Fachbereich Elektrotechnik und Informatik
Dekanat - Prüfungsausschuss

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

# Thema der Bachelorarbeit

für

Herrn Chris D e t e r

Modellierung relationaler Datenstrukturen mit Hilfe der Graphen-datenbank Neo4J

Professor Dr. Hinrichs

gefördert durch:

Ausgabedatum: 03. April 2017
Abgabedatum: 03. Juli 2017

(Professor Dr. Andreas Hanemann)
Vorsitzender des Prüfungsausschusses

## Aufgabenstellung:

### Motivation

Die MACH Software ist in 30 Jahren Entwicklungszeit zu einer umfassenden Lösung für die Bereiche Finanzmanagement, Personalmanagement und Prozessmanagement geworden. Alle Module der Software verwalten ihre Daten in einer relationalen Datenbank in mittlerweile über 1000 Tabellen. Dafür werden vier unterschiedliche Datenbankmanagementsysteme unterstützt.

Für die kontinuierliche Weiterentwicklung der MACH Software sind auch immer wieder Änderungen an den Datenstrukturen notwendig. Diese Änderungen unterliegen aufgrund der unterschiedlich eingesetzten Datenbanksysteme bestimmten Regeln. Außerdem ist es notwendig, auf die bei relationalen Datenbanken üblichen Normalisierungen zu achten, um unnötige Redundanzen in den Daten zu vermeiden.

Bislang wird für die grafische Modellierung solcher Änderungen eine Software eingesetzt, die dann ein SQL Skript erzeugt, mit dem initial die Tabellenstruktur der relationalen Datenbank angelegt werden kann. Außerdem wird diese Software dafür verwendet, um ein PDF Dokument zur Dokumentation des Datenmodells zu erstellen.

Die eingesetzte Version dieser Software ist allerdings schon einige Jahre alt und ein Update auf die nächst höhere Version hat zu ungewünschten Nebeneffekten in den Daten geführt. Dieser Umstand und die Lizenzkosten, die ein breiterer Einsatz bei der MACH AG nach sich ziehen würde, führen dazu, dass die Modellierung von Änderungen am Datenmodell mittelfristig unabhängig von dieser Software durchgeführt werden soll.

### Aufgabenstellung

Ziel ist es, den Prototyp eines Werkzeugs zur Modellierung von Datenstrukturen für relationale Datenbanken zu erstellen. Diese Strukturen sollen in einer Graphendatenbank gespeichert und bearbeitet werden, um später Analysen auf Grundlage des Graphen durchführen zu können.

Im Einzelnen sind die folgenden Aufgaben zu erledigen:

- Es sollen die folgenden Elemente relationaler Datenbanksysteme in einer Graphendatenbank abgebildet werden:
    - Tabellen
    - Spalten
    - Primärschlüssel
    - Fremdschlüssel
    - Indizes
- Es soll eine Importfunktion geschaffen werden, die es erlaubt, o.g. Elemente aus einem SQL Skript in die Graphendatenbank zu überführen.
- Es soll eine Exportfunktion geschaffen werden, die es erlaubt, o.g. Elemente aus der Graphendatenbank in ein SQL Skript zu überführen.

- Es soll eine Benutzungsoberfläche erstellt werden, die es erlaubt, o.g. Elemente anzulegen, zu bearbeiten und zu löschen.
- Es sollen Prüfungen erstellt werden, die den Benutzer beim Bearbeiten auf Probleme hinweisen:
    - Als Namen der Elemente dürfen keine in SQL reservierten Wörter verwendet werden
    - Die Namen der Elemente dürfen eine vorher festgelegte Länge nicht überschreiten

- Für Namen der Elemente soll die Benutzungsoberfläche je nach Element Vorschläge nach einem bestimmten Schema machen:
    - Primärschlüssel: „XPK“ + <Tabellenname>
    - Fremdschlüssel: „R_“ + <laufende Nummer>
    - Indizes für Fremdschlüssel: „XIF“ + <laufende Nummer> + <Tabellenname>
    - Eindeutige Indizes: „XAK“ + <laufende Nummer> + <Tabellenname>
    - Einfache Indizes: „XIE“ + <laufende Nummer> + <Tabellenname>

Optional können noch die folgenden zusätzlichen Aufgaben erledigt werden:

- Es kann zusätzlich die Prüfung auf einheitliche Groß- und Kleinschreibung bei gleichlautenden Spaltennamen erstellt werden.
- Es kann zusätzlich implementiert werden, dass beim Anlegen einer Fremdschlüsselbeziehung zwischen zwei Tabellen die notwendigen Spalten (eventuell mit abweichenden Namen) in der Tabelle mit der Referenz angelegt werden.
- Es kann zusätzlich eine graphische Darstellung von zur Bearbeitung selektierten Teilen des Graphen in der Benutzungsoberfläche implementiert werden.

Professor Dr. Hinrichs

# II. ERKLÄRUNG ZUR BACHELORARBEIT

Ich versichere, dass ich die Arbeit selbstständig, ohne fremde Hilfe verfasst habe.

Bei der Abfassung der Arbeit sind nur die angegebenen Quellen benutzt worden. Wörtlich oder dem Sinne nach entnommene Stellen sind als solche gekennzeichnet.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, insbesondere dass die Arbeit Dritten zur Einsichtnahme vorgelegt oder Kopien der Arbeit zur Weitergabe an Dritte angefertigt werden.

Lübeck, den 28 Juni 2017

# III. DANKSAGUNG

Hiermit möchte ich mich bei meinem Betreuer bei der MACH AG Lars Ulrik Wollesen bedanken, der mich an das Thema der Bachelorarbeit herangeführt hat und mir während der Entwicklung der Software jederzeit mit Rat und Tat beiseite stand.

Für die Unterstützung bei der Teststrategie und den Tests möchte ich mich bei Christian Kram und Femke Paulik bedanken.

Auch meinen Prüfern Herrn Professor Holger Hinrichs und Herrn Professor Nane Kratzke möchte ich danken.

Zum Schluss möchte ich noch meinem Team an unermüdlichen Korrekturlesern danken: Constanze Tropf, Jutta Tropf, Sonja Beer, Erdmute Stanek, Daniel Johannsen, Antonia Berg, Michael Limbecker, Arne Salveter, Phillipp Ertelt, Melanie Nommensen und Simon Krause.

# IV. INHALTSVERZEICHNIS

# V. ABBILDUNGSVERZEICHNIS

# VI. TABELLENVERZEICHNIS

# 1 EINLEITUNG

## 1.1 MOTIVATION

Die MACH AG entwickelt seit 30 Jahren eine ERP-Software für den öffentlichen Sektor [1]. Diese in Java, JavaScript und Delphi geschriebene Software verwaltet ihre Daten in einer relationalen Datenbank, in der mittlerweile über 1000 Tabellen existieren. Es werden von der MACH-Software insgesamt vier verschiedene Datenbankmanagementsysteme unterstützt, die unterschiedlichen Regeln unterliegen. Zusätzlich ist es notwendig auf die bei relationalen Datenbanken üblichen Normalisierungen zu achten, um die Stabilität und Wartbarkeit der Datenbank zu gewährleisten.

Bisher wird für Änderungen im Datenmodell die Software „Erwin Data Modeler“ in einer älteren Version eingesetzt. Die Software ist in der Lage das Datenmodell in SQL-Anweisungen umzuwandeln, diese in eine Datei zu schreiben (SQL-Skript) und zusätzlich das Datenmodell in PDF-Form zu exportieren. Das generierte SQL-Skript kann anschließend in eines der vier Datenbanksysteme eingefügt werden, um dort die Datenbank zu erstellen, die von der MACH-Software genutzt werden soll. Inzwischen ist das exportierte PDF aus der Software durch die Menge der Informationen nicht mehr lesbar, weshalb dafür ein alternatives Tool verwendet wird. Ein Update auf eine neuere Version dieser Software würde zu unerwünschten Nebeneffekten wie Inkonsistenzen im Datenbestand führen. Zusätzlich ist der Einsatz dieser Software aufgrund von Lizenzkosten kostspielig, was einen breiteren Einsatz in der Firma verhindert.

Es gibt seit längerer Zeit Bestrebungen eine Software zu schaffen, die die alte Software ablöst. Hier wird eine selbstentwickelte Software bevorzugt, da diese an die Bedürfnisse der MACH AG angepasst werden kann. Durch die Größe und die ständige Erweiterung des Datenbankmodells sind Möglichkeiten zur Analyse der Datenbankstrukturen gewünscht. Die Idee ist, eine Graphendatenbank zu nutzen, die neue Möglichkeiten zur Analyse der Datenbank bietet. Die Analysemöglichkeiten sollen helfen, Probleme in der Datenbank zu finden und die Datenbank auf Optimierungspotenziale zu untersuchen.

## 1.2 ZIELE DER ARBEIT

Aus den oben genannten Gründen ist das Ziel dieser Arbeit das Entwickeln einer effektiven Abbildung des relationalen Datenbankschemas der MACH AG in einer Graphendatenbank. Hierbei soll eine sinnvolle Abbildung in der Graphendatenbank gefunden werden, die alle relevanten Informationen aus dem relationalen Datenbankmodell darstellt. Das Datenbankmodell besteht im Wesentlichen aus Tabellen, Tabellenspalten, Indizes und Fremdschlüsseln, die miteinander in Beziehung stehen. Die

einzelnen Eigenschaften dieser Elemente müssen abgebildet werden. Hierbei muss evaluiert werden, welche als Knoten und welche als Kanten in der Graphendatenbank abgebildet werden sollen.

Des Weiteren gehört zur Aufgabenstellung das Entwickeln einer Softwarelösung, die in der Lage ist, das bisherige Datenmodell der MACH AG in die Graphendatenbank zu überführen, zu bearbeiten und in einem SQL-Format aus der Datenbank zu extrahieren. Diese Software soll prototypisch umgesetzt werden, um zu zeigen, ob dieser Lösungsansatz langfristig als Ablösung der bisherigen Modellierungssoftware geeignet ist. Zudem soll die Software später als Grundlage für die Analyse und Performanceoptimierung des Datenbankmodells der MACH AG dienen.

Der Aufwand für die Entwicklung einer grafischen Darstellung der Graphendatenbank übersteigt den Projektrahmen und wird daher nicht umgesetzt. Auch die Prüfung der einheitlichen Schreibweise von Tabellennamen wird in dieser Arbeit aufgrund der zeitlichen Begrenzung nicht implementiert.

## 1.3 Gliederung der Arbeit

Dieses Dokument geht in Kapitel 2 auf die Grundlagen von relationalen Datenbanken, Graphendatenbanken und die Abfragesprachen „Structured Query Language“ (SQL) und „Cypher“ ein. Danach folgt im Kapitel 3 die Anforderungsanalyse der Software, die bei der MACH AG eingesetzt werden soll. Alle Produktfunktionen, die später in der Software enthalten sein sollen, werden hier genauer dargestellt. Das Kapitel 4 beschreibt den Aufbau des Graphen. Es beschreibt außerdem die Abbildung der Informationen aus dem relationalen Datenbankschema innerhalb des Graphen. Zudem werden die einzelnen Knoten mit ihren Beziehungen erläutert.

Die Kapitel 5 und 6 setzen sich mit der Konzeption und Realisierung der Software auseinander. Hier werden alle technischen Aspekte, die für die Implementierung relevant sind, näher beschrieben. Im Kapitel 7 werden anschließend die Teststrategie und die einzelnen Tests der Software näher erläutert. Die zu testenden Bereiche werden in diesem Kapitel in vier Quadranten mit eigenen Testzielen eingeteilt und mit entsprechenden Tests geprüft. Die Ergebnisse dieser Tests werden präsentiert, um die Funktionsweise der Software zu bestätigen. Das Kapitel 8 beschreibt einen zusammenfassenden Rückblick auf die Arbeit, sowie die Entwicklung der Anwendung. Danach beinhaltet Kapitel 9 einen Ausblick auf eventuelle Verbesserungs- und Erweiterungsmöglichkeiten der Software.

# 2 GRUNDLAGEN

Dieses Kapitel beschreibt die Funktionsweise von relationalen Datenbanken und Graphendatenbanken, sowie die Abfragesprache SQL für relationale Datenbanken und die Abfragesprache „Cypher“ für Graphendatenbanken.

## 2.1 RELATIONALE DATENBANKEN

### 2.1.1 Datenmodell

Anfang der Siebzigerjahre wurde durch Dr. Edgar Frank Codd die Idee des relationalen Datenbankmodells in einem Artikel beschrieben [2]. Umgangssprachlich wird Relation als „Tabelle“ übersetzt, was die Darstellungsform dieses Modells am besten beschreibt, da relationale Datenbanken Informationen in tabellarischer Form darstellen [3].

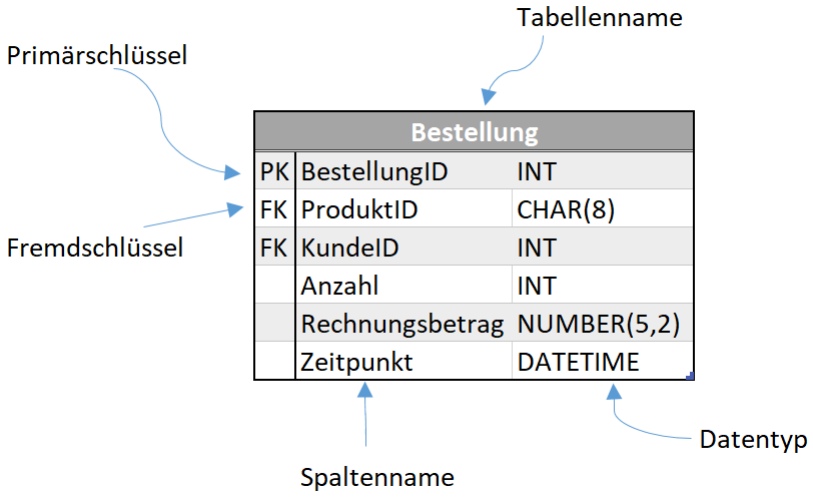


*Abbildung 1: Aufbau einer Tabelle in einer Datenbank*

Es gibt in diesen Datenbanken Tabellen mit Spalten (Abbildung 1). Jede Tabelle besitzt einen eindeutigen Tabellennamen. Jede Spalte hat einen vordefinierten Wertebereich, in dem die in der Datenbank gespeicherten Werte liegen können. Ein Datensatz (Tupel) wird in einer Zeile der Tabelle gespeichert. Die Reihenfolge der Zeilen und Spalten in der Tabelle spielt keine Rolle. Primärschlüssel werden benötigt, um Datensätze eindeutig identifizieren zu können und damit die Integrität des Datensatzes sicherzustellen [3]. Ist eine Spalte eindeutig für einen Datensatz, kann sie als Primärschlüssel verwendet werden. Es ist auch zulässig mehrere Spalten gemeinsam als Primärschlüssel zu verwenden. Wichtig ist, dass der Primärschlüssel minimal ist, damit er keine Informationen mehrfach oder Abhängigkeiten , die in der Datenbank zu inkonsistenten Zuständen

führen könnten [4]. Daher ist ein Primärschlüssel dann minimal, wenn er nicht reduziert werden kann, ohne seine Eindeutigkeit zu verlieren. [3]

In Datenbanken können Tabellen verknüpft werden, um zu vermeiden, dass identische Informationen mehrfach erfasst werden (Redundanz). Für diese Verknüpfungen werden Fremdschlüssel genutzt. Ein Fremdschlüssel verweist auf den Primärschlüssel einer fremden Tabelle und stellt eine Verknüpfung zu den Daten in der fremden Tabelle her [3]. Dieser Verweis wird Referenz genannt. Fremdschlüssel schützen die Integrität der Referenz des Datensatzes und müssen nicht eindeutig sein. Zum Beispiel können Bestellungen und Lieferanschriften in unterschiedlichen Tabellen verwaltet werden, sodass mehrere Bestellungen auf dieselbe Anschrift referenzieren. Dies bietet den Vorteil, dass Änderungen an der Anschrift nur in einem Datensatz erfolgen müssen. Ein Beispiel ist in Abbildung 2 zu sehen.

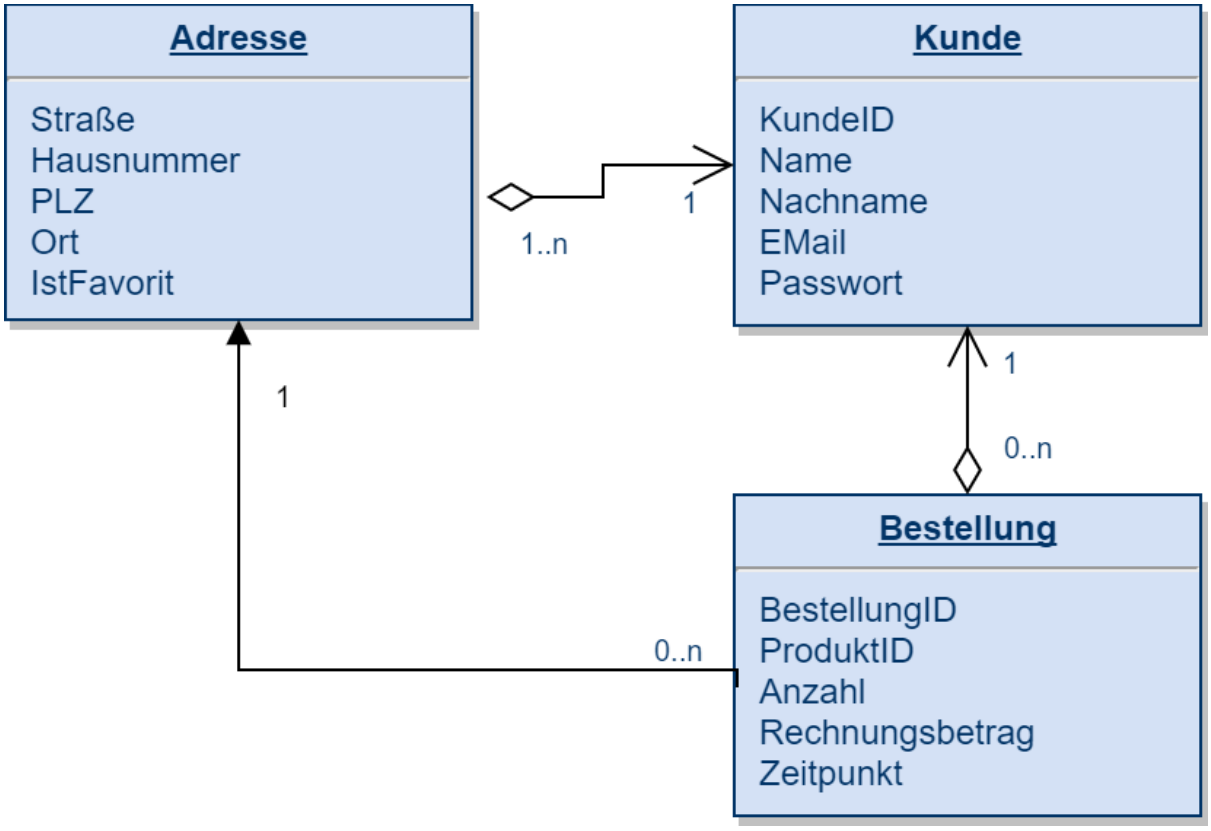


*Abbildung 2: Typische Zerlegung in einer relationalen Datenbank*

Um Redundanzen in Datenbanken zu vermeiden, wurden Normalformen als eine Art Designregeln definiert [3]. Normalformen gibt es in verschiedenen Stufen. So können Redundanzen und Inkonsistenzen in einer Datenbank verhindert und eine effiziente Datenhaltung gewährleistet werden.

Indizes werden in Datenbanksystemen eingesetzt, um den Zugriff auf Spalten in einer Datenbank zu beschleunigen. Aus diesem Grund werden in Spalten, in denen viele Such- oder Sortierzugriffe

stattfinden, Indizes erstellt. In relationalen Datenbanken wird für den Primärschlüssel automatisch ein Index erstellt [3].

Es gibt verschiedene Varianten von relationalen Datenbanken. Zum Beispiel Microsoft-SQL-Server, Oracle-Datenbanken, MySQL oder MariaDB. Diese Datenbanken haben zusätzliche herstellerspezifische Funktionen. Einige Datenbanken wie z.B. Oracle Datenbanken sind proprietär und kosten Lizenzgebühren, andere wie MariaDB sind unter einer Open Source Lizenz und können von jedem verwendet werden.

### 2.1.2 Strukturierte Abfragesprache SQL

SQL ist eine deklarative Programmiersprache (Abfragesprache), die zur Bearbeitung und Auswertung von relationalen Datenbanken genutzt wird [3]. Deklarativ bedeutet, dass das von der Datenbank gewünschte Resultat beschrieben wird und nicht die Art und Weise, wie die Informationen aus der Datenbank erhalten werden sollen [4]. SQL ist die Weiterentwicklung der Sprache SEQUEL, die Mitte der Siebzigerjahre für eines der ersten lauffähigen relationalen Datenbanksysteme geschaffen wurde. Die Sprache SQL ist durch das ANSI (American National Standards Institute) genormt. Trotzdem gibt es verschiedene SQL-Dialekte wie zum Beispiel MySQL oder Oracle-SQL. Da der offizielle Standard in vielen Systemen implementiert ist, hat dieser eine hohe Wiederverwendbarkeit und ist auf verschiedene Datenbanksysteme übertragbar [3] [4]. SQL ist die am weitesten verbreitetste Sprache für Datenbankabfragen und –interaktionen.

Eine einfache SQL-Anweisung (Statement) ist wie folgt aufgebaut [4]:

```
SELECT name
FROM Kunden
WHERE KundenID=1;
```

Diese Anweisung wählt (SELECT) das Merkmal „Name“ aus (FROM) der Tabelle „Kunden“ aus, mit der Bedingung, dass (WHERE) die *KundenID* den Wert eins hat. Es gibt drei Kategorien von Anweisungen:

- „Data Definition Language (DDL)“: enthält Anweisungen zum Erstellen, Aktualisieren und Löschen von Datenbanken, ihren Elementen und Strukturen [3]
- „Data Manipulation Language (DML)“: enthält Anweisungen zum Einfügen, Aktualisieren, Löschen und Lesen von Daten aus den Tabellen [3]
- „Data Control Language (DCL)“: enthält Anweisungen zum Verwalten der Datenbank [3]

## 2.2 GRAPHENDATENBANKEN

Graphendatenbanken werden zu den NoSQL-Datenbanken gezählt und basieren auf der Graphentheorie [5]. NoSQL steht für “Not Only SQL“ und ist ein Sammelbegriff für alle Datenbanken, die nicht auf Tabellen basieren und nicht SQL als Abfragesprache verwenden [4].

### 2.2.1 Graphentheorie

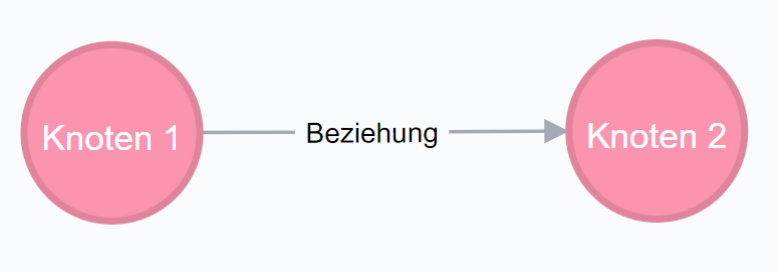


*Abbildung 3: Beispiel eines gerichteten Graphen mit zwei Knoten und einer gerichteten Kante*

Graphen bestehen aus Knoten und Kanten (Abbildung 3). Die Knoten bilden ein Objekt ab, das Informationen enthält und die Kanten verbinden Knoten miteinander [5]. Diese Kanten können eine Richtung besitzen (gerichtete Kanten oder Beziehungen), müssen es aber nicht. Gerichtete Kanten sind beispielsweise beim Abbilden von Einbahnstraßen in einem Straßennetz notwendig. In Eigenschaftsgraphen (engl. Property-Graph) besitzen die Knoten und Beziehungen des Graphens zusätzlich Namen und können verschiedene Eigenschaften zugeordnet bekommen [6]. Dies wird genutzt, um die Kanten zu gewichten. Es ist zum Beispiel möglich die Geschwindigkeit festzulegen, die man auf einer Straße, abgebildet durch eine Kante, fahren kann. Graphen sind besonders geeignet für die Abbildungen von vernetzten Daten, wie zum Beispiel Informationen aus sozialen Netzwerken. In Graphen können Personen als Knoten und ihre Verbindung miteinander über Beziehungen abgebildet werden [6]. Auch für das Abbilden von Straßennetzen und Lösen von Wegfindungsproblemen sind Graphen gut geeignet [6]. Es ist möglich einzelne Beziehungen im Graphen zu gewichten. Dies ist notwendig um den kürzesten oder längsten Weg aus der Datenbank abzufragen. Es gibt viele weitere Einsatzgebiete von Graphen und Graphendatenbanken wie zum Beispiel in der Biologie oder sozialen Studien [5].

### 2.2.2 Aufbau einer Graphendatenbank

Graphendatenbanken setzen zum Speichern von Informationen auf die Strukturen eines Eigenschaftsgraphen. Die Eigenschaften werden in der Graphendatenbank als Attribut-Wert-Paare (engl. Key-Value) gespeichert [4] [5]. Das bedeutet, dass Werte immer unter einem Schlüssel gespeichert werden und nur über diesen Schlüssel auf den gespeicherten Wert zugegriffen werden kann. In einer Graphendatenbank sind Kanten immer gerichtete Kanten. Knoten können mehrere

dieser gerichteten Kanten besitzen. Graphendatenbanken bieten genau wie Graphen große Vorteile beim Darstellen von vernetzten Daten, die über Knoten mit Beziehungen abgebildet werden können. Es ist möglich Algorithmen, die für Graphen entwickelt wurden, auch auf eine Graphendatenbank anzuwenden [5]. Im Vergleich zu den Beziehungen in Graphendatenbanken müssen diese Art von Informationen in einer relationalen Datenbank über komplizierte Verbundoperationen (engl. Joins) auf mehreren Tabellen abgebildet werden, was zu einer niedrigeren Abfragegeschwindigkeit führt [7].

### 2.2.3 Graphenorientierte Abfragesprache Cypher

Es gibt verschiedene Abfragesprachen für Graphendatenbanken. Die deklarative Abfragesprache Cypher wird für die Graphendatenbank „Neo4J“ genutzt [8]. Cypher kann Muster innerhalb einer Graphendatenbank beschreiben und abfragen. [8] Es werden Knoten und Beziehungen zwischen diesen Knoten beschrieben und anschließend dem Datenbanksystem übergeben. Das Datenbanksystem berechnet anschließend alle Knoten und Kanten, die in dieses Muster passen. [4]

```
MATCH (t:Table)-[r:foreignKey]->(i:Index)
WHERE t.name = "Reisen"
RETURN t,r,i;
```

Diese Beispielabfrage in Cypher beschreibt, dass der Knoten „Table“, mit der Bezeichnung „t“ eine Beziehung des Typs „foreignKey“ mit der Bezeichnung „r“ zu einem Knoten „Index“ mit der Bezeichnung „i“ hat. Dabei besitzt „t“ einen Namen. Von der Abfrage sollen beide Knoten und die Beziehung, die diese Beschreibung erfüllen, zurückgegeben werden. Zu sehen sind hier die drei typischen Grundelemente „Match“, „Where“ und „Return“ des Abfrageteils von Cypher. „Match“ dient zur Identifikation von Knoten, Beziehungen und zur Angabe von Suchmustern. „Where“ wird benutzt, um die Ergebnisse zu filtern. „Return“ stellt die Ergebnisse bereit und führt ggf. eine Zusammenfassung (Aggregation) der Daten durch [4] [8].

# 3 ANFORDERUNGSANALYSE

In diesem Kapitel werden die Anforderungen der MACH AG an die Software analysiert und daraus anschließend Anforderungen an die Software abgeleitet und erläutert. Danach wird der Entwurf der Software in Form eines Prototypenbaus erstellt und detailliert beschrieben.

## 3.1 SYSTEMUMFELD

Bei der MACH AG wurde für dieses Projekt der „Dell OptiPlex 9020“ als Referenzrechner herangezogen. Dieser Computer verwendet einen Intel-Core-i5-4570 mit 4 x 3,2 GHz als Prozessor und besitzt 32 GB RAM. Zusätzlich wird eine SSD als Festplatte verwendet. Als Betriebssystem dient „Windows 7 Professional“ in der 64 Bit Version. Alle Laufzeitangaben, die in den nächsten Kapiteln definiert werden, beziehen sich auf diese Hardware. Alle Tests, wie unter Anderem Performance-Tests werden auf dieser Hardware durchgeführt. Auf dem Computer stehen keine Administrator-Rechte zur Verfügung und deshalb muss die zu erstellende Software ohne Administrator-Rechte lauffähig sein. Java Development Kit 8 (JDK 8) ist auf diesem Computer installiert. Außerdem wird eine lokal laufende Installation der Graphendatenbank Neo4J vorausgesetzt, die exklusiv von der Software genutzt werden kann. Zusätzlich werden die Tests mit einer Oracle-Datenbank durchgeführt.

## 3.2 RELATIONALE DATENBANKEN BEI DER MACH AG

Bei der MACH AG werden insgesamt vier verschiedene Datenbankmanagementsysteme eingesetzt. Einige unterstützten zum Zeitpunkt der Einführung noch nicht alle SQL-Standards. Deshalb setzt die MACH AG ein vereinfachtes Schema ein, das auf ANSI-SQL basiert und von allen Datenbanken unterstützt wird. In diesem vereinfachten Schema werden zum Beispiel Funktionalitäten wie „Auto Inkrement“ und Standardwerte von Spalten nicht genutzt. Jede Tabelle hat einen eindeutigen Namen und ist in der Datenmanagement-Software einer Kategorie zugeordnet. Die Spalten (oder Felder) der einzelnen Tabellen enthalten Informationen wie Namen und Datentyp und die Information, ob dieser Wert Teil des Primärschlüssels, Teil eines Fremdschlüssels, einzigartig oder ein Pflichtfeld ist.

Es gibt in dem Datenbankmodell der MACH AG folgende verschiedene Index-Typen: Primärschlüssel, einfache Indizes, eindeutige Indizes und Indizes für Fremdschlüssel. Zusätzlich existieren auch noch Fremdschlüssel. Die Primärschlüssel verhalten sich bei der MACH AG wie im Kapitel 2.1.1 beschrieben. Sie müssen daher eindeutig sein und jede Tabelle muss einen Primärschlüssel besitzen. Indizes für Fremdschlüssel müssen einem Fremdschlüssel zugeordnet sein. Dass der Fremdschlüssel bei der MACH AG einen gesonderten Index zugeordnet bekommt, hat mit dem vereinfachten Datenbankmodell zu tun und wird von der MACH AG gefordert.

Neue Namen für Indizes und Fremdschlüssel werden anhand des Namenschemas generiert und gegebenenfalls manuell angepasst. Das Namenschema für Primärschlüssel besteht aus dem Kürzel „XPK“ gefolgt vom Tabellennamen. Einfache Indizes haben das Kürzel „XIE“ gefolgt von einer tabelleninternen Nummer und dem Tabellennamen. Eindeutige Indizes besitzen das Kürzel „XAK“ vor einer tabelleninternen Nummer und dem Tabellennamen. Indizes für Fremdschlüssel haben das Kürzel „XIF“ worauf eine Nummer folgt, die entweder tabellenintern ist oder die Nummer des Fremdschlüssels enthält. Anschließend wird der Tabellenname an den Namen angehängt. Fremdschlüssel besitzen das Kürzel „R_“ gefolgt von einer für alle Fremdschlüssel eindeutigen laufenden Nummer.

## 3.3 DDL Sprachumfang

Es gibt insgesamt vier verschiedene DDL-Anweisungs-Typen, die beim Import des SQL-Skriptes verarbeitet werden müssen und beim Export aus den Informationen der Graphendatenbank generiert werden. Die verwendeten Namen der einzelnen Elemente der Datenbank dürfen nicht über 18 Zeichen lang sein und nur aus Zahlen, Buchstaben und Unterstrichen bestehen. In den Namen sind SQL-Schlüsselwörter verboten.

Die erste Anweisung ist das „Create-Table-Statement“, das Tabellen mit Spalten erstellt. Es ist wie folgt aufgebaut:

```
CREATE TABLE <Tabelle> (
   <Feld>   <Typ>   NULL,
   <Feld>   <Typ>   NOT NULL
);
```

<Tabelle> ist der einzigartige Name der jeweiligen Tabelle. <Feld> ist der Name der Spalten innerhalb der Tabelle. Unter <Typ> wird einer der folgenden möglichen Datentypen eingefügt:

- INTEGER: Integer ist der Datentyp für Ganzzahlen.
- NUMBER(INT, INT): „Number“ ist eine Zahl mit einem Komma. Die Anzahl der Stellen vor und nach dem Komma sind in den Klammern nacheinander als Integer-Zahlen angegeben.
- VARCHAR2(INT): Hier können Zeichenketten gespeichert werden. Die Anzahl der Zeichen wird von der Ganzzahl am Ende beschrieben (INT).
- DATE: Dieser Datentyp repräsentiert ein Datum.
- SMALLINT: Hier kann eine Ganzzahl aus einem relativ kleinen Wertebereich (2 Bytes) angegeben werden.
- LONG RAW: Dies ist ein Datentyp, der nur in Oracle-Datenbanken existiert. Hier können bis zu zwei Gigabyte Binärdaten angegeben werden.

- CHAR(INT): Es können Zeichen angegeben werden. Die Anzahl der Zeichen wird von der Zahl in Klammern angegeben.
- BLOB: In diesem Datentyp werden Binärdaten gespeichert.
- CLOB: In diesem Datentyp werden Texte gespeichert.

Die zweite Anweisung erstellt einen Index und folgt nach der Anweisung zum Erstellen der Tabelle:

```
CREATE (UNIQUE) INDEX <Index> ON <Tabelle> (
   <Feld>   ASC,
   <Feld>   ASC
);
```

<Index> ist der Name des erstellten Index. Die Anweisung enthält ein optionales „Unique“ nach dem Create. Das „Unique“ macht den Index zu einem einzigartigen Index. Je nachdem was für Informationen abgebildet werden sollen, ist entweder ein normaler oder ein eindeutiger Index notwendig. Ein einzigartiger Index darf keine zwei Datensätze (Zeilen) haben, die gleiche Werte enthalten. Bei einem normalen Index ist es erlaubt, dass sich zwei Datensätze gleichen. Bei der MACH AG wird keine absteigende Spaltensortierung durch die Anweisung „DESC“ verwendet. Daher kommt hier nur die Anweisung „ASC“ für eine aufsteigende Spaltensortierung zum Einsatz. Bei allen Indizes (inklusive Primärschlüssel), ist die Reihenfolge der Felder im Gegensatz zur Tabelle relevant für die Eindeutigkeit des Indexes.

Die dritte Anweisung erzeugt einen Primärschlüssel und befindet sich hinter der Anweisung zum Erstellen der Tabelle:

```
ALTER TABLE <Tabelle>
       ADD   ( CONSTRAINT <Primärschlüssel> PRIMARY KEY (<Feld>,
               <Feld>, <Feld>)
);
```

<Primärschlüssel> ist der Name des Primärschlüssels und erstreckt sich über drei Spalten.

Die letzte Anweisung zum Erstellen des Fremdschlüssels erfolgt immer am Ende des SQL-Skriptes, nachdem alle Tabellen erzeugt wurden. Die Anweisung orientiert sich bei der Angabe der Felder an der Reihenfolge des Primärschlüssels der referenzierten Tabelle. So ist die eindeutige Zuordnung der Felder möglich. Die Feldreihenfolge muss nicht mit der Reihenfolge des zugehörigen Indexes übereinstimmen. Zusätzlich kann der Index auch der Primärschlüssel der eigenen Tabelle sein. In diesem Fall existiert kein spezieller Index für den Fremdschlüssel.

```
ALTER TABLE <Tabelle>
       ADD   ( CONSTRAINT <Fremdschlüssel>
               FOREIGN KEY (<Feld>, <Feld>)
                           REFERENCES <Referenz Tabelle> ) ;
```

<Fremdschlüssel> ist der Name des Fremdschlüssels. <Referenz Tabelle> ist der Name der Tabelle, auf die der Fremdschlüssel zeigt.

## 3.4 PRODUKTFUNKTIONSANALYSE

Dieses Unterkapitel beschreibt die Ableitung der Produktfunktionen aus der Aufgabenstellung. Laut dieser soll in einer Graphendatenbank eine Abbildung des relationalen Datenbankmodells der MACH AG umgesetzt werden. Zusätzlich soll eine Software zur Verwaltung der Graphendatenbank geschrieben werden. Die Software soll in der Lage sein, ein SQL-Skript in die Graphendatenbank zu importieren und aus der Graphendatenbank in ein SQL-Skript zu exportieren. Die Software soll Informationen aus der Graphendatenbank anzeigen und bearbeiten können. Des Weiteren soll sie sicherstellen, dass weder reservierte SQL-Wörter in Namen verwendet werden, noch dass diese Namen eine definierte Länge überschreiten. Für neue Elemente anhand eines vorgegebenen Schemas sollen neue Namen generiert werden. Von der MACH AG wird erwartet, dass diese Software für den Import der Daten nicht unverhältnismäßig lange braucht. Da der Import nur einmalig beim Einrichten der Datenbank durchgeführt wird, werden 10 Minuten als akzeptable Zeit für den Import angesehen. Das Starten der Software, das Verbinden mit der Datenbank und der Export von Informationen sollte nicht länger als 30 Sekunden auf dem Referenzrechner, der in der Entwicklung zur Verfügung steht, dauern.

Aus dieser Beschreibung ergeben sich folgende Produktfunktionen (PF) für die Anwendung:

- PF 1: Das Abbilden folgender Elemente aus einem relationalen Datenbanksystem in eine Graphendatenbank:
    - Tabellen
    - Spalten
    - Primärschlüssel
    - Fremdschlüssel
    - Indizes
- PF 2: Eine Importfunktion, welche die in PF 1 genannten Elemente aus einem SQL-Skript in die Graphendatenbank überführt
- PF 3: Eine Exportfunktion, die die in PF 1 genannten Elemente aus der Graphendatenbank in ein SQL-Skript überführt
- PF 4: Eine Funktion zum Anlegen, Bearbeiten und Löschen von allen in PF1 genannten Elementen in der Graphendatenbank

- PF 5: Alle der in PF 1 genannten Elemente dürfen keine in SQL reservierten Wörter beinhalten und eine definierte Länge nicht überschreiten
- PF 6: Es werden für neue Primärschlüssel, Fremdschlüssel und Indizes Namensvorschläge auf Basis eines, in der Aufgabenstellung definierten, Schemas erstellt
- PF 7: Das Starten der Software, Importieren, Exportieren und Laden aus der Datenbank muss für die Nutzer in einem akzeptablen Zeitraum geschehen.

## 3.5 Software Anforderungsanalyse

Aus den vorherigen Produktfunktionen lassen sich folgende konkrete Anforderungen (engl. Requirement oder kurz Req.) an die Software ableiten:

- [Req01] Eine Software, die sich mit der Graphendatenbank verbinden lässt, kann die Daten daraus laden und diese verändert abspeichern.
- [Req02] Ein SQL-Parser, der das SQL-Skript, im von der MACH AG genutzten Format, verarbeiten kann.
- [Req03] Das Speichern des geparsten SQL-Skripts in der Graphendatenbank innerhalb von zehn Minuten.
- [Req04] Das Ausgeben des Datenbankmodells in SQL-Form innerhalb von 30 Sekunden.
- [Req05] Das Darstellen von Informationen, die eine Tabelle aus dem relationalen Datenbankschema enthält.
- [Req06] Eine Suchfunktion, um Tabellen mit Autovervollständigung auswählen zu können.
- [Req07] Eine Ansicht, in der alle Tabellen angezeigt und ausgewählt werden können.
- [Req08] Das Anlegen, Bearbeiten und Löschen von Tabellen.
- [Req09] Das Anlegen, Bearbeiten und Löschen von Feldern.
- [Req10] Das Anlegen, Bearbeiten und Löschen von Indizes.
- [Req11] Das Ändern der Beziehung zwischen einem Index und den zugeordneten Feldern.
- [Req12] Das Anlegen, Bearbeiten und Löschen von Fremdschlüsseln.
- [Req13] Das Ändern der Beziehung zwischen einem Fremdschlüssel und den zugeordneten Tabellen.
- [Req14] Das Anpassen der Zuordnung zwischen den Feldern des Fremdschlüssels und den Feldern der verknüpften Tabelle.
- [Req15] Das automatische Erstellen von Feldern und Indizes eines neuen Fremdschlüssels.
- [Req16] Änderungen in der grafischen Benutzeroberfläche (GUI) sollen erst nach betätigen einer „Speichern“ -Schaltfläche in der Datenbank gespeichert werden.

- [Req17] Das Prüfen von Tabellen-, Feld-, Index- und Fremdschlüsselnamen auf SQL reservierte Wörter.
- [Req18] Das Prüfen von Tabellen-, Feld-, Index- und Fremdschlüsselnamen auf das Überschreiten einer bestimmten Länge.
- [Req19] Das Darstellen von Fehleingaben des Nutzers, die nicht [Req17] und [Req18] erfüllen.
- [Req20] Automatische Benennung von neuen Primärschlüsseln nach dem vorgegebenen Schema.
- [Req21] Automatische Benennung von neuen Indizes nach dem vorgegebenen Schema.
- [Req22] Ermittlung von neuer laufender Nummer für Fremdschlüssel.
- [Req23] Automatische Benennung von Fremdschlüsseln mit eindeutiger laufender Nummer.
- [Req24] Das Starten der Anwendung und Laden der Datenbank innerhalb von 30 Sekunden.
- [Req25] Der Importvorgang dauert nicht länger als 10 Minuten.

## 3.6 Zuordnung der Anforderungen zu Produktfunktionen

In diesem Unterkapitel werden die Produktfunktionen zur besseren Übersicht den Anforderungen gegenübergestellt.

| Anforderung | PF1 | PF2 | PF3 | PF4 | PF5 | PF6 | PF7 |
|---|---|---|---|---|---|---|---|
| **Req01 – Datenbank Verbindung** | X | | | | | | |
| **Req02 – SQL-Parser** | | X | | | | | X |
| **Req03 –SQL-Importer** | | X | | | | | |
| **Req04 – SQL-Export** | | | X | | | | X |
| **Req05 – Darstellung Tabelle** | | | | X | | | |
| **Req06 – Suchfunktion** | | | | X | | | |
| **Req07 – Anzeigen aller Tabellen** | | | | X | | | |
| **Req08 – Tabellen bearbeiten** | | | | X | | | |
| **Req09 – Felder bearbeiten** | | | | X | | | |
| **Req10 – Indizes bearbeiten** | | | | X | | | |
| **Req11 – Index Beziehung bearbeiten** | | | | X | | | |
| **Req12 – Fremdschlüssel (FK) bearbeiten** | | | | X | | | |
| **Req13 – FK Beziehung bearbeiten** | | | | X | | | |
| **Req14 – FK Feld Zuordnung** | | | | X | | | |
| **Req15 – FK Auto Index + Felder** | | | | X | | | |
| **Req16 – DB nach Speichern ändern** | | | | X | | | |
| **Req17 – Prüfung Namen auf SQL** | | | | | X | | |
| **Req18 – Prüfung Name auf Länge** | | | | | X | | |
| **Req19 – Fehleingabe anzeigen** | | | | X | X | | |
| **Req20 – Auto Name Schlüssel** | | | | | | X | |
| **Req21 – Auto Name Indizes** | | | | | | X | |
| **Req22 – Ermittlung FK Nummer** | | | | | | X | |
| **Req23 – Auto Name FK** | | | | | | X | |
| **Req24 – Starte & Lade DB in 30 Sek** | | | | | | | X |
| **Req25 – Import innerhalb 10 Min** | | | | | | | X |

*Tabelle 1: Zuordnung Anforderungen zu Produktfunktionen*

## 3.7 Entwurf der grafischen Benutzeroberfläche

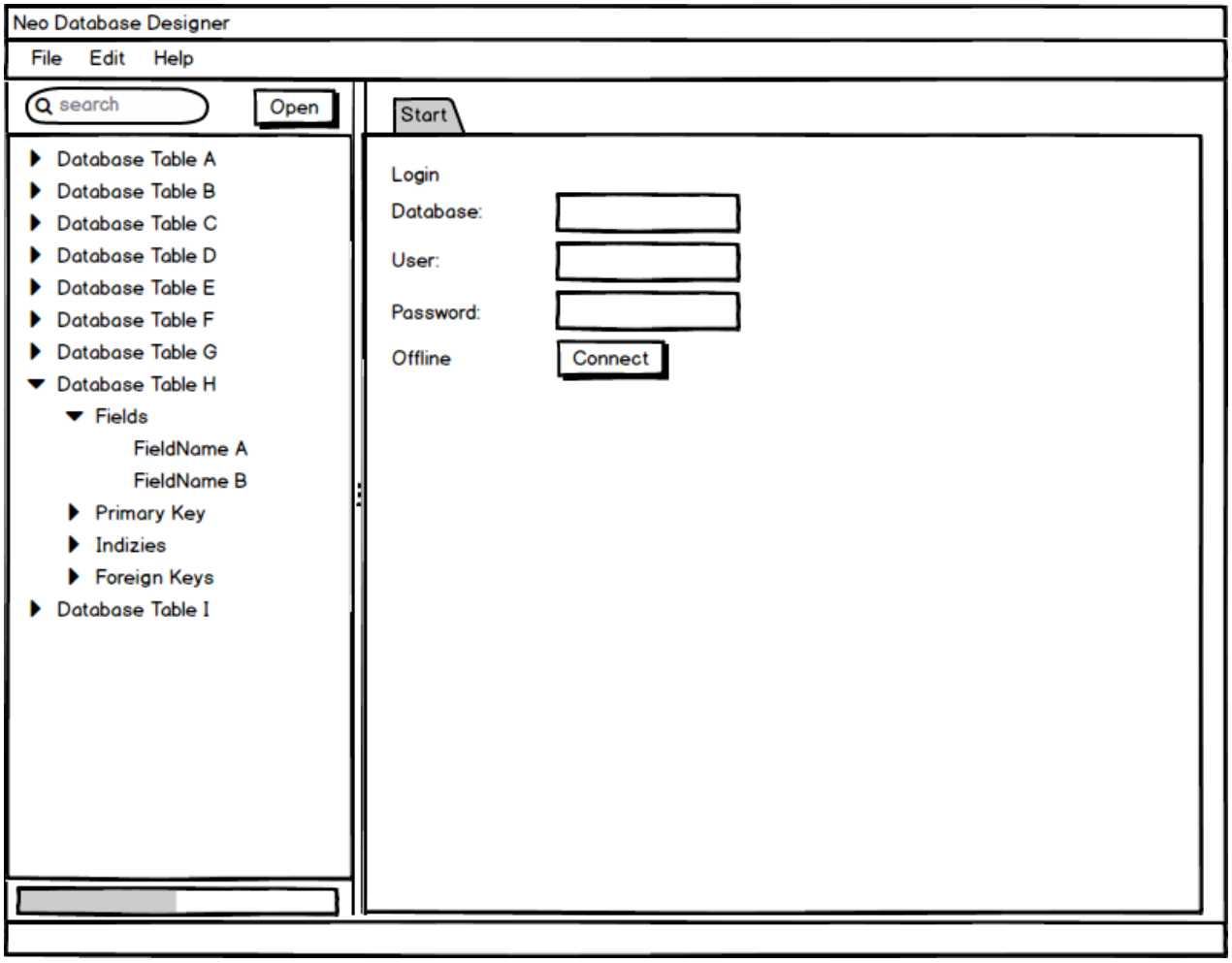


*Abbildung 4: Mockup Software Hauptfenster mit Login*

Nachdem die Software gestartet wird, erscheint das Hauptfenster (Abbildung 4). Das Hauptfenster ist unterteilt in verschiedene Bereiche. Oben befindet sich eine Menüleiste, über die verschiedene Funktionen wie zum Beispiel die Import- oder Export-Funktion erreicht werden können. Auf der linken Seite sind ein Suchfeld und eine Baumansicht zu sehen, in der alle Tabellen aufgelistet sind. Auf der rechten Seite befindet sich eine Detailansicht mit einer Registerkartenansicht, in dem es möglich ist Tabellen detaillierter anzuschauen und anzupassen.

Im Suchfeld kann ein beliebiger Tabellenname eingetippt werden und es werden Vervollständigungsvorschläge von der Software gemacht. Nachdem ein kompletter Tabellenname eingegeben wurde, kann per „Suchen“-Schaltfläche oder über die „Enter“-Taste die Detailansicht der ausgewählten Tabelle geöffnet werden. Diese Tabellen in der Baumansicht können per Klick auf das entsprechende Dreieck vor dem Text ausgeklappt werden. Hier ist es möglich, mit einem Klick auf die Felder, Indizes oder Fremdschlüssel, Informationen zur Tabelle zu sehen. Mit einem Klickt auf die Tabelle, öffnet sich auf der rechten Seite die ausgewählte Tabelle in einer Detailansicht. Es ist ebenso

möglich, auf einen Fremdschlüssel zu klicken und so die verknüpfte Tabelle des Fremdschlüssels in der Detailansicht zu öffnen.

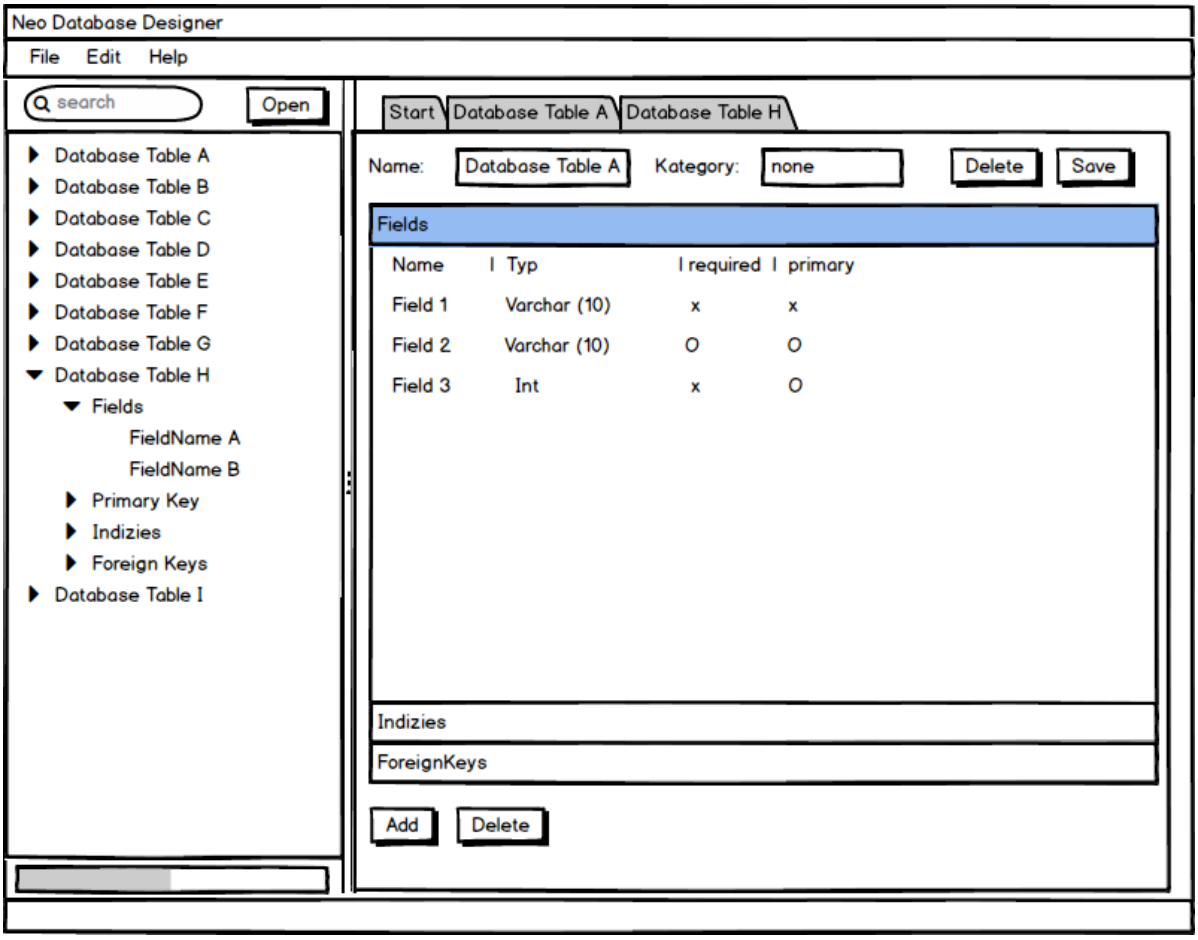


*Abbildung 5: Mockup Software Hauptfenster*

Die Detailansicht auf der rechten Seite, die in Abbildung 5 zu sehen ist, enthält eine Registerkartenansicht, in der sich beim Start der Software ein Login für die Graphendatenbank befindet. Nachdem eine Tabelle über die Suche oder die Baumansicht ausgewählt wurde, öffnet sich eine neue Registerkarte mit den Informationen zu dieser Tabelle. Über die Menüleiste kann eine neue Tabelle eingefügt werden. Es öffnet sich anschließend eine Registerkarte in der Detailansicht, in der sich die neue Tabelle befindet. In dieser Tabellenansicht kann man Namen und Kategorie der Tabelle ändern und speichern oder die Tabelle löschen. Zusätzlich werden in einer Akkordeon-Ansicht jeweils die Felder, Indizes und Fremdschlüssel als einzelne Seiten angezeigt. Diese enthalten jeweils eine Tabelle, in der die Daten bearbeitet werden können. Alle Änderungen innerhalb der Tabellen-Registerkarte werden nicht automatisch gespeichert und es ist jederzeit möglich, durch Schließen der Registerkarte alle Änderungen zu verwerfen. Erst nachdem der „Speichern“ Knopf gedrückt wurde und die Tabelle zusätzlich die interne Prüfung auf Fehler bestanden hat, werden die Änderungen in die Graphendatenbank geschrieben. Wurde in der internen Prüfung ein Fehler gefunden, wird dieser

an den Nutzer in Form einer Fehlermeldung ausgegeben. Dies würde zum Beispiel passieren, wenn eine Tabelle einen zu langen Namen bekommen hat. Vor dem Löschen einer Tabelle folgt eine Abfrage, ob der Nutzer dies wirklich will. Bestätigt der Nutzer diesen Dialog, wird die Tabelle unwiderruflich gelöscht.

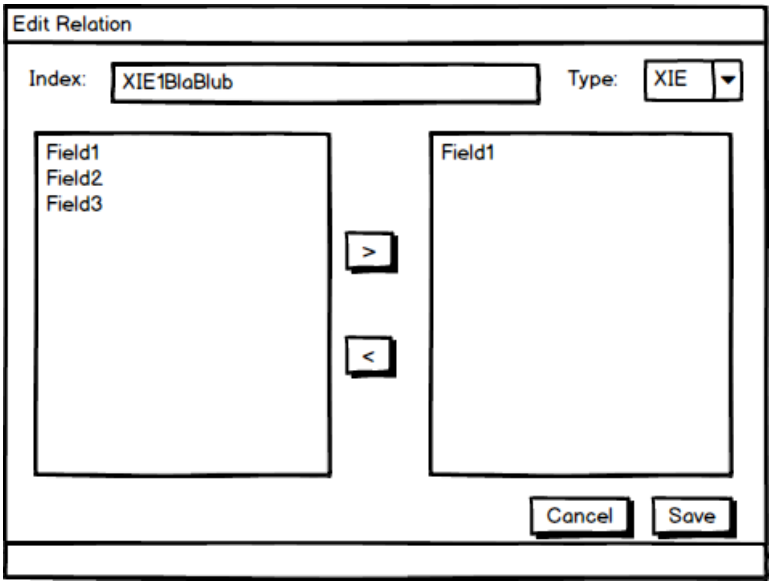


*Abbildung 6: Mockup Index Beziehung bearbeiten*

Um die Beziehungen zwischen Feldern und Indizes und den Index im Allgemeinen zu bearbeiten, gibt es zusätzlich die Möglichkeit einen Dialog zu öffnen (Abbildung 6). In diesem Dialog kann der Name des Indexes über ein Feld bearbeitet werden. Der Typ des Indexes kann über ein Dropdown-Feld festgelegt werden. Der Dialog hat auf der linken Seite ein Fenster mit allen Feldern, die in der Tabelle zur Verfügung stehen. Auf der rechten Seite werden alle Felder des Indexes aufgelistet. Mithilfe von zwei Schaltflächen ist es möglich, Felder auf der rechten Seite hinzuzufügen oder zu entfernen. Der Dialog bietet eine Schaltfläche, um die gemachten Änderungen zu verwerfen und eine Schaltfläche, welche die Änderungen in die vorherige Ansicht übernimmt, aber nicht in der Datenbank speichert. Die Änderungen werden erst mit dem Speichern der ganzen Tabelle in die Datenbank geschrieben.

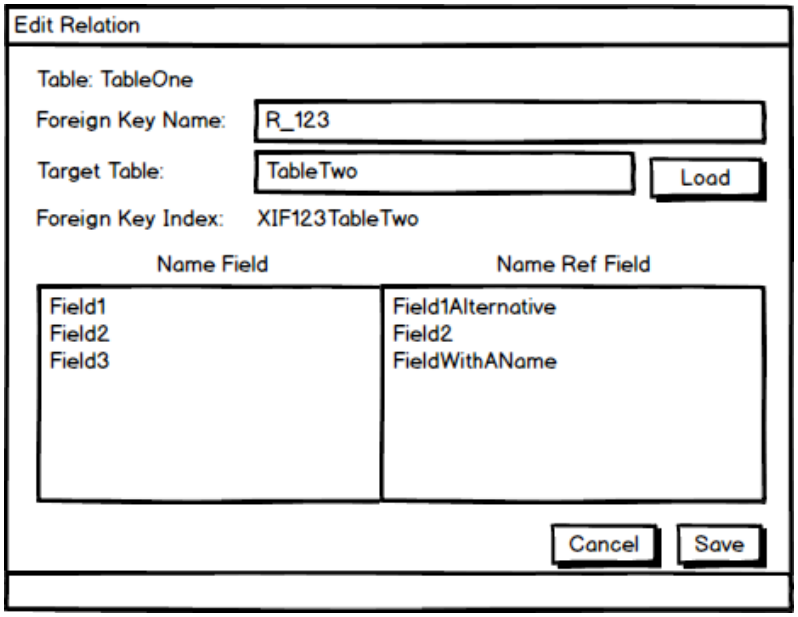


*Abbildung 7: Mockup Fremdschlüssel Beziehung bearbeiten*

Für den Fremdschlüssel gibt es ebenfalls einen Dialog zum Bearbeiten (Abbildung 7). In diesem Dialog kann zusätzlich die Zieltabelle ausgewählt werden, mit dem diese Tabelle verknüpft werden soll. Neben einem Feld zum Bearbeiten des Fremdschlüsselnamens hat dieser Dialog ein Feld, in dem die Zieltabelle eingegeben werden kann. Dieses Feld bietet eine Autovervollständigung für den Namen der Zieltabelle. Nachdem man auf eine „Laden“-Schaltfläche klickt, werden die Informationen in einer Tabelle im unteren Bereich des Dialoges geladen. Hier ist es möglich, die Namen der Felder, die dem Fremdschlüssel zugeordnet sind, zu ändern. Zusätzlich kann man die Namen der verknüpften Felder aus dem Primärschlüssel der referenzierten Tabelle sehen.

# 4 AUFBAU DES GRAPHEN

Dieses Kapitel beschreibt den Aufbau des Graphen in der Graphendatenbank. Dabei wird auf die drei Knotentypen (`Table`, `Field` und `Index`) und die sechs Beziehungen (`CONNECTED`, `DATA`; `FOREIGNKEY`; `INDEX`; `REFERENCE` und `XPK`) eingegangen. Ein Beispiel der Graphendatenbank ist in Abbildung 8 zu sehen. Es wird detailliert beschrieben, wie die Informationen aus dem relationalen Datenbankschema im Graphen verwendet werden und warum sich für die jeweilige Abbildungsform entschieden wurde.

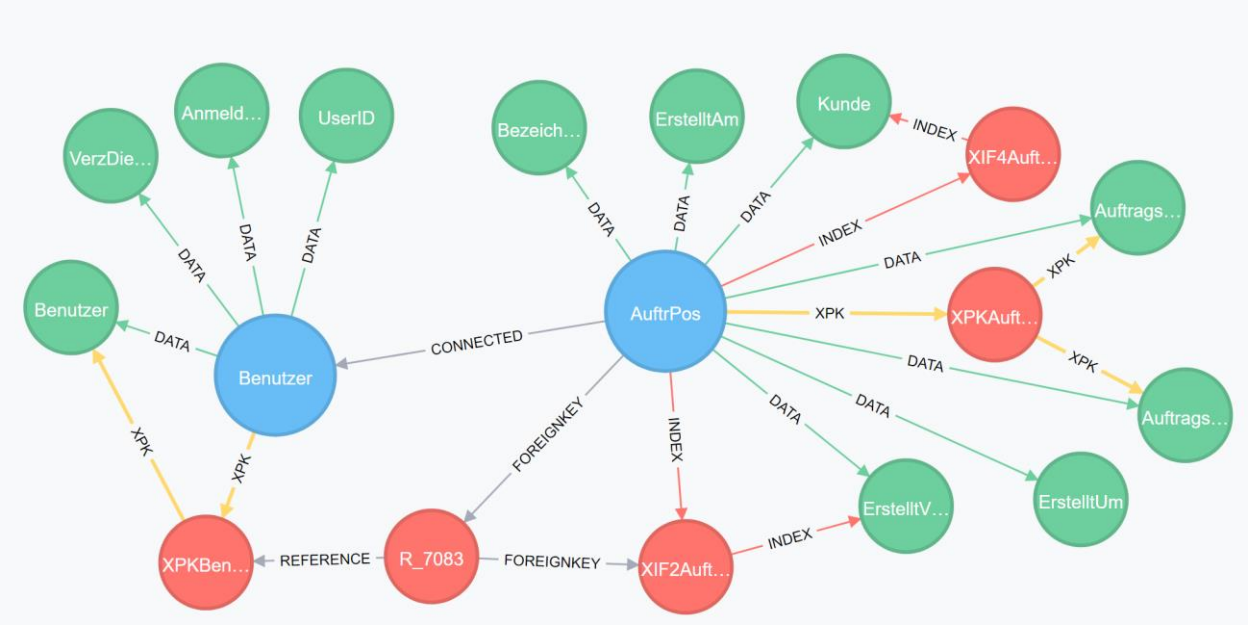


*Abbildung 8: Graphendatenbank Überblick*

## 4.1 KNOTEN

Tabellen, Spalten und Indizes aus dem relationalen Datenbankmodell werden im Graphendatenbankmodell als Knoten abgebildet. Da Primärschlüssel und Fremdschlüssel von der Funktionsweise Indizes ähneln, werden sie in der Graphendatenbank als solche behandelt. Die Unterscheidung erfolgt über ein Feld `type` innerhalb des Knotens und der Beziehung zum Tabellenknoten.

### 4.1.1 Der Tabellenknoten


*Abbildung 9: Tabellenknoten*

Der Tabellenknoten (`Table`) in Abbildung 9 repräsentiert eine Tabelle im relationalen Datenbankschema. Er hat die Eigenschaften `name` und `category`. Der Name ist der Tabellenname im relationalen Datenbankschema. Die Kategorie ordnet die Tabelle in einem Bereich der MACH Software ein. Sie soll zur Unterteilung der einzelnen Bereiche des Datenbankschemas dienen, was für den späteren Ausdruck des Datenbankmodells wichtig ist.

Da bei der Software über den eindeutigen Tabellennamen gesucht wird, ist aus Performancegründen das Feld `name` in `Table` indiziert. Dies hat den Vorteil, dass Abfragen auf den Namen einer Tabelle deutlich schneller ausgeführt werden können als zum Beispiel auf die Kategorie der Tabelle oder auf Namen von Feldern.

### 4.1.2 Der Feldknoten

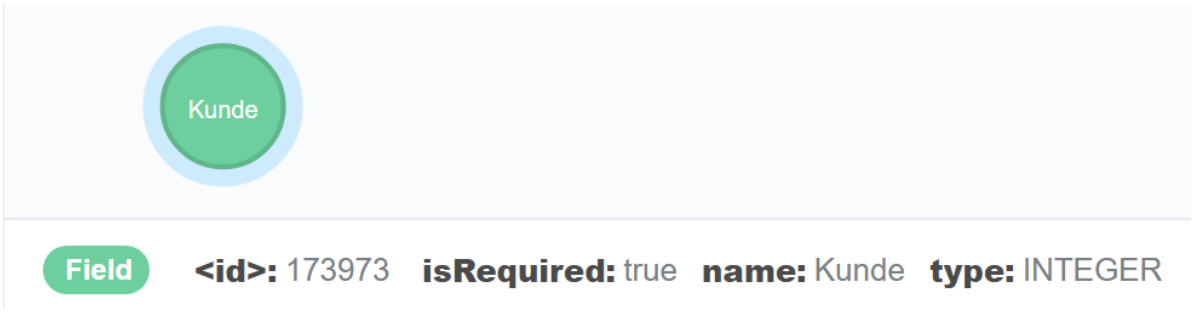


*Abbildung 10: Feldknoten*

Der Feldknoten (`Field`) in Abbildung 10 repräsentiert eine Spalte im relationalen Datenbankschema. Er hat die Eigenschaften `name`, `fkname`, `type` und `isRequired`. `name` ist der Spaltenname im relationalen Datenbankschema. `type` ist eine Textrepräsentation des Datentyps, den dieses Feld besitzt. `isRequired` beschreibt die Eigenschaft, ob das Feld ein Pflichtfeld ist. Es hat den Wert TRUE, wenn es erforderlich ist, und FALSE, wenn es nicht erforderlich ist.

Beim Entwurf dieses Feldes gab es die Überlegung, alle Datentypen, die ein Feld haben kann, als eigene Knoten in der Graphendatenbank darzustellen. Datentypen können eine Beziehung zu den Feldern haben, die sie verwenden und so konsistente Datentypenverwendung gewährleisten. Es wäre möglich, eine allgemeine Bezeichnung wie „Ganzzahl“ oder „Zeichenkette“ als Datentyp einzuführen. Diese allgemeine Bezeichnung wird nicht an einer bestimmten Datenbank festgemacht. Allerdings gibt es im Datenbankmodell verschiedene Datentypen mit einer spezifischen Größe. Konkret bei den Datentypen VARCHAR, CHAR und NUMBER gibt es verschiedene Größen, die alle als eigene Knoten hätten erfasst werden müssen. Das Zusammenfassen vieler ähnlicher Größen könnte zu unerwünschten Nebeneffekten führen, wie zum Beispiel Zugriffsfehler innerhalb der MACH Software, deutlich größeren Speicherverbrauch in der Datenbank oder das Verfälschen von Werten. Diese Fehler zu finden und zu korrigieren würde den Arbeitsaufwand dieses Projektes übersteigen. Deshalb wurde sich für die Variante entschieden, wo das Typ-Feld lediglich als Textfeld genutzt wird.

### 4.1.3 Der Indexknoten

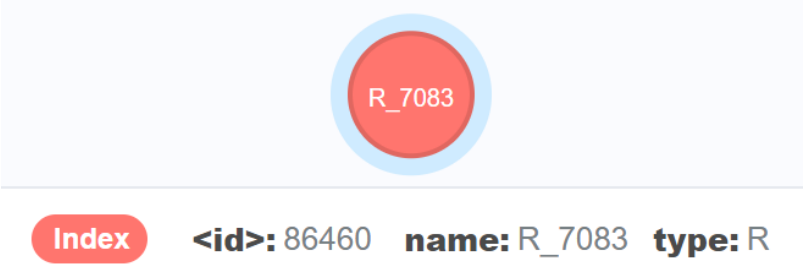


*Abbildung 11: Indexknoten*

Der Indexknoten (`Index`) in Abbildung 11 repräsentiert einen Index im relationalen Datenbankschema. Er hat die Eigenschaften `name` und `type`. `name` ist der Indexname im relationalen Datenbankschema. `type` dient zur Unterscheidung der verschiedenen Indextypen.

Aus der Analyse im Kapitel 3.3 ergeben sich folgende Indextypen:

1. Primärschlüssel: „XPK“
2. Indizes für Fremdschlüssel: „XIF“
3. Eindeutige Indizes: „XAK“
4. Einfache Indizes: „XIE“
5. Fremdschlüssel: „R“

Die Eigenschaft, ob ein Index einzigartig ist oder nicht, ergibt sich aus dem Typ des Indexes.

## 4.2 Beziehungen

Die Beziehungen zwischen den Knoten sind wichtig, um darzustellen, wie die Knoten miteinander in Verbindung stehen. Hierbei wird nicht nur die Verbindung zwischen Tabellen untereinander abgebildet, wie es für den Fremdschlüssel notwendig ist, sondern auch die Verbindung zwischen Spalten der Tabelle und Indizes innerhalb einer Tabelle. Auch die Zuordnung zu einer Tabelle wird über verschiedene Beziehungen abgebildet.

### 4.2.1 Die Beziehung DATA

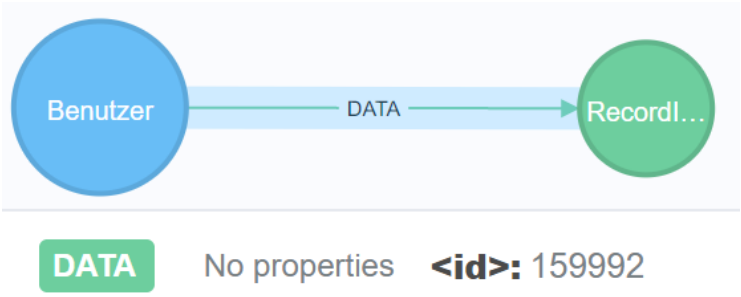


*Abbildung 12: Beziehung DATA*

Die Beziehung `DATA` in Abbildung 12 bildet die Zuordnung von einzelnen Spalten zu den zugehörigen Tabellen im relationalen Datenmodell ab. Die Beziehung wird als eine Kante, die vom Tabellenknoten auf einen Feldknoten zeigt, abgebildet.

### 4.2.2 Die Beziehung XPK

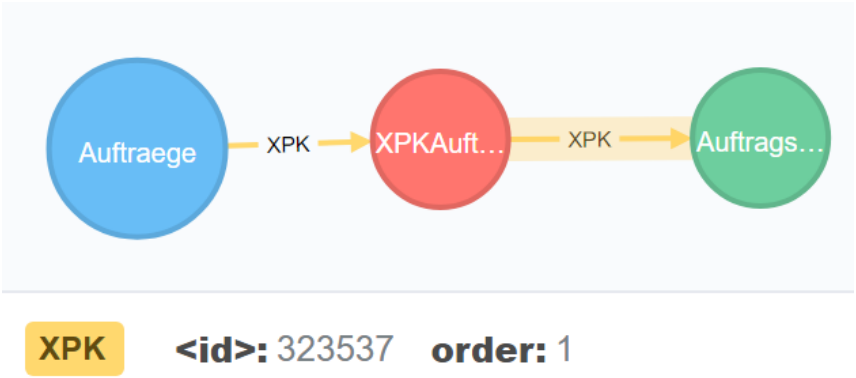


*Abbildung 13: Beziehung XPK*

Die Beziehung `XPK`, die in Abbildung 13 zu sehen ist, bildet die Zuordnung vom Primärschlüssel zu einer Tabelle und von einzelnen Spalten zu einem Primärschlüssel im relationalen Datenmodell ab. Die Beziehung wird als eine Kante dargestellt, die von einem Tabellenknoten auf einen Indexknoten des Typs `XPK` zeigt. Außerdem zeigen Kanten von dem Indexknoten auf die Feldknoten, die Teil des Primärschlüssels sind. Eine gültige Tabelle kann nur einen Primärschlüssel haben. Aus diesem Grund

kann die Beziehung von der Tabelle zum Index nur einmal existieren. Ein Index kann aber eine Beziehung zu mehreren Feldknoten haben. Zudem hat die Beziehung zwischen Primärschlüssel und Feld eine `order`-Eigenschaft. Diese Eigenschaft legt die Reihenfolge der Elemente fest, die einem Index zugeordnet sind.

### 4.2.3 Die Beziehung INDEX

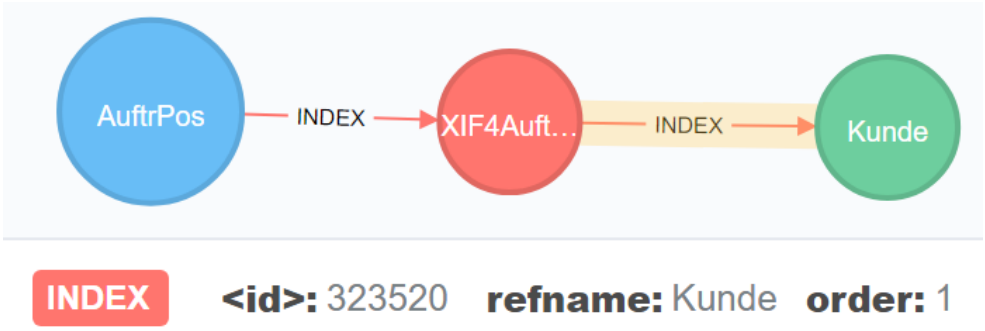


*Abbildung 14: Beziehung INDEX*

Die Beziehung `INDEX`, die in Abbildung 14 zu sehen ist, bildet die Zuordnung von Indizes zu einer Tabelle und von einzelnen Spalten zu einem Index im relationalen Datenmodell ab. Die Beziehung wird als eine Kante, die von einem `Table`-Knoten auf einen `Index`-Knoten des Typs XIF, XAK oder XIE zeigt, abgebildet. Kanten zeigen vom Indexknoten auf die Feldknoten, die Teil des Indexes sind. Hier hat nur die Beziehung zwischen dem Index und dem Feld eine `order`-Eigenschaft, welche die Reihenfolge der Elemente festlegt, die dem Index zugeordnet sind. Zusätzlich gibt es die optionale Eigenschaft `refname`, die nur bei Indizes von Fremdschlüsseln verwendet wird. Hier wird der Name des Feldes angegeben, das dieses Feld als Teil des Fremdschlüssels repräsentiert.

### 4.2.4 Die Beziehung FOREIGNKEY

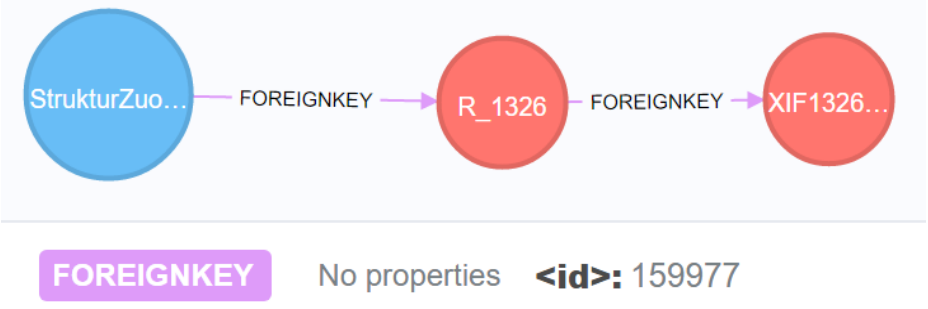


*Abbildung 15: Beziehung FOREIGNKEY*

Diese Beziehung `FOREIGNKEY` bildet den Fremdschlüssel im relationalen Datenmodell ab und ist in Abbildung 15 zu sehen. Die Beziehung wird über zwei Kanten um den Index-Knoten des Typs „R“ abgebildet. Die erste Kante geht vom Tabellenknoten der Tabelle, zu der der Fremdschlüssel gehört,

zum Indexknoten. Die zweite Kante zeigt vom Indexknoten auf einen zugehörigen Indexknoten vom Typ „XIF“ innerhalb der Tabelle. Diese Beziehung zeigt nicht auf eine dritte Tabelle. Dafür gibt es die `REFERENCE` Beziehung.

### 4.2.5 Die Beziehung REFERENCE

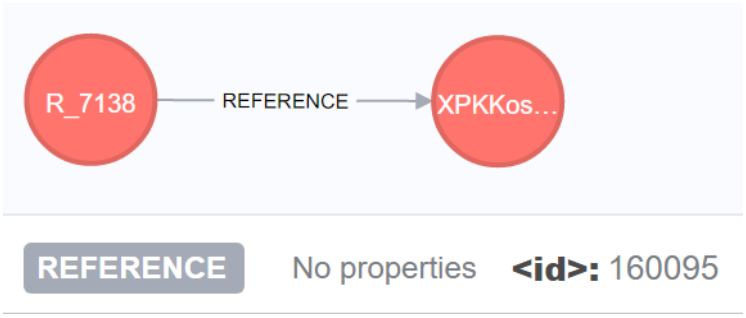


*Abbildung 16: Beziehung REFERENCE*

Die `REFERENCE`-Beziehung in Abbildung 16 bildet die Verbindung zwischen einer Tabelle und einem Fremdschlüssel im relationalen Datenmodell ab. Dabei führt von einem Indexknoten vom Typ „R“ eine Kante zu einem Indexknoten des Typs „XPK“ in der zweiten Tabelle.

### 4.2.6 Die Beziehung CONNECTED

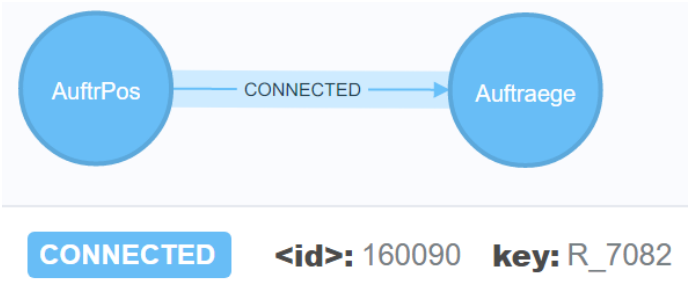


*Abbildung 17: Beziehung CONNECTED*

Die Beziehung `CONNECTED`, die in Abbildung 17 zu sehen ist, ist nicht notwendig für die Abbildung des relationalen Datenmodells. Sie wurde eingefügt, um analysierende Graphendatenbank-Abfragen zu vereinfachen. Auch beim Anzeigen der Daten aus der Graphendatenbank über Software von Dritten, erhöht diese Beziehung die Lesbarkeit der abgebildeten Datenstrukturen. Sie verbindet zwei über Fremdschlüssel verbundene Tabellenknoten zusätzlich miteinander. Diese Beziehung besitzt eine Eigenschaft, in welcher der Name des zugehörigen Fremdschlüssels festgehalten ist.

# 5 SOFTWAREARCHITEKTUR

Dieses Kapitel beschreibt die Architektur der entwickelten Software zum Importieren, Exportieren und Bearbeiten der in Kapitel 4 beschriebenen Graphendatenbank.

## 5.1 ENTWURFSENTSCHEIDUNGEN

Vor dem Beginn der Umsetzung dieser Software wurden Entwurfsentscheidungen getroffen, die hier im Detail beschrieben werden.

### 5.1.1 Programmiersprache

Für dieses Projekt wurde die Programmiersprache Java in der aktuellen Version 8 (kurz Java 8) gewählt. Die MACH AG soll später in der Lage sein, die Software eigenständig weiterzuentwickeln. Da bei der MACH AG hauptsächlich in Java programmiert wird, bietet sich diese Programmiersprache an. Zusätzlich ist eine Voraussetzung für die Verwendung der Graphendatenbank Neo4J, die Verwendung von Java 8 [9]. Der Einsatz einer anderen Programmiersprache wäre generell möglich, würde aber für spätere Entwickler, die diese Anwendung warten und erweitern wollen, zusätzlichen Mehraufwand bedeuten. Zusätzlich ist Java eine sehr weit verbreitete Sprache, was es einfacher macht passende Frameworks zu finden [10].

### 5.1.2 Graphendatenbank

Für den Einsatz einer Graphendatenbank gab es zwei Kandidaten, die mit ihrer guten Java-Unterstützung und Geschwindigkeit herausstachen. Beide stehen unter einer Open-Source-Lizenz. Die beiden Kandidaten heißen Neo4J und OrientDB.

Neo4J ist eine ACID-konforme in Java geschriebene Graphendatenbank [9]. ACID beschreibt dabei Eigenschaften, die eine Datenbankanwendung mit Transaktionen unterstützen sollte, um es mehreren Nutzern zu ermöglichen, die Datenbank gleichzeitig zu nutzen [4]. Neo4j wird seit 2003 von Neo-Technology entwickelt, erreichte 2010 die Version 1.0 und ist damit bereits länger auf dem Markt als OrientDB [7]. Die aktuelle Version 3.1.4 wird mit einer dualen Lizenz angeboten. Das bedeutet, dass es sowohl eine Business-Edition mit einer kommerziellen Lizenz als auch eine Community-Version, unter einer Open Source Lizenz gibt (GPLv3 und AGPLv3) [9]. Sie bietet den Vorteil, dass eine größere Verbreitung und Community als bei OrientDB vorhanden ist. Dies vereinfacht die Suche nach Tutorials und es stehen mehr Foren zur Verfügung, in denen Hilfe bei Problemen mit Neo4J angeboten wird. Neo4J bietet mehrere Möglichkeiten, wie es in ein Java Projekt implementiert werden kann, was bei der Entwicklung größere Freiheiten ermöglicht. Nach dem CAP-Theorem gibt es die drei Eigenschaften: Konsistenz, Verfügbarkeit und Ausfalltoleranz, die von einem verteilten System nicht

gleichzeitig erreicht werden können [4]. Konsistenz bedeutet, dass die Datenbankzustände widerspruchsfrei sind und ein Datensatz in die Datenbank entweder komplett oder gar nicht eingefügt wird. In Neo4J werden die Eigenschaften Konsistenz und Verfügbarkeit erfüllt, allerdings besitzt Neo4J keine Ausfalltoleranz. Neo4J unterstützt das Indizieren spezifischer Eigenschaften von Knoten, was Abfragen nach diesen Eigenschaften beschleunigt. Es kann über die Abfragesprachen Cypher, Gremlin und SparQL abgefragt werden [9].

OrientDB ist eine dokumentenorientierte Datenbank / Graphendatenbank unter der Open-Source-Lizenz Apache 2, die nach eigenen Angaben schneller als Neo4J sein soll [11] [12]. Sie besitzt mehr Funktionalität als Neo4J. Diese Funktionalität wie zum Beispiel Nutzerrollen, serverseitige Funktionen oder SQL-Sprachunterstützung werden für dieses Projekt allerdings nicht benötigt. Es war für die Arbeit mit OrientDB schwieriger, gute Dokumentationen und Bücher zu finden.

Da OrientDB keine reine Graphendatenbank ist und sie weniger verbreitet, sowie schlechter dokumentiert ist, fiel die Entscheidung auf Neo4J. Neo4J erscheint in dem Rahmen dieses Projektes zuverlässiger und passender für die Anforderungen, auch wenn OrientDB potenzielle Performancevorteile gegenüber Neo4J bieten könnte.

### 5.1.3 Grafische Benutzeroberfläche

Zum Erstellen der grafischen Benutzeroberfläche (GUI) wurde JavaFX ausgewählt. JavaFX ist eine Erweiterung von Java für die Entwicklung grafischer Oberflächen. Es wird als das Nachfolgeframework für das GUI-Entwicklungstoolkit Swing angesehen und ist damit der neue Java-Standard in der Benutzeroberflächenentwicklung [13] [14]. JavaFX hat deutlich modernere Elemente für die grafische Benutzeroberfläche als die Frameworks AWT, SWT oder Swing und bietet die Möglichkeit, die Oberfläche deklarativ zu beschreiben [10]. Es wird zurzeit am aktivsten von den genannten GUI-Frameworks weiterentwickelt und ist in Java 8 integriert. [10] Die Möglichkeit, die grafische Benutzeroberfläche deklarativ in JavaFX zu beschreiben, wird für die einzelnen Fenster genutzt, die im Kapitel 3.7 vorgestellt wurden. Nur das Generieren des Tabs für die Anzeige von Tabellen wird mithilfe von Code realisiert.

### 5.1.4 Kapselung vom Neo4J Framework

In dem Buch „Clean Code“ wird empfohlen, Frameworks in eigenen Wrappern (Adaptern) zu kapseln, um die Abhängigkeit von diesen zu reduzieren [15]. Daher ist für die Anbindung der Neo4J Datenbankfunktionalität ein Wrapper geplant, um die externe Bibliothek zu kapseln. Dies hat den Vorteil, dass dieses Projekt eine minimale Abhängigkeit zu dieser Bibliothek hat, was das Austauschen der Bibliothek vereinfacht. Auch wenn sich die Schnittstelle zum Framework ändern sollte, sind

notwendige Anpassungen mit dem Wrapper deutlich geringer. Zusätzlich ist es möglich eine eigene passende Schnittstelle zu schreiben, die in die Architektur dieses Projektes passt. Dieser Ansatz wird auch als Adapter-Entwurfsmuster bezeichnet.

### 5.1.5 Parser für Import

Für den Import des SQL-Skripts wurde sich für ANTLR (Another Tool for Language Recognition) entschieden. ANTLR ist ein Parser-Generator, den man nutzen kann, um strukturierten Text zu lesen, zu verarbeiten, auszuführen oder zu übersetzen [16]. Für den Einsatz von ANTLR spricht die gute Integration in Java. Zudem wird die Funktionsweise von ANTLR ausführlich im Studium an der FH Lübeck gelehrt, wodurch die Einarbeitungszeit in dieses Tool entfällt. Im Studium wurden verschiedene Techniken erläutert, wie man Informationen mit ANTLR verarbeiten kann. Daher standen sowohl das Visitor-, als auch das Listener-Pattern zur Auswahl. Es wurde sich für den Einsatz des Besucher-Entwurfsmusters (Visitor-Pattern) entschieden, da damit persönlich bessere Erfahrungen im Studium gemacht wurden.

### 5.1.6 Framework für Autovervollständigung

Für die Eingabe von Tabellennamen wurde im Suchfeld eine Autovervollständigung gewünscht. Da das Entwickeln dieser Funktionalität nicht trivial ist, besonders im Hinblick auf eine grafisch professionell aussehende Lösung und die grafische Entwicklung nicht im Vordergrund dieser Entwicklungsarbeit steht, wurde sich an dieser Stelle dafür entschieden, ein externes Framework einzusetzen. Mit einem weiteren Framework werden neue Abhängigkeiten für dieses Projekt erzeugt, allerdings ist diese Funktionalität entbehrlich und könnte bei Problemen entfernt werden. Auf der Suche nach einem passenden Framework wurde ControlsFX gefunden. Dieses Open-Source-Projekt steht unter der BSD-3-Lizenz und stellt Zusatzfunktionalitäten für JavaFX-UI-Elemente auf einem grafisch hohen Niveau zur Verfügung. Unter anderem bietet ControlsFX auch eine Autovervollständigungsfunktion [17]. Das Einbinden dieses Frameworks ist simpel und es wurde kein vergleichbares Framework mit dieser Funktionalität gefunden, weshalb es für dieses Projekt ausgewählt wurde.

### 5.1.7 Verwendete Bibliotheken

Zusätzlich zu den Java 8 Standardbibliotheken wurden folgende externe Bibliotheken als *.jar eingebunden:

| | |
|---|---|
| **ANTLR 4.6** | Diese Bibliothek wird für die Funktionen von ANTLR benötigt. |
| **Controlsfx 8.40.12** | In dieser Bibliothek ist die ControlsFX-Funktionalität enthalten. |
| **Neo4j Java Driver 1.0.6** | Diese Bibliothek wird für die Kommunikation mit der Datenbank benötigt. |

## 5.2 ARCHITEKTURBESCHREIBUNG

Als Architektur der Software wurde eine Schichtenarchitektur gewählt. Die Software wurde in die drei Bereiche UI, Core und Database unterteilt. Jede Ebene in der Architektur hat eine Aufgabe, die sie erfüllt. Dies hat den Vorteil, dass Änderungen am Quellcode nicht das gesamte System betreffen, sondern nur einzelne Ebenen. Da Ebenen untereinander nur über Interfaces kommunizieren, ist es möglich ganze Ebenen auszutauschen ohne die anderen Ebenen anzupassen.

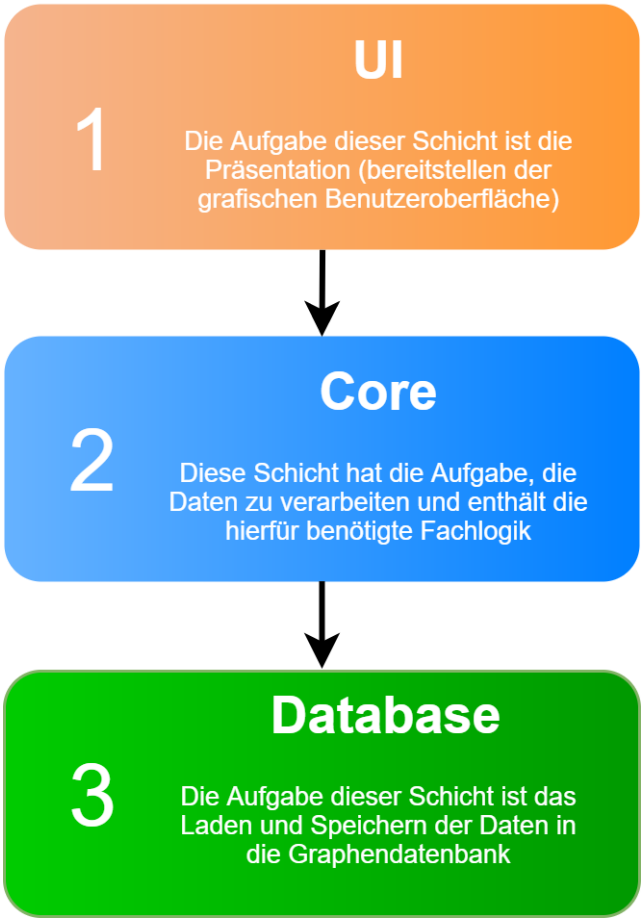


*Abbildung 18: Das Schichtenmodell*

Die Aufgabe der obersten Schicht ist die Präsentation. Diese beinhaltet das Darstellen der JavaFX-Anwendung. Der hierfür nötige Code zum Ändern der Daten in der Benutzeroberfläche befindet sich in dieser Schicht. Für den Zugriff auf die Core-Schicht gibt es zwei Interfaces in dieser. Die erste Schnittstelle ist das `Model`-Interface. Die zweite Schnittstelle ist das `Save`-Interface, mit dem die geänderten Informationen an den Core übergeben werden können.

Die nächste Schicht enthält die Fachlogik der Anwendung und wird durch das Package `core` repräsentiert. Hier wird die interne Datenhaltung geregelt. Zudem befindet sich auf dieser Schicht die Import- und Export-Logik der Software. Die Prüflogik, die vor dem Speichern ausgeführt wird, ist auf dieser Ebene enthalten. Diese Schicht greift auf zwei Interfaces der untersten Schicht zurück. Die erste Schnittstelle in der Datenbank heißt `DatabaseConnection` und stellt die Funktionen zum Verbinden mit der Datenbank und zum Laden von Informationen bereit. Die zweite Schnittstelle ist das `SaveDatabase`-Interface, das Methoden zum Speichern von Informationen bereitstellt.

Die unterste Schicht der Software hat die Aufgabe die Daten aus der Datenbank zu laden und in die Datenbank zu speichern. Daher befindet sich hier die Logik zum Generieren der Cypher-Abfragen. Zudem ist das Neo4J-Framework für den Datenbankzugriff in dieser Ebene gekapselt.

### 5.2.1 Package Struktur

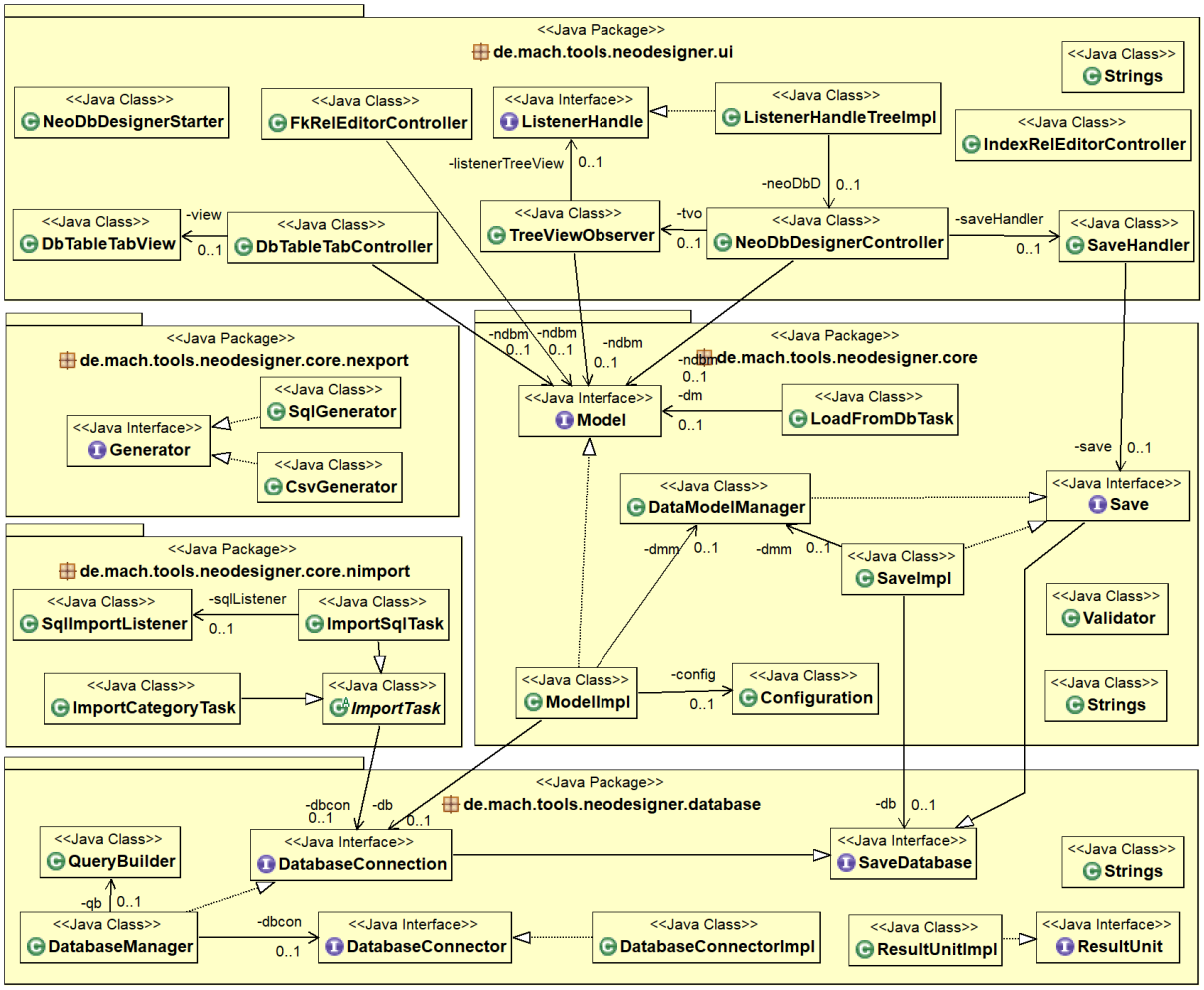


*Abbildung 19: Übersicht über die Package-Struktur der Software [gekürzt]*

Die Package-Struktur ist in Abbildung 19 zu sehen und spiegelt die Schichtenarchitektur im Wesentlichen wieder. In dem Package `ui` befindet sich die Präsentationsschicht mit all den Klassen, die dieser zugeordnet werden. In dem Package `core` liegt die Fachlogik. Zusätzlich gibt es hier drei weitere untergeordnete Packages. Dieses sind zum einen das Import- und zum anderen das Export-Package, was die jeweilige Logik zu der Thematik enthält, sowie ein drittes, nicht im Diagramm sichtbares, Package. Dabei handelt es sich um das Package `datamodel`, welches Datenstrukturen enthält, die von UI, Core und Database verwendet werden, um die Daten zu bearbeiten. Die Benutzeroberfläche verwendet hier eine angepasste Implementation dieser Datenstruktur, um die Funktionalität von JavaFX richtig nutzen zu können. Diese Datenstruktur ist im nächsten Abschnitt näher beschrieben. Das letzte Package `database` enthält die Logik, die notwendig ist, um auf die Datenbank zuzugreifen.

### 5.2.2 Interne Datenhaltung

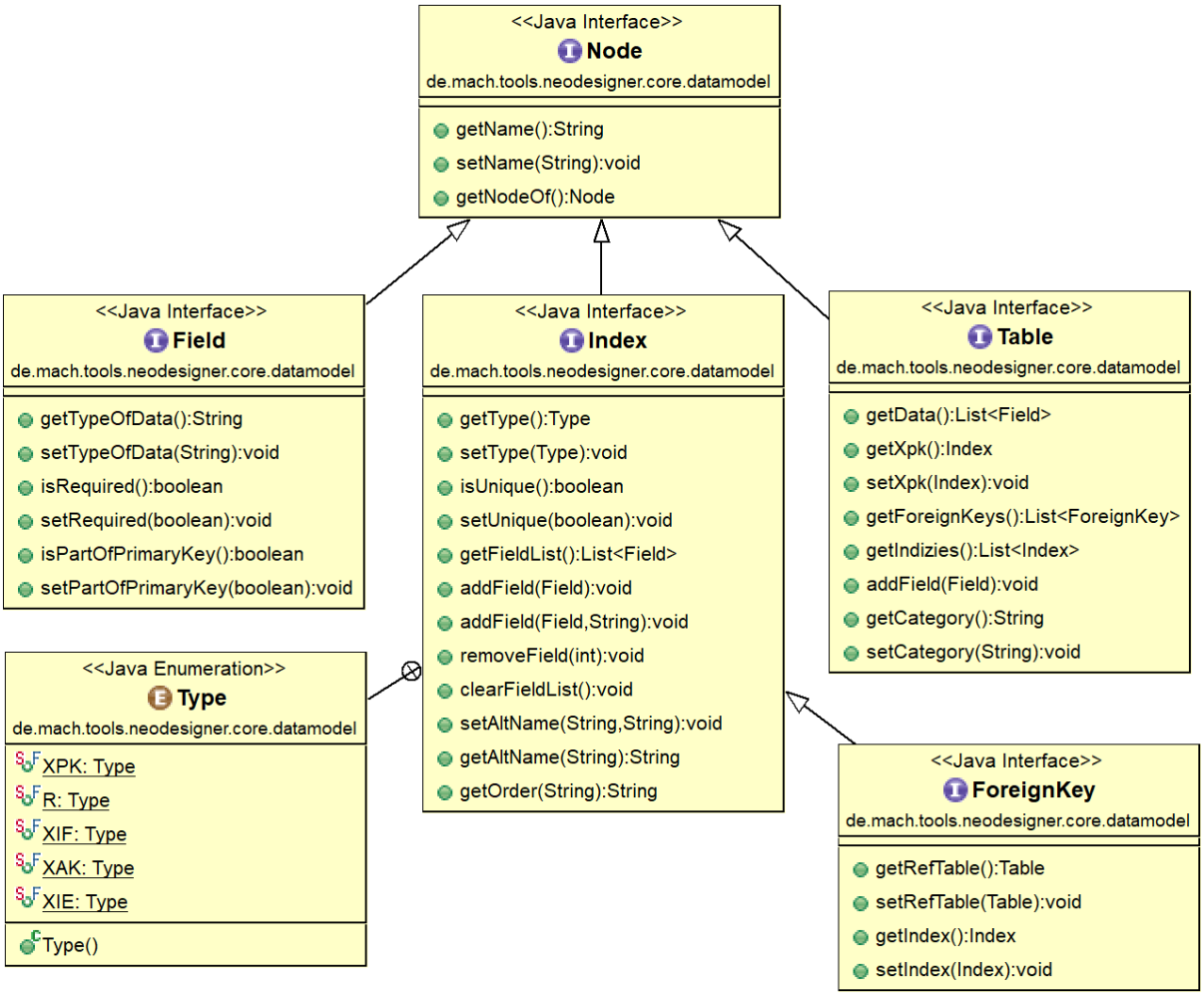


*Abbildung 20: Datenmodell der Software*

Für die interne Datenhaltung der Software wurde für jeden Knoten in der Datenbank ein Interface angelegt. In Abbildung 20 kann man sehen, wie vom zentralen `Node`-Interface die Knoten `Table` für die Tabellen, `Field` für die Spalten und `Index` für die Indizes und Primärschlüssel abgeleitet werden. Vom Indexknoten wird wiederum ein `ForeignKey`-Interface für die Fremdschlüssel abgeleitet, da sich diese in der Datenbank deutlich von den als Indexknoten gespeicherten Elementen abheben.

Zu den Schnittstellen gibt es jeweils zwei Implementationen. Die erste Implementation ist für das interne Datenmodell der Software, in das die Ergebnisse der Datenbank geladen werden. Die zweite Implementation wird für die JavaFX-Benutzeroberfläche benötigt, um diese Objekte an die Darstellung anzubinden. Die Objekte werden vom Nutzer über die grafische Benutzeroberfläche geändert und erst nachdem der Nutzer die Ansicht gespeichert hat, in die Datenbank geschrieben und in das interne Datenmodell übertragen.

# 6 IMPLEMENTIERUNG

Dieses Kapitel befasst sich mit der konkreten Implementierung der Software. Es wird auf die einzelnen Komponenten der Software eingegangen und beschrieben wie sie implementiert wurden. Ebenfalls wird hier auf die einzelnen Herausforderungen bei der Umsetzung der Bereiche eingegangen.

## 6.1 REALISIERUNG DES MODELLS (CORE)

Das Modell lässt sich in mehrere Teilbereiche unterteilen. Es enthält die Logik für den Export, Import, die Validation der Tabelle und zum Speichern der Softwarekonfiguration. Zusätzlich stellt das Modell einzelne Schnittstellen für die darüberliegende Schicht bereit. Diese Teilbereiche werden im Folgenden genauer erläutert.

### 6.1.1 Modell-Schnittstellen

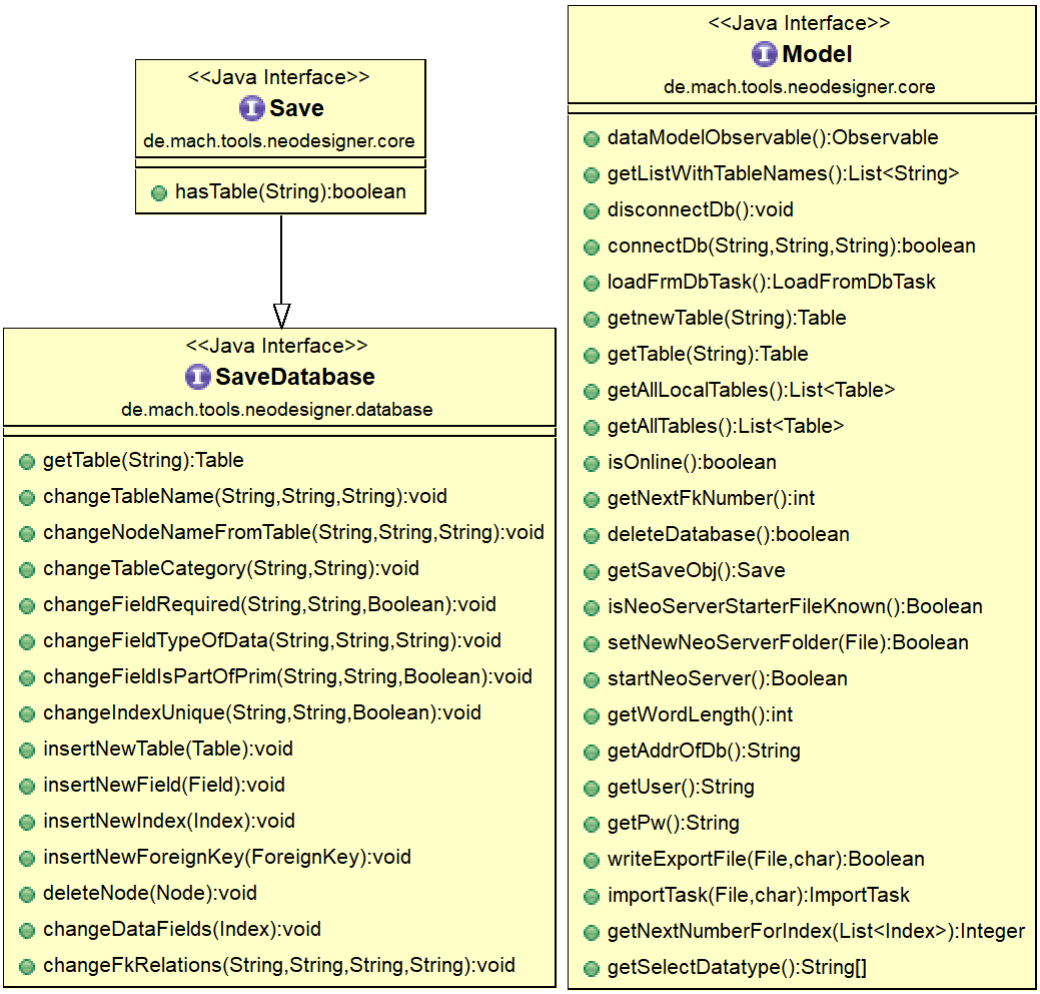


*Abbildung 21: Schnittstellen des Modells*

Im Modell werden insgesamt zwei Schnittstellen bereitgestellt namens `Model` und `Save` (Abbildung 21). Die Schnittstelle `Model` steuert die Verbindung zur Datenbank, zur Konfiguration und zum Im-

und Export. Importvorgänge laufen in einem separaten Prozess, um die Benutzeroberfläche nicht zu blockieren. Die Konfiguration wird über eine separate Klasse gesteuert, die nur über diese Schnittstelle abgefragt wird. Das Laden der Datenbank kann ebenfalls über dieses Interface gestartet werden. Dies geschieht, ebenso wie der Import, in einem separaten Prozess.

Das Modell bietet die Save-Schnittstelle, um die Daten von der Benutzeroberfläche aus zu verändern. Diese Schnittstelle basiert auf der `SaveDatabase`-Schnittstelle der Datenbank und wurde für eine saubere Architektur als eigenes Interface im Modell eingefügt. In der Klasse `DataModelManager` wird diese Schnittstelle implementiert. Hier werden die Daten aus der Datenbank in der Software zwischengespeichert. Es wurde entschieden, die gesamte Datenbank beim Start in die Software zu laden. Dies geschieht über einen Hintergrundprozess, nachdem die Software sich mit der Datenbank verbunden hat. Das hat den Vorteil, dass der Export des SQL-Skriptes und der Zugriff auf einzelne Tabellen schneller sind. Dies vereinfacht die Verwendung des Baumansicht-Elementes der grafischen Oberfläche deutlich, da die Elemente nun keine komplexe Logik zum Nachladen von fehlenden Informationen besitzen müssen. Stattdessen erweitert die `DataModelManager`-Klasse das Observable-Interface und folgt hier dem Observable-Pattern (Beobachter-Entwurfsmuster). Hiermit ist es für die darüberliegende Schicht möglich, auf Änderungen im Datenmodell zu reagieren. Während der Entwicklung mussten einige Maßnahmen getroffen werden, um die Ladegeschwindigkeit von Daten aus der Datenbank zu verbessern (siehe Kapitel 6.7). Der Arbeitsspeicherverbrauch der Software ist durch diese Art der Implementierung erhöht. Dieser Aspekt kann auf der Hardware, die unter Kapitel 3.1 beschrieben wird, vernachlässigt werden.

### 6.1.2 Funktionsweise SQL Import

Es wird eine abstrakte Klasse namens `ImportTask` als Schnittstelle für den Import eingesetzt. Dies bietet eine höhere Austauschbarkeit. Da diese Klasse als Task implementiert wurde, kann sie vom Modell aus als ein separater Task gestartet werden und blockiert so die restliche Software nicht mit ihren rechenintensiven Aufgaben. Es gibt die zwei konkreten Implementationen `ImportSqlTask` und `ImportCsvTask` für diese abstrakte Klasse. Die Implementation `ImportSqlTask` verwendet ANTLR in Kombination mit dem Besucher-Entwurfsmuster zum Importieren eines SQL-Skriptes. Es wurde eine kombinierte Grammatik für den Parser und Lexer geschrieben. Die Grammatik deckt den, in Kapitel 3.3 definierten, Sprachumfang ab. Die Grammatik besteht im ersten Teil aus einigen benötigten Lexemen.

```
SINGLE_LINE_COMMENT : '--' ~[\r\n]* -> skip; // toss out comments
ID  :   [a-zA-Z_-] ;      // match identifiers
INT :   [0-9]+ ;         // match integers
WS  :   [ \t\r\n]+ -> skip ; // toss out whitespace
```

Anschließend werden für den Parser Regeln für die einzelnen Elemente definiert, welche in den SQL-Anweisungen verwendet werden. Hierzu zählen die Tabellen-, Feld- und Indexnamen, sowie Feldtypen und Nullwerte („NULL“ und „NOT NULL“).

```
isNull: 'NULL'
;
isNotNull: 'NOT NULL'
;
tablename : ID (ID | INT)+;
fieldname : ID (ID | INT)+;
indexname : ID (ID | INT)+;

type      :   'INTEGER'
            | 'NUMBER(' INT ',' INT ')'
            | 'VARCHAR2(' INT ')'
            | 'DATE'
            | 'SMALLINT'
            | 'LONG RAW'
            | 'CHAR('INT')'
            | 'BLOB'
            | 'CLOB'
            ;
```

Schließlich folgen die komplexeren Parser-Regeln, welche die einzelnen SQL-Anweisungen verarbeiten. Diese Regeln wurden teilweise in mehrere Teilregeln zerlegt. Zum Beispiel ist das Auslesen der Felder innerhalb einer SQL-Anweisung in eine eigene Teilregel ausgelagert worden. Dies hat den Vorteil, dass sich die Einlese-Logik im Listener besser für die einzelnen Bestandteile einer SQL-Anweisung aufteilen lässt.

```
createTable :
'CREATE TABLE' tablename '(' createFieldName  ((',' createFieldName)+)?
');' # creTable
;
createIndex :
'CREATE INDEX' indexname 'ON' tablename fieldNameList # creIndex
;
createUniqueIndex :
'CREATE UNIQUE INDEX' indexname 'ON' tablename fieldNameList #
creUniqueIndex
;
primKey :
'ALTER TABLE' tablename 'ADD' '(' 'CONSTRAINT' indexname 'PRIMARY KEY'
fieldNameList ') ;' # crePrimKey
;
foreignKey:
'ALTER TABLE' tablename 'ADD' '(' 'CONSTRAINT' indexname 'FOREIGN KEY'
fieldNameList 'REFERENCES' tablename  ('ON DELETE SET NULL')? ') ;' #
creForeignKey
;
fieldNameList: '(' fieldname  ('ASC')?  ((',' fieldname  ('ASC')?  )+)?
(')' | ');') ? # crefieldNameList
;
createFieldName: fieldname type (isNull | isNotNull) #creField
;
```

Es wurde die Java-Klasse `SqlImportListener` von der mit ANTLR generierten `SQLBaseListener`-Klasse abgeleitet. Beim Betreten der einzelnen Regeln werden die Werte aus der jeweiligen SQL-Anweisung abgerufen und anschließend in eine Liste mit allen Tabellen geschrieben.

Um das Datenmodell auf der Graphendatenbank korrekt abbilden zu können, fehlen im importierten SQL-Skript einige Indizes. Diese fehlen, da sie identisch mit den Primärschlüsseln sind und daher in der Oracle-Datenbank nicht benötigt werden. In der Graphendatenbank werden sie aber benötigt, da ein kombinierter Primär- und Fremdschlüssel Änderungen am Primärschlüssel verkomplizieren würde. Deshalb werden die fehlenden Indizes für die Graphendatenbank beim Import generiert. Nachdem das SQL-Skript eingelesen wurde, wird die Liste mit allen Tabellen und ihren Werten ausgelesen und mithilfe von Importfunktionen in der Datenbank-Schicht in die Datenbank geschrieben.

Die zweite Implementation `ImportCategoryTask` importiert aus einem CSV-Dokument Kategorien für die Tabelle. Dabei erwartet der Import eine Liste mit von Kommata getrennten Werten, in der der Tabellenname der erste Wert ist und die Kategorie der letzte Wert in einer Zeile. Die Werte aus dem Dokument werden am Zeilenende und durch ein Komma getrennt und so Zeile für Zeile eingelesen. Diese Funktion wurde nicht direkt von der MACH AG gefordert. Das manuelle Zuteilen von allen Tabellen in ihre Kategorien wäre zu aufwendig gewesen und es wurde diese Funktion implementiert, um später die Einrichtung der Datenbank zu erleichtern. Es existieren keine Abnahmekriterien für diese Funktion, weshalb sie lediglich über Unit-Tests geprüft wurde.

### 6.1.3 Funktionsweise SQL-Export

Für den Export wurde das `Generator`-Interface zur Verfügung gestellt, welches die Export-Funktion austauschbar macht. Für den SQL-Export wurde die Klasse `SqlGenerator` geschrieben, welche dieses Interface implementiert. Der SQL-Export funktioniert ohne weitere Frameworks. Er orientiert sich an dem, in Kapitel 3.3, definierten Sprachumfang. Im Export werden mithilfe vom `StringBuilder` und der `String.Format`-Anweisung die Werte in definierte SQL-Anweisungsvorlagen eingefügt. So wird das Datenmodell iterativ über Schleifen abgearbeitet. Die Fremdschlüssel werden hierbei erst erstellt, nachdem alle Tabellen mit ihren Primärschlüsseln und Indizes geschrieben wurden. Da Indizes, die identisch mit dem Primärschlüssel sind, nicht erstellt werden sollen, werden diese Indizes beim Export herausgefiltert. In diesen Indizes wird zusätzlich nach einem Marker gesucht, der sie als automatisch generierte Indizes identifiziert. Nur wenn der Marker gefunden wird, werden diese Indizes nicht in die Export-Datei geschrieben. Dies ist nötig, da einige Indizes in der Oracle-Datenbank zwar die gleichen Felder wie der Primärschlüssel besitzen, aber trotzdem vorhanden sind. Ob dies ein Fehler in der Oracle-Datenbank oder gewolltes Verhalten ist, konnte im Projektzeitraum nicht geklärt werden.

Deshalb wurde davon ausgegangen, dass das Import-Skript mit dem Export- Skript identisch sein muss. Daher werden nur bei dem Import automatisch generierte Indizes im Export wieder herausgefiltert.

Zusätzlich gibt es noch einen CSV-Export, der genutzt werden kann, um die Kategorien aus der Software zu exportieren. Dieser wurde als Gegenstück zum Import der Kategorien implementiert und kann von einem weiteren Tool der MACH AG genutzt werden, um eine PDF-Version des Datenmodells zu erstellen. Diese Funktion wurde von der MACH AG nicht gefordert und wird deshalb ebenfalls über Unit-Tests geprüft.

### 6.1.4 Validator

Die Validator-Klasse bekommt beim Start eine Maximallänge für die Tabellen-, Feld- und Indexnamen übergeben. Zusätzlich besitzt sie eine hartcodierte Liste von SQL-Schlüsselwörtern (für SQL reservierte Wörter). Eine Methode prüft, ob der Name Schlüsselwörtern entspricht und ob dieser die Maximallänge nicht überschreitet. Die Methode gibt WAHR zurück, wenn der Name keine Fehler aufweist. Wenn ein Fehler gefunden wurde, schreibt die Validator-Klasse in eine Objektvariable welche Eigenschaft nicht erfüllt wurde und gibt anschließend FALSCH zurück. Eine weitere Methode prüft eine ganze Tabelle auf ihre Korrektheit. Hier wird geprüft, ob die Tabelle bereits existiert, mindestens ein Feld besitzt und ob die Namen von Feldern und Indizes die Maximallänge nicht überschreiten. Es wird auch geprüft ob keine Referenztabelle im Fremdschlüssel gesetzt wurde oder ob ein Index ohne zugeordnete Felder existiert. Wird einer dieser Fehler in der Tabelle gefunden, setzt die Methode eine genauere Fehlerbeschreibung in eine Objektvariable und gibt FALSCH zurück. Ist die Tabelle fehlerfrei, gibt die Methode WAHR zurück.

### 6.1.5 Speichern der Konfiguration

Einige Einstellungen der Software müssen permanent auf der Festplatte gespeichert werden. Hierfür wurde eine simple Klasse namens `Configuration` geschrieben. Diese Klasse schreibt die einzelnen Einstellungen wie die Login-Informationen, Informationen zu dem Speicherort der Neo4J Datenbank und die erlaubte Wortlänge in den `AppData`-Ordner. Dort erstellt sie einen Unterordner namens `NeoDatabaseDesigner` in dem die Datei gespeichert wird. Zusätzlich bietet die Software eine Funktion, um diese Informationen wieder zu laden. Das Speichern der Konfiguration wird bei jedem Verbinden mit der Datenbank ausgelöst.

## 6.2 Realisierung der Datenbank Anbindung

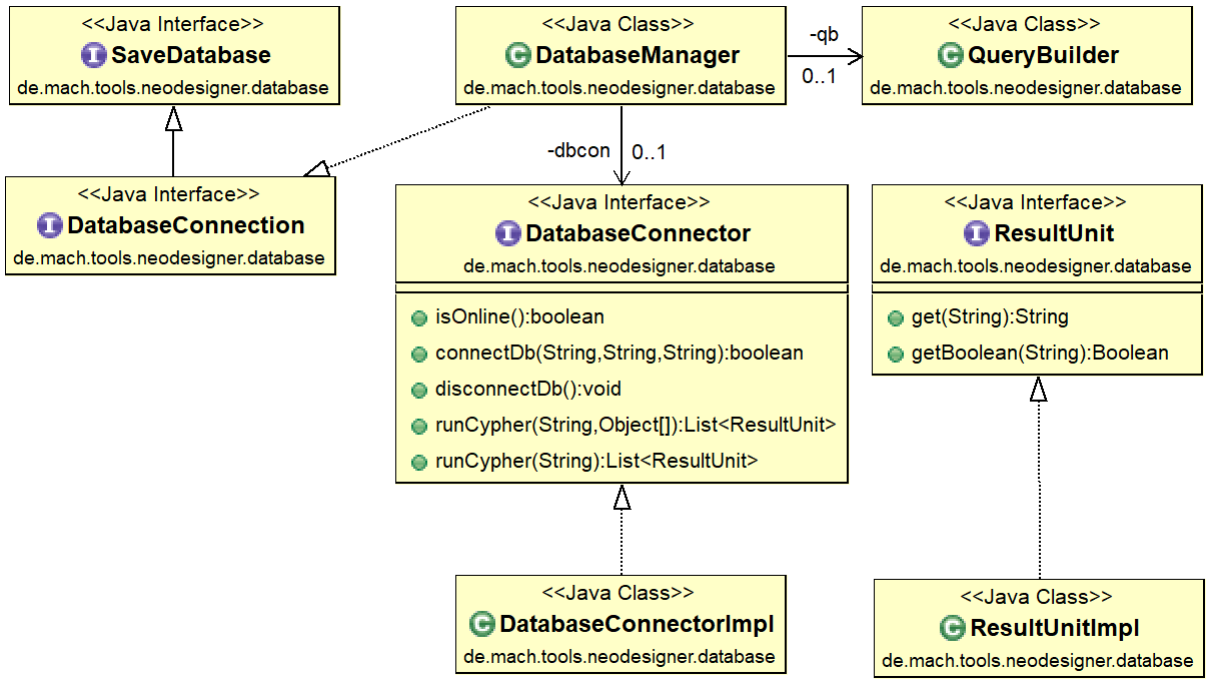


*Abbildung 22: Die Datenbankschicht im Detail*

In Abbildung 22 kann man den Aufbau der Datenbankschicht sehen. Mithilfe des Neo4J-Frameworks wird die Verbindung zur Datenbank hergestellt. Damit das Framework austauschbar bleibt, wurde dies nur in der Java-Klasse `DatabaseConnectorImpl` verwendet und über das Interface `DatabaseConnector` vom restlichen Code getrennt. Da das Abfragen von Informationen in der Datenbank über das Neo4J Framework ein `StatementResult`-Objekt zurückgibt, welches zum Framework gehört, musste dafür eine Klasse mit einem Interface geschrieben werden, um diese externe Klasse nicht direkt im Code zu verwenden. Diese Kapselung des Neo4J-Frameworks macht das Framework austauschbar, sodass es mit einem anderen Framework ausgetauscht werden könnte, das eine Verbindung zu einer Graphendatenbank mit Cypher Unterstützung herstellen kann.

Es wurde eine `QueryBuilder`-Klasse geschrieben, in der alle Cypher-Abfragen generiert werden. So sind die Änderungen an den Abfragen in dieser Klasse zentralisiert. Wiederkehrende Abfragesegmente werden in dieser Klasse zerlegt und mehrfach genutzt. So werden einzelne Abfragen aus Teilen zusammengesetzt. In der Klasse `DatabaseManager` werden sowohl der `DatabaseConnector` als auch der `QueryBuilder` genutzt und die Ergebnisse der Datenbank verarbeitet.

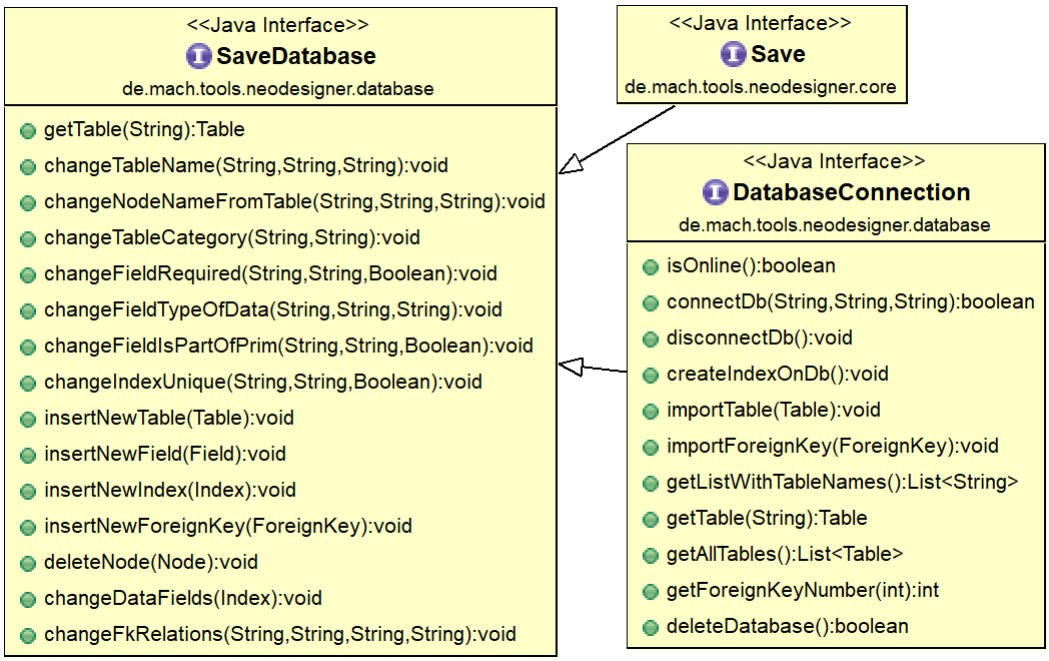


*Abbildung 23: Datenbankschnittstelleninterfaces*

Die `DatabaseManager`-Klasse implementiert die beiden Interfaces `SaveDatabase` und `DatabaseConnection`, welche von der darüberliegenden Schicht genutzt werden (Abbildung 23). `SaveDatabase` wurde als ein eigenes Interface aus `DatabaseConnection` herausgetrennt, da die darüberliegende Schicht ein Save-Interface von diesem Interface ableitet, für die aber die spezifischen Datenbankverbindungs- und Abfrageoperationen irrelevant sind.

## 6.3 Realisierung der grafischen Benutzeroberfläche

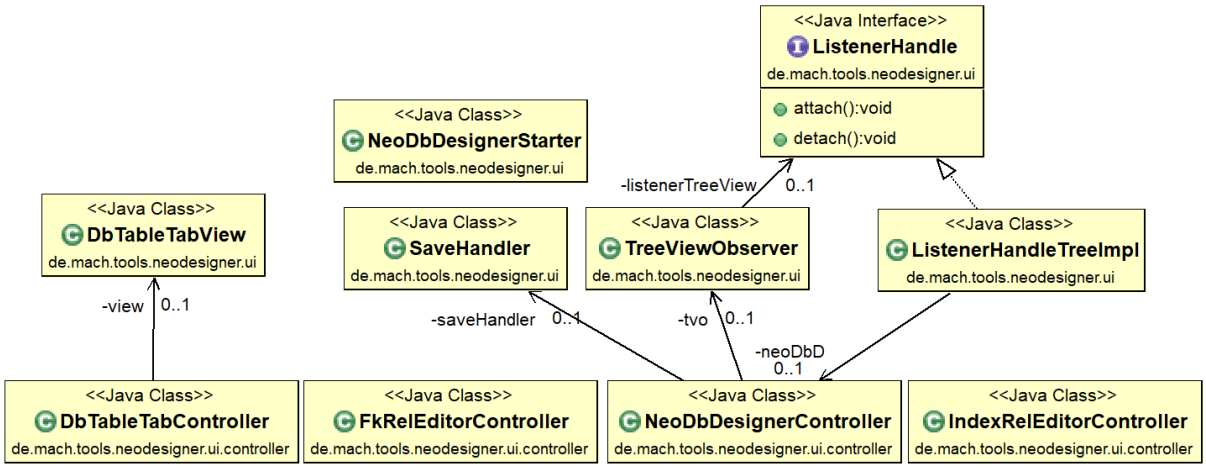


*Abbildung 24: Die Darstellungsschicht im Detail*

Die grafische Benutzeroberfläche wurde mithilfe von „*.fxml“-Dateien beschrieben. Für jede Oberfläche existiert ein Controller (Abbildung 24). Zusätzlich gibt es einen Starter, der die Oberfläche

startet und den Controller mit dem View verbindet. Es wurde das Model-View-Controller-Konzept verfolgt.

Die Tabellen in der grafischen Benutzeroberfläche arbeiten über Observable-Values, weshalb das Datenbankmodell der Core-Schicht nicht für die Darstellung der Informationen in der grafischen Benutzeroberfläche verwendet werden kann. Deshalb wurde eine alternative Implementation der Datamodel-Interfaces geschrieben. Diese kapseln intern die eigentliche Implementation, besitzen aber für die Elemente, die in der Datenbank dargestellt werden, Observable-Values, die direkt über die Tabelle geändert werden können.

Für den Treeview (Baumansicht) im Hauptfenster wurde mit dem Observable-Pattern gearbeitet. Da der `DataModelManager` das Interface Observable implementiert, ist es dem Treeview-Element möglich, auf Änderungen im `DataModelManager` zu reagierten. So kann er geänderte Daten in die Ansicht übernehmen. Da das Treeview-Element gelegentlich neu geladen werden muss, muss der Observer korrekt entfernt und beim Neuladen wieder verbunden werden. Dies wurde mithilfe einer weiteren Implementation des Observable-Patterns umgesetzt. In dieser hat der Observer eine `Attach`- und eine `Detach`-Methode, mit dem der `DataModelManager` entfernt und hinzugefügt werden kann. Der `DbTableTabController` ist für die Darstellung eines Tabs innerhalb der grafischen Benutzeroberfläche zuständig.

## 6.4 Beschreibung der fertigen grafischen Oberfläche

Nachdem die grafische Benutzeroberfläche in Kapitel 3.7 detailliert beschrieben wurde, folgt hier eine Beschreibung der fertigen Oberfläche.

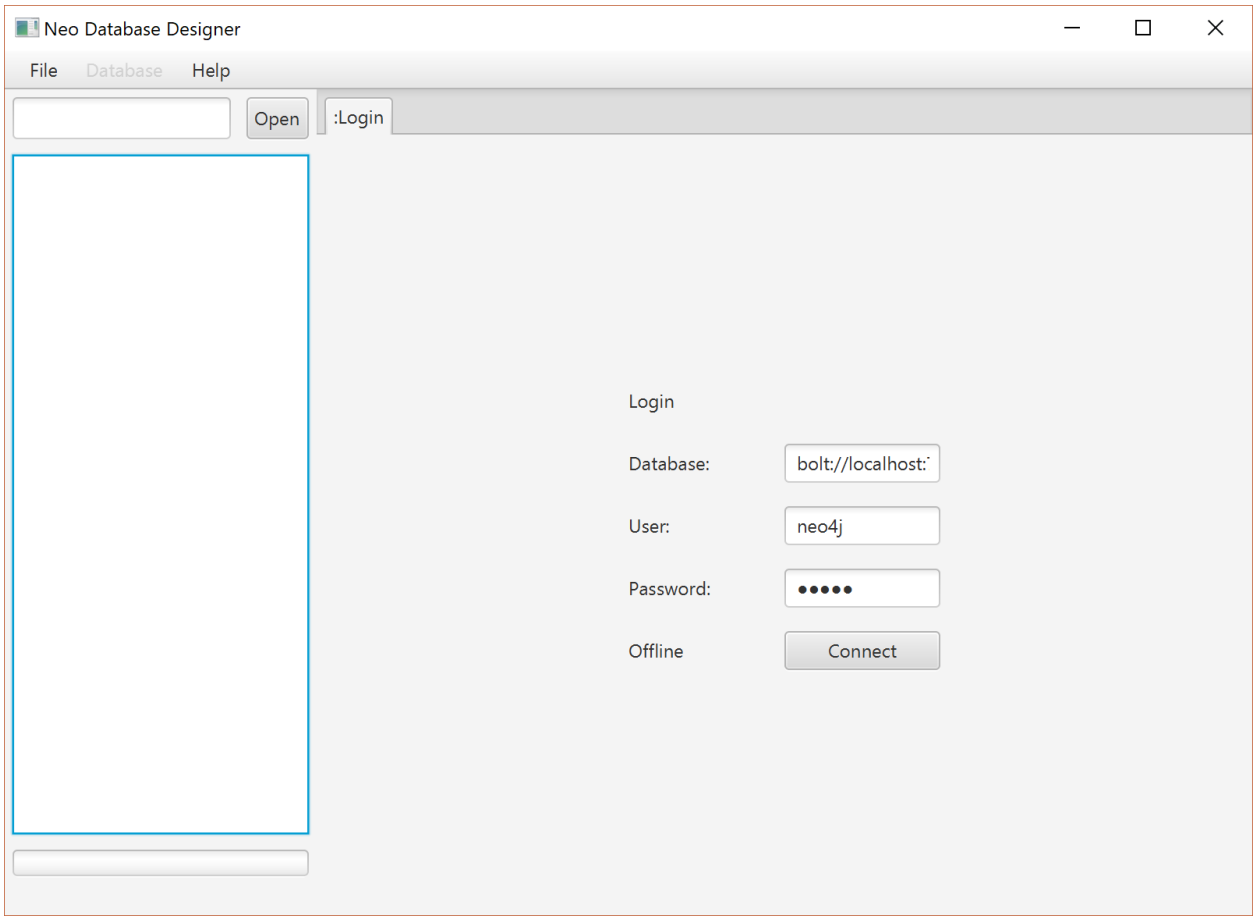


*Abbildung 25: Die Anwendung nach dem Start*

In Abbildung 25 wird die Anwendung nach dem Start dargestellt. Auf der rechten Seite können die Verbindungsdaten für die Neo4J-Datenbank eingegeben und über die „Connect“-Schaltfläche anschließend die Verbindung mit der Datenbank hergestellt werden. In der Menü-Leiste unter „File“ besteht die Möglichkeit den Neo4J-Server alternativ zu starten.

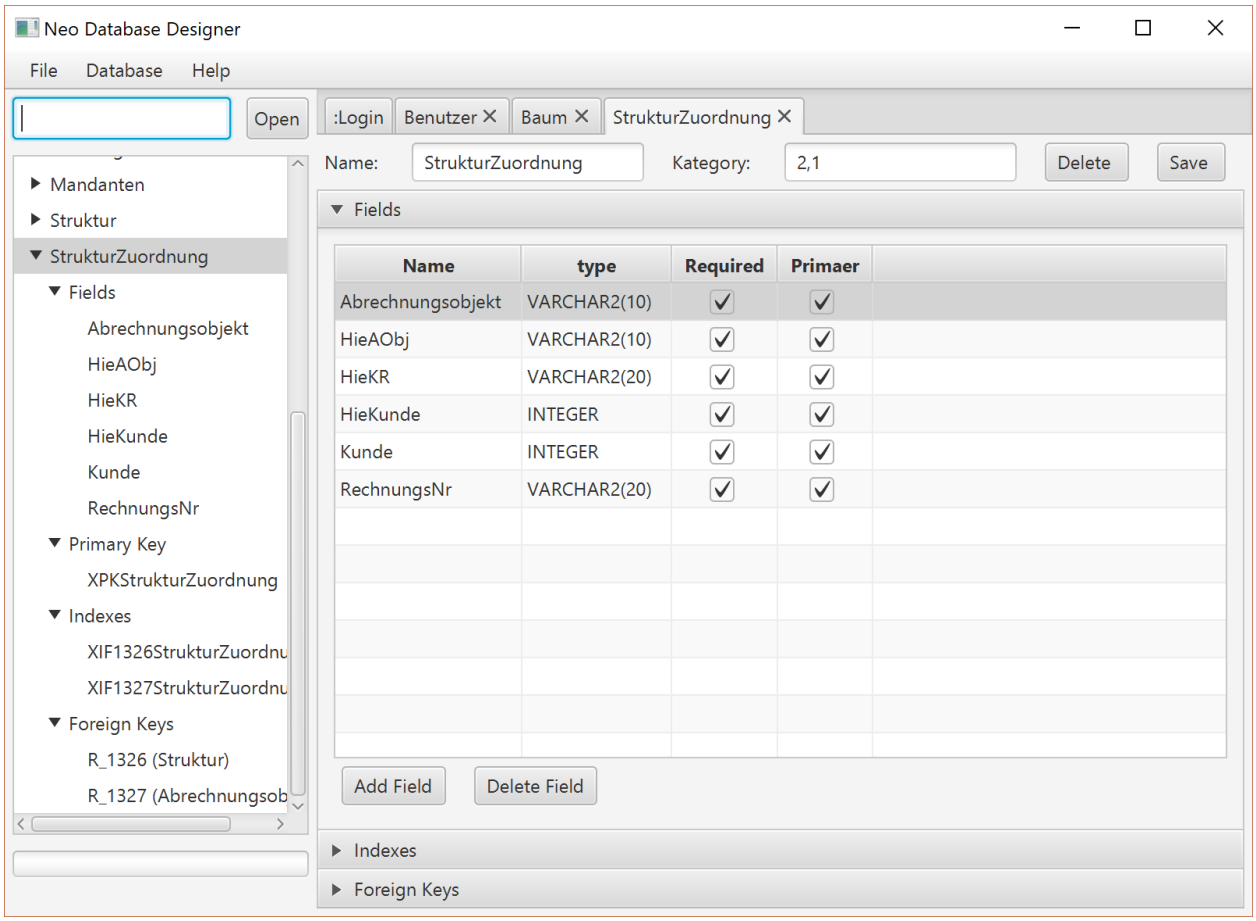


*Abbildung 26: Die Ansicht auf eine Tabelle in der Datenbank*

Wenn nach einem erfolgreichen Login eine Tabelle geöffnet wird, erscheint die Ansicht, die in Abbildung 26 zu sehen ist. Links ist das Suchfeld mit der Autovervollständigung und die Baumansicht der Tabelle zu sehen, in der die Tabelle mit ihren Elementen aufgelistet wird. Auf der rechten Seite ist die Bearbeitungsoberfläche für die Tabelle im Detail zu sehen. Beim Öffnen mehrerer Tabellen, werden diese als einzelne Tabs in der Software angezeigt, um einen einfachen Wechsel zwischen den Tabellen zu ermöglichen. Mit einem Klick auf „Indexes“ und „Foreign Keys“ können die jeweiligen Ansichten der Tabelle aufgeklappt werden. Die Menü-Leiste stellt unter „Database“ Funktionen zum Bearbeiten der Datenbank bereit. Dort kann beispielsweise eine neue Tabelle eingefügt werden.

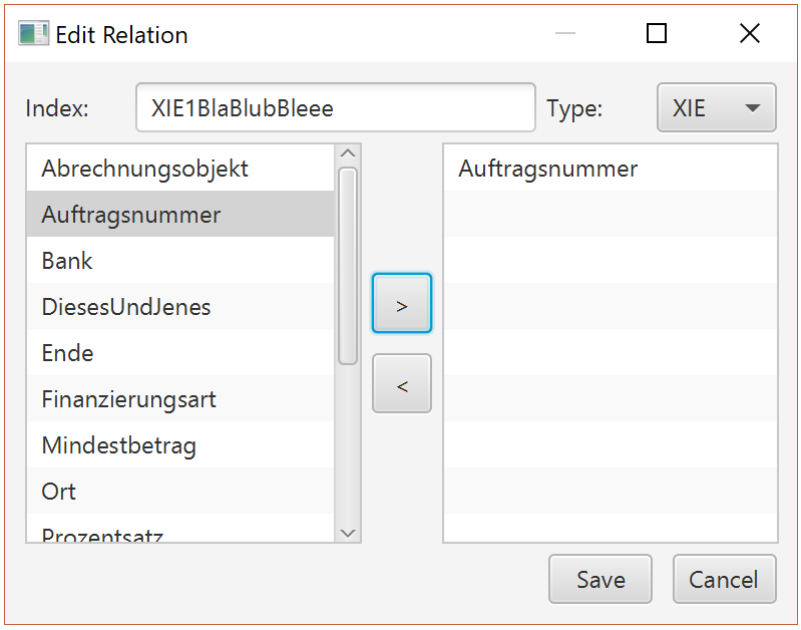


*Abbildung 27: Der Indexeditor*

Zum Bearbeiten oder Einfügen eines neuen Indexes kann der dargestellte Indexeditor (Abbildung 27) aufgerufen werden. Hier sind auf der linken Seite alle Felder der Datenbank aufgelistet. Auf der rechten Seite befinden sich die Felder, die dem Index zugeordnet sind. Die Zuordnung geschieht über die Schaltflächen zwischen den beiden Listen. Für neue Indizes wird automatisch ein Name vorgeschlagen. Über das Drop-Down-Feld kann der Typ des Indexes modifiziert werden.

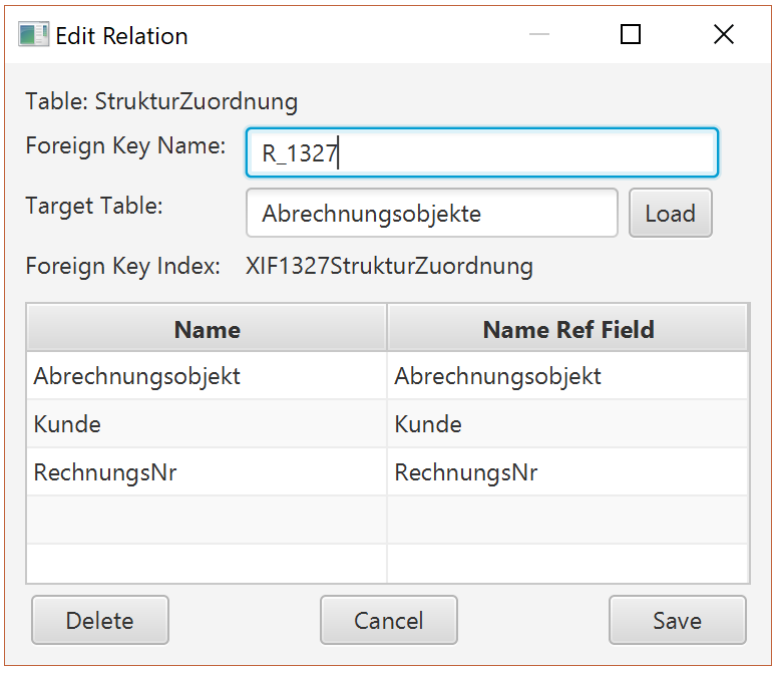


*Abbildung 28: Der Fremdschlüsseleditor*

Der Fremdschlüsseleditor, der in Abbildung 28 zu sehen ist, kann für einen vorhandenen oder neu eingefügten Fremdschlüssel aufgerufen werden. Er macht für neu aufgerufene Fremdschlüssel einen automatischen Namensvorschlag. Über „Target Table“ kann die Zieltabelle für den Fremdschlüssel per Autovervollständigung ausgewählt werden. In der Tabelle werden die zugeordneten Felder angezeigt, die per Doppelklick umbenannt werden können.

## 6.5 Qualitätssicherung während der Entwicklung

Um während der Entwicklung Fehler in der Software zu reduzieren, wurden für die IDE Plug-ins verwendet, welche geholfen haben, potenzielle Fehler in der Software während ihrer Entstehung zu entdecken und zu beheben. Folgende Plug-ins wurden in der IDE installiert:

- Checkstyle
- FindBugs
- SonarLint

Die Plug-ins „SonarLint“ und „FindBugs“ zeigen problematische Quellcodestellen auf. Das Plug-in „Checkstyle“ zeigt Formatierungsprobleme an und hilft den Code lesbarer zu formatieren. Checkstyle weist auf fehlende Dokumentation innerhalb des Quellcodes hin.

Zusätzlich bietet die IDE selbst einige Unterstützungen bei der Qualitätssicherung. Es wird zum Beispiel mithilfe sogenannter „Save Actions“ der Quellcode bei jedem Speichern formatiert. Zudem wird beispielsweise der Modifier „final“ automatisch an Variablen ergänzt, die nicht verändert werden, was als guter Stil in der Softwareentwicklung gilt [15].

## 6.6 Probleme bei der Realisierung

Während der Entwicklung der Software traten einige unerwartete Probleme auf. Ein Problem war die Zuordnung von Feldern zum Fremdschlüssel. Am Anfang des Projektes wurde angenommen, dass für jeden Fremdschlüssel ein Index existiert. Daher wäre keine direkte Beziehung zwischen Feldern und dem Fremdschlüssel notwendig gewesen. Es wurde daher nur eine Beziehung zu dem Index, der zum Fremdschlüssel gehört, erzeugt. Es stellte sich heraus, dass in der Oracle-Datenbank gelegentlich kein Index für einen Fremdschlüssel existierte. Dies wurde als ein Fehler in der Oracle-Datenbank angesehen. Bei späteren Tests fiel auf, dass die neu erzeugten Indizes, die Primärschlüssel in der Oracle Datenbank überschrieben. Dies ließ den Schluss zu, dass in der Oracle-Datenbank teilweise der Primärschlüssel als Index für einen Fremdschlüssel dient. In der Software hätte diese Verbindung aber unerwünschte Nebeneffekte nach sich gezogen. Beispielsweise würde so das Entfernen eines Fremdschlüssels den Primärschlüssel einer Tabelle löschen. Da dieser Umstand zu erheblichem

Mehraufwand für die Änderung bestehender Logik geführt hätte, wurde entschieden, diese in der Oracle-Datenbank nicht benötigten Indizes beim Export herauszufiltern und durch den Primärschlüssel zu ersetzen.

Ein weiteres Problem stellte sich erst bei späten Tests innerhalb der Software heraus. Es wurde ursprünglich angenommen, dass die Zuordnungsreihenfolge von Feldern zu Indizes keine Rolle spielte. Dies führte zu Fehlern beim Ausführen des generierten SQL-Skriptes. Deshalb wurde in die Software nachträglich die Funktionalität implementiert, die Reihenfolge der Felder in Indizes abzuspeichern. In diesem Zusammenhang ergab sich, dass die Reihenfolge von Feldern bei Indizes für Fremdschlüssel und den Fremdschlüssel selbst unterschiedlich sein kann. Der Fremdschlüssel orientiert sich bei der Reihenfolge seiner Felder an der Reihenfolge der Felder des Primärschlüssels der Referenztabelle. Dies konnte durch eine Sortierfunktion beim Export gelöst werden.

Auch bei der Zuordnung von Feldern eines Fremdschlüssels zu den Feldern der referenzierten Tabelle gab es Probleme. Ursprünglich wurde der Name des referenzierten Feldes in dem Feld selbst abgebildet. Da ein Feld mehreren Fremdschlüsseln zugeordnet sein kann, führte dies zu Problemen. Deshalb wurde diese Information in der Beziehung des Feldes zum Index abgebildet.

Es gab während der Entwicklung auch diverse kleinere Probleme, die ohne große Schwierigkeiten zu lösen waren. Hier trat das Problem auf, dass die Reihenfolge von Feldern falsch sortiert wurde. Dies lag daran, dass der Zahlenwert der Sortierung als Zeichenkette interpretiert wurde. Es ließ sich aber mit der Umwandlung der Zeichenkette in einen Zahlenwert innerhalb der Cypher-Abfrage lösen.

## 6.7 Performance Aspekte

Bei der Umsetzung des Imports kam es zu Performance-Problemen. Da der Import in zwei Teile geteilt war, musste das Problem erst lokalisiert werden. Das Performance-Problem konnte mit einfachen Zeitausgaben auf der Konsole gefunden werden. Der Parser arbeitete sehr schnell und benötigte unter einer Sekunde zum Parsen der Informationen. Das Zusammenbauen der Abfragen für die Datenbank und das Ausführen dieser in der Datenbank dauerten gemeinsam 10 Minuten. Das erste Problem ließ sich durch den Einsatz der `Stringbuilder`-Klasse in Java zum Zusammenbauen der Abfragen beheben. Dies reduzierte die Zeit, die für einen Import benötigt wurde, auf 7 Minuten. Als ein weiteres Problem stellte sich die Formulierung der Cypher-Abfragen heraus. Von der Software wurde jeder einzelne Tabellen-Knoten ohne Felder als eigene Cypher-Abfrage abgeschickt. Um die Felder und Indizes mit der Tabelle zu verbinden, musste der Tabellen-Knoten erst aus der Datenbank abgefragt werden. Diese zusätzliche Abfrage verlangsamte den Einfüge-Vorgang. In der Cypher-Abfragesprache gibt es die Möglichkeit, mehrere Abfrageteile in einer großen Abfrage zusammenzuführen. Dadurch

ließ sich pro Abfrage eine Tabelle mit Feldern und Indizes in die Datenbank einfügen. Dies sparte zusätzliche Abfragen nach bereits eingefügten Elementen in der Datenbank, was den Importvorgang auf etwa drei Minuten beschleunigte.

Es gab Performance-Probleme bei der Umsetzung des Ladevorgangs, der beim Verbinden mit der Datenbank die Daten aus dieser komplett lädt. So dauerte das Laden der Datenbank ohne optimierte Cypher-Abfragen etwa 15 Sekunden. Nachdem die Abfragen effektiver formuliert wurden, reduzierte sich die Zeit auf etwa 4 – 7 Sekunden.

# 7 TESTEN

Die in diesem Projekt verwendete Teststrategie basiert auf den agilen Testquadranten nach Marick (Abbildung 29) [18]. Softwaretests werden nach unterschiedlichen Aspekten in vier Quadranten unterteilt. Die Tests in den ersten beiden Quadranten unterstützen den Entwickler beim Entwickeln der Software. Diese Tests enthalten oft Anforderungsspezifikationen, sind eine Designhilfe und beschäftigen sich mit der Frage "Haben wir die Software richtig gebaut?". Wohingegen die Quadranten drei und vier sich mit der Frage "Haben wir die richtige Software gebaut?" beschäftigen. Hier wird die Software mit den vom Kunden spezifizierten Anforderungen abgeglichen. Es wird nicht nur geprüft, ob die Anforderung des Kunden erfüllt ist oder nicht, sondern auch welche Anforderungen besonders gut erfüllt sind und welche Optimierungspotenziale aufweisen [18]. Quadrant zwei und drei testen die Software aus einer fachlichen Sicht. Dabei werden die Funktionen, die der Kunde verwenden möchte, auf einem vom Code abstrahierten Niveau getestet. Diese Tests werden oft in einer Domain-spezifischen-Sprache beschrieben. Im Gegensatz dazu werden im Quadranten eins und vier technologische Aspekte wie Performance und Quellcode getestet [18]. Im Sinne des agilen Entwicklungsgedankens können die Tests in allen vier Quadranten durchaus parallel stattfinden. Es gibt keine zeitliche Abfolge, bzw. Phasen, wie sie aus dem klassischen Wasserfallmodell bekannt sind.

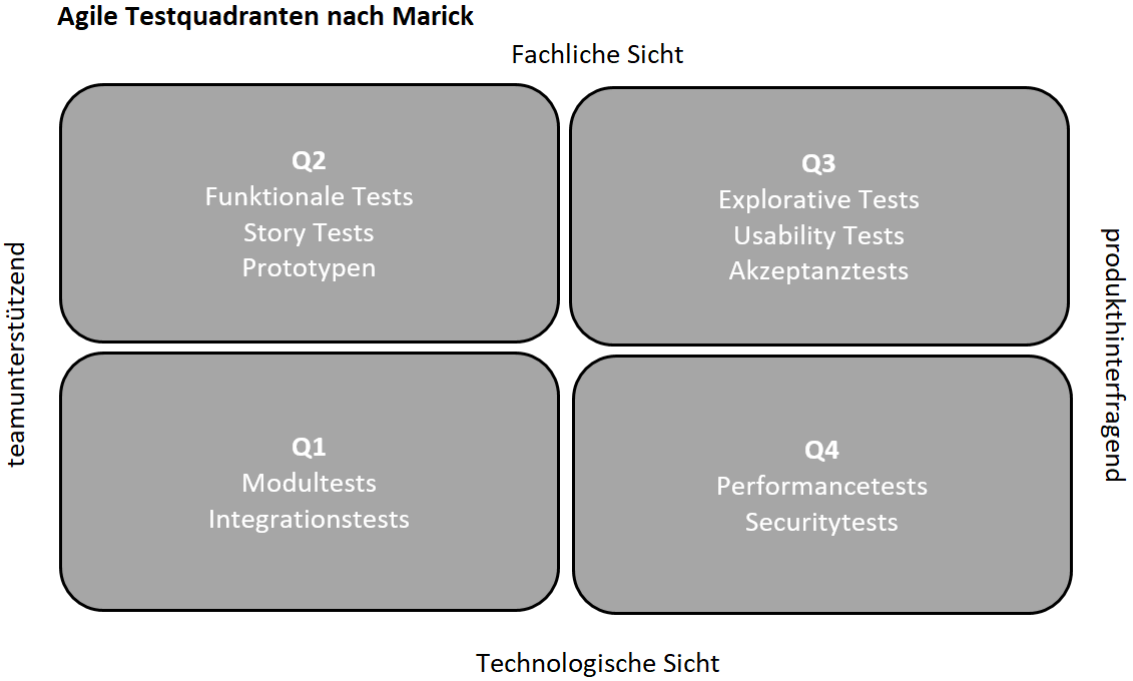


*Abbildung 29: Agile Testquadranten nach Marick*

In diesem Projekt werden im ersten Quadranten Modultests und Integrationstests automatisiert durchgeführt. Integrationstests werden für die Bereiche genutzt, die Logik aus anderen Klassen

verwenden. Der zweite Quadrant wird durch funktionale Tests abgedeckt. Hier wird der Schwerpunkt der Tests auf die grafische Benutzeroberfläche gesetzt. Der dritte Quadrant wird durch Abnahmetests auf Basis von definierten Abnahmekriterien geprüft. Im vierten und letzten Quadranten kommen in diesem Projekt Performance-Tests zum Einsatz.

Aus den drei grundlegenden Maximen modernen Softwaretestens:

1. Testen zeigt die Anwesenheit von Fehlern in genau den betrachteten Konstellationen an, nicht jedoch die generelle Abwesenheit von Fehlern.
2. Testen ist ein Feedbackmechanismus, der Informationen über das Produkt bereitstellt. Erst die Verarbeitung dieser Informationen trägt zur Qualitätsverbesserung bei.
3. Fehler verhindern, statt Fehler finden. Jeder Fehler, der bereits vor der Realisierung verhindert werden kann, reduziert den Fehlerfindungs- und Fehlerbehebungsaufwand.

Aus diesen Maximen lässt sich ableiten, dass ein vollständiges Testen nicht möglich ist. Daher wird eine Testauswahl getroffen. Diese wird aufgrund der zur Verfügung stehenden Testorakel getroffen. Als ein Testorakel, das „befragt“ werden kann, wird eine Quelle für Informationen genannt, auf dessen Basis die Tests erstellt werden [19]. In diesem Fall sind zwei Orakel ausgewählt worden. Als erstes Orakel dienen die Anforderungen und als zweites Testorakel dienen die Abnahmekriterien.

Trotz bestandener Tests, gibt es keine Garantie für die Fehlerfreiheit dieser Software. Außerdem sind viele Fehler, die beim Testen aufgetreten sind, zurück in die Implementation geflossen und wurden dort direkt behoben. Zusätzlich wurden die, in dem Kapitel 6.5 beschriebenen Maßnahmen umgesetzt, um Fehler bereits in der Realisierungsphase zu verhindern.

## 7.1 Erster Testquadrant

Der erste Testquadrant repräsentiert eine testgetriebene Entwicklung (TDD), die häufig in agilen Entwicklungen eingesetzt wird [18]. Mithilfe von JUnit werden Unit-Tests für dieses Projekt geschrieben. Unit-Tests prüfen die einzelnen Software Module isoliert auf ihre Funktionalität [20]. Hierbei wird über das Eclipse-Plugin „ECLEmma“ sichergestellt, dass die Unit-Tests eine möglichst hohe Codeabdeckung innerhalb des Modules erreichen [21].

Über Unit-Tests lassen sich nicht alle Bereiche der Software testen, weshalb zusätzlich noch Integrationstests zum Einsatz kommen. Die Aufgabe von Integrationstests ist das Testen vom Zusammenspiel verschiedener Softwaremodule [20]. Diese Tests lassen sich in Java über JUnit-Tests realisieren. Da diese Tests gerade in Verbindung mit der Datenbank sehr aufwendig sind, wurden sie nicht für alle Bereiche der Software geschrieben, sondern für besonders wichtige und besonders

schnell zu prüfende Bereiche. Andernfalls wäre selbst mit einem Mock der Datenbankverbindung der Aufwand für eine 100% Codeabdeckung deutlich größer, als er für dieses Projekt vertretbar wäre.

Unit-Tests und Integrationstests sind erfolgreich, sobald der zugehörige JUnit-Test erfolgreich ausgeführt wurde. Dies ermöglicht eine einfache Auswertung, da diese Tests entweder bestanden sind oder nicht.

## 7.2 Zweiter Testquadrant

Im zweiten Testquadranten steht der funktionale Aspekt im Vordergrund, welcher über die GUI getestet wird. Diese Tests sollen prüfen, ob die Software aus fachlicher Sicht die Anforderungen erfüllt und dienen dem Entwickler zum Testen seiner Software.

Die Automatisierung einer GUI ist mit erheblichem Initialaufwand verbunden. Es gibt zwar JavaFX-Test-Frameworks wie „TestFX“, die versprechen JavaFX-Oberflächen ausgiebig automatisch zu testen, doch befinden sich diese nach eigenen Angaben im Alpha-Status [22]. Daher wurde sich in diesem Projekt entschieden, die grafische Benutzeroberfläche über manuell durchgeführte Tests zu prüfen. Es wurden keine zusätzlichen Anstrengungen unternommen, um die grafische Benutzeroberfläche für sich zu testen. Der Mehraufwand, den genauere Tests bedeutet hätten, wäre problematisch gewesen, weshalb hier ein Kompromiss zwischen isolierter Funktion einerseits und Erstellungsaufwand bzw. erwartetem Nutzen andererseits getroffen wurde.

## 7.3 Dritter Testquadrant

Im dritten Testquadranten wird einmal über Abnahmetests und einmal über einen vergleichenden Test sichergestellt, dass die Software aus fachlicher Sicht die Anforderungen des Kunden erfüllt. Diese beiden Testformen werden im Folgenden genauer beschrieben.

### 7.3.1 Abnahmetests

Abnahmetests prüfen, ob die vom Kunden verlangten Anforderungen von der Software erfüllt werden und arbeiten mit realen Daten [20]. Die Anforderungen sind durch definierte Abnahmekriterien beschrieben. Diese Abnahmetests sollen gleichzeitig die Software im Live-Einsatz testen und verifizieren, dass die Software als Ganzes korrekt funktioniert und dem Einsatzzweck des Kunden entspricht. Diese Tests sind erfolgreich, wenn das, in dem Testfall vorher definierte, Ergebnis aufgetreten ist.

Die Abnahmetests für die Abnahmekriterien wurden in dieser Arbeit im Gherkin-Format beschrieben. Gherkin ist eine Business-Domain-Language, die natürlicher Sprache ähnelt. Mit ihr werden einzelne

Szenarien beschrieben, die durch bestimmte Ausdrücke strukturiert werden [23]. Auch die Reihenfolge ist durch diese Ausdrücke eindeutig. Jedem Szenario kann ein Abnahmekriterium zugeordnet werden. Die Phasen eines Szenarios stellen gemeinsam einen Abnahmetest dar. Ausdrücke beginnen mit den Schlüsselwörtern wie GIVEN, WHEN und THEN oder Konjunktionen wie AND oder BUT. Auch das Nutzen von Variablen mithilfe des EXAMPLES-Schlüsselwortes ist möglich.

Die definierten Abnahmekriterien sind im Anhang im Kapitel XI zu finden.

### 7.3.2 Datenbank Vergleichstest

Der Sinn des Vergleichstestes ist, sicherzustellen, dass das importierte SQL-Skript identisch mit dem SQL-Skript ist, das aus der Software exportiert wird. Da die Anweisungen in den SQL-Skripten andere Formatierungen und eine unterschiedliche Reihenfolge aufweisen können, ist eine Prüfung auf Unterschiede schwierig. Hier bietet es sich an, mit beiden SQL-Skripten jeweils eine Datenbank generieren zu lassen und diese Datenbanken anschließend zu vergleichen. Sollte das Programm korrekt arbeiten, dürfte es zwischen diesen Datenbanken keine Unterschiede geben. In diesem Fall wäre der Test bestanden. Die Überprüfung der beiden generierten Datenbanken geschieht über Liquibase [24]. Liquibase ist eine Open-Source-Bibliothek, die diverse Funktionen bietet, die helfen, Änderungen des Datenbankschema an einer Datenbank durchzuführen. Unter Anderem besitzt Liquibase eine „Diff“ Funktion, womit es möglich war die beiden generierten Datenbanken zu vergleichen. Für den Vergleich wurden folgende Zeilen (mit angepassten Nutzernamen und Passwort) in die Windows-Eingabeaufforderung eingegeben:

```
liquibase.bat --driver=oracle.jdbc.OracleDriver ^
        --url=jdbc:oracle:thin:@127.0.0.1:1521:upd ^
        --username=testuser1 ^
        --password=12345678 ^
    diff ^
        --referenceUrl=jdbc:oracle:thin:@127.0.0.1:1521:upd ^
        --referenceUsername=testuser2 ^
        --referencePassword=12345678 > result.txt
```

Es wurden in der Textdatei „result.txt“ die Ergebnisse des Vergleichs ausgegeben. Mit diesem Vergleich kann geprüft werden, ob der Import [Req02 & Req03] als auch der Export [Req04] ordnungsgemäß arbeiten.

## 7.4 Vierter Testquadrant

Mithilfe von Performance-Tests soll überprüft werden, ob das Programm die zeitlichen Anforderungen der Aufgabenstellung erfüllt. Da dem Faktor Messungenauigkeit keine hohe Bedeutung beigemessen wurde, werden die Tests mit einer Stoppuhr durchgeführt. Auch die Computersysteme sind nicht speziell für den Test konfiguriert. Eine genauere Messweise wäre nur bei sehr knappen

Testergebnissen, welche die geforderte Zeit leicht überschreiten, notwendig. Getestet werden die vier in den Anforderungen genannten zeitlichen Auflagen. Als das Testsystem wurde die Hardware, die im Kapitel 3.1 beschrieben wurde, verwendet. Die Tests werden zwischen 5 und 20mal wiederholt, um die Verlässlichkeit der Ergebnisse zu erhöhen. Diese Testreihe gilt als erfolgreich, wenn alle Tests auf allen Testsystemen erfolgreich die Zeitkriterien einhalten.

## 7.5 Evaluation der Testergebnisse

In diesem Kapitel werden die einzelnen Ergebnisse der Tests des jeweiligen Testquadranten ausgewertet und am Ende ein Testfazit gezogen.

### 7.5.1 Ergebnisse erster Testquadrant

Insgesamt gibt es 48 JUnit-Tests, die sowohl Modultests als auch Integrationstests enthalten. Die Codeabdeckung der Core-Schicht beträgt 87,42%. Die Codeabdeckung der Datenbankschicht beträgt 92,5%. Die Codeabdeckung der Präsentationsschicht beträgt 0%, da die grafische Benutzeroberfläche nicht über diesen Testquadranten geprüft wird. Eine höhere Codeabdeckung in den anderen beiden Bereichen ist schwierig, da hierfür erheblicher Testaufwand betrieben werden müsste. Es wäre zum Beispiel notwendig, die Rückgabe komplexerer Ergebnisse aus der Datenbank und den Export zu simulieren. Da diese Tests vorranging die Entwicklung unterstützen und nicht das finale Produkt abnehmen, prüfen diese Tests nur, dass keine groben Programmierfehler aufgetreten sind.

### 7.5.2 Ergebnisse zweiter und dritter Testquadrant

Die Quadranten zwei und drei sind in diesem Projektkontext nicht klar abgrenzbar, weshalb ihre Ergebnisse zusammengefasst angegeben werden.

Die funktionalen Tests für den zweiten Testquadranten sind im Anhang im Kapitel XII beschrieben. Hier folgt eine Tabelle mit den Ergebnissen dieser Tests. Bei diesen Tests wurden auch Punkte erfasst, die zur Verbesserung der Software beitragen würden, allerdings nicht in den Anforderungen genannt werden. Diese sollen bei der späteren Weiterentwicklung der Software helfen und sind ein Ergebnis des dritten Testquadranten.

| ID | Testergebnis | Bemerkungen / QS-Meldungen | Anforderung |
|---|---|---|---|
| **AL01** | bestanden | | Req01 / Req07 / Req24 |
| **AL02** | bestanden | Ladezeit bei 4 Minuten mit 1100 Tabellen | Req02 / Req03 / Req25 |
| **AL03** | bestanden | Export erfolgte in weniger als 2 Sekunden mit 1100 Tabellen | Req04 |
| **AL04** | bestanden | | Req06 |
| **AL05** | bestanden | | Req05 |
| **AL06** | bestanden | | Req08 |
| **AL07** | bestanden | | Req08 |
| **AL08** | bestanden | Es fehlt ein Indikator, an dem zu sehen ist, ob gespeichert wurde oder nicht. | Req08 / Req20 |
| **AL09** | bestanden | | Req08 |
| **AL10** | bestanden | Es ist möglich, Felder von Indizes zu bearbeiten | Req09 |
| **AL11** | bestanden | | Req09 |
| **AL12** | bestanden | Es ist möglich, Felder von Indizes zu löschen | Req09 |
| **AL13** | bestanden | | Req10 |
| **AL14** | bestanden | | Req10 / Req21 |
| **AL15** | bestanden | | Req10 |
| **AL16** | bestanden | | Req11 |
| **AL17** | bestanden | | Req11 |
| **AL18** | bestanden | | Req12 |
| **AL19** | bestanden | Es ist nicht möglich, bereits bestehende Felder oder Indizes für den Fremdschlüssel zu verwenden | Req12 / Req15 / Req22 / Req23 |
| **AL20** | bestanden | Das Löschen von Fremdschlüsseln und zugehörigen Indizes und Feldern könnte problematisch bei mehrfach in Fremdschlüsseln verwendeten Feldern werden | Req12 |
| **AL21** | bestanden | | Req13 / Req14 |
| **AL22** | bestanden | | Req16 |
| **AL23** | bestanden | | Req17 / Req19 |
| **AL24** | bestanden | | Req18 / Req19 |

*Tabelle 2: Ergebnisse Funktionale Tests*

Alle funktionalen Tests wurden bestanden.

Hier sind die Ergebnisse vom Test der Abnahmekriterien. Die Definition der Abnahmekriterien ist im Anhang unter Kapitel XI zu finden.

| Nr. | Testszenario des Abnahmekriteriums | Produkt-Funktion | Testergebnis |
|---|---|---|---|
| **1** | Prüfen der Abbildung der relationalen Datenbank in der Graphendatenbank: Tabelle | 1 | bestanden |
| **2** | Prüfen der Abbildung der relationalen Datenbank in der Graphendatenbank: Spalten, Indizes und Fremdschlüssel | 1 | bestanden |
| **3** | Prüfen der Abbildung der relationalen Datenbank in der Graphendatenbank untereinander | 1 | bestanden |
| **4** | Prüfen ob der Importvorgang startet | 2 | bestanden |
| **5** | Prüfen, ob der Import alle Tabellen importiert hat | 2 | bestanden |
| **6** | Prüfen, ob die Importfunktion Fehler zurück gibt | 2 | bestanden |
| **7** | Prüfen, ob der Export durchführbar ist | 3 | bestanden |
| **8** | Prüfen, ob das Export-Skript ausführbar ist | 3 | bestanden |
| **9** | Prüfen, ob das Export Skript gleichviele Tabellen enthält | 3 | bestanden |
| **10** | Anlegen von neuen Informationen | 4 | bestanden |
| **11** | Ändern von Informationen | 4 | bestanden |
| **12** | Löschen von Informationen | 4 | bestanden |
| **13** | Verwenden des Suchfeldes | 4 | bestanden |
| **14** | Eingabe eines zu langen Namens für die Tabelle | 5 | bestanden |
| **15** | Eingabe eines Tabellennamens mit einem in SQL reservierten Wort | 5 | bestanden |
| **16** | Anlegen eines Feldes mit einem zu langen Namen | 5 | bestanden |
| **17** | Prüfen des Namens für den Fremdschlüssel | 6 | bestanden |
| **18** | Prüfen des Namens für den Primärschlüssel | 6 | bestanden |
| **19** | Prüfen des Namens für einen Index | 6 | bestanden |
| **20** | Import Performancetest | 7 | bestanden |
| **21** | Export Performancetest | 7 | bestanden |
| **22** | Login Performancetest | 7 | bestanden |

*Tabelle 3: Ergebnisse Abnahmekriterien*

Alle Abnahmekriterien wurden erfolgreich erfüllt und damit sind diese Tests bestanden.

Zusätzlich zu den Abnahmekriterien wurde der Vergleich der Datenbanken, wie in Kapitel 7.3.2 beschrieben, durchgeführt. Dabei kam als Ergebnis das im Anhang im Kapitel X zu findende Ergebnis zurück. Dieses besagt, dass sich die Reihenfolge vieler Spalten innerhalb der Tabellen geändert hat. Diese Änderung wurde erwartet und wird nicht weiter betrachtet. Des Weiteren hat ein

Fremdschlüssel eine Änderung. Hier fehlt im Vergleich zum ursprünglichen SQL-Skript ein „ON DELETE SET NULL“. Dies war eine abgesprochene Änderung, da solche Funktionen im SQL-Skript nicht genutzt werden sollen. Da ansonsten keine Änderungen bei dem Datenbankvergleich aufgetreten sind, kann davon ausgegangen werden, dass die generierte Datenbank bei Verwendung des aktuellen MACH AG-Datenbankschemas, beim Import in die Software und anschließenden Export aus der Software nicht verfälscht wird. Damit gilt dieser Tests als bestanden.

### 7.5.3 Ergebnisse vierter Testquadrant: Performancetest

In dem ersten Test in diesen Quadranten ging es um die Messung der Verbindungszeit mit der Datenbank, das heißt, bis alle Daten nach dem Login aus der Datenbank geladen wurden. Diese wurden einmal mit einer Stoppuhr und einmal mit einer Zeitausgabe der Software geprüft. Alle Messergebnisse sind in Sekunden angegeben.

Bei den Tests mit den Namen „Software Neustart“ wurde die Anwendung, nicht aber die Datenbank, neu gestartet. Bei den Tests mit den Namen „Datenbank Neustart“ wurde die Datenbank mit der Software neu gestartet.

| Test Name | Software Zeitmessung in Sekunden | Stoppuhr in Sekunden |
|---|---|---|
| **Erster Datenbankstart** | 6,11 | 7,33 |
| **Software Neustart #1** | 2,9 | 3,52 |
| **Software Neustart #2** | 1,86 | 2,34 |
| **Software Neustart #3** | 1,63 | 2,15 |
| **Software Neustart #4** | 1,63 | 2,18 |
| **Datenbank Neustart** | 5,65 | 6,96 |
| **Software Neustart #1** | 3,06 | 4,32 |
| **Software Neustart #2** | 1,68 | 2,44 |
| **Software Neustart #3** | 1,65 | 2,5 |
| **Software Neustart #4** | 1,61 | 1,96 |
| **Datenbank Neustart #2** | 5,58 | 6,97 |
| **Datenbank Neustart #3** | 4,51 | 6,06 |
| **Datenbank Neustart #4** | 4,42 | 5,28 |
| **Datenbank Neustart #5** | 4,41 | 5,48 |
| **Datenbank Neustart #6** | 4,39 | 5,26 |

*Tabelle 4: Ergebnisse Performancetest "Login in Datenbank"*

In allen Tests wurde eine Ladezeit von 10 Sekunden deutlich unterschritten. Damit gilt der Test als bestanden. Es ist bei diesem Test aufgefallen, dass die Datenbank beim ersten Start auf einem Rechner erheblich langsamer ist. Da der Nutzer vermutlich die Datenbank einmal startet und anschließend keinen Neustart durchführt, verfälscht das die Testergebnisse etwas.

Dieser Test soll die Zeit, bis die Exportdatei geschrieben wurde, messen. Dabei wurde festgestellt, dass diese Datei zu schnell erstellt wurde, als dass eine sinnvolle Zeitmessung mit einer Stoppuhr möglich gewesen wäre.

| Test Name | Softwarezeitmessung in Sekunden |
|---|---|
| **Export Test #1** | 0,16 |
| **Export Test #2** | 0,13 |
| **Export Test #3** | 0,09 |
| **Export Test #4** | 0,09 |
| **Export Test #5** | 0,09 |

*Tabelle 5: Performancetest "Export"*

In allen Tests wurde die Exportzeit in Sekunden angegeben. Allerdings beinhaltet die Softwaremessung nicht den Teil, in dem die Datei auf die Festplatte geschrieben wird. Trotzdem dauerte kein Export insgesamt länger als 1,5 Sekunden. Die Datei wurde auf eine schnelle SSD geschrieben und nicht auf eine langsamere HDD. Da der Export trotzdem eindeutig unter 10 Sekunden lag, gilt dieser Test als bestanden.

Der dritte Test befasst sich mit dem Import von einem SQL-Skript.

| Test Name | Softwarezeitmessung | Stoppuhr in Sekunden |
|---|---|---|
| **Import Test #1** | 4:03,509s | 4:07,11 |
| **Import Test #2 (ohne Datenbank Neustart)** | 2:26,269 | 2:31,12 |
| **Import Test #3** | 3:58,298 | 4:02:94 |
| **Import Test #4** | 3:54,335 | 3:59,54 |
| **Export Test #5** | 3:57,480 | 4:02,34 |

*Tabelle 6: Performancetests Import*

Hier wurden die zeitlichen Anforderungen in allen Tests deutlich unterschritten. Des Weiteren wurde ein Unterschied in den Laufzeiten festgestellt, wenn die Datenbank nicht nach jedem Test neu gestartet wird (Siehe Import Test #2). Da der Nutzer die Datenbank normalerweise starten wird und anschließend die Software nutzt, ist das wahrscheinlichste Nutzungsszenario das einmalige Starten

der Datenbank an einem Arbeitstag. Um ein möglichst realistisches Szenario darzustellen, wurde deshalb die Datenbank vor allen anderen Tests neu gestartet. Da die Datenbank selbst beim Neustart ein wenig schneller als beim ersten Start ist, ist die Aussagekraft dieses Tests begrenzt.

## 7.6 TESTFAZIT

Das Vorgehen nach den vier Testquadranten erwies sich als ein einfacher Ansatz, der geholfen hat, die Software umfassend aus mehreren Perspektiven zu testen. Die Einteilung der Quadranten ist sinnvoll, wobei der zweite und dritte Quadrant teilweise schwer zu trennen sind. Trotzdem würde dieses Testverfahren in Zukunft wiederverwendet werden. Teilweise haben diese Test auch Fehler auf Seite des Kunden aufgedeckt, wie zum Beispiel eine „ON DELETE SET NULL“-Bedingung, die eigentlich nicht hätte existieren sollen.

Im vorherigen Kapitel wurde gezeigt, dass alle Quadranten erfolgreich getestet wurden. Dies beweist, dass die Software die Anforderungen des Kunden erfüllt. Durch nutzen der zwei Testorakel und der vier Testquadranten ist sichergestellt, dass die Software aus verschiedenen Perspektiven getestet wurde. Dies senkt die Wahrscheinlichkeit, dass Fehler nicht aufgefallen sind, deutlich. Beim Test sind Details aufgefallen, die in der Software langfristig optimiert werden sollten. Diese Details sind nicht relevant für die Anforderungen und stehen daher der Abnahme der Software nicht im Wege. Dazu zählen das Ändern und Löschen von komplexeren Fremdschlüsselstrukturen im Datenbankmodell.

# 8 ERGEBNISSE UND ZUSAMMENFASSUNG

Die Kernaufgabe dieser Arbeit war das Entwickeln einer Software, die es ermöglicht, ein relationales Datenbankmodell aus einem SQL-Skript in eine Graphendatenbank in ein geeignetes Format zu überführen und die Informationen wieder aus der Graphendatenbank zu extrahieren und zurück in ein SQL-Skript zu speichern. Ferner sollte diese Software auch in der Lage sein, das relationale Datenbankmodell innerhalb der Graphendatenbank zu modifizieren. Diese Operationen umfassen das Einfügen, Ändern und Löschen von Tabellen, Feldern, Indizes und Fremdschlüsseln.

Da bisher keine vergleichbare Software, wie die hier entwickelte Anwendung, existierte, musste eine Software neu erstellt werden. Hierfür war es notwendig, aus den Anforderungen des Kunden, der MACH AG, eine Software zu entwerfen, die diese Aufgabe lösen kann. Es war der Entwurf einer Softwarearchitektur und einer Benutzeroberfläche nötig. Zusätzlich musste die benötigte Fachlogik in die Software implementiert werden. Da die Software eine Graphendatenbank, JavaFX und ANTLR verwendet, war eine Einarbeitung in diese Komponenten notwendig. Außerdem musste sich in die Thematik von relationalen Datenbanken eingearbeitet werden, um für die Verarbeitung vom SQL-Skript und der Abbildung des relationalen Datenbankschemas in eine Graphendatenbank durchzuführen. Die Anforderungen an die Software mussten vor der Entwicklung der Software definiert und in Teilanforderungen untergliedert werden. Anschließend konnten diese Anforderungen den einzelnen Komponenten der Software zugeordnet werden.

Die Anwendung ist nun in der Lage ein SQL-Skript in die Graphendatenbank zu importieren und wieder in ein SQL-Skript zu exportieren. Des Weiteren können auch Änderungen an dem Datenmodell vorgenommen werden. Dies umfasst das Erstellen, Bearbeiten und Löschen von Tabellen, Feldern, Indizes und Fremdschlüsseln. Zusätzlich können Beziehungen zwischen Indizes bzw. Fremdschlüsseln und ihren zugeordneten Feldern geändert werden.

Aus Zeit- und Komplexitätsgründen konnten nicht alle optionalen Aufgabenteile der Aufgabenstellung durch die entworfene Software umgesetzt werden. Nicht umgesetzt wurden die grafische Darstellung und der Prüfmechanismus für eine einheitliche Groß- und Kleinschreibung. Daher ist die Software so entworfen worden, dass es möglich ist, diese Änderungen jederzeit einfach nachträglich zu implementieren.

Alle umgesetzten Anforderungen wurden durch geeignete Tests auf Korrektheit geprüft. Durch die aufgezeigte Nachweisstrategie (siehe Kapitel 7), mit der Verwendung der vier Testquadranten, ist eine umfassende Testabdeckung gewährleistet, welche die Software aus unterschiedlichsten Perspektiven getestet hat. Diese zeigen, dass das Ziel der Arbeit, die Erstellung eines prototypischen Werkzeugs zur

Modellierung von Datenstrukturen für relationale Datenbanken mithilfe von Graphendatenbanken, erreicht wurde. Die Tests haben außerdem einige Schwächen in der Software aufgedeckt, die in Zukunft optimiert werden könnten.

Der Prototyp wurde erfolgreich entwickelt und getestet. Er erfüllt die Anforderungen und hat bewiesen, dass eine Graphendatenbank dafür genutzt werden kann, ein relationales Datenbankmodell abzubilden. Ebenfalls ist es möglich dieses relationale Datenbankmodell in der Graphendatenbank zu modifizieren und wieder in ein SQL-Format zu überführen. Daher ist das Projekt als ein Erfolg anzusehen. Anzumerken ist, dass diese Anwendung nur eine prototypische Umsetzung ist, in der noch nicht alle Detailprobleme gelöst sein müssen. Insbesondere untypische Anpassungen, wie das mehrfache Vorkommen von einem Feld in verschiedenen Fremdschlüsseln, lassen sich nicht über die grafische Oberfläche der Software modellieren.

# 9 AUSBLICK

In einer weiterführenden Arbeit könnten die Analysemöglichkeiten des Datenmodells über die Graphendatenbank untersucht werden. Es könnten Schwächen gezeigt werden, die durch die Analyse über verschiedene Evaluationstechniken für Graphen aufgedeckt wurden und Optimierungen für diese vorgeschlagen werden.

Zusätzlich bietet die Software selbst einige Möglichkeiten für sinnvolle Erweiterungen. Dies umfasst nicht nur die nicht umgesetzten Funktionalitäten wie die grafische Darstellung des Graphens innerhalb der Software und das Prüfen auf eine einheitliche Schreibweise, sondern auch weitere Komfortfunktionen, wie zusätzliche Ansichten, in denen Tabellen nach Kategorien sortiert angezeigt werden können. Auch die bisherigen Funktionen könnten erweitert werden, sodass es möglich ist, kompliziertere Elemente, wie die doppelte Nutzung von Feldern für Fremdschlüssel, in der grafischen Oberfläche einzupflegen.

# VII. GLOSSAR

| | |
|---|---|
| Abstrakte Klasse: | Eine Klasse, die nur teilweise implementiert wurde. |
| Auto Inkrement: | Beschreibt in Datenbanken die Funktion, dass die Datenbank einen einzigartigen Ganzzahlwert selbstständig generiert und einfügt. |
| Boolesch: | Ein Wert, der entweder „Wahr“ oder „Falsch“ ist. |
| CSV: | Ein Dateiformat, welches mit Komma getrennte Werte enthält. |
| GUI | Siehe UI. |
| Hartcodiert: | Wenn Informationen im Quellcode fest eingefügt werden und nicht so einfach von außerhalb geändert werden können. |
| IDE: | Steht für „Integrierte Entwicklungsumgebung“. |
| Implementation: | Ist das Umsetzen von festgelegten Strukturen. |
| Integrität: | Bedeutet, dass sich die Datenbank in einen gültigen Zustand befindet und definierte Integritätsregeln erfüllt sind. |
| Interface: | Software-Schnittstelle in Java. |
| Iterativ: | Eine sich wiederholende Operation. |
| JUnit-Test: | Ein Framework, mit dem Unit-Tests in Java geschrieben werden können. |
| Klasse: | Als Klasse wird in der objektorientierten Programmierung ein abstraktes Modell für ähnliche Objekte bezeichnet. |
| Lexem: | Die kleinste Einheit eines Wortschatzes. |
| Lexer: | Steht für lexikalischer Scanner, dieser liest Text ein und zerlegt ihn in logische Einheiten. |
| Modifier: | Ist ein Schlüsselwort in der Programmierung, welches die Sichtbarkeit einer Funktion innerhalb des Codes bestimmt. |
| Observable-Interface: | Ein Interface in der Programmiersprache Java. Dies kann genutzt werden um das Observable Pattern zu implementieren. |
| Observable-Values: | Ein Wert im Observable Pattern. Dieser Wert benachrichtigt bei Veränderung seinen zugeordneten Observer. |

| | |
|---|---|
| Oracle-Datenbank: | Ist der Name von einer relationalen Datenbank, die von der Firma Oracle angeboten wird. |
| Package: | So werden die Pakete genannt, in denen eine Software untergliedert wird. |
| Parser: | Ist ein Computerprogramm, welches eine Eingabe (oft in Form von Token, die von einem Lexer produziert werden) in ein geeignetes Format umwandelt, das weiterverarbeitet werden kann. |
| Plug-in: | bezeichnet eine Softwareerweiterung, die optional genutzt werden kann. |
| Proprietär | Bezeichnet Software, die nicht unter einer freien Lizenz steht. |
| Redundanz: | Doppelte oder überflüssige Informationen. |
| SQL: | Ist eine deklarative Programmiersprache (Abfragesprache), die zur Bearbeitung und Auswertung von relationalen Datenbanken genutzt wird. |
| Schnittstelle: | Eine Schnittstelle beschreibt einen Übergabepunkt, an dem zwei unabhängige Softwareteile miteinander kommunizieren können. |
| Transaktionen: | Sind Abfolgen an Operationen, die nur als Ganzes oder gar nicht ausgeführt werden können. |
| Testorakel: | Ein Testorakel ist eine Informationsquelle über die gewünschte Testergebnisse abgeleitet werden können. |
| UI (User Interface): | Ist ein anderes Wort für grafische Benutzeroberfläche. |
| Unit-Tests: | Dienen zum Testen einer Einheit in der Software. |
| Wrapper: | Wird auch Adapter (Adapter Pattern) genannt. Kapselt bestehende Software(frameworks) und bietet Schnittstellen, über die mit der gekapselten Software kommuniziert werden kann. |

# VIII. LITERATURVERZEICHNIS


[1] M. AG, „MACH AG Lösungen," MACH AG, [Online]. Available: https://www.mach.de/loesungen/. [Zugriff am 7 April 2017].

[2] E. F. Codd, „A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM,* Bd. 13, Nr. 6, pp. 377-387, 1970.

[3] R. Adams, SQL: Eine Einführung mit vertiefenden Exkursen, München: Hanser Verlag, 2012.

[4] A. Meier, M. Kaufmann, SQL- & NoSQL-Datenbanken, Berlin Heidelberg: Springer, 2016.

[5] R. V. Bruggen, Learning Neo4J, Birmingham: Packt Publishing, 2014.

[6] I. Robinson, J. Webbe, E. Eifrem, Graph Databases, Sebastopol, CA: O'Reilly, 2015.

[7] R. Jansen, „Neo4j – NoSQL-Datenbank mit graphentheoretischen Grundlagen," Heise Medien, 15 Dezember 2010. [Online]. Available: https://www.heise.de/developer/artikel/Neo4j-NoSQL-Datenbank-mit-graphentheoretischen-Grundlagen-1152559.html. [Zugriff am 03 Mai 2017].

[8] I. Neo Technology, „What is Cypher? - The Neo4J Manual," Neo Technology, Inc., [Online]. Available: http://neo4j.com/docs/snapshot/cypher-introduction.html. [Zugriff am 21 Mai 2017].

[9] I. Neo Technology, „Neo4j: The World's Leading Graph Database," Neo Technology, Inc., [Online]. Available: https://neo4j.com/product/. [Zugriff am 03 Mai 2017].

[10] C. Ullenboom, Java ist auch eine Insel, Bonn: Galileo Press, 2014.

[11] C. Tesoriero, Getting started with OrientDB, Birmingham: Pact, 2013.

[12] O. Ltd, „OrientDB - The World's First Distributed Multi-Model NoSQL Database with a Graph Database Engine," OrientDB Ltd, [Online]. Available: http://orientdb.com/orientdb/. [Zugriff am 10 April 2017].

[13] R. Steyer, Einführung in JavaFX, Bodenheim: Springer Vieweg, 2014.

[14] B. Müller, „JavaFX im Einsatz," Alkmene Verlags- und Mediengesellschaft mbH, 12 Januar 2015. [Online]. Available: https://www.informatik-

aktuell.de/entwicklung/programmiersprachen/javafx-der-nachfolger-von-java-swing-im-einsatz.html. [Zugriff am 10 April 2017].

[15] R. C. Martin, Clean code : a handbook of agile software craftsmanship, Westford: Prentice Hall, 2015.

[16] T. Parr, The definitive ANTLR 4 reference, Frisco, TX: The Pragmatic Programmers, 2014.

[17] J. Giles, „ControlsFX," [Online]. Available: http://fxexperience.com/controlsfx/. [Zugriff am 19 April 2017].

[18] L. Crispin, J. Gregory, Agile Testing: A Practical Guide for Testers and Agile Teams, Boston: Addison Wesley, 2009.

[19] Andreas Spillner, Tilo Linz, Basiswissen Softwaretest, Heidelberg: dpunkt.verlag, 2010.

[20] L. Pilorget, Testen von Informationssystemen, Wiesbaden: Vieweg+Teubner Verlag, 2012.

[21] M. G. &. C. K. a. C. „EclEmma 2.3.3 - Java Code Coverage for Eclipse," Mountainminds GmbH & Co. KG, [Online]. Available: http://www.eclemma.org/. [Zugriff am 21 April 2017].

[22] @dainnilsson, @minisu, @hastebrot, @Philipp91, @minisu, „TestFX," GitHub, Inc., [Online]. Available: https://github.com/TestFX/TestFX. [Zugriff am 21 April 2017].

[23] C. Kram, „ABNAHMEKRITERIEN UND DREI AMIGOS," MACH AG, 22 Dezember 2016. [Online]. Available: https://www.mach.de/entwicklungs-blog-artikel/abnahmekriterien/. [Zugriff am 06 Juni 2017].

[24] N. Voxland, „Liquibase | Database Refactoring | Liquibase," [Online]. Available: http://www.liquibase.org/. [Zugriff am 20 Mai 2017].

# IX. EINRICHTUNG DER NEO4J DATENBANK (GETTING STARTED)

1. Java 8 JDK installieren
2. Herunterladen von Neo4j
    a. https://neo4j.com/download/community-edition/
    b. Für Windows die ZIP Version von Neo4J herunterladen
3. Entpacken von Neo4J
4. Die entwickelte Software starten
5. Auf „File>Start Neo4J Server“ klicken und anschließend den Ordner auswählen in dem die Neo4J Installation liegt
6. Wenn bereits ein laufender Neo4J Server vorhanden ist, können diese Schritte auch übersprungen werden
7. Wenn die Datenbank gestartet ist, kann über die entwickelte Software ein Login in die Datenbank vorgenommen werden
8. Ein SQL-Skript auswählen und importieren (Siehe Beispiel SQL im Anhang)

# X. ERGEBNIS LIQUIBASE DIFF

```
Diff Results:
Reference Database: CDTEST1 @ jdbc:oracle:thin:@127.0.0.1:1521:upd (Default
Schema: CDTEST1)
Comparison Database: CDTEST2 @ jdbc:oracle:thin:@127.0.0.1:1521:upd
(Default Schema: CDTEST2)
Compared Schemas: CDTEST1 -> CDTEST2
Product Name: EQUAL
Product Version: EQUAL
Missing Catalog(s): NONE
Unexpected Catalog(s): NONE
Changed Catalog(s): NONE
Missing Column(s): NONE
Unexpected Column(s): NONE
Changed Column(s):
 [Hier wurde die Reihenfolge der Tabellenspalten geändert – Diese
Informationen wurden aus dem Dokument entfernt, da sie nicht relevant sind
und das Dokument aufblähen]
Missing Foreign Key(s): NONE
Unexpected Foreign Key(s): NONE
Changed Foreign Key(s):
     R_3843(BELEGVORGABEN[KASNUMKREIS] -> LOOKUPKENNUNGEN[KENNUNG])
          deleteRule changed from 'importedKeySetNull' to
'importedKeyRestrict'
Missing Index(s): NONE
Unexpected Index(s): NONE
Changed Index(s): NONE
Missing Primary Key(s): NONE
Unexpected Primary Key(s): NONE
Changed Primary Key(s): NONE
Missing Sequence(s): NONE
Unexpected Sequence(s): NONE
Changed Sequence(s): NONE
Missing Stored Procedure(s): NONE
Unexpected Stored Procedure(s): NONE
Changed Stored Procedure(s): NONE
Missing Table(s): NONE
Unexpected Table(s): NONE
Changed Table(s): NONE
Missing Unique Constraint(s): NONE
Unexpected Unique Constraint(s): NONE
Changed Unique Constraint(s): NONE
Missing View(s): NONE
Unexpected View(s): NONE
Changed View(s): NONE
```

# XI. Abnahmekriterien

PF1:
Scenario: Prüfen der Abbildung der relationalen Datenbank in der Graphendatenbank: Tabelle
GIVEN Ich habe Zugriff auf die Graphendatenbank
AND Ich habe mit dem Tool ein SQL-Skript importiert
WHEN ich eine Tabelle in der Graphendatenbank anzeigen lasse
THEN hat diese Tabelle Verbindungen zu Feldern und Indizes

Scenario: Prüfen der Abbildung der relationalen Datenbank in der Graphendatenbank: Spalten, Indizes und Fremdschlüssel
GIVEN Ich habe Zugriff auf die Graphendatenbank
AND Ich habe mit dem Tool ein SQL-Skript mit der Tabelle „Mitarbeiter" und den Spalten „name" und „abteilung" importiert
WHEN ich eine Tabelle in der Graphendatenbank anzeigen lasse
THEN finde ich dort die Spalten „name" und „abteilung"

Scenario: Prüfen der Abbildung der relationalen Datenbank in der Graphendatenbank untereinander
GIVEN Ich habe Zugriff auf die Graphendatenbank
AND Ich habe mit dem Tool ein SQL-Skript importiert
WHEN ich mehrere Tabellen in der Graphendatenbank anzeigen lasse
THEN finde ich dort Verbindungen zwischen den Tabellen, womit sich der Fremdschlüssel eindeutig zu zwei Tabellen zuordnen lässt.

PF2:
Scenario: Prüfen, ob der Importvorgang startet
GIVEN Ich habe Zugriff auf die Graphendatenbank
AND Ich habe die Software gestartet
AND Ich habe die Software erfolgreich mit der Datenbank verbunden
WHEN ich den Import für ein SQL-Skript starte
THEN werden die Daten aus dem ausgewählten SQL-Skript in die Datenbank geschrieben

Scenario: Prüfen, ob der Import alle Tabellen importiert hat
GIVEN Ich habe Zugriff auf die Graphendatenbank
AND Ich habe mit dem Tool ein SQL-Skript mit 7 Tabellen importiert
WHEN ich alle Tabellen in der Graphendatenbank zähle
THEN finde ich die 7 Tabellen aus dem Skript

Scenario: Prüfen ob die Importfunktion Fehler zurückgibt
GIVEN Ich habe die Software gestartet
AND Ich habe die Software erfolgreich mit der Datenbank verbunden
WHEN ich den Import für ein SQL-Skript starte
THEN werden bis zum Abschluss des Importvorganges keine Fehler auf der Konsole ausgegeben

PF 3:
Scenario: Prüfen, ob der Export durchführbar ist
GIVEN Ich habe die Software gestartet
AND Ich habe die Software erfolgreich mit der Datenbank verbunden
WHEN ich den Exportvorgang der Datenbank starte
THEN werden SQL-Anweisungen für die Objekte in der Datenbank in die ausgewählte Datei geschrieben

Scenario: Prüfen, ob der Export Skript ausführbar ist
GIVEN Ich habe ein von der Software exportiertes SQL-Skript
AND Ich habe Zugriff auf eine Oracle Datenbank
WHEN ich das SQL-Skript importiere

THEN werden keine Fehler beim Verarbeiten des Skriptes erzeugt

Scenario: Prüfen, ob das Export Skript gleich viele Tabellen enthält
GIVEN Ich habe ein von der Software exportierte SQL-Skript mit 7 Create Table Anweisungen
WHEN ich die Tabellen prüfe
THEN finde ich dort 7 Tabellen

PF 4:
Scenario: Anlegen von neuen Informationen
GIVEN Ich habe die Software gestartet
AND Ich habe die Software erfolgreich mit der Datenbank verbunden
AND Ich habe eine Tabelle ausgewählt
WHEN ich in der Tabellenansicht den Reiter <REITER> öffne
AND ich dort ein <ELEMENT> einfüge
THEN wird diese Information in die Datenbank geschrieben
EXAMPLES

| <REITER> | <ELEMENT> |
|---|---|
| Fields | Field |
| Indexes | Index |
| ForeignKeys | ForeignKey |

Scenario: Ändern von Informationen
GIVEN Ich habe die Software gestartet
AND Ich habe die Software erfolgreich mit der Datenbank verbunden
AND Ich habe eine Tabelle ausgewählt
WHEN ich in der Tabellenansicht den Reiter <REITER> öffne
AND ich dort ein <ELEMENT> ändere
THEN wird diese Information in die Datenbank geschrieben
EXAMPLES

| <REITER> | <ELEMENT> |
|---|---|
| Fields | Field |
| Indexes | Index |
| ForeignKeys | ForeignKey |

Scenario: Löschen von Informationen
GIVEN Ich habe die Software gestartet
AND Ich habe die Software erfolgreich mit der Datenbank verbunden
AND Ich habe eine Tabelle ausgewählt
WHEN ich in der Tabellenansicht den Reiter <REITER> öffne
AND ich dort ein <ELEMENT> lösche
THEN wird diese Information aus der Datenbank entfernt
EXAMPLES

| <REITER> | <ELEMENT> |
|---|---|
| Fields | Field |
| Indexes | Index |
| ForeignKeys | ForeignKey |

Scenario: Verwenden des Suchfeldes
GIVEN Ich habe die Software gestartet
AND Ich habe die Software erfolgreich mit der Datenbank verbunden
WHEN ich im Suchfeld einen Tabellennamen eingebe
AND den Button öffnen drücke
THEN wird eine Autovervollständigung während der Eingabe angezeigt
AND die Tabelle nach dem Drücken des Buttons öffnen geöffnet

PF 5:
Scenario: Eingabe eines zu langen Namens für die Tabelle
GIVEN Ich habe die Software gestartet
AND Ich habe die Software erfolgreich mit der Datenbank verbunden
AND Ich habe eine Tabelle ausgewählt

WHEN ich in der Tabellenansicht einen Namen mit über 18 Zeichen eingebe
AND speichere
THEN wird eine Fehlermeldung ausgegeben, dass dies nicht möglich ist

Scenario: Eingabe eines Tabellennamens mit einem in SQL reservierten Wort
GIVEN Ich habe die Software gestartet
AND Ich habe die Software erfolgreich mit der Datenbank verbunden
AND Ich habe eine Tabelle ausgewählt
WHEN ich in der Tabellenansicht einen Namen mit einem in SQL reservierten Wortes eingebe wie z. B. „Select"
AND speichere
THEN wird eine Fehlermeldung ausgegeben, dass dies nicht möglich ist

Scenario: Anlegen eines Feldes mit einem zu langen Namens
GIVEN Ich habe die Software gestartet
AND Ich habe die Software erfolgreich mit der Datenbank verbunden
AND Ich habe eine Tabelle ausgewählt
WHEN ich in der Tabellenansicht ein neues Feld mit einem Namen über 18 Zeichen eingebe
AND speichere
THEN wird eine Fehlermeldung ausgegeben, dass dies nicht möglich ist

PF 6:
Scenario: Prüfen des Namens für den Fremdschlüssel
GIVEN Ich habe die Software gestartet
AND Ich habe die Software erfolgreich mit der Datenbank verbunden
AND Ich habe eine Tabelle ausgewählt
WHEN ich im Reiter „Foreign Key" einen neuen Fremdschlüssel einfüge
AND speichere
THEN wird in der GUI ein Fremdschlüssel mit „R" und einer noch nicht vergebenen laufenden Nummer erzeugt

Scenario: Prüfen des Namens für den Primärschlüssel
GIVEN Ich habe die Software gestartet
AND Ich habe die Software erfolgreich mit der Datenbank verbunden
WHEN ich eine neue Tabelle einfüge
AND speichere
THEN wird in der Datenbank ein Fremdschlüssel mit „XPK" und den Tabellennamen angelegt

Scenario: Prüfen des Namens für einen Index
GIVEN Ich habe die Software gestartet
AND Ich habe die Software erfolgreich mit der Datenbank verbunden
AND Ich habe eine Tabelle ausgewählt
WHEN ich im Reiter „Indexes" einen neuen Index einfüge
AND speichere
THEN wird in der GUI ein Index mit „XIE" und einer noch nicht vergebenen laufenden Nummer innerhalb des Indexfensters erzeugt

PF 7:
Scenario: Import Performancetest
GIVEN Ich bin angemeldet
WHEN Ich den Import eines Skriptes mit 1000 Tabellen ausführe
THEN ist dieser nach spätestens 10 Minuten abgeschlossen

Scenario: Export Performancetest
GIVEN Ich bin angemeldet
WHEN Ich den Export mit 1000 Tabellen durchführe
THEN ist dieser nach spätestens 10 Sekunden abgeschlossen

Scenario: Login Performancetest

GIVEN Ich bin nicht angemeldet
WHEN Ich mich anmelde
THEN wird die Baumansicht mit den Tabellen nach spätestens 30 Sekunden angezeigt

# XII. FUNKTIONSTESTS

| ID | Vorbedingungen & Randbedingungen | Eingabe/Interaktion | Erwartetes Ergebnis |
|---|---|---|---|
| AL01 | Anwendung gestartet | Die Verbindungsdaten der Datenbank eingeben und auf verbinden klicken | Die Software verbindet sich mit der Datenbank und läd die Informationen in die linken Baumansicht, in der nun alle Tabellen sichtbar sind. Der Vorgang dauert nicht länger als 30 Sekunden |
| AL02 | Anwendung mit Datenbank verbunden | Über File>Import SQL ein SQL-Skript (welches gültige SQL-Anweisungen enthält) auswählen und bestätigen | Die Software verarbeitet das SQL-Skript und läd es in die Datenank. Anschließend läd es die importierten Informationen in die Baumansicht auf der linken Seite. Die Software benötigt hierfür nicht länger als 10 Minuten |
| AL03 | Anwendung mit Datenbank verbunden | Über File>Export SQL den Dialog zum Speichern öffnen. Hier einen Namen und einen Ort angeben, unter dem das SQL-Skript gespeichert werden soll | Die Software exportiert das Schema der Datenbank in ein SQL-Skript |
| AL04 | Anwendung mit Datenbank verbunden | Über die Suchfunktion eine Tabelle auswählen | Die Suchfunktion schlägt automatisch Tabellennamen vor und öffnet die Detailansicht der Tabelle |
| AL05 | Tabellenansicht wurde geöffnet | Kontrolle der angezeigten Informationen | Es wird eine Tabelle mit Namen und Kategorie angezeigt. Dazu sind in den einzelnen Bereichen der Detailansicht sowohl Felder, als auch Indizes und Fremdschlüssel einsehbar |
| AL06 | Tabellenansicht wurde geöffnet | Es wird der Tabellenname und die Tabellenkategorie geändert und anschließend wird gespeichert | Die Tabelle wird angepasst in der Datenbank gespeichert |
| AL07 | Anwendung mit Datenbank verbunden | Es wird eine neue Tabelle eingefügt. | Die Tabelle wird in einer Detailansicht geöffnet |
| AL08 | Tabellenansicht wurde geöffnet | Eine neu eingefügte Tabelle wird bearbeitet und anschließend gespeichert | Die Tabelle wird in der Datenbank gespeichert und zusätzlich wird der passende Primärschlüssel generiert |

| | | | |
|---|---|---|---|
| **AL09** | Tabellenansicht wurde geöffnet | Die Tabelle wird über den Button "Löschen" gelöscht | Die Tabelle verschwindet aus der Datenbank und dem Treeview des Programms |
| **AL10** | Tabellenansicht wurde geöffnet | Es wird ein Feld geändert und anschließend wird gespeichert | Das Feld wurde geändert in die Datenbank eingetragen |
| **AL11** | Tabellenansicht wurde geöffnet | Es wird ein neues Feld eingefügt und anschließend wird gespeichert | Das Feld wurde in die Datenbank eingetragen |
| **AL12** | Tabellenansicht wurde geöffnet | Es wird ein Feld gelöscht | Das Feld wurde aus der Datenbank entfernt |
| **AL13** | Tabellenansicht wurde geöffnet | Es wird eine Index-Eigenschaft geändert und anschließend gespeichert | Die Änderungen an den Index- Feldern wurden in die Datenbank übernommen |
| **AL14** | Tabellenansicht wurde geöffnet | Es wird ein neuer Index eingefügt | Der Index wurde mit einem passend generierten Namen in der Datenbank gespeichert |
| **AL15** | Tabellenansicht wurde geöffnet | Es wird ein Index gelöscht | Der Index wurde in der Datenbank gelöscht |
| **AL16** | Tabellenansicht wurde geöffnet | Es werden zu einem Index Felder hinzugefügt | Die Feld-Index-Beziehung wurde in der Datenbank erstellt |
| **AL17** | Tabellenansicht wurde geöffnet | Es werden zu einem Index Felder entfernt | Die Feld-Index-Beziehung wurde in der Datenbank gelöscht |
| **AL18** | Tabellenansicht wurde geöffnet | Ein Fremdschlüssel wird geändert und gespeichert | Die Änderungen am Fremdschlüssel werden in die Datenbank gespeichert |
| **AL19** | Tabellenansicht wurde geöffnet | Ein Fremdschlüssel wird neu hinzugefügt | Für den Fremdschlüssel wurde ein passender Name generiert und der Fremdschlüssel wurde in der Datenbank gespeichert. Zusätzlich wurde ein passender Index generiert |
| **AL20** | Tabellenansicht wurde geöffnet | Ein Fremdschlüssel wird entfernt | Der Fremdschlüssel wird mit allen Beziehungen und zugehörigen Indizes gelöscht |

| | | | |
|---|---|---|---|
| **AL21** | Tabellenansicht wurde geöffnet | Es wird eine neue Tabelle für den Fremdschlüssel ausgewählt | Der Fremdschlüssel enthält in der Datenbank keine Verbindung zu der alten Tabelle, sondern nur zu der neu ausgewählten Tabelle. Der Fremdschlüssel hat eine Beziehung zu dem Primärschlüssel der neuen Tabelle. Alle Felder des Primärschlüssels sind im zum Fremdschlüssel gehörenden Index mit Beziehungen zu passenden Feldern abgebildet. |
| **AL22** | Tabellenansicht wurde geöffnet | Es werden Änderungen an einer Tabelle vorgenommen, aber nicht gespeichert | Die Änderungen wurden noch nicht in die Datenbank übernommen |
| **AL23** | Tabellenansicht wurde geöffnet | Es wird eine Tabelle mit einem SQL-Schlüsselwort im Namen gespeichert | Es wird eine Fehlermeldung ausgegeben, dass dies nicht möglich ist |
| **AL24** | Tabellenansicht wurde geöffnet und in der Konfiguration der Software sind 18 Zeichen als maximale Wortlänge hinterlegt | Es wird eine Tabelle mit einem zu langen Namen gespeichert | Es wird eine Fehlermeldung ausgegeben, dass dies nicht möglich ist |

# XIII. BEISPIEL SQL-SKRIPT

```
-- Table: Abteilung Kategory: none
CREATE TABLE Abteilung (
       Bezeichnung          VARCHAR2(1) NOT NULL,
       Nummer               INTEGER NOT NULL
);

ALTER TABLE Abteilung
       ADD  ( CONSTRAINT XPKAbteilung PRIMARY KEY (Nummer, Bezeichnung ) )
;

 -- Table: Dienstwagen Kategory: none
CREATE TABLE Dienstwagen (
       Kennzeichen          VARCHAR2(10) NOT NULL,
       PersonalNr           INTEGER NULL
);

CREATE INDEX XIF100Fahrzeug ON Dienstwagen
(
       Kennzeichen               ASC
);

CREATE INDEX XIF101Mitarbeiter ON Dienstwagen
(
       PersonalNr                ASC
);

ALTER TABLE Dienstwagen
       ADD  ( CONSTRAINT XPKDienstwagen PRIMARY KEY (Kennzeichen ) ) ;

 -- Table: Fahrzeug Kategory: none
CREATE TABLE Fahrzeug (
       Fahrzeugtyp          VARCHAR2(30) NOT NULL,
       Farbe                VARCHAR2(10) NULL,
       Kennzeichen          VARCHAR2(10) NOT NULL
);

ALTER TABLE Fahrzeug
       ADD  ( CONSTRAINT XPKFahrzeug PRIMARY KEY (Kennzeichen ) ) ;

 -- Table: Mitarbeiter Kategory: none
CREATE TABLE Mitarbeiter (
       Abteilung            VARCHAR2(10) NOT NULL,
       AptBezeichnung       VARCHAR2(1) NOT NULL,
       AptNummer            INTEGER NOT NULL,
       EMail                VARCHAR2(30) NULL,
       Geburtsdatum         DATE NOT NULL,
       Name                 VARCHAR2(30) NOT NULL,
       PersonalNr           INTEGER NOT NULL,
       Raum                 VARCHAR2(10) NULL,
       Telefon              VARCHAR2(10) NULL,
       Vorname              VARCHAR2(30) NOT NULL
);

CREATE INDEX XIF102Abteilung ON Mitarbeiter
(
       AptNummer                 ASC ,
       AptBezeichnung            ASC
```

```
);

ALTER TABLE Mitarbeiter
       ADD  ( CONSTRAINT XPKMitarbeiter PRIMARY KEY (PersonalNr ) ) ;

 -- Table: Schadensfall Kategory: none
CREATE TABLE Schadensfall (
       Datum                DATE NOT NULL,
       Ort                  VARCHAR2(20) NOT NULL,
       Sachbearbeiter       INTEGER NOT NULL,
       Schadenshoehe        NUMBER(18,5) NULL,
       Umstaende            VARCHAR2(40) NOT NULL,
       Verletzte            CHAR(1) NULL,
       Vorgangsnummer       INTEGER NOT NULL
);

CREATE INDEX XIF106Mitarbeiter ON Schadensfall
(
       Sachbearbeiter            ASC
);

ALTER TABLE Schadensfall
       ADD  ( CONSTRAINT XPKSchadensfall PRIMARY KEY (Vorgangsnummer ) ) ;

 -- Table: VersNehmer Kategory: none
CREATE TABLE VersNehmer (
       FuehrerscheinSeit    DATE NULL,
       Geburtstag           DATE NULL,
       Hausnummer           VARCHAR2(30) NOT NULL,
       KundenNr             INTEGER NOT NULL,
       Name                 VARCHAR2(30) NOT NULL,
       Ort                  VARCHAR2(30) NOT NULL,
       PLZ                  VARCHAR2(30) NOT NULL,
       Strasse              VARCHAR2(30) NOT NULL,
       Vorname              VARCHAR2(30) NULL
);

ALTER TABLE VersNehmer
       ADD  ( CONSTRAINT XPKVersNehmer PRIMARY KEY (KundenNr ) ) ;

 -- Table: VersVertrag Kategory: none
CREATE TABLE VersVertrag (
       Kennzeichen          VARCHAR2(10) NOT NULL,
       KundenNr             INTEGER NOT NULL,
       PersonalNr           INTEGER NOT NULL,
       Vertragsabschluss    DATE NULL,
       Vertragsart          VARCHAR2(10) NOT NULL,
       Vertragsnr           INTEGER NOT NULL
);

CREATE INDEX XIF103VersNehmer ON VersVertrag
(
       KundenNr                  ASC
);

CREATE INDEX XIF104Fahrzeug ON VersVertrag
(
       Kennzeichen               ASC
);

CREATE INDEX XIF105Mitarbeiter ON VersVertrag
```

```
(
       PersonalNr                ASC
);

ALTER TABLE VersVertrag
       ADD  ( CONSTRAINT XPKVersVertrag PRIMARY KEY (Vertragsnr ) ) ;

ALTER TABLE Dienstwagen
       ADD  ( CONSTRAINT R_100
              FOREIGN KEY (Kennzeichen)
                           REFERENCES Fahrzeug ) ;

ALTER TABLE Dienstwagen
       ADD  ( CONSTRAINT R_101
              FOREIGN KEY (PersonalNr)
                           REFERENCES Mitarbeiter ) ;

ALTER TABLE Mitarbeiter
       ADD  ( CONSTRAINT R_102
              FOREIGN KEY (AptNummer, AptBezeichnung)
                           REFERENCES Abteilung ) ;

ALTER TABLE Schadensfall
       ADD  ( CONSTRAINT R_106
              FOREIGN KEY (Sachbearbeiter)
                           REFERENCES Mitarbeiter ) ;

ALTER TABLE VersVertrag
       ADD  ( CONSTRAINT R_103
              FOREIGN KEY (KundenNr)
                           REFERENCES VersNehmer ) ;

ALTER TABLE VersVertrag
       ADD  ( CONSTRAINT R_104
              FOREIGN KEY (Kennzeichen)
                           REFERENCES Fahrzeug ) ;

ALTER TABLE VersVertrag
       ADD  ( CONSTRAINT R_105
              FOREIGN KEY (PersonalNr)
                           REFERENCES Mitarbeiter ) ;
```

# XIV. VERWENDETE VERSIONEN

Hier werden die verwendeten Softwareversionen von Frameworks und externen Tools erwähnt damit diese Projekt später besser zeitlich eingeordnet werden kann.

| Name | Version |
|---|---|
| **Java, JavaFX** | 8 |
| **ANTLR** | 4.6 |
| **Controlsfx** | 8.40.12 |
| **Neo4J Java Driver** | 1.0.6 |
| **JUnit** | 4 |
| **Neo4J** | 3.1.0 |
| **Eclipse (IDE)** | 4.6.2 |
| **EclEmma** | 2.3.3 |
| **Sonarlint** | 3.1 |

# XV. PROJEKT DOWNLOAD LINK

Dieses Projekt kann unter http://www.quhfan.de/downloads/Bachelorarbeit_Chris_Deter_2017.zip heruntergeladen werden.