder unterstrichen oder mit einem schwarzen Oval dargestellt. (vlg. [SsH08] S. 65)

In Abbildung 3.3 ist eine Übersicht über die eben beschriebenen Konzepte von ER-Modellen zu sehen. Auf die Relationship-Typen oder auch Beziehungen genannt und die Kardinalitäten wird in den nächsten beiden Unterabschnitten eingegangen.

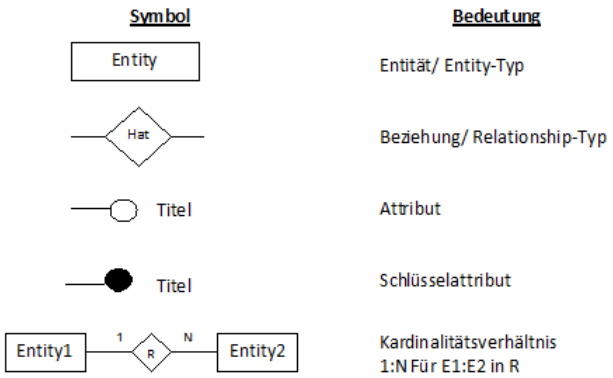

Abbildung 3.3: Einige Grundkonzepte eines ER-Modells

3.2.4 Beziehungen

Eigenschaften von Beziehungen Beziehungen werden oft mit einem *R* bezeichnet und lassen sich im Allgemeinen durch zwei wesentliche Eigenschaften beschreiben:

- wie viele Entity-Typen mit einem Beziehungstyp verbunden sind, wird durch die **Stelligkeit** angegeben
- wie viele Instanzen eines Entity-Typs jeweils in die Beziehung eingehen, wird durch die **Kardinalität** oder **Funktionalität** beschrieben.

Die Stelligkeit, die auch als Grad eines Beziehungstyps bezeichnet werden kann, beschreibt die Anzahl der am Beziehungstyp beteiligten Entity-Typen. Dies können, wie in Abbildung 3.4 zu sehen, entweder binäre oder wie in Abbildung 3.5 ternäre Beziehungstypen sein.

Zur genauen Definition ist zu sagen, dass ein Beziehungstyp mindestens zwei Entity-Typen, also binäre Beziehungstypen, miteinander verbinden muss. Er kann aber auch eine beliebige Anzahl $n \geq 2$ von Entity-Typen, also n-stellige Beziehungstypen, miteinander verbinden.

Zu einem n-stelligen Beziehungstyp R gehören n Entity-Typen $E_1, \ldots, E_n$.

(vgl. [SsH08] S. 63 - 64)

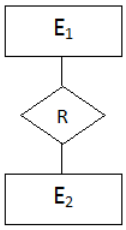

Abbildung 3.4: Binäre Beziehungstypen

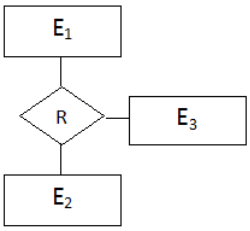

Abbildung 3.5: Ternäre Beziehungstypen

Ein weiteres wichtiges Merkmal von Beziehungstypen ist die Anzahl der beteiligten Instanzen eines Entity-Typs. Es kann beispielsweise für die Beziehung *eingeschrieben in* aus Abbildung 3.6 festgelegt werden, dass ein Student mindestens in einer Hochschule eingeschrieben sein muss (andernfalls ist es kein Student), aber auch in mehr als nur einer Hochschule eingeschrieben sein kann. Diese Festlegungen werden als **Kardinalität** von Beziehungen bezeichnet.

Abbildung 3.6: Kardinalität am Beispiel Student und Hochschule

Es gibt im Allgemeinen drei Formen von Kardinalitäten, wobei **n** und **m** für eine beliebige Anzahl > 1 stehen. Die Kardinalitäten werden hier nur für binäre Beziehungstypen betrachtet.

Die erste Form ist eine **1:1-Beziehung**, die in Abbildung 3.7 dargestellt ist. Bei dieser Beziehung ist jedem Entity von $e_1, \ldots, e_n$ vom Entity-Typ Entity1 maximal ein Entity $e_1, \ldots, e_n$ vom Entity-Typ Entity2 zugeordnet. Diese Beziehung gilt auch umgekehrt. In unseren Breitengraden ist die Beziehung "Ein Mann ist mit einer Frau verheiratet" ein gutes Beispiel dazu.

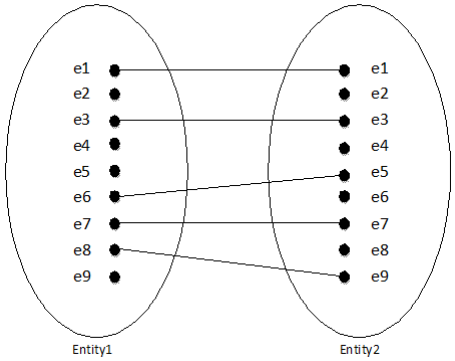

Abbildung 3.7: 1:1-Beziehung

Die zweite Form ist die **1:n-Beziehung,** welche in Abbildung 3.8 zu sehen ist. Bei dieser Beziehung sind jedem Entity von $e_1, \ldots, e_n$ vom Entity-Typ Entity1 beliebig viele Entitys vom Typ Entity2 zugeordnet, es gibt jedoch zu jedem Entity von $e_1, \ldots, e_n$ des Typs Entity2 maximal ein Entity von $e_1, \ldots, e_n$ aus dem Typ Entity1. Die Inverse hierzu ist die **n:1-Beziehung,** welche sich umgekehrt zur 1:n-Beziehung verhält. Diese wird auch als funktionale Beziehung bezeichnet. Als Beispiel kann die Beziehung "eine Mutter hat Kinder" genommen werden. Eine Mutter kann mehrere Kinder haben, aber jedes Kind hat immer nur eine biologische Mutter. Eine weitere Beziehung ist beispielsweise "Person ist Halter von KFZ". Eine Person kann der Halter von mehreren Fahrzeugen sein, aber ein Fahrzeug hat immer genau einen Halter.

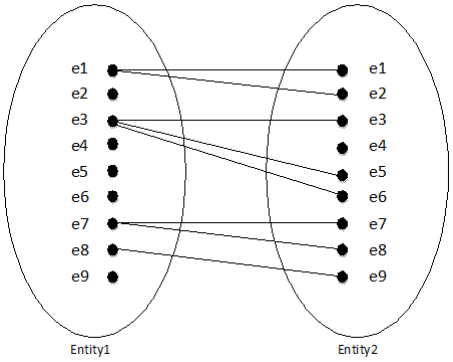

Abbildung 3.8: 1:n-Beziehung

Die letzte und allgemeinste Form ist die **n:m-Beziehung,** die in Abbildung 3.9 dargestellt ist. Hier können jedem Entity der Menge Entity1 mehrere Entitys aus Entity2 und einem Entity aus Entity2 können mehrere Entitys aus Entity1 zugeordnet sein. “Ein Kunde bestellt Produkte” ist eine mögliche Beziehung hierzu. Ein Kunde kann mehrere Produkte bestellen, und ein Produkt kann von mehreren Kunden bestellt werden. (vgl. [SsH08] S. 71 - 74)

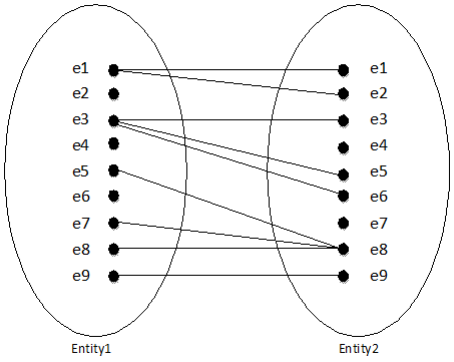

Abbildung 3.9: n:m-Beziehung

3.2.5 Integritätsbedingungen in Datenbanken

In Datenbanken existieren die physische, strukturelle und die semantische Integrität. Zur semantischen Integrität wird die werteabhängige und operationale Integrität gezählt. Im relationalen Datenmodell gehören zur strukturellen Integrität die Typ-Integrität, Schlüssel-Integrität und die Referentielle-Integrität. Da die Schlüssel-Integrität für das Verständnis der Arbeit wichtig ist, wird diese folgend näher erklärt.

Unter der Schlüssel-Integrität versteht man, dass der Wert für den Primärschlüssel immer vorhanden und eindeutig sein muss.

Neben dem eindeutigen Primärschlüssel gibt es viele weitere Schlüssel, wie z.B. zusammengesetzte Schlüssel, Fremdschlüssel u.a. in Datenbanken. Ein Fremdschlüssel ist ein Attribut, das auf den Primärschlüssel einer anderen Relation zeigt. Mindestens zwei Attribute zusammengefasst ergeben einen zusammengesetzten Schlüssel.
(vgl. [Bus10] S. 88 - 109)

3.2.6 Trigger

Ein Trigger wird zur Kontrolle von semantischen Integritätsbedingungen, die nicht auf andere Weise kontrolliert werden können, eingesetzt. Er arbeitet nach der ECA-Regel. D.h. tritt ein Ereignis (E) ein, wird die Bedingung (Condition C) der ECA-Regel geprüft und die Aktion (A) ausgeführt. (vgl. [Bus10] S. 115)

Die Struktur eines Triggers kann sich bei den unterschiedlichen DBMS unterscheiden. Der Aufbau eines Triggers in Oracle[9] nach dem Buch Grundlagen von Datenbanksystemen auf der Seite 793:

```
<Trigger> ::= CREATE TRIGGER <Trigger-Name>
(AFTER | BEFORE) <Trigger-Ereignisse> ON <Tabellenname>
[FOR EACH ROW]
[WHEN <Bedingung>]
<Trigger-Aktion>;
<Trigger-Ereignis>::= <Trigger-Ereignis> {OR <Trigger-Ereignis>}
<Trigger-Ereignis>::= INSERT | DELETE | UPDATE
[OF <Spaltenname>{,<Spaltenname>}]
<Trigger-Aktion>::=<PL/SQL-Block>
```

([EN02] S. 793)

3.2.7 Relationale Datenmodelle

Die sehr weit verbreiteten relationalen Datenbanken unterstützen ein verhältnismäßig einfaches Datenstrukturierungsmodell und der Zugang zu ihnen über Datenbanksprachen ist durch die SQL-Norm in vielen Fällen standardisiert. Außerdem werden bei diese Datenbanken die ACID-Eigenschaften[10] umgesetzt. Besonders die Konsistenz und die Dauerhaftigkeit sind für den Anwendungsfall wichtige Kriterien. Die Daten in einer Datenbank gelten als konsistent,

[9] Oracle ist ein relationales Datenbanksystem der Firma Oracle
http://www.oracle.com
[10] Atomicity, Consistency, Isolation und Durability

wenn alle Daten semantisch richtig sind. Dies kann durch Definition von Transaktionen sichergestellt werden. Die Dauerhaftigkeit besagt, dass nur konsistente Zustände dauerhaft in der Datenbank gespeichert werden. Die Dauerhaftigkeit wird von Recovery-Mechanismen des DBMS sichergestellt. (vgl. [Bus10] S. 384)

Die relationale Datenbank ist konzeptionell eine Ansammlung von Tabellen, hinter denen die mathematische Idee einer Relation steht. In Tabelle 3.1 und Tabelle 3.2 sind beispielhafte Daten von Untersuchungen sowie Patienten dargestellt.

Untersuchung	Datum	Art	Dauer (min)	Proband
	10.07.12	EKG	10:00	1
	01.05.10	Blutdruck	02:00	2
	04.12.08	EKG	12:00	2
	24.02.11	Blutdruck	05:00	5
	26.09.12	EMG	25:00	5

Tabelle 3.1: Tabelle Untersuchung

Patient	Proband	Name	Alter
	1	Müller	20
	2	Maier	20
	3	Schmidt	20
	4	Weber	20
	5	Schulz	25

Tabelle 3.2: Tabelle Patient

Abbildung 3.10 erläutert folgende begriffliche Konventionen:

Die erste Zeile gibt die Struktur einer Tabelle an. Diese Strukturinformation wird als Relationenschema bezeichnet. Alle weiteren Einträge in der Tabelle werden als Relation zu diesem Schema bezeichnet. Eine einzelne Zeile der Tabelle wird Tupel genannt. Die Spaltenüberschriften werden als Attribut bezeichnet.

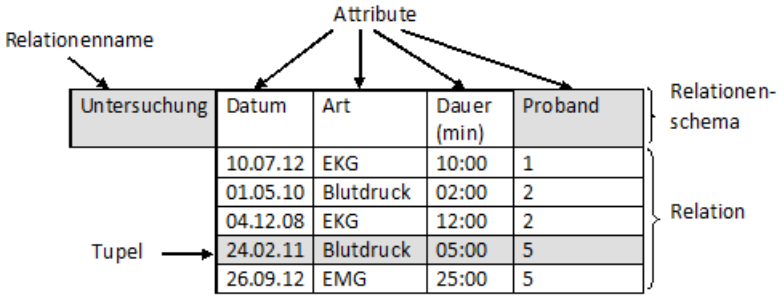

Abbildung 3.10: Begriffliche Konventionen im relationalen Modell

Anhand der Tabelle 3.1 und Tabelle 3.2 werden kurz die drei Basisanfrageoperationen auf Tabellen betrachtet.

Mit der Selektion ist es möglich, Zeilen einer Tabelle auszuwählen. Es kann eine Bedingung[11] über die Tupelwerte der zu selektierenden Zeilen angegeben werden. Die Selektion wird folgend mit SELECT bezeichnet und die Bedingung in eckige Klammern geschrieben.

```
SELECT [Proband = 5] (Untersuchung)
```

Bei dieser Anfrage werden, wie in Tabelle 3.3 zu sehen, alle Probanden mit der Nummer 5 angezeigt.

Datum	Art	Dauer (min)	Proband
24.02.11	Blutdruck	05:00	5
26.09.12	EMG	25:00	5

Tabelle 3.3: Ergebnis beim Selektieren von Zeilen

Die zweite Operation ist die Projektion, bei der Spalten ausgewählt werden. Die Projektion wird folgend mit PROJECT geschrieben:

```
PROJECT [Alter] (Patient)
```

Um eine Spalte auswählen zu können, muss hier der Attributname angegeben werden. Die Anfrage liefert als Ergebnis die folgende Tabelle mit dem Alter der Patienten:

Alter
20
25

Tabelle 3.4: Ergebnis der Projektion von Spalten

Da Tabellen als mathematische Relationen , also als (duplikatfreie) Mengen von Tupeln, interpretiert werden, werden doppelte Tupel bei der Projektion einfach entfernt.

Als letzte Operation wird der Join betrachtet. Mit ihm werden Tabellen über gleich benannte Spalten verknüpft. Dies wird erreicht, indem zwei Tupel mit den gleichen Werten verschmolzen werden.

```
PROJECT [Datum,Art] (Untersuchung)
JOIN
SELECT [Name = 'Schmidt'] (Patient)
```

[11] Diese Bedingung kann entweder wahr (true) oder falsch (false) annehmen

Das Ergebnis einer Join Operation ist eine Tabelle, die als Schema die Vereinigung der Spaltennamen der Eingangsrelation enthält. Stimmen die Werte der gemeinsamen Attribute überein, dann werden die Tupel der Eingangsrelation zu einem neuen Tupel verschmolzen. Das Ergebnis dieser Operation ist in Tabelle 3.5 zu sehen.

Datum	Proband	Name	Alter
01.05.10	2	Schmidt	20
04.12.08	2	Schmidt	20

Tabelle 3.5: Ergebnis der Join Operation

Es können ebenfalls weitere Operationen für Tabellen definiert werden. Die Operationen sind beliebig kombinierbar und bilden die sogenannte Relationenalgebra. (vgl. [SsH08] S. 11 - 15)

3.2.8 Datenmanipulation mit SQL

SQL[12] ist von der ANSI[13] und der ISO[14] standardisiert und *die* Datenbanksprache für relationale Datenbanksysteme.

Die Datenbanksprache umfasst folgende Teile: Einen Daten- und Sichtdefinitionsteil, Anfrageteil, Teil zur Formulierung von Datenänderungen, Definitionsteil für Dateiorganisationsformen und Zugriffspfade wie einen Teil mit imperativen Konstruktoren zur Definition von Funktionen und Prozeduren.

Nachfolgend werden einige wichtige SQL-Anweisungen vorgestellt.

Ein Relationenschema für eine Basisrelation wird mit der *create table* Anweisung definiert. Die Syntax ist folgendermaßen aufgebaut:

```
CREATE TABLE relationenname(
spaltenname_1 wertebereich_1 [NOT NULL],
...
spaltenname_n wertebereich_n [NOT NULL].
```

Die Relation Untersuchung kann mit der folgenden SQL-Anweisung angelegt werden:

```
CREATE TABLE Untersuchung (
ID INT NOT NULL,
Datum DATE,
```

[12] Structured Query Language ist eine Datendefinitionssprache (DDL) und eine Datenmanipulationssprache (DML)
[13] American National Standards Institute ist eine amerikanische Normungsorganisation
[14] International Standardization Organisation ist eine internationale Normungsorganisation

```
Art VARCHAR (30)NOT NULL,
Dauer TIME,
Proband INT,
PRIMARY KEY (ID) ).
```

Da die Attribute ID als Primärschlüssel und Art benötigt werden, wurden sie mit *not null* deklariert.

Tabelleninformationen können mit der *drop table* Anweisung wieder gelöscht werden:

```
DROP TABLE relationenname [RESTRICT | CASCADE].
```

Damit werden die Basisrelation aus der Datenbank sowie das Relationenschema aus dem Schemakatalog gelöscht. Mit dem Zusatz *cascade* wird das Löschen aller Sichten und Integrationsbedingungen erzwungen. Beim *restrict*, der den Defaultfall darstellt, wird, falls noch Sichten oder Integritätsbedingungen existieren, die drop Anweisung zurückgewiesen.

Das Hinzufügen, Ändern oder Löschen von Spalten sowie das Hinzufügen und Löschen von Integritätsbedingungen wird mit der *alter table* Anweisung durchgeführt:

```
ALTER TABLE relationenname modifikation.
```

Zum Kern des Anfrageteils der relationalen Datenbanksprache SQL gehören die *from*-, *select*- und *where* Klausel sowie die Darstellung der Mengenoperationen.

Mithilfe des SFW-Blocks (*select, from, where*) werden in SQL-Anfragen wie folgt formuliert:

```
SELECT projektionsliste
FROM relationenliste
[WHERE bedingung].
```

Die *Join* Anfrage auf unsere Tabelle aus dem letzten Unterabschnitt wird in SQL folgendermaßen umgesetzt:

```
SELECT Datum, p.Probnd, Name, Alter
FROM Untersuchung u, Patient p
WHERE Name = 'Schmidt' AND p.Proband = u.Proband.
```

Diese Abfrage liefert das Ergebnis, das im letzten Unterabschnitt in Tabelle 3.5 zu sehen ist. Da das Attribut Proband sowohl in der Relation Untersuchung wie auch in der Relation Patient vorhanden ist, muss in der *select* Klausel die Herkunft des Attributs Proband mit einem Präfix angegeben werden.

Mit der *delete* Anweisung lassen sich Tupel aus einer Relation löschen:

```
DELETE FROM relationenname
[WHERE bedingung].
```

Abschließend soll die grundlegende Syntax einer Prozedur erläutert werden:

```
CREATE OR ALTER PROCEDURE prozedurname
([parameter_1 ,..., parameter_n])
AS
BEGIN
[variable_1 ,..., variable_n]
prozedurkörper
END.
```

Das Schlüsselwort *as* wird als Zeichen für den Inhalt der Prozedur interpretiert, während die Schlüsselwörter *begin* und *end* eine Begrenzung für den eigentlichen Inhalt darstellen. Da es mehrere Verfahren gibt, eine Prozedur auszuführen, soll hier nur die allgemeinste aufgeführt werden:

```
EXECUTE PROCEDURE prozedurname [(eingabeparameter_1 ,..., eingabeparameter_n)]
[RETURNING_VALUES (ausgabeparameter_1 ,..., ausgabeparameter_n)].
```

(vgl. [SsH08] S. 11 - 15 und Kapitel 7)

3.3 Vor- und Nachteile von SQL-Datenbanksystemen

Vorteile SQL-Datenbanksysteme sind in sehr vielen Bereichen, wie z.B. Banken, Telekommunikationsunternehmen, Onlinewarenhäuser, u.a. effektiv im Einsatz. Sie sind im Bezug auf Verfügbarkeit und Konsistenz gut ausgereift und werden ständig verbessert. Durch die Möglichkeit, die Abfragesprache SQL einzusetzen, können u.a. sehr komplexe Anfragen durchgeführt werden. Integritätsbedingungen werden mit Hilfe vieler Mechanismen, wie z.B. verschiedenen Schlüsseln, Triggern, usw. gut umgesetzt. Das strukturierte Ablegen und Verwalten der Datenbestände ist in vielen Bereichen nötig und oft sehr hilfreich. Die Datenunabhängigkeit und Datensicherheit wird von diesen Systemen gewährleistet.

Nachteile Umso größer die Datenmengen bei SQL-Datenbanksystemen werden, desto länger braucht es, um entweder neue Daten einzufügen oder die vorhandenen Datensätze zu durchsuchen. Durch das Einfügen von Indizes in den einzelnen Tabellen ist es jedoch möglich, das Selektieren von Tupeln zu beschleunigen. Dadurch wird ein Update der einzelnen Tabellen allerdings langsamer. Dieses Konzept der Optimierung wird nicht weiter angesprochen, da es nicht Bestandteil dieser Arbeit ist. Auch das Einfügen von sehr großen Dateien (Blobs) in die Datenbank dauert Stunden. Benötigt man Daten von vielen unterschiedlichen Tabellen, so müssen diese Tabellen durch Joins verbunden werden. Diese Join-Operationen sind jedoch sehr zeitaufwendig.

Ist das Datenvolumen dieser Datenbanksysteme erreicht, so ist die bevorzugte Strategie das Aufrüsten des Datenbankservers mit neuer Hardware. Dies wird auch als vertikale Skalierung bezeichnet. Dieses Verfahren ist auf Dauer sehr kostspielig und gerät schnell an Grenzen, die nicht überwunden werden können. Die Grenzen sind erreicht, wenn es z.B. nicht mehr möglich ist, den Speicher zu vergrößern oder bessere Prozessoren nachzurüsten. Eine horizontale Skalierung wird oft von diesen Datenbanksystemen nicht oder nur sehr eingeschränkt unterstützt. Unter horizontaler Skalierung wird das Verteilen der Last auf mehrere Server verstanden. Ist das System an seine Leistungsgrenze gestoßen, so wird einfach eine neuer Server in den Verbund integriert und die Last somit weiter verteilt. Diese Methode ist kostengünstiger und weitaus effektiver.

3.4 Zusammenfassung

In diesem Kapitel wurden SQL-Datenbanksysteme vorgestellt. Diese Datenbanksysteme setzen auf eine Strukturierung in Form eines Schemas, um so Redundanzen zu vermeiden und die Integrität sicher zu stellen. Ein Datenmodell kann sehr übersichtlich in Form eines ER-Modells entwickelt werden. In diesem Modell sind alle Beziehungen und Kardinalitäten unter den Entitätstypen dargestellt. Die Erstellung dieses ER-Modells ist Bestandteil des konzeptionellen Entwurfs. Trigger, die im logischen Entwurf entwickelt werden, dienen der Sicherstellung der semantischen Integrität. Mit der Abfragesprache SQL lassen sich übergreifende Anfragen an die Datenbank stellen. Im Anwendungsfall lassen sich so z.B. bei den Messdaten Informationen über verschiedene Versuche hinweg ermitteln.

Im folgenden Kapitel sollen die Grundlagen von NoSQL-Datenbanksystemen in den Mittelpunkt gerückt werden.

4 NoSQL-Datenbanksysteme

4.1 Einleitung

Wie bei den SQL-Datenbanksystemen soll auch hier mit einer möglichen Definition des Begriffs NoSQL begonnen werden. NoSQL steht nicht, wie viele denken, für *No SQL* sondern lediglich für *Not only SQL*. Da diese Systeme noch relativ neu sind, haben sich bisher nur wenige Gremien oder Organisationen mit einer Begriffsklärung auseinandergesetzt. Eine mögliche Definition ist die deutsche Übersetzung aus dem NoSQL-Archiv[15] zitiert aus dem Buch NoSQL - Einstieg in die Welt Nichtrelationaler Web 2.0 Datenbanken.

> "Definition:
>
> Unter NoSQL wird eine neue Generation von Datenbanksystemen verstanden, die meistens einige der nachfolgenden Punkte berücksichtigen:
>
> 1. Das zugrundeliegende Datenmodell ist nicht relational.
>
> 2. Die Systeme sind von Anbeginn an auf eine verteilte und horizontale Skalierbarkeit ausgerichtet.
>
> 3. Das NoSQL-System ist Open Source.
>
> 4. Das System ist schemafrei und hat nur schwächere Schemarestriktionen.
>
> 5. Aufgrund der verteilten Architektur unterstützt das System eine einfache Datenreplikation.
>
> 6. Das System bietet eine einfache API.
>
> 7. Dem System liegt meistens auch ein anderes Konsistenzmodell zugrunde:
>
> Eventually Consistent und BASE, aber nicht ACID [...]." ([EFH+11] S. 2)

Laut Edlich wird ein DBMS als NoSQL-System angesehen, wenn es einige dieser Eigenschaften erfüllt. (vgl. [EFH+11])

[15] http://nosql-database.org

Da für die Datenbanklösung keine Mittel zur Verfügung stehen, ist die Eigenschaft drei für den Anwendungsfall ein wichtiges Kriterium. Außerdem ist auch die Eigenschaft sieben wichtig, da eine Anforderung an die Datenbank ein konsistenter Zustand der Daten ist. Hier muss geprüft werden, ob das Konsistenzmodell der jeweiligen NoSQL-Datenbanksysteme dieser Anforderung genügt.

In diesem Kapitel können ebenfalls nur die Grundlagen angesprochen werden. Für eine Vertiefung in diesem Themengebiet wird auf Fachliteratur, wie z.B. das Buch [EFH+11] verwiesen.

4.2 Historischer Hintergrund von NoSQL-Datenbanksystemen

Die Vorgänger von NoSQL-Systemen gab es bereits in den 1980er Jahren. Schon 1979 wurde eine Key/Hash-Datenbank namens DBM von Ken Thompson entwickelt. Populäre Vorreiter der heutigen NoSQL-Systeme waren z.B. Lotus Notes, GT.M und BerkleyDB. Diese gab es ebenfalls schon seit den 80er Jahren. Als 1998 Carlo Strozzi die erste Datenbank, zwar noch mit einem zugrundeliegenden relationalen Datenmodell, aber ohne SQL-API[16] entwickelte, tauchte erstmalig der Begriff NoSQL auf.

Da Postrelationale-Datenbanken, wie z.B. Grid-Datenbanken, Objektdatenbanken, XML-Datenbanken sowie viele andere, ebenfalls nichtrelationale Systeme sind, gehören sie genau genommen ebenfalls zu den NoSQL-Datenbanken. Die typischen Vertreter, zu denen Key/Value Stores, Wide Column Stores, Document Stores und Graphendatenbanken zählen, werden jedoch deutlich mehr mit dem Begriff der NoSQL-Datenbanken in Verbindung gebracht.

Als ab dem Jahr 2000 das Web 2.0 mit immer größer werdenden Datenmengen aufkam, bekam die NoSQL-Bewegung ihren eigentlichen Schub, der bis heute anhält.

(vgl. [EFH+11] S. 1)

4.3 Grundlagen von NoSQL-Datenbanksystemen

In dem folgenden Abschnitt soll neben den allgemeinen Grundlagen auch kurz auf die für diese Arbeit wichtigen Grundlagen von Wide Column Stores und Document Stores eingegangen werden.

[16] Application program interface

4.3.1 Map/ Reduce

Da die zu bewältigenden Daten immer größer wurden, mussten neue Algorithmen, Frameworks und Datenbankmanagementsysteme entwickelt werden.

Das Map/Reduce-Framework wurde von den Entwicklern Jeffrey Dean und Sanjay Ghemawat 2004 bei der Firma Google Inc. entwickelt. Auf dieses Framework hat Google Inc. im Jahr 2010 vom US-amerikanischen Patentbüro ein Patent[17] erhalten. Das Prinzip dieses Frameworks soll kurz erläutert werden. Da ein inhaltliches Verständnis für die Implementierung der Datenbank nicht notwendig ist, werden hier nur Grundlagen angesprochen.

Map/Reduce hat als Grundidee funktionale Programmiersprachen wie zum Beispiel LISP[18]. In diesen funktionalen Programmiersprachen ist das Konzept einer Funktion mathematisch klar umgesetzt. Die Funktionen *map()* und *fold()*, was auch als *reduce()* bezeichnet wird, sind Bestandteil dieser Sprachen. Diese Funktionen werden in modifizierter Form beim Map/Reduce-Algorithmus nebenläufig in zwei Phasen ausgeführt. Die Funktion *map()* wird auf alle Elemente einer Liste angewendet und gibt eine durch die Funktion modifizierte Liste zurück. *Reduce()* sammelt einzelne Funktionsergebnisse dieser Listenpaare und verkleinert sie auf einen Ausgabewert. Durch die Modifizierung im Map/Reduce-Algorithmus werden diese Funktionen, wie in Abbildung 4.1, nach dem Buch [EFH+11], dargestellt, parallel auf verschiedenen Knoten in zwei Phasen hintereinander angewendet. Dadurch lassen sich große Datenmengen bewältigen.

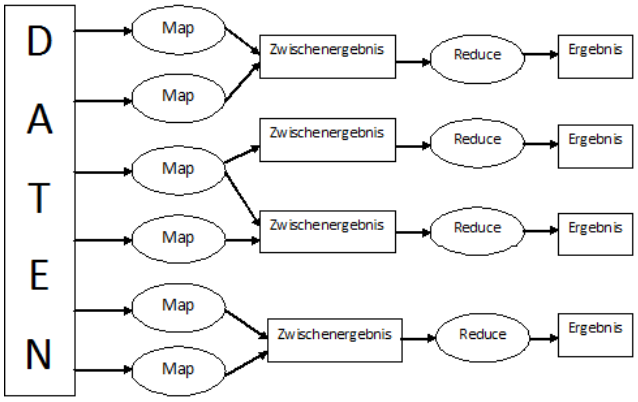

Abbildung 4.1: Datenfluss des Map/Reduce-Verfahrens

Das Map/Reduce Verfahren kommt in vielen Bereichen zur Anwendung, wie z.B. beim verteilten Suchen (Grep), Zählen von Zugriffen auf eine Webseite, Sortieren verteilter Daten

[17] http://patft.uspto.gov; Patent Nummer: US007650331
[18] LISP steht für List Processing

u.a. Implementiert wurde es unter anderem von Apache bei seinem Open Source-Java-Framework Hadoop, bei der IBM-Cell-Prozessorserie, für NVIDA GPUs, usw.

(vgl. [EFH+11] Kapitel 2.1)

4.3.2 CAP-Theroem

Auf dem ACM-Symposium über Principles of Distributed Computing im Jahr 2000 stellte Eric Brewer seine Forschungsarbeit über das CAP-Theorem vor. Seine Forschung zu verteilten Systemen an der University of California stellte erstmalig einen Bezug zwischen der bekannten Theorie und der Praxis her. (vgl. [EFH+11] S. 31) Er machte bei diesem Vortrag deutlich, dass es nicht möglich ist, bei großen verteilten Datenbanken gleichzeitig Konsistenz (Consistency), Verfügbarkeit (Availability) und Ausfalltoleranz (Partition Tolerance) zu garantieren. Seine Kernaussage zum CAP-Theorem ist, dass verteilte Datenbanken nur maximal zwei dieser Größen erreichen können. Deshalb sind die Datenbanksysteme in Abbildung 4.2, nach ([Sch12] S. 32), auch direkt auf den Achsen eingezeichnet. Diese aufgestellte Vermutung von Brewers wurde 2002 von Gilbert und Lynch bewiesen. (vgl. [Sch12] S. 15)

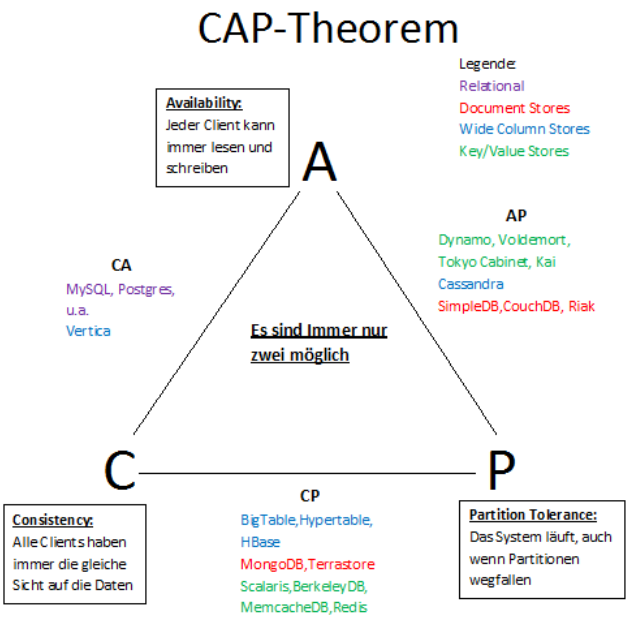

Abbildung 4.2: CAP-Theorem

Mit der **Konsistenz** ist gemeint, dass nach Abschluss einer Transaktion immer ein konsistenter Zustand in der verteilten Datenbank erreicht wird.

Die **Verfügbarkeit** steht für eine akzeptable Reaktionszeit. Die verteilte Datenbank muss zu jeder Zeit voll verfügbar sein.

Mit der **Ausfalltoleranz** ist gefordert, dass selbst nach dem Ausfall von anderen Knoten (Partitionen) die Datenbank noch komplett funktionsfähig sein muss.

(vgl. [EFH+11] Kapitel 2.2.2)

4.3.3 Wide Column Stores

Die schon aus den 80er Jahren stammende Idee zu den Wide Column Stores verfolgt den Ansatz, die Daten nicht wie bei den relationalen Modellen in einer Tabelle untereinander (reihenorientiert) zu speichern, sondern jedes Attribut in einer eigenen Tabelle hintereinander (spaltenorientiert). In der Tabelle 4.1 ist die reihenorientierte Speicherung der Daten: ID, Name und Alter zu sehen, während in Tabelle 4.2 zu erkennen ist, wie diese Daten spaltenorientiert gespeichert werden.

1, Michael, 16, 2, Andrea, 31, 3, Michaela, 24

Tabelle 4.1: Reihenorientierte Speicherung wie bei relationalen Modellen

1, 2, 3, Michael, Andrea, Michaela, 16, 31, 24

Tabelle 4.2: Spaltenorientierte Speicherung wie bei Wide Column Stores

Die Vorteile dieser Art der Speicherung liegen in der besseren Möglichkeit der Aggregation von Daten, im Caching, bei der Datenkompression sowie bei der Analyse der Daten. Jedoch ist das Suchen und Einfügen von Daten sowie das Schreiben und Lesen von Objektstrukturen durch das Springen in den Spalten und Suchen der zusammengehörenden Daten sehr aufwendig.

Zu der Kategorie der Wide Column Stores gehören Vertreter wie MonetDB, C-Store, FluidDB sowie SybaseIQ.

Die Datenbanken Cassandra, Amazon SimpleDB und HBase gehören zwar auch dieser an, weichen aber in der Frage des angebotenen Dimensionsformates von der eben beschriebenen Idee stark ab. Bedingt durch das Design von Google BigTable ist eine andere Architektur vorgegeben. Hier werden in der Regel mehrdimensionale Tabellen mit folgendem Format verwendet:

$$n \cdot [Domain\ /\ Keyspace]\ x\ [Item\ /\ Column\ Family]\ x\ [Key\ x]\ n \cdot [Key\ +\ Value]$$

Also gibt es bei diesen Systemen meistens mehrere Dimensionen aus Domains oder Keyspaces, deren kleinstes Feld ein Item oder eine Column Family bildet. In diesem kleinsten Feld gibt

es beliebig viele Key/ Value Maps, die über einen übergeordneten Key angesprochen werden können. Die unterste Ebene ist also eine Menge aus Key/ Value Maps, die eine Einheit bilden. (vgl. [EFH+11] S. 63 - 64)

CRUD-Operationen[19] Es gibt verschiedenen Ansätze, um die Daten in der Datenbank zu manipulieren. Diese Ansätze unterscheiden sich bei den einzelnen Systemen, sind jedoch grundlegend ähnlich. In den meisten Datenbanken ist es möglich, Daten über eine Programmiersprache, wie zum Beispiel Java, C++, PHP, etc., zu manipulieren.

Die wichtigsten Operationen wie das Erstellen eines Schemas sowie das Einfügen und Auslesen von Daten soll am Beispiel der Datenbank HBase in der Programmiersprache Java erklärt werden. Diese Datenbank wurde gewählt, da sie ein Open Source-Projekt ist und damit Eigenschaft drei aus der Definition zu Beginn des Kapitels erfüllt sowie eine Versionsverwaltung besitzt, mit der sich bereits eine Anforderung, die an die zukünftige Datenbank gestellt wird, umsetzen lässt.

HBase verfügt außerdem über das alternative Konsistenzmodell BASE[20], welches ebenfalls bei der Definition zu Beginn des Kapitels erwähnt wurde. Bei diesem Konsistenzmodell ist allerdings die Verfügbarkeit und nicht die für den Anwendungsfall benötigte Konsistenz die wichtigste Eigenschaft. Bei diesem Modell erreichen die Systeme einen konsistenten Zustand erst nach einem gewissen Zeitfenster der Inkonsistenz. Wird jedoch ein konsistenter Zustand wie er duch das Konsistenzmodell ACID sichergestellt wird benötigt, so muss dies mit zusätzlichen Mechanismen außerhalb des Datenbanksystems umgesetzt werden. (vgl. [EFH+11] S. 33 - 35)

Mit folgendem Quelltextauszug wird das Schema *Untersuchung* aus Kapitel 3, welches in Tabelle 3.1 auf Seite 26 zu finden ist, erstellt:

```
HTableDescriptor tabelleUntersuchung = new HTableDescriptor ("Untersuchung");
HColumnDescriptor datum = new HColumnDescriptor ("Datum");
HColumnDescriptor art = new HColumnDescriptor ("Art");
HColumnDescriptor dauer = new HColumnDescriptor ("Dauer");
HColumnDescriptor proband = new HColumnDescriptor ("Proband");
```

Der vollständige Quelltext ist in Listing 7.6 im Anhang zu finden. Wie man an diesem Beispiel gut sehen kann, gibt es bei der Datenbank HBase keine Datentypen, die definiert

[19] CRUD ist ein Akronym für die Datenbankoperationen **C**reate, **R**ead, **U**pdate und **D**elete

[20] Basically Available, Soft State, Eventually Consistent und hat als wichtigstes Kriterium die Verfügbarkeit und nicht die Konsistenz

werden müssen. Das liegt daran, dass HBase alle Daten in Byte Arrays speichert.

Das Einfügen eines Datensatzes in die Tabelle *Untersuchung* wird wie folgt umgesetzt:

```
Put insertDatum = new Put (Bytes.toBytes ("Datum"));
insertDatum.add (Bytes.toBytes ("Datum"), Bytes.toBytes ("ErstesDatum"),
Bytes.toBytes ("10.07.12"));
```

Der vollständige Quelltext ist wieder in Listing 7.7 im Anhang zu finden.

Das Auslesen von Daten lässt sich folgendermaßen realisieren:

```
Get selectDatum = new Get (Bytes.toBytes ("Datum"));
Result myResult = tabelleUntersuchung.get (get);
byte [] myValue = myResult.getValue (Bytes.toBytes ("Datum"),
Bytes.toBytes ("ErstesDatum"));
System.out.println (Bytes.toBytes (myValue));
```

In Listing 7.8 im Anhang ist der komplette Quelltext zu finden. Das Feld *ErstesDatum* ist ein Schlüssel, der zur Identifizierung der Zeile herangezogen wird.

Werden Fremdschlüssel benötigt, so müssen diese in Form von zusätzlichen Tabellen umgesetzt werden. Im Beispiel aus Kapitel 3 war das Attribut Proband in der Tabelle Untersuchung ein Fremdschlüssel auf die Tabelle Patient. In Tabelle 4.3 ist eine Umsetzung in HBase dargestellt. Der RowKey ist der eindeutige Schlüssel (Primärschlüssel) der Zeile und muss in HBase zwingend ein Byte Array sein.

Tabelle:Untersuchung		
RowKey:	UntersuchungID	
Family	data:	Columns: Datum, Art, Dauer

Tabelle:Patient		
RowKey:	ProbandID	
Family	data:	Columns: Name, Alter

Tabelle:Patient-Versuch		
RowKey:	ProbandID\x00UntersuchungID	
Family	data:	Columns: Timestamp

Tabelle 4.3: Fremdschlüssel in HBase

Möchte man komplexere Anfragen stellen, so ist dies entweder über das Map/ Reduce Verfahren oder über selbst entwickelte Funktionen möglich. Auch alle anderen etwas komplexeren Aufgaben, wie z.B. das Schützen von Datensätzen oder ähnliches, ist nur über die Anwendungsschicht, durch eigenständig entwickelte Funktionen möglich. Im Gegensatz zu den

SQL-Datenbanken gibt es in HBase keine Datentypen. Alle Daten müssen in der Datenbank in einem Byte Array gespeichert werden. Durch das nötige Umwandeln sind Operationen sehr komplex und zeitaufwendig. Außerdem muss in der Anwendung sichergestellt werden, dass nur korrekte Daten in die entsprechenden Tabellen eingefügt werden. Das Darstellen von Beziehungen lässt sich bei diesen Systemen nur sehr kompliziert durch das Verschachteln von Tabellen erreichen. (vgl. [Geo11])

4.3.4 Document Stores

Bei den Document Stores handelt es sich um schemafreie Systeme, die die Daten als zusammenhängende Dokumente in einer JSON[21] oder BSON[22] Struktur abspeichern. Jedes Dokument wird über eine eindeutige ID referenziert. Um ein schnelleres Suchen zu ermöglichen, können beliebige Keys als Indizes definiert werden. Ein JSON Dokument kann wie folgt aufgebaut sein:

```
"Titel" : "Untersuchung",
"Datum" : "10.07.12",
"Art" : "EKG",
"Dauer" : "10:00"
```

Die Stärke der Document Stores, im Gegensatz zu den Wide Column Stores, besteht darin, dass sie die Daten, sofern sie als Dokument im System gespeichert werden, interpretieren können. Dadurch kann sehr einfach nach einzelnen Begriffen im Dokument gesucht werden.

Vertreter dieser Document Stores sind zum Beispiel CouchDB, RavenDB, OrientDB, ThruDB oder MongoDB.

CRUD-Operationen Auch bei den Document Stores können verschiedenen Programmiersprachen, wie z.B. Java, C++ oder PHP zur Datenmanipulation verwendet werden. Am Beispiel von MongoDB, die ebenfalls ein Open Source-Projekt ist, und der Programmiersprache Java sollen grundlegende Operationen vorgestellt werden. Auch MongoDB bietet ebenso wie HBase nur *Eventual Consistency* und stellt damit keinen konsistenten Zustand der Datenbank sicher.

Da diese Datenbanken schemalos sind, ist es nicht nötig, ein Schema, eine Datenbank oder

[21] JSON steht für Java Script Object Notation
[22] BSON steht für Binary JSON Format

Collections[23] zu definieren. Beim ersten Einfügen eines Dokuments[24] wird die Datenbank und die Collection zur Laufzeit erzeugt. Da bei der Implementierung das Einfügen von sehr großen Dateien getestet werden soll, ist es wichtig zu wissen, dass es durch eine Restriktion nicht möglich ist, Dokumente im BSON Format einzufügen, die größer als 16 Megabyte sind. Sollen größere Dateien eingefügt werden, so wird dies in MongoDB durch den Mechanismus GridFS, bei dem die Daten gesplittet und über mehrere Dokumente verteilt werden, gelöst. Dadurch können auch große Binärdateien, Bilder, Videos oder ähnliches gespeichert werden.

Das Erstellen der Datenbank und einer Collection wird wie folgt implementiert:

```
Mongo m = new Mongo ("localhost", 27017);
DB db = m.getDB ("Messdaten");
DBCollection proband = db.getCollection ("Proband");
```

Das Einfügen von Daten in die Collection wird folgendermaßen umgesetzt:

```
BasicDBObject insertProband = new BasicDBObject ();
insertProband.put ("ProbandID", 1);
insertProband.put ("Alter", 38);
prband.insert (insertProband);
```

Die vollständigen Quelltexte sind in Listing 7.13 und Listing 7.14 im Anhang dargestellt.

Sollen, wie in Tabelle 3.3 auf Seite 27, alle Probanden mit der Nummer 5 ausgegeben werden, so lässt sich das mit folgendem Quelltext umsetzen:

```
BasicDBObject query = new BasicDBObject ();
query.append ("Proband", 5);
DBCursor cursor = collection.find (query);
while (cursor.hasNext () )
System.out.println (cursor.next () );
```

Das Einfügen von Fremdschlüsseln ist auch bei diesen Systemen nicht ganz einfach. Eine Beziehung zwischen unterschiedlichen Collections wird in Java durch den Datentyp DBRef erreicht. In diesem wird der Name der referenzierten Collection und die eindeutige Objekt Id gespeichert. Wie ein Fremdschlüssel zwischen der Tabelle Versuch und Proband hergestellt und wie auf ihn zugegriffen wird, ist in Listing 4.1 dargestellt.

[23] Jede Datenbank unterteilt sich in Collections, die mit den Tabellen in der relationalen Welt vergleichbar sind

[24] Was in etwa den Zeilen in relationalen Datenbanken entspricht

Listing 4.1: Fremdschlüssel erzeugen und darauf zugreifen

```
DBRef versuchProband = new DBRef (db, "Proband", ProbandID);
DBObject proband = versuchProband.fetch ();

//Fremdschlüssel anlegen
DBObject versuch = BasicDBObjectBuilder.start ()
        .add ("VersuchID", 1)
        .add ("ProbandID", versuchProband)
        .add ("Datum", "27-01-2011")
        .add ("Dauer", "00:10:00")
        .get ();
collection.save (versuch);

DBObject nameProband = collection.findOne ();
DBRef probandObj = (DBRef) nameProband.get ("ProbandID");
probandObj.fetch ();
```

Sind komplexere Interaktionen mit der Datenbank nötig, so muss dies entweder mit dem Map/ Reduce Verfahren oder einer eigenen Implementierung umgesetzt werden. Im Gegensatz zu relationalen Datenbanken müssen alle Operationen mit der Datenbank vorher ausgearbeitet und in einer Programmiersprache implementiert werden. Dadurch ist es bei diesen Systemen nicht möglich flexibel und schnell auf Änderungen in Abläufen zu reagieren. Da die eingefügten Daten in keiner Weise geprüft werden, muss auch dies in der Anwendung berücksichtigt werden. (vgl. [EFH+11] Kapitel 4.2)

4.4 Vor- und Nachteile von NoSQL-Datenbanksystemen

Vorteile Laut dem Buch [EFH+11] ist die einfache horizontale Skalierung sowie die leicht zu implementierende Replikation ein klarer Vorteil von NoSQL-Datenbanken. Der Begriff der horizontalen Skalierung steht für das Hinzufügen immer neuer Datenbankserver zum bestehenden System. Dadurch kann enorm viel Geld gespart werden, denn es ist nicht mehr nötig, teure Server mit hochgerüsteter Hardware zu kaufen. Ein weiterer Vorteil nach dem Buch [EFH+11] ist die gute Handhabung von großen Datenmengen bis in den Petabyte-Bereich. Demnach kann sehr einfach und schnell mit Millionen von Einträgen umgegangen werden.

Nachteile Will man komplexe Zusammenhänge unter den Datenbeständen darstellen, so sind diese Systeme dafür nicht gut geeignet. Dazu ist ein erheblicher Mehraufwand nötig. Die komplexe Datenmanipulation muss erst sehr aufwendig im Bereich der Anwendung implementiert werden.

4.5 Zusammenfassung

In diesem Kapitel stellte sich heraus, dass NoSQL-Datenbanken ein grundlegend anderes Konzept als relationale Datenbanken haben. Dadurch sind sie in vielen Bereichen flexibler, können sehr gut in verteilten Systemen eingesetzt werden und sind für das Bewältigen von sehr großen Datenmengen konzipiert.

Datenmanipulationen müssen allerdings aufwendig in Programmiersprachen, wie z.B. Java implementiert werden. Dadurch ist es nicht möglich, wie bei den relationalen Datenbanken spontan auf Änderungen in der Struktur oder im Arbeitsablauf zu reagieren. Kommen z.B. weitere Anfragen, die zur Zeit noch nicht berücksichtigt werden, aber für die Auswertung entscheidend sind, hinzu, so ist es bei den relationalen Datenbanken durch die integrierte Datenbanksprache SQL möglich, diese sofort umzusetzen. Bei den NoSQL-Systemen muss dafür erst die Anwendung angepasst werden. Des Weiteren ist das Darstellen von Beziehungen sehr komplex. Egal, ob es wie bei MongoDB, mit dem Datentyp *DBRef* oder bei HBase durch zusätzliche Tabellen umgesetzt wird, es ist in jedem Fall sehr aufwendig. Im Gegensatz zu relationalen Datenbanken ist bei diesen Systemen die Typ-Integrität nicht gewährleistet. Diese muss durch die Anwendung sichergestellt werden. Ein weiterer wichtiger Unterschied zwischen SQL- und NoSQL-Datenbanksystemen sind die Konsistenzmodelle. Bei den SQL-Systemen wird meist das Konsistenzmodell ACID eingesetzt, welches einen konsistenten Zustand der Daten durch Transaktionen, die vom Anwender definiert werden müssen, sicherstellt. Bei den NoSQL-Systemen wird häufig das Konsistenzmodell BASE verwendet, welches aber nicht den konsistenten Datenzustand, sondern die Verfügbarkeit der Daten in den Vordergrund stellt. Wird bei diesen Systemen dennoch ein konsistenter Zustand der Daten benötigt, so muss auch dies in der Anwendung, also außerhalb des Datenbanksystems, umgesetzt werden.

Abschließend kann man sagen, dass bei den NoSQL-Datenbanken viele Funktionalitäten erst in einer Anwendung implementiert werden müssen, die bei den relationalen Datenbanken schon vorhanden sind.

5 Evaluation und Implementierung einer Datenbanklösung

In Kapitel 2 wurde bereits ausführlich die vorhandene Struktur und die Vielfältigkeit der durchgeführten Versuche betrachtet. Um eine effektive Datenbanklösung zu finden, muss im ersten Schritt analysiert werden, welche Anforderungen an die Datenbank gestellt werden. Im nächsten Schritt wird aus den Ergebnissen dieser Analyse im konzeptionellen Entwurf ein ER-Modell entwickelt. Nach der Wahl der Datenbankmanagementsysteme (DBMS) wird im logischen Entwurf ein Datenmodell für die relationalen Datenbanken und MongoDB sowie eine Tabellenstruktur für HBase entworfen. Anschließend werden die gewählten Datenbanken implementiert und es wird geprüft, welche den Anforderungen entsprechen. Nach dem Benchmark-Test, bei dem geprüft wird, wie lange zwei ausgewählte Datenbanken für Standardoperationen benötigen, wird in der Zusammenfassung dieses Kapitels die Entscheidung getroffen und begründet, welches System am besten für die Lösung der Problemstellung geeignet ist.

5.1 Anforderungsanalyse

Die Anforderungsanalyse teilt sich in die funktionalen Anforderungen, bei denen festgelegt wird, welche Funktionen die Datenbanklösung erfüllen muss und die nichtfunktionalen Anforderungen, bei denen festgelegt wird, welche Eigenschaften diese erfüllen muss.

Funktionale Anforderungen In der Datenbank sollen die Versuchsdaten zu den durchgeführten medizinischen Messungen, die Metainformationen (Header, Signal, Kommentar) zu den jeweiligen Versuchen sowie Daten zu den Probanden für mindestens zehn Jahre gespeichert werden.

Zu den Probanden sollen folgende Daten gespeichert werden: Eindeutige ID, Alter, Geschlecht, Größe, Gewicht, Armlänge und Beinlänge.

Für einen durchgeführten Versuch werden folgende Daten aufgenommen: eindeutige ID, Datum, Dauer, Versuchsart, Ergebnis und Dateiname. Diese Daten dürfen nach dem Einfügen in die Datenbank nur zu Zwecken der Auswertung ausgelesen, aber nicht verändert oder gelöscht werden. Die Datensätze, die ausgewertet wurden, müssen separat, aber in Beziehung zu den originalen Datensätzen, mit einer eindeutigen ID und dem neuen Ergebnis abgespeichert werden. Es muss auch aufgenommen werden können, von welchen Probanden die Versuche durchgeführt wurden. Versuchsergebnisse von mindestens 13 GB Größe sollen in der Datenbank gespeichert und wieder ausgelesen werden können. Sollte dies nicht möglich oder unpraktikabel sein, so muss sichergestellt werden, dass eine Beziehung zwischen den Metadaten und den Versuchsergebnissen hergestellt werden kann.

Für den Header sollen folgende Daten gespeichert werden: Eindeutige ID, Anzahl der Signale, Abtastrate, Anzahl der Signalpunkte und Kommentare. Außerdem muss ein Header, wenn vorhanden, einem Versuch zugeordnet werden können.

Für die Signale werden folgende Daten aufgenommen: Eindeutige ID, Zeilennummer, Anzahl der Bits, Umrechnungsfaktor, ADC Resolution, ADC Zero, Anfangswert, Skalierungsfaktor, Achsenverschiebung und Kommentare. Die Signale müssen einem Header zugeordnet werden können.

Für Kommentare sollen folgende Daten gespeichert werden: Eindeutige ID, Zeilennummer und der Kommentar. Ein Kommentar muss einem Versuch zugeordnet werden können.

Zu den Hebe- und Bewegungsversuchen müssen die Arm- und Beinlängen der entsprechenden Probanden ausgelesen werden können.

Zu den Eye Tracking Versuchen soll das Alter und Geschlecht der entsprechenden Probanden ausgelesen werden können.

Es muss möglich sein, zu einem Versuch einzelne Metainformationen, wie z.B. Kommentare oder die Anzahl der Signale des Versuchs auszulesen.

Es sollte möglich sein, spontan weitere Anfragen an die Datenbank zu stellen.

Nichtfunktionale Anforderungen Das System muss nach einem Systemfehler vollständig wiederhergestellt werden können.

Es sollte intuitiv zu bedienen, die Struktur sollte nicht zu komplex und es soll keine große Einarbeitung nötig sein.

Zur Leistung und Effizienz lässt sich sagen, dass die Datenbanklösung auf keinen Fall länger für das Speichern von Versuchsdaten benötigen soll als das zurzeit eingesetzte System.

Falls zu einem späteren Zeitpunkt neue Messgeräte eingesetzt werden, so muss das System problemlos angepasst oder verändert werden können.

Es muss sichergestellt werden, dass nur autorisierte Nutzer auf die Datenbank zugreifen können, der Datenbestand konsistent und verfügbar ist sowie die Ergebnisse von Interaktionen mit der Datenbank immer korrekt sind. Eine Hochverfügbarkeit der Datenbank ist aber nicht nötig.

Aufgrund fehlender Mittel sollen die Kosten für die Datenbanklösung so gering wie möglich sein.

Aus diesen gesammelten Informationen ergeben sich folgende Anforderungen an die Datenbanklösung:

DB allgemein	1	In der Datenbank sollen Daten über Versuche, Probanden, Header, Signale und Kommentare für mindestens 10 Jahre gespeichert werden.
	2	Es müssen Daten mit einer Größe von mindestens 13 GB oder ein Verweis auf diese Daten in der Datenbank gespeichert werden können.
	3	Unterschiedliche Dateiformate sollten in der Datenbank gespeichert und wieder ausgelesen werden können.
	4	Das System muss nach einem Systemfehler vollständig wiederhergestellt werden können.
	5	Es sollte intuitiv bedienbar sein.
	6	Die Struktur sollte möglichst nicht zu komplex sein.
	7	Es soll keine große Einarbeitung nötig sein.
	8	Das Speichern von Versuchsdaten soll nicht länger als mit dem zur Zeit eingesetzten System dauern.
	9	Das System muss problemlos angepasst oder verändert werden können.
	10	Nur autorisierte Nutzer dürfen auf die Datenbestände zugreifen.
	11	Die Datenbank muss einen konsistenten Zustand haben.
	12	Sie sollte möglichst immer verfügbar sein.
	13	Das Ergebnis einer Datenbankabfrage muss immer korrekt sein.
	14	Die Kosten für die Datenbanklösung sollen so gering wie möglich sein.
Proband	15	Für den Proband werden folgende Daten aufgenommen: Eindeutige ID, Alter, Geschlecht, Größe, Gewicht, Armlänge und die Beinlänge.
Versuch	16	Für den Versuch werden folgende Daten aufgenommen: Eindeutige ID, Datum, Dauer, Versuchsart, Ergebnis und Dateiname.
	17	Probanden, die den Versuch durchgeführt haben, müssen aufgenommen werden.
	18	Die Rohdaten dürfen nicht verändert oder gelöscht werden.
Nachfolger	19	Für den Nachfolger werden folgende Daten aufgenommen: Eindeutige ID, Datum, Ergebnis und Dateiname.
	20	Ausgewertete Daten müssen in Bezug auf die Rohdaten gespeichert werden können.
Header	21	Für den Header werden folgende Daten aufgenommen: Eindeutige ID, Anzahl der Signale, Abtastrate, Anzahl der Signalpunkte und Kommentare.
	22	Zu einem Header muss genau ein Versuch aufgenommen werden.
Signal	23	Für ein Signal werden folgende Daten aufgenommen: Eindeutige ID, Zeilennummer, Anzahl der Bits, Umrechnungsfaktor, ADC Resolution, ADC Zero, Anfangswert, Skalierungsfaktor, Achsenverschiebung und Kommentar.
	24	Zu einem Signal muss genau ein Header aufgenommen werden.
Kommentar	25	Für Kommentare werden folgende Daten aufgenommen: eindeutige ID, Zeilennummer und Kommentare.
	26	Zu einem Kommentar muss genau ein Versuch aufgenommen werden.
Anfragen	27	Bei Hebe- und Bewegungsversuchen muss die Arm- und Beinlänge des Probanden ausgelesen werden.
	28	Beim Eye Tracking Versuch muss das Alter und Geschlecht des Probanden ausgelesen werden.
	29	Bei Versuchen muss es möglich sein, Metainformationen, wie z.B. einen Kommentar oder die Anzahl der Signale, auszulesen.
	30	Es sollte möglich sein, spontan weitere Anfragen an die Datenbank zu stellen.
	31	Zu einem Nachfolger muss der Rohdatensatz ausgelesen werden können.

Tabelle 5.1: Gestellte Anforderungen an das Datenbanksystem

5.2 Konzeptioneller Entwurf

Ein mögliches ER-Diagramm ist in Abbildung 5.1 zu sehen. Die Kardinalitäten im ER-Diagramm, wie im Kapitel 3.2.6 bei den Entitätstypen vorgestellt, ergeben sich aus der Erkenntnis, dass es einen Versuch gibt, welcher viele Nachfolger haben kann, ein Nachfolger aber genau einen Vorgänger haben muss. Weiterhin kann ein Proband an vielen Versuchen teilnehmen, aber ein Versuch hat immer nur einen Probanden. Bei der Verbindung zwischen der Tabelle Versuch und Header handelt es sich um eine 1:1-Beziehung. (vgl. Kapitel 3.2.7 Beziehungen) Das heißt, ein Versuch kann einen Header haben, ein Header muss aber genau einem Versuch zugeordnet sein. Bei der Verbindung zwischen Header und Signal ist es so, dass ein Header mindestens ein Signal haben muss, aber mehrere Signale haben kann. Ein oder mehrere Signale hingegen haben immer genau einen zugeordneten Header. Abschließend ist die Beziehung zwischen Versuch und Kommentar zu betrachten. Ein Versuch kann einen oder mehrere Kommentare haben, aber ein oder mehrere Kommentare sind immer genau einem Versuch zugeordnet. (vgl. Tabelle 5.2)

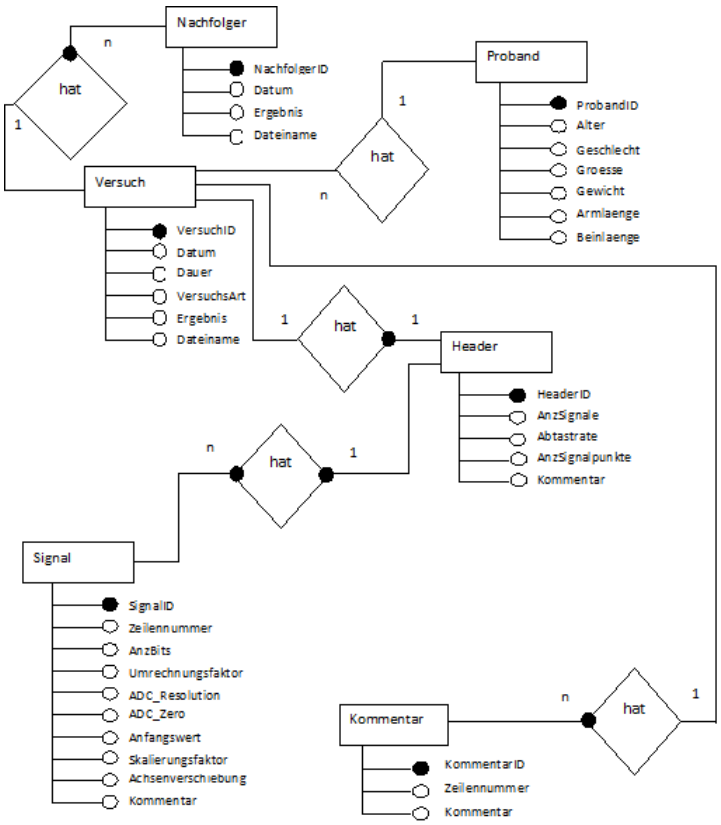

Abbildung 5.1: Entwickeltes ER-Modell

Bei den Entitätstypen *Versuch, Nachfolger, Proband, Header, Signal* und *Kommentar* sind

jeweils die IDs eindeutig identifizierend.

In Tabelle 5.2 ist das Data-Dictionary, mit Datentypen zu den Attributen, zu sehen. Pflichtfelder sind mit *not null* sowie optionale Felder mit *null* gekennzeichnet.

Versuch			
	VersuchID:	integer not null	
	Datum:	date not null	
	Dauer:	time not null	
	Versuchsart:	varchar (255) not null	
	Ergebnis:	blob not null	
	Dateiname	varchar(255) not null	
Nachfolger			(Versuch)**Hat**(Nachfolger)
	NachfolgerID:	integer not null	Ein Versuch kann mehrere
	Datum:	date not null	Nachfolger haben. Ein Nachfolger
	Ergebnis:	blob not null	hat immer genau einen Vorgänger.
	Dateiname	varchar (255) not null	
Proband			
	ProbandID:	integer not null	(Versuch)**Hat**(Proband)
	Alter:	integer null	Ein Versuch kann genau
	Geschlecht:	varchar (15) null	einen Probanden haben.
	Groesse:	integer null	Ein Proband kann an mehreren
	Armlaenge:	integer null	Versuchen teilnehmen
	Beinlaenge:	integer null	
Header			(Header)**Hat**(Versuch)
	HeaderID:	integer not null	Ein Header hat genau
	AnzSignale:	integer null	einen Versuch. Ein Versuch
	Abtastrate:	integer null	kann genau einen Header haben.
	AnzSignalpunkte:	integer null	
	Kommentar:	varchar (255) null	
Signal			
	SignalID:	integer not null	(Signal)**Hat**(Header)
	Zeilennummer:	integer null	Ein Signal hat genau
	AnzBits:	integer null	einen Header. Ein Header
	Umrechnungsfaktor:	integer null	kann mehrere Signale haben.
	ADC_Resolution:	integer null	
	ADC_Zero:	integer null	
	Anfangswert:	integer null	
	Skalierungsfaktor:	integer null	
	Achsenverschiebung:	integer null	
	Kommentar:	varchar (255) null	
Kommentar			
	KommentarID:	integer not null	(Kommentar)**Hat**(Versuch)
	Zeilennummer:	integer null	Ein Kommentar hat genau
	Kommentar:	varchar (255) null	einen Versuch.Ein Versuch
			kann mehrere Kommentare haben.

Tabelle 5.2: Data-Dictionary zum ER-Modell

Folgende Datentypen wurden verwendet:

- integer: Entspricht einer ganzen Zahl
- varchar (n): Zeichenkette von variabler Länge n
- date: Datum ohne Zeitangabe
- time: Zeitangabe
- blob: Binärdatei

5.3 Wahl des Datenbankmanagementsystems

Bei der Evaluation einer geeigneten Datenbanklösung werden die relationalen DBMS MySQL[25], Oracle[26] 11g XE und Oracle 11g Release 2 als Enterprise Edition betrachtet.

Einige Unterschiede, die zur Wahl dieser Datenbanken herangezogen wurden, sind in Tabelle 5.3 aufgeführt.

Bezeichnung	**MySQL**	**Oracle XE**	**Oracle EP**	**HBase**	**MongoDB**
Preis	Frei	Frei	~500 Euro pro Nutzer	Frei	Frei
Auf unterschiedlichen Architekturen einsetzbar	ja	bedingt	ja	bedingt	ja
Schema/DB-Typ	relational	relational	relational	Column Store	Document Store
Speicherbeschränkung	nein	ja	bedingt	nein	nein
Einfaches Sichern der Datenbank (Backup)	ja	ja	ja	ja	ja
verteilte Systeme (Cluster)	nein	nein	nein	ja	ja
Datenbankabfragen	SQL	SQL	SQL	Anwendung	Anwendung
Trigger	ja	ja	ja	nein	nein
Anforderung: 1-3, 12-13 15-16, 19, 21, 23, 25	ja	ja	ja	ja	ja
Anforderung: 9	ja	ja	ja	bedingt	bedingt
Anforderung: 14	ja	ja	nein	ja	ja
Anforderung: 20, 31	ja	ja	ja	ja	bedingt
Anforderung: 4-8, 10-11 17, 18, 22, 24, 26-30	ja	ja	ja	bedingt	bedingt

Tabelle 5.3: Einige Unterschiede einzelner Datenbanken

[25] http://www.mysql.de
[26] http://www.oracle.com

Die in der Tabelle 5.3 aufgeführten relationalen Datenbanken wurden ausgewählt, da sie die gestellten Anforderungen erfüllen, weit verbreitet sind und bis auf Oracle 11g XE bereits im Fachbereich zum Einsatz kommen. Die Möglichkeit der kostenlosen Nutzung der Datenbanken MySQL und Oracle 11g XE war ein weiteres entscheidendes Kriterium.

Bei den NoSQL-Datenbanksystemen ist Googls BigTable[27] durch seine Versionsverwaltung, die mithilfe von Timestamps zu den Datensätzen umgesetzt wird, ein guter Kandidat. Mit dieser Versionsverwaltung kann die Anforderung, ausgewertete Datensätze immer im Bezug zu den eigentlichen Rohdaten zu speichern (Anforderung 20 in der Tabelle 5.1 auf Seite 46), sehr gut umgesetzt werden. Da es sich hier jedoch um eine proprietäre Technologie von Google handelt, die Dritten nicht zur Verfügung gestellt wird, wird in dieser Arbeit das Datenbanksystem HBase[28], welches als Clone von BigTable gilt, und ein Open Source Projekt ist, betrachtet. Zusätzlich soll geprüft werden, ob alle gestellten Anforderungen mit der Datenbank MongoDB[29] umgesetzt werden können. Durch die in Kapitel 4.2.4 bei den CRUD-Operationen vorgestellte GridFS-API können bei dieser Datenbank unterschiedliche Dateiformate in die Datenbank eingefügt und wieder ausgelesen werden.

Als Testumgebung wird für jedes Datenbanksystem eine VirtualBox[30] der Firma Oracle eingerichtet. Alle Systeme haben eine Systemfestplatte von 20 Gigabyte und eine Festplatte für das Datenbanksystem von 80 Gigabyte sowie 2 Gigabyte Arbeitsspeicher. Für MySQL und Oracle 11g Release 2 Enterprise Edition wurde als Betriebssystem Windows 7 in der 64 Bit Version gewählt. Für Oracle 11g XE ist nur eine 32 Bit Version verfügbar, aus diesem Grund wurde hier ein Windows 7 als 32 Bit System gewählt. Für HBase wurde ein Ubuntu 10.04 in einer 64 Bit Version gewählt, da es für ein Linux System entwickelt wurde und nur mit Hilfe von Cygwin[31] überhaupt unter Windows installiert werden kann. Außerdem war es nötig, für HBase ein Java Paket der Firma Sun in der Version 1.6.26 zu installieren. Die Datenbank MongoDB konnte unter dem Betriebssystem Windows 7 in der 64 Bit Version installiert werden. Auch bei diesem System wurde das Java Paket der Firma Sun in der Version 1.6.26 installiert.

Für alle Virtuellen Maschinen (VirtualBox) galten zu jeder Zeit die gleichen Rahmenbedingungen. Sie liefen nacheinander auf dem selben physischen System mit immer den gleichen physischen Ressourcen.

[27] http://research.google.com/archive/bigtable.html
[28] http://wiki.apache.org/hadoop/Hbase/Stargate
[29] http://www.mongodb.org/
[30] Ein System um virtuell Hardware zu Simulieren und einem Gastsystem zur Verfügung zu stellen
[31] http://www.cygwin.com

5.4 Logischer Entwurf

In diesem Abschnitt wird das entwickelte konzeptionelle Modell in einem Datenmodell der gewählten Datenbanksysteme abgebildet.

5.4.1 Relationale-Datenbankmanagementsysteme

Aus dem entwickelten konzeptionellen Entwurf ergibt sich das relationale Schema, welches in Abbildung 5.2 zu sehen ist.

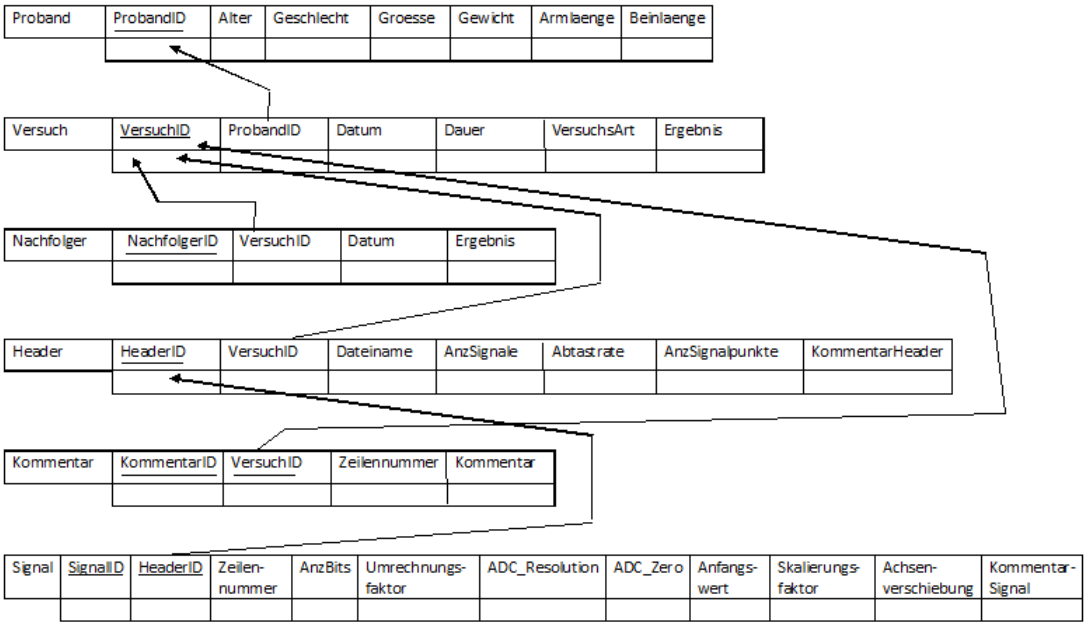

Abbildung 5.2: Relationales Schema der Datenbank

Die Rohdaten in der Tabelle Versuch müssen durch einen Trigger vor dem Ändern oder Löschen geschützt werden. Da die Trigger bei den jeweiligen DBMS eine unterschiedlich Syntax aufweisen, werden sie direkt bei der Implementierung der jeweiligen Datenbanken aufgeführt. (vgl. Kapitel 3.2.9)

5.4.2 NoSQL-Datenbankmanagementsysteme

Für die Datenbank HBase kann ebenfalls der entwickelte konzeptionelle Entwurf verwendet werden. Das resultierende Datenmodell ist in Tabelle 5.4 zu sehen.

Tabelle:Proband			
RowKey:	ProbandID		
Family	data:	Columns:	Alter,Geschlecht,Groesse,Gewicht, Armlaenge,Beinlaenge

Tabelle:Versuch			
RowKey:	VersuchID		
Family	data:	Columns:	Timestamp,Dauer,VersuchsArt,Dateiname
	content:	Columns:	raw

Tabelle:Proband-Versuch			
RowKey:	ProbandID\x00VersuchID		
Family	data:	Columns:	Timestamp

Tabelle:Header			
RowKey:	HeaderID		
Family	data:	Columns:	AnzSignale,Abtastrate,AnzSignalpunkte, Kommentar

Tabelle:Versuch-Header			
RowKey:	VersuchID\x00HeaderID		
Family	data:	Columns:	Timestamp

Tabelle:Signal			
RowKey:	SignalID		
Family	data:	Columns:	Zeilennummer,AnzBits,Umrechnungsfaktor, ADC_Resolution,ADC_Zero,Anfangswert, Skalierungsfaktor,Achsenverschiebung, Kommentar

Tabelle:Header-Signal			
RowKey:	HeaderID\x00SignalID		
Family	data:	Columns:	Timestamp

Tabelle:Kommentar			
RowKey:	KommentarID		
Family	data:	Columns:	Zeilennummer,Kommentar

Tabelle:Versuch-Kommentar			
RowKey:	VersuchID\x00KommentarID		
Family	data:	Columns:	Timestamp

Tabelle 5.4: Datenmodell für die Datenbank HBase

Da es in diesem System das Konzept von Fremdschlüsseln nicht gibt, muss zu jeder Beziehung, die dargestellt werden soll, eine zusätzliche Tabelle erstellt werden. Diese Tabellen verknüpfen die beiden originalen Tabellen und stellen so eine Beziehung zwischen ihnen her.

Die Datenbank MongoDB ist an sich schemafrei, es kann aber dennoch ein ähnliches Schema wie bei den relationalen Datenbanken verwendet werden. In diesen Datenbanken gibt es Collections, die den Tabellen eines relationalen DBMS entsprechen. Die Collections enthalten Dokumente, die den Zeilen in einem relationalen DBMS entsprechen. Das Konzept der Fremdschlüssel wird entweder über eine manuelle Referenz, was dem Prinzip der Fremdschlüssel

bei relativen Datenbanken am ehesten entspricht und speicherplatzschonender umgesetzt werden kann, oder mittels der Referenzierung über den Datentyp DBRef implementiert. Die Verknüpfung bei DBRef wird durch die Kombination aus eindeutiger ID (ObjectID) und dem Namen der Collection umgesetzt. Es ist also eine Struktur, die bei diesem Verfahren abgespeichert werden muss. Da die Struktur ähnlich einer relationalen Datenbank ist, kann das Schema aus Abbildung 5.2 verwendet werden.

Der Schutz der Rohdaten ist bei diesen Systemen nur durch eine entsprechende Implementierung in der Anwendung möglich. Hier muss explizit verhindert werden, dass die eingefügten Rohdatensätze verändert oder gelöscht werden können.

5.5 Evaluation eines geeigneten relationalen Datenbankmanagementsystems

Als erste Datenbank wird MySQL 5.5.16, gefolgt von Oracle 11g XE geprüft, da diese keine zusätzlichen Kosten verursachen.

Zu Beginn muss ein Nutzer in der Datenbank angelegt werden. Alle Interaktionen mit der Datenbank wurden so weit möglich in SQL, welches in Kapitel 3.2.11 vorgestellt wurde, umgesetzt.

Anschließend konnten die erarbeiteten Tabellen mit der Standard-Storage-Engine *InnoDB* unter dem erstellten Nutzer eingefügt werden. Die Storage-Engine InnoDB gewährleistet im Gegensatz zu anderen Storage-Engines von MySQL die Transaktionssicherheit und die referenzielle Integrität über Fremdschlüssel. Besonders die referentielle Integrität über Fremdschlüssel, welche im Kapitel 3.2.8 besprochen wurde, zu gewährleisten, war ein wichtiger Aspekt. Die Absicherung der Rohdaten wurde mit einem Trigger umgesetzt. Dieser erlaubt lediglich das Einfügen, jedoch nicht das Ändern oder Löschen eines Versuches. Da es in MySQL keine Fehlererkennung gibt, muss das in irgendeiner Form nachgebildet werden. Dies wird umgesetzt, indem der Trigger bei einem Update auf die Tabelle Versuch in eine nicht existierende Tabelle etwas eintragen soll. Dadurch wird ein Fehler ausgelöst und die gesamte Aktion nicht ausgeführt. Der Trigger ist wie folgt umgesetzt:

```
CREATE TRIGGER VersuchNichtAenderbar
BEFORE UPDATE OR DELETE
ON Versuch
FOR EACH ROW
BEGIN
```

```
INSERT INTO Versuch_Nicht_Aenderbar
VALUES ('Versuch kann nicht nachtraeglich geaendert werden!');
END;
```

Zu Testzwecken wurden bis zu 20 Datensätze in die Tabellen eingefügt. Da die *insert* Befehle bei den unterschiedlichen Tabellen gleich aufgebaut sind, soll hier stellvertretend nur einer gezeigt werden:

```
INSET INTO Proband (AlterP,Geschlecht,Groesse,Gewicht,Armlaenge,Beinlaenge)
VALUES (30,'Maennlich',178,80,100,110);
```

Das Auslesen eines Versuches aus der Datenbank und anschließende Speichern im Dateisystem wurde mit der Prozedur aus Listing 7.1 gelöst.

Hierbei stellte sich heraus, dass das Speichern und Auslesen eines Blobs allein mit SQL-Mitteln nicht so einfach möglich ist. MySQL speichert lediglich die ersten 64 Kilobyte der Daten und schneidet den Rest ab. Die Lösung dieses Problems ist im Benutzerhandbuch von MySQL zu finden. (vgl. [MyS12]) Hierzu muss die Dateigröße in der Datenbank mit abgelegt werden, da sie zum späteren Auslesen der Daten erforderlich ist. Das bedeutet, die Tabelle Versuch und Nachfolger muss für die MySQL-Datenbank wie folgt angepasst werden:

Versuch	(VersuchID,ProbandID,Datum,Dauer,VersuchsArt,Dateigroesse,Ergebnis, Dateiname)

Tabelle 5.5: Überarbeitete Tabelle für den Versuch

Nachfolger	(NachfolgerID,VersuchID,Datum,Dateigroesse,Ergebnis, Dateiname)

Tabelle 5.6: Überarbeitete Tabelle für den Nachfolger eines Versuchs

Das Speichern und Auslesen wird mit der Programmiersprache C-Sharp umgesetzt, da dies eine vorgeschlagene Lösung im MySQL-Benutzerhandbuch ist. Der entsprechende Quelltextes ist im Anhang in Listing 7.2 abgebildet.

Die Binärdaten, die zum Testen verwendet wurden, wurden mit dem Tool Dummy File Creator[32] von Nikko Cheng in nicht komprimierbarer Größe von einem Megabyte bis 30 Gigabyte erzeugt und in die Datenbank eingefügt. Im Zuge der Optimierung einiger Datenbanksysteme, wie z.B. Oracle werden große Binärdaten komprimiert eingefügt. Diese Art der Optimierung wird nicht von allen Datenbanksystemen angewendet. Um jedoch unter

[32] http://www.mynikko.com

gleichen Bedingungen ein Datenbanksystem auszuwählen, wurden nicht komprimierbare Testdaten erzeugt.

Hierbei musste festgestellt werden, dass die Grenze der einzufügenden Dateien bei 252 Megabyte lag. Bei dem Versuch, eine größere Binärdatei einzufügen, brach das System mit einem Speicherzugriffsfehler ab. Da jedoch Testdaten in einer Größe von mindestens 14 Gigabyte eingefügt werden müssen, kann eine MySQL-Datenbank die Ergebnisse in diesen Größen nicht direkt speichern.

Hier bietet sich aber die Möglichkeit, diese großen Daten nicht direkt zu speichern, sondern sie im dafür ausgelegten Dateisystem zu belassen und lediglich den Pfad zu diesen Daten abzuspeichern. Dies ist deutlich schneller und auch die allgemeine Praxis bei solch großen Dateien. Hier ist es denkbar, den Pfad relativ in der Datenbank zu speichern und eine dazugehörige Konfigurationsdatei anzulegen. Bei einer Änderung des Pfades müsste dann lediglich die Konfigurationsdatei angepasst werden. Durch die Möglichkeit, Sicherheitsrichtlinien im Dateisystem festzulegen, können somit die gespeicherten Rohdaten wie gefordert geschützt werden. Ein Ändern oder Löschen ist dann nicht mehr möglich, womit Anforderung 18 erfüllt ist.

Das Schema muss ebenfalls nicht verändert werden. Anstelle des Ergebnisses (Blob) muss einfach nur der Pfad zu den Daten, zum Beispiel mit dem Datentyp *varchar()* gespeichert werden. Der Konverter liest zurzeit die Versuchsdaten aus dem Dateisystem aus und erzeugt daraus die Headerdatei. Dies müsste bei dieser Vorgehensweise nicht verändert werden. Er muss lediglich zusätzlich den Dateipfad zu den Versuchsdaten in der Datenbank speichern. So ist eine übersichtliche und strukturierte Verwaltung der im Dateisystem liegenden Daten möglich.

Als zweites mögliches Datenbanksystem wurde Oracle 11g XE betrachtet. Wie bei MySQL muss erst ein Nutzer mit entsprechendem Tablespace erstellt werden. Bei der Erstellung des Tablespace tritt bereits das erste Problem auf. Bei Oracle 11g XE ist es nicht möglich, einen Tablespace, der größer als elf Gigabyte ist, zu erstellen bzw. alle entstehenden Versuchsdaten direkt abzuspeichern. Außerdem können Dateien nur bis zu einer maximalen Größe von 4 Gigabyte verarbeitet werden. Genau wie bei MySQL besteht auch hier die Möglichkeit, die Daten im Dateisystem zu belassen und nur den Pfad zu diesen Dateien in der Datenbank zu speichern.

Als letzte relationale Datenbank wird Oracle 11g Release 2 in der Enterprise Ausführung untersucht. Wie bei der kostenfreien Version von Oracle muss ebenfalls bei der Enterprise Edition ein Nutzer mit entsprechendem Tablespace angelegt werden. Wie im Listing 7.3 im

Anhang zu sehen, ist dies bei Oracle etwas anders als bei MySQL.

Hier liegt die Grenze bei ca. 31,99 Gigabyte (die obere Grenze wurde genau auf 2^{32}Blöcke á 8 Kilobyte festgelegt, diese Blockgröße kann jedoch angepasst werden). Das reicht, um das aufkommende Datenvolumen zu speichern. Im nächsten Schritt können die Tabellen und Trigger erstellt werden. Es gibt auch hier einen Trigger, der für das Schützen der Rohdaten verantwortlich ist.

```
CREATE OR REPLACE TRIGGER VersuchNichtAenderbar
BEFORE UPDATE OR DELETE
ON Versuch
BEGIN
RAISE_APPLICATION_ERROR (-20202,'Versuch kann nicht nachtraeglich
geaendert werden!');
END;
/
```

Wie man sehen kann, gibt es in Oracle eine Fehlerbehandlung, die für diese Zwecke verwendet werden kann.

Im Gegensatz zu MySQL, welches den Primärschlüssel bei jedem neu eingefügten Datensatz durch die Funktion *auto_increment* automatisch inkrementieren[33] kann, ist es bei Oracle erforderlich, diese Funktionalität mit Hilfe von Sequenzen und Triggern nachzubilden, was am Beispiel der Tabelle Versuch nachfolgend gezeigt werden soll.

```
CREATE SEQUENCE zaehlerTabelleVersuch;
CREATE OR REPLACE TRIGGER AutoIncrementVersuch
BEFORE INSERT
ON Versuch
FOR EACH ROW
BEGIN
IF (:NEW.VERSUCHID IS NULL) THEN
SELECT zaehlerTabelleVersuch.NEXTVAL INTO :NEW.VersuchID FROM DUAL;
END IF;
END;
/
```

Im Anschluss wurde das System wieder mit den Testdaten befüllt. Ein beispielhafter *insert* Befehl ist nachfolgend dargestellt.

[33]Beim inkrementieren wird der Wert der Variablen in jedem Schritt um eins erhöht.

```
INSERT INTO Proband (ProbandID,AlterP,Geschlecht,Groesse,Gewicht,
Armlaenge,Beinlaenge) VALUES (NULL,30,'Maennlich',178,80,100,110);
```

Vor dem Einfügen eines Datensatzes in die Tabelle Versuch ist zu beachten, dass vom Datenbankadministrator ein Verzeichnis in der Datenbank definiert und dem Nutzer ein Lese- und Schreibrecht in diesem erteilt werden muss.

```
CREATE OR REPLACE DIRECTORY BLOB AS 'E:\';
GRANT read,write ON DIRECTORY BLOB TO weidner;
```

Hier müssen die Rohdaten gespeichert werden, bevor sie in die Datenbank verschoben werden können. Die Prozedur zum Einfügen eines Blobs in die Datenbank ist im Anhang in Listing 7.4 , das Auslesen des Blobs und Speichern im Dateisystem in Listing 7.5 zu sehen.

Die Prozedur zum Auslesen des Blobs kann nach dem Erstellen im Datenbanksystem mittels des Aufrufs

```
EXECUTE selectBlob (VersuchID);
```

gestartet werden. Die Prozedur benötigt als Parameter die VersuchID, um die richtigen Versuchsdaten auszuwählen und diese im Dateisystem abspeichern zu können. Wie in der Prozedur zu sehen, ist es bei Oracle nicht nötig, die Dateigröße zu speichern, da das Datenbanksystem diese bereits im System abgespeichert hat und so jederzeit auf die Größe aller abgespeicherten Dateien zugegriffen werden kann.

Bei den durchgeführten Tests wurden Binärdaten in der Größenordnung von einem Megabyte bis 30 Gigabyte problemlos in der Datenbank gespeichert und auch wieder ausgelesen. Näheres zu den einzelnen Tests wird in Kapitel 5.8 beschrieben.

Oracle 11g Release 2 in der Enterprise Edition ist die einzige Datenbank, welche von den drei getesteten Systemen in der Lage ist, die Ergebnisse direkt in der Datenbank zu speichern. Jedoch ist es aufgrund der sehr hohen Kosten dieses Systems eher zu empfehlen, eine kostenlose Datenbank für diese Aufgabe einzusetzen und die Versuchsergebnisse nicht direkt, sondern lediglich den Verweis auf das Dateisystem in der Datenbank zu speichern.

5.6 Evaluation eines geeigneten NoSQL-Datenbankmanagementsystems

Als erste Datenbank wird HBase betrachtet. Hier ist es nicht nötig wie bei den relationalen Datenbanken einen Nutzer oder einen Tablespace anzulegen. Ist die Datenbank gestartet, so

können die Tabellen, wie bei den Grundlagen in Kapitel 4.2.3 gezeigt, über eine unterstützte Programmiersprache erstellt werden. Wie im Anhang in Listing 7.9 zu sehen, wurden die Tabellen mit Hilfe der Java-API erstellt. Ein beispielhafter *insert* ist im Anhang in Listing 7.12 zu sehen. Die Testdaten wurden mit dem im Anhang befindlichen Quellcode aus Listing 7.10 in die Datenbank geschrieben und mit Listing 7.11 wieder ausgelesen. In HBase gibt es keine unterschiedlichen Datentypen. Alle Daten in HBase werden in einem Bytearray gespeichert. Dadurch muss jede Datei erst aufwendig in Java eingelesen, in ein Bytearray umgewandelt und kann dann erst in die Datenbank geschrieben werden. Dieses Verfahren ist sehr umständlich und dauert bei großen Datenmengen extrem lange. Der Versuch, eine Testdatei mit einer Größe von 250 Megabyte in die Datenbank einzufügen, wurde nach 68 Minuten abgebrochen. Das Problem liegt an der aufwendigen Umwandlung der Daten in Java. Laut HBase Dokumentation[34] werden große Dateien direkt im Hadoop-Dateisystem (HDFS)[35] gespeichert. Ein Fragment aus dem Quelltext ist in Listing 5.1 zu sehen. Durch diese Vorgehensweise wird diese Umwandlung der Daten umgangen.

Listing 5.1: HBase Lösung zum Speichern großer Datenmengen

```
//upload bigFile by hadoop directly
HBaseBigFile blob = new HBaseBigFile ();
File file = new File ("/usr/lib/hbase/bin/test5.dat");
FileInputStream stream = new FileInputStream (file);
Path rootPath = new Path ("/data/");
String filename = "test5.dat";
blob.uploadFile (rootPath, filename, stream, true);

//receive file stream from hadoop
Path p = new Path (rootPath, filename);
InputStream is = blob.path2Stream (p, 4096);
```

Die großen Versuchsdaten direkt in der Datenbank zu speichern, ist auch hier nicht möglich. Eine Alternative ist das Speichern dieser Daten im Hadoop-Dateisystem, welches bei der Einrichtung jedoch sehr komplex ist.

Nach der Implementierung des Hadoop-Dateisystems muss die Datenbank HBase installiert und für dieses Dateisystem konfiguriert werden. Ein weiterer negativer Aspekt ist die Tatsache, dass es nicht so einfach möglich ist, komplexe Suchanfragen zu stellen. Dies und auch das geforderte Schützen der Rohdaten kann nur über die Anwendungsebene erreicht werden. Um die benötigten Beziehungen unter den Tabellen herzustellen, mussten neue Tabellen erzeugt werden, die jeweils zwei andere Tabellen miteinander verknüpften. Dieses Verfahren ist sehr kompliziert und man kann leicht den Überblick verlieren. Jegliche Logik,

[34] http://wiki.apache.org/hadoop/Hbase

[35] Hadoop Distributed File System ist ein leistungsfähiges Dateisystem zur Speicherung sehr großer Datenmengen

um diese Beziehungen zu nutzen, muss ebenfalls erst in der Anwendungsebene implementiert werden.

Die Schlussfolgerung daraus ist, dass es nur mit erheblichem Mehraufwand möglich ist, alle gestellten Anforderungen zu erfüllen. Für die Lösung der Aufgabenstellung ist die Datenbank HBase aus diesen Gründen schlecht geeignet.

Als zweites System aus der NoSQL-Welt wird die Datenbank MongoDB, die bereits im Kapitel 4.2.4 vorgestellt wurde, untersucht. Bei dieser Datenbank ist es ebenfalls nicht nötig, einen Nutzer, einen Tablespace oder ein Schema anzulegen. Allerdings muss hier eine Datenbank und die Collections angelegt werden, die vorher aber nicht explizit definiert werden müssen. Sie werden zur Laufzeit beim ersten Einfügen eines Dokuments automatisch erzeugt. Ein Beispielquelltext für das Erstellen einer Datenbank, Collection und das Einfügen eines Datensatzes in die Collection ist in Listing 7.15 im Anhang zu sehen.

In Listing 7.16 im Anhang ist beschrieben, wie in MongoDB über die GridFS-API große Dateien in die Datenbank transferiert werden können. Eine genaue Analyse des zeitlichen Aufwands wird im nächsten Abschnitt erläutert.

Negativ bei MongoDB, ebenfalls bei HBase, ist die Auslagerung der komplexen Logik in die Anwendungsschicht. Wenn umfangreiche Suchanfragen gestellt werden müssen, so müssen diese erst aufwendig implementiert werden. Auch das Schützen der Rohdaten muss über eine eigens entwickelte Implementierung in der Anwendung umgesetzt werden. Das Darstellen von Fremdschlüsseln ist bei diesem System besser umgesetzt als bei HBase, bedarf aber erheblicher Einarbeitungszeit. Die Dokumentation[36] zu MongoDB ist ziemlich umfangreich, bietet aber in speziellen Fällen, wie z.B. bei den Fremdschlüsseln, nur sehr kurze Quelltextfragmente, welche auch nicht näher erklärt werden.

5.7 Benchmark-Test der evaluierten Datenbanken

In diesem Abschnitt sollen die gemessenen Zeiten, die ausschlaggebend für die Wahl des Datenbanksystems waren, für die ausgewählten Datenbanksysteme erläutert werden. Dazu zählt die Zeit, die benötigt wurde, um Datenbankoperationen, wie zum Beispiel das Einfügen oder Suchen von bestimmten Datensätzen, durchzuführen.

Oracle 11g Release 2 Enterprise Edition Bei dem relationalen Datenbanksystem Oracle 11g R2 kann die Datenbanksprache SQL genutzt werden, um gezielte Anfragen an die

[36] http://www.mongodb.org/display/DOCS/Home

Datenbank zu stellen. So ist es z.B. möglich, sich zu einem bestimmten Versuch den Probanden mit allen über ihn gespeicherten Daten ausgeben zu lassen oder zu allen Versuchen mit der Versuchsart Bewegung die Beinlängen von den Probanden aufzulisten. Das SQL-Statement sieht folgendermaßen aus:

```
SELECT p.Beinlaenge
FROM Versuch v, Proband p
WHERE v.VersuchsArt = 'Bewegung' AND v.ProbandID = p.ProbandID;
```

Alle durchgeführten Anfragen an das Datenbanksystem benötigten aufgrund der sehr geringen Anzahl an Datensätzen im Testsystem weniger als eine Sekunde für die Ausführung der Kommandos.

Die meiste Zeit wurde benötigt, um entweder die Versuchsdaten in die Datenbank zu schreiben oder sie aus der Datenbank auszulesen und wieder im Dateisystem zu speichern. Eine Übersicht über die Größe der Dateien und die benötigte Zeit für das Einfügen oder Auslesen der Testdaten ist in Tabelle 5.7 abgebildet.

Bei diesem Datenbanksystem ist das Selektieren von Informationen aufgrund seiner nicht komplexen Struktur sehr schnell. Das Einfügen von großen Daten in eine Datenbank ist jedoch nicht besonders sinnvoll. Schon beim Einfügen einer vier Gigabyte großen Datei wird die Zeit, die beim Einfügen einer 14 Gigabyte großen Datei ins Dateisystem benötigt wird, um mehr als das Doppelte überschritten. Das Speichern dieser großen Daten direkt in der Datenbank ist demnach keine praktikable Lösung.

MongoDB Wie schon mehrfach erwähnt, müssen bei der Datenbank MongoDB komplexere Anfragen aufwendig implementiert werden. Ein recht einfaches Beispiel, bei dem die Bewegungsversuche ausgegeben werden, ist in Listing 5.2 dargestellt.

Listing 5.2: Beispiel für eine in Java implementierte Anfrage an die Datenbank

```
BasicDBObject query = new BasicDBObject ();
query.append ("VersuchsArt", "Bewegung");
DBCursor cursor = collection.find (query);
while (cursor.hasNext () )
        System.out.println (cursor.next () );
```

Wie in Tabelle 5.7 zu sehen, ist es ebenfalls in der Datenbank MongoDB nicht möglich, alle aufkommenden Versuchsdaten direkt in der Datenbank zu speichern. Das Einfügen der 14 Gigabyte großen Binärdatei wurde nach 24 Minuten mit einer Fehlermeldung abgebrochen. Um derart große Daten einzufügen, müsste der Arbeitsspeicher enorm vergrößert werden. Das Zuordnen der eingefügten Datei erweist sich als schwierig. Durch die GridFS-API

werden große Dateien automatisch in zwei Collections gespeichert. Die Metadaten, wie z.B. Dateiname, Spezifikationen des Dateityps u.a. werden in der Collection *files* und die eigentliche Datei wird in der Collection *chunks*, aufgeteilt in jeweils 1,99 Gigabyte große Fragmente, gespeichert. Dadurch muss in der Anwendungsschicht eine zusätzliche Verknüpfung mit den anderen gespeicherten Datensätzen geschaffen werden.

Größe	Oracle 11g R2		MongoDB	
	Einfügen in DB	Auslesen aus DB	Einfügen in DB	Auslesen aus DB
1 MB	00:00:00,91 Std	00:00:00,47 Std	00:00:01,08 Std	00:00:01,11 Std
5 MB	00:00:01,29 Std	00:00:00,69 Std	00:00:01,55 Std	00:00:01,33 Std
15 MB	00:00:02,69 Std	00:00:01,23 Std	00:00:02,05 Std	00:00:01,61 Std
50 MB	00:00:07,53 Std	00:00:03,09 Std	00:00:02,86 Std	00:00:02,90 Std
100 MB	00:00:22,11 Std	00:00:06,91 Std	00:00:03,71 Std	00:00:05,02 Std
250 MB	00:00:54,95 Std	00:00:18,02 Std	00:00:07,45 Std	00:00:15,87 Std
500 MB	00:01:53,18 Std	00:00:32,11 Std	00:01:24,44 Std	00:00:36,62 Std
1 GB	00:03:38,07 Std	00:01:30,11 Std	00:03:07,77 Std	00:01:27,31 Std
4 GB	00:11:44,52 Std	00:07:17,14 Std	00:15:18,72 Std	00:07:23,57 Std
14 GB	00:50:24,38 Std	00:25:15,81 Std	-	-
30GB	01:40:43,38 Std	00:46:33,67 Std	-	-

Tabelle 5.7: Übersicht über die benötigte Zeit beim Einfügen und Auslesen von Daten

5.8 Benchmark-Test einer Alternativlösung

Da sich herausgestellt hat, dass das direkte Speichern großer Versuchsdaten in einer Datenbank sehr zeitaufwendig, nur mit der teuren Datenbank Oracle 11g R2 Enterprise Edition möglich und aus diesem Grund nicht vertretbar ist, wird das Speichern des Dateipfades in der Datenbank als weitere Möglichkeit getestet.

Für diesen Test wird die Datenbank MySQL mit der in Kapitel 5.6 ausgearbeiteten Struktur umgesetzt. Im Data-Dictionary der Tabellen Versuch und Nachfolger muss lediglich das Attribut Ergebnis und Dateigröße gelöscht werden.

Versuch		
	VersuchID:	integer not null
	Datum:	date not null
	Dauer:	time not null
	Versuchsart:	varchar (255) not null
	Dateiname:	varchar (255) not null
Nachfolger		
	NachfolgerID:	integer not null
	Datum:	date not null
	Dateiname:	varchar (255) not null

Tabelle 5.8: Überarbeitung des Data-Dictionary

Weitere Änderungen sind nicht nötig. Die schon implementierte MySQL-Datenbank kann für diesen Test verwendet werden. Es muss lediglich die Tabelle Versuch und Nachfolger angepasst werden. Mit folgenden SQL-Befehlen wurde dies umgesetzt:

```
ALTER TABLE Versuch Drop Column Ergebnis;
ALTER TABLE Nachfolger Drop Column Ergebnis;
```

Zusätzlich wird die Dateigröße und der Trigger zum Schutz der Rohdaten nicht mehr benötigt. Bei dieser Lösung verbleiben die Versuchsergebnisse im Dateisystem und müssen über die Sicherheitseinstellungen des Dateisystems vor dem Ändern oder Löschen geschützt werden. Die Dateigröße und der Trigger können mit den SQL-Befehlen:

```
ALTER TABLE Versuch DROP COLUMN Dateigroesse;
ALTER TABLE Nachfolger DROP COLUMN Dateigroesse;
DROP TRIGGER VersuchNichtAenderbar;
```

gelöscht werden.

Im Anschluss werden Testdaten in die Datenbank eingefügt und die benötigte Zeit für das Einfügen wie auch für das Selektieren von Zeilen gemessen.

MySQL	
Operation	**Benötigte Zeit**
Einfügen von 50 Datensätzen pro Tabelle	00:00:05,27 Std
Einfügen von 100 Datensätzen pro Tabelle	00:00:09,71 Std
Einfügen von 150 Datensätzen pro Tabelle	00:00:11,27 Std
Einfügen von 200 Datensätzen pro Tabelle	00:00:18,69 Std
Beinlänge von allen Probanden mit Veruschsart Bewegung ausgeben. Ergebnis: 96 Zeilen	00:00:00,31 Std
Armlänge von allen Probanden mit Versuchsart Heben ausgeben. Ergebnis: 39 Zeilen	00:00:00,13 Std
Alter von allen Probanden mit Veruschsart Eye Tracking ausgeben. Ergebnis: 34 Zeilen	00:00:00,11 Std
Kommentare von allen Versuchen mit der Versuchsart EKG ausgeben. Ergebnis: 32 Zeilen	00:00:00,07 Std

Tabelle 5.9: Übersicht über die benötigte Zeit beim Einfügen und Auslesen von Daten in MySQL

Wie in Tabelle 5.9 zu sehen, werden selbst für Einfügeoperationen von mehr als 2000 Datensätzen (200 pro Tabelle) nur knapp 19 Sekunden benötigt. Auch für das Selektieren und Ausgeben von Datensätzen wird nicht einmal eine Sekunde benötigt.

Diese Lösungsmöglichkeit wird in Anbetracht der gemessenen Zeiten für Datenbankoperationen sowie das Erfüllen aller Anforderungen als effektivste Lösung angesehen.

5.9 Zusammenfassung

In diesem Kapitel wurde nach Herausstellung aller an die Datenbank gestellten Anforderungen durch den späteren Nutzer ein konzeptioneller Entwurf mittels ER-Modell erstellt. Im Anschluss konnte ein Datenbankmanagementsystem ausgewählt werden. Nachdem die Datenbanken Oracle 11g R2 Enterprise Edition, Oracle 11g XE, MySQL, HBase und MongoDB ausgewählt wurden, konnte das Datenmodell für die jeweiligen Datenbanken im logischen Entwurf entwickelt werden. Im Anschluss konnte evaluiert werden, welche von den ausgewählten Datenbanken allen Anforderungen entspricht. Im letzten Abschnitt dieses Kapitels, im Benchmark-Test, wurde geprüft, wie lange eine ausgewählte relationale Datenbank und eine NoSQL-Datenbank für Standardoperationen, wie das Suchen von Datensätzen oder das Einfügen und Auslesen von großen Dateien, benötigen. Augenmerk lag hier besonders auf dem Einfügen und Auslesen großer Dateien.

Es hat sich herausgestellt, dass nur die Datenbank Oracle 11g R2 Enterprise Edition in der Lage war, alle Versuchsdaten direkt in der Datenbank zu speichern. Wie sich aber beim

Benchmark-Test herausstellte, ist das Speichern aller Daten direkt in der Datenbank bei sehr großen Daten eine sehr langsame Lösung und daher nicht zu empfehlen. Einziger Vorteil bei dieser Lösung ist, dass die Datenbank, kompakt mit allen Daten, relativ einfach gesichert und wiederhergestellt werden kann.

Es ist aber auch möglich und gängige Praxis, die Datenbank mittels der Datenbanksoftware und die im Dateisystem abgelegten Daten separat zu sichern.

Die Lösung, lediglich die Metadaten und einen Verweis, auf die im Dateisystem abgelegten Daten in der Datenbank zu speichern, wird daher bevorzugt. Dieser Weg ist, wie in Tabelle 5.9 mit der Datenbank MySQL bewiesen wurde, deutlich schneller und ebenfalls mit allen untersuchten Datenbanken möglich.

Da die NoSQL-Datenbanken für große Datenmengen im Petabyte-Bereich mit Zeilenzahlen im hohen Millionenbereichen konzipiert wurden, sind diese für das Speichern der Metadaten deutlich unterfordert. Außerdem ist es sehr schwer und mit einem erheblichen Mehraufwand verbunden, die benötigten Beziehungen abzubilden. Auch das Umsetzen von komplexen Suchanfragen, die zum korrekten Auswerten der Messdaten benötigt werden, ist bei diesen Systemen nur sehr aufwändig in der Anwendungsschicht zu lösen. Abschließend kommt hinzu, dass für diese Systeme eine enorme Einarbeitungszeit für die Implementierung und die spätere Administration notwendig ist.

Eine gute und ebenso schnelle Lösung, die auch durch die Entscheidungsmatrix in Tabelle 5.11 gestützt wird, ist die Datenbank MySQL. Sie ist im Gegensatz zu Oracle 11gR2 Enterprise Edition kostenlos und unterliegt keinen Einschränkungen wie Oracle 11gR2 XE, z.B. beim möglichen nutzbaren Arbeitsspeicher oder der Wahl des Betriebssystems. Da MySQL im Fachbereich schon im Einsatz ist, kann auf Strategien für das Administrieren und das Sichern der Datenbank, ohne nötige Einarbeitung, zurückgegriffen werden.

In Tabelle 5.10 ist zu sehen, welche gestellten Anforderungen aus Tabelle 5.1 auf Seite 46 von den einzelnen Datenbanken erfüllt werden konnten.

Anforderung	wichtig	Oracle EP	Oracle XE	MySQL	HBase	MongoDB
1	ja	relationales Schema			Tabellen	Collection
2	ja	Blob	Nur das Speichern eines Verweises möglich			
3	nein	Speichern von Binärdateien			Anwendung (AW)	
4	ja	Backup / ACID			Backup/ AW	Backup/ AW
5	nein	Bestandteil Grundstudium			-	-
6	nein	Bestandteil Grundstudium			-	-
7	ja	Bestandteil Grundstudium			-	-
8	nein	direkte Speichern der Binärdateien (Blob) in DB dauert länger				
9	ja	Abfragesprache SQL			-	-
10	ja	Rechtevergabe an Nutzer			-	-
11	ja	ACID			BASE	BASE
12	nein	Nicht Hochverfügbar (kein Cluster)				
13	ja	Abfragesprache SQL			Anwendung	
14	ja	-	Kostenlos			
15	ja	relationales Schema			Tabellen	Collection
16	ja	relationales Schema			Tabellen	Collection
17	ja	relationales Schema			Anwendung	
18	ja	Trigger			Anwendung	
19	ja	relationales Schema			Tabellen	Collection
20	ja	relationales Schema			Versionsverw.	DBRef
21	ja	relationales Schema			Tabellen	Collection
22	ja	relationales Schema			Anwendung	
23	ja	relationales Schema			Tabellen	Collection
24	ja	relationales Schema			Anwendung	
25	ja	relationales Schema			Tabellen	Collection
26	ja	relationales Schema			Anwendung	
27	ja	relationales Schema			extra Tabellen	DBRef
28	ja	relationales Schema			extra Tabellen	DBRef
29	ja	relationales Schema			extra Tabellen	DBRef
30	ja	Abfragesprache SQL			-	-
31	ja	relationales Schema			Versionsverw.	DBRef

Tabelle 5.10: Erfüllte Anforderungen der DBMS

Nach Erwägung all dieser gesammelten Erkenntnisse wurde zur Unterstützung der Wahl eines geeigneten Datenbanksystems folgende Entscheidungsmatrix erstellt:

Entscheidungsmatrix		Oracle EP		Oracle XE		MySQL		HBase		MongoDB	
Kriterien	g_i	Bw	g_i	Bw	g_i	Bw	g_i	Bw	g_i	Bw	g_i
Erfüllung der Anforderungen	40	10	400	9	360	9	360	7	280	8	320
Kosten	20	0	0	10	200	10	200	10	200	10	200
einfache Implementierung	15	4	60	4	60	10	150	2	30	3	45
einfache Administration	15	4	60	4	60	8	120	3	45	3	45
Beschränkungen der Datenbank	10	10	100	6	60	9	90	8	80	8	80
Summe	100	28	620	33	720	46	920	30	635	32	690

g_i = Gewichtung, Bw = Bewertung
1 = sehr schlecht, 10 = sehr gut

Tabelle 5.11: Entscheidungsmatrix zur Wahl eines Datenbanksystems

Zur Beschränkung der Datenbank lässt sich sagen, dass die Datenbank Oracle 11gR2 XE im Arbeitsspeicher, im Tablespace sowie in der Wahl des Betriebssystems beschränkt ist. Bei MySQL können keine Dateien über 250 MB eingefügt werden. Bei HBase und MongoDB können ebenfalls keine großen Dateien eingefügt werden und sie sind auch bzgl. des Betriebssystems beschränkt.

Die Erfüllung der Anforderungen wurden mit 40 Prozent bemessen. Da es bei den Datenbanken Oracle XE und MySQL nicht möglich war alle Testdaten direkt in der Datenbank zu speichern, diese aber dennoch alle Anforderung erfüllen, wurden 90 Prozent der möglichen Punkte vergeben. Die Datenbanken HBase und MongoDB können enigie Anforderungen nicht erfüllen und wurden aus diesem Grund schlechter bewertet. Oracle in der Enterprise Edition erhielt alle möglichen Punkte, da die Anforderung 14 bzgl. der möglichst geringen Kosten, die von diesem Datenbanksystem nicht erfüllt werden konnte, extra bewertet wurde. Da keine Gelder zur Anschaffung von Hard- oder Software für den Anwendungsfall zur Verfügung stehen wurde die Anforderung 14 als sehr wichtig erachtet und aus diesem Grund mit 20 Prozent einzeln bewertet. Insgesamt wurden damit 60 Prozent der Punkte für die Erfüllung der Anforderungen vergeben. Die übrigen 40 Prozent wurden unter den anderen drei Kriterien nach Wichtigkeit aufgeteilt.

In Anbetracht der gesammelten Erkenntnisse kommt der Autor zu dem Ergebnis, dass MySQL die effektivste Datenbank zur Speicherung der medizinischen Messdaten im Fachbereich Informatik und Medien ist.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Ziel dieser Bachelorarbeit war es, ein SQL- oder NoSQL-Datenbanksystem zu finden, in welchem die aufkommenden medizinischen Messdaten möglichst effizient gespeichert werden können.

Zu Beginn wurden im Kapitel 2 die unterschiedlichen medizinischen Messgeräte in Bezug auf die Funktion und die entstehenden Daten untersucht sowie eine Analyse der Arbeitsabläufe und beteiligten Personen an einem Versuch erstellt. Im Anschluss wurden im Kapitel 3 und im Kapitel 4 die benötigten Grundlagen zum Verständnis der Lösungsansätze für diese Aufgabe geschaffen. Im Kapitel 5 wurden diese Grundlagen angewendet, die nötigen Modelle entwickelt und die ausgewählten Datenbanken, MySQL, Oracle 11gR2 XE, Oracle 11gR2 Enterprise Edition, HBase und MongoDB, implementiert und getestet.

Der Vergleich von SQL- und NoSQL-Datenbanksystemen beinhaltet im Wesentlichen auch die Untersuchung, inwieweit Konzepte aus dem NoSQL-Bereich für den konkreten Anwendungsfall sinnvoll sind. Auf konzeptioneller Ebene unterscheiden sich SQL- und NoSQL-Datenbanksysteme wie folgt:

NoSQL:

- keine relationale Struktur
- keine Typ-Integrität
- Anfragen kompliziert über Programmiersprache wie z.B. Java
- konsistenter Zustand muss über Anwendung sichergestellt werden

SQL:

- relationale Struktur
- Typ-Integrität ist sichergestellt
- Anfragesprache SQL
- konsistenter Zustand durch ACID mittels Definierter Transaktionen sichergestellt

Der Autor kam zu dem Ergebnis, dass die relationale Struktur zwischen den Entitäten sehr wichtig für die Auswertung der Versuchsdaten ist und nicht bei jeder neu hinzukommenden Anfrage an die Datenbank die Anwendung neu implementiert weden sollte. Nach dem Einsatz einer Datenbank für diesen Anwendungsfall kommen sehr wahrscheinlich neue, vielleicht sehr komplexe, Anfragen an die Datenbank zur Auswertung der Versuchsergebnisse hinzu. Die einfachste und beste Möglichkeit dies umzusetzen ist die Abfragesprache SQL. Da das Lehr- und Forschungspersonal aus Informatikern besteht und relationale Datenbanken sowie die Abfragesprache SQL Bestandteil des Grundstudiums sind, können diese möglichen neuen Anfragen an die Datenbank auch schnell formuliert weden. Außerdem sind NoSQL-Datenbanken für Zeilen im Millionenbereich ausgelegt und da große Dateien nicht direkt in diesen Systemen gespeichert werden können, wären sie mit der ausschließlichen Speicherung von Metadaten unterfordert. Auch erfordert die Implementierung und Administration eine enorme Einarbeitungszeit. In Anbetracht dieser Ergebnisse sind NoSQL-Datenbanksysteme für diesen Anwendungsfall zurzeit nicht besonders geeignet.

Des Weiteren wurde festgestellt, dass es nicht sehr sinnvoll ist, große Dateien im Gigabyte-Bereich direkt in einer Datenbank zu speichern. Das Speichern dieser Datenmengen ist sehr langsam und nur mit der teuren Datenbank Oracle in der Enterprise Edition möglich. Eine sehr gute und auch deutlich schnellere Lösung ist das Speichern der Messdaten im dafür ausgelegten und optimierten Dateisystem sowie das Speichern der Metadaten mit einem Verweis auf die im Dateisystem abgelegten Messdaten in einer Datenbank. Somit ist es jetzt möglich, eine Beziehung zwischen den Metadaten und den Messdaten herzustellen.

Bei diesem Lösungsweg werden alle Probandendaten und die Metainformationen zu den Versuchsergebnissen nicht mehr in unterschiedlichen Ordnern und Dateiformaten unübersichtlich im Dateisystem, sondern gesammelt und jederzeit leicht auswertbar in einer Datenbank gespeichert. Durch den in der Datenbank gespeicherten Verweis zu den Versuchsergebnissen im Dateisystem lassen sich diese gut verwalten und in Verbindung mit allen anderen Informationen auswerten. Ein großer Vorteil besteht darin, dass sich mit Hilfe der Abfragesprache SQL sehr schnell und effektiv Metadaten auswerten lassen. Die gesamte Struktur ist somit viel übersichtlicher als die momentan im Fachbereich angewandte Art der Datenverwaltung.

Die Datenbank MySQL wird als beste Lösungsmöglichkeit angesehen. Sie ist, im Gegensatz zu Oracle 11gR2 Enterprise Edition, kostenlos und unterliegt keinen Einschränkungen wie die Datenbank Oracle 11gR2 XE. Ein weiterer wichtiger Grund für diese Entscheidung ist die Tatsache, dass MySQL im Gegensatz zu Oracle 11gR2 XE im Fachbereich schon im Einsatz ist. Dadurch kann auf ausgearbeitete Strukturen und Funktionalitäten sowohl für das Implementieren als auch für das Administrieren zurückgegriffen werden.

6.2 Ausblick

Die Daten sollen für mindestens zehn Jahre abrufbar sein, dadurch werden mit der Zeit viele Datensätze bei den Metadaten entstehen. Das bedeutet, dass auf Dauer alle Operationen deutlich langsamer werden. Da der Schwerpunkt dieser Arbeit die Evaluierung eines geeigneten Datenbanksystems war, wurde die Optimierung nicht weiter betrachtet. Die Möglichkeiten dieser, sei es durch Indizes oder das Konzept der Partitionierung, sollten auf jeden Fall geprüft werden.

7 Anhang

Dieser Anhang enthält die wichtigsten Quelltexte die im Rahmen dieser Bachelorarbeit erstellt wurden.

Quelltexte zu MySQL

Listing 7.1: Prozedur um bei MySQL einen Blob im Dateisystem zu speichern

```
drop procedure if exists dump_blob;
delimiter //

create procedure dump_blob()
begin
        set @query = concat('select␣Ergebnis␣from␣Versuch␣where␣
           VersuchID=1␣into␣dumpfile␣"E:/blob.dat"');
        prepare write_file from @query;
        execute write_file;
end //
delimiter ;
```

Listing 7.2: Blob bei MySQL in der Datenbank Speichern oder Auslesen

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using MySql.Data.MySqlClient;
using System.IO;


namespace insert_blob_mysql
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
private void button1_Click(object sender, EventArgs e)
{
    insertBlob();
}

private void button2_Click(object sender, EventArgs e)
{
    selectBlob();
}

public void insertBlob()
{
    MySql.Data.MySqlClient.MySqlConnection conn;
    MySql.Data.MySqlClient.MySqlCommand cmd;

    conn = new MySql.Data.MySqlClient.MySqlConnection();
    cmd = new MySql.Data.MySqlClient.MySqlCommand();

    string SQL;
    UInt32 FileSize;
    byte[] rawData;
    FileStream fs;

    conn.ConnectionString = "server=127.0.0.1;uid=root;" + "
       pwd=␣;database=weidner_db;";
    try
    {
        fs = new FileStream(@"C:\test.dat", FileMode.Open,
           FileAccess.Read);
        FileSize = (UInt32)fs.Length;

        rawData = new byte[FileSize];
        fs.Read(rawData, 0, (int)FileSize);
        fs.Close();

        conn.Open();

        SQL = "INSERT␣INTO␣Versuch␣VALUES(NULL
           ,10,'2012-07-01','02:00:00','ProCompInfiniti',?
           FileSize,?File)";

        cmd.Connection = conn;
        cmd.CommandText = SQL;
        cmd.Parameters.Add("?FileSize", FileSize);
        cmd.Parameters.Add("?File", rawData);

        cmd.ExecuteNonQuery();

        MessageBox.Show("File␣Inserted␣into␣database␣
           successfully!", "Success!", MessageBoxButtons.OK,
            MessageBoxIcon.Asterisk);

        conn.Close();
    }
    catch (MySql.Data.MySqlClient.MySqlException ex)
```

```
        {
            MessageBox.Show("Error␣" + ex.Number + "␣has␣
                occurred:␣" + ex.Message, "Error",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    public void selectBlob()
    {
        MySql.Data.MySqlClient.MySqlConnection conn;
        MySql.Data.MySqlClient.MySqlCommand cmd;
        MySql.Data.MySqlClient.MySqlDataReader myData;

        conn = new MySql.Data.MySqlClient.MySqlConnection();
        cmd = new MySql.Data.MySqlClient.MySqlCommand();

        string SQL;
        UInt32 FileSize;
        byte[] rawData;
        FileStream fs;

        conn.ConnectionString = "server=127.0.0.1;uid=root;" + "
            pwd=␣;database=weidner_db;";
        SQL = "SELECT␣Ergebnis,␣Dateigroesse␣FROM␣Versuch␣where␣
            VersuchID=9";

        try
        {
            conn.Open();

            cmd.Connection = conn;
            cmd.CommandText = SQL;

            myData = cmd.ExecuteReader();

            if (!myData.HasRows)
                throw new Exception("There␣are␣no␣BLOBs␣to␣save"
                    );

            myData.Read();

            FileSize = myData.GetUInt32(myData.GetOrdinal("
                Dateigroesse"));
            rawData = new byte[FileSize];

            myData.GetBytes(myData.GetOrdinal("Ergebnis"), 0,
                rawData, 0, (int)FileSize);

            fs = new FileStream(@"E:\test20.dat", FileMode.
                OpenOrCreate, FileAccess.Write);
            fs.Write(rawData, 0, (int)FileSize);
            fs.Close();

            MessageBox.Show("File␣successfully␣written␣to␣disk!"
                , "Success!", MessageBoxButtons.OK,
                MessageBoxIcon.Asterisk);
```

```
                myData.Close();
                conn.Close();
            }
            catch (MySql.Data.MySqlClient.MySqlException ex)
            {
                MessageBox.Show("Error␣" + ex.Number + "␣has␣
                    occurred:␣" + ex.Message, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }


    }
}
```

Quelltexte zu Oracle

Listing 7.3: Erstellen eines Nutzers in Oracle

```
-- DB Bereich und User anlegen
create tablespace weidner_db
datafile 'weidner.dbf' size 31G reuse
default storage (initial 10K next 50K
minextents 1 maxextents 999) online;

create user weidner identified by password
default tablespace weidner_db
temporary tablespace temp
profile default
account unlock
quota unlimited on weidner_db;

grant connect to weidner;
grant create procedure to weidner;
grant create table to weidner;
grant create trigger to weidner;
grant create view to weidner;
grant create sequence to weidner;
grant create public synonym to weidner;
grant create session to weidner;
grant drop public synonym to weidner;
```

Listing 7.4: Prozedur zum Einfügen eines Blobs in Oracle

```
-- Beim ersten Ausfuehren erst einw Directory als root erstellen und
     die Rechte auf diese Directory an den Nutzer vergeben
-- Linux: CREATE OR REPLACE DIRECTORY BLOB AS '/opt/oracle/admin/
    orcl/Blob/' Windows: Create OR REPLACE DIRECTORY BLOB AS 'E:\';
-- GRANT read, write ON DIRECTORY BLOB TO weidner;

DECLARE
v_bfile BFILE;
v_blob BLOB;
```

```
BEGIN
INSERT INTO Versuch (VersuchID,ProbandID,Datum,Dauer,VersuchsArt,
   Ergebnis,Dateiname)
values (null, 1, to_date('01.06.2012','dd.mm.yyyy'), 120, '
   ProCompInfiniti', empty_blob(), 'C:\test.dat')
RETURN Ergebnis INTO v_blob;

v_bfile :=BFILENAME ('BLOB','blobDie.dat');
Dbms_Lob.Fileopen(v_bfile,Dbms_Lob.File_Readonly);
Dbms_Lob.Loadfromfile(v_blob,v_bfile,Dbms_Lob.Getlength(v_bfile));
Dbms_Lob.Fileclose(v_bfile);

COMMIT;
END;
/
```

Listing 7.5: Prozedur zum Auslesen eines Blobs in Oracle

```
create or replace procedure selectBlob (
  i_versuch_id in number) is

cursor s_blob (s_blob_id in versuch.versuchid%type) is
    select ergebnis
    from versuch
    where versuchid = s_blob_id;
    tmp_blob versuch.ergebnis% type;

        v_position number :=0;
        v_amount number;

        v_file utl_file.file_type;

begin
        if s_blob%isopen then
                close s_blob;
        end if;
        open s_blob (i_versuch_id);
        fetch s_blob into tmp_blob;

        v_file := utl_file.fopen (
        location => 'BLOB',
        filename => 'blob.dat',
        open_mode => 'wb',
        max_linesize => 32000
        );
        while v_position < dbms_lob.getlength(tmp_blob) loop
        v_amount := (dbms_lob.getlength(tmp_blob)) - v_position;
        if v_amount > 32000 then
        v_amount := 32000;
        end if;
        utl_file.put_raw (
        file => v_file,
        buffer => dbms_lob.substr (
        lob_loc => tmp_blob,
        amount => v_amount,
        offset => v_position + 1
```

```
        ),
        autoflush => false
        );
        v_position := v_position + v_amount;
        end loop;
        utl_file.fflush (
        file => v_file
        );
        utl_file.fclose (
        file => v_file
        );

end;
/
```

Quelltexte zu HBase

Listing 7.6: Quelltext zum Erstellen eines Schemas in HBase

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;

public class MyHbaseTable {

  public static void main (String[] args) throws IOException {
    HTableDescriptor tabelleUntersuchung = new HTableDescriptor ("
       Untersuchung");
    HColumnDescriptor datum = new HColumnDescriptor ("Datum");
    HColumnDescriptor art = new HColumnDescriptor ("Art");
    HColumnDescriptor dauer = new HColumnDescriptor ("Dauer");
    HColumnDescriptor proband = new HColumnDescriptor ("Proband");

    myTable.addFamily (datum);
    myTable.addFamily (art);
    myTable.addFamily (dauer);
    myTable.addFamily (proband);

    Configuration myConfig = HBaseConfiguration.create ();
    HBaseAdmin myAdmin = new HBaseAdmin (myConfig);
    myAdmin.createTable (tabelleUntersuchung);
  }
}
```

Listing 7.7: Quelltext zum Einfügen von Daten in HBase

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;
```

```
public class myInsert {

  public static void main (String[] args) throws IOException {
    Configuration myConfig = HBaseConfiguration.create ();
    HTable tabelleUntersuchung = new HTable (myConfig, "Untersuchung
       ");

    Put insertDatum = new Put (Bytes.toBytes ("Datum") );
    insertDatum.add (Bytes.toBytes ("Datum"), Bytes.toBytes ("
       ErstesDatum"), Bytes.toBytes ("10.07.12" );

    Put insertArt = new Put (Bytes.toBytes ("Art") );
    insertArt.add (Bytes.toBytes ("Art"), Bytes.toBytes ("ErsteArt")
       , Bytes.toBytes ("EKG") );

    Put insertDauer = new Put (Bytes.toBytes ("Dauer") );
    insertDauer.add (Bytes.toBytes ("Dauer"), Bytes.toBytes ("
       ErsteDauer"), Bytes.toBytes ("10:00" );

    Put insertProband = new Put (Bytes.toBytes ("Proband") );
    insertProband.add (Bytes.toBytes ("Proband"), Bytes.toBytes ("
       ErsterProband"), Bytes.toBytes (1) );

    tableHeader.put (insertDatum);
    tableHeader.put (insertArt);
    tableHeader.put (insertDauer);
    tableHeader.put (insertProband);
  }
}
```

Listing 7.8: Quelltext zum Auslesen von Daten in HBase

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;

public class MyHbaseTable {

  public static void main (String[] args) throws IOException {
    Configuration myConfig = HBaseConfiguration.create();
    HTable tabelleUntersuchung = new HTable (myConfig, "Untersuchung
       ");

    Get selectDatum = new Get (Bytes.toBytes ("Datum") );
    Result myResult = tabelleUntersuchung.get (get);
    byte[] myValue = myResult.getValue (Bytes.toBytes ("Datum"),
       Bytes.toBytes ("ErstesDatum") );

    System.out.println (Bytes.toString (myValue) );
  }
}
```

Listing 7.9: Quelltext zum Erstellen der Tabellen in HBase

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;


public class MyHbaseTable {

  public static void main (String[] arg) throws IOException {
    //Tabelle Proband anlegen
    HTableDescriptor tableProband = new HTableDescriptor ("Proband")
       ;
    HColumnDescriptor probandID = new HColumnDescriptor ("ProbandID"
       );
    HColumnDescriptor alter = new HColumnDescriptor ("Alter");
    HColumnDescriptor geschlecht = new HColumnDescriptor ("
       Geschlecht");
    HColumnDescriptor groesse = new HColumnDescriptor ("Groesse");
    HColumnDescriptor gewicht = new HColumnDescriptor ("Gewicht");
    HColumnDescriptor armlaenge = new HColumnDescriptor ("Armlaenge"
       );
    HColumnDescriptor beinlaenge = new HColumnDescriptor ("
       Beinlaenge");

    tableProband.addFamily (probandID);
    tableProband.addFamily (alter);
    tableProband.addFamily (geschlecht);
    tableProband.addFamily (groesse);
    tableProband.addFamily (gewicht);
    tableProband.addFamily (armlaenge);
    tableProband.addFamily (beinlaenge);

    //Tabelle Versuch anlegen
    HTableDescriptor tableVersuch = new HTableDescriptor ("Versuch")
       ;
    HColumnDescriptor versuchID = new HColumnDescriptor ("VersuchID"
       );
    HColumnDescriptor datum = new HColumnDescriptor ("Datum");
    HColumnDescriptor dauer = new HColumnDescriptor ("Dauer");
    HColumnDescriptor versuchsart = new HColumnDescriptor ("
       VersuchsArt");
    HColumnDescriptor ergebnis = new HColumnDescriptor ("Ergebnis");

    ergebnis.setMaxVersions(10);
    tableVersuch.addFamily (versuchID);
    tableVersuch.addFamily (datum);
    tableVersuch.addFamily (dauer);
    tableVersuch.addFamily (versuchsart);
    tableVersuch.addFamily (ergebnis);

    //Tabelle Proband-Versuch anlegen
    HTableDescriptor tableProbandVersuch = new HTableDescriptor ("
       Proband-Versuch");
```

```
HColumnDescriptor foreignKeyPV = new HColumnDescriptor ("
   ProbandID\\x00VersuchID");
HColumnDescriptor datumProbandVersuch = new HColumnDescriptor ("
   Timestamp");

tableProbandVersuch.addFamily (foreignKeyPV);
tableProbandVersuch.addFamily (datumProbandVersuch);

//Tabelle Header anlegen
HTableDescriptor tableHeader = new HTableDescriptor ("Header");
HColumnDescriptor headerID = new HColumnDescriptor ("HeaderID");
HColumnDescriptor dateiname = new HColumnDescriptor ("Dateiname"
   );
HColumnDescriptor anzSignale = new HColumnDescriptor ("
   AnzSignale");
HColumnDescriptor abtastrate = new HColumnDescriptor ("
   Abtastrate");
HColumnDescriptor anzSignalpunkte = new HColumnDescriptor ("
   AnzSignalpunkte");
HColumnDescriptor kommentarHeader = new HColumnDescriptor ("
   KommentarHeader");

tableHeader.addFamily (headerID);
tableHeader.addFamily (dateiname);
tableHeader.addFamily (anzSignale);
tableHeader.addFamily (abtastrate);
tableHeader.addFamily (anzSignalpunkte);
tableHeader.addFamily (kommentarHeader);

//Tabelle Versuch-Header anlegen
HTableDescriptor tableVersuchHeader = new HTableDescriptor ("
   Versuch-Header");
HColumnDescriptor foreignKeyVH = new HColumnDescriptor ("
   VersuchID\\x00HeaderID");
HColumnDescriptor datumVersuchHeader = new HColumnDescriptor ("
   Timestamp");

tableVersuchHeader.addFamily (foreignKeyVH);
tableVersuchHeader.addFamily (datumVersuchHeader);

//Tabelle Signal anlegen
HTableDescriptor tableSignal = new HTableDescriptor ("Signal");
HColumnDescriptor signalID = new HColumnDescriptor ("SignalID");
HColumnDescriptor zeilennummerS = new HColumnDescriptor ("
   Zeilennummer");
HColumnDescriptor anzBits = new HColumnDescriptor ("AnzBits");
HColumnDescriptor umrechnungsfaktor = new HColumnDescriptor ("
   Umrechnungsfaktor");
HColumnDescriptor adcResolution = new HColumnDescriptor ("
   ADC_Resolution");
HColumnDescriptor adcZero = new HColumnDescriptor ("ADC_Zero");
HColumnDescriptor anfangswert = new HColumnDescriptor ("
   Anfangswert");
HColumnDescriptor skalierungsfaktor = new HColumnDescriptor ("
   Skalierungsfaktor");
```

```
HColumnDescriptor achsenverschiebung = new HColumnDescriptor ("
   Achsenverschiebung");
HColumnDescriptor kommentarSignal = new HColumnDescriptor ("
   KommentarSignal");

tableSignal.addFamily (signalID);
tableSignal.addFamily (zeilennummerS);
tableSignal.addFamily (anzBits);
tableSignal.addFamily (umrechnungsfaktor);
tableSignal.addFamily (adcResolution);
tableSignal.addFamily (adcZero);
tableSignal.addFamily (anfangswert);
tableSignal.addFamily (skalierungsfaktor);
tableSignal.addFamily (achsenverschiebung);
tableSignal.addFamily (kommentarSignal);

//Tabelle Header-Signal anlegen
HTableDescriptor tableHeaderSignal = new HTableDescriptor ("
   Header-Signal");
HColumnDescriptor foreignKeyHS = new HColumnDescriptor ("
   HeaderID\\x00SignalID");
HColumnDescriptor datumHeaderSignal = new HColumnDescriptor ("
   Timestamp");

tableHeaderSignal.addFamily (foreignKeyHS);
tableHeaderSignal.addFamily (datumHeaderSignal);

//Tabelle Kommentar anlegen
HTableDescriptor tableKommentar = new HTableDescriptor ("
   Kommentar");
HColumnDescriptor kommentarID = new HColumnDescriptor ("
   KommentarID");
HColumnDescriptor zeilennummerK = new HColumnDescriptor ("
   Zeilennummer");
HColumnDescriptor kommentar = new HColumnDescriptor ("Kommentar"
   );

tableKommentar.addFamily (kommentarID);
tableKommentar.addFamily (zeilennummerK);
tableKommentar.addFamily (kommentar);

//Tabelle Versuch-Kommentar anlegen
HTableDescriptor tableHeaderKommentar = new HTableDescriptor ("
   Versuch-Kommentar");
HColumnDescriptor foreignKeyHK = new HColumnDescriptor ("
   VersuchID\\x00KommentarID");
HColumnDescriptor datumHeaderKommentar = new HColumnDescriptor (
   "Timestamp");

tableHeaderKommentar.addFamily (foreignKeyHK);
tableHeaderKommentar.addFamily (datumHeaderKommentar);

//Tabellen in die Datenbank einfÃŒgen
Configuration myConfig = HBaseConfiguration.create ();
HBaseAdmin myAdmin = new HBaseAdmin(myConfig);
myAdmin.createTable(tableProband);
```

```
    myAdmin.createTable(tableVersuch);
    myAdmin.createTable(tableProbandVersuch);
    myAdmin.createTable(tableHeader);
    myAdmin.createTable(tableVersuchHeader);
    myAdmin.createTable(tableSignal);
    myAdmin.createTable(tableHeaderSignal);
    myAdmin.createTable(tableKommentar);
    myAdmin.createTable(tableHeaderKommentar);
  }
}
```

Listing 7.10: Quelltext für das Einfügen eines Blobs in HBase

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;

public class CreateSchema {

  public static void main (String[] args) throws IOException {
    //Zählt die eingetragenen Zeilen in der DB
    int zaehler = 0;
        int z = 0; //zählt die Zeilen der Datei um die Größe des
           Arrays festzulegen
        String line, line2;

        //Tabelle zum Einfügen festlegen
        Configuration config = HBaseConfiguration.create ();
        HTable table = new HTable(config, "Versuch");

        //In welche Zeile das Ergebnis soll
        Put p = new Put (Bytes.toBytes ("Ergebnis") );

        try {
        //Datei einlesen
                File file = new File ("/usr/lib/hbase/bin/test5.dat"
                   );
                BufferedReader in = new BufferedReader (new
                   FileReader (file) );

                //Um die benötigte Arraygröße zu bestimmen
                while ((line = in.readLine () ) != null) {
                        z++;
                }

                BufferedReader in2 = new BufferedReader (new
                   FileReader (file) );
                String [] feld = new String [z];
```

```
                //Datei auslesen und jede Zeile einzeln in die DB
                    schieben
                while ((line2 = in2.readLine () ) != null) {
                        feld [zaehler] = line2;

                        p.add (Bytes.toBytes ("Ergebnis"), Bytes.
                            toBytes (zaehler),
                                Bytes.toBytes (feld [zaehler]) );

                        table.put (p);
                        zaehler++;
                }
                //Um die eingefügte Anzahl von Zeilen zu wissen (
                    nötig um die Datei richtig aus der Datenbank zu
                    holen)
                zaehler--;

                //Anzahl der Zeilen in die DB schreiben
                p.add (Bytes.toBytes ("Zaehler"), Bytes.toBytes ("
                    Ende"), Bytes.toBytes (zaehler) );
                table.put (p);
                in.close ();
                in2.close ();
                System.out.println ("Fertig");
        } catch (IOException ex) {
                ex.printStackTrace ();
        }
  }
}
```

Listing 7.11: Quelltext für das Auslesen eines Blobs in HBase

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;

public class CreateSchema {

  public static void main (String[] args) throws IOException {
        //Tabelle auslesen
        boolean ende = false;
        byte gh;
        int i = 0; //zur Umwandlung des Zählers in integer
        int j = 0; //Zählvariable

        Configuration config = HBaseConfiguration.create ();
        HTable table = new HTable (config, "Versuch");

        Get get = new Get (Bytes.toBytes ("Ergebnis") );
        Result result = table.get (get);
```

```
        byte[] zaehler = result.getValue(Bytes.toBytes ("Zaehler"),
             Bytes.toBytes ("Ende"));
        //Der Zähler für die eingefügten Zeilen, aber in Form '\x00\
           x00\x00\x00'
        gh = zaehler[0];

        while (!ende) {
                //Umwandlung des Zählers in einen integer
                i = gh;
                i &= 0x000000FF;

                //Der Wert im Byte Array ist nur an der letzten
                   Stelle Größer 0 (Je nach Anzahl eingefügter
                   Zeilen variiert diese Stelle)
                if (i > 0)
                        ende = true;
                else {
                        j++;
                        gh = zaehler [j];
                }
        }

        //Datei auslesen und auf dem Bildschirm ausgeben, hier kann
           die Datei auch wieder ins Dateisystem geschrieben werden.
        for (int k = 0; k <= i; k++) {
                byte[] value = result.getValue(Bytes.toBytes ("
                   Ergebnis"), Bytes.toBytes (k));
                System.out.println(Bytes.toString(value));
        }
  }
}
```

Listing 7.12: Quelltext für das Einfügen von Daten in HBase

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;

public class myInsert {

  public static void main (String[] args) throws IOException {
    Configuration myConfig = HBaseConfiguration.create ();
    HTable tableHeader = new HTable (myConfig, "Header");

    Put insertHeaderID = new Put (Bytes.toBytes ("HeaderID") );
    insertHeaderID.add (Bytes.toBytes ("HeaderID"), Bytes.toBytes ("
       FirstHeader"), Bytes.toBytes (1) );

    Put insertDateiname = new Put (Bytes.toBytes ("Dateiname") );
    insertDateiname.add (Bytes.toBytes ("Dateiname"), Bytes.toBytes
       ("FirstDateiname"), Bytes.toBytes ("/usr/bin/test.dat") );
```

```
    Put insertAnzSignale = new Put (Bytes.toBytes ("AnzSignale") );
    insertAnzSignale.add (Bytes.toBytes ("AnzSignale"), Bytes.
       toBytes ("FirstSignal"), Bytes.toBytes (2) );

    Put insertAbtastrate = new Put (Bytes.toBytes ("Abtastrate") );
    insertAbtastrate.add (Bytes.toBytes ("Abtastrate"), Bytes.
       toBytes ("FirstAbtastrate"), Bytes.toBytes (512) );

    Put insertAnzSignalpunkte = new Put (Bytes.toBytes ("
       AnzSignalpunkte") );
    insertAnzSignalpunkte.add (Bytes.toBytes ("AnzSignalpunkte"),
       Bytes.toBytes ("FirstSignalpunkt"), Bytes.toBytes (12) );

    Put insertKommentarHeader = new Put (Bytes.toBytes ("
       KommentarHeader") );
    insertKommentarHeader.add (Bytes.toBytes ("KommentarHeader"),
       Bytes.toBytes ("FirstKommentar"), Bytes.toBytes ("Das␣ist␣der
       ␣erste␣Kommentar␣zu␣einem␣Header!") );


    tableHeader.put (insertHeaderID);
    tableHeader.put (insertDateiname);
    tableHeader.put (insertAnzSignale);
    tableHeader.put (insertAbtastrate);
    tableHeader.put (insertAnzSignalpunkte);
    tableHeader.put (insertKommentarHeader);
  }
}
```

Quelltexte zu MongoDB

Listing 7.13: Quelltext zum Einfügen von Daten in MongoDB

```
import java.net.UnknownHostException;
import com.mongodb.*;

public class MyMongoDBTable {

  public static void main (String[] args) {
    try {
        Mongo myMongo = new Mongo ();
        DB myDb = myMongo.getDB ("Untersuchung");
        DBCollection myCollection = myDb.getCollection ("
           Untersuchungsergebnis");

        BasicDBObject myErgebnis = new BasicDBObject ();
        myErgebnis.put ("Untersuchung1", 1);

        myCollection.insert (myErgebnis);

        myMongo.close ();
    } catch (UnknownHostException ex) {
        ex.printStackTrace ();
    } catch (MongoException ex) {
        ex.printStackTrace ();
```

```
    }
  }
}
```

Listing 7.14: Quelltext zum Lesen von Daten in MongoDB

```
import java.net.UnknownHostException;
import com.mongodb.*;

public class MyMongoDBTable {

  public static void main (String[] args) {
    try {
        Mongo myMongo = new Mongo ();
        DB myDb = myMongo.getDB ("Untersuchung");
        DBCollection myCollection = myDb.getCollection ("
           Untersuchungsergebnis");

        DBObject myErgebnis = myCollection.findOne ();
        System.out.println (myErgebnis);

        DBCursor myCursor = myCollection.find ();
        while (myCursor.hasNext () )
            System.out.println (myCursor.next () );

        BasicDBObject query = new BasicDBObject ("Untersuchung1", 1)
           ;

        myErgebnis = myCollection.find (query);
        System.out.println (myErgebnis);

        DBCursor myCursor = myCollection.find (query);
        while (myCursor.hasNext () )
            System.out.println (myCursor.next () );

        myMongo.close ();
    } catch (UnknownHostException ex) {
        ex.printStackTrace ();
    } catch (MongoException ex) {
        ex.printStackTrace ();
    }
  }
}
```

Listing 7.15: Quelltext für das Erstellen einer DB, Collection und Einfügen von Daten

```
import java.net.UnknownHostException;
import com.mongodb.*;

public class create_mongodb {

  public static void main(String[] args) {
        try {
                Mongo m = new Mongo ("localhost",27017);
                DB db = m.getDB ("Messdaten");
                DBCollection proband = db.getCollection ("Proband");
```

```
                DBCollection versuch = db.getCollection ("Versuch");
                DBCollection nachfolger = db.getCollection ("
                   Nachfolger");
                DBCollection header = db.getCollection ("Header");
                DBCollection signal = db.getCollection ("Signal");
                DBCollection kommentar = db.getCollection ("
                   Kommentar");

                //Datensatz mit dem ersten Probanden anlegen
                BasicDBObject proband1 = new BasicDBObject ();
                proband1.put ("ProbandID", 1);
                proband1.put ("Alter", 38);
                proband1.put ("Geschlecht", "Weiblich");
                proband1.put ("Groesse", 180);
                proband1.put ("Gewicht", 80);
                proband1.put ("Armlaenge", 100);
                proband1.put ("Beinlaenge", 110);

                proband.insert (proband1);

        } catch (UnknownHostException ex) {
                ex.printStackTrace ();
        } catch (MongoException ex) {
                ex.printStackTrace ();
        }
  }
}
```

Listing 7.16: Einfügen eines Blobs über die GridFS-API

```
import java.awt.List;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.net.UnknownHostException;

import com.mongodb.Mongo;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
import com.mongodb.DBCursor;
import com.mongodb.MongoException;
import com.mongodb.gridfs.GridFS;
import com.mongodb.gridfs.GridFSDBFile;
import com.mongodb.gridfs.GridFSInputFile;

public class MyMongo {

  public static void main(String[] args) {
        Mongo myMongo;
        try {
                myMongo = new Mongo();
                DB db = myMongo.getDB("myDB");
```

```
                InputStream inputStream = new FileInputStream(new
                    File("D:\\test.dat"));
                GridFS storeGridFS = new GridFS(db);
                GridFSInputFile gridFSInputFile = storeGridFS.
                    createFile(inputStream);
                gridFSInputFile.setFilename("test.dat");
                gridFSInputFile.save();

                myMongo.close();
        } catch (UnknownHostException e) {
                e.printStackTrace();
        } catch (MongoException e) {
                e.printStackTrace();
        } catch (FileNotFoundException e) {
                e.printStackTrace();
        }
  }
}
```

Quelltext zu Matlab

Listing 7.17: Quelltext für eine Matlab Datei

```
%% bvh reader
% Frames als Zeilenvektoren aus bvh-Datei einlesen
% Frames auf int16 abbgebildet, Offsets der Segmente,
    Abbildungsfaktor
function result = readbvh2(filename)

fid = fopen(filename, 'rt');
data = [];
offsets = [];
faktors =[];
frame_offsets = [];
header_joints = {};
header_channels = {};
num = 0;
line_number=0;

%finding ROOT (special Treatment!)
while num == 0
    tline = fgetl(fid);
    bvh_root = strfind(tline, 'ROOT');
    num = length(bvh_root);
end

x = textscan(tline(bvh_root+4:end), '%s');
x = x{:};
header_joints = vertcat(header_joints, x, x);
frewind(fid);
num = 0;
% header bis tag 'MOTION' parsen
while num == 0
   tline = fgetl(fid);
   offs = strfind(tline, 'OFFSET');
```

```
   joints = strfind(tline, 'JOINT');
   chans =  strfind(tline, 'CHANNELS');
   hip = strfind(tline, 'ROOT');

   if offs > 0,
      x = textscan(tline(offs+6:end), '%f');
      frame_offsets = [frame_offsets, x{1}];
   end

   if joints > 0,
      x = textscan(tline(joints+5:end), '%s');
      x = x{:};
      header_joints = vertcat(header_joints, x);
   end

    if chans > 0,
      x = textscan(tline(chans+10:end), '%s');
      x = x{:};
      header_channels = vertcat(header_channels, x);
   end

   frametime = strfind(tline, 'Frame Time:');
   num = length(frametime);
   line_number=line_number+1;
end
framerate = textscan(tline(frametime+11:end), '%f');
framerate = 1/framerate{1};
%generate data
frames = importdata(filename,' ',line_number);
frames = frames.data;
fclose(fid);
for i=1:size(frames,2)
    [col, faktor, offset]=double2int16(frames(:,i));
    offsets = vertcat(offsets, offset);
    data = horzcat(data, col);
    faktors = vertcat(faktors, faktor);
end
%generate header

%write 2 files
samples = writeDataFile(data, filename);
writeHeaderFile(header_joints, header_channels, frame_offsets,
   faktors, offsets, framerate, samples, filename);
```

Literaturverzeichnis

[Adm12] Admin, ACM S.: *ACM Sigmod*. http://www.sigmod.org/search?b_start:int=60&SearchableText=Edgar%20F.%20Codd. Version: 2012. – Online; Letzter Zugriff 09.07.2012

[Boe03] Boesch, Dr. med.: *EKG Grundlagen*. http://www.dr-boesch.ch/medicine/ekg/ekg-teil0-ableitungen.htm. Version: 2003. – Online; Letzter Aufruf 22.06.2012

[Bus10] Busse, S. ; Fachhochschule Brandenburg (Hrsg.): *Datenbanken 1*. Fachhochschule Brandenburg, 2010

[CS01] Claus, V. ; Schwill, A.: *Duden Informatik*. Meyers Lexikonredaktion, Dudenverlag Mannheim, Leipzig, Wien, Zürich, 2001

[DIFKO12] Dipl.-Inform. (FH) Katja Orlowski, M.Sc.: *Komplexpraktikum Sommersemester 2012 Praktikumsversuch mit den mobilen Sensoren SHIMMER und Kinect*, 2012. https://moodle.fh-brandenburg.de/file.php/217/Gang/PraktikumGang.pdf. – Online; Letzter Aufruf 22.06.2012

[EFH+11] Edlich, S. ; Friedland, A. ; Hampe, J. ; Brauer, B. ; Brückner, M.: *NoSQL Einstieg in die Welt Nichtrelationaler Web 2.0 Datenbanken*. Carl Hanser Verlag München, 2011

[EN02] Elmasri, R. ; Navathe, S.B.: *Grundlagen von Datenbanksystemen*. Pearson Studium, 2002

[FWBRB07] Faeskorn-Woyke, H. ; Bertelsmeier, B. ; Riemer, P. ; Bauer, E.: *Datenbanksysteme Theorie und Praxis mit SQL 2003, Oracle und MySQL*. Pearson Studium, 2007

[Geo11] George, L.: *HBase The Definitive Guide*. OReilly Media, 2011

[MK10] Markus Kirchgeorg, Dr. med.: *Ultraschall*. http://www.netdoktor.de/Diagnostik+Behandlungen/Untersuchungen/

Ultraschall-Sonografie-341.html. Version: 03 2010. – Online; Letzter Aufruf 22.06.2012

[MyS12] MySQL ; Oracle (Hrsg.): *MySQL 5.1 Referenzhandbuch*. Oracle, 2012. http://dev.mysql.com/doc/refman/5.1/de/connector-net-using-blob.html. – Online; Letzter Zugriff 07.07.2012

[Neu] Neumann, P.: *Neue Google Eye Tracking Studie*. http://www.blogtopf.de/google/neue-google-eye-tracking-studie/http://www.blogtopf.de/google/google-suchverhalten-eye-tracking/. – Onlien; letzter Zugriff 27.07.2012

[PB10a] Peter Borlinghaus, Dr. med.: *Blutdruckmessung*. http://www.netdoktor.de/Diagnostik+Behandlungen/Untersuchungen/Blutdruckmessung-215.html. Version: 07 2010. – Online; Letzter Aufruf 22.06.2012

[PB10b] Peter Borlinghaus, Dr. med.: *Elektrokardiografie*. http://www.netdoktor.de/Diagnostik+Behandlungen/Untersuchungen/Elektrokardiografie-EKG-249.html. Version: 03 2010. – Online; Letzter Aufruf 22.06.2012

[Sch10a] Schneider, Manfred: *Elektroenzephalografie*. http://www.netdoktor.de/Diagnostik+Behandlungen/Untersuchungen/Elektroenzephalografie-EEG-1172.html. Version: 05 2010. – Online; Letzter Aufruf 23.06.2012

[Sch10b] Schneider, Manfred: *Elektromyografie*. http://www.netdoktor.de/Diagnostik+Behandlungen/Untersuchungen/Elektromyografie-EMG-1173.html. Version: 05 2010. – Online; Letzter Aufruf 23.06.2012

[Sch12] Schoenefeld, F.: *NoSQL - Please Wie Web2.0, Big Data und die Cloud neue Datenbanksysteme erfordern und hervorbringen*. http://www2.htw-dresden.de/~dbst/material/20120208_169_Schoenefeld.pdf. Version: Februar 2012. – Online; letzter Zugriff 27.07.2012

[Sen03] Sengbusch, Peter v.: *Mikroskopie*. http://www.biologie.uni-hamburg.de/b-online/d03/03.htm. Version: 07 2003. – Online; Letzter Aufruf 23.06.2012

[SsH08] Saake, G. ; sattler, K.U. ; Heuer, A.: *Datenbanken Konzepte und Sprachen*. Redline GmbH Heidelberg, 2008

[TP11] T.Hillebrand ; P.Tschackert: *Evaluierung kamerabasierter Tracking-Ansätze für Motion Capture Applikationen*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2011. http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/hillebrand.pdf. – Online; Letzter Aufruf 23.06.2012

[Wik12a] Wikipedia: *Blickerfassung — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Blickerfassung&oldid=103842159. Version: 2012. – Online; Letzter Aufruf 24.06.2012

[Wik12b] Wikipedia: *Kinect — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Kinect&oldid=103797409. Version: 2012. – Online; Letzter Aufruf 24.06.2012

[Wik12c] Wikipedia: *Tupel — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Tupel&oldid=105360433. Version: 2012. – Online; Letzter Zugriff 10.07.2012

[Wik12d] Wikipedia: *Tupel (Informatik) — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Tupel_(Informatik)&oldid=98147133. Version: 2012. – Online; Letzter Aufruf 10.07.2012

Glossar

API Schnittstelle für Anwendungsprogramme.

BSON Binary Java Script Object Notation ist ein kompaktes Datenformat in einer Binären Form

bvh Biovision Hierarchy ist ein Dateiformat um Motion Capture Daten zu speichern

c3d Coordinate 3D Dateien sind eine Teilmenge eines allgemeineren ADTech Dateiformats. Sie halten Daten und Parameter in einer Datei

CRUD-Operationen CRUD ist ein Akronym für die Datenbankoperationen Create, Read, Update und Delete

csv Comma-Separated Values ist eine formatierte Textdatei

depth Schärfentiefe ist ein Format in dem Tiefeninformationen enthalten sind

ee Eine Textdatei die bei der EOG Untersuchung entsteht.

EEG Bei der Elektroenzephalografie werden Gehirnströme gemessen

eeg- Eine Binärdatei, die bei der EOG Untersuchung entsteht.

EKG Bei der Elektrokardiografie wird eine Kurve des Herzschlages aufgezeichnet

EMG Bei der Elektromyografie werden Muskelaktivitäten gemessen

Frameworks Ein Programmiergerüst, das in der Softwareentwicklung zum Einsatz kommt.

horizontale Skalierung Bei einer horizontalen Skalierung (Scale-out) versteht man das Zusammenfassen von Nodes zu Cluster-Systemen.

JSON Java Script Object Notation ist ein kompaktes Datenformat in lesbarer Textform

MoCap Motion Capture Verfahren um eine Bewegungserfassung durchzuführen

NFS Network File System ist ein von Sun Microsystems entwickeltes Protokoll, das den Zugriff auf Dateien über ein Netzwerk ermöglicht.

NoSQL NoSQL wird meist als Not only SQL übersetzt.

rgb Rot Grün Blau ist ein Format, in dem Bilddateien gespeichert weden

SQL SQL steht für Structured Query Language und ist eine Datenbanksprache zur Beschreibung und Manipulation von Datenstrukturen in relationalen Datenbanken.

Tablespace Bereich eines Datenbanknutzers, in dem alle dem Nutzer zugeordneten Objekte (wie z.B. Tabellen,Trigger,Indexe usw.) gespeichert werden.

tif Tagged Image File ist ein Dateiformat zur Speicherung von Bilddateien.

Trigger Funktion, insbesondere relationler Datenbankmanagementsysteme, bei der beim Eintreffen eines definierten Ereignisses eine ebenfalls vorher definierte Tätigkeit ausgeführt wird.

vertikale Sakallierung Bei einer vertikalen Skallierung werden die Datenbankserver mit der neusten Hardware aufgerüstet.

wmv Windows Media Video ist ein Windows Audio/Video-Format

xml Extensible Markup Language ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit zum Thema

Evaluation von SQL- und NoSQL-Datenbanksystemen zur Speicherung medizinischer Messdaten

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Brandenburg/Havel, den

Unterschrift