



Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Entwicklung einer Personalstammdatenbank als SaaS

Bachelorarbeit

Name des Studiengangs

Wirtschaftsinformatik

Fachbereich 4

vorgelegt von

Max Sven Freudenberg

Datum:

Berlin, 09.02.2024

Erstgutachter: Prof. Dr.-Ing. Ingo Claßen

Zweitgutachter: Prof. Dr. Martin Kempa

Abstract

Die Analyse von Personaldaten hat in den vergangenen Jahren vor allem in der Privatwirtschaft eine zunehmende Bedeutung erfahren. Durch die Entwicklung digitaler Technologien können hierfür in immer größeren Umfang softwarebasierte Lösungen eingesetzt werden, wobei speziell Cloud-Computing immer häufiger eingesetzt werden.

Im Zuge dieser Abschlussarbeit wird eine historisierende Personalstammdatenbank als Software as a Service entwickelt, mit der es möglich ist, dass verschiedene Mandanten Ihre Daten auf derselben Datenbank speichern können. Deswegen muss sichergestellt sein, dass ein Mandant nicht auf die Daten anderer Mandanten zugreifen kann. Hierfür werden die Anforderungen für ein solches Datenbanksystem definiert und darauf aufbauend eine Datenbankstruktur entwickelt. Schwerpunkt dieser Arbeit ist hierbei die Mandantenfähigkeit der Datenbank sowie die Implementierung von beispielhaften Funktionalitäten, die zeigen, wie Daten in der Datenbank gespeichert, ausgelesen, manipuliert und entfernt werden können. Der im Zuge dieser Arbeit entwickelte Prototyp bildet eine Grundlage, auf der ein weiterer Ausbau möglich ist.

Inhaltsverzeichnis

Abstract	II
Abkürzungsverzeichnis	V
Abbildungsverzeichnis	VI
Tabellenverzeichnis	VIII
1 Einleitung.....	1
1.1 Relevanz des Themas	1
1.2 Zielsetzung und Abgrenzung	2
1.3 Aufbau der Arbeit	3
1.4 gendergerechte Sprache	3
2 Personalwirtschaftliche Grundlagen	4
2.1 Sozialversicherungen.....	4
2.1.1 Krankenversicherung.....	4
2.1.2 Pflegeversicherung	5
2.1.3 Arbeitslosenversicherung.....	6
2.1.4 Rentenversicherung	7
2.1.5 Unfallversicherung	7
2.1.6 Gesetzliche Umlagen	8
2.1.7 Versicherungspflichten nach Mitarbeitertypen	9
2.2 Entgelt und Tarifbindung.....	11
2.3 Personalstammdaten	12
3 Technologische Grundlagen	13
3.1 Relationale Datenbanken und SQL.....	13
3.2 Cloud Computing und SaaS	15
3.3 Mandantenfähigkeit und Row Level Security.....	16
3.4 Testen	19
3.5 Verwendete Technologien	20
3.5.1 Excel.....	20
3.5.2 PostgreSQL, PL/pgSQL und DBeaver	20
3.5.3 Python und PyCharm.....	21
4 Anforderungen	22
4.1 Nichtfunktionale Anforderungen	24
4.2 Funktionale Anforderungen	25
4.2.1 User Szenario.....	26
4.2.2 Use Cases.....	27
5 Beschreibung des Prototypen anhand ausgewählter Aspekte	40

5.1	Registrierung, Erstellung und Entfernung von Mandant, Nutzer und Administrator	40
5.2	Struktur des Datenimports	43
5.3	Mandantenfähige Datenbank	45
5.4	Datenmodellierung	48
5.4.1	Historisierung der Daten	49
5.4.2	Sozialversicherungen	50
5.4.3	Entgelt	52
6	Tests	54
7	Fazit	61
7.1	Ergebnisse der Arbeit	61
7.2	Zukunftsausblick	62
	Quellenverzeichnis	IX
	Protokoll ChatGPT	XXI
	Anhang A – Datenmodell	XXIV
	A.1 - Krankenkassen	XXIV
	A.2 – Sonstige Sozialversicherungen	XXV
	A.3 – Entgelt	XXVII
	A.4 – Sonstiges Datenmodell	XXVIII
	Anhang B – Screenshot: Ordnerstruktur	XXX
	Anhang C – Screenshot: bestandene Tests	XXX
	Eigenständigkeitserklärung	XXXI

Abkürzungsverzeichnis

BSI	Bundesamt für Sicherheit in der Informationstechnik
DGUV	Deutsche Gesetzliche Unfallversicherung
DBMS	Datenbankmanagementsystem
IaaS	Infrastructure as a Service
KMU	Kleine und mittlere Unternehmen
o.D.	ohne Datum
PaaS	Platform as a Service
PKV	Private Krankenversicherung
RLS	Row Level Security
SaaS	Software as a Service
SQL	Structured Query Language

Abbildungsverzeichnis

Abbildung 1: Übersicht für welche Beschäftigungsformen welche Sozialabgaben zu leisten sind; eigene Darstellung mit Excel	11
Abbildung 2: Aufbau einer Tabelle in einer relationalen Datenbank, eigene Darstellung mit Excel	13
Abbildung 3: Links: 1:1-Beziehung, Mitte: 1:n-Beziehung, Rechts: n:m-Beziehung; eigene Darstellung mit draw.io	14
Abbildung 4: jeder Mandant hat eine eigene Datenbank; entnommen aus Somasundar, Harish (2021)	17
Abbildung 5: Alle Mandanten nutzen eine Datenbank mit separaten Schemata; entnommen aus Somasundar, Harish (2021)	17
Abbildung 6: Identifier in Spalte "TenantId" bei einer Datenbank mit geteiltem Schema; entnommen aus Somasundar, Harish (2021).....	18
Abbildung 7: Hybride mandantenfähige Datenbank; entnommen aus Somasundar, Harish (2021) ...	18
Abbildung 8: Code, mit der ein Mandant und ein Administrator erzeugt wird; eigener Code dargestellt mit carbon.now.sh	40
Abbildung 9: Funktion zur Erstellung eines neuen Nutzers; eigener Code dargestellt mit carbon.now.sh	41
Abbildung 10: Funktion, mit der Nutzer sein Passwort ändern kann; eigener Code dargestellt mit carbon.now.sh	42
Abbildung 11: Funktion zur Entfernung eines Nutzer-Accounts; eigener Code dargestellt mit carbon.now.sh	42
Abbildung 12: Funktion zur Entfernung aller Daten eines Mandanten aus dem System; eigener Code dargestellt mit carbon.now.sh	43
Abbildung 18: Darstellung der Datenpipeline, eigene Darstellung.....	44
Abbildung 13: beispielhafte Tabelle mit Attribut "Mandant_ID"; eigene Darstellung mit draw.io	45
Abbildung 14: Implementation von Row Level Security für beispielhafte Tabelle; eigener Code dargestellt mit carbon.now.sh	46
Abbildung 15: Implementation der Rolle "tenant_user"; eigener Code dargestellt mit carbon.now.sh	46
Abbildung 16: Sicherstellung des Datenbankzugriffs über "tenant_user" bei einer beispielhaften PL/pgSQL-Funktion; eigener Code dargestellt mit carbon.now.sh	47
Abbildung 17: beispielhafter Python-Code, mit der die Übergabe der Mandant-ID sichergestellt wird; eigener Code dargestellt mit carbon.now.sh	48
Abbildung 19: Beispiel für Historisierung; eigene Darstellung mit draw.io	49
Abbildung 20: möglicher Eintrag, wenn "Datum_Bis" kein Primärschlüssel-Teil ist; eigene Darstellung mit Excel	50
Abbildung 22: beispielhafter Aufbau einer Testklasse mit mehreren Test- sowie setUp()- und tearDown()-Funktion; eigener Code dargestellt mit carbon.now.sh	55
Abbildung 23: test_set_up()-Funktion, welche das Testschema inklusive Datenbank und Skripte erstellt; eigener Code dargestellt mit carbon.now.sh.....	56
Abbildung 24: für diese Arbeit typische setUp()-Funktion; eigener Code dargestellt mit carbon.now.sh	57
Abbildung 25: "tearDown"-Funktion ruft "test_tear_down()" auf, um Testschema zu entfernen; eigener Code dargestellt mit carbon.now.sh	57
Abbildung 26: typische Integrationstest, der einen erfolgreichen Datenbankeintrag prüft; eigener Code dargestellt mit carbon.now.sh	58

Abbildung 27: Verwendung der „assertRaises“- und "assertTrue"-Funktion; eigener Code dargestellt mit carbon.now.sh.....	59
Abbildung 28: Test, welcher die erfolgreiche Löschung des Administrator-Accounts prüft; eigener Code dargestellt mit carbon.now.sh	60
Abbildung 29: Verwendete Lösung von chatGPT, dargestellt mit carbon.now.sh	XXIII

Tabellenverzeichnis

Tabelle 1: "Mandant- und Administrator-Account erstellen" ; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126.....	28
Tabelle 2: "Nutzer anlegen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126...	28
Tabelle 3: struktureller Aufbau von Use Cases, bei denen neue Daten eingetragen werden; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126	29
Tabelle 4: Use Cases, die der in Tabelle 3 dargestellten Ablaufstruktur folgen; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126	31
Tabelle 5: "Neuen Mitarbeiter anlegen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126.....	32
Tabelle 6: "Außertariflichen Vergütungsbestandteil einfügen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126.....	32
Tabelle 7: "Adresse aktualisieren"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126	33
Tabelle 8: "Mitarbeiterentlassung eintragen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126	34
Tabelle 9: "Abteilungshierarchie erstellen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126	35
Tabelle 10: "Krankenversicherungsbeiträge aktualisieren"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126.....	36
Tabelle 11: "SQL-Abfrage ausführen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126	36
Tabelle 12: "Mitarbeiterdaten entfernen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126	37
Tabelle 13: "Entsperrung eines Nutzer-Accounts"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126	37
Tabelle 14: "Neuvergabe des Nutzerpassworts nach Entsperrung"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126.....	38
Tabelle 15: "Nutzer-Account entfernen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126	39
Tabelle 16: "Alle Daten des Mandanten entfernen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126	39

1 Einleitung

Die vorliegende Arbeit „Entwicklung einer Personalstammdatenbank als SaaS“ dient der Erlangung des akademischen Grades Bachelor of Science des Studienganges Wirtschaftsinformatik an der Hochschule für Technik und Wirtschaft Berlin. In diesem Kapitel wird die Relevanz des Themas, die Zielsetzung und Abgrenzung sowie den Aufbau der Arbeit beschrieben. Zudem wird ein Hinweis zum Gendern gegeben.

1.1 Relevanz des Themas

Wenn ein Unternehmen einen neuen Mitarbeiter einstellt, müssen verschiedene personenbezogene Daten erhoben und gespeichert werden. Neben den Namen und der Anschrift des Mitarbeiters müssen für die staatlichen Behörden beispielsweise die Steueridentifikationsnummer und Sozialversicherungsnummer angegeben werden können. Unternehmensintern muss unter anderem erfasst werden, welches Entgelt der Mitarbeiter erhält oder in welcher Abteilung und eventuell Tochtergesellschaft der Mitarbeiter tätig ist. In der vordigitalen Zeit geschah dies häufig mit Personalkarteien. Diese sind allerdings fehlerträchtig und aufwendig bei der Korrektur beziehungsweise Aktualisierung, weswegen im Zuge der Digitalisierung die Erfassung auf digitalem Wege empfohlen wird.¹ Zudem können mithilfe digitaler Technologien Personaldaten einmalig gespeichert werden, was als redundanzfreie Speicherung bezeichnet wird.² Hierfür sind relationale Datenbank geeignet.³ Neben der reinen Erfassung der Daten ist auch deren Analyse zwecks Definition, Überwachung und Erfüllung von personalwirtschaftlichen Zielen von Bedeutung.⁴

Allerdings benötigen die Entwicklung, Wartung und Nutzung von relationalen Datenbanken und weiteren Softwarelösungen technisches Verständnis und Ressourcen, welche insbesondere für Start-Ups als zu kapital- und personalintensiv betrachtet werden, weswegen Cloud Computing eine immer wichtigere Option beim Aufbau von IT-Infrastruktur spielt.⁵

Folgerichtig verbreitet sich die Nutzung von Cloud-Technologien zunehmend. Laut einer Presseinformation der bitkom (sic!) von 2023 nutzen bereits 89% der Unternehmen Cloud-

¹ Vgl. Bröckermann, Reiner (2021): Seite 22

² Vgl. Bröckermann, Reiner (2021): Seite 23

³ Vgl. Steiner, René (2021): Seite 14

⁴ Vgl. Bröckermann, Reiner (2021): Seite 399f.

⁵ Vgl. Kratzke, Nane (2022): Seite 1

Lösungen und in den nächsten, zum damaligen Zeitpunkt, 5 Jahren „[...] wollen 56% aller Unternehmen mehr als die Hälfte ihrer IT-Anwendungen aus der Cloud betreiben.“⁶

1.2 Zielsetzung und Abgrenzung

Ziel dieser Arbeit ist die Entwicklung einer prototypischen Personalstammdatenbank als SaaS mit Schwerpunkt auf die Bedürfnisse von privaten Unternehmen. Dabei werden neben Stammdaten auch weitere Daten berücksichtigt, die eng mit den Stammdaten zusammenhängen. Von zentraler Bedeutung für diese Arbeit ist die Mandantenfähigkeit, mit der sichergestellt werden soll, dass verschiedene Unternehmen dasselbe Datenbanksystem verwenden, allerdings nicht die Möglichkeit haben, Daten von jeweils anderen Unternehmen einsehen zu können. Da es mehrere Mandantensysteme gibt, werden diese vorgestellt. Zur Anwendung kommt aber nur exakt eines dieser Systeme.

Ein weiteres Ziel ist, dass die Daten historisiert gespeichert werden können, damit auch vergangenheitsbezogene Abfragen mit nicht mehr aktuellen Daten möglich sind. So sollen Entwicklungen und Änderungen aus der Retrospektive betrachtet werden können.

Im Rahmen dieser Arbeit wird die Personalstammdatenbank zwar als SaaS entwickelt, allerdings nicht in einer Cloud zur Verfügung gestellt. Ebenso wird kein Frontend oder eine Web-App entwickelt, über die die Dateneingabe abgewickelt wird. Für die Dateneingaben werden Excel-Dateien verwendet.

Diese Arbeit erhebt nicht den Anspruch darauf, alle möglichen Anwendungsfälle einer Personalstammdatenbank zu berücksichtigen. Es werden aber beispielhafte Funktionalitäten implementiert, wobei die Anlage eines neuen Mitarbeiters zentral ist. Anhand der implementierten Funktionalitäten wird aber der grundsätzliche Ablauf von Dateneingabe bis Speicherung in der Datenbank aufgezeigt.

Zwecks Prüfung der in dieser Arbeit implementierten Funktionalitäten werden Tests implementiert. Diese stellen allerdings nur einen Teil aller notwendigen Tests dar, welche zur Entwicklung und Auslieferung einer Personalstammdatenbank gehören. Es wird allerdings erläutert, welche Testarten es gibt und wie sie in der Praxis ausgeführt werden können.

⁶ Vgl. bitkom (2023): Cloud-Nutzung wird rasant zunehmen

1.3 Aufbau der Arbeit

Kapitel 2 befasst sich mit den personalwirtschaftlichen Grundlagen. Hierbei wird auf die Frage eingegangen, was Stammdaten sind und wie sie sich von anderen Daten abgrenzen. Zudem wird der Aufbau des deutschen Sozialversicherungssystems und Entgelttarife erklärt.

In Kapitel 3 werden technische Grundlagen wie relationale Datenbanken und SQL, Cloud Computing und SaaS, Mandantenfähigkeit und Testing besprochen. Zudem wird ein Überblick über die in dieser Arbeit verwendeten Technologien gegeben.

In Kapitel 4 werden die Anforderungen an die Personalstammdatenbank erläutert, wobei in funktionale und nicht-funktionale Anforderungen unterschieden wird.

Kapitel 5 befasst sich mit der praktischen Entwicklung des Systems und zeigt unter anderem den Prozess auf, wie die Datenübertragung von Eingabe bis Speicherung in der Datenbank abläuft.

In Kapitel 6 werden die Tests beschrieben, welche die implementierten Funktionalitäten auf ihre Ausführungsqualität prüfen.

In Kapitel 7 werden die Ergebnisse dieser Arbeit zusammengefasst und einen Ausblick gegeben.

1.4 gendergerechte Sprache

Für diese Arbeit wird ausschließlich das generische Maskulinum verwendet. Dies soll keine Diskriminierung anderer Geschlechter darstellen, sondern ausschließlich der besseren Lesbarkeit dienen. Personenbezeichnungen beziehen sich stets auf alle Geschlechter.

2 Personalwirtschaftliche Grundlagen

In diesem Kapitel werden die personalwirtschaftlichen Grundlagen, welche für die Implementation der im Rahmen dieser Arbeit entwickelten Personalstammdatenbank benötigt werden, besprochen. Da die Sozialversicherungen und das Entgelt einen bedeutenden Anteil der Datenbank ausmachen, werden diese Themen fokussiert dargestellt. Zudem wird erläutert, was Stammdaten allgemein sind, und Beispiele in Bezug auf Personalwirtschaft genannt.

2.1 Sozialversicherungen

Sozialversicherungen haben die Aufgabe, den Menschen Schutz vor „[...] allgemeine Lebensrisiken wie Arbeitslosigkeit, Krankheit, Pflegebedürftigkeit, Invalidität, Arbeitsunfall und Berufskrankheit und zum anderen eine Absicherung für das Alter [...]“⁷ zu bieten. In Deutschland besteht die Sozialversicherung aus den fünf Bestandteilen Krankenversicherung, Pflegeversicherung, Rentenversicherung, Arbeitslosenversicherung und Unfallversicherung.⁸ Zudem gibt es gesetzliche Umlagen, welche Entgeltfortzahlung im Krankheitsfall, Mutterschutz und Insolvenzausfallgeld regeln.⁹ In diesem Kapitel werden die fünf Sozialversicherungszweige und gesetzlichen Umlagen beschrieben und welche Sonderregeln für Diese jeweils gelten.

2.1.1 Krankenversicherung

Die Krankenversicherung sichert Arbeitnehmer gegen Krankheiten ab, indem Kosten für beispielsweise ärztliche Behandlungen und Medikamente übernommen¹⁰ werden. Der allgemeine Beitragssatz für gesetzlich Versicherte beträgt im Jahr 2024 14,6% und der verminderte Beitragssatz 14,0%, wobei diese paritätisch, also zu gleichen Teilen, von Arbeitgeber und Arbeitnehmer bezahlt wird.¹¹ Beim verminderten Beitragssatz besteht im Gegensatz zum allgemeinen Beitragssatz kein Anspruch auf Krankengeld. Dies betrifft

⁷ Bundesministerium für Arbeit und Soziales (2021): Sozialversicherung

⁸ Vgl. Bundesministerium für Arbeit und Soziales (2021): Sozialversicherung

⁹ Vgl. Krankenkasseninfo (o.D.): U1 Umlage, U2 und U3

¹⁰ Vgl. Bundesministerium für Gesundheit (2016): Leistungskatalog der Krankenversicherung

¹¹ Vgl. Die Techniker (o.D. – a): TK-Beitragssatz

beispielsweise Empfänger von Altersrenten, welche nebenbei noch einer sozialversicherungspflichtigen Beschäftigung nachgehen.¹²

Des Weiteren wird ein Zusatzbeitrag erhoben, dessen Höhe von der Krankenkasse abhängig ist, bei der der Arbeitnehmer Mitglied ist und seit dem 01.01.2019 ebenfalls wieder paritätisch von Arbeitgeber und Arbeitnehmer bezahlt wird.¹³

Die Beitragsbemessungsgrenze, also das Gehalt, ab dem die Beiträge nicht mehr ansteigen, liegt seit 2024 bei 62.100€/Jahr.¹⁴

Arbeitnehmer haben die Möglichkeit, in eine private Krankenversicherung zu wechseln, wenn sie 2024 ein Gehalt von mehr als 69.300€/Jahr, der sogenannten Jahresarbeitsentgeltgrenze, erhalten. Zusätzlich gibt es eine besondere Jahresarbeitsentgeltgrenze für Personen, die am 31.12.2002 privat krankenversichert waren, und 2024 bei 62.100€ Jahresgehalt liegt. Dies wurde eingeführt, weil zum Jahreswechsel 2002/2003 die Beitragsbemessungsgrenze von der Jahresarbeitsentgeltgrenze entkoppelt wurden und so den betroffenen Personen, die zwar über der Beitragsbemessungsgrenze, aber nun unter der Jahresarbeitsentgeltgrenze liegen, Bestandsschutz zu gewähren.¹⁵

2.1.2 Pflegeversicherung

Die Pflegeversicherung sichert Arbeitnehmer bei Pflegebedürftigkeit ab und übernimmt, häufig allerdings nur teilweise, Kosten für die Pflege bzw. leistet Zahlungen, wenn eine Person Angehörige pflegt. Die Pflegeversicherung ist eine eigenständige Säule der Sozialversicherung, ist aber eng an die Krankenversicherung gekoppelt.¹⁶ So sind die Beitragsbemessungsgrenzen und die Jahresarbeitsentgeltgrenzen identisch mit denen der Krankenversicherung.^{17 18}

Der allgemeine Beitragssatz beträgt seit Juli 2023 3,4%, welcher paritätisch von Arbeitgeber und Arbeitnehmer bezahlt wird. Es gibt allerdings Sonderregeln. So beträgt der Arbeitgeberbeitrag in Sachsen 1,2%, im Rest der Bundesrepublik aber 1,7%.¹⁹ Kinderlose

¹² Vgl. Die Techniker (o.D. - b): ermäßigter Beitragssatz

¹³ Vgl. Bundesministerium für Gesundheit (2023 - b): Beiträge

¹⁴ Vgl. Die Techniker (o.D. - c): Wie hoch sind die Beitragsbemessungsgrenzen?

¹⁵ Vgl. Die Techniker (o.D. - d): Was ist die Jahresarbeitsentgeltgrenze und wie hoch ist sie?

¹⁶ Bundesministerium für Gesundheit (2023 - c): Die Pflegeversicherung

¹⁷ Vgl. Die Techniker (o.D. - c): Wie hoch sind die Beitragsbemessungsgrenzen?

¹⁸ Vgl. Die Techniker (o.D. - d): Was ist die Jahresarbeitsentgeltgrenze und wie hoch ist sie?

¹⁹ Vgl. Die Techniker (o.D. - e): Pflegeversicherungsbeitrag

Beschäftigte, welche das 23. Lebensjahr abgeschlossen haben, zahlen einen Zusatzbeitrag von 0,6%. Jüngere kinderlose Beschäftigte und Beschäftigte, welche vor 1940 geboren wurden, sind von diesem Zusatzbeitrag befreit.²⁰ Der Arbeitnehmerbeitrag ändert sich mit der Anzahl der Kinder unter 25 wie folgt:

- Beschäftigte unter 23 ohne Kinder und Beschäftigte, welche vor 1940 geboren wurden: 1,7%
- Beschäftigte ab 23 Jahre und ohne Kinder: 2,3%
- Beschäftigte mit Kindern, die alle über 25 sind: 1,7%
- Ein Kind unter 25: 1,7%
- Zwei Kinder unter 25: 1,45%
- Drei Kinder unter 25: 1,2%
- Vier Kinder unter 25: 0,95%
- Fünf oder mehr Kinder unter 25: 0,7%

Die Vergünstigungen für Eltern mit mehr als einem Kind unter 25 gelten auch, wenn die Eltern selbst noch unter 23 sind.²¹

2.1.3 Arbeitslosenversicherung

Die Arbeitslosenversicherung unterstützt Personen, welche ihr Erwerbseinkommen verlieren, durch Arbeitslosengeld. Träger der Arbeitslosenversicherung ist die Bundesagentur für Arbeit.²² In die Arbeitslosenversicherung müssen, bis auf einige Ausnahmen wie beispielsweise Werkstudenten, alle Beschäftigten einzahlen.²³ Der aktuelle Beitragssatz beträgt 2,6% und wird zu gleichen Teilen von Arbeitgeber und Arbeitnehmer bezahlt.²⁴ Die Beitragsbemessungsgrenzen liegen in West-Berlin und West-Deutschland im Jahr 2024 bei 90.600€ Jahresentgelt und in Ost-Berlin und Ost-Deutschland bei 89.400€.²⁵

²⁰ Vgl. Deutsche Rentenversicherung (o.D. - a): Beitragszuschlag für Kinderlose

²¹ Vgl. Die Techniker (o.D. - e): Pflegeversicherungsbeitrag

²² Vgl. Deutsche Rentenversicherung (o.D. - b): Arbeitslosenversicherung / Arbeitsförderung

²³ Vgl. SGB III §27 (1997): Arbeitsförderung

²⁴ Vgl. Die Techniker (o.D. - f): Wie sind die aktuellen Beitragssätze in der Sozialversicherung

²⁵ Vgl. Die Techniker (o.D. - c): Wie hoch sind die Beitragsbemessungsgrenzen?

2.1.4 Rentenversicherung

Die Rentenversicherung zahlt Personen nach Ausscheiden aus dem Arbeitsleben im Alter und ihren Hinterbliebenen im Todesfall eine Rente aus.²⁶ Sie hat 16 verschiedene Träger, die meist einer bestimmten Region der Bundesrepublik Deutschland zugeordnet ist.²⁷ Bis auf wenige Ausnahmeregelungen sind alle Arbeitnehmer, bei der Rentenversicherung pflichtversichert.²⁸ Der aktuelle Beitragssatz beträgt 18,6%, welche paritätisch von Arbeitgeber und Arbeitnehmer bezahlt werden.²⁹ Die Beitragsbemessungsgrenzen sind identisch mit denen der Arbeitslosenversicherung.³⁰

2.1.5 Unfallversicherung

Die gesetzliche Unfallversicherung unterstützt mit Prävention, Rehabilitation und Entschädigung, wenn Beschäftigte im Rahmen ihrer Arbeit verunfallen. Sie wird von den Berufsgenossenschaften und Unfallkassen getragen und unterstehen dem Spitzenverband „Deutsche Gesetzliche Unfallversicherung“ (DGUV).³¹

Die gesetzliche Unfallversicherung wird durch Beiträge der Unternehmen finanziert. Im Gegensatz zu den anderen vier Zweigen der Sozialversicherung werden sie nicht ausschließlich auf Grundlage des Arbeitsentgeltes der Mitarbeiter berechnet. Neben den Arbeitsentgelten werden bei der Ermittlung der Beiträge der Finanzbedarf der jeweiligen Berufsgenossenschaften und die Gefahrklassen, also die Abstufung nach Unfallgefahr, berücksichtigt. Die genaue Beitragshöhe des Unternehmens wird erst mit Ablauf des Kalenderjahres ermittelt.³²

Da ein Unternehmen den Finanzbedarf seiner Berufsgenossenschaft nicht genau kennt, ist es somit nicht möglich, die genaue Höhe der Unfallversicherungsbeiträge des aktuellen Jahres beziehungsweise Monats genau vorzuberechnen. Möchte ein Unternehmen nun die Höhe des Unfallversicherungsbeitrags für einen bestimmten Mitarbeiter kalkulieren, geht das nur mit der Beitragshöhe des Vorjahres und mit folgender Berechnung:

²⁶ Vgl. Deutsche Rentenversicherung (o.D. - c): Unternehmensprofil

²⁷ Vgl. Deutsche Rentenversicherung (o.D. - d): Meinen Rentenversicherungsträger finden

²⁸ Vgl. Deutsche Rentenversicherung (o.D. - e): Pflichtversicherte und freiwillig Versicherte

²⁹ Vgl. Die Techniker (o.D. - f): Wie sind die aktuellen Beitragssätze in der Sozialversicherung

³⁰ Vgl. Die Techniker (o.D. - c): Wie hoch sind die Beitragsbemessungsgrenzen?

³¹ Vgl. Deutsche Gesetzliche Unfallversicherung (o.D. - a): Wir über uns

³² Vgl. Deutsche Gesetzliche Unfallversicherung (o.D. - b): Kein Buch mit sieben Siegeln: Die Beitragsberechnung

Arbeitsentgelt Mitarbeiter / Summe aller unfallversicherungspflichtigen Arbeitsentgelte *
Unfallversicherungsbeitrag

2.1.6 Gesetzliche Umlagen

Neben den fünf Sozialversicherungszweigen gibt es noch drei gesetzliche Umlagen, an den die Arbeitgeber unter Umständen teilnehmen müssen.

Die Umlage U1 gleicht die Entgeltfortzahlungskosten für kranke Mitarbeiter bis zu einem bestimmten Anteil des Gehalts aus. Teilnahmeverpflichtet sind dabei aber nur Unternehmen mit regelmäßig nicht mehr als 30 Arbeitnehmern.³³ Die Beitragshöhe hängt von der Krankenkasse ab, bei dem der erkrankte Mitarbeiter versichert ist, sowie vom Erstattungssatz, also der prozentuale Anteil des Gehalts, welches erstattet wird. Viele Krankenkassen bieten mehrere Erstattungssätze an, die alle wiederum einen verschiedenen Beitragssatz haben.³⁴

Die Umlage U2 übernimmt die Entgeltfortzahlungskosten für Mitarbeiterinnen, welche sich in Mutterschutz befinden, in kompletter Höhe.³⁵ Der Mutterschutz gilt für schwangere Frauen, welche sich im Zeitraum von sechs Wochen vor bis acht Wochen nach der Geburt befinden (bei behinderten Kindern, Früh- oder Mehrlingsgeburten zwölf Wochen nach Geburt) und/oder in einer Tätigkeit beschäftigt sind, die einem generellem Beschäftigungsverbot für Schwangere unterliegen und/oder aufgrund eines ärztlichen Attests aufgrund ihrer Schwangerschaft nicht arbeiten dürfen.³⁶ Die Beitragshöhe hängt auch hier von der Krankenkasse ab, bei der die schwangere Mitarbeiterin Mitglied ist.³⁷ Alle Arbeitgeber müssen an der U2-Umlage teilnehmen.³⁸

Die dritte gesetzliche Umlage ist die Insolvenzgeldumlage, die für fast alle privaten Arbeitgeber (bis auf Privathaushalte und Wohnungseigentümergeinschaften) verpflichtend ist und bis auf ausländische Saisonarbeitskräfte alle Arbeitnehmer umfasst, und

³³ Vgl. Die Techniker (o.D. - g): Entgeltfortzahlungsversicherung - Wer kann sich versichern?

³⁴ Vgl. KKH (o.D.): Absicherung im Krankheitsfall, Mutterschutz und Insolvenz

³⁵ Vgl. KKH (o.D.): Absicherung im Krankheitsfall, Mutterschutz und Insolvenz

³⁶ Vgl. Die Techniker (o.D. - h): U2 - Mutterschaft: Welche Beschäftigungsverbote gibt es?

³⁷ Vgl. KKH (o.D.): Absicherung im Krankheitsfall, Mutterschutz und Insolvenz

³⁸ Vgl. Die Techniker (o.D. - g): Entgeltfortzahlungsversicherung - Wer kann sich versichern?

dient dazu, im Falle einer Insolvenz des Arbeitgebers die Entgelte der Arbeitnehmer zu sichern.³⁹ Die Beitragshöhe liegt seit 2023 bei 0,06%.⁴⁰

2.1.7 Versicherungspflichten nach Mitarbeitertypen

Das im Rahmen dieser Arbeit entwickelte Datenmodell richtet sich an die Bedürfnisse der Privatwirtschaft. Hier gibt es eine Vielzahl von Mitarbeitertypen. Berücksichtigt werden hier Angestellte, kurzfristige beschäftigte Angestellte, privat versicherte Angestellte, Werkstudenten, Auszubildende, Minijobber, kurzfristig beschäftigte Minijobber, Pflichtpraktika, freiwillige Vor- und Nachpraktika ohne Entgelt, verpflichtende Vor- und Nachpraktika und freiwillige Vor- und Nachpraktika mit Entgelt.

Sofern sie die Jahresentgeltgrenze nicht überschreiten, sind nicht kurzfristig beschäftigte Arbeitnehmer und Auszubildende in allen Zweigen der Sozialversicherung und den gesetzlichen Umlagen voll versicherungspflichtig (Umlage U1 nur, wenn Unternehmen weniger als 30 Angestellte hat).^{41 42 43}

Dem gegenüber sind kurzfristig beschäftigte Arbeitnehmer „[...] in allen Versicherungszweigen versicherungs- sowie beitragsfrei, sofern keine berufsmäßige Beschäftigung vorliegt.“⁴⁴ Ausnahmen sind hierbei die gesetzliche Unfallversicherung⁴⁵ und die drei gesetzlichen Umlagen⁴⁶, wo deren Entgelt berücksichtigt beziehungsweise Beiträge zu entrichten sind. Eine kurzfristige Beschäftigung liegt vor, wenn die Dauer drei zusammenhängende Monate oder 70 Tage im Jahr nicht überschreitet. Keine berufsmäßige Beschäftigung liegt vor, wenn die Tätigkeit von untergeordneter wirtschaftlicher Bedeutung für die betreffende Person ist.⁴⁷

Arbeitnehmer, welche über der Jahresentgeltgrenze liegen, müssen zwar nicht mehr in die gesetzliche Kranken- und Pflegeversicherung einzahlen, sind aber verpflichtet, sich privat zu kranken- und pflegeversichern⁴⁸. Privat versicherte Arbeitnehmer haben Anspruch auf einen

³⁹ Vgl. Die Techniker (o.D. – i): Die Insolvenzgeldumlage 2024

⁴⁰ Vgl. Deutsche Rentenversicherung (o.D. - f): Insolvenzgeldumlage

⁴¹ Vgl. AOK (o.D. - a): Sozialversicherungspflicht und -freiheit

⁴² Vgl. AOK (o.D. - b): Auszubildende und Sozialversicherung

⁴³ Vgl. Deutsche Rentenversicherung (o.D. - f): Insolvenzgeldumlage

⁴⁴ Deutsche Rentenversicherung (o.D. - g): Kurzfristige Beschäftigung

⁴⁵ Vgl. Bundesministerium für Arbeit und Soziales (2024): Kurzfristige Beschäftigung

⁴⁶ Vgl. IHK Regensburg (o.D.): Minijobs und Kurzfristige Beschäftigung

⁴⁷ Vgl. Deutsche Rentenversicherung (o.D. - g): Kurzfristige Beschäftigung

⁴⁸ Vgl. Bundesministerium für Gesundheit (2023 - d): Private Krankenversicherung (PKV)

Zuschuss vom Arbeitgeber.⁴⁹ ⁵⁰ Weiterhin sind diese Arbeitnehmer verpflichtend in der Renten- und Arbeitslosenversicherung⁵¹, sowie in der Unfallversicherung⁵² versichert und zudem umlagepflichtig.⁵³

Werkstudenten sind nicht über den Arbeitgeber kranken-, pflege- und arbeitslosenversichert. Für die Kranken- und Pflegeversicherung sind Werkstudenten eigenverantwortlich, beispielsweise indem sie über ihre Familie versichert sind oder eine studentische Kranken- und Pflegeversicherung abschließen.⁵⁴ Die Renten⁵⁵- und Unfallversicherung⁵⁶ über den Arbeitgeber besteht weiterhin. Sie sind zudem umlagepflichtig.⁵⁷

Für Minijobber hat der Arbeitgeber Minijob-Pauschalen für die Kranken-, Renten- und Unfallversicherung sowie für die drei gesetzlichen Umlagen zu entrichten. Pflege- und Arbeitslosenversicherung entfallen.⁵⁸ Der Minijobber muss zudem Pauschalen zur Rentenversicherung leisten, wenn er sich nicht davon befreien lässt.⁵⁹

Kurzfristige Minijobber sind umlagepflichtig und sind in der Unfallversicherung zu berücksichtigen. In den restlichen Sozialversicherungszweigen werden keine Beiträge abgeführt.⁶⁰

Verpflichtende Praktika während des Studiums sind nicht kranken-, pflege-, arbeitslosen- und rentenversicherungspflichtig, unabhängig davon, wie lange es dauert und wie hoch das Entgelt ist.⁶¹

Für freiwillige Vor- und Nachpraktika (einschließlich Schülerpraktikum) ohne Entgelt gelten dieselben Regelungen.⁶²

⁴⁹ Vgl. Die Techniker (o.D. - j): Wie hoch ist der Arbeitgeberzuschuss, wenn meine Beschäftigten privat versichert sind?

⁵⁰ Vgl. Bundesministerium für Gesundheit (2017): Private Pflege-Pflichtversicherung

⁵¹ Vgl. BARMER (2022): Einzugsstelle Sozialversicherung

⁵² Vgl. Bundesministerium für Arbeit und Soziales (o.D.): Wer ist unfallversichert?

⁵³ Vgl. BKK Landesverband Mitte (o.D.): Häufig gestellte Fragen

⁵⁴ Vgl. AOK Nordost (2024): Beitrag zur Krankenversicherung bei einem Nebenjob als Studierender

⁵⁵ Vgl. AOK (o.D. - c): Beschäftigung von Werkstudenten

⁵⁶ Vgl. BGHW (o.D.): Wer ist versichert?

⁵⁷ Vgl. AOK (o.D. - d): Werkstudenten beschäftigen

⁵⁸ Vgl. Minijob-Zentrale (o.D. - a): Der gewerbliche Minijob: Abgaben und Steuern

⁵⁹ Vgl. Minijob-Zentrale (o.D. - b): Rentenversicherungspflicht im Minijob

⁶⁰ Vgl. Minijob-Zentrale (o.D. - a): Der gewerbliche Minijob: Abgaben und Steuern

⁶¹ Vgl. Die Techniker (o.D. - k): Sind Praktikant:innen sozialversicherungspflichtig oder -frei?

⁶² Vgl. Die Techniker (o.D. - l): Welche Praktika sind vollständig versicherungsfrei?

Ist ein Vor- oder Nachpraktikum verpflichtend, besteht Versicherungspflicht in allen Zweigen der Sozialversicherung, selbst wenn kein Entgelt gezahlt wird.⁶³

Freiwillige Vor- und Nachpraktika, welche vergütet werden, sind sozialversicherungspflichtig. Liegt das Entgelt bei nicht mehr als 538€, kann die Minijob-Regelung in Anspruch genommen werden. Bei freiwilligen Praktika während des Studiums greift die Werkstudentenregelung.⁶⁴

Alle Praktikanten sind in der Unfallversicherung zu berücksichtigen⁶⁵. Praktikanten sind umlagepflichtig, sofern sie ein Entgelt bekommen werden.⁶⁶

Folgende Abbildung verdeutlicht, welche Sozialversicherungen bei welchem Mitarbeitertypen über den Arbeitgeber erfolgen und welche nicht:

	Nicht kurzfristige Angestellte	kurzfristige Angestellte	Privat Versicherte	Werkstudenten	Auszubildende	nicht kurzfristige Minijobber	kurzfristige Minijobber	Pflichtpraktika	freiwillige Vor-/Nach-/Schülerpraktika ohne Entgelt	verpflichtende Vor-/Nachpraktika	freiwillige Praktika, Entgelt über 538€
krankenversichert	Ja	Nein	nein	nein	Ja	Pauschale	keine Pauschale	nein	nein	ja	ja
pflegeversichert	Ja	Nein	nein	nein	Ja	keine Pauschale	keine Pauschale	nein	nein	ja	ja
arbeitslosenversichert	Ja	Nein	ja	Nein	Ja	keine Pauschale	keine Pauschale	nein	nein	ja	ja
rentenversichert	Ja	Nein	ja	Ja	Ja	Pauschale für AG; AN-Pauschale bei Nichtbefreiung	keine Pauschale	nein	nein	ja	ja
unfallversichert	Ja	ja	ja	Ja	Ja	Pauschale	Pauschale	ja	ja	ja	ja
umlagepflichtig	ja	Ja	ja	ja	ja	Pauschale	Pauschale	ja, bei Entgelt	nein	ja, bei Entgelt	ja

Abbildung 1: Übersicht für welche Beschäftigungsformen welche Sozialabgaben zu leisten sind; eigene Darstellung mit Excel

2.2 Entgelt und Tarifbindung

Das Entgelt ist die Gegenleistung der Arbeitgeber für die erbrachte Arbeit der Arbeitnehmer. Es wird zwischen verschiedenen Entgeltformen unterschieden: Arbeitsentgelte, Honorare als Entgelte für freie Mitarbeiter und leistungsbezogene Komponente wie beispielsweise Prämienlohn. Für angestellte Arbeitnehmer sind besonders Ersteres von Belang. Das Entgelt setzt sich zusammen aus einer Grundvergütung, wie zum Beispiel ein Tarifgehalt, und eventuell zusätzlichen Vergütungen wie Zulagen.⁶⁷

Tarifverträge sind Vereinbarungen zwischen einem Arbeitgeber oder Arbeitgeberverband und einer oder unter Umständen auch mehreren Gewerkschaften. Es ist zwischen Haustarifverträgen und Verbandstarifverträgen zu unterscheiden, wobei der Haustarif nur bei

⁶³ Vgl. Die Techniker (o.D. - m): Welche Vor- und Nachpraktika sind versicherungspflichtig?

⁶⁴ Vgl. AOK (o.D. - e): Praktikanten in der Sozialversicherung

⁶⁵ Vgl. BG BAU (o.D.): Praktikanten

⁶⁶ Vgl. Haufe (o.D.): Praktikanten: Beurteilung in der Entgeltabrechnung / 3.4 Beiträge zur Unfallversicherung sowie den Umlagen U1, U2 und U3

⁶⁷ Vgl. Bröckermann, Reiner (2021): Seite 189f.

einem Arbeitgeber gilt, während Verbandstarife für alle Arbeitgeber eines Arbeitgeberverbandes gelten. Wichtiger Bestandteil von Tarifverträgen sind dabei die Entgeltregelungen für die angestellten Mitarbeiter. Die Entgelte werden hierbei regelmäßig, meist nach einem Jahr, neu verhandelt.⁶⁸

Gemäß dem statistischen Bundesamt waren 2022 in Deutschland rund 41% der Arbeitnehmer in einer tarifvertraglich geregelten Beschäftigung tätig.⁶⁹

2.3 Personalstammdaten

Im Daten- und Informationsmanagement werden Daten in Bezug auf die Dauer ihrer Gültigkeit in zwei Arten unterschieden: Stamm- und Bewegungsdaten. Stammdaten werden als Daten definiert, die sich nach Eingabe nur selten ändern.⁷⁰ In Bezug auf Personaldaten sind dies beispielsweise Name, Adresse, Geburts-, Eintritts- und Austrittsdatum, Geschlecht, Tarife, Entgelte, die wöchentliche Arbeitszeit sowie Steuerklasse, Tätigkeit und Kostenstelle bzw. Abteilungszugehörigkeit.⁷¹ Nach der oben beschriebenen Definition können auch der Sozialversicherungsstatus, also die Frage ob beispielsweise jemand privat oder gesetzlich krankenversichert ist beziehungsweise die Versicherungspflicht in der Arbeitslosen-, Renten- und Unfallversicherung, die Teilnahmepflicht an den gesetzlichen Umlagen sowie den Mitarbeitertyp zu den Stammdaten zählen, da sich diese ebenfalls nur selten für einen Arbeitnehmer ändern.

Bewegungsdaten entstehen erst im Zuge eines Geschäftsvorfalles und müssen stets neu eingegeben oder berechnet werden.⁷² Dies sind im personalwirtschaftlichen Kontext beispielsweise Gehaltsabrechnungen.

⁶⁸ Vgl. Bröckermann, Reiner (2021): Seite 193

⁶⁹ Vgl. destatis (o.D.): Tarifbindung von Arbeitnehmern

⁷⁰ Vgl. Hildebrand, Knut / Gebauer, Marcus / Hinrichs, Holger / Mielke, Michael (2018): Seite 145

⁷¹ Vgl. Bröckermann, Reiner (2021): Seite 407

⁷² Vgl. Hildebrand, Knut / Gebauer, Marcus / Hinrichs, Holger / Mielke, Michael (2018): Seite 145

3 Technologische Grundlagen

In diesem Kapitel werden technologische Grundlagen in ihrer Theorie abgehandelt. Es wird aber kein umfassender Einblick gegeben, sondern nur die Themen abgehandelt, welche für die Entwicklung der Personalstammdatenbank von Bedeutung sind. Zudem werden die Technologien benannt, die im Rahmen dieser Arbeit zum Einsatz gekommen sind.

3.1 Relationale Datenbanken und SQL

Relationale Datenbanken bilden ein Datenbankmodell, wo Daten in mindestens einer meist aber mehreren über Schlüssel verbundene Tabellen gespeichert und abgefragt werden können. Sie sind die am häufigsten verwendeten Datenbanken.⁷³ Eine Erweiterung von relationalen Datenbanken sind sogenannte objektrrelationale Datenbanken. Diese können zusätzlich beliebige benutzerdefinierte Datentypen verwenden, während relationale Datenbanken festgelegte Standarddatentypen haben.⁷⁴

Neben den Tabellen bestehen (objekt-)relationale Datenbanken aus folgenden Bestandteilen:

- Datensätze, welche die Zeilen einer Tabelle ausmachen.
- Attribute, welche die Spalten einer Tabelle darstellen.
- Datenfelder, welche einen konkreten Wert in einer Spalte eines Datensatzes beschreiben.
- Schlüssel, welche jeden Datensatz eindeutig identifizieren und als Verknüpfung zwischen zwei Tabellen einer Datenbank dienen.⁷⁵

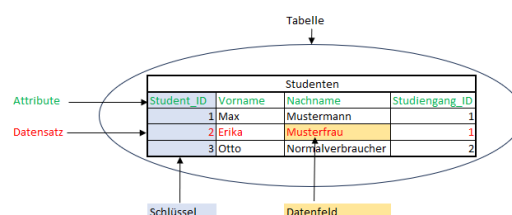


Abbildung 2: Aufbau einer Tabelle in einer relationalen Datenbank, eigene Darstellung mit Excel

Die Tabellen stehen in Beziehung zueinander. Es wird zwischen drei Beziehungstypen unterschieden: 1:1-, 1:n- und m:n-Beziehung. Bei einer 1:1-Beziehung ist ein Datensatz aus Tabelle A mit genau einem Datensatz aus Tabelle B verbunden. Die Verknüpfung mit genau

⁷³ Vgl. Bühler, Peter / Schlaich, Patrick / Sinner, Dominik (2019): Seite 52

⁷⁴ Vgl. Steiner, René (2021): Seite 8

⁷⁵ Vgl. Bühler, Peter / Schlaich, Patrick / Sinner, Dominik (2019): Seite 52f.

einem Datensatz gilt wechselseitig.⁷⁶ So hat beispielsweise in einer Hochschuldatenbank, bei der es eine Tabelle „Studierendenausweise“ und eine Tabelle „Studenten“, jeder Student genau ein Studiausweis und jeder Studiausweis ist genau einem Studenten zugeordnet.

Bei einer 1:n-Beziehung ist ein Datensatz aus Tabelle A mit einem Datensatz aus Tabelle B verknüpft, welche mit mehreren Datensätzen aus Tabelle A verbunden sein kann.⁷⁷ So hat ein Studiengang in der Regel mehrere Studenten, aber jeder Student ist genau einem Studiengang zugeordnet (sofern die Hochschule nur die Einschreibung in genau einem Studiengang erlaubt).

Bei einer n:m-Beziehung kann ein Datensatz aus Tabelle A mit mehreren Datensätzen aus Tabelle B verknüpft sein und auch andersrum kann ein Datensatz aus Tabelle B mit mehreren Datensätzen aus Tabelle A verknüpft sein. Technisch muss bei einer n:m-Beziehung eine Zwischentabelle eingefügt werden, wodurch aus einer n:m- zwei 1:n-Beziehungen entstehen. Dies ist notwendig, da andernfalls keine eindeutige Beziehung zwischen zwei Tabellen hergestellt werden kann. Zwischentabellen können darüber hinaus zusätzliche Informationen über die Beziehung enthalten.⁷⁸ So hat beispielweise ein Student mehrere Vorlesungen und eine Vorlesung hat mehrere Studenten. Eine Vorlesung wird vom Studenten in einem bestimmten Semester belegt. In folgender Abbildung werden diese Beziehungstypen exemplarisch dargestellt:

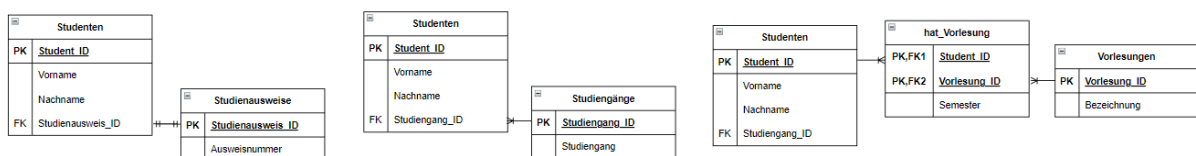


Abbildung 3: Links: 1:1-Beziehung, Mitte: 1:n-Beziehung, Rechts: n:m-Beziehung; eigene Darstellung mit draw.io

SQL steht für „Structured Query Language“ und gilt als Standardsprache für relationale Datenbanken.⁷⁹ Die Sprache besteht aus vier Bestandteilen: Data Definition Language für den Aufbau von Datenstrukturen wie beispielsweise Tabellen, Data Manipulation Language für die Eingabe, Löschung und Veränderung von Daten, Data Retrieval Language für die Abfrage von

⁷⁶ Vgl. Bühler, Peter / Schlaich, Patrick / Sinner, Dominik (2019): Seite 58

⁷⁷ Vgl. Bühler, Peter / Schlaich, Patrick / Sinner, Dominik (2019): Seite 58

⁷⁸ Vgl. Bühler, Peter / Schlaich, Patrick / Sinner, Dominik (2019): Seite 58f.

⁷⁹ Vgl. Steiner, René (2021): Seite 143

Daten und Data Security Language für die Einrichtung von Schutzmaßnahmen gegen unberechtigte Zugriffe.⁸⁰

3.2 Cloud Computing und SaaS

Software as a Service (SaaS) ist neben Infrastructure as a Service (IaaS) und Platform as a Service (PaaS) eines von drei Service-Modellen des Cloud Computings. Gemäß dem Bundesamt für Sicherheit in der Informationstechnik (BSI) konnte sich bisher keine generelle Definition für Cloud Computing durchsetzen.⁸¹ Allerdings wird in der Fachwelt meist die Definition der National Institute of Standards and Technology (NIST), der US-amerikanischen Standardisierungsstelle, herangezogen, welche lautet:

"Cloud Computing ist ein Modell, das es erlaubt bei Bedarf, jederzeit und überall bequem über ein Netz auf einen geteilten Pool von konfigurierbaren Rechnerressourcen (z. B. Netze, Server, Speichersysteme, Anwendungen und Dienste) zuzugreifen, die schnell und mit minimalem Managementaufwand oder geringer Serviceprovider-Interaktion zur Verfügung gestellt werden können."⁸²

Auf Grundlage dieser Definition charakterisieren fünf Eigenschaften das Cloud Computing. Die folgenden Beschreibungen sind von Kratzke entnommen⁸³:

On-Demand Self-Service: der Cloud-Nutzer kann IT-Ressourcen nach Bedarf anfordern, ohne dass der Anbieter händisch eingreifen muss.

Netzwerkzugriff: IT-Ressourcen sind über das Internet mithilfe standardisierter Internetprotokolle verfügbar.

Elastizität: IT-Ressourcen werden bei Bedarf schnell bereitgestellt und nach erfolgter Nutzung ebenso schnell wieder freigegeben.

Messung der Ressourcennutzung: die Nutzung der IT-Ressourcen wird überwacht, um eine dynamische Anpassung des Bedarfs vornehmen zu können.

⁸⁰ Vgl. Steiner, René (2021): Seite 6f.

⁸¹ Vgl. Bundesamt für Sicherheit in der Informationstechnik (o.D.): Was ist Cloud Computing?

⁸² Bundesamt für Sicherheit in der Informationstechnik (o.D.): Was ist Cloud Computing?

⁸³ Vgl. Kratzke, Nane (2022): Seite 13

Ressourcen-Pooling: die IT-Ressourcen werden durch den Anbieter so gebündelt, dass der Zugriff darauf durch die Cloud-Nutzer in einem Multi-Tenant-Modell organisiert wird.

Bei SaaS handelt es sich um Anwendungssoftware und deren Infrastruktur, welche von Anbietern über die Cloud zur Verfügung gestellt werden. Hierbei kann es sich beispielsweise um Office-Produkte, Kommunikationssoftware oder auch Datenbanken handeln. In der Folge müssen Kunden diese Software weder selbst entwickeln noch die Infrastruktur besitzen, betreiben und warten, um solche Produkte nutzen zu können.⁸⁴

Bei IaaS werden Hardware wie beispielsweise Server, Speicher und Netzwerkinfrastruktur zur Verfügung gestellt, mit der Nutzer unter anderem technische Lastspitzen abfangen können.⁸⁵

Bei PaaS wird eine komplette Infrastruktur und dazugehörige Schnittstellen angeboten, welche von nutzereigenen Anwendungen verwendet wird.⁸⁶

Der ausgelagerte Anteil einer Anwendung ist also bei SaaS am größten, da nicht nur Hardware-Ressourcen (IaaS) und Infrastruktur (PaaS), sondern die komplette Anwendung vom Anbieter zur Verfügung gestellt werden.⁸⁷

3.3 Mandantenfähigkeit und Row Level Security

Der Begriff Mandantenfähigkeit bzw. Multi-tenancy beschreibt, dass mehrere voneinander unabhängige Kunden, gemeinsam eine Applikation, beispielsweise eine Datenbank, nutzen. Die Kunden werden in dem Zusammenhang als Mandanten (englisch: tenants) bezeichnet.⁸⁸ Dabei muss sichergestellt sein, dass die Daten eines Mandanten nicht für andere Mandanten sichtbar sind.⁸⁹

Es werden in der Regel zwischen drei verschiedenen Modellen zur Implementierung von Mandantenfähigkeit in Bezug auf Datenbanken unterschieden: eine Datenbank pro Mandant, eine geteilte Datenbank mit einem Schema pro Mandanten und eine geteilte Datenbank mit

⁸⁴ Vgl. Kratzke, Nane (2022): Seite 16

⁸⁵ Vgl. Kratzke, Nane (2022): Seite 15

⁸⁶ Vgl. Bundesamt für Sicherheit in der Informationstechnik (o.D.): Was ist Cloud Computing?

⁸⁷ Vgl. Kratzke, Nane (2022): Seite 14

⁸⁸ Vgl. Somasundar, Harish (2021): Database Multi tenancy

⁸⁹ Vgl. Lippert, Jan (2019): Mandantenfähigkeit mit PostgreSQL

einem geteilten Schema für alle Mandanten.⁹⁰ Daneben werden manchmal auch hybride Modelle genannt, die mehrere der oben genannten Modelle kombinieren.⁹¹

Bei einer Datenbank pro Mandant wird für jeden Mandanten eine eigene Datenbank zur Verfügung gestellt. Es liegt dann an der Applikation, dass bei einer Anfrage eines Mandanten seine Datenbank verwendet wird.⁹² Wird ein neuer Mandant erstellt, muss für ihn auch eine neue Datenbank erstellt werden.⁹³ Nachfolgende Abbildung illustriert diese Vorgehensweise:

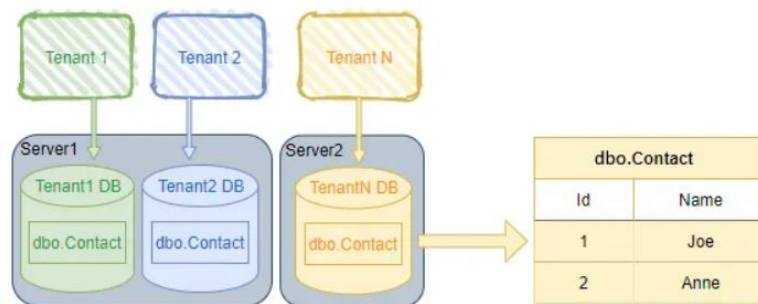


Abbildung 4: jeder Mandant hat eine eigene Datenbank; entnommen aus Somasundar, Harish (2021)

Beim zweiten Modell teilen sich alle Mandanten eine Datenbank, haben aber jeweils ein eigenes Schema. In dem Fall muss die Applikation dann sicherstellen, dass das entsprechende Schema des Mandanten angesprochen wird.⁹⁴ Das Prinzip wird in nachfolgender Abbildung dargestellt:

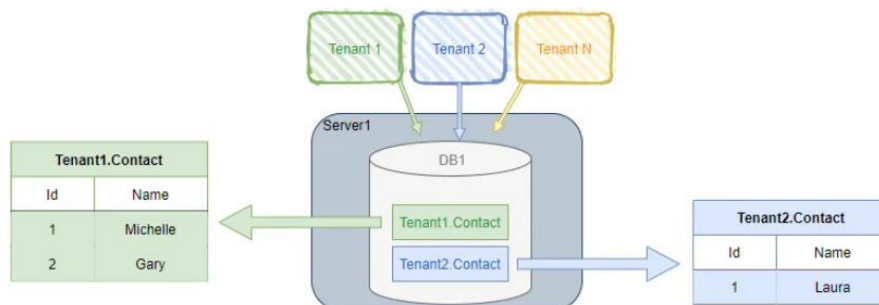


Abbildung 5: Alle Mandanten nutzen eine Datenbank mit separaten Schemata; entnommen aus Somasundar, Harish (2021)

Im dritten Modell arbeiten alle Mandanten in derselben Datenbank auf demselben Schema. Hier ist es notwendig, dass jeder einzelne Datensatz mit einem Identifier ausgestattet ist, der einem Mandanten eindeutig zugeordnet ist.⁹⁵ Der Identifier wird in einer eigens hierfür

⁹⁰ Vgl. Lippert, Jan (2019): Mandantenfähigkeit mit PostgreSQL

⁹¹ Vgl. Somasundar, Harish (2021): Database Multi tenancy

⁹² Vgl. Lippert, Jan (2019): Mandantenfähigkeit mit PostgreSQL

⁹³ Vgl. Somasundar, Harish (2021): Database Multi tenancy

⁹⁴ Vgl. Lippert, Jan (2019): Mandantenfähigkeit mit PostgreSQL

⁹⁵ Vgl. Lippert, Jan (2019): Mandantenfähigkeit mit PostgreSQL

hinzugefügten Spalte in jeder Tabelle der Datenbank eingetragen.⁹⁶ Dies wird in folgender Abbildung verdeutlicht:

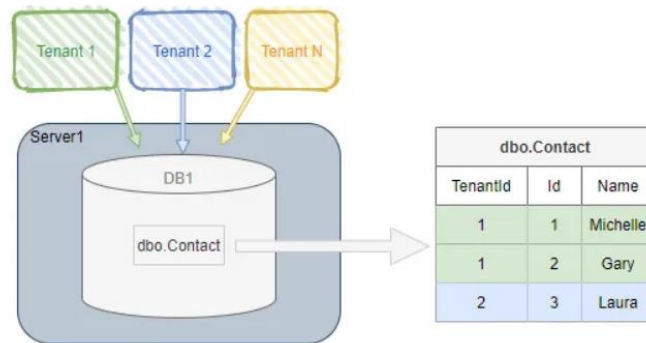


Abbildung 6: Identifier in Spalte "TenantId" bei einer Datenbank mit geteiltem Schema; entnommen aus Somasundar, Harish (2021)

Bei diesem Ansatz kann Row Level Security (RLS) eingesetzt werden. Dies ist eine Filtertechnologie, welche die Zugriffsmöglichkeiten auf Daten auf Grundlage der Identifier regelt. In Abbildung 6 ist der Identifier das Attribut „TenantId“. Hierbei wird bei einer Abfrage der Identifier-Wert des aktuellen Anwenders mit den Werten von „TenantId“ abgeglichen und nur die Daten zur Verfügung gestellt, wo diese übereinstimmen. Auf Datensätze mit einem abweichenden „TenantId“-Wert kann somit nicht zugegriffen werden.⁹⁷

Ein Beispiel für ein hybrides Modell ist ein System, bei dem es mehrere Datenbanken gibt, von denen manche von mehreren Mandanten und andere Datenbanken nur von einem Mandanten verwendet werden.⁹⁸ Solch ein Modell zeigt folgende Abbildung:

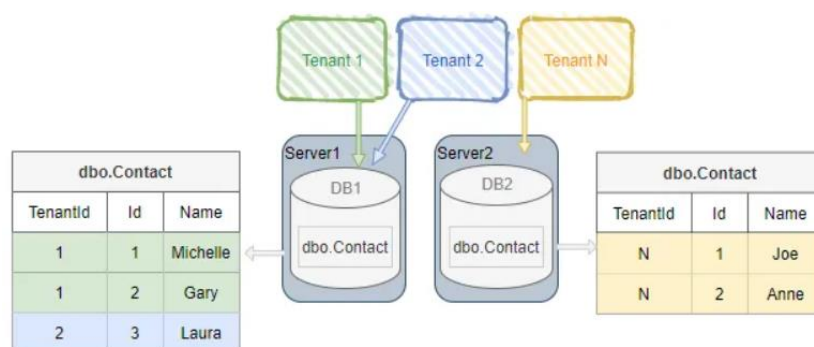


Abbildung 7: Hybride mandantenfähige Datenbank; entnommen aus Somasundar, Harish (2021)

⁹⁶ Vgl. Somasundar, Harish (2021): Database Multi tenancy

⁹⁷ Vgl. aws (o.D.): Empfehlungen auf der Sicherheit auf Zeilenebene (sic!)

⁹⁸ Vgl. Somasundar, Harish (2021): Database Multi tenancy

3.4 Testen

In der Programm- beziehungsweise Software-Entwicklung ist es notwendig, die entwickelten Funktionen auf ihre Richtigkeit zu prüfen. Dies kann manuell geschehen, doch mit der zunehmenden Anzahl an Funktionen würde dies immer aufwendiger werden. Zudem können durch nachträgliche Code-Änderungen in Funktionen, die zuvor funktionierten und erfolgreich manuell getestet wurden, zu Fehlern kommen, die bei ausbleibender manueller Neu-Testung übersehen werden könnten. Deswegen ist es sinnvoll, Testprogramme zu schreiben, welche die Prüfung der Funktionen automatisiert vornehmen.

Es kann zwischen vier verschiedenen Arten von Tests unterschieden werden, welche aufeinander aufbauen: Komponententests (auch „Unit-Tests“ genannt), Integrationstests, Systemtests und Abnahmetests.⁹⁹

Unit-Tests beziehungsweise Komponententests sind automatisierte Testprogramme, die einzelne Funktionen oder Klassen darauf prüfen, ob sie sich erwartungsgemäß verhalten. Sie stellen damit die kleinteiligsten Tests dar. Integrationstests gehen einen Schritt weiter und testen, ob das Zusammenspiel zwischen mehreren Funktionen wie geplant abläuft.¹⁰⁰

Systemtests prüfen die Funktionsfähigkeit eines ganzen Systems. Grundlage hierfür sind die Anforderungsspezifikationen, welche vorab definiert werden. Zudem werden hier Performanz-, Last-, Stress- und Robustheitstests durchgeführt, um zu prüfen, ob auch qualitative Anforderungen erfüllt werden. Zudem werden in Systemintegrationstests auch das Zusammenspiel mit anderen Systemen geprüft.¹⁰¹

Abnahmetests werden unter realen Einsatzbedingungen durchgeführt, um zu testen, ob das System die Anforderungen des Kunden erfüllt. Vor allem die Gebrauchstauglichkeit für die Anwender ist von Bedeutung.¹⁰²

⁹⁹ Vgl. Winter, Mario / Ekssir-Monfared, Mohsen / Sneed, Harry M. / Seidle, Richard / Borner, Lars (2013): Seite 41f.

¹⁰⁰ Vgl. Kersken, Sascha (2019): Seite 722

¹⁰¹ Vgl. Winter, Mario / Ekssir-Monfared, Mohsen / Sneed, Harry M. / Seidle, Richard / Borner, Lars (2013): Seite 41

¹⁰² Vgl. Winter, Mario / Ekssir-Monfared, Mohsen / Sneed, Harry M. / Seidle, Richard / Borner, Lars (2013): Seite 42

3.5 Verwendete Technologien

In diesem Kapitel werden die Programmiersprachen, Bibliotheken und Software-Tools vorgestellt, die im Rahmen dieser Arbeit Verwendung finden.

3.5.1 Excel

Excel ist eine Tabellenkalkulationssoftware der Firma Microsoft, die unter anderem für die Datenanalyse eingesetzt werden kann.¹⁰³ Im Rahmen dieser Arbeit wird Excel für die Dateneingabe verwendet.

3.5.2 PostgreSQL, PL/pgSQL und DBeaver

Für die Erstellung der Personalstammdatenbank wurde PostgreSQL verwendet. PostgreSQL ist ein objekt-relationales Datenbankmanagementsystem (DBMS), dessen Ursprung im POSTGRES-Projekt der University of California at Berkeley Computer Science Department liegt. Es handelt sich um eine Open-Source-Software und unterstützt einen großen Teil des SQL-Standards.¹⁰⁴

PL/pgSQL ist eine prozedurale Programmiersprache mit denen Funktionen innerhalb des PostgreSQL-Datenbankmanagementsystems geschrieben werden können. Wenn ein Nutzer eine festgelegte Reihenfolge von SQL-Befehlen an das DBMS sendet, muss für jede Abfrage eine neue Verbindung zur Datenbank hergestellt werden. Dies führt zu einer ständigen Kommunikation mit dazugehörigen Verbindungsaufbau und -abbau zwischen dem Client und dem Datenbankserver. Mit PL/pgSQL können SQL-Befehle gruppiert und hintereinander ausgeführt werden. Der Nutzer muss nun nicht mehr jeden SQL-Befehl einzeln aussenden, sondern nutzt die PL/pgSQL-Funktion, wodurch mit einer einmaligen Datenbankverbindung sämtliche benötigten SQL-Abfragen abgearbeitet werden können. So kann die Anzahl der benötigten Verbindungen zur DBMS reduziert werden.¹⁰⁵

DBeaver ist ein Datenbankmanagement-Tool, dass verschiedene SQL-Datenbanksysteme wie PostgreSQL und MySQL, aber auch NoSQL-Systeme wie MongoDB unterstützt.¹⁰⁶ Im Rahmen

¹⁰³ Vgl. Kronthaler, Franz (2021): Seite 11

¹⁰⁴ Vgl. PostgreSQL (o.D. - a): What is PostgreSQL?

¹⁰⁵ Vgl. PostgreSQL (o.D. - b): PL/pgSQL - Overview

¹⁰⁶ Vgl. DBeaver (o.D.): About DBeaver

dieser Arbeit wird DBeaver verwendet, um die Datenbank und PL/pgSQL-Funktionen zu entwickeln.

3.5.3 Python und PyCharm

Python ist eine interpretierte Programmiersprache, dessen Quellcode ausgeliefert und während der Laufzeit übersetzt wird. Es handelt sich zudem um eine Multiparadigmensprache, welche nicht nur objektorientierte, sondern auch imperative und funktionale Aspekte berücksichtigt.¹⁰⁷

Neben der Standardbibliothek werden folgende Erweiterungen für Python verwendet:

Psycopg2: ermöglicht eine Verbindung von einem Python-Programm zu einem PostgreSQL-DBMS.¹⁰⁸

Pandas: stellt unter anderem Data Frames für die Datenverarbeitung bereit. Dataframes sind Tabellen, welche mit Spalten- und Zeilenbeschriftungen ausgestattet sind.¹⁰⁹ Pandas wird im Rahmen dieser Arbeit verwendet, um die in Excel eingeschriebenen Daten in Python zu importieren, wo diese dann verarbeitet und letztendlich an die Datenbank übergeben werden.

Unittest: ein Framework, welche die Erstellung von automatisierten Tests in Python unterstützt.¹¹⁰

PyCharm ist eine Entwicklungsumgebung für Python, welche von der Firma JetBrains entwickelt wurde. Neben einer kostenpflichtigen Variante gibt es eine Community Edition, welche Open Source und somit frei verfügbar ist.¹¹¹ Letzteres wird für die Entwicklung der Python-Programme verwendet.

¹⁰⁷ Vgl. Kersken, Sascha (2019): Seite 530f.

¹⁰⁸ Vgl. Psycopg2 (o.D.): PostgreSQL database adapter for Python

¹⁰⁹ Vgl. Pandas (o.D.): Intro to pandas

¹¹⁰ Vgl. Python (o.D. - a): unittest - Unit testing framework

¹¹¹ Vgl. JetBrains (o.D.): PyCharm

4 Anforderungen

Anforderungen beschreiben Fähigkeiten und Bedingungen, die ein System auf Wunsch des Kunden und/oder Gesetzgebers erfüllen können muss. Es werden hierbei zwischen zwei Arten von Anforderungen unterschieden: funktionale Anforderungen und nicht-funktionale Anforderungen.¹¹²

Nicht-funktionale Anforderungen beschreiben die Qualitätsansprüche und wie ein System arbeiten soll. Hierfür wurden mithilfe der „ISO/IEC 25010 Software Produkt Qualitätsmodell“ Merkmale definiert, um Qualität sicherzustellen¹¹³. Diese Merkmale sind:

Geeignete Funktionalität: dieses Merkmal beschreibt, ob das System alle geforderten Funktionen vollständig besitzt, korrekt ausführt und diese angemessen nutzbar sind.¹¹⁴

Performanz/Effizienz: Es muss sichergestellt sein, dass sowohl die Antwortzeit des Systems als auch dessen Ressourcenverbrauch in einem akzeptablen Rahmen bleiben.¹¹⁵

Kompatibilität: Das System muss, vor allem wenn es sich in einer gemeinsamen Umgebung mit anderen Systemen befindet, störungsfrei und ohne Beeinflussung durch anliegende Systeme ausführbar sein. Falls die Umsysteme für die Arbeitsfähigkeit des Systems benötigt werden, sind angemessene Schnittstellen zu implementieren.¹¹⁶

Benutzbarkeit: die Anwendung des Systems ist möglichst so zu gestalten, dass es leicht erlernen- und bedienbar ist. Fehler durch den Nutzer sind möglichst schon bei der Eingabe zu verhindern. Zudem ist auf eine angenehme Benutzerschnittstelle zu achten.¹¹⁷

Zuverlässigkeit: Das System sollte möglichst störungsfrei und verfügbar sein. Bei einem Ausfall von eventuell vorhandenen Umsystemen soll das System ohne Datenverlust und/oder -inkonsistenzen angemessen reagieren können. Bei einem Ausfall des Systems soll eine Wiederherstellung des Zustandes von vor dem Absturz in angemessener Zeit möglich sein.¹¹⁸

¹¹² Vgl. Hruschka, Peter (2023): Seite 11ff.

¹¹³ Vgl. Tremp, Hansruedi (2022): Seite 106

¹¹⁴ Vgl. Tremp, Hansruedi (2022): Seite 107

¹¹⁵ Vgl. Tremp, Hansruedi (2022): Seite 107

¹¹⁶ Vgl. Tremp, Hansruedi (2022): Seite 108

¹¹⁷ Vgl. Tremp, Hansruedi (2022): Seite 108f.

¹¹⁸ Vgl. Tremp, Hansruedi (2022): Seite 109f.

Sicherheit: Das System muss abgesichert werden vor unberechtigten Zugriffen und Veränderungen. Zudem sollen erfolgte Handlungen nachweisbar und die Informationsquellen des Systems einsehbar sein. Eine Authentizitätsprüfung ist einzurichten.¹¹⁹

Wartbarkeit: Änderungen an einer Systemkomponente sollten möglichst geringe Auswirkungen auf andere Systemkomponenten haben. Damit Funktionen verändert oder erweitert werden können, ist eine Dokumentation und Kommentierung einzurichten. Eine Mehrfachverwendbarkeit von Funktionen ist, sofern sinnvoll möglich, anzustreben. Die einzelnen Funktionen eines Systems sollten zudem gut testbar sein.¹²⁰

Übertragbarkeit: Hier geht es darum, ob das System mit sich verändernden Umsystemen umgehen kann. Zudem sollen eine problemlose Installation und Deinstallation möglich sein. Außerdem ist zu klären, mit wieviel Aufwand das System durch ein anderes ersetzt werden kann.¹²¹

Funktionale Anforderungen beschreiben, wozu das System dienen soll, welche Funktionen es bereitstellen muss und wie es sich verhalten soll. Die geforderten Funktionen ergeben sich einerseits aus dem Anwendungsumfeld und somit den Wünschen des Kunden und andererseits aus den nicht-funktionalen Anforderungen.¹²² Bezüglich Letzterem seien als Beispiel Funktionen zur Absicherung vor unberechtigtem Zugriff genannt.

Zur Beschreibung funktionaler Anforderungen können Use Cases verwendet werden. Es sind mindestens so viele Use Cases zu schreiben, bis jede funktionale Anforderung mindestens einmal beschrieben ist.¹²³ Wenn Use Cases verwendet werden, ist auch der Einsatz von User Szenarios sinnvoll. Hier wird in leicht verständlicher Sprache geschrieben, wie verschiedene Nutzer mit dem System umgehen müssen, damit sie eine bestimmte Aufgabe erfüllen können.¹²⁴

Use Cases bauen auf den User Szenarios auf und beschreiben den Ablauf eines Anwendungsfalls, die Bedingungen, die erfüllt sein müssen, damit ein Anwendungsfall

¹¹⁹ Vgl. Tremp, Hansruedi (2022): Seite 110f.

¹²⁰ Vgl. Tremp, Hansruedi (2022): Seite 111f.

¹²¹ Vgl. Tremp, Hansruedi (2022): Seite 112f.

¹²² Vgl. Tremp, Hansruedi (2022): Seite 120

¹²³ Vgl. Tremp, Hansruedi (2022): Seite 122

¹²⁴ Vgl. Crowder, James A. / Hoff, Curtis W. (2022): Seite 125

bearbeitet werden kann, die gewünschten Ergebnisse und wie im Fehlerfall gehandelt werden kann.¹²⁵

4.1 Nichtfunktionale Anforderungen

Im Rahmen dieser Arbeit werden nichtfunktionale Anforderungen nur teilweise berücksichtigt, da einige von ihnen von Faktoren wie Budget, Ressourcen und Zeit abhängig sind, die Einfluss auf beispielsweise die Effizienz oder Sicherheit haben. Im Folgenden werden dennoch alle in Kapitel 4 angesprochenen Merkmale nach „ISO/IEC 25010 Software Produkt Qualitätsmodell“ auf das zu entwickelnde System angewandt. Dadurch wird verdeutlicht, wie die nichtfunktionalen Anforderungen erfüllt sein müssen, damit die Personalstammdatenbank in der Praxis eingesetzt werden kann.

Geeignete Funktionalität: das System erfüllt alle Anforderungen, die aus den Use Cases hervorgehen.

Performanz/Effizienz: die Interaktion mit der Datenbank erfolgt in angemessenem Zeitrahmen. Sowohl Datenmanipulation als auch Abfragen werden in wenigen Sekunden durchgeführt.

Kompatibilität: der Einsatz der Personalstammdatenbank beeinflusst keine andere Software wie beispielsweise E-Mail-Programme oder sonstige Bürosoftware. Falls eine Verknüpfung mit einem anderen System, beispielsweise einer Datenvisualisierungssoftware, erwünscht ist, muss sichergestellt werden, dass die Interaktion zwischen beiden Systemen den Erfordernissen entspricht und Umsysteme weiterhin nicht beeinflusst werden.

Benutzbarkeit: Das System ist so zu gestalten, dass die Eingabe von Daten möglichst leicht erlernbar und durchzuführen ist. Im Fall eines fehlerhaften Gebrauchs gibt das System Meldungen zurück, durch welche die Fehler behoben werden können.

Zuverlässigkeit: die Personalstammdatenbank ist auch nutzbar, wenn bereits viele Nutzer unter Umständen auch von diversen Mandanten auf das System zugreifen. Bei einem Ausfall des Systems ist sichergestellt, dass keine Daten verloren gehen.

¹²⁵ Vgl. Crowder, James A. / Hoff, Curtis W. (2022): Seite 126

Sicherheit: der Zugang ist stets nur authentifizierten und autorisierten Nutzern möglich. Zudem muss sichergestellt sein, dass Nutzer eines Mandanten nicht die Daten eines fremden Mandanten sehen können.

Wartbarkeit: der Quellcode des Systems muss so gestaltet sein, dass bei Änderungen anliegende Umsysteme nicht beeinflusst werden. Zudem ist der Quellcode so zu gestalten, dass bei Änderung einer Funktion nicht die Funktionalität anderer Funktionen beeinträchtigt oder ungewollt verändert wird.

Übertragbarkeit: das System ist so zu gestalten, dass es plattformunabhängig verwendet werden kann. Es soll also beispielsweise keine Bedeutung haben, welches Betriebssystem der Rechner des Nutzers hat.

4.2 Funktionale Anforderungen

In diesem Kapitel werden alle Use Cases vorgestellt, die im Rahmen dieser Arbeit implementiert wurden. Diese handeln überwiegend Anwendungsfälle ab, bei denen Daten in die Datenbank eingetragen, gelesen, aktualisiert oder gelöscht werden. Die implementierten Anwendungsfälle sind beispielhaft und umfassen nicht alle Interaktionsmöglichkeiten, die ein Nutzer mit der Personalstammdatenbank haben muss. Da zudem das entwickelte SaaS-System im Zuge der Arbeit nicht in eine Cloud hochgeladen wird, werden somit die Funktionalitäten, die ein SaaS-System darin benötigt, hier nicht berücksichtigt.

Es gibt für die entwickelte Personalstammdatenbank zwei Rollen: der Administrator und Nutzer. Der Administrator meldet sein Unternehmen im System an, erstellt neue Nutzer, entsperrt gegebenenfalls Nutzer, wenn sie ihr Passwort wiederholt falsch eingeben, kann Nutzer auch wieder entfernen und alle Daten aus der Datenbank löschen. Die Nutzer sind die eigentlichen Anwender, welche die Dienste der Personalstammdatenbank nutzen, indem beispielsweise neue Daten eingetragen, aktualisiert oder abgefragt werden. Jeder Nutzer hat dieselben Interaktionsmöglichkeiten. Eine Ausdifferenzierung von Zugriffsberechtigungen zwischen mehreren Nutzern findet im Rahmen dieser Arbeit nicht statt.

4.2.1 User Szenario

In diesem Kapitel wird der Teil des User Szenarios vorgestellt, welcher im Rahmen der Arbeit berücksichtigt und umgesetzt wird:

Ein Administrator erstellt für das Unternehmen einen Account. Dies ist der Mandant. Gleichzeitig wird auch ein Administrator-Account erzeugt.

Der Administrator erstellt über den Mandanten einen Nutzer-Account und vergibt ein Passwort für ihn. Mit diesem Nutzer-Account kann ein Anwender auf die Datenbank zugreifen. Bei der Erstanmeldung ist jedoch erst eine Passwortneuvergabe durch den Nutzer notwendig.

Ein Nutzer legt Sozialversicherungsdaten an. Dazu gehören die Bezeichnungen von Krankenkassen, deren Beitragssätze und Umlagen. Es werden Beitragsbemessungs- und Jahresentgeltgrenzen hinterlegt. Es werden Beitragssätze zur Pflege-, Renten- und Arbeitslosenversicherung hinterlegt. Zudem wird die Berufsgenossenschaft hinterlegt und, sofern vorhanden, die Unfallversicherungsbeiträge angelegt.

Ein Nutzer gibt, sofern vorhanden, die Daten des Tarifvertrags in die Datenbank ein. Dazu gehören die Tarifbezeichnungen, die Gewerkschaft, mit der sie ausgehandelt wurden und die einzelnen Vergütungsbestandteile.

Ein Nutzer gibt folgende Daten ein, die ebenfalls benötigt werden, um einen Mitarbeiter anzulegen: Geschlecht, Beschäftigungsform, Steuerklasse, Abteilung, Jobtitel, Erfahrungsstufe, Firma, Austrittsgründe und dessen Kategorien.

Ein Nutzer legt einen neuen Mitarbeiter mit seinen persönlichen Daten, seiner Adresse und seiner persönlichen und dienstlichen Telefonnummer und E-Mail-Adresse an. Falls der Mitarbeiter außertariflich bezahlt wird, wird auch seine Vergütung eingegeben. Andernfalls wird der neue Mitarbeiter einem Tarif zugeordnet.

Aufgrund einer Änderung der Beitragssätze zur gesetzlichen Krankenversicherung werden Diese aktualisiert.

Da ein Mitarbeiter umgezogen ist, wird seine Adresse aktualisiert.

Im Unternehmen wurde eine Abteilung unter eine andere eingeordnet. Diese Hierarchie wird in der Datenbank aktualisiert.

Ein Mitarbeiter wird entlassen, weswegen seine Daten aktualisiert werden. Das Entlassungsdatum wird eingetragen und der Mitarbeiter mit einem der möglichen Austrittsgründe verknüpft.

Aus Datenschutzgründen müssen die Daten eines entlassenen Mitarbeiters entfernt werden.

Ein Nutzer hat sein Passwort mehrfach falsch eingegeben und wurde deswegen gesperrt. Der Administrator kann diesen Nutzer entsperren und erstellt dabei auch ein neues Passwort für den Nutzer.

Der Administrator entfernt einen Nutzer-Account.

Das Unternehmen beschließt, die Dienstleistung der Personalstammdatenbank nicht mehr zu verwenden. Der Administrator entfernt alle Daten des Unternehmens aus der Datenbank, entfernt alle Nutzer-Accounts, den Mandanten und zuletzt seinen eigenen Account.

4.2.2 Use Cases

Auf Grundlage des in Kapitel 4.2.1 dargestellten User Szenarios werden nun die entsprechenden Use Cases vorgestellt. Da es sich somit lediglich um Use Cases handelt, welche im Rahmen der Arbeit auch umgesetzt werden, wird kein Anspruch auf Vollständigkeit erhoben.

Nr. 1	Mandant- und Administrator-Account erstellen
Beschreibung	Es wird je ein Account für den Mandanten und den Administratoren erstellt.
Akteur	Mitarbeiter – der zukünftige Administrator
Bedingung(en)	Keine technischen Vorbedingungen
Ablauf	<ol style="list-style-type: none">1. Mitarbeiter öffnet Programm.2. Mitarbeiter gibt die Firma an, für die ein Mandant erstellt werden soll. Sowohl für den Mandanten als auch für den Administrator muss zudem je ein Passwort vergeben werden. Der Mitarbeiter hinterlegt zudem seinen Vor- und Nachnamen.

	3. Mitarbeiter betätigt Funktion, mit der Mandant- und Administrator-Account erzeugt wird, indem er die Python-Datei „main“ im Ordner „\src\main“ öffnet, dort die Funktion eingibt und anschließend ausführen lässt.
Ergebnis(se)	Mandanten- und Administrator-Account ist erzeugt.
Alternative(n)	Die Anmeldung wird abgebrochen oder schlägt fehl, womit kein Mandant und kein Administrator erzeugt wird.

Tabelle 1: "Mandant- und Administrator-Account erstellen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126

Nr. 2	Nutzer anlegen
Beschreibung	Um auf der Datenbank operieren zu können, reicht es nicht, nur einen Mandanten zu erzeugen. Es muss nun ein Nutzer erzeugt werden, der dem Mandanten zugeordnet wird. Der Mitarbeiter hat über seinen Nutzer-Account nun Zugriff auf die Datenbank.
Akteur	Administrator
Bedingung(en)	Mandant ist erzeugt.
Ablauf	<ol style="list-style-type: none"> 1. Administrator loggt sich ein. 2. Administrator gibt Personalnummer, Vorname, Nachname und Passwort des Mitarbeiters an, für den ein Nutzer-Account erstellt werden soll, an. 3. Administrator betätigt die Nutzer-anlegen-Funktion, indem er die Python-Datei „main“ im Ordner „\src\main“ öffnet, dort die Funktion eingibt und anschließend ausführen lässt.
Ergebnis(se)	Mitarbeiter ist nun als Nutzer der Datenbank angelegt.
Alternative(n)	<ol style="list-style-type: none"> 1. Der Vorgang wird abgebrochen oder schlägt fehl und es wird kein Nutzer angelegt. 2. Wenn bereits ein Nutzer desselben Mandanten mit derselben Personalnummer angelegt ist, erscheint eine Fehlermeldung, da innerhalb eines Mandanten jede Personalnummer nur einmal als Nutzer verwendet werden kann.

Tabelle 2: "Nutzer anlegen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126

Die Use Cases Nr. 3 bis Nr. 26 beschäftigen sich alle mit dem Eintragen von Daten in die Datenbank, welche benötigt werden, damit neue Mitarbeiter angelegt werden können. Für das Eintragen eines neuen Mitarbeiters ist es nicht von Bedeutung, in welcher Reihenfolge die Use Cases mit den Nummern 3 bis 26 ausgeführt werden. Es ist ausschließlich von Bedeutung, dass alle Daten eingetragen sind, die mit dem neuen Mitarbeiter verknüpft werden sollen. Da sich der Ablauf gleicht, wird der Ablauf einmal exemplarisch dargestellt und anschließend die Use Cases genannt, die dieser Struktur folgen:

	Daten eintragen – struktureller Aufbau
Beschreibung	Es werden Daten in die Datenbank eingetragen.
Akteur	Nutzer
Bedingung(en)	Mandant ist angelegt. Mitarbeiter ist als Nutzer in Datenbank registriert.
Ablauf	<ol style="list-style-type: none"> 1. Nutzer öffnet entsprechende Excel-Datei im Ordner „src/main/“. 2. Nutzer trägt die entsprechenden Werte ein. 3. Excel-Datei speichern und schließen. 4. Nutzer betätigt Funktion, mit der die Daten aus der Excel-Datei in die Datenbank übertragen werden, indem er die Python-Datei „main“ im Ordner „\src\main“ öffnet, dort die Funktion eingibt und anschließend ausführen lässt.
Ergebnis(se)	Daten sind in Datenbank angelegt.
Alternative(n)	Wenn in der Excel-Datei notwendige Eintragungen fehlen oder fehlerhafte Werte vorhanden sind, erscheint eine Fehlermeldung und die Eintragung in die Datenbank wird unterbrochen.

Tabelle 3: struktureller Aufbau von Use Cases, bei denen neue Daten eingetragen werden; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126

Folgende Use Cases folgen dieser Struktur:

Nr.	Use Case	Zusätzliche Bedingungen
3	Geschlecht anlegen	
4	Mitarbeitertyp eintragen	
5	Steuerklasse eintragen	
6	Abteilung eintragen	

7	Jobtitel eintragen	
8	Erfahrungsstufe eintragen	
9	Unternehmen eintragen	
10	Austrittsgrundkategorie eintragen	
11	Austrittsgrund eintragen	Die entsprechende Austrittsgrundkategorie ist in Datenbank hinterlegt.
12	Gesetzliche Krankenversicherungsbeiträge eintragen	
13	Gesetzliche Krankenkasse eintragen	
14	Private Krankenkasse eintragen	
15	Gemeldete Krankenkasse eintragen	
16	Arbeitnehmerbeitrag zur Pflegeversicherung eintragen	
17	Arbeitgeberbeitrag zur Pflegeversicherung eintragen	
18	Arbeitslosenversicherungsbeiträge eintragen	
19	Rentenversicherungsbeiträge eintragen	
20	Minijobbeiträge eintragen	
21	Berufsgenossenschaft eintragen	
22	Unfallversicherungsbeiträge eintragen	Sowohl die Berufsgenossenschaft, die den Beitrag erhebt, als auch die Firma, die sie bezahlen muss, sind in der Datenbank angelegt.
23	Gewerkschaft eintragen	
24	Tarif eintragen	Die Gewerkschaft, mit der der Tarifvertrag ausgehandelt wurde, ist in der Datenbank eingetragen.
25	Vergütungsbestandteil eintragen	
26	Tarif mit Vergütungsbestandteil verknüpfen	Sowohl Vergütungsbestandteil als

		auch Tarifbezeichnung ist in der Datenbank hinterlegt.
--	--	--

Tabelle 4: Use Cases, die der in Tabelle 3 dargestellten Ablaufstruktur folgen; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126

Nr. 27	Neuen Mitarbeiter anlegen
Beschreibung	Es wird ein neuer Mitarbeiter in der Datenbank angelegt.
Akteur	Nutzer
Bedingung(en)	Mandant ist angelegt. Mitarbeiter ist als Nutzer in Datenbank registriert. Je nachdem, ob der Mitarbeiter über den Arbeitgeber gesetzlich oder privat krankenversichert oder anderweitig versichert ist, muss die entsprechende Krankenkasse vorhanden sein. Ist der Mitarbeiter tariflich angestellt, muss der entsprechende Tarif vorhanden sein. Des Weiteren müssen folgende Daten in Bezug auf den Mitarbeiter in der Datenbank hinterlegt sein: Geschlecht, Mitarbeitertyp, Steuerklasse, Abteilung, Jobtitel, Erfahrungsstufe, Unternehmen. Je nachdem, welche Sozialversicherungspflichten der Mitarbeiter hat, müssen die Krankenversicherungsbeiträge, die Anzahl der Kinder, ob der Arbeitsort in Sachsen liegt und die Arbeitslosen- und Rentenversicherungsbeiträge angelegt sein. Ist der Mitarbeiter ein Minijobber, werden die Sozialversicherungsdaten nicht benötigt, dafür aber die Minijobdaten.
Ablauf	<ol style="list-style-type: none"> 1. Nutzer öffnet Excel-Datei „1 Mitarbeiter.xlsx“ im Ordner „src/main/4 neuen Mitarbeiter anlegen“. 2. Nutzer trägt die entsprechenden Werte ein. 3. Excel-Datei speichern und schließen. 4. Nutzer betätigt Funktion, mit der die Daten aus der Excel-Datei in die Datenbank übertragen werden, indem er die Python-Datei „main“ im Ordner „\src\main“ öffnet, dort die Funktion eingibt und anschließend ausführen lässt.
Ergebnis(se)	Ein neuer Mitarbeiter ist in der Datenbank angelegt und mit allen für ihn relevanten Daten verknüpft.

Alternative(n)	Wenn die Excel-Datei unvollständig ausgefüllt ist, fehlerhafte Werte eingetragen werden oder Daten, welche in vorherigen Use Cases hätten eingetragen werden müssen, fehlen, erscheint eine Fehlermeldung und die Eintragung in die Datenbank wird unterbrochen.
----------------	--

Tabelle 5: "Neuen Mitarbeiter anlegen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126

Nr. 28	Außertariflichen Vergütungsbestandteil einfügen
Beschreibung	Ein Vergütungsbestandteil, dessen Betrag und ab wann es gültig ist, wird mit einem außertariflichen Mitarbeiter verknüpft.
Akteur	Nutzer
Bedingung(en)	Mandant ist angelegt. Mitarbeiter ist als Nutzer in Datenbank registriert. Sowohl Vergütungsbestandteil als auch die Personalnummer des Mitarbeiters ist in der Datenbank hinterlegt.
Ablauf	<ol style="list-style-type: none"> 1. Nutzer öffnet Excel-Datei „2 außertariflicher Verguetungsbestandteil.xlsx“ im Ordner „src/main/4 neuen Mitarbeiter anlegen“. 2. Nutzer trägt die entsprechenden Werte ein. 3. Excel-Datei speichern und schließen. 4. Nutzer betätigt Funktion, mit der die Daten aus der Excel-Datei in die Datenbank übertragen werden, indem er die Python-Datei „main“ im Ordner „\src\main“ öffnet, dort die Funktion eingibt und anschließend ausführen lässt.
Ergebnis(se)	Ein außertariflicher Vergütungsbestandteil ist in der Datenbank mit dessen Betragshöhe und Gültigkeitsdatum mit dem Mitarbeiter verknüpft.
Alternative(n)	Wenn die Excel-Datei unvollständig ausgefüllt ist, fehlerhafte Werte eingetragen werden oder der Vergütungsbestandteil und/oder der Mitarbeiter nicht vorhanden ist, erscheint eine Fehlermeldung und die Eintragung in die Datenbank wird unterbrochen.

Tabelle 6: "Außertariflichen Vergütungsbestandteil einfügen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126

Nr. 29	Adresse aktualisieren
Beschreibung	Einem Mitarbeiter wird eine neue Adresse zugewiesen.
Akteur	Nutzer
Bedingung(en)	Mandant ist angelegt. Mitarbeiter ist als Nutzer in Datenbank registriert. Die Personalnummer des Mitarbeiters, dessen Adresse aktualisiert werden soll, ist in der Datenbank hinterlegt. Zudem muss seine bisherige in der Datenbank hinterlegt sein.
Ablauf	<ol style="list-style-type: none"> 1. Nutzer öffnet Excel-Datei „1 Update Adresse.xlsx“ im Ordner „src/main/update personenbezogene Daten“. 2. Nutzer trägt die entsprechenden Werte ein. 3. Excel-Datei speichern und schließen. 4. Nutzer betätigt Funktion, mit der die Daten aus der Excel-Datei in die Datenbank übertragen werden, indem er die Python-Datei „main“ im Ordner „\src\main“ öffnet, dort die Funktion eingibt und anschließend ausführen lässt.
Ergebnis(se)	Die neue Adresse inklusive des Datums, ab dem diese Adresse gültig ist, ist dem Mitarbeiter zugeordnet. Bei der alten Adresse steht nun drin, bis wann er dort gewohnt hat. Dies ist stets ein Tag vor dem Datum, ab dem er in der neuen Adresse gemeldet ist.
Alternative(n)	Wenn die Excel-Datei unvollständig ausgefüllt ist, fehlerhafte Werte eingetragen werden oder keine alte Adresse vorhanden ist, erscheint eine Fehlermeldung und die Eintragung in die Datenbank wird unterbrochen.

Tabelle 7: "Adresse aktualisieren"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126

Nr. 30	Mitarbeiterentlassung eintragen
Beschreibung	Das Datum, an dem ein Mitarbeiter das Unternehmen verlässt, und der Austrittsgrund werden dem Mitarbeiter zugeordnet.
Akteur	Nutzer
Bedingung(en)	Mandant ist angelegt. Mitarbeiter ist als Nutzer in Datenbank registriert. Die Personalnummer des Mitarbeiters, der das Unternehmen verlässt, ist in der Datenbank hinterlegt. Zudem muss auch der dazugehörige Austrittsgrund in der Datenbank gespeichert sein.

Ablauf	<ol style="list-style-type: none"> 1. Nutzer öffnet Excel-Datei „2 Update Mitarbeiterentlassung.xlsx“ im Ordner „src/main/update personenbezogene Daten“. 2. Nutzer trägt die entsprechenden Werte ein. 3. Excel-Datei speichern und schließen. 4. Nutzer betätigt Funktion, mit der die Daten aus der Excel-Datei in die Datenbank übertragen werden, indem er die Python-Datei „main“ im Ordner „\src\main“ öffnet, dort die Funktion eingibt und anschließend ausführen lässt.
Ergebnis(se)	Das Entlassungsdatum ist in der Datenbank in allen personenbezogenen Tabellen eingetragen. Der Mitarbeiter ist mit dem entsprechenden Austrittsgrund verknüpft.
Alternative(n)	Wenn die Excel-Datei unvollständig ausgefüllt ist, fehlerhafte Werte eingetragen werden oder der Mitarbeiter und/oder der Austrittsgrund in der Datenbank nicht vorhanden ist, erscheint eine Fehlermeldung und die Eintragung in die Datenbank wird unterbrochen.

Tabelle 8: "Mitarbeiterentlassung eintragen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126

Nr. 31	Abteilungshierarchie erstellen
Beschreibung	Zwei Abteilungen werden in Bezug zueinander gesetzt. Eine Abteilung ist dabei der anderen untergeordnet.
Akteur	Nutzer
Bedingung(en)	Mandant ist angelegt. Mitarbeiter ist als Nutzer in Datenbank registriert. Beide Abteilungen sind in der Datenbank hinterlegt.
Ablauf	<ol style="list-style-type: none"> 1. Nutzer öffnet Excel-Datei „3 Update Abteilungshierarchie.xlsx“ im Ordner „src/main/update personenbezogene Daten“. 2. Nutzer trägt die entsprechenden Werte ein. 3. Excel-Datei speichern und schließen. 4. Nutzer betätigt Funktion, mit der die Daten aus der Excel-Datei in die Datenbank übertragen werden, indem er die Python-Datei „main“ im Ordner „\src\main“ öffnet, dort die Funktion eingibt und anschließend ausführen lässt.

Ergebnis(se)	Eine Abteilung ist einer anderen Abteilung unter- bzw. übergeordnet.
Alternative(n)	Wenn die Excel-Datei unvollständig ausgefüllt ist, fehlerhafte Werte eingetragen werden oder mindestens eine von beiden Abteilungen in der Datenbank nicht vorhanden ist, erscheint eine Fehlermeldung und die Eintragung in die Datenbank wird unterbrochen.

Tabelle 9: "Abteilungshierarchie erstellen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126

Nr. 32	Krankenversicherungsbeiträge aktualisieren
Beschreibung	Die Krankenversicherungsbeiträge oder Teile davon haben sich verändert und werden in der Datenbank eingetragen.
Akteur	Nutzer
Bedingung(en)	Mandant ist angelegt. Mitarbeiter ist als Nutzer in Datenbank registriert. Es existiert bereits mindestens einen Datensatz mit den Krankenversicherungsbeiträgen.
Ablauf	<ol style="list-style-type: none"> 1. Nutzer öffnet Excel-Datei „1 Krankenversicherungsbeitraege.xlsx“ im Ordner „src/main/update Sozialversicherungsdaten“. 2. Nutzer trägt die entsprechenden Werte ein. 3. Excel-Datei speichern und schließen. 4. Nutzer betätigt Funktion, mit der die Daten aus der Excel-Datei in die Datenbank übertragen werden, indem er die Python-Datei „main“ im Ordner „\src\main“ öffnet, dort die Funktion eingibt und anschließend ausführen lässt.
Ergebnis(se)	Die neuen Krankenversicherungsbeiträge inklusive des Datums, ab dem sie gültig sind, sind eingetragen. Bei den alten Krankenversicherungsbeiträgen steht nun drin, bis zu welchem Datum sie gültig waren. Dies ist ein Tag vor dem Datum, ab dem die neuen Beiträge gültig sind.
Alternative(n)	Wenn die Excel-Datei unvollständig ausgefüllt ist, fehlerhafte Werte eingetragen werden oder keine veralteten Krankenversicherungsbeiträge

	vorhanden sind, erscheint eine Fehlermeldung und die Eintragung in die Datenbank wird unterbrochen.
--	---

Tabelle 10: "Krankenversicherungsbeiträge aktualisieren"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126

Nr. 33	SQL-Abfrage ausführen
Beschreibung	Ein Nutzer möchte Daten aus der Datenbank abfragen und gibt deswegen über die Funktion „abfrage_ausfuehren()“ einen SQL-Befehl ein.
Akteur	Nutzer
Bedingung(en)	Mandant ist angelegt. Mitarbeiter ist als Nutzer in Datenbank registriert.
Ablauf	Nutzer öffnet die Python-Datei „main“ im Ordner „\src\main“, gibt dort die Funktion ein lässt sie dann ausführen.
Ergebnis(se)	Daten werden ausgegeben.
Alternative(n)	Wenn der SQL-Befehl fehlerhaft ist, wird das Programm abgebrochen.

Tabelle 11: "SQL-Abfrage ausführen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126

Nr. 34	Mitarbeiterdaten entfernen
Beschreibung	Alle personenbezogenen Daten eines Mitarbeiters werden aus der Datenbank gelöscht.
Akteur	Nutzer
Bedingung(en)	Mandant ist angelegt. Mitarbeiter ist als Nutzer in Datenbank registriert. Die Personalnummer des Mitarbeiters, dessen Daten gelöscht werden sollen, muss in der Datenbank hinterlegt sein.
Ablauf	<ol style="list-style-type: none"> 1. Nutzer öffnet Excel-Datei „Personalnummer.xlsx“ im Ordner „src/main/delete personenbezogene Daten“. 2. Nutzer trägt die entsprechenden Werte ein. 3. Excel-Datei speichern und schließen. 4. Nutzer betätigt Funktion, mit der die Daten aus der Excel-Datei in die Datenbank übertragen werden, indem er die Python-Datei „main“ im Ordner „\src\main“ öffnet, dort die Funktion eingibt und anschließend ausführen lässt.

Ergebnis(se)	Sämtliche Daten aus allen Tabellen, die einen direkten Bezug zum Mitarbeiter mit der entsprechenden Personalnummer haben, sind unwiderbringlich gelöscht.
Alternative(n)	Wenn die Excel-Datei unvollständig ausgefüllt ist oder die eingegebene Personalnummer nicht existiert, erscheint eine Fehlermeldung und die Eintragung in die Datenbank wird unterbrochen.

Tabelle 12: "Mitarbeiterdaten entfernen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126

Nr. 35	Entsperrung eines Nutzer-Accounts
Beschreibung	Der Nutzer hat sich durch dreimalige Eingabe eines Passworts hintereinander gesperrt. Um die Sperre aufzuheben, kontaktiert er den Administrator, der die Entsperrung vornimmt und ein neues Übergangspasswort für den Nutzer wählt
Akteur	Administrator
Bedingung(en)	Nutzer-Account ist gesperrt.
Ablauf	<ol style="list-style-type: none"> 1. Administrator loggt sich ein 2. Administrator ermittelt mithilfe der Personalnummer des gesperrten Nutzers dessen Nutzer-Account. 3. Administrator wählt ein neues Übergangspasswort, welches er zweimal eingibt. 4. Administration betätigt die Entsperrungsfunktion. 5. Administrator übermittelt dem nun entsperrten Nutzer sein neues (Übergangs-)Passwort
Ergebnis(se)	Der Nutzer ist entsperrt und hat ein neues (Übergangs-)Passwort.
Alternative(n)	<ol style="list-style-type: none"> 1. Der Administrator benutzt eine falsche Personalnummer, wodurch die Entsperrung des betroffenen Nutzer-Accounts fehlschlägt. 2. Die zweite Passworteingabe unterscheidet sich von der ersten Eingabe, wodurch die Entsperrung scheitert.

Tabelle 13: "Entsperrung eines Nutzer-Accounts"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126

Nr. 36	Neuvergabe des Nutzerpassworts nach Entsperrung
Beschreibung	Der Nutzer hat sich durch dreimalige Eingabe eines Passworts hintereinander gesperrt und wurde bereits vom Administrator wieder entsperrt und hat das neue vom Administrator gewählte Passwort erhalten. Nun muss sich der Nutzer damit einloggen und erst ein neues Passwort erstellen, damit er mit der Datenbank interagieren kann. Dies ist notwendig, damit der Nutzer-Account mit einem Passwort ausgestattet ist, dass der Administrator nicht kennt.
Akteur	Nutzer
Bedingung(en)	Nutzer-Account ist entsperrt.
Ablauf	<ol style="list-style-type: none"> 1. Nutzer loggt sich mit dem vom Administrator gewählten Passwort ein. 2. Nutzer gibt sein neues Passwort zweimal ein. 3. Nutzer betätigt die Passwort-ändern-Funktion.
Ergebnis(se)	Der Nutzer hat nun ein neues Passwort
Alternative(n)	Die zweite Passwort-Eingabe unterscheidet sich von der ersten Eingabe, womit die Passwortänderung fehlschlägt.

Tabelle 14: "Neuvergabe des Nutzerpassworts nach Entsperrung"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022):
Seite 126

Nr. 37	Nutzer-Account entfernen
Beschreibung	Ein Nutzer-Account wird nicht mehr benötigt und soll aus dem System entfernt werden.
Akteur	Administrator
Bedingung(en)	Zu entfernender Nutzer-Account ist vorhanden.
Ablauf	<ol style="list-style-type: none"> 1. Administrator loggt sich in seinem Account ein 2. Administrator entfernt mithilfe der Nutzer-Entfernen-Funktion und der Personalnummer des betreffenden Nutzers dessen Account.
Ergebnis(se)	Der Nutzer-Account ist entfernt.

Alternative(n)	<ol style="list-style-type: none"> 1. Wenn eine nichtexistierende Personalnummer eingegeben wird, wird der Nutzeraccount nicht gelöscht. Stattdessen erscheint eine Fehlermeldung. 2. Wird eine falsche, aber existierende, Personalnummer verwendet, wird der falsche Account entfernt.
----------------	--

Tabelle 15: "Nutzer-Account entfernen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126

Nr. 38	Nutzung der Personalstammdatenbank beenden
Beschreibung	Es werden sämtliche Daten eines Mandanten entfernt. Ebenso werden der Mandant selbst und deren Nutzer sowie der Administrator selbst aus dem System gelöscht.
Akteur	Administrator
Bedingung(en)	Mandant- und Administrator-Account ist angelegt.
Ablauf	Der Administrator gibt in der Funktion "entferne_mandant_nutzer_und_admin" den Mandantennamen, dessen Passwort zweimal, seine Personalnummer und sein Passwort ebenfalls zweimal ein. Es werden zuerst alle Daten des Mandanten aus allen Tabellen der Datenbank entfernt und anschließend das Mandant- und Administrator-Objekt gelöscht.
Ergebnis(se)	Alle Daten und Accounts sind entfernt.
Alternative(n)	Das Unternehmen entscheidet sich um und nutzt das System weiterhin.

Tabelle 16: "Alle Daten des Mandanten entfernen"; eigene Darstellung in Anlehnung an Crowder, Hoff (2022): Seite 126

5 Beschreibung des Prototypen anhand ausgewählter Aspekte

In diesem Kapitel werden beispielhafte Aspekte der im Rahmen dieser Arbeit getätigten Implementierungen vorgestellt. Es wird somit nicht das komplette System, allerdings die zentralen Eigenschaften des Systems erklärt.

5.1 Registrierung, Erstellung und Entfernung von Mandant, Nutzer und Administrator

Um die Dienste der Personalstammdatenbank nutzen zu können, ist im ersten Schritt die Registrierung eines neuen Mandanten notwendig. Dies geschieht mithilfe der Funktion „registriere_mandant_und_admin“ der Klasse „Login“, der man folgende Werte übergeben muss: den Mandantennamen, ein Mandantenpasswort, eine Wiederholung des Mandantenpassworts (um sicherzustellen, dass die Ersteingabe auch so erfolgte wie geplant), die Personalnummer des Administrators, dessen Vor- und Nachname, dessen Passwort und auch hier eine Wiederholung. Im Zuge der Registrierung wird stets der Mandant und ein Administrator erzeugt, wie folgender Code ausschnittsweise zeigt:

```
from src.main.Mandant import Mandant
from src.main.Administrator import Administrator

class Login:

    def __init__(self, schema='public'):

        if schema != 'public' and schema != 'temp_test_schema':
            raise(ValueError("Diese Bezeichnung fuer ein Schema ist nicht erlaubt!"))

        self.schema = schema
        self.liste_mandanten = []
        self.liste_admins = []

    def registriere_mandant_und_admin(self, mandantennamen, mandantenpasswort, mandantenpasswort_wiederholen,
                                     admin_personalnummer, admin_vorname, admin_nachname, adminpasswort,
                                     adminpasswort_wiederholen):

        neuer_mandant = Mandant(mandantennamen, mandantenpasswort, mandantenpasswort_wiederholen, self.schema)
        neuer_admin = Administrator(neuer_mandant, admin_personalnummer, admin_vorname, admin_nachname,
                                   adminpasswort, adminpasswort_wiederholen, self.schema)

        self.liste_mandanten.append(neuer_mandant)
        self.liste_admins.append(neuer_admin)
```

Abbildung 8: Code, mit der ein Mandant und ein Administrator erzeugt wird; eigener Code dargestellt mit carbon.now.sh

Die Erzeugung von des Mandanten und des Administrators geschieht über die Erstellung eines jeweiligen Objekts, welche jeweils in den Listen „liste_mandanten“ und „liste_admins“, die in der „Login“-Klasse hinterlegt sind und wo sie gespeichert werden. Ist die Erstellung der Objekte erfolgreich, so werden die Daten auch in der Datenbank gespeichert. Bei der Speicherung der Datensätze in der Datenbank wird folgerichtig jeweils eine ID für

Administrator und Mandant zurückgegeben, welche in der Variable „mandant_id“ des soeben erzeugten Mandant-Objekts beziehungsweise in der Variable „admin_id“ des soeben erzeugten Administrator-Objekts gespeichert.

Sollen nun für den Mandanten Nutzer erzeugt werden, welche mit der Datenbank interagieren, ist es zuerst notwendig, dass sich der Administrator über die Funktion „login_admin“ einloggt. Die Administrator-Klasse verfügt über eine Funktion „nutzer_anlegen“, die ein Objekt aus der „Nutzer“-Klasse erzeugt:

```
def nutzer_anlegen(self, personalnummer, vorname, nachname, password, password_wiederholen):  
    nutzer = Nutzer(self.mandant.get_mandant_id(), personalnummer, vorname, nachname, password,  
                    password_wiederholen, self.schema)  
    self.mandant.get_nutzerliste().append(nutzer)
```

Abbildung 9: Funktion zur Erstellung eines neuen Nutzers; eigener Code dargestellt mit carbon.now.sh

Das „Nutzer“-Objekt wird einerseits dem Mandanten zugeordnet, in dem das Objekt einer Liste hinzugefügt wird, die sich in der „Mandant“-Klasse beziehungsweise dem jeweiligen Objekt befindet und andererseits besitzt die „Nutzer“-Klasse beziehungsweise deren Objekte die Variable „Mandant_ID“, in der die ID des Mandanten gespeichert wird, zu dem es gehört.

Der Nutzer kann sich nun ebenfalls einloggen. Hierfür bietet die „Login“-Klasse die Funktion „login_nutzer“ an. Hierbei muss er den Namen des Mandanten, dessen Passwort und sein eigenes Passwort richtig eingeben. Scheitert dies, so wird eine Fehlermeldung ausgegeben. Gibt der Nutzer das Passwort dreimal hintereinander falsch ein, so wird der Zugang des Nutzers gesperrt.

Eine Entsperrung des Nutzers ist nur durch den Administrator möglich, der hierfür die Funktion „nutzer_ent Sperren“ implementiert. Er wählt für den Nutzer ein neues Passwort aus und übergibt dies mit der Mandant_ID und der Personalnummer des Nutzers der PL/pgSQL-Funktion „nutzer_ent Sperren“, welche das alte Nutzerpasswort mit dem neuen Passwort überschreibt und die Sperrung zurücksetzt.

Wenn sich der Nutzer zum ersten Mal nach Erstellung seines Accounts oder nach einer Entsperrung einloggt, muss er sein Passwort mithilfe der „Nutzer“-Funktion „password_aendern“ ändern. Dies ist eine Sicherheitsmaßnahme, da nach der Erstellung des Accounts oder nach der Entsperrung, welche beide vom Administrator vorgenommen werden,

der Administrator das Passwort des jeweiligen Nutzers kennt. Durch die erzwungene Passwortänderung wird sichergestellt, dass der Administrator das Passwort nicht mehr kennt.

```
def password_aendern(self, neues_passwort, neues_passwort_wiederholen):
    if neues_passwort != neues_passwort_wiederholen:
        raise(ValueError("Zweite Passwortheingabe ist anders als erste Passwortheingabe!"))

    conn = datenbankverbindung_aufbauen()

    nutzer_insert_query = f"set search_path to {self.schema};" \
        f"CALL nutzerpassword_aendern('{self.mandant_id}', '{self.personalnummer}', " \
        f"'{neues_passwort}');"

    cur = conn.cursor()
    cur.execute(nutzer_insert_query)

    # Commit der Aenderungen
    conn.commit()

    # Cursor und Konnektor zu Datenbank schliessen
    cur.close()
    conn.close()

    self.neues_passwort_geaendert = True
```

Abbildung 10: Funktion, mit der Nutzer sein Passwort ändern kann; eigener Code dargestellt mit carbon.now.sh

Sollte ein Nutzer-Account nicht mehr benötigt werden, so nutzt der Administrator die „nutzer_entfernen“-Funktion, welche seine Klasse beziehungsweise dessen Objekte implementiert. Hierfür muss der Administrator die Personalnummer des zu entfernenden Nutzers angeben. Es wird nun die gleichnamige PL/pgSQL-Funktion „nutzer_entfernen“ aufgerufen, welche den entsprechenden Datensatz des Nutzers aus der Tabelle „Nutzer“ entfernt. Anschließend wird das entsprechende Nutzer-Objekt aus der Liste „lister_nutzer“ des Mandant-Objekts entfernt. Der Nutzer existiert nicht mehr und kann sich folglich auch nicht mehr einloggen.

```
def nutzer_entfernen(self, personalnummer):
    nutzer_entfernt = False

    for i in range(len(self.get_mandant().get_nutzerliste())):
        if self.get_mandant().get_nutzerliste()[i].get_personalnummer() == personalnummer:
            conn = datenbankverbindung_aufbauen()

            # Nutzer aus Datenbank entfernen
            nutzer_delete_query = f"set search_path to {self.schema};" \
                f"CALL nutzer_entfernen({self.get_mandant().get_mandant_id()}, " \
                f"'{personalnummer}');"

            cur = conn.cursor()
            cur.execute(nutzer_delete_query)

            # Commit der Aenderungen
            conn.commit()

            # Cursor und Konnektor zu Datenbank schliessen
            cur.close()
            conn.close()

            # Nutzer aus Liste 'lister_nutzer' des Mandant-Objekt entfernen
            self.get_mandant().get_nutzerliste().remove(self.get_mandant().get_nutzerliste()[i])

            nutzer_entfernt = True
            print(f"Nutzer {personalnummer} wurde entfernt!")

    if not nutzer_entfernt:
        raise (ValueError(f"Nutzer {personalnummer} existiert nicht!"))
```

Abbildung 11: Funktion zur Entfernung eines Nutzer-Accounts; eigener Code dargestellt mit carbon.now.sh

Sollte entschieden werden, die Dienste der Personalstammdatenbank nicht mehr zu nutzen, kann der Mandant, all dessen Daten aus der Datenbank, all dessen Nutzer und der Administrator mithilfe der Funktion „entferne_mandant_nutzer_und_admin“ der Login-Klasse entfernt werden. Im ersten Schritt werden alle Daten aus der Personalstammdatenbank gelöscht, was mithilfe des Aufrufs der PL/pgSQL-Funktion „delete_mandantendaten“ ausgeführt wird. Im zweiten Schritt wird das Administrator-Objekt, welches zum zu entfernenden Mandant-Objekt gehört, aus der Liste „liste_admins“ der „Login“-Klasse entfernt. Anschließend wird das entsprechende Mandant-Objekt aus der Liste „liste_mandanten“ entfernt. Da die Nutzer-Objekte direkt im soeben entfernten Mandant-Objekt gespeichert sind, sind somit auch alle Nutzer entfernt.

```
def entferne_mandant_nutzer_und_admin(self, mandantenname, mandantenpassword, mandantenpassword_wiederholen,
                                     admin_personalnummer, adminpassword, adminpassword_wiederholen):

    if mandantenpassword != mandantenpassword_wiederholen or adminpassword != adminpassword_wiederholen:
        raise (ValueError("Passwort falsch eingegeben."))

    # Login des Admins
    admin = self.login_admin(mandantenname, mandantenpassword, admin_personalnummer, adminpassword)

    # Entfernung aller der Daten des Mandanten aus der Datenbank
    export_daten = [admin.get_mandant().get_mandant_id()]

    conn = datenbankbverbindung_aufbauen()
    cur = conn.cursor()

    cur.execute(f"set search_path to {self.schema}; call delete_mandantendaten(%s)", export_daten)

    # Commit der Aenderungen
    conn.commit()

    # Cursor und Konnektor zu Datenbank schliessen
    cur.close()
    conn.close()

    # Entfernung des Administrators aus Login_Liste "liste_admins"
    for i in range(len(self.liste_admins)):
        if self.liste_admins[i].get_personalnummer() == admin_personalnummer and \
            self.liste_admins[i].get_mandant().get_mandantenname() == mandantenname:
            del self.liste_admins[i]

    # Entfernung des Mandanten aus Login-Liste "liste_mandanten"
    for i in range(len(self.liste_mandanten)):
        if self.liste_mandanten[i].get_mandantenname() == mandantenname:
            del self.liste_mandanten[i]
```

Abbildung 12: Funktion zur Entfernung aller Daten eines Mandanten aus dem System; eigener Code dargestellt mit carbon.now.sh

5.2 Struktur des Datenimports

Die Verarbeitung der Daten, welche in die Datenbank eingespeist werden sollen, verläuft in drei Schritten, die wie in folgender Abbildung dargestellt, ablaufen:

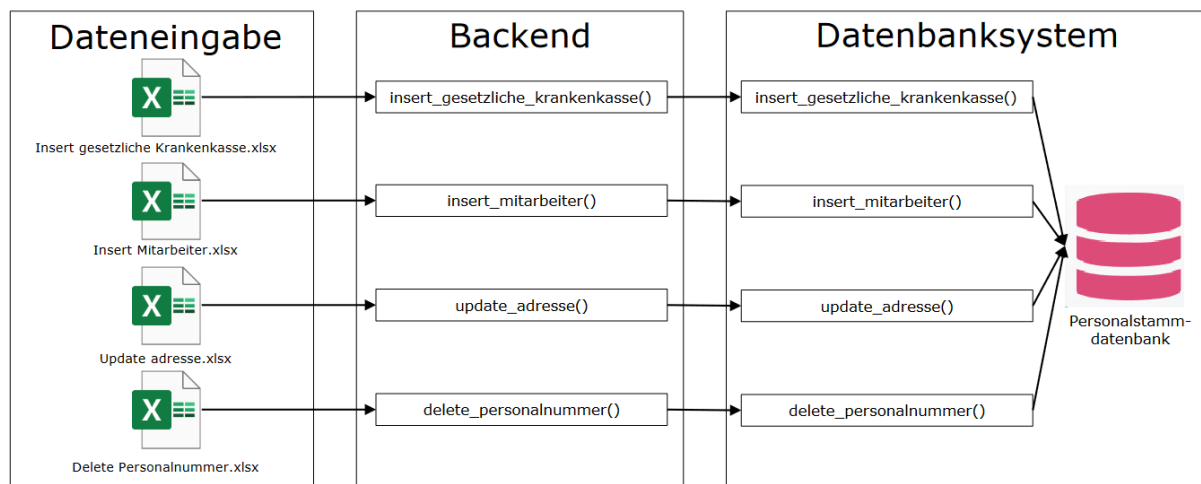


Abbildung 13: Darstellung der Datenpipeline, eigene Darstellung

Die Interaktion mit der Datenbank erfolgt zum größten Teil ausschließlich über Nutzer-Accounts. Administratoren interagieren mit der Datenbank nur, um neue Nutzer anzulegen, sie zu entsperren, zu entfernen oder wenn die Nutzung der Personalstammdatenbank beendet werden soll (siehe Kapitel 4.2.2 Use Cases).

Der erste Schritt ist die Dateneingabe, welche über Excel-Dateien erfolgt. Je nachdem, was für Daten in die Datenbank eingetragen werden sollen, muss hierfür die jeweilige Excel-Datei geöffnet werden und dort die entsprechenden Daten eingetragen werden.

Der zweite Schritt ist die Datenverarbeitung, die teilweise über das Backend, welches in der Programmiersprache Python geschrieben ist, ablaufen. Für jede Excel-Datei gibt es eine entsprechende Funktion, welche ausschließlich aus exakt einer Excel-Tabelle Daten importiert und wo geprüft wird, ob die Daten den richtigen Datentyp haben, ob Pflichtfelder ausgefüllt sind oder ob die Daten den Begrenzungen entsprechen. Beispiele für Letzteres sind unter anderem, ob eine Zeichenkette nicht mehr Zeichen enthält, als die Datenbank verarbeiten kann oder ob bei Zahlenwerten ein Maximalwert nicht überschritten werden. In einigen Fällen werden Daten in einen anderen Datentyp umgewandelt.

Werden die Überprüfungen im Backend vollständig bestanden, werden im dritten Schritt die Dateien an das Datenbanksystem übertragen, wo sie jedoch nicht direkt in die Datenbank geschrieben werden, sondern von PL/pgSQL-Skripten weiterverarbeitet werden. Jedes PL/pgSQL-Skript arbeitet exakt mit einer Python-Funktion zusammen und umgekehrt. Hierbei übergibt stets die Python-Funktion Daten an das PL/pgSQL-Skript. Um zu verdeutlichen, dass jeweils eine Python-Funktion und ein PL/pgSQL-Skript zusammenarbeiten, tragen diese stets

denselben Namen. So übergibt die Python-Funktion „insert_gesetzliche_krankenkasse()“ Daten an das PL/pgSQL-Skript „insert_gesetzliche_krankenkasse()“. In den Skripten wird beispielsweise geprüft, ob diese Daten bereits vorhanden sind und wenn ja, gegebenenfalls nicht erneut eingetragen, weil sie zum Beispiel gegen unique-constraints verstoßen. Verstoßen die übergebenen Daten gegen keine Regeln, werden die Daten mit SQL-Befehlen, welche direkt in den PL/pgSQL-Skripten enthalten sind, in die Datenbank eingetragen.

5.3 Mandantenfähige Datenbank

Für die Implementation der Mandantenfähigkeit wird für diese Arbeit das Modell gewählt, bei der die Daten aller Mandanten auf einer gemeinsamen Datenbank in einem gemeinsamen Schema gespeichert werden. Im ersten Schritt der Implementation erfordert dies für alle Tabellen der Personalstammdatenbank die Einrichtung eines Attributs, welche den Mandanten eindeutig identifiziert (siehe Abschnitt „3.3 Mandantenfähigkeit“). In dieser Arbeit heißt dieses Attribut stets „Mandant_ID“.

Austrittsgründe	
PK	<u>Austrittsgrund_ID</u>
FK	Mandant_ID
	Austrittsgrund
FK	Kategorie_Austrittsgruende_ID

Abbildung 14: beispielhafte Tabelle mit Attribut "Mandant_ID"; eigene Darstellung mit draw.io

Im zweiten Schritt muss sichergestellt werden, dass sich jeder Nutzer mit der Mandant-ID seines Mandanten ausweisen muss und in der Folge auch nur diese Daten einsehen beziehungsweise manipulieren und im Bedarfsfall entfernen kann. Hierfür wird RLS eingesetzt, welche das gewährleistet.¹²⁶ RLS muss für jede Tabelle einzeln implementiert werden, was in dieser Arbeit folgendermaßen durchgeführt wird:

¹²⁶ PostgreSQL (o.D. - c): Row Security Policies

```

create table Austrittsgruende (
  Austrittsgrund_ID serial primary key,
  Mandant_ID integer not null,
  Austrittsgrund varchar(32) not null,
  Kategorie_Austrittsgruende_ID integer not null,
  unique(Mandant_ID, Austrittsgrund),
  constraint fk_Austrittsgruende_mandanten
    foreign key (Mandant_ID)
      references Mandanten(Mandant_ID),
  constraint fk_austrittsgruende_austrittsgrundkategorien
    foreign key (Kategorie_Austrittsgruende_ID)
      references Kategorien_Austrittsgruende(Kategorie_Austrittsgruende_ID)
);
alter table Austrittsgruende enable row level security;
create policy FilterMandant_austrittsgruende
  on Austrittsgruende
  using (Mandant_ID = current_setting('app.current_tenant')::int);

```

Abbildung 15: Implementation von Row Level Security für beispielhafte Tabelle; eigener Code dargestellt mit carbon.now.sh

Zuerst wird die Tabelle implementiert. Anschließend wird mithilfe des Befehls „alter table <<Tabellenname>> enable row level security;“ die Nutzung von RLS ermöglicht. Danach wird für die jeweilige Tabelle eine Regel erstellt, welche besagt, dass nur auf diejenigen Datensätze zugegriffen werden kann, die im Attribut „Mandant_ID“ denselben Wert haben, wie sie auch der Nutzer des jeweiligen Mandanten hat.

Damit diese Zugriffsbeschränkung funktioniert, muss im dritten Schritt für das Datenbanksystem eine eigene Rolle implementiert werden, welche RLS nicht umgehen kann. Wenn in PostgreSQL eine neue Datenbank erstellt wird, wird stets auch eine Administrator-Rolle erzeugt, welcher in dieser Arbeit „postgres“ lautet. Diese Administrator-Rolle umgeht aber RLS. Damit RLS funktioniert, wird eine weitere Rolle implementiert, welche in dieser Arbeit „tenant_user“ heißt:

```

create role tenant_user;
grant usage on schema public to tenant_user;
alter default privileges in schema public grant select, insert, update, delete on tables to tenant_user;
alter default privileges in schema public grant usage on sequences to tenant_user;

```

Abbildung 16: Implementation der Rolle "tenant_user"; eigener Code dargestellt mit carbon.now.sh

In der ersten Zeile wird die Rolle „tenant_user“ erstellt. In Zeile 2 wird der Rolle „tenant_user“ der Zugriff auf das schema „public“ eingeräumt, in der die Personalstammdatenbank implementiert ist. In der dritten Zeile erhält „tenant_user“ das Recht, bestimmte SQL-Befehle auszuführen. Im Rahmen dieser Arbeit darf die Rolle „tenant_user“ die SELECT-, INSERT-, UPDATE- und DELETE-Befehle ausführen. In Zeile 4 wird sichergestellt, dass Sequenzen eingesetzt werden können. Im Rahmen dieser Arbeit wird die

Sequenz „serial“ für den Befehl „serial primary key“ bei der Implementation des Primärschlüssels in den Tabellen verwendet. Damit ist es möglich, dass bei jedem neu angelegten Datensatz unabhängig von Mandantenzugehörigkeit der Primärschlüssel um den Wert 1 höher ist als der vorherige Datensatz. Würde „tenant_user“ diese Sequenzen-Berechtigung nicht zugeordnet werden, käme es zu Fehlermeldungen.

Die Rolle „tenant_user“ umgeht RLS nicht. Deswegen muss im vierten Schritt mithilfe des Python- und PL/pgSQL-Codes sichergestellt werden, dass, sobald auf Daten der Personalstammdatenbank zugegriffen wird, nicht über die Rolle „postgres“ sondern über die Rolle „tenant_user“ der Datenzugriff erfolgt. Hierfür wird folgende Lösung in den PL/pgSQL-Funktionen implementiert:

```
create or replace procedure insert_austrittsgrund(
    p_mandant_id integer,
    p_austrittsgrund varchar(32),
    p_austrittsgrundkategorie varchar(16)
) as
$$
declare
    v_kategorie_austrittsgruende_id integer;
begin
    set session role tenant_user;
    execute 'SET app.current_tenant=' || p_mandant_id;

    execute 'SELECT kategorie_austrittsgruende_id FROM kategorien_austrittsgruende WHERE austrittsgrundkategorie = $1'
        into v_kategorie_austrittsgruende_id using lower(p_austrittsgrundkategorie);

    insert into
        Austrittsgruende(Mandant_ID, Austrittsgrund, Kategorie_Austrittsgruende_ID)
    values
        (p_mandant_id, p_austrittsgrund, v_kategorie_austrittsgruende_id);

    exception
    when unique_violation then
        raise exception 'Austrittsgrund ''%'' bereits vorhanden!', p_austrittsgrund;

end;
$$
language plpgsql;
```

Abbildung 17: Sicherstellung des Datenbankzugriffs über „tenant_user“ bei einer beispielhaften PL/pgSQL-Funktion; eigener Code dargestellt mit carbon.now.sh

Zu Beginn einer PL/pgSQL-Funktion wird die Rolle von der Admin-Rolle „postgres“ für die Dauer der Session auf „tenant_user“ umgestellt. Eine Session startet mit der Datenbankverbindung und endet mit dem Abbruch derselben, wenn die PL/pgSQL-Funktion vollständig ausgeführt wird oder durch einen Fehler der Abbruch der Funktionsausführung erfolgt. Nach Beendigung der Session wird die ausführende Rolle automatisch wieder auf die Admin-Rolle „postgres“ zurückgestellt. Unmittelbar nachdem die auszuführende Rolle auf „tenant_user“ umgestellt wurde, wird die gültige Mandant-ID übergeben, dessen Daten eingesehen beziehungsweise manipuliert werden dürfen. Der Befehl korrespondiert hierbei mit der RLS-Policy. Da dem PL/pgSQL-Funktionen stets Python-Funktionen vorgelagert ist,

muss der Python-Code so beschaffen sein, dass einerseits stets die Mandant_ID übergeben wird und andererseits die Mandant_ID durch den Nutzer und/oder dem Administrator nicht manuell geändert werden kann. Dies erfolgt durch folgende Weise:

```
def _import_excel_daten(self, excel_datei_pfad):

    df_daten = pd.read_excel(f"{excel_datei_pfad}", index_col='Daten', na_filter=False)
    daten = list(df_daten.iloc[:, 0])

    return daten

def insert_austrittsgrund(self, neuanlage_austrittsgrund):

    if not self.neues_passwort_geaendert:
        raise(ValueError("Ihr Administrator hat ein neues Passwort vergeben. Bitte wechseln Sie Ihr Passwort!"))

    # Import der Daten aus der Excel-Datei in das Pandas-Dataframe und Uebertragung in Liste "daten"
    daten = self._import_excel_daten(neuanlage_austrittsgrund)

    # Daten aus importierter Excel-Tabelle '1 insert personenbezogene Daten/9 Austrittsgrund.xlsx' pruefen
    austrittsgrund = self._existenz_str_daten_feststellen(daten[0], 'Austrittsgrund', 16, True)
    austrittsgrundkategorie = self._existenz_str_daten_feststellen(daten[1], 'Austrittsgrundkategorie', 16, True)

    export_daten = [self.mandant_id, austrittsgrund, austrittsgrundkategorie]
    self._export_zu_db('insert_austrittsgrund(%s,%s,%s)', export_daten)

def _export_zu_db(self, stored_procedure, export_daten):

    conn = datenbankverbindung_aufbauen()
    cur = conn.cursor()

    cur.execute(f"set search_path to {self.schema}; call {stored_procedure}", export_daten)

    # Commit der Aenderungen
    conn.commit()

    # Cursor und Konnektor zu Datenbank schliessen
    cur.close()
    conn.close()
```

Abbildung 18: beispielhafter Python-Code, mit der die Übergabe der Mandant-ID sichergestellt wird; eigener Code dargestellt mit carbon.now.sh

Mithilfe der Funktion „_import_excel_daten()“ werden Daten aus der Excel-Datei importiert. Anschließend werden die zuvor in der Excel-Datei eingegebenen Daten auf Richtigkeit geprüft und gemeinsam mit der Mandant-ID, welche in jedem Nutzerobjekt als Variable gespeichert ist, in eine Liste abgelegt, welche „export_daten“ heißt. Dann wird die Liste an die Funktion „_export_zu_db()“ übergeben, die dann die Daten an die korrespondierende PL/pgSQL-Funktion übergibt (in dem Fall „insert_austrittsgrund()“). So ist sichergestellt, dass stets die Mandant-ID übergeben wird und dann im PL/pgSQL-Code nur die entsprechenden Daten einsehbar sind.

5.4 Datenmodellierung

In diesem Kapitel werden zentrale Aspekte des Datenmodells erläutert und vorgestellt. Hierbei werden Argumente dargelegt, wieso das Datenmodell in der bestehenden Form

erstellt wurde. Es werden jeweils zum betreffenden Thema ausschnittsweise das Datenmodell gezeigt. Das vollständige Datenmodell ist im Kapitel „Anhang A – Datenmodell“ einsehbar. Attribute, die keinem not-null-constraint unterliegen, also wo es möglich ist, dass für dieses Attribut in einem Datensatz kein Wert drinsteht, sind mit einem „*“-Zeichen versehen.

5.4.1 Historisierung der Daten

Ziel dieser Arbeit ist es auch, die Daten zu historisieren. Historisieren bedeutet in diesem Kontext, dass veraltete Daten nicht überschrieben werden, sondern dass stattdessen angegeben wird, für welchen Zeitraum sie gültig sind beziehungsweise waren. Somit ist es möglich, vergangenheitsbezogene Analysen durchzuführen, auch wenn die Daten nicht mehr aktuell sind. Im Rahmen dieser Arbeit wird die Historisierung meist über die Verbindungstabellen in n:m-Beziehungen realisiert:

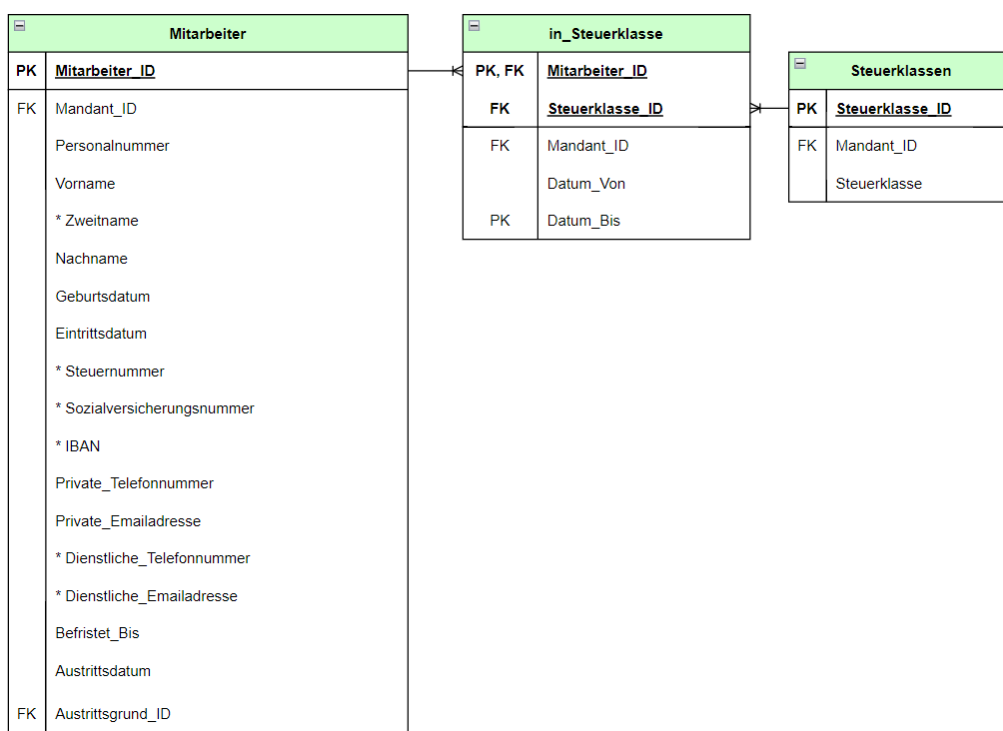


Abbildung 19: Beispiel für Historisierung; eigene Darstellung mit draw.io

Hier wird die Historisierung am Beispiel der Steuerklasse des Mitarbeiters dargestellt. In der Verbindungstabelle „in_Steuerklasse“ werden die beiden Attribute „Datum_Von“ und „Datum_Bis“ implementiert, welche den Gültigkeitszeitraum angeben, in der ein Mitarbeiter in einer bestimmten Steuerklasse gemeldet ist. Der Primärschlüssel dieser Tabelle setzt sich aus „Mitarbeiter_ID“ und „Datum_Bis“ zusammen. Das System ist so aufgebaut, dass bei

aktuellen Einträgen in der Spalte „Datum_Bis“ der Wert „9999-12-31“ steht. Dadurch, dass die Attribute „Mitarbeiter_ID“ und „Datum_Bis“ gemeinsam den Primärschlüssel bilden, ist sichergestellt, dass ein Mitarbeiter nicht gleichzeitig in zwei Steuerklassen eingetragen ist. Wäre „Datum_Bis“ nicht Teil des Primärschlüssels, wäre folgender Eintrag möglich:

Mitarbeiter (gekürzt)					in Steuerklasse					Steuerklassen		
Mitarbeiter_ID	Mandant_ID	Personalnummer	Vorname	Nachname	Mitarbeiter_ID	Steuerklasse_ID	Mandant_ID	Datum_Von	Datum_Bis	Steuerklasse_ID	Mandant_ID	Steuerklasse
1	1	M100001	Max	Mustermann	1	1	1	2020-01-01	9999-12-31	1	1	1
					1	2	1	2022-01-01	9999-12-31	2	1	4

Abbildung 20: möglicher Eintrag, wenn "Datum_Bis" kein Primärschlüssel-Teil ist; eigene Darstellung mit Excel

Der in der obigen Abbildung dargestellte Sachverhalt lautet, dass ein Mitarbeiter, seit dem 01.01.2020 in Steuerklasse 1 ist. Da er Ende 2021 heiratet, wechselt er zum Jahresbeginn 2022 in die Steuerklasse 4. Durch einen Eingabefehler wurde aber „Datum_Bis“ für Steuerklasse 1 nicht auf „2021-12-31“ gesetzt, weswegen er laut Datenbank seit Januar 2022 gleichzeitig in Steuerklasse 1 und 4 gemeldet ist, was rechtlich nicht möglich ist. Ist „Datum_Bis“ Teil des Primärschlüssels, käme bei dem Versuch, den Eintrag wie in Abbildung 16 zu tätigen, eine Fehlermeldung, da der unique-constraint verletzt wird. Somit kann durch dieses System fehlerhafte Einträge in Bezug auf Gültigkeitszeiträume vermieden werden. Wichtig für diese Konstruktion ist auch, dass „Steuerklasse_ID“ nicht Teil des Primärschlüssels ist. Wäre dieses Attribut gemeinsam mit „Mitarbeiter_ID“ und „Datum_Bis“ der zusammengesetzte Primärschlüssel, würde dies dazu führen, dass die in Abbildung 6 gezeigten Eintragungen funktionieren würden und somit etwas darstellen, was es aufgrund gesetzlicher Beschränkungen nicht geben kann.

5.4.2 Sozialversicherungen

Im Rahmen dieser Arbeit wurden drei verschiedene Arten von Beziehungen zu Krankenkassen identifiziert, zu die der Arbeitgeber stehen kann: zum Einen gibt es gesetzlich versicherte Arbeitnehmer, die somit in einer gesetzlichen Krankenkasse kranken- und pflegeversichert sind. Zum Zweiten sind ein Teil der Arbeitnehmer möglicherweise über eine private Krankenkasse versichert und zum Dritten gibt es Arbeitnehmer (zum Beispiel kurzfristig Beschäftigte und Werkstudenten), welche anderweitig versichert sind, aber über den Arbeitgeber umlagepflichtig sind. Diese drei Beziehungsarten werden folgend „Krankenkassenarten“ bezeichnet. Die Teilung in drei Tabellen „gesetzliche_Krankenkassen“, „Privatkrankenkassen“ und „gemeldete_Krankenkassen“ ist deswegen notwendig, weil jede

dieser Krankenkassenarten individuelle Eigenschaften aufweisen. Für gesetzliche Krankenkassen ist ein Zusatzbeitrag zu entrichten, für privat krankenversicherte Arbeitnehmer zahlt der Arbeitgeber Zuschüsse zur privaten Kranken- und Pflegeversicherung und für anderweitig Versicherte gibt es neben der Krankenkasse selbst keine weiteren zusätzlichen Daten, die gespeichert werden müssen. Durch die Dreiteilung ist eine Trennung dieser drei Krankenkassenarten sichergestellt. In jeden dieser drei Fälle muss der Arbeitgeber aber die drei Umlagen entrichten, weswegen die Tabelle „Umlagen“ nicht in drei Tabellen aufgesplittet werden muss, sondern zu den drei Krankenkassenarten jeweils eine n:m-Beziehung unterhält. Stattdessen wird in der Tabelle das Attribut „privat_gesetzlich_oder_anders“ implementiert, welche angibt, zu welcher Krankenkassenart dieser Datensatz jeweils gehören soll. Der dazugehörige Teil des Datenmodells, welches dies beschreibt, ist im Anhang A in „A.1 – Krankenkassen“ hinterlegt.

Die allgemeinen gesetzlichen Krankenversicherungsbeiträge sind von den einzelnen gesetzlichen Krankenkassen entkoppelt, weswegen sie auch im Datenmodell voneinander separiert sind. Da die Arbeitgeber- Und Arbeitnehmerbeiträge zur Pflegeversicherung auf verschiedenen Grundlagen beruhen (Ersteres vom Standort des Arbeitgebers, wo der Mitarbeiter tatsächlich arbeitet, Letzteres von der Anzahl der Kinder), werden Diese ebenfalls getrennt voneinander angeordnet (siehe Abschnitt „2.1.1 Krankenversicherung“ und „2.1.2 Pflegeversicherung“). Auch die Arbeitslosen- und Rentenversicherungsbeiträge sind voneinander und den oben genannten Versicherungen separiert. Deswegen ist auch hier eine entkoppelte Datenmodellierung angebracht. In der Praxis werden in den Tabellen „Arbeitslosenversicherungen“ und „Rentenversicherungen“ maximal nur je ein Datensatz je Mandant anfallen, da für alle Arbeitnehmer, sofern sie arbeitslosen- und/oder rentenversicherungspflichtig sind, dieselben Beitragssätze, Beitragsbemessungs- und Jahresentgeltgrenzen gelten (siehe Abschnitt „2.1.3 Arbeitslosenversicherung“ und „2.1.4 Rentenversicherung“). Diese Tabellen sind aber technisch notwendig, damit eine Historisierung in Bezug auf die Beitragssätze wie auch in Bezug zu den Mitarbeitern möglich ist. Da für Minijobber Pauschalen erhoben werden und nicht von den bereits erläuterten gesetzlichen Sozialversicherungsbeiträgen in einer direkten Abhängigkeit sind, werden diese ebenfalls separat modelliert. Je nachdem, ob der Minijobber kurzfristig oder nicht kurzfristig beschäftigt ist, fallen andere Beiträge beziehungsweise Beitragssätze an (siehe Abschnitt „2.1.7 Versicherungspflichten nach Beschäftigungsformen“), was durch das boolean-Attribut

„kurzfristig_beschäftigt“ in der Tabelle „Minijobs“ technisch berücksichtigt wird. Der dazugehörige Teil des Datenmodells, welches dies beschreibt, ist im Anhang A in „A.2 – Sonstige Sozialversicherungen“ hinterlegt.

Unfallversicherungsbeiträge werden nicht pro Mitarbeiter, sondern pro Unternehmen erhoben (siehe Abschnitt „2.1.5 Unfallversicherung“). Deswegen ist die Berufsgenossenschaft über die Beziehungstabelle „Unfallversicherungsbeiträge“ in einer direkten Beziehung mit der Tabelle „Unternehmen“. Der dazugehörige Teil des Datenmodells, welches dies beschreibt, ist im Anhang A in „A.2 – Sonstige Sozialversicherungen“ hinterlegt.

5.4.3 Entgelt

Mitarbeiter eines Unternehmens können entweder tariflich oder ohne Tarif bezahlt werden. Bei einem Unternehmen, welches keinen Tarifvertrag hat, werden folglich alle Mitarbeiter außertariflich bezahlt. In Unternehmen, welche einen Tarifvertrag haben, werden die meisten Mitarbeiter nach Tarif vergütet. Allerdings gibt es auch in solchen Unternehmen häufig Mitarbeiter, die außertariflich bezahlt werden. Das Datenmodell muss also in der Lage sein, all diese Fälle abzubilden.

Sowohl Tarif- wie auch Nicht-Tarifbeschäftigte haben diverse Vergütungsbestandteile. In jedem Fall existiert das Grundgehalt. Daneben werden häufig weitere Vergütungsbestandteile wie beispielsweise Urlaubs- oder Weihnachtsgeld angeboten. Die Vergütungsbestandteile, welche das jeweilige Unternehmen anbietet, werden in der Tabelle „Verguetungsbestandteile“ gespeichert. Für diejenigen Mitarbeiter, die außertariflich bezahlt werden, werden in der Tabelle „Ausser tarifliche“ ein Datensatz angelegt. Je nachdem, welche Vergütungsbestandteile der außertariflich beschäftigte Mitarbeiter erhält, werden Diese über die Tabelle „hat_Verguetungsbestandteil_AT“ mit dem außertariflichen Mitarbeiter verknüpft. Bekommt ein solcher Mitarbeiter also beispielsweise neben dem Grundgehalt noch Urlaubs- und Weihnachtsgeld, müssen diese drei Vergütungsbestandteile zuerst in der Tabelle „Verguetungsbestandteile“ hinterlegt und dann mit dem Mitarbeiter verknüpft. Als Ergebnis gibt es dann drei Datensätze in „hat_Verguetungsbestandteile_AT“ für diesen einen Mitarbeiter. Hier wird auch der Betrag genannt und für welchen Zeitraum diese Beträge gelten.

Tariflich bezahlte Mitarbeiter werden einem Tarif zugeordnet. Da Tarifverträge meist mit Gewerkschaften verhandelt werden, kann jeder Tarif einer Gewerkschaft zugeordnet werden,

was über die Tabelle „Gewerkschaften“ gewährleistet wird. Erstellt ein Unternehmen eine Tariftabelle ohne Beteiligung einer Gewerkschaft, muss man in der Tabelle „Gewerkschaften“ statt der Gewerkschaft angeben, dass dieser unternehmensintern erstellt wurde. Alle Mitarbeiter desselben Tarifs erhalten bei Vollzeit dieselben Gehälter und Vergütungsbestandteile. Aus dem Grund ist die Tabelle „Verguetungsbestandteile“ mit der Tabelle „Tarife“ über die Tabelle „hat_Verguetungsbestandteil_Tarif“ miteinander verknüpft. Wie in der Tabelle „hat_Verguetungsbestandteile_AT“ wird auch hier in „hat_Verguetungsbestandteil_Tarif“ der Betrag des jeweiligen Vergütungsbestandteil und deren Gültigkeitsdauer gespeichert.

Mitarbeiter, welche kein Entgelt beziehen, beispielsweise unbezahlte Praktikanten, müssen in diesem Teil des Datenmodells nicht erfasst werden. Es steht dem Unternehmen aber frei, solche Mitarbeiter beispielsweise bei „Ausser tarifliche“ anzulegen und den Betrag des Grundgehaltes in „hat_Verguetungsbestandteile_AT“ mit 0€ anzugeben.

Der dazugehörige Teil des Datenmodells, welches dies beschreibt, ist im Anhang A in „A.3 – Entgelt“ hinterlegt.

6 Tests

Um die im Rahmen dieser Arbeit implementierten Funktionalitäten auf richtige Ausführung zu prüfen, wurden automatisierte Tests geschrieben. Da nicht alle für die Praxis benötigten Funktionalitäten implementiert sind, decken die Tests somit auch nicht alle notwendigen Prüfungen ab, um sicherzustellen, dass das System im Gesamten so arbeitet, wie es zu erwarten ist. So sind Systemtests, welche die Funktionsfähigkeit des gesamten Systems prüfen (siehe Abschnitt „3.4 Testen“), nicht möglich, da das System nur prototypisch entwickelt ist und auch die Aufsetzung des Systems in einer Cloud nicht vorgenommen wird. Auch Abnahmetests können nicht durchgeführt werden, da es im Rahmen dieser Arbeit nicht zu einem realen Einsatz in einem Unternehmen kommt.

Die Mehrheit der implementierten Funktionalitäten greifen auf die Personalstammdatenbank zu. Der Zugriff auf die Datenbank bedingt meist mindestens eine Python-Funktion und eine PL/pgSQL-Funktion (siehe Abschnitt „5.2 Struktur des Datenimports“), weswegen bei einem solchen Test mehrere Funktionen aufgerufen werden. Somit handelt es sich hierbei um Integrationstests, welche mehrere Funktionen im Verbund prüfen (siehe Abschnitt „3.4 Testen“).

Jeder Test besteht aus drei Teilen: die `setUp()`-Funktion, die eigentliche Testfunktion und die `tearDown()`-Funktion. Ersteres und Letzteres sind beides Python-spezifische Funktionen. Die `setUp()`-Funktion dient der Vorbereitung des eigentlichen Tests und wird jedes Mal unmittelbar vor dem Start einer Test-Funktion aufgerufen.¹²⁷ Die `tearDown()`-Funktion wird stets unmittelbar nach einer Test-Funktion aufgerufen und kann gemäß der Python-Dokumentation verwendet werden, um die Ergebnisse aufzuzeichnen.¹²⁸ Sie kann aber auch verwendet werden, um den Ausgangszustand wiederherzustellen.

Der grundsätzliche Aufbau einer Testklasse, welche mehrere Test-Funktionen und sowohl `setUp()`- als auch `tearDown()`-Funktion implementieren, sieht folgendermaßen aus:

¹²⁷ Python (o.D. - b): `setUp()`

¹²⁸ Python (o.D. - c): `tearDown()`

```

import unittest

class TestKlasse(unittest.TestCase):

    def setUp(self):
        """
        Vorbereitungsmaßnahmen, welche fuer die Tests stets benoetigt werden
        """

    def testfunktion_1(self):
        """
        Testfunktion, welche eine oder mehrere Funktionen (bei Integrationstests) prueft
        """

    def testfunktion_2(self):
        """
        Testfunktion, welche eine oder mehrere Funktionen (bei Integrationstests) prueft
        """

    def testfunktion_3(self):
        """
        Testfunktion, welche eine oder mehrere Funktionen (bei Integrationstests) prueft
        """

    def tearDown(self):
        """
        Speichert Ergebnisse und/oder kann zur Wiederherstellung der Ausgangssituation verwendet werden
        """

```

Abbildung 21: beispielhafter Aufbau einer Testklasse mit mehreren Test- sowie setUp()- und tearDown()-Funktion; eigener Code dargestellt mit carbon.now.sh

Im Rahmen dieser Arbeit dient die setUp()-Funktion dazu, ein Testschema aufzubauen, welcher exakt dieselbe Personalstammdatenbank mit ihren PL/pgSQL-Skripten enthält wie im Schema „public“, wo die Produktivdatenbank implementiert ist. Hierfür ruft die setUp()-Funktion die eigens geschriebene Funktion „test_set_up()“ auf, die das vornimmt:

```

def test_set_up():

    testschema = 'temp_test_schema'

    # SQL-Code erstellen, mit dem das Testschema erstellt werden kann
    erstelle_schema = f"create schema if not exists {testschema};\n" \
        f"set search_path to {testschema};\n"

    erstelle_tenant_user = "create role tenant_user;\n"

    berechtige_tenant_user_temp_test_schema = f"grant usage on schema {testschema} to tenant_user;\n" \
        f"alter default privileges in schema {testschema} grant select, " \
        f"insert, update, delete on tables to tenant_user;\n" \
        f"alter default privileges in schema {testschema} grant usage on " \
        f"sequences to tenant_user;"

    conn = datenbankverbindung_aufbauen()

    cur = conn.cursor()

    # Auslesen, ob die Rolle 'tenant_user' bereits existiert...
    cur.execute("select rolname from pg_roles where rolname = 'tenant_user'")

    with open("../2 erstelle Tabellen und Stored Procedures.sql") as file:
        # ... falls Rolle 'tenant_user' nicht vorhanden, dann neben Testschema auch 'tenant_user' erstellen ...
        if cur.fetchone() is None:
            setup_testschema = erstelle_schema + \
                erstelle_tenant_user + \
                berechtige_tenant_user_temp_test_schema + \
                file.read()
            cur.execute(setup_testschema)
        # ... falls ja, dann Testschema erstellen, ohne dass die Rolle 'tenant_user' erstellt wird
        else:
            setup_testschema = erstelle_schema + \
                berechtige_tenant_user_temp_test_schema + \
                file.read()
            cur.execute(setup_testschema)

    conn.commit()

    # Cursor und Konnektor zu Datenbank schliessen
    cur.close()
    conn.close()

    return testschema

```

Abbildung 22: test_set_up()-Funktion, welche das Testschema inklusive Datenbank und Skripte erstellt; eigener Code dargestellt mit carbon.now.sh

Es wird das Testschema „temp_test_schema“ erstellt und angewiesen, dass darauffolgender SQL-Code nicht im „public“-Schema sondern in „temp_test_schema“ ausgeführt wird. Die Rolle „tenant_user“, welche RLS berücksichtigt (siehe Abschnitt „5.3 Mandantenfähige Datenbank“), wird im Bedarfsfall erstellt und mit eingeschränkten Zugriffsrechten versehen. Dieser Code wird mit dem SQL-Code, welcher die Datenbank und die Skripte erstellen und welcher in der Datei „2 erstelle Tabellen und Stored Procedures.sql“ gespeichert ist, kombiniert und ausgeführt. Falls die Rolle „tenant_user“ bereits existiert, wird das Testschema ohne „tenant_user“ erstellt.


```
def setUp(self):

    self.testschema = test_set_up()

    # Notwendige Standardvorbereitungen
    login = Login(self.testschema)
    login.registriere_mandant_admin('Testfirma', 'mandantenpw', 'mandantenpw', 'M100000', 'Otto',
                                   'Normalverbraucher', 'adminpw', 'adminpw')
    self.admin = login.login_admin('Testfirma', 'mandantenpw', 'M100000', 'adminpw')
    self.admin.nutzer_anlegen('M100001', 'Erika', 'Musterfrau', 'nutzerpw', 'nutzerpw')

    self.nutzer = login.login_nutzer('Testfirma', 'mandantenpw', 'M100001', 'nutzerpw')
    self.nutzer.passwort_aendern('neues passwort', 'neues passwort')

    # Zusätzliche testklassenspezifische Vorbereitungen
    self.nutzer.insert_austrittsgrundkategorie(
        'testdaten_insert_austrittsgrundkategorie/Austrittsgrundkategorie.xlsx')
```

Abbildung 23: für diese Arbeit typische setUp()-Funktion; eigener Code dargestellt mit carbon.now.sh

Nach Erstellung des Testschemas wird in der setUp()-Funktion noch notwendige Vorbereitungen getroffen. Dies betrifft vor allem die Registrierung eines Mandanten und Administratoren, die Anlage eines Nutzers und dessen Login, damit mithilfe dieses Nutzers in den Test-Funktionen dessen Funktionalitäten geprüft werden können. In einigen Fällen sind weitere Vorbereitungen notwendig, welche in der „setUp()“-Funktion vorgenommen werden. So muss, wenn beispielsweise die Eintragung eines neuen Austrittsgrundes (zum Beispiel „Umsatzrückgang“) getestet werden soll, zuerst die entsprechende Austrittsgrundkategorie (bei „Umsatzrückgang“ ist das „betriebsbedingt“), vorab eingetragen werden, da jeder Austrittsgrund einer Kategorie zugeordnet werden muss.

In der „tearDown()“-Funktion wird nach Durchlauf des Tests anschließend das komplette Testschema entfernt. Hierfür wird die eigens geschriebene „test_tear_down()“ aufgerufen:

```
def tearDown(self):

    test_tear_down()

def test_tear_down():

    conn = datenbankbverbindung_aufbauen()

    cur = conn.cursor()
    cur.execute(f"set role postgres;"
               f"DROP SCHEMA temp_test_schema CASCADE;")
    conn.commit()
    cur.close()
    conn.close()
```

Abbildung 24: „tearDown“-Funktion ruft „test_tear_down()“ auf, um Testschema zu entfernen; eigener Code dargestellt mit carbon.now.sh

Die Entfernung des Testschemas erfolgt durch den SQL-Befehl „DROP SCHEMA temp_test_schema CASCADE;“. Die komplette Entfernung des Testschemas nach Ablauf jeder

Testfunktion erfolgt deshalb, weil sichergestellt sein soll, dass jeder Test auf einer komplett leeren Datenbank erfolgt. Würde die Entfernung des Testschema nicht erfolgen, könnten Eintragungen aus vorherigen Tests das Verhalten eines darauffolgenden Tests beeinflussen, was nicht gewünscht ist.

Integrationstests, welche einen erfolgreichen Eintrag in der Datenbank prüfen, besitzen meist folgendes Schema:

```
def test_erfolgreicher_eintrag(self):
    """
    Test prueft, ob ein Austrittsgrund eingetragen wird.
    """
    self.nutzer.insert_austrittsgrund('testdaten_insert_austrittsgrund/Austrittsgrund.xlsx')

    # Inhalt aus Tabelle ziehen, um zu pruefen, ob der Datensatz angelegt wurde
    ergebnis = self.nutzer.abfrage_ausfuehren("SELECT * FROM austrittsgruende")
    self.assertEqual(str(ergebnis), "[[1, 1, 'Umsatzrueckgang', 1]]")
```

Abbildung 25: typische Integrationstest, der einen erfolgreichen Datenbankeintrag prüft; eigener Code dargestellt mit carbon.now.sh

Es erfolgt zuerst der Eintrag in die Datenbank mithilfe der entsprechenden Funktionalität. Anschließend wird eine SQL-SELECT-Abfrage an das Datenbanksystem gesendet, welche den Inhalt der Tabelle aufruft, welche die Daten nach dem Eintrag enthalten muss. Anschließend wird geprüft, ob das tatsächliche Ergebnis mit dem erwarteten Ergebnis übereinstimmt. Hierfür wird die „assertEqual()“-Funktion verwendet, welche abprüft, ob der Inhalt der beiden übergebenen Pflicht-Parameter (das tatsächliche und erwartete Ergebnis) exakt übereinstimmen. Sobald es einen Unterschied gibt, wird eine Fehlermeldung ausgegeben, womit der Test nicht bestanden ist.¹²⁹

Im Rahmen dieser Arbeit wird meist die „assertEqual()“-Funktion verwendet, um zu prüfen, ob Ergebnisse übereinstimmen. Das unittest-Framework von Python implementiert allerdings weitere „assert“-Funktionen, welche für Tests verwendet werden können. Es werden noch die „assertTrue()“-Funktion, welche prüft, ob es sich um den boolean-Wert „True“ handelt¹³⁰, und die „assertRaises()“-Funktion, welche prüft, ob eine Exception ausgelöst wird, verwendet.¹³¹ Diese Funktionen werden im Rahmen dieser Arbeit verwendet, um zu testen, ob bei einem Eintrag in die Datenbank vom entsprechenden PL/pgSQL-Skript erwartungsgemäß eine Exception geworfen wird, wenn diese Daten bereits hinterlegt sind und nicht mehrfach

¹²⁹ Python (o.D. - d): assertEquals()

¹³⁰ Python (o.D. - e): assertTrue()

¹³¹ Python (o.D. - f): assertRaises()

eingetragen werden dürfen, weil sie sonst beispielsweise gegen den unique-constraint einer Tabelle verstoßen:

```
def test_kein_doppelter_eintrag_Abteilung_und_abkuerzung_identisch(self):
    """
    Test prueft, ob bei wiederholtem Aufruf der Methode 'insert_abteilung' mit derselben Abteilung und Abkuerzung
    dieser nicht mehrfach eingetragen wird. Beim zweiten Eintrag muss eine Fehlermeldung ausgegeben werden.
    """
    self.nutzer.insert_abteilung('testdaten_insert_abteilung/Abteilung.xlsx')

    # Versuch, denselben Wert noch einmal einzutragen
    with self.assertRaises(Exception) as context:
        self.nutzer.insert_abteilung('testdaten_insert_abteilung/Abteilung.xlsx')

    erwartete_fehlermeldung = "FEHLER: Abteilung 'Human Resources Personalcontrolling' bereits vorhanden!"
    tatsaechliche_fehlermeldung = str(context.exception)
    self.assertTrue(tatsaechliche_fehlermeldung.startswith(erwartete_fehlermeldung))

    # Inhalt aus Tabelle ziehen, um zu pruefen, ob der Datensatz auch nur einmal angelegt wurde
    ergebnis = self.nutzer.abfrage_ausfuehren("SELECT * FROM abteilungen")
    self.assertEqual(str(ergebnis), "[(1, 1, 'Human Resources Personalcontrolling', 'HR PC', None)]")
```

Abbildung 26: Verwendung der „assertRaises“- und „assertTrue“-Funktion; eigener Code dargestellt mit carbon.now.sh

Zuerst werden Daten das erste Mal in eine Datenbank eingetragen und direkt danach wird versucht, exakt dieselben Daten ein weiteres Mal einzutragen. Da in dem Fall ein constraint verletzt wird, muss eine Fehlermeldung ausgegeben werden, was mit „assertRaises()“ geprüft wird. Es soll ein bestimmter Text ausgegeben werden. Hierfür wird mithilfe der „assertTrue“-Funktion geprüft, ob zurückgegebene Fehlermeldung tatsächlich wie erwartet lautet. Um sicher zu gehen, dass die Daten tatsächlich kein zweites Mal eingetragen wurden, wird dennoch nochmal eine SQL-SELECT-Abfrage an das Datenbanksystem gesendet und geprüft, ob die Daten tatsächlich nur einmal hinterlegt sind.

Des Weiteren werden noch die „assertIsNotNone()“- und die „assertIsNone()“-Funktionen verwendet, welche prüfen, ob eine Variable nicht „None“ beziehungsweise „None“ ist.¹³² Diese Funktionen werden verwendet, um zu prüfen, ob beispielsweise die Entfernung eines Administrators erfolgreich durchgeführt wird, wenn ein Unternehmen den Service der Personalstammdatenbank nicht mehr verwenden möchte und deswegen seinen Mandanten- und den dazugehörigen Administrator-Account löscht:

¹³² Python (o.D. - g): assertIsNone()

```

def test_admin_entfernt(self):
    """
    Test prueft, ob Administrator entfernt wird
    """

    # Pruefung, ob Administrator vorhanden ist
    admin = None

    for i in range(len(self.login.liste_admins)):
        if self.login.liste_admins[i].get_personalnummer() == 'M100000' and \
            self.login.liste_admins[i].get_mandant().get_mandantenname() == 'Testfirma':
            admin = self.login.liste_admins[i]

    self.assertIsNotNone(admin)

    # Administrator (und Mandant) entfernen
    self.login.entferne_mandant_nutzer_und_admin('Testfirma', 'mandantenpw', 'mandantenpw', 'M100000', 'adminpw',
                                                'adminpw')

    # Pruefen, ob Administrator nun entfernt ist
    admin = None

    for i in range(len(self.login.liste_admins)):
        if self.login.liste_admins[i].get_personalnummer() == 'M100000' and \
            self.login.liste_admins[i].get_mandant().get_mandantenname() == 'Testfirma':
            admin = self.login.liste_admins[i]

    self.assertIsNone(admin)

```

Abbildung 27: Test, welcher die erfolgreiche Löschung des Administrator-Accounts prüft; eigener Code dargestellt mit *carbon.now.sh*

Zuerst wird mithilfe der „assertIsNotNone()“-Funktion geprüft, ob der zu entfernende Administrator tatsächlich existiert. Anschließend wird mithilfe der Funktion „entferne_mandant_nutzer_und_admin()“ sowohl der Mandantenaccount (welche sämtliche Nutzer-Accounts beinhaltet) und der Administrator-Account gelöscht. Danach wird mithilfe der „assertIsNone()“-Funktion geprüft, ob der Administrator-Account nun entfernt ist.

7 Fazit

In diesem Kapitel wird einerseits zusammengefasst, was im Zuge dieser Arbeit entwickelt wurde und andererseits, welche Maßnahmen noch ergriffen werden müssen, damit aus dem Prototypen ein vollständiges System wird, welches in der Praxis angewendet werden kann.

7.1 Ergebnisse der Arbeit

Ziel dieser Arbeit war es, ein Prototyp für eine historisierende Personalstammdatenbank zu entwickeln, auf die über diverse Funktionalitäten mit ihr interagiert werden kann. Diese wurden zum Teil über Python-Funktionen und zum Teil über PL/pgSQL-Skripte, welche zusammenarbeiten, für beispielhafte Use Cases implementiert. Darüber hinaus sollte die Datenbank mandantenfähig sein, damit sie als SaaS später in einer Cloud zur Verfügung gestellt werden kann. Zudem ist das Datenmodell historisiert gestaltet, in dem Gültigkeitszeiträume eingetragen werden müssen. Somit sind auch vergangenheitsbezogene Betrachtungen möglich, selbst wenn die Daten nicht mehr aktuell sein könnten.

Hierfür wurden die für diese Arbeit notwendigen Grundlagen der Personalwirtschaft, insbesondere in Bezug auf Sozialversicherungen, Entgelt, aber auch die Definition, was Personalstammdaten sind, beschrieben. Zudem wurden auch die technologischen Grundlagen, welche für diese Arbeit benötigt wurden, beleuchtet. Dies betrifft vor allem den Aufbau und Funktionsweise von (objekt-)relationalen Datenbanken, Cloud Computing/SaaS und Mandantenfähigkeit. Da durch fehlerhaften Code fehlerhaftes Verhalten resultieren kann, wird auch Testing behandelt.

Die Datenbank muss so aufgebaut sein, dass Informationen, welche im Zusammenhang mit den Mitarbeitern stehen, wie beispielsweise Sozialversicherungsbeiträge und Entgelttarife, separat von einer Mitarbeitereintragung in die Datenbank eingespeichert werden können. Da die Datenbank mit Historisierung arbeitet, sind auch das Datum, ab wann Daten gültig sind, anzugeben. Wenn sich beispielsweise Beitragssätze für eine Sozialversicherung ändern, muss es möglich sein, eine Aktualisierung durchzuführen, in dessen Zuge die veralteten Daten mit dem Datum versehen werden, ab dem sie nicht mehr aktuell sind, und die neuen Daten mit einem Datum versehen werden, ab dem sie gültig sind. Dies ist in der Regel der darauffolgende Tag, nachdem veraltete Datensätze nicht mehr gültig sind.

Um sicherzustellen, dass die implementierten Funktionalitäten sich wie erwartet verhalten, wurden Tests implementiert, die dieses Verhalten abprüfen. Hierbei wurden nicht nur erfolgreiche Testfälle, also Fälle, wo sich eine Funktionalität wie erwartet verhält, sondern auch Ausnahmefalltests, in der Fehlermeldungen erscheinen und die Funktionalität abgebrochen werden müssen, entwickelt.

Erste grundlegende Maßnahmen zur Sicherheit des Systems wurden ergriffen, indem eine Registratur von Mandant und Administrator eingeführt wurde. Nachdem der Administrator Nutzer-Accounts erstellt hat, müssen sich Administrator und Nutzer im System mithilfe ihrer Personalnummer und ihren Passwörtern und zusätzlich mit dem Mandantenpasswort einloggen. Wird das Passwort dreimal hintereinander falsch eingegeben, wird der Nutzer gesperrt. Die Entsperrung kann nur durch den Administrator erfolgen, der hierfür zusätzlich ein neues Passwort an den Nutzer vergibt. Loggt sich der Nutzer anschließend wieder ein, muss er sein Passwort erneut ändern, bevor er seine Funktionalitäten anwenden kann. Dies erfolgt, damit auch der Administrator nicht mehr das Passwort des entsprechenden Nutzers kennt.

Soll das System nicht mehr genutzt werden, so kann der Administrator sämtliche Daten seines Mandanten entfernen sowie seinen als auch den Mandanten-Account – und somit auch die Nutzer-Accounts, da sie im Mandanten-Account gespeichert sind – löschen.

7.2 Zukunftsausblick

Der derzeitige Prototyp ist für einen realen Praxiseinsatz noch nicht fertig entwickelt. Es müssen weitere Sicherheitsmaßnahmen ergriffen werden. So sind die Passwörter in der Datenbank einsehbar, da sie unverschlüsselt in den Tabellen „Mandanten“, „Administratoren“ und „Nutzer“ zu finden sind. Die Passwörter sollten zum einen verschlüsselt sein und zum anderen nur als „*-Zeichen zu sehen sein. Zusätzlich wäre es sinnvoll, eine Zwei-Faktor-Authentifizierung über E-Mail oder Smartphone einzurichten, bei der der Anwender sich bestätigen muss. So wäre der Zugriff weiterhin nicht gegeben, wenn eine nicht-berechtigte Person es schaffen sollte, sich die Login-Daten eines Nutzers und/oder Administrators zu beschaffen. Auch die Daten zur Datenbankverbindung im Python-Code ist nicht verschlüsselt und muss beispielsweise mit Umgebungsvariablen vorgenommen werden.

Des Weiteren muss noch ein Frontend entwickelt werden, worüber die Dateneingabe erfolgt. Die bisherige Lösung, dies über Excel-Dateien zu tätigen, ist fehleranfällig, da bei Excel unter anderem die Gefahr besteht, dass automatische Korrekturen durch die Tabellenkalkulationssoftware vorgenommen werden, diese im schlechtesten Fall unbemerkt bleiben, dann fehlerhaft sind und in dem Zustand in die Datenbank übertragen werden. Zudem ist es beim derzeitigen Stand noch notwendig, auf die main-Datei zuzugreifen und dort den Python-Code einzugeben, wenn entsprechende Funktionalitäten ausgeführt werden sollen. Bei einem praktisch einsatzbereiten System darf der Anwender keinerlei Zugriff auf den Backend- oder gar Datenbank-Code haben, da er sonst Code ändern kann.

Beim derzeitigen Stand hat der Anbieter keinerlei Funktionalitäten, um im System einzugreifen. Deswegen muss für ihn noch ein Provider-Administrator implementiert werden, der beispielsweise Mandanten sperren kann, wenn sie beispielsweise fällige Gebühren nicht zahlen. Zudem sollte er die Möglichkeit haben, Administratoren auf Mandantenseite zu entsperren, wenn Diese das Passwort dreimal hintereinander falsch eingetragen haben.

Da im Rahmen dieser Arbeit die SaaS-Personalstammdatenbank nicht in eine Cloud hochgeladen wird, sind keine Funktionen, welche gegebenenfalls für die Interaktion mit einer Cloud notwendig sind, implementiert. Diese müssen noch zur Verfügung gestellt werden. Nachdem das System in die Cloud hochgeladen wurde, müssen noch Systemtests durchgeführt werden, um zu prüfen, ob eine zufriedenstellende Performanz gewährleistet ist.

Speziell im Bereich der Datenaktualisierung müssen noch weitere Funktionen implementiert werden. Bisher sind lediglich beispielhafte Aktualisierungsfunktionen für Adressänderungen, Mitarbeiterentlassung, der Höhe der Krankenversicherungsbeiträge und die Erstellung von Abteilungshierarchien implementiert worden. Es müssen aber für alle möglichen Fälle von Änderungen entsprechende Funktionen zur Verfügung stehen. Als Beispiele für noch zu implementierende Änderungsfunktionen seien hier der Wechsel einer Krankenkasse, der Wechsel in eine andere Abteilung oder der Wechsel von einer gesetzlichen zur privaten Kranken- und Pflegeversicherung (und andersrum) genannt.

Es sollte noch die Möglichkeit zur Verfügung gestellt werden, jedem Nutzer einen individuellen Umfang an Zugriffsrechten zu geben. So sollten beispielsweise Nutzer, die sich nicht mit Sozialversicherungen auskennen, nicht die Möglichkeit haben, in diesem Bereich Daten hinzuzufügen beziehungsweise zu manipulieren.

Die Datenbank ist so gestaltet, dass theoretisch das Datenmodell um weitere Tabellen erweitert werden kann. So könnten zum Beispiel Tabellen angelegt werden, die die Abwesenheitszeiten von Mitarbeitern erfassen. Zudem kann die Datenbank um das Thema Lohnsteuer erweitert werden. Dann wäre es beispielsweise möglich, die Höhe der Lohnsteuer für die Mitarbeiter zu berechnen. Diese genannten Beispiele sind aber keine Stammdaten mehr, weswegen Diese im Rahmen dieser Arbeit nicht behandelt wurden. Würde man die Datenbank um sämtliche Daten erweitern, welche in der Personalwirtschaft anfallen, hätte das den Vorteil, dass ein Unternehmen die Datenerfassung der kompletten Personalwirtschaft in dieser Datenbank vornehmen könnte.

Quellenverzeichnis

AOK (o.D. - a): Sozialversicherungspflicht und -freiheit; Online:

<https://www.aok.de/fk/sozialversicherung/sozialversicherungspflicht-und-freiheit/>
[letzter Aufruf: 09.02.2024]

AOK (o.D. - b): Auszubildende und Sozialversicherung; Online:

<https://www.aok.de/fk/sozialversicherung/ausbilden/auszubildende-und-sozialversicherung/> [letzter Aufruf: 09.02.2024]

AOK (o.D. - c): Beschäftigung von Werkstudenten; Online:

<https://www.aok.de/fk/sozialversicherung/studenten-und-praktikanten/beschaeftigung-von-werkstudenten/> [letzter Aufruf: 09.02.2024]

AOK (o.D. - d): Werkstudenten beschäftigen; Online:

<https://www.deine-gesundheitswelt.de/firmenkunden/news/werkstudenten> [letzter Aufruf: 09.02.2024]

AOK (o.D. - e): Praktikanten in der Sozialversicherung; Online:

<https://www.aok.de/fk/sozialversicherung/studenten-und-praktikanten/praktikanten-in-der-sozialversicherung/> [letzter Aufruf: 09.02.2024]

AOK Nordost (2024): Beitrag zur Krankenversicherung bei einem Nebenjob als Studierender;

Online:

https://www.aok.de/pk/krankenkassenbeitraege/nebenverdienst/?s_kwcid=AL!10517!3!614135137324!e!!g!!werkstudent+krankenversicherung&cid=aok%3B1%3B7%3BMW%3B1%3B1%3Bwerkstudent+krankenversicherung%3B%3B%3B%3B%3B%3B%3B%3B%3B%3BNBrand_ZG%3BWerkstudent%3Bkid5692962bid17227657&gclid=EAlaIQobChMIpLzh8bqZhAMVhE5BAh3BeAtkEAAYASAAEgJTdvD_BwE#c1590608016
[letzter Aufruf: 09.02.2024]

aws (o.D.): Empfehlungen auf der Sicherheit auf Zeilenebene (sic!); Online:

https://docs.aws.amazon.com/de_de/prescriptive-guidance/latest/saas-multitenant-managed-postgresql/rls.html [letzter Aufruf: 09.02.2024]

BARMER (2022): Einzugsstelle Sozialversicherung; Online:

[https://www.barmer.de/firmenkunden/sozialversicherung/sozialversicherungslexikon/einzugsstelle-1057732#Einzugsstelle was gilt bei privat krankenversicherten Besch00E4ftigte n-1057732](https://www.barmer.de/firmenkunden/sozialversicherung/sozialversicherungslexikon/einzugsstelle-1057732#Einzugsstelle_was_gilt_bei_privat_krankenversicherten_Beschu00E4ftigte_n-1057732) [letzter Aufruf: 09.02.2024]

BG BAU (o.D.): Praktikanten; Online:

<https://www.bgbau.de/service/haeufig-nachgefragt/unfallversicherung-a-z/praktikanten> [letzter Aufruf: 09.02.2024]

BGHW (o.D.): Wer ist versichert?; Online:

<https://www.bghw.de/der-versicherungsschutz-der-bghw/wer-ist-versichert> [letzter Aufruf: 09.02.2024]

bitkom (2023): Cloud-Nutzung wird rasant zunehmen; Online:

<https://www.bitkom.org/Presse/Presseinformation/Cloud-Report-2023-Nutzung-rasant-zunehmen> [letzter Aufruf: 09.02.2024]

BKK Landesverband Mitte (o.D.): Häufig gestellte Fragen; Online:

<https://www.bkkmitte.de/bkk-aag/ausgleichsverfahren/haeufig-gestellte-fragen.html#AllgemeineFragen> [letzter Aufruf: 09.02.2024]

Bröckermann, Reiner (2021): Personalwirtschaft – Lehr- und Übungsbuch für Human

Resources Management; 8. Auflage; Stuttgart: Schäffer-Poeschel Verlag für

Wirtschaft · Steuern · Recht GmbH; ePDF-ISBN: 978-3-7910-5174-1

Bühler, Peter / Schlaich, Patrick / Sinner, Dominik (2019): Datenmanagement – Daten –

Datenbanken - Datensicherheit; Berlin: Springer Vieweg; e-ISBN: 978-3-662-55507-1;

<https://doi.org/10.1007/978-3-662-55507-1>

Bundesamt für Sicherheit in der Informationstechnik (o.D.): Was ist Cloud Computing?

Online: [https://www.bsi.bund.de/DE/Themen/Unternehmen-und-](https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Empfehlungen-nach-Angriffszielen/Cloud-Computing/Grundlagen/grundlagen_node.html)

[Organisationen/Informationen-und-Empfehlungen/Empfehlungen-nach-](https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Empfehlungen-nach-Angriffszielen/Cloud-Computing/Grundlagen/grundlagen_node.html)

[Angriffszielen/Cloud-Computing/Grundlagen/grundlagen_node.html](https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Empfehlungen-nach-Angriffszielen/Cloud-Computing/Grundlagen/grundlagen_node.html) [letzter Aufruf:

09.02.2024]

Bundesministerium für Arbeit und Soziales (2021): Sozialversicherung; Online:

<https://www.bmas.de/DE/Soziales/Sozialversicherung/sozialversicherung-art.html>

[letzter Aufruf: 09.02.2024]

Bundesministerium für Arbeit und Soziales (2024): Kurzfristige Beschäftigung; Online:

[https://www.bmas.de/DE/Soziales/Sozialversicherung/Geringfuegige-](https://www.bmas.de/DE/Soziales/Sozialversicherung/Geringfuegige-Beschaeftigung/kurzfristige-beschaeftigung-art.html)

[Beschaeftigung/kurzfristige-beschaeftigung-art.html](https://www.bmas.de/DE/Soziales/Sozialversicherung/Geringfuegige-Beschaeftigung/kurzfristige-beschaeftigung-art.html) [letzter Aufruf: 09.02.2024]

Bundesministerium für Arbeit und Soziales (o.D.): Wer ist unfallversichert?; Online:

[https://www.bmas.de/DE/Soziales/Gesetzliche-Unfallversicherung/Fragen-und-](https://www.bmas.de/DE/Soziales/Gesetzliche-Unfallversicherung/Fragen-und-Antworten/faq-wer-ist-unfallversichert-art.html)

[Antworten/faq-wer-ist-unfallversichert-art.html](https://www.bmas.de/DE/Soziales/Gesetzliche-Unfallversicherung/Fragen-und-Antworten/faq-wer-ist-unfallversichert-art.html) [letzter Aufruf: 09.02.2024]

Bundesministerium für Gesundheit (2016): Leistungskatalog der Krankenversicherung;

Online: [https://www.bundesgesundheitsministerium.de/service/begriffe-von-a-](https://www.bundesgesundheitsministerium.de/service/begriffe-von-a-z/l/leistungskatalog)

[z/l/leistungskatalog](https://www.bundesgesundheitsministerium.de/service/begriffe-von-a-z/l/leistungskatalog) [letzter Aufruf: 09.02.2024]

Bundesministerium für Gesundheit (2017): Private Pflege-Pflichtversicherung; Online:

[https://www.bundesgesundheitsministerium.de/service/begriffe-von-a-z/p/private-](https://www.bundesgesundheitsministerium.de/service/begriffe-von-a-z/p/private-pflege-pflichtversicherung)

[pflege-pflichtversicherung](https://www.bundesgesundheitsministerium.de/service/begriffe-von-a-z/p/private-pflege-pflichtversicherung) [letzter Aufruf: 09.02.2024]

Bundesministerium für Gesundheit (2023 - b): Beiträge; Online:

<https://www.bundesgesundheitsministerium.de/beitraege.html> [letzter Aufruf: 09.02.2024]

Bundesministerium für Gesundheit (2023 - c): Die Pflegeversicherung; Online:

<https://www.bundesgesundheitsministerium.de/themen/pflege/online-ratgeber-pflege/die-pflegeversicherung> [letzter Aufruf: 09.02.2024]

Bundesministerium für Gesundheit (2023 - d): Private Krankenversicherung (PKV); Online:

<https://www.bundesgesundheitsministerium.de/private-krankenversicherung>
[letzter Aufruf: 09.02.2024]

Crowder, James A. / **Hoff**, Curtis W. (2022): Requirements Engineering: Laying a Firm

Foundation; Cham (Schweiz): Springer Nature Switzerland; e-ISBN: 978-3-030-91077-8

DBeaver (o.D.): About DBeaver; Online: <https://dbeaver.com/docs/dbeaver/> [letzter Aufruf:

09.02.2024]

destatis (o.D.): Tarifbindung von Arbeitnehmern; Online:

<https://www.destatis.de/DE/Themen/Arbeit/Arbeitsmarkt/Qualitaet-Arbeit/Dimension-5/tarifbindung-arbeitnehmer.html> [letzter Aufruf: 09.02.2024]

Deutsche Gesetzliche Unfallversicherung (o.D. - a): Wir über uns; Online:

<https://www.dguv.de/de/wir-ueber-uns/index.jsp> [letzter Aufruf: 09.02.2024]

Deutsche Gesetzliche Unfallversicherung (o.D. - b): Kein Buch mit sieben Siegeln: Die

Beitragsberechnung; Online:

https://www.dguv.de/de/ihr_partner/unternehmen/beitragsberechnung/index.jsp
[letzter Aufruf: 09.02.2024]

Deutsche Rentenversicherung (o.D. - a): Beitragszuschlag für Kinderlose; Online:

https://www.deutsche-rentenversicherung.de/DRV/DE/Experten/Arbeitgeber-und-Steuerberater/summa-summarum/Lexikon/B/beitragszuschlag_fuer_kinderlose.html [letzter Aufruf: 09.02.2024]

Deutsche Rentenversicherung (o.D. - b): Arbeitslosenversicherung / Arbeitsförderung;

Online: https://www.deutsche-rentenversicherung.de/SharedDocs/Glossareintraege/DE/A/arbeitslosenversicherung_g_arbeitsfoerderung.html [letzter Aufruf: 09.02.2024]

Deutsche Rentenversicherung (o.D. - c): Unternehmensprofil; Online:

<https://www.deutsche-rentenversicherung.de/Bund/DE/Ueber-uns/Organisation/unternehmensprofil.html> [letzter Aufruf: 09.02.2024]

Deutsche Rentenversicherung (o.D. - d): Meinen Rentenversicherungsträger finden; Online:

<https://www.deutsche-rentenversicherung.de/DRV/DE/Beratung-und-Kontakt/Beratung-suchen-und-buchen/Meinen-Traeger-finden/meinen-traeger-finden.html> [letzter Aufruf: 09.02.2024]

Deutsche Rentenversicherung (o.D. - e): Pflichtversicherte und freiwillig Versicherte; Online:

https://www.deutsche-rentenversicherung.de/DRV/DE/Rente/Arbeitnehmer-und-Selbststaendige/02_Pflicht-und-freiwillig-Versicherte/pflicht-und-freiwillig-versicherte_Inhalt_01_selbstaendig_und_pflichtversichert.html [letzter Aufruf: 09.02.2024]

Deutsche Rentenversicherung (o.D. - f): Insolvenzgeldumlage; Online:

<https://www.deutsche-rentenversicherung.de/DRV/DE/Experten/Arbeitgeber-und-Steuerberater/summa-summarum/Lexikon/I/insolvenzgeldumlage.html> [letzter Aufruf: 09.02.2024]

Deutsche Rentenversicherung (o.D. - g): Kurzfristige Beschäftigung; Online:

https://www.deutsche-rentenversicherung.de/DRV/DE/Experten/Arbeitgeber-und-Steuerberater/summa-summarum/Lexikon/K/kurzfristige_beschaeftigung.html

[letzter Aufruf: 09.02.2024]

Die Techniker (o.D. – a): TK-Beitragssatz; Online:

<https://www.tk.de/techniker/leistungen-und-mitgliedschaft/informationen-versicherte/beitraege-beitragssaetze/tk-beitragssatz-und-zusatzbeitrag-2021472>

[letzter Aufruf: 09.02.2024]

Die Techniker (o.D. - b): ermäßigter Beitragssatz; Online:

<https://www.tk.de/techniker/leistungen-und-mitgliedschaft/informationen-versicherte/veraenderung-berufliche-situation/versichert-als-arbeitnehmer/beitraege-fuer-beschaeftigte/ermaessigter-beitragssatz-personenkreis-2005506> [letzter Aufruf: 09.02.2024]

Die Techniker (o.D. - c): Wie hoch sind die Beitragsbemessungsgrenzen?; Online:

<https://www.tk.de/firmenkunden/versicherung/beitraege-faq/zahlen-und-grenzwerte/beitragsbemessungsgrenzen-2033026> [letzter Aufruf: 09.02.2024]

Die Techniker (o.D. - d): Was ist die Jahresarbeitsentgeltgrenze und wie hoch ist sie?; Online:

<https://www.tk.de/firmenkunden/versicherung/beitraege-faq/zahlen-und-grenzwerte/hoehe-der-jahresarbeitsentgeltgrenze-2033028?tkcm=aaus> [letzter Aufruf: 09.02.2024]

Die Techniker (o.D. - e): Pflegeversicherungsbeitrag; Online:

<https://www.tk.de/firmenkunden/versicherung/beitraege-faq/pflegereform-2023/wie-hoch-ist-pv-beitrag-ab-01072023-2149454?tkcm=aaus> [letzter Aufruf: 07.01.2024]

Die Techniker (o.D. - f): Wie sind die aktuellen Beitragssätze in der Sozialversicherung?;

Online: <https://www.tk.de/firmenkunden/versicherung/beitraege-faq/beitragssaetze/aktuelle-beitragssaetze-in-der-sozialversicherung-2031554>

[letzter Aufruf: 09.02.2024]

Die Techniker (o.D. - g): Entgeltfortzahlungsversicherung - Wer kann sich versichern?;

Online: <https://www.tk.de/firmenkunden/versicherung/versicherung-faq/entgeltfortzahlungsversicherung/wer-kann-sich-in-entgeltfortzahlungsversicherung-versichern-2031638> [letzter Aufruf: 09.02.2024]

Die Techniker (o.D. - h): U2 - Mutterschaft: Welche Beschäftigungsverbote gibt es?; Online:

<https://www.tk.de/firmenkunden/versicherung/versicherung-faq/entgeltfortzahlungsversicherung/u2-mutterschaft-welche-beschaefligungsverbote-gibt-es-2156464> [letzter Aufruf: 09.02.2024]

Die Techniker (o.D. - i): Die Insolvenzgeldumlage 2024; Online:

<https://www.tk.de/firmenkunden/service/fachthemen/jahreswechsel/insolvenzgeldumlage-2024-2047584> [letzter Aufruf: 09.02.2024]

Die Techniker (o.D. - j): Wie hoch ist der Arbeitgeberzuschuss, wenn meine Beschäftigten privat versichert sind? Online:

<https://www.tk.de/firmenkunden/versicherung/beitraege-faq/arbeitsentgelt/hoehe-arbeitgeberzuschuss-zur-privaten-krankenversicherung-2034496?tkcm=aaus> [letzter Aufruf: 09.02.2024]

Die Techniker (o.D. - k): Sind Praktikant:innen sozialversicherungspflichtig oder -frei?;

Online: <https://www.tk.de/firmenkunden/versicherung/versicherung-faq/praktikanten/sind-praktikanten-sozialversicherungspflichtig-oder-sv-frei-2036246?tkcm=aaus> [letzter Aufruf: 09.02.2024]

Die Techniker (o.D. - l): Welche Praktika sind vollständig versicherungsfrei?; Online:

<https://www.tk.de/firmenkunden/versicherung/versicherung-faq/praktikanten/welche-praktika-sind-vollstaendig-versicherungsfrei-2036546>

[letzter Aufruf: 09.02.2024]

Die Techniker (o.D. - m): Welche Vor- und Nachpraktika sind versicherungspflichtig?; Online:

<https://www.tk.de/firmenkunden/versicherung/versicherung-faq/praktikanten/vor-und-nachpraktika-versicherungspflichtig-2036562> [letzter Aufruf: 09.02.2024]

Hildebrand, Knut / Gebauer, Marcus / Hinrichs, Holger / Mielke, Michael (2018): Daten-

und Informationsqualität – Auf dem Weg zur Information Excellence; 4. Auflage;

Springer Fachmedien Wiesbaden GmbH; e-ISBN: 978-3-658-21994-9; doi:

<https://doi.org/10.1007/978-3-658-21994-9>

Hruschka, Peter (2023): Business Analysis und Requirements Engineering – Produkte und

Prozesse verbessern; 3. Auflage; Carl Hanser Verlag München; e-Book-ISBN: 978-3-

446-47819-0

IHK Regensburg (o.D.): Minijobs und Kurzfristige Beschäftigung; Online:

<https://www.ihk.de/regensburg/fachthemen/recht/arbeitsrecht/arbeitsvertrag-und-beschaeftigungsverhaeltnisse/mini-jobs-und-kurzfristige-beschaeftigung-712378>

[letzter Aufruf: 09.02.2024]

KKH (o.D.): Absicherung im Krankheitsfall, Mutterschutz und Insolvenz; Online:

<https://www.kkh.de/firmenkunden/beitrag-sozialversicherung/erstattung-umlage>

[letzter Aufruf: 09.02.2024]

Krankenkasseninfo (o.D.): U1 Umlage, U2 und U3; Online:

<https://www.krankenkasseninfo.de/zahlen-fakten/umlage/> [letzter Aufruf:

09.02.2024]

Kratzke, Nane (2022): Cloud-Native Computing - Software Engineering von Diensten und Applikationen für die Cloud; Carl Hanser Verlag München; e-Book-ISBN: 978-3-446-47284-6

Haufe (o.D.): Praktikanten: Beurteilung in der Entgeltabrechnung / 3.4 Beiträge zur Unfallversicherung sowie den Umlagen U1, U2 und U; Online:
https://www.haufe.de/personal/haufe-personal-office-platin/praktikanten-beurteilung-in-der-entgeltabrechnung-34-beitraege-zur-unfallversicherung-sowie-den-umlagen-u1-u2-und-u3_idesk_PI42323_HI11782380.html [letzter Aufruf: 09.02.2024]

JetBrains (o.D.): PyCharm; Online: <https://www.jetbrains.com/de-de/pycharm/>
[letzter Aufruf: 09.02.2024]

Kersken, Sascha (2019): IT-Handbuch für Fachinformatiker – Der Ausbildungsbegleiter; 9. Auflage; Berlin: Springer-Verlag GmbH Deutschland; ISBN: 978-3-662-62302-2

Kronthaler, Franz (2021): Statistik angewandt mit Excel – Datenanalyse ist (k)eine Kunst; 2. Auflage; Berlin: Springer Spektrum; e-ISBN: 978-3-662-62302-2;
<https://doi.org/10.1007/978-3-662-62302-2>

Lippert, Jan (2019): Mandantenfähigkeit mit PostgreSQL; Online:
<https://www.triology.de/blog/mandantenfaehigkeit-mit-postgresql> [letzter Aufruf: 09.02.2024]

Minijob-Zentrale (o.D. - a): Der gewerbliche Minijob: Abgaben und Steuern; Online:
https://www.minijob-zentrale.de/DE/fuer-gewerbetreibende/abgaben-und-steuern/abgaben-und-steuern_node.html#doc2a008d70-e046-422e-9866-9f09807a5c11bodyText1 [letzter Aufruf: 09.02.2024]

Minijob-Zentrale (o.D. - b): Rentenversicherungspflicht im Minijob; Online:

<https://www.minijob-zentrale.de/DE/die-minijobs/rentenversicherungspflicht/detailseite.html?nn=d3eac782-e1c0-4e7a-a5d6-99cc11a3e236> [letzter Aufruf: 09.02.2024]

Pandas (o.D.): Intro to pandas; Online:

https://pandas.pydata.org/docs/getting_started/index.html#intro-to-pandas [letzter Aufruf: 09.02.2024]

PostgreSQL (o.D. - a): What is PostgreSQL?; Online:

<https://www.postgresql.org/docs/current/intro-what-is.html> [letzter Aufruf: 09.02.2024]

PostgreSQL (o.D. - b): PL/pgSQL - Overview; Online:

<https://www.postgresql.org/docs/current/plpgsql-overview.html> [letzter Aufruf: 09.02.2024]

PostgreSQL (o.D. - c): Row Security Policies; Online:

<https://www.postgresql.org/docs/current/ddl-rowsecurity.html> [letzter Aufruf: 09.02.2024]

Psycopg2 (o.D.): PostgreSQL database adapter for Python; Online:

<https://www.psycopg.org/docs/> [letzter Aufruf: 09.02.2024]

Python (o.D. - a): unittest - Unit testing framework; Online:

<https://docs.python.org/3/library/unittest.html> [letzter Aufruf: 09.02.2024]

Python (o.D. - b): setUp(); Online:

<https://docs.python.org/3/library/unittest.html#unittest.TestCase.setUp> [letzter Aufruf: 09.02.2024]

Python (o.D. - c): tearDown(); Online:

<https://docs.python.org/3/library/unittest.html#unittest.TestCase.tearDown> [letzter Aufruf: 09.02.2024]

Python (o.D. - d): assertEquals(); Online:

<https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertEquals>
[letzter Aufruf: 09.02.2024]

Python (o.D. - e): assertTrue(); Online:

<https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertTrue>
[letzter Aufruf: 09.02.2024]

Python (o.D. - f): assertRaises(); Online:

<https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertRaises>
[letzter Aufruf: 09.02.2024]

Python (o.D. - g): assertIsNone(); Online:

<https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIsNone>
[letzter Aufruf: 09.02.2024]

SGB III §27 (1997): Arbeitsförderung; Online:

https://www.gesetze-im-internet.de/sgb_3/_27.html [letzter Aufruf: 09.02.2024]

Somasundar, Harish (2021): Database Multi tenancy; Online:

<https://medium.com/@harish.somasundar14/database-multi-tenancy-7c8dbe848d50> [letzter Aufruf: 09.02.2024]

Steiner, René (2021): Grundkurs Relationale Datenbanken – Einführung in die Praxis der Datenbankentwicklung für Ausbildung, Studium und IT-Beruf; 10. Auflage; Wiesbaden: Springer Vieweg; e-ISBN: 978-3-658-32834-4; <https://doi.org/10.1007/978-3-658-32834-4>

Tremp, Hansruedi (2022): Agile objektorientierte Anforderungsanalyse – Planen – Ermitteln - Analysieren – Modellieren – Dokumentieren - Prüfen; Springer Fachmedien Wiesbaden GmbH; e-ISBN: 978-3-658-37194-4; <https://doi.org/10.1007/978-3-658-37194-4>

Winter, Mario / Ekssir-Monfared, Mohsen / Sneed, Harry M. / Seidle, Richard / Borner, Lars (2013): Der Integrationstest – Von Entwurf und Architektur zur Komponenten- und Systemintegration; Carl Hanser Verlag München; e-ISBN: 978-3-446-42951-2

Protokoll ChatGPT

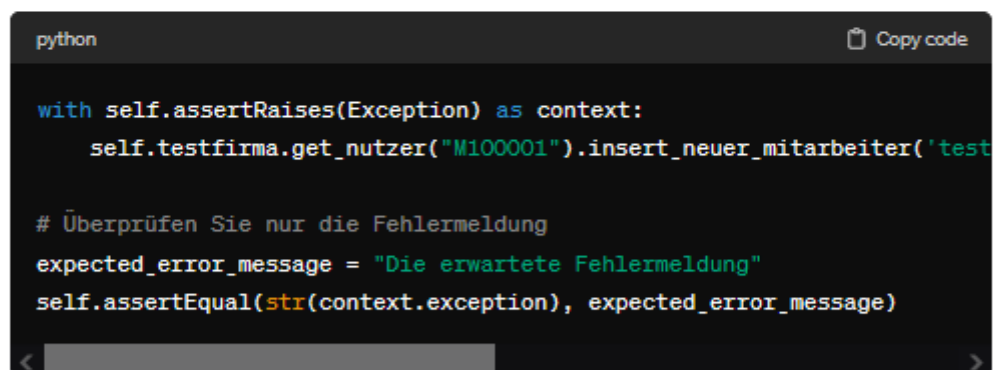
You

Ich möchte ein Unittest schreiben. Bei AssertRaises muss bisher die FEHLER- und die CONTEXT-Meldung verglichen werden. Ich möchte nur die Fehlermeldung vergleichen muessen. Wie geht das. Das ist der bisherige Code: with
self.assertRaises(Exception) as context:
 self.testfirma.get_nutzer("M100001"). \
 insert_neuer_mitarbeiter('testdaten_insert_mitarbeiter/Mitarbeiter.xlsx',
self.testschema)

ChatGPT

In Python können Sie die `assertRaises`-Methode mit einem Kontext-Manager verwenden, um auf eine bestimmte Fehlermeldung zu prüfen. Wenn Sie nur die Fehlermeldung überprüfen möchten und nicht den gesamten Kontext, können Sie das Attribut `exc_msg` des Kontext-Managers verwenden.

Hier ist ein Beispiel:



```
python Copy code  
  
with self.assertRaises(Exception) as context:  
    self.testfirma.get_nutzer("M100001").insert_neuer_mitarbeiter('test  
  
# Überprüfen Sie nur die Fehlermeldung  
expected_error_message = "Die erwartete Fehlermeldung"  
self.assertEqual(str(context.exception), expected_error_message)
```

Ersetzen Sie "Die erwartete Fehlermeldung" durch den tatsächlichen Text der Fehlermeldung, den Sie erwarten. Die `str(context.exception)` gibt den Text der Ausnahme zurück, den Sie dann mit der erwarteten Fehlermeldung vergleichen können.

Dadurch können Sie nur die Fehlermeldung vergleichen und müssen sich nicht um den gesamten Kontext kümmern.



You

Entschuldigung, ich habe nicht den ganzen Code mitgegeben. Bisher muss es so verglichen werden: `with self.assertRaises(Exception) as context:`

```
self.testfirma.get_nutzer("M100001"). \
    insert_neuer_mitarbeiter('testdaten_insert_mitarbeiter/Mitarbeiter.xlsx',
self.testschema)
```

```
self.assertEqual(str(context.exception), "FEHLER: Personalnummer 'M100002'
bereits vorhanden!\n"
```

```
        "CONTEXT: PL/pgSQL-Funktion
insert_tbl_mitarbeiter(integer,character"
        "varying,character varying,character varying,character
varying,date,"
        "date,character varying,character varying,character
varying,character "
        "varying,character varying,character varying,character
varying,date) "
```

"Zeile 43 bei RAISE\n") Ich möchte, dass die CONTEXT-Meldung nicht berücksichtigt werden muss. Wie geht das?



ChatGPT


Verstanden, Sie möchten nur die FEHLER-Meldung überprüfen und nicht den gesamten Kontext. Sie können dies erreichen, indem Sie nur den Anfang der Fehlermeldung vergleichen, anstatt die gesamte Zeichenkette. Hier ist ein Beispiel:

```
python Copy code

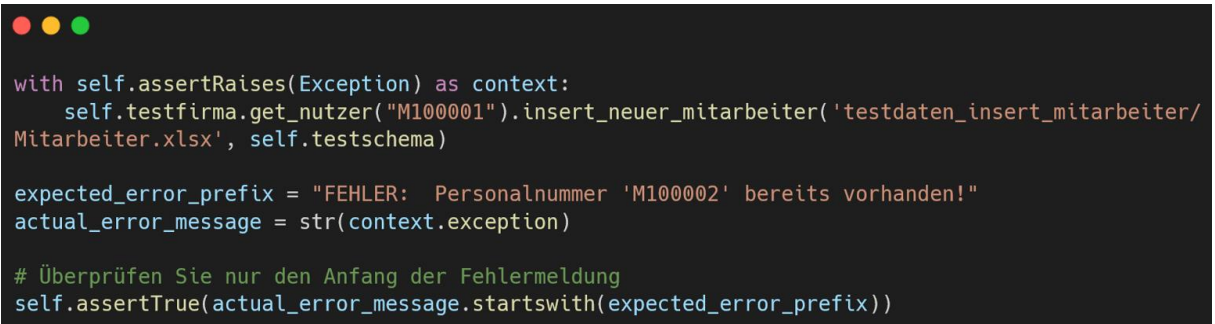
with self.assertRaises(Exception) as context:
    self.testfirma.get_nutzer("M100001").insert_neuer_mitarbeiter('test

expected_error_prefix = "FEHLER: Personalnummer 'M100002' bereits vorh
actual_error_message = str(context.exception)

# Überprüfen Sie nur den Anfang der Fehlermeldung
self.assertTrue(actual_error_message.startswith(expected_error_prefix))
```

In diesem Beispiel vergleicht die `assertTrue`-Aussage, ob die tatsächliche Fehlermeldung mit dem erwarteten Präfix beginnt. Dadurch müssen Sie nur sicherstellen, dass der relevante Teil der Fehlermeldung übereinstimmt, und Sie können die CONTEXT-Meldung ignorieren. 

Vollständiger von chatGPT erzeugter Code, welcher für die Arbeit verwendet wurde:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains Python code for testing an exception.

```
with self.assertRaises(Exception) as context:
    self.testfirma.get_nutzer("M100001").insert_neuer_mitarbeiter('testdaten_insert_mitarbeiter/
Mitarbeiter.xlsx', self.testschema)

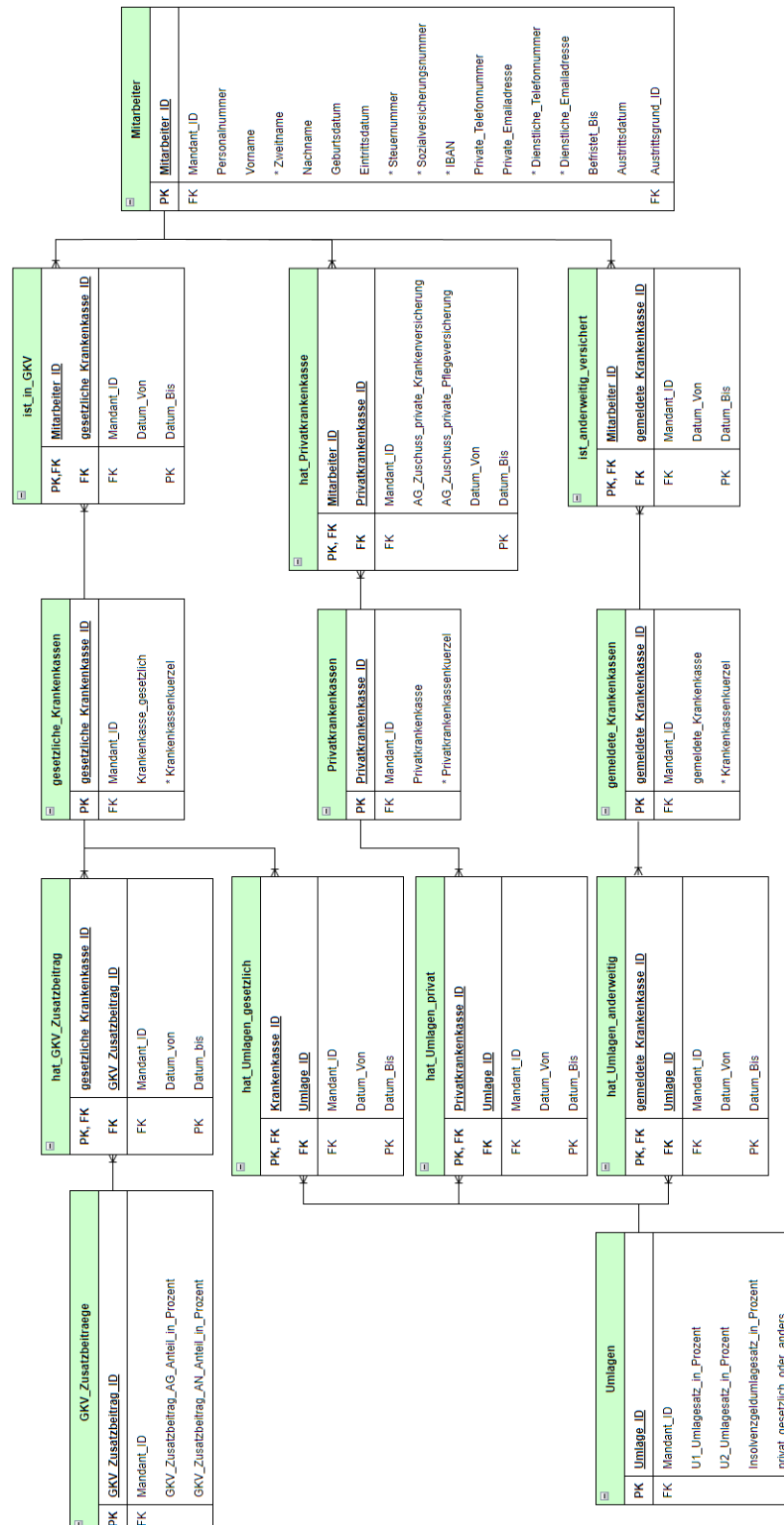
expected_error_prefix = "FEHLER: Personalnummer 'M100002' bereits vorhanden!"
actual_error_message = str(context.exception)

# Überprüfen Sie nur den Anfang der Fehlermeldung
self.assertTrue(actual_error_message.startswith(expected_error_prefix))
```

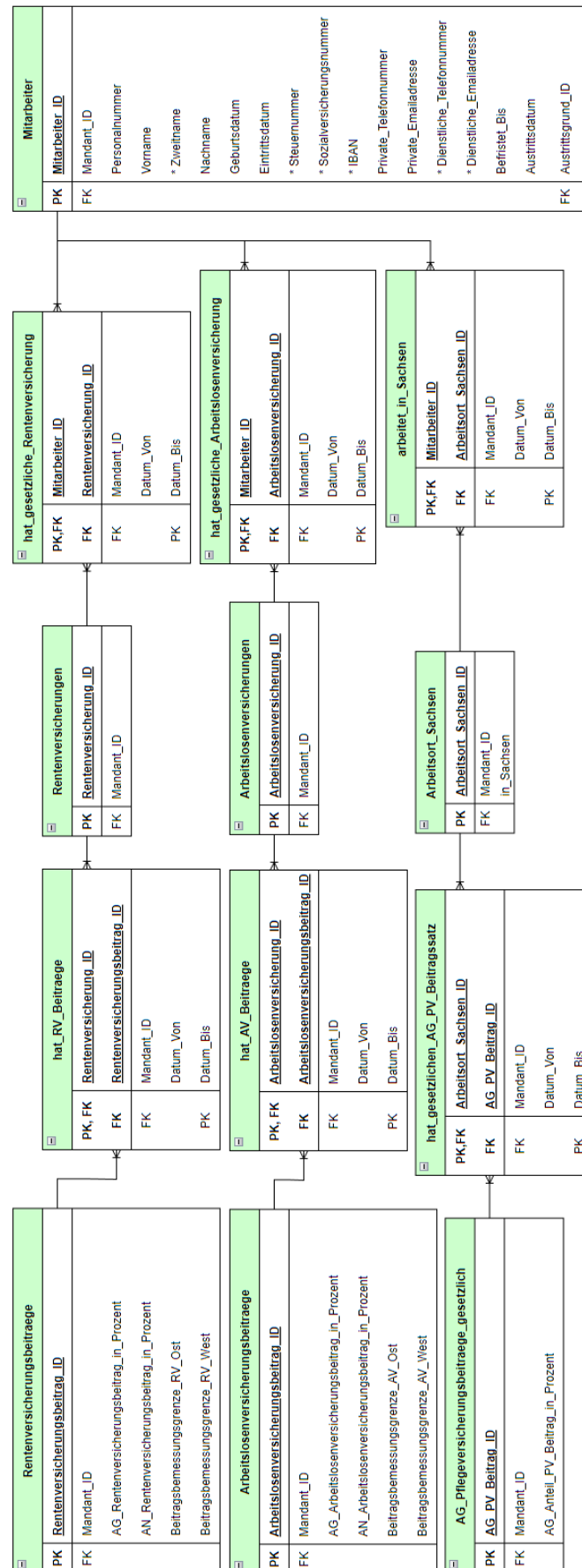
Abbildung 28: Verwendete Lösung von chatGPT, dargestellt mit carbon.now.sh

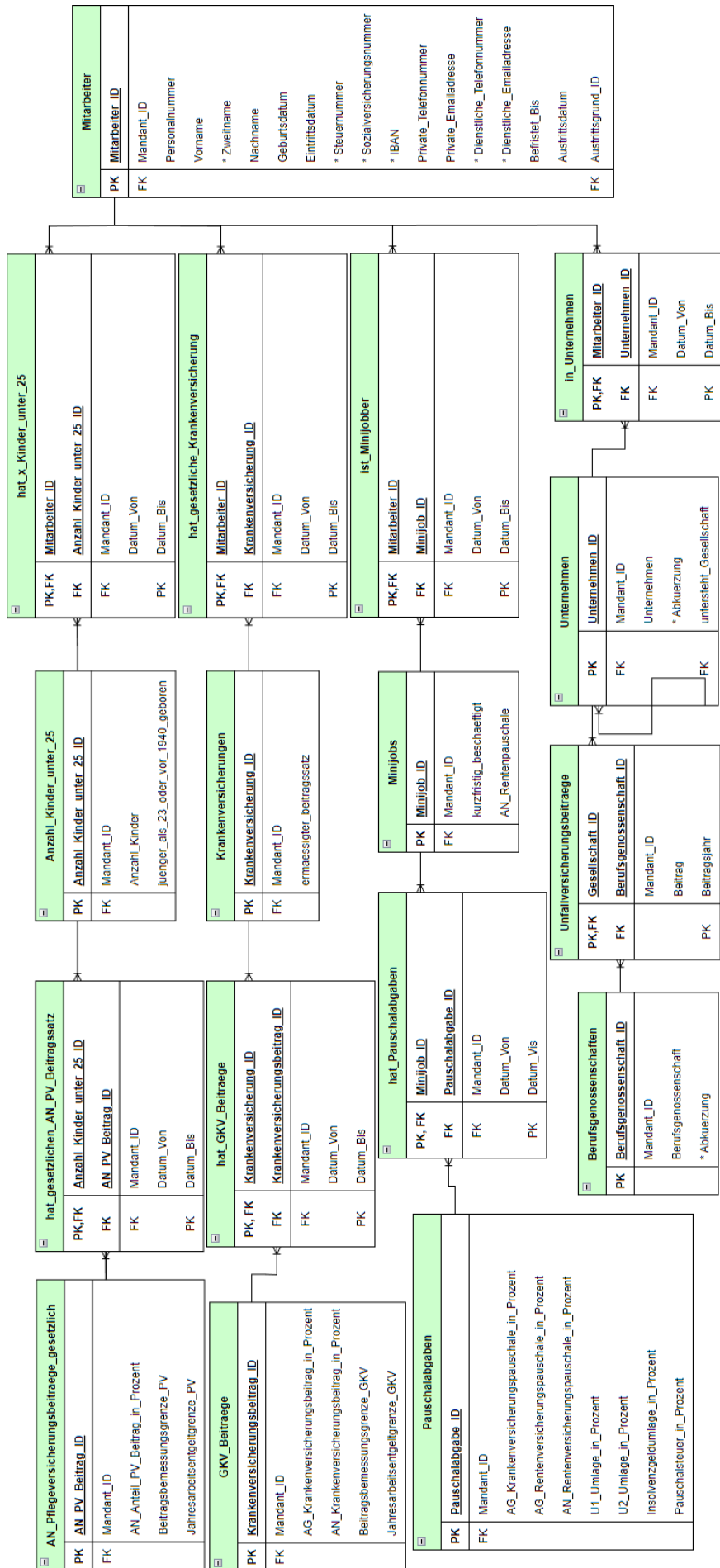
Die folgenden Abbildungen zeigen zusammen das komplette Datenmodell.

A.1 - Krankenkassen

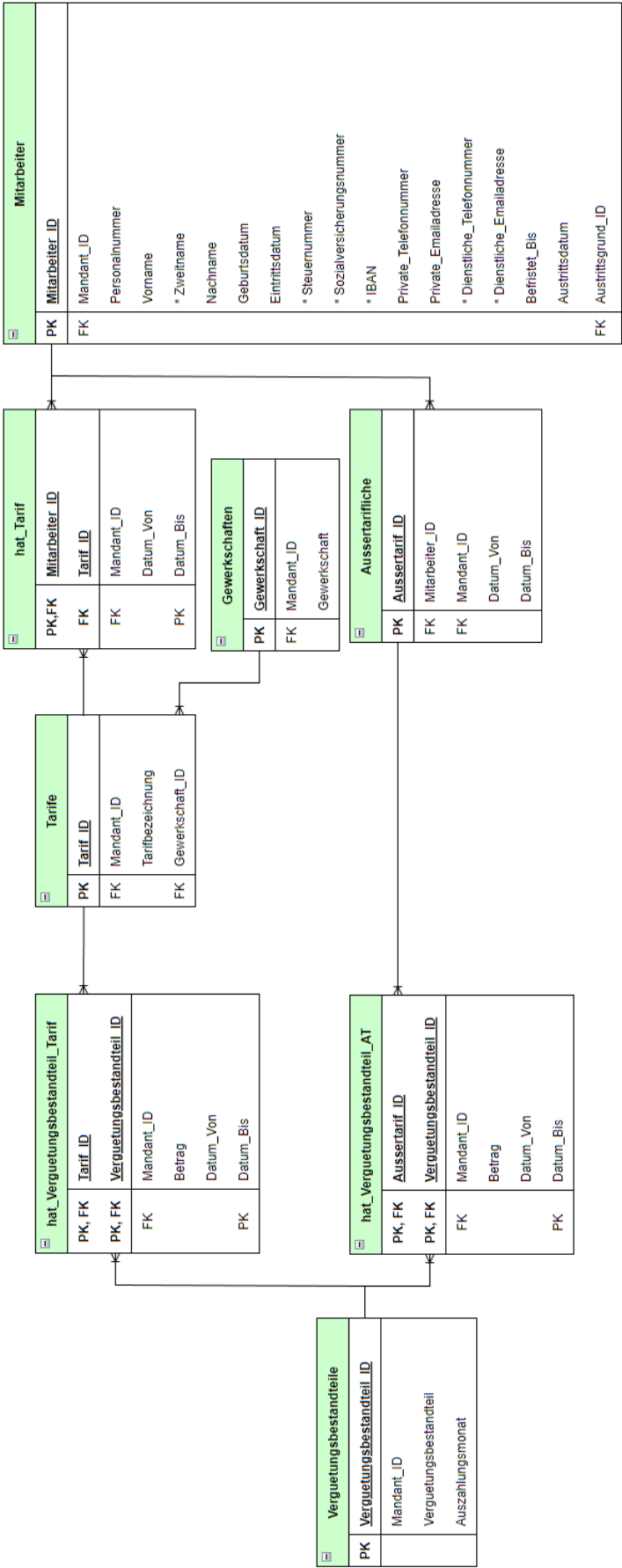


A.2 – Sonstige Sozialversicherungen

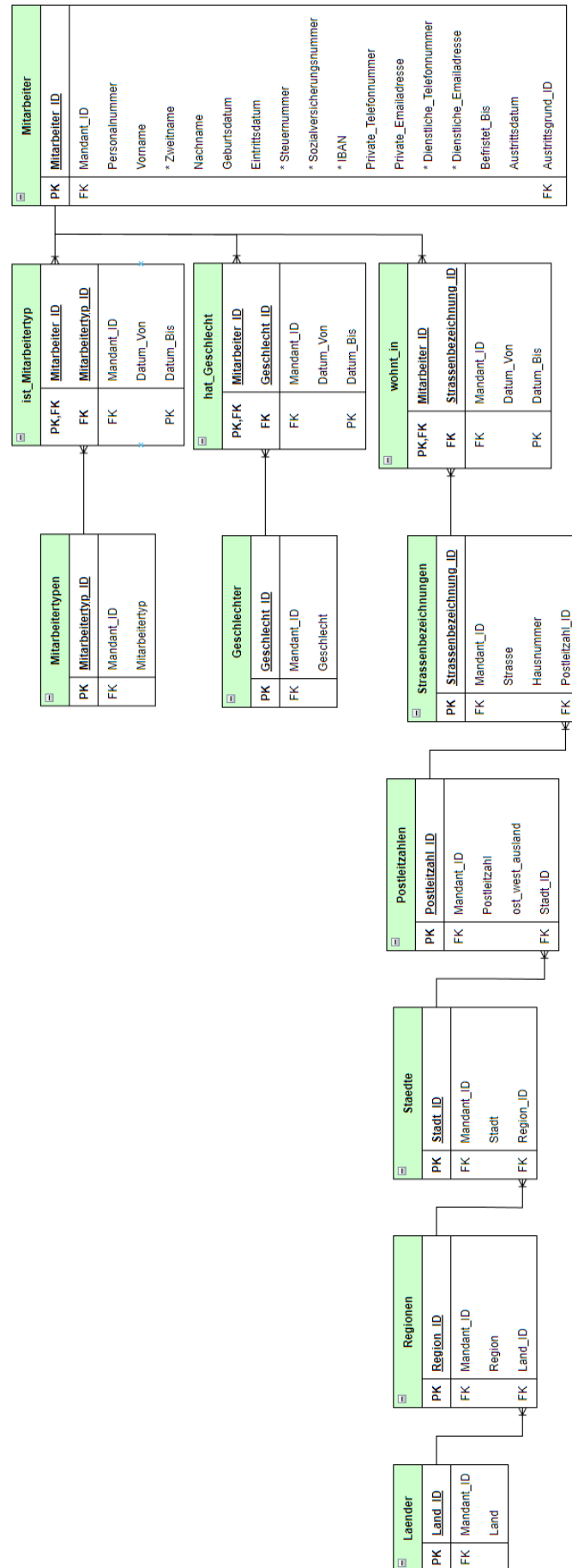


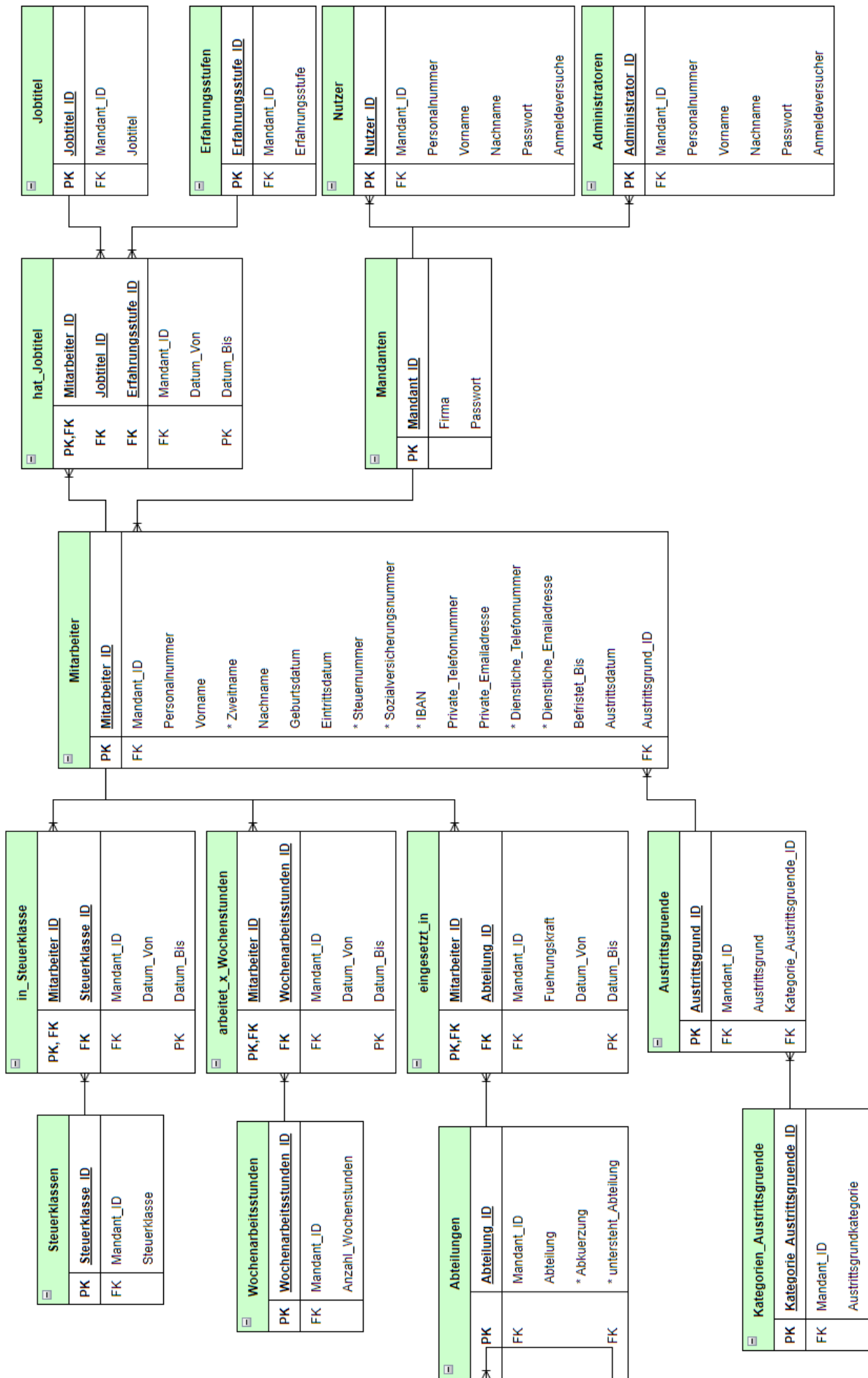


A.3 – Entgelt



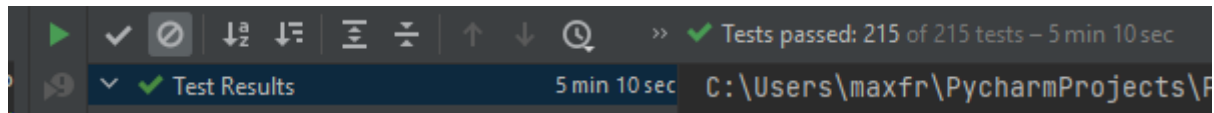
A.4 – Sonstiges Datenmodell





Anhang B – Screenshot: Ordnerstruktur

Anhang C – Screenshot: bestandene Tests



Eigenständigkeitserklärung

Ich erkläre hiermit, dass

- Ich die vorliegende wissenschaftliche Arbeit selbstständig und ohne unerlaubte Hilfe angefertigt habe,
- Ich andere als die angegebenen Quellen und Hilfsmittel nicht benutzt habe,
- Ich die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe,
- Die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfbehörde vorgelegen hat.

Berlin, 09.02.2023


Max Sven Freudenberg