technische universität dortmund

IT in Produktion und Logistik

Datenbankdesign für den internationalen Bildungsaustausch: Strategien und Umsetzung

Masterarbeit zur Erlangung des Grades M. Sc

Vorgelegt von:	Sami Mazeg
Matrikelnummer:	211177
Studiengang:	Wirtschaftsingenieurwesen

Ausgabedatum:	01.04.2024
Abgabedatum:	17.09.2024

Erstprüfer:	Dr. Anne Antonia Scheidler
Zweitprüfer:	Florian Hochkamp

Technische Universität Dortmund

Fakultät Maschinenbau

Fachgebiet IT in Produktion und Logistik

Inhaltsverzeichnis

Abbildungsverzeichnis I
Tabellenverzeichnis II
Algorithmenverzeichnis III
Abkürzungsverzeichnis IV
1 Einleitung 1
2 Methodische Grundlagen zur Entwicklung und Implementierung eines automatisierten Datenbanksystems 4
2.1 Grundlagen der Datenbanken 4
2.2 Relationales Datenmodell und Datenbankdesign-Prinzipien 8
2.3 Structured Query Language 12
2.4 Python zur Entwicklung einer datengetriebenen Applikation und Zugriffsberechtigungen in Datenbanken 21
3 Konzeption und Umsetzung des Datenbanksystems und Analyse der Automatisierungspotenziale 24
3.1 Funktionale Trennung des Datenbanksystems und Restrukturierung der Datentabellen 24
3.2 Erstellung und Umformung von Datenabfragen mithilfe von SQL 30
3.3 Darstellung der vorliegenden Prozesse und Erkennung von Automatisierungspotenzialen 37
4 Entwicklung der automatisierten Import- und Exportfunktionalitäten in Python 41
4.1 Entwicklung einer Benutzeroberfläche und des automatisierten Datenimports 41
4.2 Automatisierte Daten-Exporte und Dokumentenerstellungen 49
5 Implementierung und Validierung von Datenbankfunktionalitäten in bestehender Systemlandschaft 56
5.1 Implementierung eines Zugriffskonzeptes 56
5.2 Implementierung des Datenbanksystems und der Datenapplikation 58
5.3 Diskussion der Forschungsfragen und Fazit 60
6 Zusammenfassung und Ausblick 63
Literaturverzeichnis 65
Eidesstattliche Versicherung 73

Abbildungsverzeichnis

Abbildung 1: Vom Weltausschnitt zur Datenbank 5
Abbildung 2: Schalenmodell für die Funktionen eines DBMS 7
Abbildung 3: Namenskonvention der Backend-Tabellen 25
Abbildung 4: Auftrennung der ursprünglichen Tabelle „ERASMUS_Bewerbungen“ 25
Abbildung 5: Auftrennung der ursprünglichen Tabelle „ERASMUS_Incomings“ 26
Abbildung 6: Auftrennung der ursprünglichen Tabellen 27
Abbildung 7: Ursprüngliche Struktur der Stammdatentabelle der Outgoings 28
Abbildung 8: Restrukturierte Stammdatentabelle der Outgoings 29
Abbildung 9: Neue Bewerbungsdatentabelle der Outgoings 29
Abbildung 10: Fremdschlüsselverknüpfung von Tabellen 30
Abbildung 11: SQL-Abfragen mit zugehöriger Funktionsbeschreibung 31
Abbildung 12: Visualisierung der Verknüpfung der Tabellen 33
Abbildung 13: Visualisierung der Verknüpfung der Tabellen 34
Abbildung 14: Prozessdiagramm des Nominierungsprozesses 39
Abbildung 15: Prozessdiagramm des Anerkennungsprozesses 40
Abbildung 16: Root-Ebene des ERASMUS-Datenverarbeiters 42
Abbildung 17: Current-Ebene des Datenimports 43
Abbildung 18: Current-Ebene des Datenimports nach Auswahl einer Excel-Datei 44
Abbildung 19: Current-Ebene des Datenimports bei fehlenden Stammdaten 48
Abbildung 20: Current-Ebene des Datenimports bei einem Datentypenkonflikt 49
Abbildung 21: Erste current-Ebene der ToR-Erstellung 50
Abbildung 22: Zweite current-Ebene der ToR-Erstellung 50
Abbildung 23: Zuordnung ToR Formfelder 51
Abbildung 24: Erste current-Ebene zur Erstellung des Nominierungsbescheides 53
Abbildung 25: Zweite current-Ebene zur Erstellung des Nominierungsbescheides 53
Abbildung 26: Zweite current-Ebene zur Erstellung der Anerkennungsvereinbarung 54

Tabellenverzeichnis

Tabelle 1: Kernbegriffe innerhalb relationaler Datenbanken 9
Tabelle 2: Beispieltabelle für Produkte und Umsätze 10
Tabelle 3: Beispieltabelle für persönliche Informationen 10
Tabelle 4: Beispieltabelle für Produktmarken 11
Tabelle 5: Auflistung Vereinigungsoperatoren in SQL mit Algebra-Formel 16
Tabelle 6: Zugriffsmatrix für die Backend- und Frontend-Datenbanken 57

Algorithmenverzeichnis

Algorithmus 1: Syntax eines SELECT-Befehls 13
Algorithmus 2: SELECT-Befehl, um alle Einträge einer Tabelle auszugeben 14
Algorithmus 3: SELECT-Befehl um Daten einzelner Spalten auszugeben....14
Algorithmus 4: Formatierter SELECT-Befehl 14
Algorithmus 5: Syntax des SELECT-Befehls 14
Algorithmus 6: Beispiel einer verschachtelten SQL-Abfrage 15
Algorithmus 7: Beispiel eines WHERE-Befehls mit logischen Operatoren 15
Algorithmus 8: Beispiel einer Vereinigung mit dem UNION-Befehl....16
Algorithmus 9: Beispiel einer Vereinigung mit dem EXCEPT-Befehl....17
Algorithmus 10: Beispiel einer Vereinigung mit dem INTERSECT-Befehl....17
Algorithmus 11: Syntax des JOIN-Befehls....17
Algorithmus 12: Beispiel eines INNER JOIN-Befehls 18
Algorithmus 13: Beispiel eines LEFT JOIN-Befehls....18
Algorithmus 14: Syntax des ORDER BY-Befehls 18
Algorithmus 15: Beispiel einer SQL-Abfrage mit COUNT-Funktion 19
Algorithmus 16: Beispiel eines INSERT INTO-Befehls....20
Algorithmus 17: Beispiel eines UPDATE-Befehls 20
Algorithmus 18: Beispiel eines verschachtelten UPDATE-Befehls....20
Algorithmus 19: Beispiel eines DELETE-Befehls....21
Algorithmus 20: Erstellung einer Benutzeroberfläche mit Tkinter 22
Algorithmus 21: SELECT-Befehl „Outgoings alle Prioritäten und Bewerbungsdaten“ 32
Algorithmus 22: FROM-Befehl „Outgoings alle Prioritäten und Bewerbungsdaten“....32
Algorithmus 23: Berechnung der ECTS-Summe der Outgoings....33
Algorithmus 24: Berechnung der tatsächlich anerkannten ECTS-Summe der Outgoings...33
Algorithmus 25: Berechnung der Quote anerkannter und absolvierter Fächer....34
Algorithmus 26: Auswahl der Nominierungsdeadline....35
Algorithmus 27: Bedingung des „INNER-JOINS“ zur Auswahl der Nominierungsdeadline .35
Algorithmus 28: Berechnung der freien Plätze an einer Partneruniversität 36
Algorithmus 29: SQL-Bedingung der Summenfunktion 36
Algorithmus 30: Programmcode zur Erstellung eines Buttons für die Import-Applikation....44
Algorithmus 31: Programmcode für die back_to_root-Funktion....45
Algorithmus 32: Programmcode zur generischen Erstellung der INSERT-Skripte in SQL ..46
Algorithmus 33: Beispiel einer Ausgabe der INSERT-Skripte....47

Abkürzungsverzeichnis

SQL	Structured Query Language
DBMS	Datenbank-Verwaltungssystem
GUI	Grafische Benutzeroberfläche
ToR	Transcript of Records
RTDBS	Echtzeit-Datenbanksystem
DDL	Data-Retrieval-Language

1 Einleitung

Die rasante Entwicklung der Datenbanktechnologien hat zu neuen Richtungen und Herausforderungen geführt, die sich auf die Gesellschaft und die organisatorischen Abläufe auswirken (Rivero et al. 2005) . Schon seit den späten 1960er Jahren etablieren sich Datenbank-Verwaltungssysteme (DBMS) als unverzichtbare Mittler zwischen den Programmen und den Daten, die sie verarbeiten (Unterstein und Matthiessen 2012). Diese Evolution wird durch das unaufhaltsame Wachstum von Datenvolumen in fast allen Lebens- und Geschäftsbereichen vorangetrieben, was Datenbanken unerlässlich macht (Steiner 2017). Die Qualität der Informationen hängt von der Verfügbarkeit, Korrektheit und Vollständigkeit der Daten ab, und die Integration von Daten ist für die Erfüllung organisatorischer Aufgaben entscheidend (Meier und Kaufmann 2019). Dies erfordert ein effektives Datenmanagement auf Führungsebene, das neben der technischen Infrastruktur auch die Planung und Gestaltung von Datenflüssen umfasst (Meier und Kaufmann 2019). Akademische Einrichtungen stehen bei der Datenbankverwaltung vor zahlreichen Herausforderungen. Begrenzte Ressourcen, einschließlich eines unzureichenden Budgets und Personals, sind ein großes Hindernis für die Entwicklung und den Einsatz von institutionellen Verwaltungssystemen (Joo et al. 2019). Renommierte Universitäten profitieren dabei oft von besserem Zugang zu Technologie, auch wenn die Implementierung zeitaufwendig ist (Kustitskaya et al. 2023). Probleme mit der Datenqualität sind weit verbreitet, wobei die Datenverwaltung für die Aufrechterhaltung genauer und aktueller Informationen entscheidend ist (Astuti et al. 2024). Die schiere Menge an Daten und die Sensibilität bestimmter Informationen stellen erhebliche Herausforderungen dar (Joo et al. 2019). Die Hochschulen kämpfen mit der Umwandlung traditioneller Lehrmethoden in Onlineformate, die durch E-Learning-Plattformen unterstützt werden und erhebliche Mengen an Studierendendaten erzeugen (Kustitskaya et al. 2023). Technische Probleme wie Netzwerkprobleme, geringe Bandbreite und unzureichende Computersysteme behindern die Zugänglichkeit und Nutzung von Datenbanken zusätzlich (Ivongbe et al. 2021). Darüber hinaus erschweren mangelndes Bewusstsein, unzureichende Wartung und die Fähigkeiten des Personals, effektiv nach Informationen zu suchen, die Datenbankverwaltung (Ivongbe et al. 2021). Die Bewältigung dieser Herausforderungen ist von entscheidender Bedeutung für die Umsetzung und Verbesserung von Bildungsprozessen in akademischen Einrichtungen.

In Anbetracht dieser Entwicklungen wird deutlich, dass ein durchdachtes und automatisiertes Datenbanksystem in Bildungseinrichtungen zunehmend unverzichtbar ist. Ein solches System ermöglicht nicht nur eine effiziente Datenverwaltung, sondern gewährleistet auch die Sicherheit und Integrität der Bildungsdaten im globalen Kontext. In dieser Arbeit ist daher ein Datenbanksystem für den internationalen Bildungsaustausch zu entwickeln und umzusetzen, das diesen Anforderungen gerecht wird. Die verfolgte Strategie zielt darauf ab, sowohl die Effizienz der Datenverwaltung und Prozesse im Zusammenhang mit den Daten zu maximieren als auch den sicheren Umgang mit den sensiblen Bildungsdaten zu gewährleisten. Die vorliegende Arbeit basiert auf der Prämisse, dass kein grundlegend neues DBMS einzuführen ist, sondern dass die bestehenden Strukturen und Systeme innerhalb der akademischen Einrichtung genutzt werden, wobei Microsoft Access als etabliertes DBMS beibehalten wird. Allerdings ist die Struktur der bestehenden Tabellen von Grund auf neu zu definieren, um eine stabile und konsistente Datenbasis zu schaffen. Dies umfasst unter anderem die Einführung standardisierter Namenskonventionen, die sorgfältige Festlegung von Identifizierungsmerkmalen sowie die Trennung der Tabellen nach sinnvollen Gesichtspunkten. Zudem ist die Datenbank in eine Backend-Datenbank und eine Frontend-Datenbank aufzuteilen, die für Abfragen und andere benutzerbezogene Interaktionen optimiert ist. Durch diese Maßnahmen sollen die Datenbankstruktur deutlich stabiler gestaltet und die Integrität der Daten nachhaltig gestärkt werden. Gleichzeitig soll die Effizienz der Verwaltungsprozesse durch die Integration von Python-Code und einer grafischen Benutzeroberfläche (GUI) erhöht werden. Dies soll eine Automatisierung des Datenimports und -exports sowie die automatisierte Erstellung relevanter Dokumente für

die institutionellen Abläufe ermöglichen. Der Einsatz von unterschiedlichen Programmiersprachen und Software-Bibliotheken zur Modernisierung und Effizienzsteigerung von Datenbanksystemen und datengetriebenen Prozessen an Hochschulen bietet die Möglichkeit, die Nachteile, die durch begrenzte finanzielle Mittel, Zeitressourcen und Personal entstehen, zumindest teilweise auszugleichen. Dadurch können Hochschulen wettbewerbsfähiger werden im Vergleich zu Unternehmen und Universitäten, die bereits hochoptimierte Datenbanksysteme nutzen. Aljarallah & Dutta (2022) schlagen einen integrierten Rahmen zur Automatisierung der Dokumentenvorbereitung für die akademische Akkreditierung an saudi-arabischen Universitäten vor und betonen die Notwendigkeit, den manuellen Aufwand zu verringern (Aljarallah und Dutta 2022). Nakkeeran et al. (2023) stellen eine Python-Anwendung vor, die unterschiedliche Python-Bibliotheken verwendet, um die Sitzordnung für Prüfungen zu automatisieren und so den Zeit- und Arbeitsaufwand für Prüfungsabteilungen erheblich zu reduzieren (Nakkeeran et al. 2023). Diese Studien zeigen das Potenzial der softwarebasierten Automatisierung im akademischen Umfeld und konzentrieren sich auf die Verbesserung der Effizienz und Genauigkeit, sowie die Verringerung des manuellen Arbeitsaufwandes. Die Implementierungen umfassen in der Regel Datenbankintegration, Dokumentenverarbeitung und benutzerfreundliche Schnittstellen, um die Akzeptanz zu erleichtern und die Verwaltungsprozesse zu rationalisieren (Aljarallah und Dutta 2022; Nakkeeran et al. 2023).

Angesichts der beschriebenen Herausforderungen und der zunehmenden Komplexität in der Verwaltung von Bildungsdaten stellt sich die zentrale Frage, wie bestehende Altsysteme in akademischen Einrichtungen durch gezielte Verbesserungen und den Einsatz moderner Technologien optimiert werden können. Vor diesem Hintergrund wird in dieser Arbeit untersucht, ob und in welchem Maße die grundlegende Neustrukturierung einer bestehenden Datenbank, kombiniert mit der Implementierung einer Software-Applikation zur datengetriebenen Automatisierung von Hochschulprozessen, zu einer signifikanten Effizienzsteigerung und Stabilisierung der Projektprozesse beitragen kann. Um diese übergeordnete Fragestellung zu beantworten, wird die Arbeit durch mehrere spezifische Forschungsfragen strukturiert. Zunächst stellt sich die Frage, welche konkreten Verbesserungen in der Datenqualität und -konsistenz durch die Neustrukturierung der Tabellen, die Festlegung von Namenskonventionen und die Implementierung von Primärschlüsseln erreicht werden können. Weiterhin stellt sich die Frage, inwieweit die bestehende Infrastruktur, insbesondere Microsoft Access, durch die Integration von Python-Code und einer grafischen Benutzeroberfläche effizient erweitert und optimiert werden kann. Zusätzlich wird geprüft, wie die Aufteilung der Datenbank in Backend- und Frontend-Komponenten zur Optimierung der Abfragen und benutzerbezogenen Interaktionen beiträgt. Ein weiterer Schwerpunkt liegt auf der Analyse der Auswirkungen, die die Automatisierung von Datenimporten und -exporten und der Dokumentenerstellung auf die Reduzierung des manuellen Aufwands und die Genauigkeit der administrativen Prozesse hat. Schließlich wird untersucht, wie die Implementierung datengetriebener Prozesse und Automatisierungen zur Verringerung der Abhängigkeit von finanziellen Mitteln, Zeitressourcen und Personal beitragen kann, um die Wettbewerbsfähigkeit der Hochschule im Vergleich zu Einrichtungen mit hochoptimierten Datenbanksystemen zu verbessern. Diese Forschungsfragen bilden den Rahmen für die anschließende Analyse und dienen als Grundlage, um den Erfolg der in dieser Arbeit entwickelten und implementierten Maßnahmen zu bewerten.

Zur Behandlung der Forschungsfragen werden die Grundlagen der Datenbanksysteme erläutert, wobei ein besonderes Augenmerk auf das relationale Datenmodell und die Prinzipien des Datenbankdesigns gelegt wird. Es folgt eine detaillierte Betrachtung der Structured Query Language (SQL) als zentrales Werkzeug zur Datenmanipulation und -abfrage. Zusätzlich wird die Verwendung von Python zur Entwicklung automatisierter Prozesse, einschließlich des Datenimports und -exports sowie des Befüllens von Vorlagen, beleuchtet. Die Benutzerverwaltung und die verschiedenen Zugriffsebenen eines Datenbanksystems runden diesen methodischen Teil ab. Im Anschluss daran erfolgt die Konzeption und Umsetzung des Datenbanksystems, wobei insbesondere die funktionale Trennung des Datenmodells und der Datentabellen sowie die Erstellung von Datenabfragen mithilfe von SQL im Fokus stehen. Es wird auch eine Analyse der bestehenden Prozesse mithilfe von Prozessdiagrammen durchgeführt, um Automati-

sierungspotenziale zu erkennen und zu nutzen. Daraufhin wird die Entwicklung der automatisierten Import- und Exportfunktionalitäten in Python beschrieben. Dies umfasst sowohl die Entwicklung einer GUI als auch die Implementierung automatisierter Datenimporte und Exporte sowie die automatisierte Erstellung relevanter Dokumente im Kontext der Hochschulbildung. In der folgenden Phase wird das entwickelte Datenbanksystem in die bestehende Systemlandschaft implementiert und validiert. Hierzu werden die Erstellung und Umsetzung eines Zugriffskonzepts sowie die Validierung des Systems und des automatisierten Datenverkehrs behandelt. Eine Diskussion und Interpretation der Ergebnisse in Bezug auf die einzelnen Forschungsfragen schließen diesen Abschnitt ab. Zum Abschluss der Arbeit erfolgen eine Zusammenfassung der wesentlichen Erkenntnisse und ein Ausblick auf die zukünftigen Entwicklungen des Datenbanksystems. Es werden mögliche Erweiterungen des Systems und eine Vision eines vollautomatisierten Datenbanksystems skizziert.

2 Methodische Grundlagen zur Entwicklung und Implementierung eines automatisierten Datenbanksystems

In diesem Kapitel werden die methodischen Grundlagen zur Entwicklung und Implementierung eines automatisierten Datenbanksystems detailliert erläutert. Zunächst werden die grundlegenden Konzepte von Datenbanksystemen vorgestellt. Darauf aufbauend werden das relationale Datenmodell sowie die Prinzipien des Datenbankdesigns näher betrachtet. Im Anschluss wird SQL als zentrales Werkzeug zur Datenmanipulation und -abfrage eingeführt. Ein weiterer Schwerpunkt wird auf die Anwendung von Python zur Entwicklung automatisierter Prozesse, einschließlich des Datenimports und -exports und des Befüllens von Dokumentenvorlagen gelegt. Abschließend werden die Benutzerverwaltung und die verschiedenen Zugriffsebenen eines Datenbanksystems thematisiert, um die Sicherheit und Effizienz des Systems zu gewährleisten.

2.1 Grundlagen der Datenbanken

Die Entwicklung von Datenbanksystemen wurde durch die Notwendigkeit vorangetrieben, immer größere und komplexere Datenmengen effizient zu verarbeiten. Frühe Entwicklungen konzentrierten sich auf die Integration der Datenverarbeitung und die Einführung von Konzepten zur Datenunabhängigkeit (Remus 1976). Mit wachsenden Datenmengen kamen neue Ansätze auf, darunter verteilte und föderierte Datenbanken, NoSQL-Systeme und In-Memory-Datenbanken (Böhm et al. 2015). Die Ausweitung computergestützter technischer Anwendungen hat zu einem raschen Wachstum der Datenmengen geführt, was die Bedeutung von Datenbanksystemen in zahlreichen Bereichen unterstreicht (Mitschang 1984). Die laufende Debatte über Big Data dreht sich darum, ob es sich um eine revolutionäre Veränderung oder eine natürliche Entwicklung in der Datenverarbeitung handelt, wobei sich die Diskussionen auf die Bedeutung von Volumen, Vielfalt, Geschwindigkeit, Wahrhaftigkeit und Wert im modernen Datenmanagement konzentrieren (Böhm et al. 2015). Neben der steigenden Datenmenge in Datenbanken erhöht sich auch die Komplexität der Datenbanken und die zeitliche Dringlichkeit der Abfragen. Echtzeit-Datenbanksysteme (RTDBS) haben sich als entscheidende Integration von herkömmlicher Datenbankverwaltung und Echtzeit-Computing herauskristallisiert, um den wachsenden Anforderungen komplexer Anwendungen mit zeitkritischen Daten gerecht zu werden (Kao und Garcia-Molina 1994). Diese Systeme sind für die Verarbeitung von Arbeitslasten mit sich ständig ändernden Zuständen und Transaktionen konzipiert, die Fristen oder zeitlichen Beschränkungen unterliegen (Lindström 2007). Zu den RTDBS-Anwendungen gehören beispielsweise der Aktienhandel, die Navigation und die computerintegrierte Fertigung (Lam und Kuo 2013). Der Bereich der RTDBS-Forschung, der in den späten 1980er Jahren begann, entwickelt sich ständig weiter und befasst sich mit Herausforderungen in Bereichen wie eingebetteten Systemen und Multidatenbankumgebungen (Ramamritham 1996). RTDBS müssen ACID-Eigenschaften beibehalten und gleichzeitig Echtzeitanforderungen erfüllen, was sie zu einem einzigartigen und wichtigen Forschungsbereich in der Datenbankforschung macht (Ramamritham 1996). Mehrere dieser Attribute und Eigenschaften lassen sich auch einem relationalen Datenbanksystem zuschreiben, welches innerhalb dieser Ausarbeitung näher betrachtet wird. Auf die einzelnen Attribute und Design-Prinzipien, welche bei der Modellierung eines Datenbanksystems beachtet werden müssen, wird in Abschnitt 2.2 näher eingegangen.

Die Modellierung des Datenmodells bildet die Grundlage für die Entwicklung eines Datenbanksystems. „Das „Abbild der Realität“ wird mit Hilfe eines Datenmodells gewonnen. D.h. der Ersteller der Datenbank analysiert den Weltausschnitt und erstellt - mit Hilfe eines theoretischen Instrumentariums - ein Datenmodell. In diesem sollte die Semantik des Anwendungsbereichs (die (benötigten) Strukturen, Regeln, Fakten, usw.) erfasst sein“ (Staud 2005, S. 3). Bei der Modellierung eines Systems können zwei verschiedene Ansätze gewählt werden, den der prozessorientierten- und der datenorientierten-Modellierung. Im Umgang mit Daten wird eine Kombination beider Vorgehensweisen empfohlen, da ein Informationssystem sowohl die dynamischen Prozesse unterstützen als auch die entsprechenden statischen Daten enthalten

muss, um ordnungsgemäß zu funktionieren (Brock 2023). Künzle & Reichert zeigen die Grenzen traditioneller Workflow-Management-Systeme bei der Handhabung datengesteuerter Prozesse auf und betonen die Notwendigkeit einer integrierten Sicht auf Prozesse und Daten (Künzle und Reichert 2009). Bauer schlägt eine Vormodellierung der erwarteten Flexibilität zur Erstellungszeit vor, um den Laufzeitaufwand zu reduzieren und die Sicherheit in prozessgesteuerten Informationssystemen zu erhöhen (Bauer 2017). Oberweis (2001) stellt einen wissensbasierten Ansatz für die Gestaltung von Customer-Relationship-Management-Systemen vor, der Geschäftsprozessmanagement- und Wissensmanagementkonzepte integriert (Oberweis et al. 2001). Knackstedt & Dahlke plädieren für eine systematische Analyse und optimale Gestaltung von Geschäftsprozessen in CRM-Systemen und schlagen den Einsatz von Informationsmodellen zur Konzeptualisierung der Kundenintegration vor (Knackstedt und Dahlke 2002). Gemeinsam betonen diese Arbeiten die Wichtigkeit, Prozess- und Datenorientierung auszubalancieren, Flexibilität einzubeziehen und Kundenanforderungen bei der Systemmodellierung und -entwicklung zu berücksichtigen. Auf Basis dessen wird eine nachvollziehbare, modulare, flexible, deklarative, transparente Entwicklung bevorzugt, welche dennoch eine klar formulierte Anforderung des jeweiligen Kunden voraussetzt (Brock 2023).

Die Erstellung eines Datenmodells ist theoriespezifisch was bedeutet, dass eine Datenbanktheorie die Grundlage der Modellierung bildet, beispielsweise die relationale Datenbanktheorie, welche in dieser Ausarbeitung beleuchtet wird (Staud 2005). In Abbildung 1 ist der Transformationsprozess von einem Weltausschnitt zu einer Datenbank dargestellt. Hier ist zu erkennen, dass der Weltausschnitt durch die Modellierungstheorie und im Folgenden die relationale Datenbanktheorie in ein Datenmodell überführt wird. Dieses wird wiederum durch ein Datenbanksystem, auf welches noch genauer eingegangen wird in eine Datenbank umgesetzt.

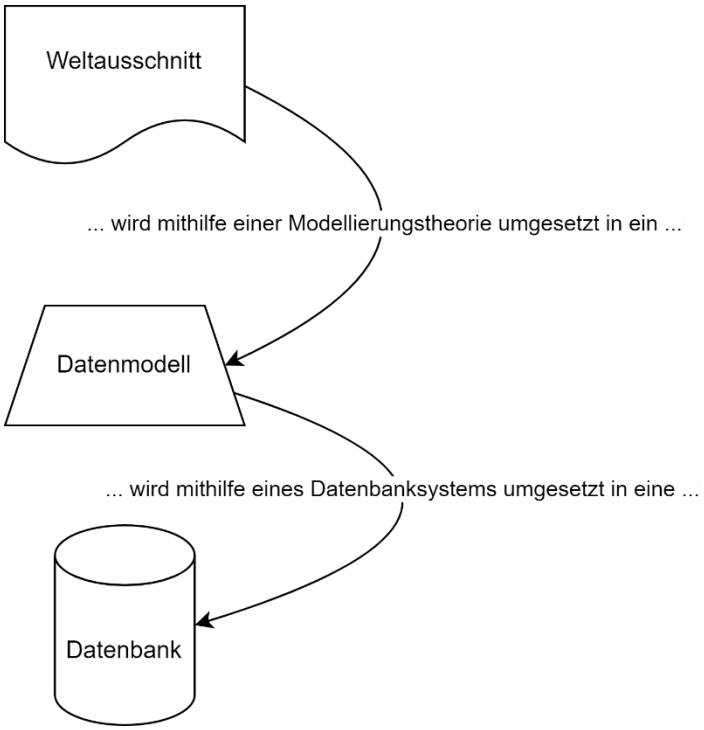


Abbildung 1: Vom Weltausschnitt zur Datenbank (in Anlehnung an Staud 2005, S. 18)

Datenbankmanagementsysteme (DBMS) sind Softwarelösungen, die unabhängig von Anwendungen Daten beschreiben, speichern und abfragen (Meier und Kaufmann 2019). Das DBMS bildet das zentrale Element der Datenbank und umfasst wesentliche Funktionen, welche für

den Umgang mit den Daten benötigt werden (Steiner 2017). Sie bieten eine logische Schnittstelle für Benutzer und Anwendungen zur Interaktion mit gespeicherten Daten und setzen logische Operationen in physische Datenzugriffe um (Dippold et al. 2005). Moderne DBMS-Architekturen unterstützen große Datenmengen, gleichzeitige Lese- und Schreiboperationen und hohe Datensicherheit (Dadam 1986). Die Forschung in diesem Bereich umfasst verteilte Datenbanken, Mehrrechnersysteme und fehlertolerante Designs (Dadam 1986). Fortgeschrittene Speichersysteme, wie das Darmstädter Datenbankkernsystem, erweitern die Dateiverwaltung um Datenbankfunktionalitäten, einschließlich Transaktionsmanagement und optionaler Versionierung (Deppisch et al. 1985). Diese Systeme ermöglichen einen effizienten Zugriff auf einzelne Daten und Datensätze, wobei Konzepte wie offene oder verschachtelte Transaktionen zur Verbesserung der Leistung und Flexibilität eingesetzt werden (Deppisch et al. 1985). Jedes DBMS enthält eine Speicher- und eine Verwaltungskomponente (Mullins 2007). Die Speicherkomponente umfasst alle in organisierter Form gespeicherten Daten sowie deren Beschreibung (Bachman et al. 1969). Die Verwaltungskomponente enthält eine Abfrage- und Datenmanipulationssprache zur Auswertung und Bearbeitung der Daten und Informationen (Mullins 2007). Diese Komponente dient nicht nur als Benutzerschnittstelle, sondern verwaltet auch Zugriffs- und Bearbeitungsberechtigungen für Benutzer (Meier und Kaufmann 2019).

Zu den geforderten Funktionen eines DBMS gehören unter anderem die Unterstützung eines Datenmodells, die Bereitstellung einer formalen Sprache, das Management von Transaktionen, die Zugangskontrolle sowie die Sicherstellung der semantischen Integrität (Thuraisingham 2000; Fong und Kimbleton 1980). DBMS benötigen mehrere Schlüsseleigenschaften, um Datenqualität und -sicherheit zu gewährleisten. Neben der Unterstützung des Modells für die zu verwaltenden Daten, ist vor allem die Unterstützung einer formalen Sprache wesentlich. Diese ermöglicht es Nutzern, die Datenstruktur zu definieren, auf die Daten zuzugreifen und die Daten zu verarbeiten. Ein heutiger Standard bildet dabei die Zugriffssprache SQL (Meier und Kaufmann 2019). Zudem ist die Unterstützung einer formalen Sprache unerlässlich, um semantische Integritätsbeschränkungen auszudrücken, die Eigenschaften und Beziehungen zwischen Datenobjekten beschreiben (Hammer und McLeod 1975). Die semantische Integrität gewährleistet, dass die Daten logisch korrekt bleiben und eine gültige Abstraktion der realen Welt darstellen (Fong und Kimbleton 1980). Zudem muss ein DBMS auch in kritischen Situationen, wie einem Systemabsturz in der Lage sein die semantische Integrität aufrechtzuerhalten und die Daten wiederherzustellen ohne diese vollständig zu verlieren (Staud 2005). Die Aspekte Transaktionsmanagement und Zugangskontrolle müssen auf einer Ebene betrachtet werden, da sie stark miteinander korrelieren. Sie sind entscheidend für die Regelung des Benutzerzugriffs und die Aufrechterhaltung der Datengenauigkeit (Bertino und Ferrari 1998). Dazu gehört, dass Geschäftsregeln durchgesetzt und Änderungen nur autorisierten Benutzern anvertraut werden (Botha 2002). Ein optimal eingesetztes DBMS ermöglicht den gleichzeitigen Zugriff mehrerer Nutzer auf dieselben Daten und kontrolliert diese Zugriffe zusätzlich (Staud 2005). Diese Merkmale tragen gemeinsam zur Aufrechterhaltung der Datenqualität, -sicherheit und -genauigkeit in DBMS bei und sind daher für eine effektive Datenverwaltung in Unternehmen unerlässlich. Nach Schicker (2017) sorgt ein DBMS dafür, dass die Verbindung zu den Daten nicht mehr direkt erfolgt. Schicker (2017) führt an, dass die Endanwender nicht mehr in Kontakt mit den physischen Daten kommen und lediglich über eine Schnittstelle auf physische Daten zugreifen. Das DBMS fungiert dabei als Übersetzer zwischen den Anforderungen des Endanwenders und den Zugriffen auf die Datenbank. Dennoch besteht für die Endanwender immer noch die Möglichkeit die Daten ohne Zwischenschritt zu manipulieren, da die Schnittstelle simpel ist. (Schicker 2017) Das Prinzip der Abstraktion in Datenbanken schirmt die Nutzer von der internen Komplexität ab und ermöglicht es ihnen, Datenstrukturen zu definieren und darauf zuzugreifen, ohne sich um die Details der physischen Speicherung zu kümmern (Benedikt und Senellart 2012). Zudem sind einige der bereits genannten Eigenschaften und Funktionen eines DBMS auch als Vorteile hervorzuheben, wie zum Beispiel der Zugriffsschutz, da ein Verwaltungssystem nicht nur Zugriffe ermöglicht, sondern diese auch verfolgt und auf Basis eines Berechtigungssystems einschränken kann. Zudem wird der Mehrfachzugriff erst durch das DBMS ermöglicht, welcher ein wesentlicher Bestandteil für die Nut-

zung einer Datenbank im industriellen Einsatz bildet (Schicker 2017). Neben den bereits genannten Punkten umfasst ein DBMS weitere Funktionalitäten, welche in Abbildung 2 dargestellt sind. Eine weitere hier abgebildete Funktion ist der Report-Generator, folglich die Erstellung von aufbereiteten Berichten auf Basis von Datensätzen in den Basistabellen. Dies ist vor allem für Unternehmensberichte und Kennzahlendarstellungen in regelmäßigen Abständen und für wiederkehrende Tätigkeiten wesentlich (Steiner 2017). Als weiterer Punkt, welcher auch innerhalb dieser Ausarbeitung behandelt wird, ist der Datenkonverter für die Export-/ und Importfunktion zu nennen. Dieser bildet die Schnittstelle zwischen den Datenbanktabellen und dem Format der Dokumente in die Daten importiert oder aus denen Daten extrahiert werden. Der Menügenerator ermöglicht die Erstellung von Menüs für Datenbankaktionen und erleichtert so die Benutzerführung durch verschiedene Datenbankanwendungen (Steiner 2017).

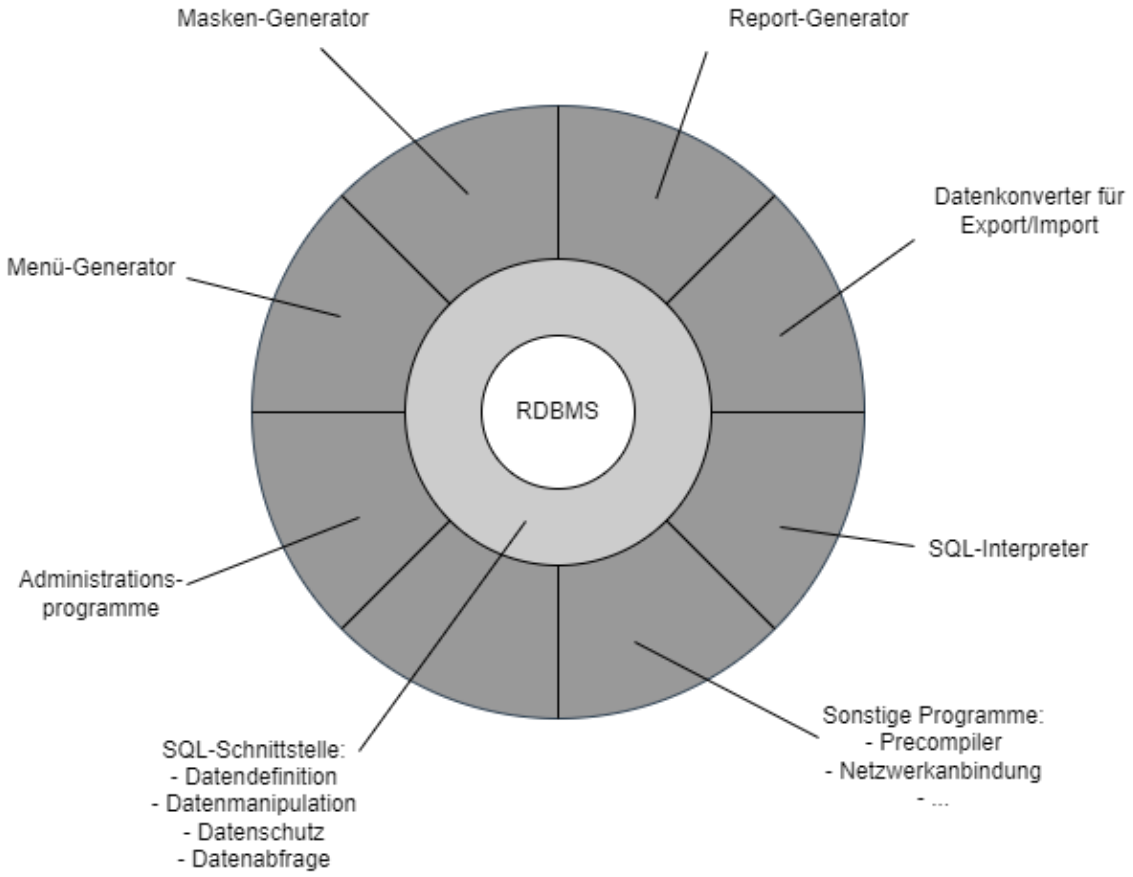


Abbildung 2: Schalenmodell für die Funktionen eines DBMS (in Anlehnung an Steiner 2017, S. 8)

Die mehrfach erwähnte Datenbank bildet die Basis des Gesamtsystems in einer Produktionsumgebung, da in ihr die Daten gespeichert werden. Genauer gesagt ist eine Datenbank eine von einem DBMS verwaltete Ansammlung von Datenobjekten, die miteinander in Verbindung stehen (Schicker 2017). Datenbanken sind ebenso Datensammlungen, auf die mehrere Nutzer für verschiedene Zwecke zugreifen können und die darauf abzielen, Daten unabhängig von bestimmten Programmen zu verwalten (Zauner und Schrempf 2009). Eines der wichtigsten Merkmale von Datenbanken ist die strukturelle Unabhängigkeit, die es den Nutzern ermöglicht, mit den Daten zu interagieren, ohne deren Speicherorganisation zu kennen (Ehrich 1973). In Entwurfsumgebungen müssen Datenbanken eine integrierte Datenverwaltung bieten, Konsistenz gewährleisten und effiziente Zugriffsfunktionen bereitstellen (Pahle und Hübel 1992). Zusätzlich muss eine Datenbank verhindern, dass unberechtigte Benutzer Daten sichten oder manipulieren können und sollte es ermöglichen, die internen Daten und Strukturen zu ändern,

ohne dass der Benutzer seine Oberfläche und Funktionen anpassen muss (Steiner 2017). Die Eigenschaften von Datenbanken können variieren, und im Folgenden werden einige in dieser Ausarbeitung als ideal definierte Merkmale einer Datenbank genannt, wie etwa eine klare Struktur, Datenunabhängigkeit, Flexibilität und Datenintegrität. Datenbanken sollten mehrere Schlüsseleigenschaften aufweisen, um die Anforderungen von unterschiedlichen Projekten effektiv zu erfüllen. Datenunabhängigkeit trennt logische und physische Datendarstellungen und erhöht die Flexibilität bei der Nutzung und Entwicklung von Datenbanken (Hellerstein et al. 1999). Klare Strukturen und Datenflexibilität werden durch die strukturelle Anpassungsfähigkeit angesprochen, die es Datenbanken ermöglicht, sich an wechselnde Anforderungen anzupassen (Snuggs et al. 1974). Eine Datenbank soll zudem überschaubar sein, das bedeutet, dieselben Informationen sollen nicht mehrfach gespeichert oder kontrolliert werden (Staud 2005). Hierdurch sollen Redundanzen in der Informationslage vermieden werden. Datenintegrität, eine weitere wesentliche Eigenschaft, gewährleistet die Genauigkeit und Konsistenz der gespeicherten Informationen (Engles 1972). Durch die Einbeziehung dieser Merkmale können Datenbanksysteme eine solide Grundlage für die effektive Verwaltung und Nutzung von Daten in Unternehmen bieten.

2.2 Relationales Datenmodell und Datenbankdesign-Prinzipien

Heute werden im Wesentlichen vier Datenbankmodelle unterschieden. Dazu zählen relationale, objektorientierte, hierarchische und netzwerkartige Datenbanken (Bercich 2002). Relationale Datenbanken haben seit ihrer Einführung durch E. F. Codd 1970 eine bemerkenswerte Evolution durchlaufen und sich als stabiles Konzept erwiesen (Vossen 1999). Trotz ihrer Dominanz in den letzten Jahrzehnten bestehen weiterhin Herausforderungen bei der effizienten Verwaltung hierarchischer Daten in relationalen Systemen (Brunel und Finis 2013). Bereits in den 1980er Jahren, als die ersten relationalen Datenbankprodukte auf den Markt kamen, wurden die Grenzen dieser Technologie für bestimmte Anwendungen diskutiert (Blaser 1995). Relationale Datenbanken wurden in den vergangenen Jahrzehnten stetig mehr eingesetzt (Schicker 2017). Trotz dieses Fortschritts sind viele der heutzutage genutzten Datenbanken bereits über mehrere Jahrzehnte alt und basieren nach wie vor auf anderweitigen Datenbankstrukturen (Schicker 2017). Relationale Datenbanken bilden den Standard für Datenmanagement in Unternehmen (Pernul und Unland 2003). Sie unterscheiden sich von hierarchischen Datenbanken durch ihren logischen Aufbau. Anstatt die Daten hierarchisch in einer Datei zu organisieren, werden sie in Form von Tabellen abgelegt, geordnet nach Themengebieten (Steiner 2017). Relationale Datenmodelle speichern Daten in eindimensionalen Tabellen und stellen Verknüpfungen zwischen unterschiedlichen Tabellen her (Staud 2005). Diese Verknüpfungen erfolgen über die Inhalte und Identifikationsschlüssel einzelner Einträge (Staud 2005). Während kleine Tabellen bereits mit wenig Expertise und in kurzer Zeit erstellt werden können, erfordern größere Datenbanken mit einer Vielzahl an Tabellen mehr Expertise und einen sauberen und durchdachten Aufbau (Schicker 2017). Im industriellen Betrieb bestehen relationale Datenbanken aus hunderten oder tausenden Tabellen, weshalb ein gut durchdachter Datenbankaufbau von entscheidender Bedeutung für einen sicheren Betrieb ist (Schicker 2017). Aktuelle Modelle kämpfen jedoch mit einer unvollständigen Datendarstellung, was zu Informationsverlusten bei der Speicherung führt (Panse 2009). Zur Bewältigung dieser Probleme ist regelmäßig eine Datenbankreorganisation erforderlich, die die Überwachung und Beeinflussung der physischen Datenspeicherung beinhaltet (Dorendorf und Küspert 2000). In der Forschung wird weiterhin nach Lösungen für die Herausforderungen bei der Datenbankreorganisation gesucht, insbesondere für große Datenbanken und hochverfügbare Systeme (Dorendorf und Küspert 2000).

Neue Tabellen können mit geringem Aufwand hinzugefügt oder entfernt werden, wodurch die gesamte Datenbankstruktur mit ebenfalls geringem Aufwand neu aufgesetzt werden kann (Schicker 2017). Weiterhin sind Zugriffe auf Tabellen einfach zu programmieren, was zu dem bereits erwähnten anteilig großen Einsatz der relationalen Datenbank in der Praxis führt (Schicker 2017). Relationale Datenbanken sind zwar weit verbreitet, stehen aber bei der modernen Datenverwaltung vor mehreren Herausforderungen. Dazu gehören Skalierbarkeitsprobleme,

Schwierigkeiten bei der Verarbeitung von Echtzeitdaten und unstrukturierten Daten (Harpreet Kaur et al. 2013). Zudem können einige Daten nur redundant in Tabellenform abgespeichert werden (Schicker 2017). Ein weiterer Nachteil ist die sinkende Überschaubarkeit mit steigender Anzahl an Tabellen, sodass alle relationalen Datenbanken nach einigen Jahren restrukturiert werden sollten, um die Übersichtlichkeit länger bewahren zu können (Steiner 2017). In Tabelle 1 werden wesentliche Begriffe im Umfang dieser Ausarbeitung mit ihrer formellen und informellen Bezeichnung aufgelistet. In relationalen DBMS werden Tabellen formell als Relationen bezeichnet (siehe Tabelle 1). Eine Relation ist eine zweidimensionale Struktur, die durch Zeilen und Spalten gekennzeichnet ist, wobei jeder Eintrag einen Datenelementwert darstellt (Deen 1977). Ein Tupel beschreibt ein Objekt vollständig und besteht aus Attributen, die spezifische Eigenschaften dieses Objekts darstellen (Staud 2005). In relationalen Datenbanken ist ein Tupel eine geordnete Menge von Elementen (Michael Smith et al. 2021). Ein Tupel stellt dabei genau eine Zeile innerhalb einer Relation dar, wie in Tabelle 1 aufgeführt. Attribute werden informell als Spalten einer Tabelle bezeichnet (siehe Tabelle 1). Sie sind jeweils mit einem bestimmten Bereich verbunden, der ihre möglichen Werte einschränkt (Michael Smith et al. 2021). Jedes Tupel muss durch einen Identifikationsschlüssel oder auch Primärschlüssel eindeutig identifizierbar sein. Ein Primärschlüssel ist ein eindeutiger Bezeichner (siehe Tabelle 1) für jedes Tupel in einer Relation, der für die Datenintegrität und eine effiziente Abfrage entscheidend ist (Kvet und Matiasko 2019). Auf den Einsatz eines Primärschlüssels wird im späteren Verlauf genauer eingegangen. Die Kardinalität einer Beziehung bezieht sich auf die Anzahl der darin enthaltenen Tupel (Ahad et al. 1989). Diese Konzepte sind grundlegend für das relationale Modell, bei dem Daten in Tabellen (Relationen) mit definierten Attributen und Abhängigkeiten organisiert werden.

Tabelle 1: Kernbegriffe innerhalb relationaler Datenbanken (in Anlehnung an Schicker 2017, S. 26)

Formale relationale Bezeichner	**Informelle Bezeichnung**
Relation	Tabelle
Tupel	Eine Zeile einer Tabelle
Kardinalität	Anzahl an Zeilen einer Tabelle
Attribut	Eine Spalte einer Tabelle
Primärschlüssel	Eindeutiger Bezeichner

Tupel und Attribute treten innerhalb von Relationen nicht geordnet auf, was als weitere Eigenschaft dieser genannt werden kann (Schicker 2017). Obwohl die Einträge keiner Struktur folgen, können einzelne Tupel in Beziehung zueinander gesetzt werden. Dabei gibt es unterschiedliche Möglichkeiten und Arten von Beziehungen, welche in unterschiedlichen Anwendungsfällen Einsatz finden. Da es eine Vielzahl an Beziehungsarten gibt, wird folglich nur auf wesentliche Beziehungen im Umfang dieser Arbeit eingegangen. Die drei Hauptformen von Beziehungen zwischen Datensätzen sind 1:1-, 1:n- und n:m-Beziehung, die jeweils unterschiedliche physische Merkmale aufweisen (Driscoll 1978). Die 1:1-Beziehung ist beim Datenbankdesign besonders wichtig (Bernick 2003). Eine 1:1-Beziehung verknüpft jeweils ein Tupel einer Tabelle mit einem Tupel einer anderen Tabelle. Ein Beispiel hierfür ist eine Datenbank, welche zwei Tabellen für die Speicherung von Daten über Personen und Smartphones beinhaltet. Hierbei hat jede Person genau ein Smartphone und jedes Smartphone genau einen Besitzer. Mit einer 1:n-Beziehung kann ein Tupel einer Relation mit mehreren Tupeln einer weiteren Relation verknüpft werden. Das bereits eingeführte Beispiel wird aufgegriffen, um eine weitere Relation erweitert, welche Daten über die Smartphone-Hersteller speichert. Eine Person ist immer noch Besitzer eines Smartphones und somit in Beziehung zu einem Smartphone-Hersteller. Dennoch können einem Smartphone-Hersteller mehrere Personen zugeordnet werden. Die dritte Beziehung wird als n:m-Beziehung bezeichnet, welche mehrere Tupel

einer Tabelle mit mehreren Tupeln einer anderen Tabelle verknüpft (Staud 2005). N:m-Beziehungen sind besonders komplex, wobei verschiedene Typen dieser Beziehung existieren (Shoval 1993). Ein Beispiel für diese Beziehung ist die Verknüpfung des Akteures Studierender mit den Vorlesungen die Studenten besuchen können. Ein Student kann unterschiedliche Vorlesungen besuchen und gleichzeitig besuchen mehrere Studenten ein und dieselbe Vorlesung.

Ein Primärschlüssel ist ein Attribut oder eine minimale Kombination von Attributen, die jedes Tupel eindeutig identifiziert (Meier und Kaufmann 2019). Der Primärschlüssel muss eindeutig sein und darf während des Vorhandenseins des Tupels nicht angepasst werden (Steiner 2017). Ein Fremdschlüssel ist ein Attribut in einer Tabelle, das den Primärschlüssel einer anderen Tabelle referenziert (Meier und Kaufmann 2019). Ein Fremdschlüssel kann auch Nullwerte annehmen (Steiner 2017). Eine Verbindung zweier Tabellen über einen Primärschlüssel kann anhand von den Tabellen Tabelle 2 und Tabelle 3 aufgezeigt werden. In Tabelle 2 werden für jeden Verkäufer deren verkauften Produkte und die damit erzielten Umsätze gespeichert. Hierbei ist die Spalte „VerkNr“ nicht der Primärschlüssel der Tabelle, da die Verkäufernummer mehrfach auftritt. In diesem Beispiel wäre eine Kombination der Attribute zwischen Verkäufernummer und dem Produktnamen als Primärschlüssel zu definieren. In Tabelle 3 werden für jeden Verkäufer die jeweiligen persönlichen Daten, wie der Postleitzahl und dem Wohnort gespeichert. In dieser Tabelle ist der Primärschlüssel die Verkäufernummer, da Daten zu den jeweiligen Personen lediglich einfach abgespeichert werden. Im Hinblick auf die bereits erläuterten Beziehungen handelt es sich in diesem Beispiel um eine 1:1-Beziehung in Tabelle 3 und eine 1:n-Beziehung in Tabelle 2. Um nun den Fremdschlüssel in dieses Beispiel einzubringen, wird zusätzlich die Tabelle 4 betrachtet. In Tabelle 4 werden Daten zu den Marken zu den entsprechenden Produkten gespeichert. In dieser Tabelle ist der Primärschlüssel der Produktname, da dieser einzigartig ist. Die entsprechenden Produktnamen treten dennoch ebenso in Tabelle 2 mehrfach auf. Nun wäre laut der bereits erbrachten Erläuterung der Fremdschlüssel die Spalte „Produktname“ in Tabelle 2, welche gleichzeitig den Primärschlüssel in Tabelle 4 bildet. Somit lassen sich diese beiden Tabellen in Beziehung zueinander setzen.

Tabelle 2: Beispieltabelle für Produkte und Umsätze (in Anlehnung an Schicker 2017, S. 28)

VerkNr	**VerkName**	**Produktname**	**Umsatz**
V1	Meier	Waschmaschine	11000
V1	Meier	Herd	5000
V1	Meier	Kühlschrank	1000
V2	Schneider	Herd	4000
V2	Schneider	Kühlschrank	3000
V3	Müller	Staubsauger	1000

Tabelle 3: Beispieltabelle für persönliche Informationen (in Anlehnung an Schicker 2017, S. 39)

VerkNr	**VerkName**	**PLZ**	**Wohnort**
V1	Meier	80331	München
V2	Schneider	70173	Stuttgart
V3	Müller	50667	Köln

Tabelle 4: Beispieltabelle für Produktmarken

Produktname	Marke
Waschmaschine	WaschSuper
Herd	HeißeWare
Kühlschrank	KalteKiste

Nachdem auf die grundlegende Funktionsweise und die Struktur von relationalen Datenbanken eingegangen wurde, werden nun wesentliche Prinzipien und Eigenschaften betrachtet, welche bei der Erstellung einer relationalen Datenbank eingehalten und beachtet werden sollten. Datenkonsistenz ist eine entscheidende Eigenschaft von relationalen Datenbanken und umfasst verschiedene Aspekte der Datenintegrität. Datenkonsistenz bedeutet, dass Daten und Informationen innerhalb einer Datenbank eindeutig und konkret beschrieben und gespeichert werden (Steiner 2017). Hierbei sollte gewährleistet sein, dass Daten nicht mehrfach und nur an einem Ort vorliegen. Beispielsweise sollten Personaldaten zu einer bestimmten Person nur einfach vorliegen und wenn mehrere Personen beispielsweise denselben Namen teilen, müssen diese durch weitere Identifikatoren voneinander klar getrennt gespeichert werden. Im Zusammenhang mit den Identifikatoren ist das Prinzip der Objektintegrität essenziell, um Probleme im Datenbankbetrieb zu vermeiden. Das Prinzip beschreibt die Notwendigkeit eines vollständig beschriebenen Primärschlüssels für jedes Tupel, andersherum kann ein Tupel unvollständig definiert, demnach beispielsweise Leerfelder enthalten solange der Schlüssel vollständig ist (Staud 2005). Ein weiteres Prinzip im Zusammenhang mit Primär- und Fremdschlüsseln ist das Prinzip der referentiellen Integrität. Die referentielle Integrität sorgt für die Konsistenz zwischen gekoppelten Tabellen, indem sie gültige Referenzen zwischen Daten erzwingt (Michael R. Blaha 2005). Es besagt, dass eine Verknüpfung zwischen Primär- und Fremdschlüssel nur durchgeführt werden darf, wenn der Fremd- und Primärschlüssel in der jeweiligen Tabelle ausdefiniert sind (Staud 2005). Somit soll vermieden werden, dass Beziehungen zu Tabellen hergestellt werden, welche noch nicht definiert sind und dadurch Fehlbeziehungen zu leeren oder undefinierten Tabellen entstehen. Traditionelle Einschränkungen der referentiellen Integrität können jedoch zu restriktiv sein, wenn es um unvollständige Beziehungen mit Nullwerten geht (Levene und Loizou 2001). Das klassische relationale Modell setzt vollständige Datenkonsistenz voraus, was bei der Integration von Daten aus mehreren, potenziell inkonsistenten Quellen problematisch sein kann. Deshalb erweitert das flexible relationale Modell das klassische Modell um die Unterstützung inkonsistenter Daten und bietet eine Semantik für Datenbankoperationen in solchen Szenarien (Agarwal et al. 1995). Trotz ihrer Bedeutung kann die referentielle Integrität zu Anomalien bei der Datenmanipulation führen, z. B. zur Blockierung gültiger Operationen oder zur Erzeugung unvorhersehbarer Ergebnisse in Abhängigkeit von der Reihenfolge der Durchsetzung von Beschränkungen (Markowitz 1990).

Eine Kerneigenschaft jeder Datenbank ist die langfristige Datenhaltung und ein weiteres Kernprinzip die persistente Datenhaltung. Dies beschreibt die langfristige und beständige Speicherung von Daten, welche für jede Datenverwaltung über einen längeren Zeitraum von großer Bedeutung ist (Staud 2005). Das Prinzip der semantischen Integrität in relationalen Datenbanken umfasst die Gewährleistung, dass die Daten den realen Bereich, den sie modellieren, genau repräsentieren (Hammer und McLeod 1975). Hierfür können Daten bereits bei der Einpflegung überprüft und bei falschen Datentypen entfernt werden (Schicker 2017). Zuverlässigkeit, ein weiteres Kernprinzip, umfasst auch den Schutz vor unerlaubten Zugriffen. Alle Aufrufe auf die Datenbank, einschließlich detaillierten Informationen zum Benutzer, sollten dokumentiert werden, um nachträglich unbefugte Zugriffe zu erkennen und festzustellen, welche Daten

manipuliert wurden (Schicker 2017). Diese Ansätze umfassen die Überwachung und Analyse von Nutzeranfragen, Zugriffsmustern und Sitzungsdaten, um anomales Verhalten zu erkennen (Sallam und Bertino 2018; Kamra et al. 2008). Einige Methoden verfolgen Zugriffshäufigkeiten und erstellen Nutzerprofile, um potenzielle Insider-Bedrohungen in Echtzeit zu identifizieren (Sallam und Bertino 2018). Ausfallsicherheit erfordert die ständige Dokumentation aller manipulierten Daten auf dem eigenen Server, um die Rekonstruktion wichtiger Daten auch in Notfallsituationen wie Rechnerabstürzen oder Unwettern und Bränden zu gewährleisten (Schicker 2017). Eine Datenbank sollte auch kontrolliert werden, was unter anderem die Sammlung von Daten über die Nutzung der Daten einbezieht. Die Überwachung ist für die Aufrechterhaltung der Sicherheit, der Audit-Fähigkeiten und der Erfüllung der Erwartungen der Benutzer von Datenbanksystemen unerlässlich (Guynes 1987).

Wie bereits im vorherigen Abschnitt erwähnt, ist Konsistenz eine wesentliche Eigenschaft im Datenbankmanagement. Transaktionen in Datenbanksystemen sind Sammlungen von Aktionen, die konsistente Transformationen von Systemzuständen sicherstellen (Elmagarmid 1991). „Unter Konsistenz erhaltenden Operationen […] [werden Operationen verstanden], die, ausgehend von konsistenten Datenbanken, diese Konsistenz beibehalten“ (Schicker 2017, S. 17). Um die Konsistenz der Datenbank aufrechtzuerhalten, müssen Transaktionen sorgfältig entworfen werden, um komplexe Operationen wie Löschungen zu verarbeiten, ohne Einschränkungen zu verletzen (van Hee et al. 2009). Grundlegende Eigenschaften, welche mit Transaktionen in Verbindung stehen, sind die ACID-Eigenschaften. Die ACID-Eigenschaften (Atomicity, Consistency, Isolation, Durability) sind grundlegend für herkömmliche DBMS und gewährleisten die Integrität und Zuverlässigkeit der Daten (Frank 2010). „Eine Transaktion ist aus der Sicht eines Datenbanksystems die mächtigste (Macro-) Operation auf den Datenbeständen, die atomar auszuführen ist. Sie wird entweder vollständig ausgeführt oder gar nicht.“ (Meyer-Wegener 1988) Atomarität in Datenbanken bezieht sich auf eine Eigenschaft, die gewährleistet, dass Transaktionen als eine einzige, unteilbare Einheit ausgeführt werden (Biswas et al. 2009). Bei mehrstufigen Transaktionen erkennt ein Modell der Atomarität an, dass Operationen auf niedrigerer Sicherheitsebene unabhängig von Operationen auf höherer Ebene übertragen oder abgebrochen werden müssen (Blaustein et al. 1993). Die Laufzeiterkennung von Atomaritätsverletzungen in nebenläufigen Programmen kann durch graphenbasierte Algorithmen erreicht werden, die aufgezeichnete Ereignisse und Synchronisationsmuster analysieren (Wang und Stoller 2006). Diese Ansätze zur Atomarität in verschiedenen Kontexten zeigen, wie wichtig sie für die Gewährleistung der Datenkonsistenz und Transaktionsintegrität in verschiedenen Datenbanksystemen sind. Die Eigenschaft Isolation in Datenbanktransaktionen stellt sicher, dass konkurrierende Operationen sich nicht gegenseitig beeinträchtigen und die Konsistenz gewahrt bleibt (Rastogi et al. 1993). Die Eigenschaft der Dauerhaftigkeit gewährleistet insbesondere, dass erfolgreiche Transaktionen bestehen bleiben und nicht rückgängig gemacht werden (Butnaru 2006). Zusammenfassend lässt sich sagen, dass relationale Datenbanken aufgrund ihrer einfachen und intuitiven Struktur weit verbreitet sind. Sie bieten eine leistungsfähige und flexible Möglichkeit zur Verwaltung und Abfrage großer Datenmengen und sind daher die dominierende Technologie in der Praxis. Die logische Strukturierung und Normalisierung der Daten helfen, Redundanzen zu minimieren und die Integrität der Daten zu gewährleisten, was sie zu einer bevorzugten Wahl für zahlreiche Anwendungen macht.

2.3 Structured Query Language

SQL steht für „Structured Query Language“ und hat sich in den letzten Jahrzehnten zu der relevantesten Datensprache und Datenbankschnittstelle entwickelt (Melton 1996). SQL wurde zunächst bei der Firma IBM unter dem Namen SEQEL entwickelt und von Oracle weiterentwickelt unter dem jetzt bekannten Kürzel SQL (Schicker 2017). Mittlerweile liegen drei verschie-

dene Entwicklungsstufen mit SQL1, SQL2 und SQL3 vor, wobei SQL2 den aktuellen Praxisstandard darstellt und deshalb im Umfang dieser Ausarbeitung behandelt wird (Schicker 2017). SQL dient zur Erstellung von Tabellen und zum Zugriff auf Datenbanken, wobei mittlerweile eine Vielzahl von Benutzeroberflächen entwickelt wurden, welche einen komfortablen Zugriff für Anwender auf die Sprache SQL bieten (Bühler 2019). Innerhalb der SQL-Sprache gibt es unterschiedliche Dialekte auf verschiedenen Plattformen und Implementierungen (Kline und Kline 2004). Dennoch ist SQL ISO-standardisiert und somit eine der einzigen Tabellensprachen, welche einer vollständigen Standardisierung nahekommt (Bühler 2019). Hinter der Abkürzung ISO verbirgt sich die internationale Organisation für Standardisierung. Ebenso ist die Sprache SQL auch ANSI-standardisiert, somit vom amerikanischen Institut für Standards (Meier und Kaufmann 2019). SQL ist nicht nur für Endanwender über die grafischen Oberflächen hilfreich, sondern vor allem auch für Datenbankprogrammierer (Schicker 2017). Dabei können Datenbankprogrammierer mithilfe von SQL-Tabellen in relationalen Datenbanken erzeugen, löschen, ändern und Daten in verschiedener Art einpflegen (Bühler 2019). DBMS wie Microsoft Access erlauben zudem Datenbanken zu erstellen und zu manipulieren ohne weitere Vorkenntnisse zu besitzen. Dennoch können auch in DBMS weitere Funktionen ausgeschöpft und komplexere Datenbänke aufgebaut werden (Bühler 2019). SQL gilt als sogenannte nicht prozedurale Programmiersprache, bedeutet, dass der Anwender nicht den Weg zum Erlangen der Daten definiert, sondern lediglich die gewünschte Ergebnismenge (Schicker 2017). SQL wird deshalb auch als beschreibende Sprache bezeichnet, da sie das gewünschte Ergebnis beschreibt und die notwendigen Rechenschritte nicht beachtet (Meier und Kaufmann 2019). Auch die Menge an Daten, die mit einer SQL-Abfrage manipuliert werden, ist nicht immer dieselbe. So können ganze Tabellen oder einzelne Abschnitte innerhalb einer Tabelle betrachtet und manipuliert werden (McFadyen und Kanabar 1991).

Im Folgenden wird nun die Funktionsweise der Datenbanksprache SQL detaillierter betrachtet. SQL umfasst unterschiedliche Schlüsselelemente. Die Data-Definition-Language (DDL) wird zur Beschreibung von Datenbankstrukturen verwendet, während Data-Manipulation-Language die Datenmanipulation durch Operationen wie Einfügen, Aktualisieren und Löschen ermöglicht (Setiyadi 2021). Die DDL bildet den Grundbaustein der Sprache SQL, da die Datendefinition für den Aufbau der Datenstruktur verwendet wird. Hiermit lassen sich Tabellen, sowie Felder innerhalb der Tabellen konfigurieren und die entsprechenden Datentypen anpassen (Steiner 2017). Die Data-Retrieval-Language, die oft als Teil von DML betrachtet wird, konzentriert sich auf das Abrufen von Daten mithilfe von SELECT-Anweisungen (Setiyadi 2021). Dabei sind die Ausprägungen, nach denen diese Daten abgefragt werden, frei wählbar (Steiner 2017). Data-Security-Language behandelt die Sicherheits- und Integritätsaspekte von Datenbanken (van der Lans 1986). Zu den jeweiligen Elementen der Sprache SQL existieren jeweils unterschiedliche Befehle, welche unter richtiger Anwendung eine Vielzahl von Möglichkeiten bieten, um nahezu alle Operationen auf einen Datenbestand durchzuführen. Eine Datenabfrage sorgt für die Betrachtung einer Teilmenge des Datenbestandes. Dabei werden Daten eingelesen und aufbereitet, ohne eine Veränderung der Daten vorzunehmen. Innerhalb von SQL wird eine Datenabfrage mithilfe des SELECT-Befehls durchgeführt, dessen Syntax lautet wie folgt:

Algorithmus 1: Syntax eines SELECT-Befehls (in Anlehnung an Schicker 2017, S. 16)

```
1. SELECT Spalten FROM Tabelle WHERE Bedingungen;
```

Die Platzhalter „Spalten“ und „Tabelle“ geben an, welche Daten aus welcher Tabelle ausgewählt werden sollen, während der Platzhalter „Bedingungen“ logische Abfragen spezifiziert, um die Auswahl weiter einzuschränken (Schicker 2017). Als Ergebnis wird ein Teilausschnitt der Tabelle an den Endanwender zurückgegeben. Ein SELECT-Befehl muss keine Bedingung enthalten und kann somit vereinfacht wie folgt formuliert werden:

Algorithmus 2: SELECT-Befehl, um alle Einträge einer Tabelle auszugeben (in Anlehnung an Schicker 2017, S. 101)

```
1. SELECT * FROM Personal;
```

Allgemein gesagt beginnt jeder SELECT-Befehl mit dem Indikator „SELECT“, worauf der Bezeichner „FROM“ folgt. Nach dem Bezeichner „FROM“ folgt der Tabellenname von der Tabelle, aus der die Daten abgerufen werden sollen. Zwischen den Bezeichnern „SELECT“ und „FROM“ werden die Spalten angegeben, die abgefragt werden sollen, wodurch eine Filterung des Datenbestandes ermöglicht wird. Wenn nicht alle Spalten in der Ausgabemenge benötigt werden, sondern beispielsweise nur die Spalten „Wohnort“ und „Hobby“ kann der SQL-Befehl folgendermaßen umgeformt werden:

Algorithmus 3: SELECT-Befehl um Daten einzelner Spalten auszugeben (in Anlehnung an Schicker 2017, S. 101)

```
1. SELECT Wohnort, Hobby FROM Personal;
```

Innerhalb dieser Ausarbeitung werden reservierte Wörter der SQL-Sprache in Großbuchstaben dargestellt, um diese hervorzuheben. Dennoch spielen bei der Formulierung der SQL-Befehle Groß- und Kleinschreibung keine übergeordnete Rolle, sodass diese flexibel definiert werden können (Schicker 2017). Die Befehle können zusätzlich in verschiedene Spalten aufgeteilt werden, was in zahlreichen Fällen zu einer besseren Lesbarkeit führt (Schicker 2017). Ein Beispiel des vorherigen Befehls mit den genannten Veränderungen ist im Folgenden dargestellt:

Algorithmus 4: Formatierter SELECT-Befehl (in Anlehnung an Schicker 2017, S. 102)

```
1. SElecT Wohnort,
2. Hobby FroM Personal;
```

Durch die Einfachheit der Befehle wird bereits deutlich, dass SQL eine mächtige Sprache ist, um mit geringem Aufwand große Datenmengen abzufragen. Hierbei muss der Endanwender nicht den Ort beschreiben, an dem die Daten liegen, sondern lediglich definieren welche Daten er abfragen möchte. Wie die Daten abgespeichert sind, ist dem Endanwender in diesem Fall unschlüssig und wird lediglich durch unterschiedliche Laufzeiten bei den Abfragen bemerkbar (Schicker 2017). Der grundlegende SELECT-Befehl kann um weitere Bezeichner ergänzt werden um die Komplexität der Abfrage zu steigern (Eckstein und Schultz 2017). Die grundlegende Syntax des SELECT-Befehls mit den typischen Bezeichnern umfasst folgende Struktur:

Algorithmus 5: Syntax des SELECT-Befehls (in Anlehnung an Schicker 2017, S. 102)

```
1. SELECT [ ALL | DISTINCT ] Spaltenauswahlliste
2. FROM Tabellenliste
3. [ WHERE Bedingung ]
4. [ GROUP BY Spaltenliste ]
5. [ HAVING Bedingung ] ];
```

Auf die einzelnen Bezeichner wird in den weiteren Abschnitten noch genauer eingegangen. Vorab gilt es dennoch zu erwähnen, dass ein SELECT-Befehl auch aus mehreren Hauptteilen bestehen kann (Kaur 2017). Diese werden dann mithilfe von Vereinigungsoperatoren, wie zum Beispiel den Operatoren „UNION“, „EXCEPT“ und „INTERSECT“ miteinander verbunden. Die Komplexität des SELECT-Befehls kann zusätzlich durch eine geschachtelte Abfrage erhöht werden. Hierbei werden mehrere SELECT-Befehle ineinander definiert. Ein Beispiel für eine verschachtelte Abfrage lautet wie folgt:

Algorithmus 6: Beispiel einer verschachtelten SQL-Abfrage (in Anlehnung an Schicker 2017, S. 116)

```
1. SELECT Name
2. FROM PERSONAL
3. WHERE Lohn >= ALL (SELECT Lohn FROM PERSONAL);
```

Der Befehl lässt sich in eine äußere und eine innere SELECT-Anweisung unterteilen. In der Äußeren werden alle Namen aus der Personaltabelle abgerufen. Die Liste wird dann um eine Bedingung erweitert. In der Bedingung wird geprüft, ob der Lohn der Personen aus der äußeren Liste größer oder gleich dem Lohn aus der inneren Liste ist. Der Bezeichner „ALL“ sorgt dafür, dass der Vergleich mit allen Einträgen vollzogen wird (Kaufmann und Meier 2023). Das Resultat ist somit die Menge aller Mitarbeiter, welche den höchsten Lohn laut den vorliegenden Einträgen haben. Die Abfrage von korrelierenden Tabellen innerhalb eines SELECT-Befehls kann vorteilhaft, aber gleichzeitig mit Blick auf die Laufzeit kostspielig sein (Simon 2023). Ein weiterer häufig verwendeter SQL-Befehl ist der WHERE-Befehl, welcher in Zusammenhang mit dem SELECT-Befehl eingesetzt wird. Mithilfe des WHERE-Befehls kann der Datenausschnitt heruntergefiltert werden und stellt somit die bereits genannte Bedingung innerhalb der Struktur des SELECT-Befehls dar. Vor allem bei großen Datenmengen kann es hilfreich sein die Datenmenge auf den wesentlichen Teilausschnitt zu reduzieren. Hierbei können Daten herausgefiltert werden, die eine bestimmte Bedingung erfüllen oder auch nicht erfüllen. Wenn die ausgewerteten Daten Nullwerte beinhalten, kann eine Bedingung auch als nicht wahr ausgewertet werden. Dadurch gilt das Testen mit Nullwerten generell als schwierig und der Vergleich mit dem Gleichheitsoperator (=) wird fehlschlagen (Leonid Libkin und Liat Peterfreund 2020). Um dennoch auf Nullwerte zu testen, muss der Ausdruck „IS NULL“ oder „IS NOT NULL“ verwendet werden (Simon 2023). SQL bietet zusätzlich die Möglichkeit mehrere Bedingungen auf einmal zu prüfen und dabei zu unterscheiden, ob alle Bedingungen erfüllt oder nicht erfüllt sein müssen oder immer nur eine der Bedingungen. Dafür existieren die logischen Operatoren „AND“ und „OR“. Ein Beispiel für eine WHERE-Bedingung mit logischen Operatoren lautet:

Algorithmus 7: Beispiel eines WHERE-Befehls mit logischen Operatoren

```
1. SELECT *
2. FROM Mitarbeitende
3. WHERE Ort='Musterstadt' AND Name='Mustermann';
```

In diesem Beispiel werden die Bedingungen nach dem Wohnort des Mitarbeitenden und seinem Namen gleichzeitig geprüft. Gleichzeitig wird die Prüfung im Detail nicht durchgeführt, sondern nacheinander, sodass eine Teilmenge gebildet wird und diese durch die zweite Bedingung noch weiter heruntergefiltert wird (Kaufmann und Meier 2023). Somit gibt der SELECT-Befehl aus diesem Beispiel alle Einträge aus, bei denen ein Mitarbeiter den Nachnamen „Mustermann“ hat und gleichzeitig in „Musterstadt“ wohnt. Wenn der logische Operator „AND“ durch den Operator „OR“ ersetzt wird, müssen nicht beide Bedingungen gleichzeitig erfüllt sein. Dann werden alle Einträge ausgegeben, die entweder den Wohnort „Musterstadt“ oder

den Nachnamen „Mustermann“ beinhalten. Alle bisher betrachteten Befehle beziehen sich auf einen Datenausschnitt, oft ist aber die Betrachtung unterschiedlicher Datenausschnitte relevant und vor allem die Kombination der unterschiedlichen Datenausschnitte. Für die gemeinsame Betrachtung mehrerer Datenausschnitte existieren in SQL die Mengenoperatoren, welche gemeinsam mit ihrer Bezeichnung und der Algebra-Formel in Tabelle 5 aufgelistet sind. Der UNION-Operator ermöglicht die Kombination von Ergebnissen aus zwei oder mehr SELECT-Anweisungen oder Tabellen (Kaur 2017). Die Erweiterung um die Bezeichnung „ALL“ sorgt dafür, dass auch Duplikate mit in der finalen Datenmenge angezeigt werden. Dabei werden die gesamten Teilmengen miteinander verbunden. Im Gegensatz dazu sorgt der INTERSECT-Operator für die Ausgabe aller Zeilen, welche in beiden Teilmengen vorhanden sind, somit der gemeinsame Schnitt der Teilmengen (Dunn et al. 2002). Die beiden ersten Operatoren geben alle Gemeinsamkeiten und im Falle der Vereinigung noch weitere Daten aus. Der EXCEPT-Operator gibt lediglich alle Einträge einer Tabelle oder Datenmenge aus, welche nicht in einer weiteren Tabelle oder Datenmenge vorhanden sind, somit die Differenz der beiden Teilmengen (Dunn et al. 2002).

Tabelle 5: Auflistung Vereinigungsoperatoren in SQL mit Algebra-Formel (in Anlehnung an Schicker 2017, S. 138)

Bezeichnung	Algebra	SQL
Vereinigung	R1 U R2	UNION
Schnitt	R1 ∩ R2	INTERSECT
Differenz	R1 \ R2	EXCEPT

Bei der Verwendung der Mengenoperatoren müssen einige Regeln beachtet werden, so zum Beispiel die Anzahl der Spalten der Tabellen, die miteinander verbunden werden sollen. Die Spaltenanzahl und die Typen müssen übereinstimmen, damit die Mengenoperatoren auf diese Tabelle angewendet werden können (Berg Hansen 2020). Die Namen für die kombinierte Tabelle werden nur aus dem ersten SELECT-Befehl verwendet, sodass alle weiteren Spaltennamen ignoriert werden (Kaur 2017). Die einzelnen SELECT-Befehle können Bedingungen und auch Gruppierungs-Befehle beinhalten, aber keine Sortierungs-Befehle (Simon 2023). Diese können erst auf die Gesamtmenge nach den Mengenoperationen angewendet werden. Ein Beispiel für eine Vereinigungsoperation in SQL lautet:

Algorithmus 8: Beispiel einer Vereinigung mit dem UNION-Befehl (in Anlehnung an Schicker 2017, S. 122)

```
1. SELECT *
2. FROM Sportclub
3. UNION
4. SELECT *
5. FROM Fotoclub;
```

Diese Operation kann nur dann durchgeführt werden, wenn die beiden Tabellen vereinigungsverträglich sind (Kaufmann und Meier 2023). Unter der Annahme, dass die Spalten und die Datentypen der beiden Beispieltabellen identisch sind ist das Ergebnis die Gesamtmenge beider Tabellen ausgenommen von den Duplikaten. Damit die Duplikate mit erfasst werden müsste, wie bereits im vorherigen Abschnitt erwähnt der UNION-Operator durch ein „UNION ALL“ ersetzt werden (Kaufmann und Meier 2023). Dasselbe Beispiel mit dem EXCEPT-Operator sieht wie folgt aus:

Algorithmus 9: Beispiel einer Vereinigung mit dem EXCEPT-Befehl (in Anlehnung an Schicker 2017, S. 122)

```
1. SELECT *
2. FROM Sportclub
3. EXCEPT
4. SELECT *
5. FROM Fotoclub;
```

Durch diesen SQL-Befehl werden alle Einträge ausgegeben, welche nur in der Tabelle „Sportclub“ vorhanden sind. Einträge aus der Tabelle „Sportclub“ die auch in der Tabelle „Fotoclub“ vorhanden sind entfallen aus der Ergebnismenge (Kaufmann und Meier 2023). Sind genau die Daten als Ergebnismenge gewünscht, die sowohl in der Tabelle „Sportclub“ als auch in der Tabelle „Fotoclub“ vorliegen kann der Mengenoperator „INTERSECT“ verwendet werden und zu diesem SQL-Befehl umgeformt werden:

Algorithmus 10: Beispiel einer Vereinigung mit dem INTERSECT-Befehl (in Anlehnung an Schicker 2017, S. 122)

```
1. SELECT *
2. FROM Sportclub
3. INTERSECT
4. SELECT *
5. FROM Fotoclub;
```

Die Mengenoperatoren stellen nicht die einzige Möglichkeit dar, mehrere Tabellen zu einer Ergebnismenge zu bündeln. Eine weitere Methode, welche zusätzliche Vorteile gegenüber den Mengenoperatoren birgt, sind die JOIN-Operatoren (Kaur 2017). Die grundlegende Syntax eines JOIN-Befehles lautet:

Algorithmus 11: Syntax des JOIN-Befehls

```
1. SELECT Spalten
2. FROM Tabelle A JOIN Tabelle B;
```

Der SELECT-Befehl wird um den JOIN-Operator erweitert und lediglich die Tabelle ergänzt, welche mit der Basistabelle verbunden werden soll. Der Unterschied zu den Mengenoperatoren ist, dass die Spaltennamen, -anzahl und –typen nicht identisch sein müssen (Simon 2023). Bei der Angabe der auszuwählenden Spalten innerhalb des SELECT-Befehls ist es deshalb auch möglich nicht nur Spalten aus der Tabelle „Tabelle A“ zu definieren, sondern auch Spalten aus der Tabelle „Tabelle B“, welche mit Tabelle „Tabelle A“ verbunden wird. Somit wird nicht nur die Zeilenanzahl der Gesamtmenge verringert oder erhöht, sondern auch die Spaltenanzahl kann angepasst werden. Zudem kann durch die JOIN-Operatoren eine temporäre neue Tabelle erstellt werden, welche spezifische Ergebnisse in neuer Darstellungsweise liefern kann (Simon 2023). Für die weitere Ausarbeitung sind zwei übergeordnete JOIN-Typen wesentlich. Der Standard „JOIN“ ist der „INNER JOIN“, welcher bei fehlender Spezifizierung automatisch angenommen wird (Simon 2023). Der „INNER JOIN“ verbindet beide Tabellen und gibt alle Einträge aus bei denen eine Übereinstimmung auf das zu prüfende Element vorliegt (Mielebacher 2024). Der „OUTER JOIN“ erweitert den „INNER JOIN“ um nicht übereinstimmende Zeilen und fungiert in der Grundform wie der Vereinigungsoperator „UNION“. Der „OUTER JOIN“ lässt sich in einen „LEFT JOIN“, „RIGHT JOIN“ und „FULL JOIN“ unterteilen

(Starbuck 2023). Der „FULL JOIN“ sorgt für dieselbe Ergebnismenge wie ein „UNION“ mit der Ausnahme, dass in diesem Fall zusätzlich auf eine Bedingung geprüft werden kann (Simon 2023). Der „LEFT JOIN“ und „RIGHT JOIN“ verbinden alle Zeilen der ausgewählten Tabellen, die das entsprechende Element verbinden und die jeweiligen Einträge jeweils aus nur einer Tabelle (Starbuck 2023). Bei einem „LEFT JOIN“ werden die nicht übereinstimmenden Zeileneinträge aus Tabelle „Tabelle A“ hinzugezogen, bei einem „RIGHT JOIN“ die Einträge aus Tabelle „Tabelle B“. Für ein besseres Verständnis werden erneut Beispiele der einzelnen SQL-Befehle angeführt, beginnend mit einem „INNER JOIN“:

Algorithmus 12: Beispiel eines INNER JOIN-Befehls

```
1. SELECT a.Wohnort, b.Beruf
2. FROM Tabelle1 a
3. INNER JOIN Tabelle2 b ON
4. a.Mitarbeiternummer = b. Mitarbeiternummer;
```

Der „JOIN“ wird zwischen den Tabellen „Tabelle1“ und „Tabelle2“ durchgeführt und das verbindende Element ist die Spalte „Mitarbeiternummer“, welche in beiden Tabellen existiert. Hierbei werden alle Einträge in die Ergebnismenge übernommen, bei denen die Mitarbeiternummer aus der Tabelle „Tabelle1“ der Mitarbeiternummer aus der Tabelle „Tabelle2“ entspricht. Ausgewählt werden in dem SELECT-Befehl der Wohnort der Mitarbeitenden aus der Tabelle „Tabelle1“ und der Beruf derselben Mitarbeiter aus der Tabelle „Tabelle2“. Für dasselbe Beispiel lautet der SQL-Befehl mit einem „LEFT JOIN“ wie folgt:

Algorithmus 13: Beispiel eines LEFT JOIN-Befehls

```
1. SELECT a.Wohnort, b.Beruf
2. FROM Tabelle1 a
3. LEFT JOIN Tabelle2 b ON
4. a.Mitarbeiternummer = b. Mitarbeiternummer;
```

Bis auf den JOIN-Operator bleibt der SELECT-Befehl identisch und dennoch ist die Ergebnismenge eine deutlich andere. In diesem Fall werden alle Einträge übernommen, bei denen die Mitarbeiternummern aus den Tabellen „Tabelle1“ und „Tabelle2“ übereinstimmen, demzufolge dieselbe Ergebnismenge aus dem „INNER JOIN“. Ergänzt wird die Ergebnismenge um alle Einträge aus der Tabelle „Tabelle1“ die nicht in der Tabelle „Tabelle2“ vorliegen. Der „RIGHT JOIN“ erzielt das umgekehrte Ergebnis, somit das Einbeziehen von allen nicht übereinstimmenden Werten aus Tabelle „Tabelle2“. Die verschiedenen Verbindungstypen ermöglichen eine flexible Erstellung von Ergebnisteilmengen und können in ihrer Komplexität beliebig gesteigert werden. Einträge in relationalen Tabellen liegen ungeordnet vor, sodass auch die Ausgaben nach den bisherigen SQL-Operationen ungeordnet sind. Dennoch gibt es die Möglichkeit, diese Zeilen während einer Datenaufbereitung in eine logische Reihenfolge zu überführen, somit beispielsweise nach einem bestimmten Kriterium zu sortieren. Für diesen Anwendungsfall existiert der ORDER BY-Befehl, dessen Syntax lautet wie folgt:

Algorithmus 14: Syntax des ORDER BY-Befehls

```
1. SELECT Spalten
2. FROM Tabelle
3. WHERE Bedingung
4. ORDER BY Kriterium;
```

Sowohl in der Syntax von SQL als auch bei der rechnerischen Auswertung der Befehle wird der Befehl „ORDER BY“ an das Ende gestellt (Simon 2023). Hierdurch verändert sich lediglich die Reihenfolge der Dateneinträge und die Ergebnismenge bleibt identisch. Zusätzlich kann die Ergebnismenge auch nach mehreren Spalten auf einmal sortiert werden. Im Standard werden die Werte aufsteigend sortiert und mit der Zugabe des Operators „DESC“ für „descending“ werden die Werte in absteigender Reihenfolge sortiert (Mielebacher 2024). Der Operator „ASC“ für „ascending“ kann hinzugefügt werden, hat im Normalfall aber keinen Einfluss auf die Sortierung da dieser Operator meist der Standardwert ist (Mielebacher 2024). Auch die Datentypen haben einen Einfluss auf die Sortierung der Ergebnismenge werden. So sortieren einige DBMS Groß- und Kleinbuchstaben getrennt und manche nicht (Simon 2023). Damit auf Basis einer Ergebnismenge oder Tabelle Berechnungen durchgeführt werden können, existieren Aggregatfunktionen. Dabei werden die Daten zunächst aggregiert und je nach Funktion unterschiedlich behandelt und in einer temporären Tabelle zwischengespeichert (Badia 2020). Zu den wichtigsten Aggregatfunktionen gehören die Funktionen „count()“, „min()“, „max()“, „sum()“, „avg()“ sowie „string_agg()“ und „group_concat()“. Die count-Funktion zählt die Anzahl der Zeilen in einer Spalte, welche innerhalb der Funktion angegeben werden muss, und kann als Ergebnis innerhalb einer Spalte ausgegeben oder bereits im SELECT-Befehl definiert werden (Starbuck 2023). Die min- und max-Funktion geben den jeweiligen letzten und ersten Wert aus einer sortierten Reihenfolge wieder (Schicker 2017). Im Umgang mit Zahlen gibt es die sum- und avg-Funktion, die aus allen Einträgen einer numerischen Spalte die Summe oder den Durchschnitt bilden (Mielebacher 2024). Für Zeichenketten existieren Verkettungsfunktionen wie die string_agg-Funktion oder die group_concat-Funktion (Simon 2023). Hierbei gilt es zu beachten, das Nullwerte nicht betrachtet demzufolge bei der Gruppierung und Berechnung übersprungen werden (Schicker 2017). Für eine bessere Übersicht innerhalb der SQL-Befehle und Berechnungsschritte wird häufig empfohlen die Zwischenergebnisse in temporären Tabellen zu speichern und sie dann weiterzuverarbeiten (Simon 2023). Ein Beispiel für eine Zählung wäre die Anzahl der Mitarbeiter in einem Unternehmen, die in einer bestimmten Abteilung arbeiten. Der SQL-Befehl dieser beispielhaften Aufforderung lautet wie folgt:

Algorithmus 15: Beispiel einer SQL-Abfrage mit COUNT-Funktion

```
1. SELECT COUNT(Name)
2. FROM Mitarbeiter
3. WHERE Abteilung='Entwicklung';
```

Das Ergebnis ist eine einzelne Tabelle mit einem einzigen Wert, der Anzahl der Mitarbeiter innerhalb der Entwicklungsabteilung. Insofern Mitarbeiter nicht mehrfach in der Tabelle „Mitarbeiter“ vorkommen oder ehemalige Mitarbeiter in den Datenbeständen vorhanden sind entspricht die Anzahl der genauen momentanen Mitarbeiteranzahl in der entsprechenden Abteilung.

Bisher sind lediglich Daten verbunden, manipuliert oder aggregiert worden doch ein wesentlicher Teil der Arbeit mit Datenbanken besteht darin, Daten in Tabellen mithilfe von SQL-Befehlen einzupflegen. Bei der Einpflegung von Daten muss beachtet werden, dass das Identifizierungsmerkmal noch nicht in der Tabelle vorhanden ist. Wenn der Primärschlüssel bereits in der Tabelle vorliegt, können die Daten nicht eingepflegt, sondern diese müssen aktualisiert werden (Mielebacher 2024). Daten können mithilfe des „INSERT INTO“-Befehls in Tabellen eingepflegt werden. Die Dateninhalte müssen innerhalb von Hochkommata oder Anführungszeichen gesetzt werden (Bühler 2019). Ein Beispiel für das Hinzufügen von Daten in die exemplarische Tabelle „Mitarbeiter“ sieht wie folgt aus:

Algorithmus 16: Beispiel eines INSERT INTO-Befehls (in Anlehnung an Schicker 2017, S. 135)

```
1. INSERT INTO Mitarbeiter (Name, Straße, Stadt, Abteilung)
2. VALUES
3. ('Müller', 'Riesweg', 'Olten', 'Buchhaltung'),
4. ('Minder', 'Freiestrasse', 'Zürich', 'Entwicklung');
```

Der INSERT INTO-Befehl wird um die jeweilige Tabelle ergänzt, in die die Daten gepflegt werden sollen, in diesem Beispiel die Tabelle „Mitarbeiter“. Darauf folgt der Bezeichner „VALUES“ gefolgt von den Datenreihen, die hinzugefügt werden sollen. Hierbei ist zu beachten, dass die Daten in der richtigen Reihenfolge vorliegen müssen, folglich in diesem Beispiel entsprechend der Spaltenreihenfolge „Nachname“, „Straße“, „Stadt“ und „Abteilung“. Es kann eine beliebige Anzahl an Einträgen definiert und gleichzeitig importiert werden, solange die Datentypen und die Anzahl der Einträge der Tabelle entsprechen (Badia 2020). In diesem Beispiel werden zwei neue Tupel in die Tabelle hinzugefügt. Diese Art von Dateneinpflegung bietet sich bei einer geringen Anzahl von Dateneinträgen an. Im Hinblick auf die global wachsenden Datenvolumen innerhalb von Datenbanken sind andere Methoden für den Datenimport notwendig. Bulk Loading ist ein Verfahren zum effizienten Einfügen großer Datenmengen in Datenbankindexstrukturen (van den Bercken et al. 1997). Zudem existieren Extract-Transform-Load-Werkzeuge (ETL-Werkzeuge) und weitere Programmierschnittstellen, die den Datenimport für den Anwender vereinfachen (Kaufmann und Meier 2023). Zuletzt werden die beiden SQL-Befehle „UPDATE“ und „DELETE“ betrachtet, welche die Möglichkeit bieten bereits eingetragene Daten zu ändern oder zu löschen. Der UPDATE-Befehl ermöglicht das Ändern von bereits gespeicherten Zeichenketten, Zahlen, berechneten Werten und Werten aus anderen Tabellen (Mielebacher 2024). Ein Beispiel für eine UPDATE-Anweisung lautet:

Algorithmus 17: Beispiel eines UPDATE-Befehls (in Anlehnung an Schicker 2017, S. 134)

```
1. UPDATE Mitarbeiter
2. SET Name = 'Meier', PLZ= '123456, Stadt= 'Neustadt'
3. WHERE Mitarbeiternummer = 5;
```

Auf den Bezeichner „UPDATE“ folgt die Tabelle, in der die Dateneinträge angepasst werden sollen. Auf den SET-Operator folgen die spezifischen Veränderungen, die an dem Datenbestand durchgeführt werden. In diesem Fall das Ändern des Namens in „Meier“ die Postleitzahl in „123456“ und die Stadt in „Neustadt“. Die Bedingung definiert, welche Einträge aktualisiert werden sollen. In diesem Beispiel sind das alle Einträge der Mitarbeiter mit der Mitarbeiternummer 5. Wenn die Bedingung entfällt, werden alle Einträge entsprechend des SET-Befehls angepasst (Bühler 2019). Die Komplexität kann beliebig gesteigert werden, sodass auch Unterabfragen oder weitere Funktionen in das UPDATE-Statement aufgenommen werden können. Ein Beispiel mit einer Unterabfrage lautet:

Algorithmus 18: Beispiel eines verschachtelten UPDATE-Befehls

```
1. UPDATE Mitarbeiter
2. SET Mitarbeiternummer = 10
3. WHERE Id =(SELECT Id
4. FROM Personalkatalog WHERE Name = 'Meier'');
```

Hierdurch lassen sich Informationen aus verschiedenen Tabellen beliebig zusammentragen und in neue Tabellen überschreiben. Dies ist vor allem sinnvoll, wenn sich grundlegende Informationen innerhalb einer Tabelle ändern und diese in weitere Tabellen übertragen werden sollen. Durch Einbeziehen weiterer bisher betrachteter Befehle, wie zum Beispiel den Bedingungen oder Mengenoperatoren ermöglicht SQL eine präzise Steuerung der anzupassenden Datensätze und ein mächtiges Werkzeug für die Datenmanipulation. Der DELETE-Befehl ermöglicht das Löschen von Datensätzen unter der Voraussetzung der entsprechenden Zugriffsrechte für die Durchführung des Löschvorgangs (Simon 2023). Dabei können einzelne Datensätze oder auch ganze Tabellen gelöscht werden. Die Ergänzung des DELETE-Befehls um Bedingungen ermöglicht die gezielte Entfernung einzelner Dateneinträge (Mielebacher 2024). Das folgende Beispiel umfasst die Löschung von Dateneinträgen auf Basis von definierten Bedingungen:

Algorithmus 19: Beispiel eines DELETE-Befehls (in Anlehnung an Mielebacher 2024, S. 133)

```
1. DELETE FROM Mitarbeiter
2. WHERE Mitarbeiternummer = 5;
```

Ähnlich zu dem SELECT-Befehl folgt nach dem Bezeichner „DELETE“ ein FROM-Operator und die entsprechende Tabelle, aus der die Daten entfernt werden sollen. Die Bedingung wird zuletzt definiert und prüft in diesem Fall auf die Mitarbeiternummer 5. Somit werden alle Einträge zu dem Mitarbeiter mit der entsprechenden Mitarbeiternummer dauerhaft aus der Datenbank gelöscht. Neben diesem Beispiel besteht auch die Möglichkeit die Bedingungen umfassender zu definieren und somit mehr Dateneinträge zu löschen. Ein Beispiel wäre die Schließung eines Werkes in einer bestimmten Stadt und die Entlassung der entsprechenden Mitarbeiter an dem Standort. Mithilfe des DELETE-Befehls und der Prüfung auf den entsprechenden Standorteintrag können alle Einträge der Mitarbeiter des entsprechenden Standortes gleichzeitig gelöscht werden. Bei der Verwendung des DELETE-Befehls, insbesondere in Produktionsdatenbanken, ist Vorsicht geboten, um die referentielle Integrität zu wahren (Bronselaer et al. 2015). Um Fehler zu minimieren, ist es ratsam, zunächst einen SELECT-Befehl zu verwenden, um die Daten zu überprüfen, bevor sie in einen DELETE-Befehl umgewandelt werden (Kellenberger und Everest 2021)

2.4 Python zur Entwicklung einer datengetriebenen Applikation und Zugriffsberechtigungen in Datenbanken

Python wird als eine leicht zu erlernende Sprache angesehen, insbesondere für Anfänger (Hunt 2019). Die einfache Syntax führt im Vergleich zu komplexeren Sprachen wie Java zu weniger Fehlern (Mannila et al. 2006). Je weiter die Lernenden jedoch fortschreiten, desto schwieriger kann es werden, sich in dem umfangreichen Ökosystem der Bibliotheken und Werkzeuge von Python zurechtzufinden (Nelli 2015). Die Einfachheit der Sprache kann für die Vermittlung von Programmierkonzepten von Vorteil sein, da sie sich eng an das mathematische Denken anlehnt und an zahlreichen führenden Universitäten verwendet wird (Bogdanchikov et al. 2013). Studien haben gezeigt, dass das Erlernen von Python den Übergang zu fortgeschritteneren Sprachen wie Java nicht behindert (Mannila et al. 2006). Während die Einfachheit von Python für Anfänger von Vorteil ist, kann die Beherrschung der fortgeschrittenen Funktionen und das Verständnis des umfangreichen Ökosystems überwältigend sein (Hunt 2019). Diese Komplexität unterstreicht den Bedarf an strukturierten Lernpfaden und einer Anleitung zur Auswahl geeigneter Bibliotheken für bestimmte Anwendungen. Innerhalb dieser Ausarbeitung wird der Fokus innerhalb der methodischen Grundlagen in Verbindung mit Python auf die Erstellung einer grafischen Benutzeroberfläche (GUI) gelegt. GUIs haben die In-

teraktion zwischen Menschen und Computern revolutioniert, indem sie komplexe Befehlssprachen durch visuelle Elemente ersetzt haben, die der Benutzer direkt bedienen kann (Herman und Aburdene 1991). GUIs ermöglichen es den Nutzern, Befehle durch Mausklicks und Tastendruck auszuführen, was die Software benutzerfreundlicher macht, und Fehler reduziert (Basak und Roy 2019). Das grundlegende Problem einer GUI ist die fehlende Kompatibilität über verschiedene Plattformen hinweg, sodass im Zweifel unterschiedliche GUIs für unterschiedliche Plattformen definiert werden müssen (Moruzzi 2020). Dieses Problem wird von der Standard-GUI von Python durch eine Kompatibilität mit den Betriebssystemen Linux, Windows und macOS nahezu beseitigt (Thanasis Stamos 2007). Die Standard-GUI heißt Tkinter und wurde erstmals im Jahr 1991 präsentiert (Moruzzi 2020). Wie bereits als eine der Herausforderungen von Python erwähnt, gibt es auch für diese GUI-Bibliothek zahlreiche Alternativen. Dennoch wird sich innerhalb dieser Ausarbeitung lediglich auf die Tkinter-Bibliothek bezogen, da diese die meisten Funktionen enthält und im Gegensatz zu den anderen Bibliotheken mehr Plattformen unterstützt.

Tkinter bietet eine Reihe von vordefinierten Widgets wie Schaltflächen, Beschriftungen und Kontrollkästchen, die leicht in GUI-Anwendungen implementiert werden können (Elumalai 2021). Die ereignisgesteuerte Architektur von Tkinter ermöglicht die Erstellung interaktiver Schnittstellen, bei denen Aktionen durch Benutzerinteraktionen wie Schaltflächenklicks oder Mausbewegungen ausgelöst werden (Elumalai 2021). Die einzelnen Schaltflächen und Bausteine in Tkinter sind in Objekte und Klassen unterteilt. Damit ein grundlegendes Fenster erstellt wird muss zunächst ein Tk-Objekt instaziiert werden (Charatan und Kans 2022). Das Fenster bildet dann die Grundlage für alle weiteren Bausteine, welche sich innerhalb dieses Fensters befinden werden. Dennoch besteht die Möglichkeit neben dem Hauptfenster, weitere Fenster zu erstellen, welche nicht innerhalb des Hauptfensters angezeigt werden. Dies ist möglich, indem die Klasse „TopLevel“ verwendet wird (Charatan und Kans 2022). Die Bibliothek bietet die drei Hauptgeometriemanager „pack()“, „grid()“ und „place()“ für die Organisation von Widgets in Fenstern an (Hunt 2023). Je nach angestrebter Strategie ist eine Funktion sinnvoller als die anderen. Das Fenster ist nicht starr und kann regelmäßig durch das Senden von Nachrichten manipuliert und erneuert werden. Dadurch lässt sich der Zustand des Fensters ändern, eine bestimmte Operation in dem Fenster durchführen oder ein grafisches Objekt anzeigen. Es besteht auch die Möglichkeit die Komponenten unterschiedlicher Fenster miteinander zu verbinden und bei der Aktion innerhalb eines Fensters eine Reaktion innerhalb eines weiteren Fensters hervorzurufen (Hunt 2023). Die folgenden Begriffe und Klassen sind für die weitere Ausarbeitung wesentlich und wurden teilweise bereits erläutert, darunter die Tk-Klasse, die Toplevel-Klasse, die Frame-Klasse, Widgets und die Canvas-Klasse. Diese Elemente spielen eine zentrale Rolle bei der Entwicklung der Benutzeroberfläche und werden im weiteren Verlauf der Arbeit genauer betrachtet. Die Tk-Klasse bildet das Haupt- oder auch Stammfenster der gesamten Anwendung. Es gilt als Hauptcontainer und ist verantwortlich für die Verknüpfung aller weiteren Klassen und die Kontrolle und Weiterleitung von ein- und ausgehenden Ereignissen (Hunt 2023). Die TopLevel-Klasse ist für die Erstellung und Handhabung zusätzlicher Fenster neben dem Hauptfenster zuständig. Die Fenster können als weitere Dialogfelder, Menüfenster oder Pop-up-Fenster genutzt werden. Die Frame-Klasse dient lediglich zur Erstellung von Rahmen innerhalb von Fenstern. Die einzige Aufgabe der Klasse besteht darin, die Widgets und Fenster zu strukturieren und zu organisieren (Elumalai 2021). Neben den bereits erwähnten Widgets gibt es noch die Canvas-Klasse, welche die Möglichkeit bietet, Freihandzeichnungen innerhalb der GUI durch den Endanwender erstellen zu lassen (Hunt 2023). Die Umsetzung einer einfachen GUI sieht mittels Tkinter wie folgt aus:

Algorithmus 20: Erstellung einer Benutzeroberfläche mit Tkinter

```
1. from tkinter import Tk, Label
2. root = Tk()
3. label = Label(root,
4. text='Hello world')
5. label.pack()
```

```
6. root.mainloop()
```

Mit diesem Programmcode wird eine Nachricht mit dem Inhalt „Hello world“ innerhalb eines Fensters angezeigt. Die wenigen Zeilen Programmcode sollen hervorheben, wie simpel die Erstellung einer GUI mittels Tkinter ist. Innerhalb von Abschnitt 4.1 wird auf diesem Einstieg in Tkinter aufgebaut, um komplexere Benutzeroberflächen und die Verknüpfung von ereignisgesteuerten Anwendungen umzusetzen.

Die Kontrolle des Datenbankzugriffs ist in Mehrbenutzerumgebungen von entscheidender Bedeutung, wobei verschiedene Ansätze in unterschiedlichen DBMS implementiert sind. SQLite, das wegen seiner Effizienz weit verbreitet ist, verfügt über keine integrierte Mehrbenutzer-Zugriffskontrolle, was die Entwicklung von SeSQLite veranlasste, eine obligatorische Zugriffskontrolle (Mutti et al. 2015). Während herkömmliche SQL-Datenbanken robuste Sicherheitsfunktionen bieten, fehlt es NoSQL-Datenbanken, die für unstrukturierte Daten entwickelt wurden, oft an fortschrittlichen Autorisierungs- und Zugriffskontrollfunktionen (Mohamed et al. 2022). Zu den Zugriffskontrollmodellen gehören diskretionäre, obligatorische und rollenbasierte Richtlinien, wobei objektorientierte Datenbanken aufgrund ihrer einzigartigen Struktur spezielle Ansätze erfordern (Ahmad 1996). Um den Schutz von Datenbanken zu verbessern, wird ein mehrschichtiges Autorisierungssystem vorgeschlagen, das den Zugriff in die Schichten Nur-Lese-, Bearbeiter- und Administratorrechte aufteilt, wobei jede nachfolgende Schicht auf der vorherigen aufbaut (Voitovych et al. 2018). Diese Weiterentwicklungen zielen darauf ab, umfassende Sicherheitslösungen für verschiedene Datenbankmodelle und Anwendungsfälle bereitzustellen. Ein ähnliches Modell findet sich bei Microsoft Access, wo der Schutz der Datenbank typischerweise durch ein einheitliches Passwort erfolgt. Die Verwaltung von Benutzerkonten und deren Berechtigungen in einem Datenbanksystem wird üblicherweise durch einen Administrator durchgeführt (Elmasri und Navathe 2016). Aufgrund der umfangreichen Rechte, die Administratorkonten innehaben, ist es von kritischer Bedeutung, dass diese Konten durch angemessene Sicherheitsmaßnahmen geschützt werden, um das Risiko eines unbefugten Zugriffs zu minimieren und die systemweite Sicherheit zu erhöhen (Mielebacher 2024). Eine effiziente Methode zur Verwaltung von Berechtigungen in Datenbanksystemen ist die Implementierung von Berechtigungsstufen (Tsolkas 2017). Diese Stufen kategorisieren die verfügbaren Berechtigungen in ein Spektrum, das von „keine Berechtigung“ bis hin zu „vollständigen Berechtigungen“ reicht (Tsolkas 2017). Durch diese graduelle Abstufung wird der Prozess der Berechtigungszuweisung und -verwaltung erheblich vereinfacht, indem eine klare und strukturierte Hierarchie geschaffen wird, die die Zuordnung und Überprüfung von Zugriffsrechten erleichtert. Ein wesentlicher Fokus moderner DBMS liegt auf dem Schutz sensibler Informationen, wie zum Beispiel Personaldaten (Schicker 2017). Um die Sicherheit dieser Daten zu gewährleisten, implementieren verschiedene DBMS spezifische Zugangsberechtigungen, die den Zugriff auf diese sensiblen Informationen streng reglementieren (Schicker 2017). Die Verwaltung von Benutzeridentifikationen und Passwörtern durch das DBMS spielt dabei eine zentrale Rolle, da nur Benutzer mit entsprechenden Autorisierungen Zugriff auf die freigegebenen Segmente der Datenbank erhalten. Diese strikte Zugriffskontrolle ist entscheidend, um die Vertraulichkeit und Integrität der in der Datenbank gespeicherten sensiblen Daten zu sichern.

3 Konzeption und Umsetzung des Datenbanksystems und Analyse der Automatisierungspotenziale

In diesem Kapitel werden die Konzeption und praktische Umsetzung des Datenbanksystems detailliert beschrieben. Zunächst wird die funktionale Trennung des Datenbanksystems und der Datentabellen erläutert, um eine klare Struktur und hohe Effizienz zu gewährleisten. Im Weiteren werden die Methoden zur Erstellung von Datenabfragen mithilfe von SQL vorgestellt, die essenziell für die Datenanalyse und -verarbeitung sind. Ein zentraler Aspekt dieses Kapitels ist die Darstellung der bestehenden Prozesse mithilfe von Flussdiagrammen innerhalb des Datenbanksystems. Dabei wird besonderes Augenmerk auf die Identifikation und Nutzung von Automatisierungspotenzialen gelegt, um die Effizienz und Genauigkeit des Systems zu steigern.

3.1 Funktionale Trennung des Datenbanksystems und Restrukturierung der Datentabellen

Im Zuge dieser Ausarbeitung werden verschiedene Akteure innerhalb des Datenbanksystems behandelt und vor allem die Daten der externen und internen Studenten sind am wesentlichsten. Damit die Einheitlichkeit der Bezeichnung dieser Akteure in dieser Ausarbeitung und in den entworfenen Datenbankkomponenten gegeben ist werden Studenten ausländischer Universitäten als „Incomings“ und Studenten der Universität TU Dortmund als „Outgoings“ bezeichnet. Das vorliegende Datenbanksystem besteht zu Beginn lediglich aus einer Access-Datenbank. In dieser Datenbank liegen alle Stammdaten der einzelnen Studenten sowie weiterer Akteure des Projektes, wie zum Beispiel der Mentoren oder Partneruniversitäten. In derselben Datenbank sind alle Datenabfragen definiert und die Masken zur manuellen Eingabe von Daten. Der Status Quo ist, dass die vorliegende Datenbank von mehreren Endanwendern an einem Lehrstuhl gleichzeitig verwendet wird oder eine Kopie der Datenbank lokal angefertigt und diese bearbeitet wird. Dadurch entsteht das Risiko der doppelten Datenhaltung, sodass keine einheitliche Datenhaltung gewährleistet ist. Zudem können Tabelleneinträge ungehindert gelöscht oder Abfragen manipuliert werden.

Im Hinblick auf diese Risiken und Ineffizienzen wird die Datenbank in eine Frontend-Datenbank und eine Backend-Datenbank aufgeteilt. Die Backend-Datenbank bildet das Fundament des gesamten Datenbanksystems und umfasst alle Tabellen, die die Daten der einzelnen Akteure enthalten. Die Backend-Datenbank soll in einem freigegebenen Verzeichnis auf dem internen Server gespeichert werden, um eine gleichzeitige Nutzung durch mehrere Benutzer zu ermöglichen. Dennoch sollen die Daten nur von ausgewählten Anwendern angepasst oder eingesehen werden. Deshalb wird in Kapitel 5.1 ein Zugriffskonzept entworfen, sodass gewährleistet ist, dass nur ausgewählte Benutzer die Daten manipulieren können. Die Frontend-Datenbank umfasst alle SQL-Abfragen und greift auf die Daten im Backend zu und verarbeitet diese und bereitet sie für den Nutzer auf. Hierbei werden keine Daten in der Backend-Datenbank angepasst, sie werden lediglich abgerufen. Die Frontend-Datenbank kann ebenfalls gemeinsam genutzt werden. Bei Bedarf an individuellen Anpassungen der Abfragen kann die Frontend-Datenbank lokal kopiert und in der Kopie modifiziert werden. Dadurch wird sichergestellt, dass die Frontend-Datenbank im freigegebenen Verzeichnis stets den aktuellen und korrekten Datenbestand widerspiegelt und die Abfragen funktionsfähig sind. Für die Aufteilung der Datenbank bietet Microsoft Access eine Funktion zur Trennung der Daten von den Abfragen an. Dadurch lässt sich eine bereits bestehende Datenbank effizient aufteilen, sodass die Abfragen nach der Auftrennung immer noch mit den entsprechenden Datenquellen verbunden sind. Hierdurch entfällt die Aufgabe eine Datenbrücke zwischen Frontend- und Backend-Datenbank herzustellen. Neben der funktionalen Trennung des Gesamtsystems ist vor allem eine Aufteilung und Restrukturierung der Datentabellen für eine bessere Nachvollziehbarkeit und höhere Effizienz essenziell. Ein Großteil der vorhandenen Tabellen ist mit Daten überladen,

wodurch erhebliches Verbesserungspotenzial in der Aufteilung dieser Tabellen in Untertabellen besteht. Dafür ist zunächst eine Namenskonvention zu definieren, um eine einheitliche Bezeichnung zu gewährleisten und sicherzustellen, dass auch zukünftig weitere Tabellen dieser Konvention folgen. Die Namenskonvention der Tabellen ist in Abbildung 3 dargestellt, aus der hervorgeht, dass jeder Tabellenname aus drei Teilen besteht. In diesem Fall ist der Anfang der Namenskonvention stets gleich, da im Rahmen dieser Ausarbeitung nur das ERASMUS-Projekt behandelt wird. Bei der Einbindung weiterer Projekte in die Datenbankumgebung kann der Projektpräfix jedoch ausgetauscht werden. Auf die Projektbezeichnung folgt der Name des entsprechenden Akteurs, dessen Daten in der vorliegenden Tabelle erfasst werden. Neben den beiden bereits erwähnten Akteuren „Incomings“ und „Outgoings“ gibt es noch die Partneruniversitäten, Mentoren und Lehrstühle als weitere Akteure. Die Namenskonvention wird abschließend durch den entsprechenden Datentypus spezifiziert. Der Datentypus gibt die inhaltliche Bedeutung der Daten an, wobei beispielsweise Stammdaten alle grundlegenden Informationen der Akteure wie Namensbezeichnungen, Geburtsorte oder Kontaktinformationen umfassen. Für jeden Akteur existiert eine Stammdatentabelle. Darüber hinaus werden individuelle Tabellen erstellt, welche spezifische Informationen enthalten, die für die einzelnen Akteure in den Prozessen benötigt werden.

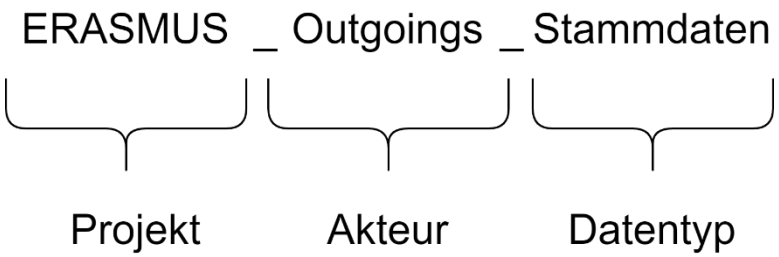


Abbildung 3: Namenskonvention der Backend-Tabellen

Wie in Abbildung 4 dargestellt, wird die derzeit existierende Tabelle „ERASMUS_Bewerbungen“, die zunächst als einzelne Tabelle vorliegt, im Rahmen der Restrukturierung in drei separate Untertabellen aufgeteilt. Die Untertabellen beinhalten Stammdaten, Bewerbungsdaten und Anerkennungsdaten. Die Bewerbungsdaten enthalten alle Daten, welche sich auf die gewünschte Partneruniversität, die individuellen Leistungen der Studenten und deren Prioritäten beziehen. Die Anerkennungsdaten enthalten wesentliche Informationen über die abgelegten Prüfungen der Studierenden an den Partneruniversitäten und sind deshalb vor allem nach einer erfolgreichen Nominierung relevant.

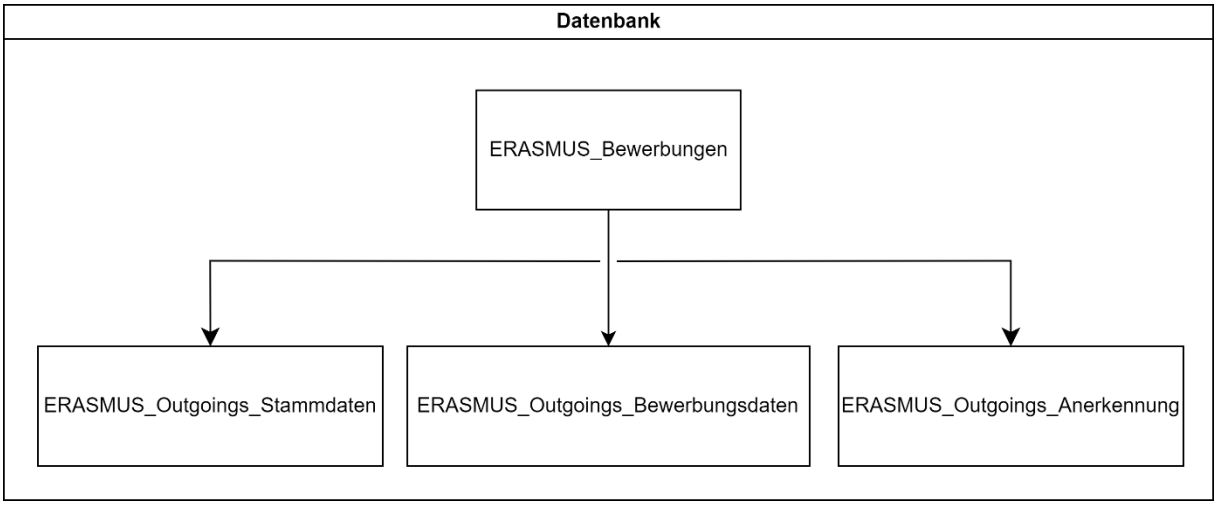


Abbildung 4: Auftrennung der ursprünglichen Tabelle „ERASMUS_Bewerbungen“

Die Tabelle, die Informationen über Incomings speichert, wird derzeit als „ERASMUS_Incomings“ bezeichnet. Die Tabelle ist analog zur bereits durchgeführten Aufteilung der anderen Tabelle zu restrukturieren und gemäß den festgelegten Namenskonventionen umzubenennen. Dabei erfolgt die Trennung der Daten in entsprechende Untertabellen, um eine konsistente und prozesskonforme Struktur zu gewährleisten. Neben einer Stammdatentabelle für die grundlegenden Informationen über die Incomings sind zwei weitere Tabellen in Abbildung 5 dargestellt. Analog zu den Tabellen der Outgoings werden Daten über die jeweilige Bewerbung in einer Bewerbungstabelle abgespeichert. Daten über abgelegte Prüfungen und Noten werden in der ToR-Tabelle gespeichert. Die Abkürzung ToR steht in diesem Fall für Transcript of Records (ToR), eine Dokumentation aller abgelegten Prüfungen. Die Daten der Incomings lassen sich durch die geringe Datenmenge und Anzahl an Studierenden ebenfalls in lediglich einer oder zwei Tabellen übersichtlich ablegen. Dennoch sollte ein Konzept in jedem Fall durchgängig verfolgt werden, sodass es nicht zu Komplikationen bei der Erweiterung der Systeme kommt. Zudem besteht die Möglichkeit, dass die Anzahl an Incomings in Zukunft rasant steigen wird, sodass eine Restrukturierung bereits eine optimale Basis für diese Skalierung bietet.

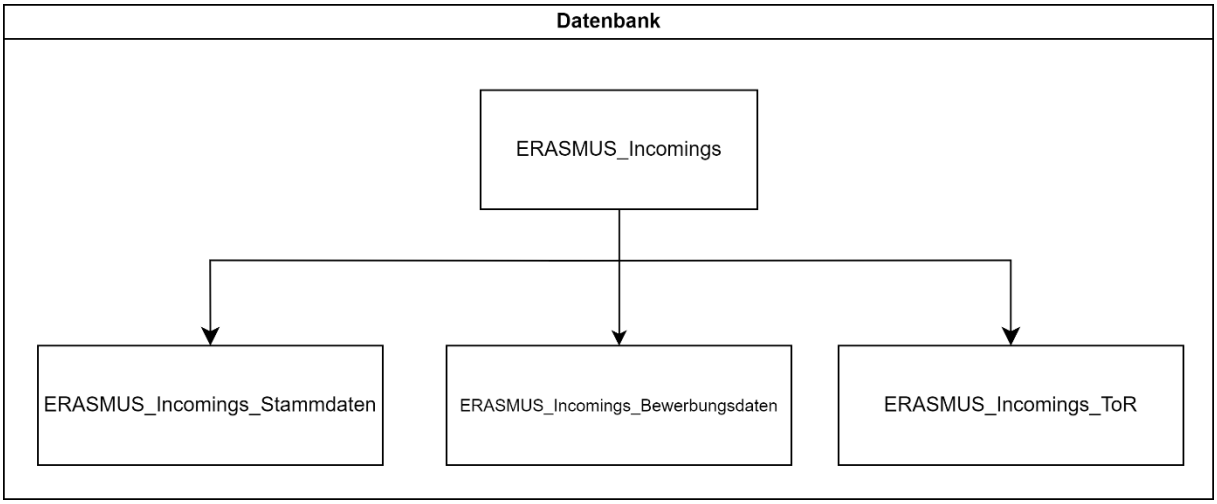


Abbildung 5: Auftrennung der ursprünglichen Tabelle „ERASMUS_Incomings“

Für die Daten der Lehrstühle liegen keine Daten über die Stammdaten hinaus vor, sodass für die Lehrstühle nur eine Tabelle bestehen bleibt, welche gemäß der bereits definierten Namenskonvention angepasst wird. Die Aufteilung der derzeit bestehenden Tabelle „ERASMUS_Partner“ in zwei Einzeltabellen bietet sich durch eine Vielzahl an Datenattributen an. Eine Übersicht der Umbenennung und Erstellung der Tabellen ist in Abbildung 6 dargestellt. Für den Akteur „Mentor“ existiert bisher keine Tabelle, sodass hierfür erstmalig zwei Tabellen angelegt werden, eine für die Stammdaten und eine für die Einsatzdaten. Die Einsatzdaten umfassen alle Daten über das eingesetzte Jahr der Mentoren, den zugeordneten Incoming in diesem Jahr und weitere Daten, welche sich je Einsatz voneinander unterscheiden können.

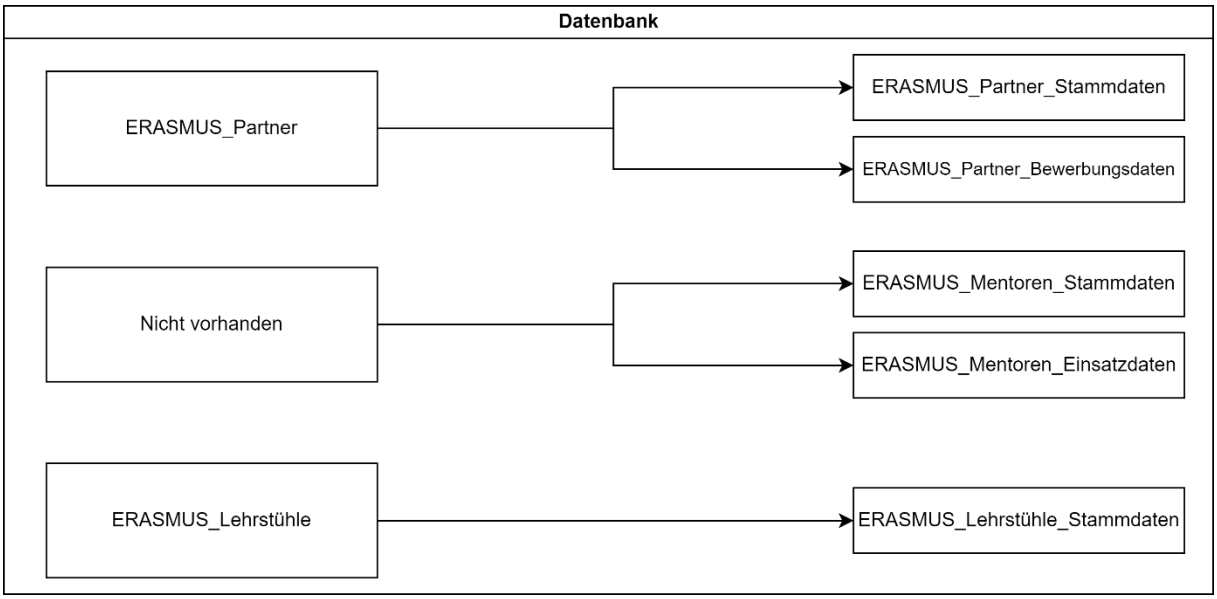


Abbildung 6: Auftrennung der ursprünglichen Tabellen

Nachdem die Grundstruktur der Tabellen und somit die Basis des Datensystems restrukturiert sind, liegt der nächste Fokus in der Betrachtung und Bearbeitung der vorliegenden Primär- und Fremdschlüssel. Wie bereits in Abschnitt 2.2 erläutert, sind eine klare Definition und die optimale Auswahl eines geeigneten Schlüssels von höchster Wichtigkeit. In dem hier betrachteten Projekt besteht ein akutes Problem in der Mehrfachbewerbung von Studierenden. Dadurch entstehen in der Datenbank Mehrfacheinträge, welche ohne klare Definition der Schlüssel zu weitgreifenden Folgeproblemen führen können. In Abbildung 7 ist ein Ausschnitt der Tabelle „ERASMUS_Outgoings_Stammdaten“ dargestellt. Dieser Ausschnitt umfasst die Daten, welche für die Identifizierung des jeweiligen Outgoings derzeit genutzt werden. Der Primärschlüssel der Tabelle ist in Abbildung 7 mit einem Schlüsselsymbol markiert, in diesem Fall die Spalte Matrikelnummer. Der Primärschlüssel muss in jedem Fall eindeutig sein, da jeder Student nur eine Matrikelnummer hat. Daher erscheint es naheliegend, die Matrikelnummer als Primärschlüssel zu verwenden. Wie jedoch in Abbildung 7 ersichtlich ist, besteht das Risiko einer Verletzung der referenziellen Integrität, wenn sich die Stammdaten des Studierenden ändern. Es soll jedoch gewährleistet sein, dass auch vergangene Daten der Studierenden gespeichert werden. Aus diesem Grund wird derzeit an die Matrikelnummer die Ziffer 0 angehängt, um die Möglichkeit zu schaffen, die Daten eines Studenten mehrfach zu erfassen. Dies führt jedoch dazu, dass die eindeutige Zuordnung eines Primärschlüssels zu einem Studenten verloren geht und das Prinzip der Datenkonsistenz, wie in Abschnitt 2.2 beschrieben, verletzt wird. Des Weiteren unterscheiden sich die beiden Datensätze in den identifizierenden Merkmalen nicht und lediglich weitere Ausprägungen, wie die Partneruniversität oder das akademische Jahr haben sich bei der erneuten Bewerbung verändert. Der in Abschnitt 2.2 erläuterte kombinierte Schlüssel kann hier verwendet werden, um eine eindeutige Beziehung zwischen einem Outgoing und mehreren Dateneinträgen zu erzielen. Der kombinierte Schlüssel kann lediglich innerhalb der Abfragen in der Frontend-Datenbank und bei der Dateneinpflegung in die Dokumentenvorlagen verwendet werden. Damit die referenzielle Integrität in der Backend-Datenbank gewährt ist muss immer noch ein eindeutiger Primärschlüssel gewählt und in den Untertabellen referenziert werden. Die bisher gewählte Information „Zeitraum“, welche festhält, ob der Student im Winter- oder Sommersemester sein Auslandssemester antritt, ist in Kombination mit der Matrikelnummer ebenfalls nicht eindeutig.

ERASMUS_Outgoings_Stammdaten			
Matrikelnummer	**Nachname**	**Vorname**	**Zeitraum**
883217	Mustermann	Max	WS
8832170	Mustermann	Max	WS

Abbildung 7: Ursprüngliche Struktur der Stammdatentabelle der Outgoings

Abbildung 8 zeigt die neue Struktur der Tabelle „ERASMUS_Outgoings_Stammdaten“, welche nun gemäß der Bezeichnung nur noch Stamminformationen der Outgoings beinhaltet. Dies bedeutet, dass alle Informationen über den Zeitraum des Auslandsaufenthaltes, das akademische Jahr, die Partneruniversität und weitere Merkmale, welche sich innerhalb einer weiteren Bewerbung verändern können, nicht mehr innerhalb der Stammdatentabelle gespeichert werden. Somit existiert für jeden Outgoing exakt eine Relation innerhalb der Stammdaten. Für den Fall, dass sich Informationen bei einer neuen Bewerbung innerhalb der Stammdaten ändern, wie zum Beispiel der Nachname, dann wird keine neue Relation angelegt, sondern der bestehende Eintrag des Outgoing überschrieben. Welchen Einfluss diese Änderung auf die bereits existierenden Relationen des Studierenden in anderen Tabellen hat wird zu einem späteren Zeitpunkt betrachtet. Der Primärschlüssel ist immer noch die Matrikelnummer des Outgoing, aber gilt nur innerhalb der Stammdaten als Primärschlüssel dieses Akteurs. Damit ist der Schlüssel für jede Relation eindeutig. Das Attribut „Zeitraum“ wird durch die Restrukturierung gemäß Abbildung 9 in die Bewerbungs- und Anerkennungstabelle umgelagert und in „Zeitpunkt“ umbenannt. Das akademische Jahr ist in diesem Fall nicht als einziges weiteres Identifizierungsmerkmal zu wählen, da ein Outgoing sich innerhalb eines akademischen Jahres zwei Mal bewerben kann. Doch die Kombination aus dem akademischen Jahr und dem Zeitpunkt ermöglicht eine eindeutige Zuordnung der Outgoings zu jedem Zeitpunkt, obwohl diese nun mehrere Einträge in den Untertabellen mit derselben Matrikelnummer haben können. Dieselbe Restrukturierung der Schlüssel wird auch für die Stammdatentabelle und die weiteren Untertabellen der Incomings durchgeführt.

ERASMUS_Outgoings_Stammdaten			
Matrikelnummer	**Nachname**	**Vorname**	**Geschlecht**
883217	Mustermann	Max	männlich
739423	Musterfrau	Maya	weiblich

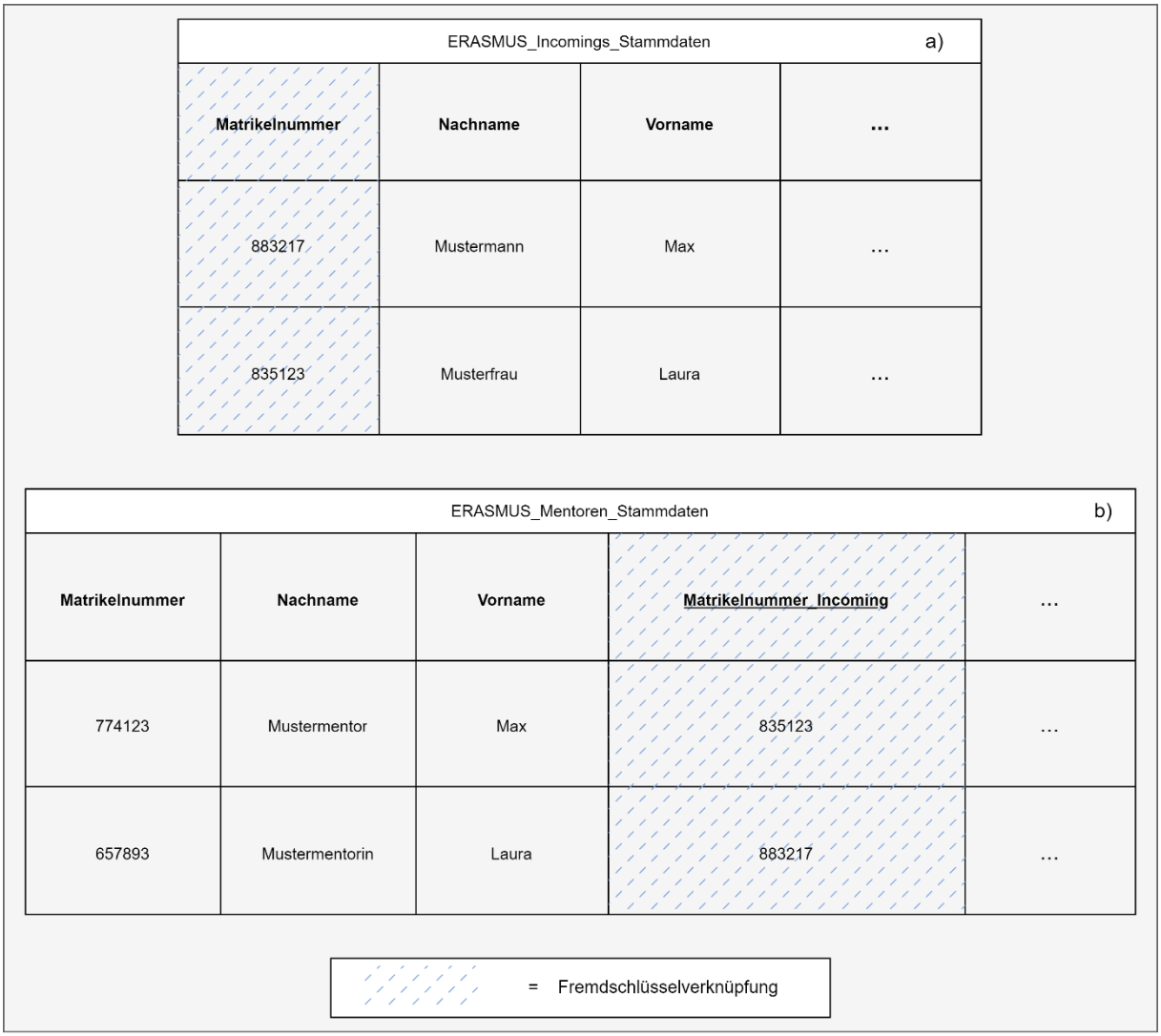
Abbildung 8: Restrukturierte Stammdatentabelle der Outgoings

ERASMUS_Outgoings_Bewerbungsdaten			
ID	**Matrikelnummer**	**Zeitpunkt**	**Akadem. Jahr**
1	883217	WiSe	2022/23
2	883217	SoSe	2024/25

Abbildung 9: Neue Bewerbungsdatentabelle der Outgoings

Der in Abschnitt 2.2 beschriebene Fremdschlüssel wird im Kontext des Mentorenmappings verwendet, um die Zuordnung der Incomings zu ihren Mentoren zu ermöglichen. Wie in Abbildung 10 dargestellt, wird das identifizierende Merkmal „Matrikelnummer“ der Incomings in die Stammdaten der Mentoren übertragen, um eine eindeutige Beziehung zwischen den beiden Gruppen herzustellen. In Abbildung 10 deuten die blau hervorgehobenen Felder in Tabelle a) auf die Übertragung des Fremdschlüssels „Matrikelnummer“ in die Stammdaten der Incomings hin, während in Tabelle b) die Matrikelnummer in den Stammdaten der Mentoren ergänzt wird. Diese Ergänzung stellt sicher, dass eine eindeutige relationale Verbindung zwischen den Incomings und ihren Mentoren etabliert wird. Die Abbildung verdeutlicht, dass es sich hierbei um eine 1:1-Beziehung handelt, da jedem Incoming genau ein Mentor zugewiesen ist und umgekehrt. Dies wird in Tabelle a) durch die Zuordnung der Matrikelnummer in den Incomings-Stammdaten und in Tabelle b) durch die Verknüpfung der Matrikelnummer in den Stammdaten

der Mentoren visualisiert. Die farbliche Hervorhebung in beiden Tabellen hebt dabei die Felder hervor, die für die Herstellung dieser relationalen Verknüpfung entscheidend sind.



ERASMUS_Incomings_Stammdaten a)

Matrikelnummer	Nachname	Vorname	…
883217	Mustermann	Max	…
835123	Musterfrau	Laura	…

ERASMUS_Mentoren_Stammdaten b)

Matrikelnummer	Nachname	Vorname	Matrikelnummer_Incoming	…
774123	Mustermentor	Max	835123	…
657893	Mustermentorin	Laura	883217	…

= Fremdschlüsselverknüpfung

Abbildung 10: Fremdschlüsselverknüpfung von Tabellen a) „ERASMUS_Incomings_Stammdaten“-Tabelle b) „ERASMUS_Mentoren_Stammdaten“-Tabelle

3.2 Erstellung und Umformung von Datenabfragen mithilfe von SQL

Das Datenbanksystem entfaltet seinen zentralen Nutzen durch die Erstellung von SQL-Abfragen, die Daten aus den Datentabellen der Backend-Datenbank abrufen, gemäß der SQL-Logik transformieren und für den Nutzer aufbereiten. Dabei können Beziehungen zwischen den Tabellen hergestellt und Berechnungen auf der Grundlage der Backend-Daten durchgeführt werden. Aufgrund der Vielzahl von Akteuren und der bereits langen Laufzeit des Projektes ist die Anzahl der individuellen Abfragen hoch und steigt jedes Semester weiter an. Innerhalb der bestehenden Datenbank existiert bereits eine Vielzahl von Abfragen, von denen ein Großteil nicht den gewünschten Anforderungen entspricht und einige Abfragen redundant sind. Trotzdem gibt es neue Anforderungen an das Projekt und neue Prozesse, die hinzugefügt werden müssen, was die Erstellung neuer Abfragen erforderlich macht. Ähnlich wie bei den Tabellen wird auch hier eine Namenskonvention für die Abfragen festgelegt, die im Gegensatz zu den Namenskonventionen für Tabellen nicht zwingend eingehalten werden muss. Da die Frontend-Datenbank mit den Abfragen dupliziert und angepasst werden kann, ist eine Umbenennung

der Abfragen nach eigenem Ermessen möglich und sogar erwünscht. In der Grundform beginnen die Abfragen jeweils mit dem Namen des entsprechenden Akteurs, gefolgt von Informationen, die den Nutzen der Abfrage hinreichend genau definieren. In Abbildung 11 sind die Abfragen im Zusammenhang mit ihrer jeweiligen Hauptfunktion aufgelistet. Grün markierte Abfragen werden neu erstellt, während alle anderen Abfragen umstrukturiert und durch weitere SQL-Befehle ergänzt werden.

SQL-Abfragen	
Abfragenbezeichnung	**Funktion**
Incomings Übersicht	Übersicht aller Incomings mit relevanten Daten
Incomings ToR erstellt	Status der ToR-Erstellung
Incomings zu Kursen	Mapping Incomings zu Kursen an Partneruni
Incomings zu Mentoren	Mapping Incomings zu Mentoren
Incomings zu Wohnraum	Mapping Incomings zu dem zugeteilten Wohnraum
Mentoren zu Incomings	Mentoren Übersicht mit den jeweiligen Incomings
Outgoings Übersicht	Übersicht aller Outgoings mit relevanten Daten
Outgoings alle Prioritäten und Bewerbungsdaten	Wahlprioritäten der Outgoings und Bewerbungsdaten
Outgoings anerkannte Kurse	Auflistung der anerkannten Kurse der Outgoings mit Infos zu Kursen
Outgoings Annahmeerklärung eingereicht	Status der Annahmeerklärung der Outgoings
Outgoings ECTS Übersicht	Outgoings ECTS pro Fach und Durchschnittszahlen
Outgoings Inforveranstaltung	Status der Teilnahem der Outgoings an Infoveranstaltung
Outgoings nach akad. Jahr und zuget. Hochschule	Outgoings gefiltert nach akad. Jahr und zuget. Hochschule
Outgoings nicht anerkannte Kurse	Übersicht Outgoings, welche Kurse nicht anerkannt bekommen haben
Outgoings Nominierung + Nominierungsfrist	Outgoings Nominierungsdatum und Nominierungsfrist der Partneruniversitäten
Partner Agreement Gültigkeit	Übersicht der Partneruniversitäten mit entsprechenden Daten der Gültigkeitsverträge
Partner empfohlene Studiengänge	Mapping Studiengänge zu Partneruniversitäten
Partner freie Plätze	Anzahl freier Plätze pro Partnerhochschule
Partner Kontakte	Ansprechpersonen der Partneruniversitäten
Partner Kurswahl	Übersicht Kurse an den Partneruniversitäten
Partner Nominierungsdeadlines	Übersicht Nominierungsdeadlines der Partneruniversitäten
Partner Sprachanforderungen	Übersicht Sprachanforderungen der Partneruniversitäten

= Noch nicht existierende Abfragen

Abbildung 11: SQL-Abfragen mit zugehöriger Funktionsbeschreibung

Aufgrund der Vielzahl an Abfragen wird in dieser Ausarbeitung nicht auf jede einzelne Abfrage im Detail eingegangen. Einige Abfragen werden exemplarisch behandelt und diskutiert. Eine

der bedeutendsten Abfragen ist die Abfrage „Outgoings alle Prioritäten und Bewerbungsdaten“. Diese Abfrage umfasst die Wahl der Partneruniversitäten, Informationen über den Status der Nominierung, das akademische Jahr, den Status der Teilnahme an Informationsveranstaltungen sowie weitere essenzielle Daten. Die Abfrage vereint persönliche Informationen der Studierenden mit deren Bewerbungsdaten, indem eine Verbindung zwischen den zwei Tabellen „ERASMUS_Outgoings_Stammdaten“ und „ERASMUS_Outgoings_Bewerbungsdaten“, hergestellt wird. Diese Verbindung wird in Abbildung 12 dargestellt und durch einen „INNER JOIN“ realisiert, der auf der Matrikelnummer als definierte Bedingung basiert. Die Abfrage beginnt mit der Spezifikation der auszuwählenden Spalten, die sowohl persönliche Informationen als auch Bewerbungsdetails der Studierenden umfassen und sieht wie folgt aus:

Algorithmus 21: SELECT-Befehl „Outgoings alle Prioritäten und Bewerbungsdaten“

```
1. SELECT ERASMUS_Outgoings_Stammdaten.Nachname,
2. ERASMUS_Outgoings_Stammdaten.Vorname,
3. ERASMUS_Outgoings_Stammdaten.Matrikelnummer,
4. ERASMUS_Outgoings_Bewerbungsdaten.Zeitpunkt,
5. ERASMUS_Outgoings_Bewerbungsdaten.[1_Prioritaet],
6. ERASMUS_Outgoings_Bewerbungsdaten.[2_Prioritaet],
7. ERASMUS_Outgoings_Bewerbungsdaten.[3_Prioritaet],
8. ERASMUS_Outgoings_Bewerbungsdaten.[Weiteres],
9. ERASMUS_Outgoings_Bewerbungsdaten.Sprachkenntnisse,
10.   ERASMUS_Outgoings_Bewerbungsdaten.[Bewerbungsdokumente_Kom-
   plett],
11.   ERASMUS_Outgoings_Bewerbungsdaten.Anmerkungen,
12.   ERASMUS_Outgoings_Bewerbungsdaten.[Zugeteilte_Hochschule],
13.   ERASMUS_Outgoings_Bewerbungsdaten.Annahmeerklaerung,
14.   ERASMUS_Outgoings_Bewerbungsdaten.[Bewerbung_Zurückgezogen],
15.   ERASMUS_Outgoings_Bewerbungsdaten.[Nominierung_Erfolgt],
16.   ERASMUS_Outgoings_Bewerbungsdaten.Infoveranstaltung_Teilge-
   nommen,
17.   ERASMUS_Outgoings_Bewerbungsdaten.[Datum_Infoveranstaltung],
18.   ERASMUS_Outgoings_Bewerbungsdaten.Akademisches_Jahr
```

Die Struktur des FROM-Teils der Abfrage zeigt, dass die Tabelle „ERASMUS_Outgoings_Stammdaten“ als Ausgangspunkt dient:

Algorithmus 22: FROM-Befehl „Outgoings alle Prioritäten und Bewerbungsdaten“

```
1. FROM ERASMUS_Outgoings_Stammdaten INNER JOIN
2. ERASMUS_Outgoings_Bewerbungsdaten ON
3. ERASMUS_Outgoings_Stammdaten.Matrikelnummer =
4. ERASMUS_Outgoings_Bewerbungsdaten.Matrikelnummer;
```

Der INNER JOIN verbindet diese Tabelle mit der Tabelle „ERASMUS_Outgoings_Bewerbungsdaten“ basierend auf der bereits genannten spezifischen Bedingung, der Matrikelnummer. Durch die Matrikelnummer gelingt eine eindeutige Verbindung der Basisdaten mit einer beliebigen Anzahl an Einträgen aus der Tabelle der Bewerbungsdaten. Dadurch wird sichergestellt, dass die Informationen korrekt zugeordnet werden und keine falschen Verbindungen zwischen den Daten entstehen. Das Ergebnis dieser Abfrage ist eine konsolidierte Übersicht, die die persönlichen Daten der Studierenden, ihre Präferenzen hinsichtlich der Partneruniversitäten, den Status der Nominierung und Teilnahme an Informationsveranstaltungen sowie weitere relevante Bewerbungsdetails umfasst. Dies ermöglicht eine umfassende Analyse und

Verwaltung der Daten der Outgoings und unterstützt die effiziente Organisation und Nachverfolgung der Bewerbungs- und Nominierungsprozesse innerhalb des Projekts.

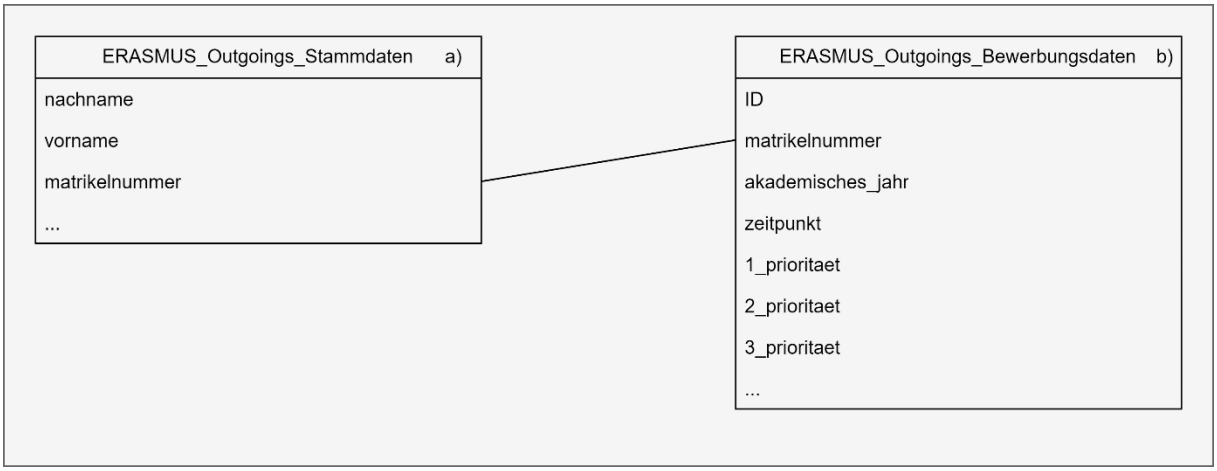


Abbildung 12: Visualisierung der Verknüpfung der Tabellen a) „ERASMUS_Outgoings_Stammdaten"-Tabelle und b) „ERASMUS_Outgoings_Bewerbungsdaten"-Tabelle

Die Verwendung eines „INNER JOIN" ist in diesem Kontext besonders geeignet, da nur die Datensätze berücksichtigt werden, die in beiden Tabellen vorhanden sind und die definierten Verknüpfungsbedingungen erfüllen. Dies reduziert das Risiko von Inkonsistenzen und stellt sicher, dass die resultierende Datenmenge sowohl vollständig als auch genau ist. Die Abfrage „Outgoings ECTS Übersicht" dient zur Berechnung von Anteilen und Bestehensquoten für Studierende im Rahmen des Austauschsemesters, basierend auf ihren anerkannten ECTS-Punkten. Diese Abfrage verwendet die Tabelle „ERASMUS_Outgoings_Anerkennungsdaten" und beinhaltet mehrere aggregierte Berechnungen sowie die Nutzung von IIf-Anweisungen zur Bestimmung spezifischer Werte. Zu Beginn der Abfrage werden die Matrikelnummer, der Zeitpunkt des Studienaufenthaltes und das akademische Jahr der Studierenden aus der Tabelle „ERASMUS_Outgoings_Anerkennungsdaten" ausgewählt. Diese Felder bilden die Grundlage für die Gruppierung der Daten mittels der GROUP BY-Klausel. Ein wesentlicher Bestandteil dieser Abfrage ist die Berechnung der Summe der anerkennungsfähigen ECTS-Punkte. Dies wird durch folgende Anweisung erreicht, welche die Gesamtanzahl der ECTS-Punkte aggregiert, die potenziell anerkannt werden können:

Algorithmus 23: Berechnung der ECTS-Summe der Outgoings

```
1. Sum(ERASMUS_Outgoings_Anerkennungsdaten.[ects_tu_modul])
2. AS GesamtEctsAnerkennungsfähig
```

Darüber hinaus berechnet die Abfrage die Summe der tatsächlich anerkannten ECTS-Punkte. Hierfür wird folgende IIf-Anweisung verwendet:

Algorithmus 24: Berechnung der tatsächlich anerkannten ECTS-Summe der Outgoings

```
1. Sum(IIf(ERASMUS_Outgoings_Anerkennungsdaten.anerkannt=True,
2. ERASMUS_Outgoings_Anerkennungsdaten.[ects_tu_modul], 0))
3. AS GesamtEctsAnerkannt
```

Diese Anweisung prüft, ob das Feld „Anerkannt“ den Wahrheitswert „True“ aufweist. Falls dies zutrifft, werden die entsprechenden ECTS-Punkte summiert, andernfalls wird ein Wert von Null hinzugefügt. Diese Berechnung ermöglicht die Ermittlung der tatsächlich anerkannten ECTS-Punkte. Ein weiterer wichtiger Aspekt der Abfrage ist die Berechnung der Quote der anerkannten ECTS-Punkte in Prozent. Dies wird durch die folgende Anweisung erreicht:

Algorithmus 25: Berechnung der Quote anerkannter und absolvierter Fächer

```
1. IIf(Sum(ERASMUS_Outgoings_Anerkennungsdaten.[ects_tu_modul])=
2. 0, 0, Sum(IIf(ERASMUS_Outgoings_Anerkennungsdaten.anerkannt=
3. True, ERASMUS_Outgoings_Anerkennungsdaten.[ects_tu_modul],0))/
4. Sum(ERASMUS_Outgoings_Anerkennungsdaten.[ects_tu_modul])100)
5. AS QuoteAnerkannt_in_Prozent
```

Hierbei wird zunächst überprüft, ob die Summe der anerkennungsfähigen ECTS-Punkte gleich Null ist. Falls dies der Fall ist, wird der Wert Null zurückgegeben, um eine Division durch Null zu vermeiden. Andernfalls wird die Summe der anerkannten ECTS-Punkte durch die Summe der anerkennungsfähigen ECTS-Punkte geteilt und mit 100 multipliziert, um den Prozentsatz zu berechnen. Zusammengefasst ermöglicht dieses SQL-Statement eine detaillierte Analyse der anerkennungsfähigen und tatsächlich anerkannten ECTS-Punkte sowie die Berechnung der Anerkennungsquote in Prozent. Die Verwendung von IIf-Anweisungen stellt sicher, dass die Berechnungen nur unter bestimmten Bedingungen durchgeführt werden, und gewährleistet die Genauigkeit und Relevanz der resultierenden Daten. Diese Methode bietet eine präzise Möglichkeit zur Überwachung und Bewertung der akademischen Leistung und Anerkennung der Studierenden im Rahmen des Austausches.

Der Nominierungsprozess ist der Kernprozess des Projekts in Bezug auf die Outgoings, da dieser die Zuteilung der Studierenden an die Partneruniversitäten umfasst. Die Hauptabfrage, welche von den Mitarbeitenden im Zusammenhang mit dem Nominierungsprozess verwendet wird, ist die SQL-Abfrage „Outgoings_Nominierung erfolgt + Nominierungsfrist“. Das SQL-Statement dient zur Abfrage relevanter Informationen zu den Bewerbungen von Outgoings im Rahmen des ERASMUS-Programms. Es verknüpft die Tabellen „ERASMUS_Outgoings_Stammdaten“, „ERASMUS_Outgoings_Bewerbungsdaten“ und „ERASMUS_Partner_Bewerbungsdaten“, um spezifische Details zu den Studierenden sowie zu den Partnerhochschulen abzurufen und eine Nominierungsfrist basierend auf dem Studienaufenthalt zu berechnen. In Abbildung 13 werden die drei verknüpften Tabellen mit den jeweils für diese Abfragen relevanten Daten aus der jeweiligen Tabelle dargestellt. Die Verbindung der Tabellen wird durch die Verbindungslinien in Abbildung 13 angedeutet und wird auf Basis des Namens der Partneruniversität und der Matrikelnummer der Outgoings umgesetzt.

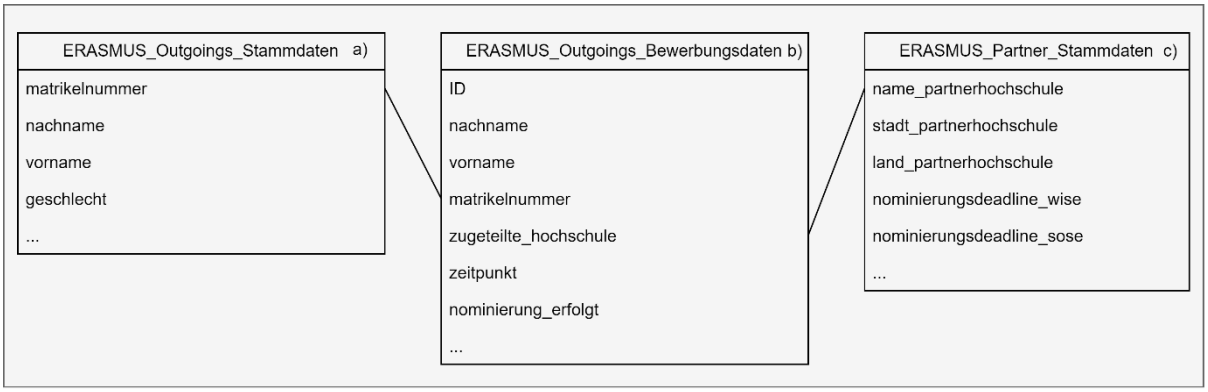


Abbildung 13: Visualisierung der Verknüpfung der Tabellen a) „ERASMUS_Outgoings_Stammdaten“, b) „ERASMUS_Outgoings_Bewerbungsdaten“ und c) „ERASMUS_Partner_Stammdaten“

Die Abfrage beginnt mit der Auswahl einer Reihe von Attributen aus den Tabellen „ERASMUS_Outgoings_Stammdaten“ und „ERASMUS_Outgoings_Bewerbungsdaten“. Diese Attribute umfassen den Nachnamen und Vornamen, die Matrikelnummer, den Zeitpunkt, den Status der Nominierung (Nominierung erfolgt) und die zugeteilte Hochschule. Ein wesentlicher Bestandteil der Abfrage ist die Berechnung der Nominierungsdeadline, die in einer neu erstellten Spalte mit der Bezeichnung „Nominierungsdeadline“ ausgegeben wird. Hierbei wird eine IIf-Anweisung verwendet, um die richtige Nominierungsfrist basierend auf dem Studienaufenthalt zu bestimmen. Die Anweisung für die Bestimmung der Nominierungsdeadline lautet:

Algorithmus 26: Auswahl der Nominierungsdeadline

```
1. IIf(ERASMUS_Outgoings_Bewerbungsdaten.zeitpunkt='WiSe',
2. ERASMUS_Partner_Bewerbungsdaten.[nominierunsdeadline_wise],
3. ERASMUS_Partner_Bewerbungsdaten.[nominierunsdeadline_sose])
4. AS Nominierungsdeadline
```

Diese Anweisung überprüft, ob der Wert im Feld „Zeitpunkt“ gleich „WiSe“ (Wintersemester) ist. Falls dies der Fall ist, wird die Nominierungsdeadline für das Wintersemester aus der Tabelle „ERASMUS_Partner_Bewerbungsdaten“ ausgewählt. Andernfalls wird die Nominierungsdeadline für das Sommersemester (Nominierunsdeadline SoSe) ausgewählt. Die Verbindung zwischen den beiden Tabellen wird durch einen „INNER JOIN“ hergestellt. Die Bedingung für die Verbindung lautet:

Algorithmus 27: Bedingung des „INNER-JOINS“ zur Auswahl der Nominierungsdeadline

```
1. ERASMUS_Outgoings_Bewerbungsdaten.[Zugeteilte Hochschule] =
2. ERASMUS_Partner_Bewerbungsdaten.[Name Partnerhochschule]
```

Diese Bedingung sorgt dafür, dass die Datensätze nur dann verbunden werden, wenn der Name der zugeteilten Hochschule in der Tabelle „ERASMUS_Outgoings_Bewerbungsdaten“ mit dem Namen der Partnerhochschule in der Tabelle „ERASMUS_Partner_Bewerbungsdaten“ übereinstimmt. Das Ergebnis dieser Abfrage ist eine umfassende Übersicht, die nicht nur grundlegende Informationen über die Outgoings und deren Nominierungsstatus enthält, sondern auch die entsprechende Nominierungsfrist auswählt, basierend auf dem Semester, in dem der Studienaufenthalt stattfindet. Diese Informationen sind entscheidend für die Verwaltung und Nachverfolgung der Bewerbungs- und Nominierungsprozesse im ERASMUS-Programm und stellen sicher, dass die Studierenden fristgerecht nominiert werden. Die Nutzung der IIf-Anweisung ermöglicht eine dynamische und kontextabhängige Berechnung der Nominierungsfristen, wodurch die Flexibilität und Genauigkeit der Datenverarbeitung erheblich verbessert werden.

Im Gegensatz zu den Daten der Incomings und Outgoings werden die Stammdaten der Partneruniversitäten selten um neue Dateneinträge ergänzt. Dennoch existieren einige Abfragen im Zusammenhang mit den Daten der Partneruniversitäten, welche die Grundlage wesentlicher Prozesse innerhalb des Projekts bilden. Dazu gehört die Abfrage „Partner_Zu_Freie Plätze“, welche auf Basis von Verträgen mit den Partneruniversitäten die derzeitigen freien Plätze an diesen berechnet. Dabei bleiben die Stamm- und Bewerbungsdaten der Partneruniversitäten in der Regel konstant, während ausschließlich die Daten der zugewiesenen Studierenden variabel sind und die endgültige Anzahl der verfügbaren Plätze bestimmen. Das SQL-Statement zur Berechnung der freien Plätze an den Partneruniversitäten im ERASMUS-Programm filtert die Datenbestände zuvor auf ein bestimmtes akademisches Jahr. Es verwendet

die Tabellen „ERASMUS_Partner_Bewerbungsdaten“ und „ERASMUS_Outgoings_Bewerbungsdaten“ und verbindet diese durch einen INNER JOIN, basierend auf dem Namen der Partnerhochschule. Die Abfrage beginnt mit der Definition des Parameters „Gewünschtes Akademisches Jahr“, der vom Endanwender festgelegt werden muss. Dieser Parameter wird zur Filterung der Daten verwendet, um die freien Plätze für das spezifische akademische Jahr zu berechnen. Im SELECT-Teil der Abfrage werden Informationen über den Namen der Partnerhochschule, Anmerkungen zur Anzahl freier Plätze, die maximale Anzahl an Plätzen für die Studierenden und die berechnete Anzahl der freien Plätze gesammelt. Letztere wird durch Subtraktion der Anzahl der zugewiesenen Studierenden von der maximalen Anzahl der Plätze bestimmt. Die Berechnung der freien Plätze erfolgt durch folgende Anweisung:

Algorithmus 28: Berechnung der freien Plätze an einer Partneruniversität

```
1. (ERASMUS_Partner_Bewerbungsdaten.[student_mobility] –
2. SUM(IIF(ERASMUS_Outgoings_Bewerbungsdaten.
3. [zugeteilte_hochschule] =
4. ERASMUS_Partner_Bewerbungsdaten.[name_partnerhochschule], 1,
5. 0))) AS [Freie Plätze]
```

Diese Anweisung nutzt die Funktion „SUM(IIF())“ um zu zählen, wie viele Studierende der jeweiligen Partnerhochschule zugewiesen werden, wobei jedes Mal, wenn die nachfolgende Bedingung erfüllt ist der Wert 1 addiert wird:

Algorithmus 29: SQL-Bedingung der Summenfunktion

```
1. ERASMUS_Outgoings_Bewerbungsdaten.[zugeteilte_hochschule] =
2. ERASMUS_Partner_Stammdaten.[name_partnerhochschule]
```

Die Gesamtsumme dieser Werte wird dann von der maximalen Anzahl an verfügbaren Plätzen an der Partneruniversität abgezogen. Der „INNER JOIN“ verbindet die Tabellen „ERASMUS_Partner_Bewerbungsdaten“ und „ERASMUS_Outgoings_Bewerbungsdaten“ durch die Bedingung, dass der Name der Partnerhochschule in beiden Tabellen übereinstimmen muss. Dies gewährleistet, dass nur Daten kombiniert werden, die in Verbindung zueinanderstehen. Hierbei gilt es zu erwähnen, dass der Name der Partneruniversität eindeutig ist, sodass ein kombinierter Schlüssel in diesem Fall nicht benötigt wird. Zusätzlich filtert die WHERE-Klausel die Daten nach dem akademischen Jahr, das dem angegebenen Parameter „Gewünschtes Akademisches Jahr“ entspricht, wodurch die Abfrage auf das gewünschte Jahr beschränkt wird. Durch eine „GROUP BY“-Klausel werden die Ergebnisse nach dem Namen der Partnerhochschule, den Anmerkungen zur Student Mobility, der maximalen Anzahl an Plätzen und dem akademischen Jahr gruppiert. Dies ist notwendig, um die aggregierten Berechnungen korrekt durchzuführen und sicherzustellen, dass die freien Plätze für jede Partnerhochschule individuell berechnet werden. Zusammengefasst ermöglicht diese Abfrage eine genaue Bestimmung der aktuell freien Plätze an den Partneruniversitäten für ein spezifisches akademisches Jahr, basierend auf den bestehenden Verträgen und den bereits zugewiesenen Studierenden. Diese Informationen sind essenziell für die Planung und Verwaltung der Studienplatzvergabe an Partneruniversitäten im Rahmen des Projekts.

3.3 Darstellung der vorliegenden Prozesse und Erkennung von Automatisierungspotenzialen

Das im Umfang dieser Ausarbeitung betrachtete Projekt umfasst zahlreiche unterschiedliche Prozesse, welche bisher ineffizient ablaufen und Teilschritte überwiegend manuell durchgeführt werden. Diese Ineffizienz äußert sich in einem erheblichen Arbeitsaufwand für die Organisation, Bereinigung sowie manuelle Im- und Exporte der Daten in das bestehende Datenbanksystem. Nach der bereits aufgezeigten funktionalen Umstrukturierung der Datenbank und der optimierten Abfrage von Daten innerhalb dieser, werden in diesem Abschnitt zunächst Prozessdiagramme aufgestellt und hinreichend analysiert. Hierbei gilt es zu erwähnen, dass die vorliegenden Prozesse für eine bessere Darstellbarkeit vereinfacht skizziert und Teilprozesse bewusst ignoriert werden, da die Prozesse nicht immer identisch ablaufen und Teilaspekte häufig angepasst werden. Die folgenden Abbildungen umfassen deshalb die wesentlichsten und grundsätzlichsten Teilschritte innerhalb der Prozessabläufe. Die Diagramme dienen zur Erkennung von eben diesen Ineffizienzen, sodass ineffiziente Teilschritte zukünftig automatisiert ablaufen und die Mitarbeiter entlastet werden. Prozessschritte, welche ein Automatisierungspotenzial aufweisen werden innerhalb der Diagramme grün markiert. Zusätzlich wird hierdurch die Protokollierung, das Error-Handling und die Fehlervermeidung erheblich verbessert. Die manuelle Dateneingabe in relationale Datenbanken birgt ein inhärentes Risiko von Fehlern und Ungenauigkeiten. Kisor Ray et al. (2015) weisen auf das Potenzial für manuelle Eingabefehler in klinischen Datenbeständen hin, die zu falschen Diagnosen führen können (Ray et al. 2015). F. Evert et al. (1999) entwickelten eine relationale Datenbankimplementierung für agrarökologische Forschungsdaten und betonten die Bedeutung effizienter Datenladeverfahren (van Evert et al. 1999). Zur Erleichterung der Datenintegration stellten Nyulas et al. (2007) DataMaster vor, ein Plug-in für den Import von Schemata und Daten aus relationalen Datenbanken in Tools zur Ontologieentwicklung (Csongor Nyulas et al. 2007). Diese Studien unterstreichen insgesamt die Herausforderungen des manuellen Datenimports und schlagen verschiedene Lösungen vor, darunter ein verbessertes Datenbankdesign, spezialisierte Eingabesprachen und Tools für die Datenintegration, um die mit einer falschen Datenimplementierung oder -löschung verbundenen Risiken zu mindern.

Der zentrale Prozess im Rahmen des Projekts ist der Nominierungsprozess, welcher die Bewerbung von ein- und ausgehenden Studierenden für einen Studienplatz im In- oder Ausland umfasst. Studierende können sich über ein in Typo3 erstelltes Formular bewerben, das auf einer Website eingebettet ist. Typo3 ist ein Open Source Content Management-System, welches zum Aufbau einfacher Webseiten bis hin zur vollumfänglichen Workflowerstellung genutzt werden kann (Meyer und Helmich 2011). Der aktuelle Zustand dieser Formulare stellt ein erhebliches Risiko dar, da Studierende falsche Informationen übermitteln können. Dies liegt daran, dass die Formulare derzeit keine strengen Validierungsmechanismen beinhalten, um die Richtigkeit und Konsistenz der eingegebenen Daten zu gewährleisten. Wenn fehlerhafte Daten oder falsche Datentypen übermittelt werden, kann dies zu erheblichen Problemen beim automatisierten Import in eine Datenbank führen. Ein zentrales Risiko besteht darin, dass die übermittelten Datentypen nicht den erwarteten Spezifikationen entsprechen. Beispielsweise können numerische Werte als Text oder in einem ungültigen Format eingegeben werden, was dazu führt, dass die Datenbank die Informationen nicht korrekt importieren kann. Dies zwingt die Administratoren, die Daten manuell zu überprüfen und anzupassen, was nicht nur zeitaufwändig und ressourcenintensiv ist, sondern auch das Risiko menschlicher Fehler erhöht. Darüber hinaus führt die mangelnde Validierung der Eingaben zu Inkonsistenzen in den Daten. Wenn zum Beispiel Zahlen oder Texte in verschiedenen Formaten eingegeben werden können, ohne dass eine Überprüfung stattfindet, können nachgelagerte Prozesse, die auf diesen Daten basieren, wie beispielsweise die Berechnung freier Studienplätze an Partnerhochschulen, fehlschlagen. Diese fehlerhaften Daten können zu falschen Ergebnissen führen, die im schlimmsten Fall Entscheidungen beeinflussen, die auf diesen ungenauen Daten basieren. Insgesamt gefährdet die derzeitige Form der Datenübermittlung nicht nur die Integrität und Qualität der gespeicherten Daten, sondern erhöht auch den Aufwand und die Komplexität bei

der Datenverarbeitung erheblich. Es ist daher unerlässlich, dass die Formulare so angepasst werden, dass sie eine strenge Validierung der eingegebenen Daten sicherstellen, um diese Risiken zu minimieren und einen reibungslosen, fehlerfreien Datenimport zu ermöglichen. Deshalb besteht der erste Schritt zur Automatisierung der Prozesse in der Anpassung der bestehenden Formulare und der Nutzung der in Typo3 vorhandenen Validierungsfunktionen. Diese Validierungsfunktionen dienen dazu, die Eingaben in einem Formular vor der Datenübermittlung zu überprüfen und gegebenenfalls Fehlermeldungen an die Benutzer zurückzugeben. Dadurch wird sichergestellt, dass nur gültige Datentypen übermittelt werden, wodurch eine Automatisierung der Datenimporte erst ermöglicht wird.

Abbildung 14 veranschaulicht den Ablauf des Bewerbungs- und Nominierungsprozesses im Rahmen des Projekts und gliedert sich in einen "Outgoing"-Bereich für die Studierenden sowie den Bereich "Internationales Team" für das zuständige Hochschulteam. Der Prozess beginnt mit dem ersten Schritt, in dem sich die Studierenden über ein Formular bewerben, wie in Abbildung 14 als "Bewerben über Formular" dargestellt. Diese erste Aktion erfolgt im Bereich "Outgoing" und bildet den Ausgangspunkt des gesamten Prozesses. Nach der Einreichung des Formulars übernimmt das Team „Internationales“ die Verantwortung für die Datenverarbeitung. In der Abbildung ist der nächste Schritt als "Daten in Datenbank pflegen" gekennzeichnet. Hierbei werden die vom Studierenden übermittelten Daten in die Backend-Datenbank eingepflegt. Dieser Schritt ist von zentraler Bedeutung, da die Qualität und Vollständigkeit der Daten den weiteren Verlauf des Prozesses maßgeblich beeinflussen. Im darauffolgenden Schritt, der in der Abbildung als "Daten sichten" dargestellt ist, überprüft das Team „Internationales“ derzeit die gepflegten Daten auf Korrektheit und Vollständigkeit. Diese Überprüfung ist essenziell, um sicherzustellen, dass keine fehlerhaften oder unvollständigen Daten in den weiteren Prozess gelangen, was potenziell schwerwiegende Auswirkungen haben könnte. Zukünftig kann die Validierung deutlich schneller und mit geringerem personellem Aufwand erfolgen. Neben einer Validierung werden die Daten der Studierenden gemäß den Anforderungen der Partneruniversitäten gesichtet und einzelne Studierende als mögliche Nominierungskandidaten definiert. Sobald die Daten geprüft sind, sendet das Team „Internationales“ an mögliche Kandidaten eine Nominierungsbestätigung, wie in Abbildung 14 gezeigt. Diese Bestätigung signalisiert, dass der Bewerber von der Hochschule für das Austauschprogramm an der Partneruniversität nominiert wird. An dieser Stelle erfolgt eine Entscheidung, die in der Abbildung 14 durch ein Raute-Element mit der Beschriftung "Nominiert" dargestellt wird. Sollte die Entscheidung positiv ausfallen (Ja), wird der Studierende für die nächste Phase freigegeben. Anderenfalls endet der Nominierungsprozess ohne eine Nominierung an einer Partneruniversität. Im Bereich "Outgoing" sendet der Studierende daraufhin seine Bewerbung an die Partnerhochschule, wie durch den Schritt "Bewerbung senden" illustriert. Dies ist eine notwendige Fortsetzung des Prozesses nach der internen Nominierung. Abschließend erhält der Studierende eine Rückmeldung von der Partnerhochschule, die in der Abbildung als "Zusage der Hochschule" bezeichnet ist. Diese Zusage stellt den erfolgreichen Abschluss des Bewerbungsverfahrens dar. Der Nominierungsprozess wird in Abbildung 14 in vereinfachter Form skizziert, wobei Teilschritte und Prozesse, die nur in Ausnahmefällen relevant sind, außer Acht gelassen werden. Solche Ausnahmefälle treten häufig an universitären Einrichtungen auf, da die individuellen Situationen der Studierenden unterschiedlich sein können. Wesentlich ist, dass der hier dargestellte Prozess im Grundaufbau für jeden Outgoing identisch ist.

Nachdem ein Outgoing sich über das Formular beworben hat, können die Daten aus einem Zwischenspeicher im Excel-Format heruntergeladen werden. Daraufhin müssen die Daten derzeit manuell in die Backend-Datenbank, folglich in die Basistabellen eingepflegt werden. In diesem Schritt liegt das größte Automatisierungspotenzial gemessen an der Arbeitszeit für die manuelle Einpflegung der Daten und der Fehleranfälligkeit dieses Prozessschrittes. Die Zeitersparnis bei der Einpflegung der Daten nimmt mit steigender Bewerbungszahl überproportional zu, da vorliegende Excel-Dokumente unübersichtlicher werden und fehlerhafte Eintragungen häufiger auftreten. Auf das Einpflegen der Daten folgt die Sichtung der Daten, um zu entscheiden, ob ein Outgoing an einer Partneruniversität nominiert wird. Auch eine Datensichtung wäre automatisiert möglich. In diesem Fall müssten Referenzbereiche und Parameter definiert werden, durch deren Auswertung eine Entscheidung getroffen werden kann, welche

Studierenden prinzipiell abgelehnt oder welche angenommen werden. Da bisher keine entsprechenden Definitionen vorliegen, wird vorerst auf eine Automatisierung der Datensichtung verzichtet. Sobald eine Entscheidung darüber getroffen wird, welche Outgoings an den Partneruniversitäten nominiert werden, müssen die Studierenden mittels einer Nominierungsbestätigung über diese Entscheidung informiert werden. Die Nominierungsbestätigungen basieren auf Vorlagen und sind in ihrer Grundform für jeden Outgoing identisch, sodass auch dieser Schritt automatisiert werden kann. Dabei wird die Automatisierung auf die Erstellung und Befüllung der Dokumente beschränkt, während der Versand per E-Mail aus Revisionsgründen manuell erfolgen soll. Die letzten beiden Teilschritte im Nominierungsprozess, wie in Abbildung 14 dargestellt, liegen in der Verantwortung der Outgoings und fallen somit außerhalb des hier behandelten Rahmens. Nach ihrer Nominierung müssen sich die Outgoings eigenständig an einer Partneruniversität bewerben und auf eine Zusage warten.

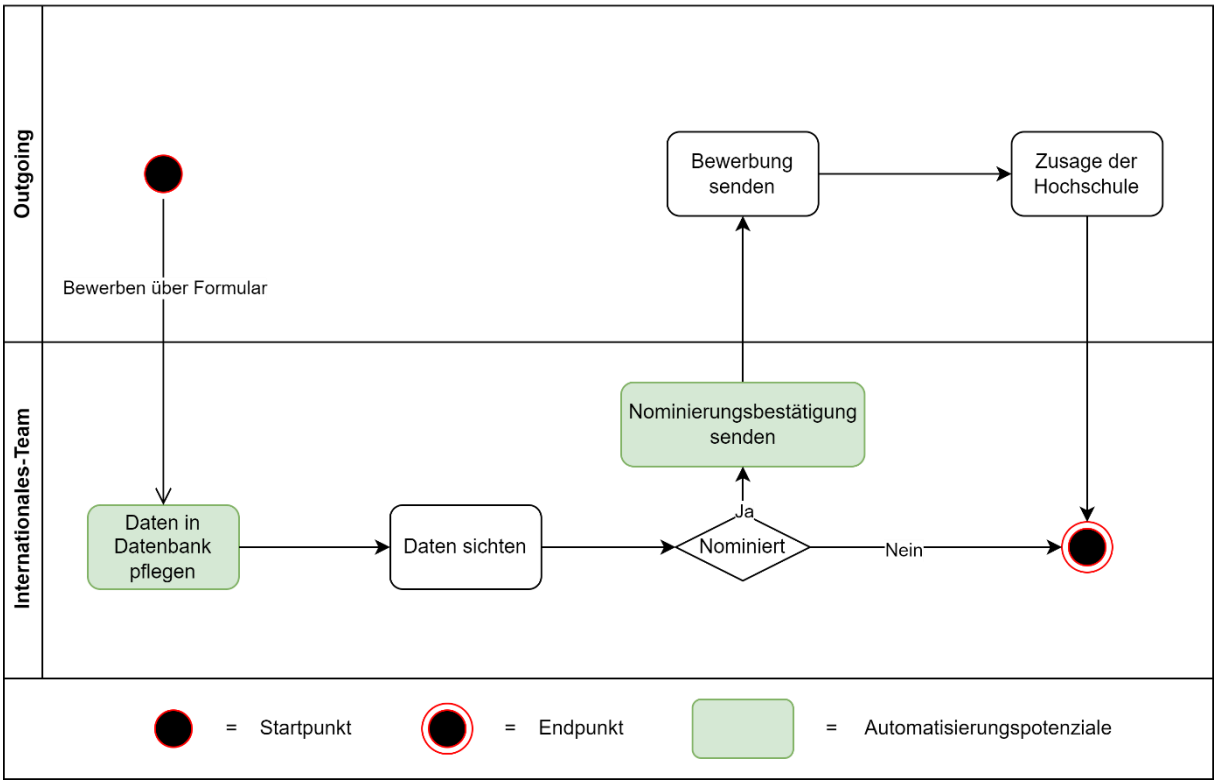


Abbildung 14: Prozessdiagramm des Nominierungsprozesses

In Abbildung 15 wird der Anerkennungsprozess einer absolvierten Prüfungsleistung eines Outgoing an einer Partneruniversität skizziert. Analog zu dem Nominierungsprozess werden Teilprozesse, welche eine Ausnahme bilden und nicht auf jeden Outgoing abbildbar sind außer Betracht gelassen. Den Beginn des Anerkennungsprozesses bildet das Ausfüllen eines Anerkennungsantrages seitens des Outgoings. Diese Daten, werden als PDF-Dokument per E-Mail an das Team „Internationales“ übermittelt. Die Daten werden dann erneut manuell in die Datenbank eingepflegt und an den zuständigen Lehrstuhl weitergeleitet. Der Lehrstuhl prüft nun, ob der Kurs an der Partnerhochschule den Anerkennungsanforderungen entspricht und somit anerkennungsfähig ist. Nach Sichtung der Daten und einer Beratungszeit wird die Entscheidung dem Team „Internationales“ mitgeteilt und die Zustimmung oder Ablehnung in die Datenbank gepflegt. In Abbildung 15 ist ebenfalls dargestellt, dass im Falle einer Ablehnung zusätzlich ein Grund für die Ablehnung mit in der Datenbank hinterlegt wird. Der zuvor von dem Outgoing ausgefüllte Anerkennungsantrag wird nun um die Zu- oder Absageinformationen ergänzt und an den Outgoing zurückgesendet. Wie bereits im Nominierungsprozess sind in Abbildung 15 nun zwei Prozessschritte skizziert, welche außerhalb der Beteiligung des

Teams „Internationales“ liegen, sodass auf diese Schritte keinen Einfluss genommen werden kann. Der Outgoing muss daraufhin seine Prüfung an der Partneruniversität ablegen und seinen Anerkennungsantrag mit einer Unterschrift finalisieren und an das Team „Internationales“ in einem weiteren Schritt zurücksenden. Die finale Anerkennung wird nun in der entsprechenden Tabelle gepflegt und der Prozessablauf endet. Innerhalb des Anerkennungsprozesses können alle Teilprozesse, welche in der Verantwortlichkeit des Team „Internationales“ liegen automatisiert werden. Die Prozesse des Outgoings und des zuständigen Lehrstuhles werden nicht in die Automatisierung integriert.

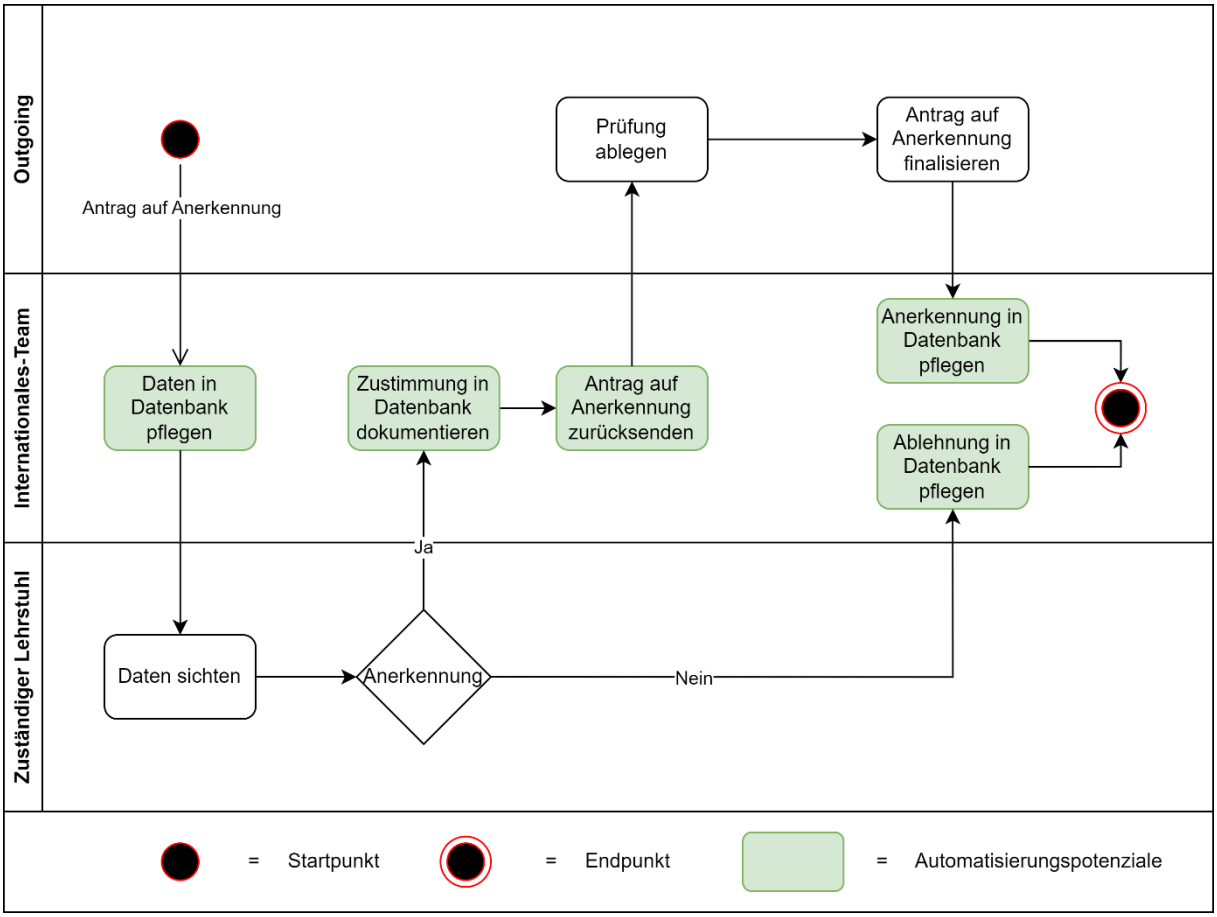


Abbildung 15: Prozessdiagramm des Anerkennungsprozesses

Neben den Datenbewegungen innerhalb dieser beiden Prozesse gibt es weitere Dokumente, die in unregelmäßigen Abständen erstellt werden. Ein Beispiel hierfür ist das ToR, welches eine Leistungsübersicht eines Incomings darstellt. Dieses Dokument, welches als PDF vorliegt, wird nach der Beendigung eines Auslandssemesters dem Incoming zugesendet. Das ToR basiert erneut auf einer Vorlage und wird derzeit manuell erstellt. Die eingepflegten Daten liegen innerhalb der Datenbank vollständig vor, sodass auch die Erstellung und Befüllung des ToR automatisiert ablaufen kann.

4 Entwicklung der automatisierten Import- und Exportfunktionalitäten in Python

In diesem Kapitel wird die Entwicklung der automatisierten Import- und Exportfunktionalitäten mithilfe von Python detailliert beschrieben. Zunächst werden die Erstellung einer benutzerfreundlichen Oberfläche und die Implementierung des automatisierten Datenimports erläutert. Darauf aufbauend werden die Entwicklung von automatisierten Exportprozessen und die Erstellung relevanter Dokumente beschrieben, um den Datenaustausch und die Berichterstattung zu optimieren. Ziel dieses Kapitels ist es, die Effizienz und Genauigkeit der Datenverarbeitungsprozesse durch Automatisierung zu erhöhen und gleichzeitig die Benutzerfreundlichkeit zu verbessern.

4.1 Entwicklung einer Benutzeroberfläche und des automatisierten Datenimports

Eine effektive Gestaltung der Benutzeroberfläche (UI) ist entscheidend für den Erfolg von Anwendungen. Zu den Schlüsselkomponenten eines guten UI-Designs gehören eine benutzerfreundliche Informationsdarstellung, effiziente Dateneingabemethoden und die Berücksichtigung von Benutzereigenschaften und -kontext (Boonlit und Dongsong 2005). Iterative Design- und Testprozesse werden empfohlen, um Schnittstellen zu schaffen, die den Benutzer effektiv durch Interaktionen führen und Kommunikationsfehler bewältigen (Candace Kamm 1994). Insgesamt können gut gestaltete UI technologische Einschränkungen überwinden und die Benutzerzufriedenheit bei verschiedenen Anwendungstypen erhöhen. Wie bereits in Abschnitt 2.4 erwähnt, bietet Tkinter als Erweiterung für Python die Möglichkeit zur Erstellung von UIs. Innerhalb der Python-Applikation werden verschiedene Funktionen benötigt, sowohl für den Datenimport als auch für die automatisierte Dokumentenerstellung. In Abbildung 16 ist der erste Entwurf der Benutzeroberfläche für das ERASMUS-Datenverarbeitungstool dargestellt. In der Abbildung sind die verschiedenen Knöpfe für die einzelnen Funktionen abgebildet, wobei der Knopf für die Import-Funktion, eine Überkategorie für alle Datenimportfunktionen, eingeblendet ist.

Die größten Zeitersparnisse in den aktuellen Prozessen des Projekts liegen innerhalb der Automatisierung der Datenverkehre, somit dem automatisierten Datenimport und Datenexport. Die Daten der Studierenden, welche für die Einpflegung in die Datenbank benötigt werden, werden innerhalb der bereits angepassten Formulare gesammelt. Die angesammelten Daten können im Anschluss als Excel-Datei heruntergeladen werden. Derzeit wird mit einer E-Mail als Ausgabeformat gearbeitet aus der die Daten manuell herausgeschrieben und in die Datenbank eingepflegt werden. Die angepassten Formulare stellen sicher, dass die Excel-Dateien alle benötigten Informationen enthalten und die korrekten Datentypen verwendet werden. Dennoch ist die Formatierung der Dateien nicht optimal. Einige Zeilen enthalten zusätzliche Informationen, die für die aktuellen Prozesse irrelevant sind. Diese Zeilen müssen in einem ersten Vorverarbeitungsschritt entfernt werden, damit die relevanten Spalten im Programm korrekt eingelesen werden können. Microsoft Access bietet leistungsstarke Datenimportfunktionen an, die es den Benutzern ermöglichen, externe Datenquellen aus verschiedenen Datenbanktypen über ODBC-Treiber zu verknüpfen oder zu importieren (Collins 2021). Das Risiko bei der Nutzung der vordefinierten Funktionen ist durch die spezielle Situation des Projekts gegeben und der bereits behandelten Zuordnung der Identifikationsschlüssel. In Microsoft Access wird bei der Eingabe von Datensätzen lediglich überprüft, ob die Datentypen die Datenaufnahme zulassen und alle erforderlichen Felder ausgefüllt sind. Hierbei ist essenziell, dass alle Spalten identisch sind und auch in der richtigen Reihenfolge vorliegen. Dies gilt es in der eigenen Applikation zu vereinfachen indem aus einer vorliegenden Excel nur Informationen extrahiert werden, die auch explizit benötigt werden. Zudem sollen die Daten nicht in derselben Reihenfolge

vorliegen müssen, obwohl dies für eine bessere Übersichtlichkeit zu empfehlen ist. Zusätzlich prüft die Funktion innerhalb von Microsoft Access ausschließlich auf eine Doppelung der Primärschlüssel bei der Einpflegung der Daten in die Datenbank. In dem hier behandelten Projekt ist der definierte Primärschlüssel die automatisch erstellte ID innerhalb der Tabellen, demnach eine Information, welche in den Excel-Dateien aus den Formulardaten nicht vorliegt. Die zusätzlich definierten kombinierten Schlüssel, welche vor allem für die Zuordnung der Dateninhalte innerhalb der Abfragen benötigt werden, sind von dieser Prüfung ausgeschlossen. Das liegt daran, dass diese identifizierenden Merkmale nicht direkt als Schlüssel definiert sind und deshalb auch nicht weiter von Microsoft Access als solche betrachtet werden. Deshalb ist der größte Vorteil einer eigenen Importfunktion, dass überprüft werden kann, ob es identische Einträge, folglich der Übereinstimmung jedes Eintrages bis auf die automatisch erstelle ID, gibt. Dadurch ist gewährleistet, dass Daten nur dann abgespeichert werden, wenn sie innerhalb der Tabellen neu sind und somit eine doppelte Datenhaltung vermieden werden kann.

Ein weiterer Vorteil einer individuell programmierten Datenschnittstellenapplikation ist die Entfernung der Notwendigkeit die Backend-Datenbank für das Befüllen und Abfragen der Tabellen zu öffnen. In Abschnitt 5.1 wird genauer auf die Zugriffskonzepte eingegangen, dennoch sei hier bereits zu erwähnen, dass der direkte Zugriff auf die Backend-Datenbank das Risiko der Manipulation der Stammdatentabellen erhöht. Da die Importfunktion von Microsoft Access nur innerhalb der Datenbank selbst ausgeführt werden kann muss der Endanwender diese aktiv öffnen, um Daten halbautomatisiert einzupflegen. Dadurch entsteht das Risiko, dass die Tabellen angepasst oder gelöscht werden können. Durch die Python-Applikation wird eine Schnittstelle zu der Backend-Datenbank aufgebaut und der Anwender kann dieselben und zusätzliche Befehle über eine Benutzeroberfläche ausführen, ohne die Datenbank selbst geöffnet zu haben. Dadurch lassen sich die Zugriffe besser regulieren und verfolgen.

Abbildung 16: Root-Ebene des ERASMUS-Datenverarbeiters

Mit einem Klick auf den Knopf zum Einpflegen von Daten gelangt der Anwender in die Übersicht des Datenimports, welche in Abbildung 17 dargestellt ist. Hier wird der Datenimport in

drei Einzelschritte untergliedert. Wesentlich ist, dass der Benutzer eine Übersicht darüber bekommt in welcher Reihenfolge der Datenimport durchgeführt werden soll. Deshalb sind wie in Abbildung 17 aufgezeigt zunächst zwei der drei Knöpfe ausgeschaltet, sodass der Benutzer nicht in der Lage ist einen Schritt des Datenimports zu überspringen. Zusätzlich erhält der Anwender durch einen Knopf mit der Beschriftung „Zurück“ die Möglichkeit in die Funktionsübersicht zurückzukehren. Des Weiteren ist unterhalb des Zurück-Knopfes ein Infotext eingeblendet, der sich entsprechend des derzeitigen Schrittes anpasst. Dieser dient erneut dazu, dem Benutzer zu signalisieren in welcher Phase des Datenimports er sich befindet. Zusätzlich wird der Infotext im Falle einer Fehlermeldung dazu verwendet diese dem Nutzer anzuzeigen, sodass eine Fehlerbehandlung erleichtert wird. Der erste Schritt der Dateneinpflegung besteht in der Auswahl einer Excel-Datei, in der sich die zu importierenden Daten befinden.

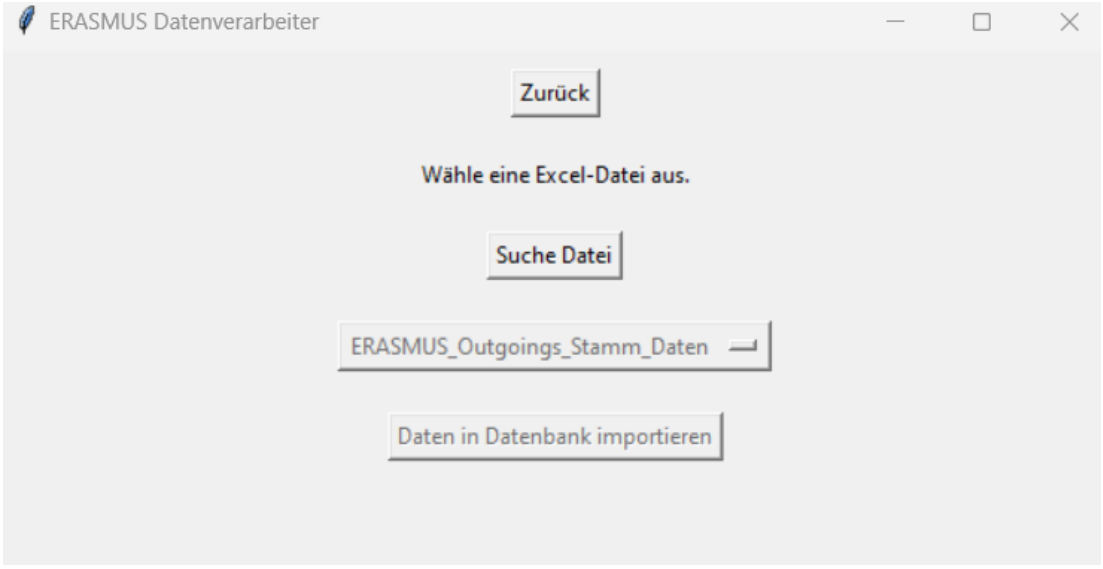


Abbildung 17: Current-Ebene des Datenimports

Nachdem der Benutzer die entsprechende Excel-Datei ausgewählt hat, springt die Anwendung in die Ansicht, welche in Abbildung 18 dargestellt ist. In dieser Abbildung sind nun die beiden inaktiven Knöpfe aktiviert. Der Infotext weist in der neuen Ansicht den Benutzer nun darauf hin, dass die entsprechende Tabelle aus dem Dropdown-Menü ausgewählt werden muss in die die Daten importiert werden sollen. Der letzte Schritt besteht in dem Klicken des Knopfes unterhalb der Tabellenauswahl, damit die Daten in die Datenbank importiert werden. Hierbei wird innerhalb des Infotextes eine Nachricht ergänzt, die entweder über den erfolgreichen Import in die Datenbank berichtet oder die entsprechende Fehlermeldung an den Benutzer zurückgibt. Eine mögliche Fehlermeldung ist das Fehlen von Spalten in der ausgewählten Excel-Tabelle, wenn diese mit der Tabelle aus der Datenbank verglichen wird. In diesem Fall werden neben der Anzahl der fehlenden Spalten auch die expliziten Namen der Spalten aufgelistet, welche in der Excel-Datei fehlen. Dadurch ist der Benutzer in der Lage diese auf fehlende Spalten zu überprüfen oder die Rechtschreibung der Spaltennamen anzupassen. Durch diesen Infotext ist der Anwender auch dazu aufgefordert die Einheitlichkeit der Bezeichnungen zu bewahren. Zudem werden Unstimmigkeiten zwischen der Tabelle und der Ausgabe der Formulare direkt ersichtlich. Beispielsweise können Anpassungen in den Formularen durch eine Erweiterung in der entsprechenden Tabelle vergessen und durch den Infotext kontrolliert werden. Der vorgestellte Prozess sieht für den Import in jede Basistabelle der Backend-Datenbank identisch aus. Der Endanwender muss jedoch eine andere Excel-Datei und Zieltabelle auswählen.

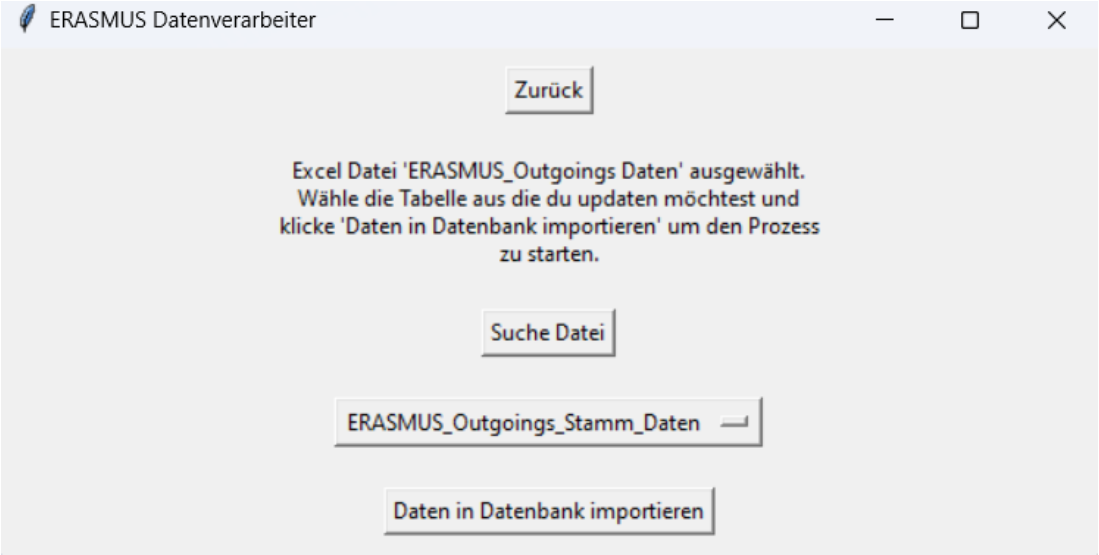


Abbildung 18: Current-Ebene des Datenimports nach Auswahl einer Excel-Datei

Durch den Ansatz zuerst eine Benutzeroberfläche aufzusetzen, deren Hintergrundfunktionen nicht definiert sind ist nun ein klarer Rahmen gesetzt, welche Funktionen wann und in welcher Art und Weise benötigt werden. Innerhalb der Python-Applikation müssen Funktionen für die Suche nach Dateien auf dem jeweiligen Rechner, die Abfrage der Tabellen aus der Backend-Datenbank und die Einpflegung der Daten in diese erstellt werden. Bevor auf die Implementierung der einzelnen Funktionen eingegangen wird, wird zunächst die Navigation der Benutzeroberfläche behandelt. Die Knöpfe und Übersichten aus den vorherigen Abbildungen bestehen aus leeren Hüllen, somit Knöpfe, die keine Funktion erfüllen. Deshalb müssen zunächst Funktionen hinter den jeweiligen Knöpfen hinterlegt und die einzelnen Ansichten sinnvoll miteinander verknüpft werden. Dazu lassen sich zwei Ebenen definieren, wobei die oberste Ebene die root-Ebene ist. Die root-Ebene ist die Startübersicht, auf der sich alle Knöpfe befinden, welche eine Navigation in die einzelnen Funktionen ermöglichen. Die zweite Ebene wird als current-Ebene bezeichnet und umfasst die jeweilige Funktionsebene, die über die root-Ebene geschaltet wird. In Ausnahmefällen gibt es mehrere current-Ebenen, wenn innerhalb einer Funktion zwei oder mehr Navigationen erfolgen. Der Grundaufbau jedes Knopfes in der root-Ebene sieht identisch aus und unterscheidet sich nur durch dessen Ausprägungen. Folgende Zeilen erstellen den Knopf für die Datenimport-Funktion:

Algorithmus 30: Programmcode zur Erstellung eines Buttons für die Import-Applikation

```
1. btn_open_import_app = tk.Button(root, text="Import App öffnen",
2. command=open_import_app)
3. btn_open_import_app.pack(pady=10)
4. root_widgets.append(btn_open_import_app)
```

Die relevantesten Aspekte dieses Ausschnittes ist der hinterlegte „Command“, der eine Navigation in die nächste Ebene ermöglicht, welche noch definiert werden muss. In der letzten Zeile wird der Knopf zu den „root_widgets“ hinzugefügt. Hierdurch lassen sich die Knöpfe aus der root-Ebene in einem Speicher erfassen. Dieser Zwischenschritt ist wesentlich, wenn aus der current-Ebene zurück in die root-Ebene navigiert wird. Der folgende Ausschnitt zeigt die Logik hinter dem Wechsel zwischen der root- und der current-Ebene:

Algorithmus 31: Programmcode für die back_to_root-Funktion

```
1. def back_to_root():
2.     global current_widgets
3. for widget in current_widgets:
4.     widget.pack_forget()
5. for widget in root_widgets:
6.     widget.pack(pady=10)
```

Die Funktion „back_to_root“ wird jeweils hinter dem Zurück-Knopf in den current-Ebenen hinterlegt. Bei dem Aufruf der Funktion werden alle Knöpfe und Texte in der current-Ebene gelöscht, somit alle Anzeigeflächen, welche in der root-Ebene nicht mehr benötigt werden. In einem zweiten Schritt werden alle Knöpfe und Texte, welche zu Beginn der root-Ebene zugeordnet wurden, wieder hinzugefügt, sodass der Ausgangszustand der root-Ebene wiederhergestellt ist. Mit Betätigen des „Daten Einpflegen“-Knopfes gelangt der Anwender daraufhin in ein neues Fenster, welches für den Datenimport genutzt wird. Die Basisfunktion unterteilt sich in weitere Funktionen, welche für die einzelnen Schritte innerhalb des Datenimports zuständig sind. In der Basisfunktion werden zunächst alle Knöpfe definiert und weitere Funktionen für das Dropdown-Menü oder die Einstellung des Datenimport-Knopfes, der zunächst deaktiviert ist. Der erste Schritt des Importprozesses besteht in der Auswahl einer Excel-Datei. Beim Betätigen des „Suche Datei“-Knopfes gelangt der Anwender in eine Ordneransicht, in der eine Excel auf dem Rechner ausgewählt werden kann. Bei erfolgreicher Auswahl einer Excel-Datei wird der Datenimport-Knopf aktiviert. Nun kann der Anwender eine neue Tabelle auswählen oder die voreingestellte Basistabelle für den Import verwenden. Mit Betätigen des Import-Knopfes wird zunächst eine die Funktion „process_excel“ gestartet, welche als Parameter den Pfad der Excel-Datei und den Namen der Tabelle, in die die Daten importiert werden sollen, erhält. Innerhalb der process_excel-Funktion werden die Dateneinträge ausgelesen zusammen mit den Spaltennamen der Excel-Datei. Die zwischengespeicherten Daten werden an die insert_into_access_database-Funktion weitergegeben. Diese Funktion beinhaltet die Kernfunktionen für die Einpflegung der Daten und ist generisch aufgebaut, sodass eine Funktion für alle Daten verwendet werden kann. Als Eingabeparameter benötigt die Funktion die Datentabelle, die Daten und die Spalten aus der Excel-Datei. Mit dem Namen der Datentabelle wird zunächst die Tabelle aus der Backend-Datenbank aufgerufen und die Spaltennamen extrahiert. Die Spaltennamen aus der Backend-Datenbank werden nun mit den Spalten der Excel-Datei verglichen und das Ergebnis wird in zwei Speichervariablen abgelegt. Wenn eine Spalte in beiden Dateien vorliegt, wird sie in einer Variable mit der Bezeichnung „matching_columns“ gespeichert. Insofern nach dem Vergleich Spalten in der Backend-Tabelle fehlen werden alle fehlenden Spalten in einer zweiten Variable mit der Bezeichnung „missing_columns“ gespeichert. Damit Daten importiert werden können müssen alle Spalten der Datenbank vorhanden, somit die Variable „missing_columns“ leer sein. Wenn keine Spalten fehlen, wird der Bau der Datenimportskripte gestartet. Sobald Spalten fehlen, werden die fehlenden Spaltennamen zurückgegeben und in der Infobox mit dem Hinweis die Excel-Datei auf fehlende Spalten zu überprüfen an den Anwender zurückgegeben.
Innerhalb der Anwendung wird nun geprüft, ob es sich bei der ausgewählten Tabelle um eine Tabelle für Stammdaten handelt. Innerhalb der Tabellen der Stammdaten darf der Primärschlüssel jeweils nur einmal vorkommen. Im Gegensatz dazu, können in allen anderen Tabellen mehrere Einträge zu demselben Schlüssel vorliegen. Daher unterscheidet sich die Logik bei der Dateneinpflegung in eine Stammdatentabelle von den restlichen Tabellen. Deshalb wird auf Basis des Namens der übergebenen Tabelle geprüft, ob es sich um eine Tabelle für

Stammdaten handelt. Daraufhin wird der jeweilige Primärschlüssel in der Variable „key_column“ zwischengespeichert. Bei der Tabelle für die Daten der Partneruniversitäten ist der Primärschlüssel der Name der Universität. Für alle weiteren Tabellen ist der Primärschlüssel die Matrikelnummer der Studierenden. In einem zweiten Schritt wird nun geprüft, ob ein Eintrag zu den übergebenen Primärschlüsseln vorliegt. Diese Prüfung wird für jeden Eintrag einzeln durchgeführt, sodass die Einträge individuell behandelt werden können. Dadurch kann gewährleistet werden, dass Fehler durch die Einpflegung von mehrfachen Primärschlüsseln vermieden als auch neue Einträge regulär eingepflegt werden. Falls ein Primärschlüssel in der Tabelle bereits vorliegt, somit der Studierende oder die Partneruniversität bereits in den Stammdaten existiert, wird der Eintrag durch ein Update-Statement aktualisiert. Obwohl Stammdaten häufig konstant sind, werden auch diese Daten in gewissen Situationen aktualisiert, sodass die Möglichkeit diese automatisiert zu aktualisieren und dennoch die Integrität der Daten zu bewahren wesentlich für eine langfristige Planung ist. In dem Fall, dass der Primärschlüssel noch nicht vorhanden ist, wird der Eintrag regulär in die Datenbank eingepflegt. Alle weiteren Tabellen werden ebenfalls auf doppelte Einträge bei der Einpflegung neuer Datenmengen überprüft. In diesem Fall werden nicht die Primärschlüssel verglichen, sondern die einzelnen Tupel. Es gilt zu erwähnen, dass die Überprüfung lediglich alle Attribute beinhaltet, welche während des Datenimports relevant sind, somit die Daten, die aus den Formularen extrahiert werden. Das Vorliegen eines identischen Tupels sorgt nicht für das Aktualisieren des Eintrages, da die einzelnen Attribute alle identisch sind. Somit werden nur Einträge hinzugefügt, die sich mindestens innerhalb eines Attributes unterscheiden oder vollkommen neu sind. Die Anzahl der bereits existierenden Tupel wird innerhalb der Applikation für jede Datenmenge gezählt und in einer Variable gespeichert. Am Ende des Datenimportes, werden die Anzahl der bereits existierenden Tupel an den Benutzer über die Info-Box ausgegeben.
Wie bereits in Abschnitt 2.3 erläutert, werden innerhalb eines Insert-Statements in SQL zunächst die Spaltennamen definiert, die befüllt werden sollen, gefolgt von den Informationen, welche eingepflegt werden. Die folgenden Zeilen erstellen die Hülle für ein Insert-Statement in SQL, wobei die Länge des Statements abhängig von den übergebenen Datenbankspalten ist:

Algorithmus 32: Programmcode zur generischen Erstellung der INSERT-Skripte in SQL

```
1.         if table_name == "ERASMUS_Outgoings_Anerkennung":
2.             placeholders = ', '.join(['?'] * len
3.             (matching_columns))
4.             placeholders += ', ?, ?'
5.             columns_string = ', '.join([f"[{col}]" for col
6.             in matching_columns]) +
7.             ', [frist_finale_bearbeitung],[frist_lehrstuhl]'
8.      else:
9.          placeholders = ', '.join(['?'] *
10.            len(matching_columns))
11.            columns_with_brackets = [f"[{col}]" for col in
12.            matching_columns]
13.            # Joining the column names with commas
14.            columns_string = ', '.join(columns_with_brackets)
15.   sqlInsert = f"INSERT INTO {table_name} ({columns_string})
16.               VALUES ({placeholders})"
```

Hierbei werden zwei Fälle definiert, der erste gilt für den Fall, dass Daten in die Tabelle “ERASMUS_Outgoings_Anerkennungsdaten” eingepflegt werden und der andere Fall gilt für alle wei-

teren Datenimporte. Die Ausnahme der Anerkennungsdaten wird definiert, da bei der Einpflegung der Daten zwei Spalteninhalte innerhalb der Funktion berechnet werden müssen und nicht aus der entsprechenden Excel-Tabelle entnommen werden. Hierbei handelt es sich um die Einträge für die Fristen zur Nominierung und die Fristen der Lehrstühle um die Anträge zur Anerkennung zu bearbeiten. Beide Einträge gehören zum Datentyp “Datum” und werden auf Basis des Tages der Einpflegung der Daten automatisch erstellt und das SQL-Statement wird um diese beiden Einträge ergänzt. Der zweite Fall gilt für alle weiteren Tabellen und benötigt diesen zusätzlichen Schritt nicht, da die beiden Spalten nur in der Anerkennungstabelle existieren. Der erste Schritt umfasst die Erstellung von Platzhaltern für die einzufügenden Daten, wobei die Anzahl der Platzhalter mit der Anzahl der Spalten korrespondiert. Anschließend wird eine Zeichenfolge generiert, die die Namen der Spalten miteinander verknüpft. Das Insert-Statement wird schließlich aus diesen Komponenten zusammengesetzt, wodurch ein exemplarisches SQL-Statement wie folgt aussieht:

Algorithmus 33: Beispiel einer Ausgabe der INSERT-Skripte

```
1. INSERT INTO ERASMUS_Outgoings_Anerkennungsdaten
2. ('Vorname', 'Nachname', 'Matrikelnummer')
3. VALUES ('Max', 'Mustermann', '678123')
```

Das Insert-Statement wird in einem Zwischenspeicher abgelegt, damit die Hülle am Ende mit den Daten befüllt werden kann. Basierend auf der Tabelle, welche betrachtet wird, werden nun die Dateneinträge in das SQL-Statement übernommen oder die zwei bereits aufgezeigten Datumsspalten berechnet und ergänzt. Der letzte Schritt besteht in der Aktualisierung der Infobox mit der Nachricht über den erfolgreichen Datenimport oder einer Fehlermeldung. Danach wird die back_to_root-Funktion aufgerufen, welche die „current_widgets“ ausblendet und die „root_widgets“ wieder einblendet, sodass der Prozess abgeschlossen ist.
Damit die Fehlerbehandlung verbessert wird und der Benutzer wiederkehrende Fehler vermeiden und Fehler außerhalb der Datenapplikation selbst beheben kann, ist zuletzt eine Verbesserung der Ausgabe an den Benutzer vorgesehen. Die vordefinierten Funktionen innerhalb der Python-Bibliotheken haben unterschiedliche Fehlermeldungen basierend auf der spezifischen Situation, in der sie eingesetzt werden. Einige spezifische Fehler besitzen spezifische Fehlernummern und Fehlertypen. Dem Benutzer sollen lediglich Fehlermeldungen angezeigt werden, welche leicht verständlich sind und deren Ursache nicht durch eine Anpassung der Funktionen innerhalb der Applikation behoben werden muss. Im Verlaufe der Entwicklung dieser Applikation haben sich regelmäßig zwei Fehler aufgetan, welche durch die Excel-Datei mit den importierenden Daten ausgelöst werden. Die beiden Fehler werden anhand eines Beispiels erläutert und die Ausgabe für den Benutzer, sowie der Lösungsweg aufgezeigt. Bei dem ersten Fehler handelt es sich um einen logischen Fehler in dem Prozessablauf der Dateneinpflegung. Die Reihenfolge, in der die Daten in die unterschiedlichen Tabellen eingepflegt werden, ist in einigen Fällen relevant. Durch die bereits erläuterte Relation von Stammdatentabellen und allen anderen Tabellen ist das Einpflegen von Daten zuerst in die Stammdatentabellen notwendig. Beispielsweise können Bewerbungsdaten eines Incomings nicht ohne vorherige Einpflegung der Stammdaten dieses Incomings importiert werden, da die Relation dieser beiden Tabellen diesen Schritt untersagt. Das bereits erläuterte Prinzip der referenziellen Integrität untersagt das Einpflegen von Einträgen in die Bewerbungsdaten, wenn bisher kein Eintrag mit demselben Primärschlüssel bereits in den Stammdaten vorliegt. Insofern dennoch versucht wird zuerst Daten in die kontextbezogenen Tabellen einzupflegen wird folgende Fehlermeldung innerhalb der Applikation angezeigt:

('23000', "[23000] [Microsoft][ODBC-Treiber für Microsoft Access] Der Datensatz kann nicht hinzugefügt oder geändert werden, da ein Datensatz in der Tabelle 'ERASMUS_Incomings_Stammdaten' mit diesem Datensatz in Beziehung stehen muss. (-1613) (SQLExecDirectW)")

Auf Basis dieser Fehlermeldung wird eine vereinfachte Form dieser Fehlermeldung an den Benutzer innerhalb des Informationsfeldes ausgegeben. In Abbildung 19 ist die Ausgabe für den Benutzer für diesen spezifischen Fehler abgebildet. In diesem Fall ist der Lösungsweg innerhalb der Ausgabe bereits mitgeliefert, da zuerst die Stammdaten und in diesem Beispiel dann die Bewerbungsdaten des Incomings gepflegt werden müssen.

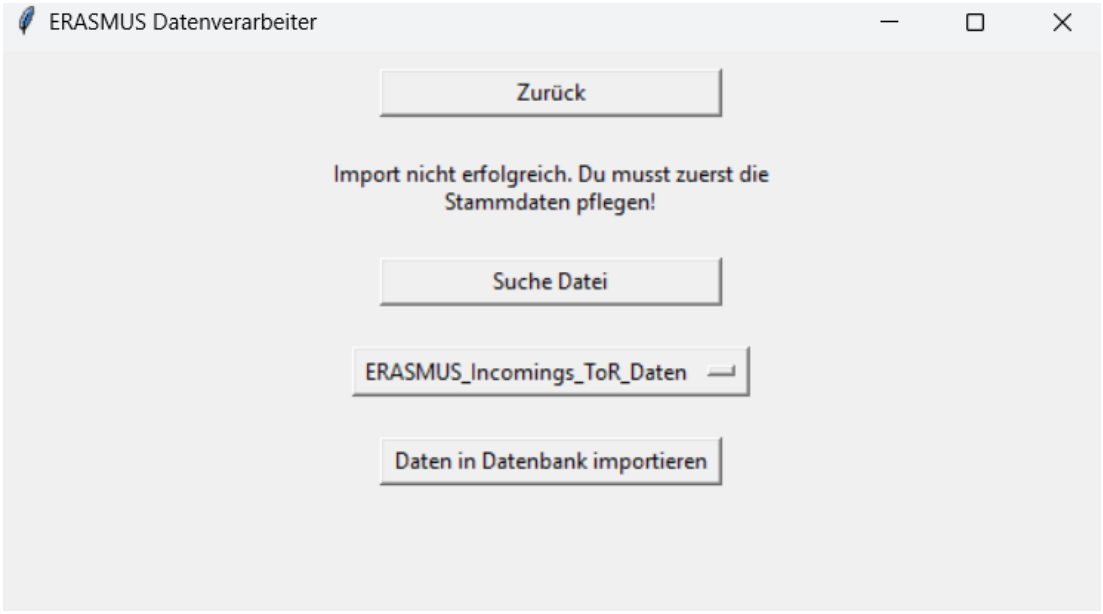


Abbildung 19: Current-Ebene des Datenimports bei fehlenden Stammdaten

Der zweite Fehler kann durch die Anpassung der Formulare bereits vermieden werden, tritt dennoch häufig bei manuellen Anpassungen der Excel-Dateien auf. Die Fehlermeldung des zweiten Fehlers lautet wie folgt:

('22018', '[22018] [Microsoft][ODBC-Treiber für Microsoft Access] Datentypenkonflikt in Kriterienausdruck. (-3030) (SQLExecDirectW)')

Der Fehler wird durch einen Konflikt des Datentyps aus der Excel-Datei mit dem Datentyp des entsprechenden Attributs aus der Tabelle in der Datenbank ausgelöst. Da die Formulare bereits in einem vorherigen Abschnitt so angepasst sind, dass sie die Eintragung unzulässiger Datentypen bereits untersagen, tritt dieser Fehler nur bei manueller Anpassung der Excel-Dateien auf. Dennoch gilt es zu erwähnen, dass diese Ausnahmesituationen ebenfalls beachtet und behandelt werden müssen. Dadurch wird die Stabilität der Prozesse auch in Grenzfällen gewährleistet. Die Ausgabe für den Benutzer in erneut vereinfachter Form der Fehlermeldung ist in Abbildung 20 dargestellt. Die Fehlerbehandlung für den zweiten Fehler ist nicht so intuitiv, wie für den ersten Fehler. Die Fehlermeldung gibt lediglich den Grund und nicht den spezifischen Ursprung des Fehlers mit aus, sodass nicht klar definiert ist welcher Eintrag diesen Fehler hervorgerufen hat. Dadurch muss die Excel-Datei manuell auf Unstimmigkeiten

überprüft werden. In zahlreichen Fällen sind diese Ausnahmen auffällig, da der Fehler häufig auftritt, wenn anstatt einer Zahl Buchstaben hinzugefügt werden, oder ein Datum nicht vorliegt.

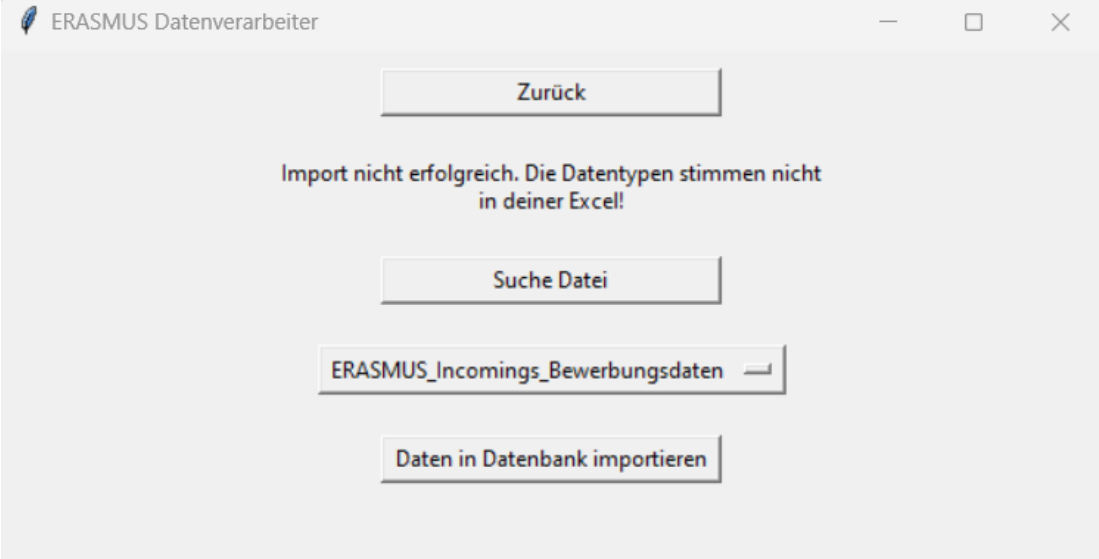


Abbildung 20: Current-Ebene des Datenimports bei einem Datentypenkonflikt

4.2 Automatisierte Daten-Exporte und Dokumentenerstellungen

Die Reihenfolge der Definition der automatisierten Funktionen ist bewusst so gewählt, dass die Importfunktionen zuerst definiert werden und daraufhin die Export- und Dateierstellungsfunktionen. Das liegt daran, dass die Datenimportfunktionen die Grundlage für alle weiteren Funktionen bilden. Sie ermöglichen das Einlesen von Dateien und das Auslesen von Daten aus der Datenbank. Die Datenerstellungsfunktionen benötigen eine zusätzliche Funktionsebene, die der Befüllung und Anpassung von Dateivorlagen. Die Vorlagen liegen als Word-Dateien vor und werden bisher manuell befüllt. Es ist erneut darauf hinzuweisen, dass die manuelle Eingabe von Daten fehleranfällig ist und erheblichen Arbeitsaufwand verursacht. Der erste Schritt zur Datenbefüllung besteht in der Umwandlung der vorliegenden Word-Dateien in das PDF-Vorlagenformat. Im Gegensatz zu Word-Vorlagen ermöglichen PDF-Vorlagen die Definition einzelner Felder mit spezifischen Datentypen und Feldnamen. Zwar ist eine ähnliche Umsetzung in Word möglich, erfordert jedoch einen weitaus komplexeren Ansatz. Darüber hinaus gibt es eine größere Anzahl von Python-Bibliotheken, die die Befüllung von PDF-Vorlagen unterstützen, weshalb PDF-Vorlagen die Grundlage für die weiteren Funktionen bilden. Die Feldnamen werden gemäß den Spaltennamen der Tabellen definiert, um Einheitlichkeit zu gewährleisten. Aufgrund der erheblichen Unterschiede zwischen den einzelnen Vorlagen kann keine generische Funktion definiert werden, die alle Befüllungen der unterschiedlichen Vorlagen umfasst. Daher wird für jede der Funktionen in der zweiten current-Ebene eine separate Hauptfunktion definiert. Die Funktionen, welche den Knöpfen in der root-Ebene und der ersten current-Ebene zugeordnet sind, sind generisch aufgebaut, sodass sie für alle drei Prozesse wiederverwendet werden können.

In Abbildung 21 ist die erste current-Ebene der ToR-Erstellung abgebildet, in der zunächst das akademische Jahr ausgewählt werden muss. Dieser Schritt dient dazu, die Auswahl der Incomings bereits auf diejenigen zu beschränken, die einem spezifischen akademischen Jahr zugeordnet sind. Danach wird der Benutzer innerhalb des Infotextes darauf hingewiesen einen Ablageordner für die finale Datei auszuwählen. Hierfür eignen sich Ordnerstrukturen, sodass für jede der Funktionen eigene Ordner existieren und die einzelnen Dokumente klar voneinander getrennt abgelegt werden können. Nach der Auswahl des Ablageortes gelangt der Nutzer

in die zweite current-Ebene, welche in Abbildung 22 dargestellt ist. Diese ist von der Struktur her ähnlich zu der des Datenimports, damit dem Nutzer die Ausführung der Funktion durch den einfachen, intuitiven und verwandten Aufbau erleichtert wird. In der zweiten current-Ebene besteht der erste Schritt darin, einen Incoming aus einem Dropdown-Menü auszuwählen. Dank der vorherigen Auswahl des akademischen Jahres zeigt das Dropdown-Menü nur eine vorgefilterte Menge an Incomings an, was die Auswahl erheblich erleichtert. In diesem Fall wird eine Basisauswahl vorgeschlagen, die bei Bedarf angepasst werden kann. Daraufhin werden die Knöpfe für den Datenimport aktiviert. Hierbei gibt es einen Knopf für die Erstellung des ausgewählten Incomings und einen weiteren Knopf, um das ToR für alle Incomings aus dem Menü auf einmal zu erstellen. Dies gibt dem Nutzer die Möglichkeit, die Daten aller Incomings eines akademischen Jahres zu sammeln und alle Dateien gleichzeitig zu erstellen, um den Vorgang noch effizienter zu gestalten. Im Anschluss können alle ToRs an die entsprechenden Incomings versendet werden. Durch die Anpassung des Ablaufes wird sichergestellt, dass der Nutzer gezielt und effizient mit einer vorgefilterten Auswahl arbeiten kann, was die Übersichtlichkeit und die Effizienz der ToR-Erstellung erhöhen sollen.

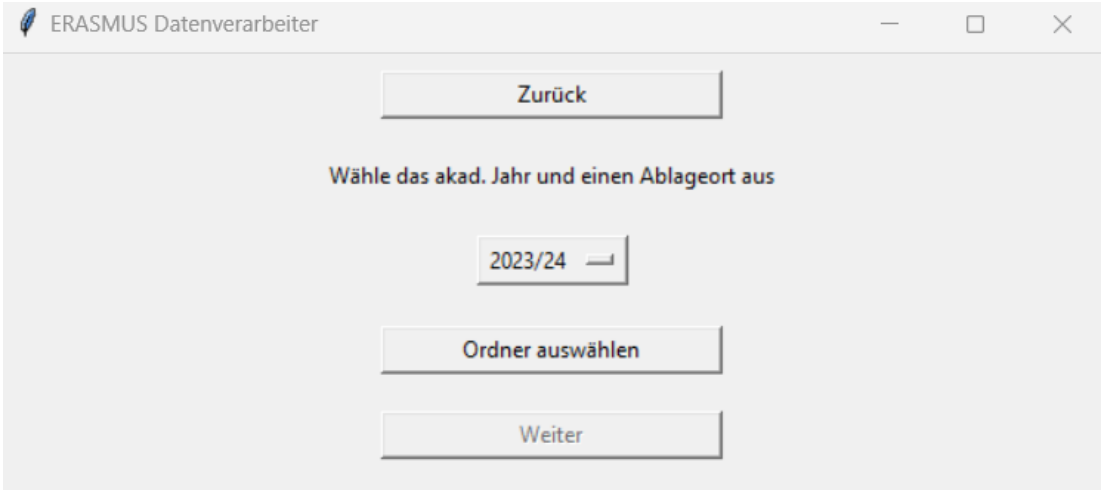


Abbildung 21: Erste current-Ebene der ToR-Erstellung

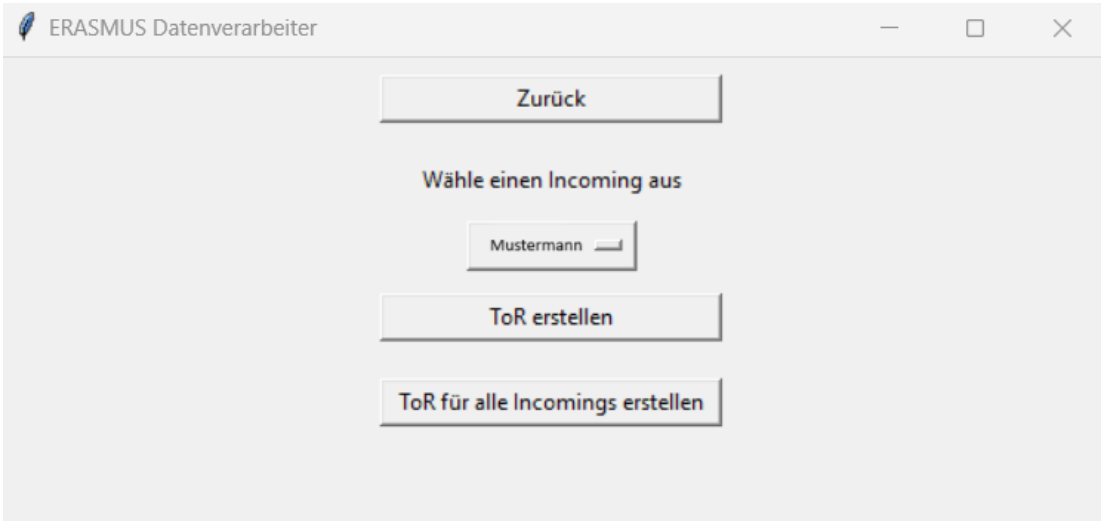


Abbildung 22: Zweite current-Ebene der ToR-Erstellung

Ein zentraler Aspekt beim Ausbau der Funktionen nach dem Datenimport ist die Wiederverwendbarkeit von Teilfunktionen. So kann beispielsweise die bestehende Funktion zur Auswahl einer Excel-Datei durch den Benutzer auch zur Auswahl eines Ablageordners umfunktioniert werden. Ebenso lässt sich das Dropdown-Menü zur Auswahl einer Tabelle in ein Dropdown-Menü zur Auswahl eines Incomings umgestalten. Darüber hinaus beziehen sich zusätzlich definierte Funktionen auf die Erstellung und Befüllung der PDF-Datei. Der fill_pdf_template-Funktion werden die Parameter des Namens des Incomings, der Pfad der PDF-Vorlage und der Pfad des Ablageordners übergeben. Innerhalb der Funktion wird zu Beginn ein Datenverzeichnis erstellt, welches die Verbindung der Tabelleneinträge zu den Feldern in der PDF-Vorlage herstellt. Da es sich bei dem ToR um eine Tabelle handelt, existieren auch hier mehrere Einträge zu denselben Spalten aus der Backend-Tabelle. Dadurch werden die Feldnamen in der PDF-Vorlage so definiert, dass jeder Name mit dem Spaltennamen beginnt und jedes Feld zusätzlich einen numerischen Wert enthält. Beispielhaft wird in Abbildung 23 aufgezeigt wie dadurch die Verbindung zwischen der Tabelle in der Datenbank und der Tabelle in der PDF-Vorlage gebildet wird. Die Abbildung zeigt die Verbindung für drei Einträge in der Backend-Tabelle für das ToR eines zuvor ausgewählten Incomings. Die Felder innerhalb der PDF-Vorlage müssen eindeutig sein und sollen dennoch demselben Schema folgen. Deshalb ist jedes Feld definiert als der Spaltenname addiert um die derzeitige Nummer des Eintrages. Dadurch ist die Zuordnung flexibel und erweitert sich automatisch mit zunehmenden Einträgen aus der Tabelle. In Abbildung 23 wird die Herkunft der Feldnamen in a) durch das farbliche Raster dargestellt, da jede Überschneidung aus Spaltenname und Zahl des Eintrages den dementsprechenden Feldnamen auf der rechten Seite widerspiegelt. Farblich sind jeweils die einzelnen Zeilen und demzufolge die Nummerierungen hervorgehoben, da die Funktion in Python die Zuordnung Zeile für Zeile herstellt. Die Funktion für die Zuordnung stoppt automatisch, wenn keine Zeilen für den Incoming in der Tabelle vorhanden sind. Dieser Aufbau fördert die Erweiterbarkeit des Systems und erleichtert die Erhöhung der Anzahl der möglichen Zeilen in der PDF-Vorlage.

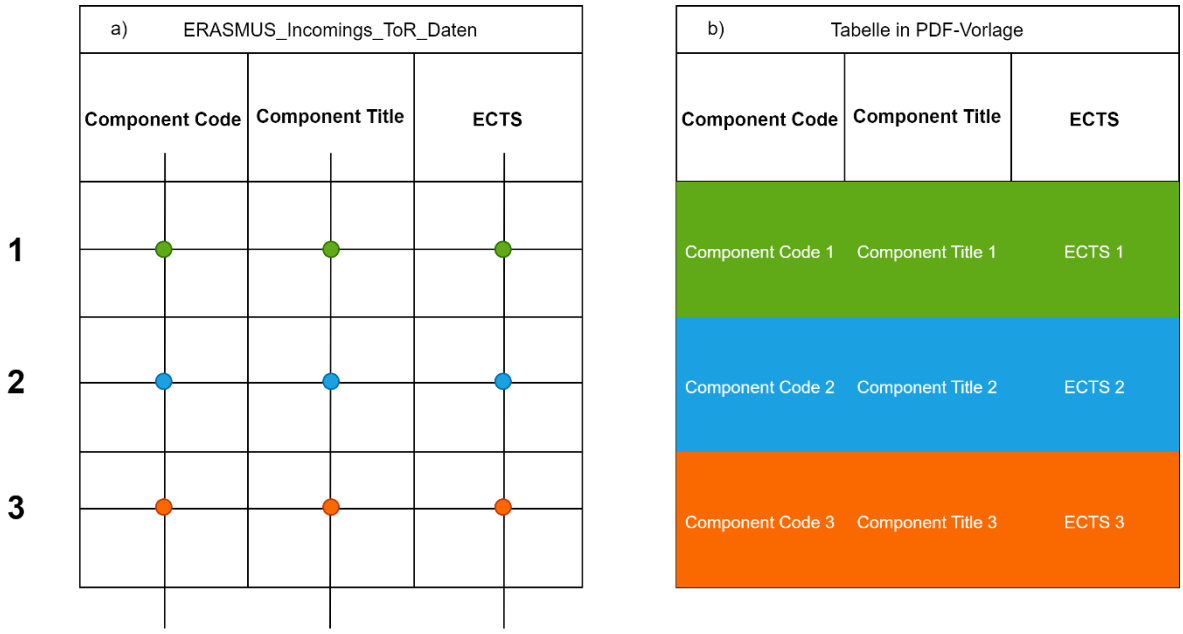


Abbildung 23: Zuordnung ToR Formfelder a) Struktur der Tabelle „ERASMUS_Incomings_ToR_Daten“ mit einem farblichen Raster b) Struktur der Tabelle in der PDF-Vorlage des ToR mit farblich hervorgehobenen Formfeldern

Nachdem die Zuordnung für die Tabelle der ToR-Vorlage der Incomings vollständig definiert ist, fehlt noch die Berechnung der Summe der ECTS der bestandenen Prüfungen der Incomings. Dafür werden in einer Schleife alle Einträge durchlaufen und die Summe aus den ECTS gebildet. Hierbei sollen nur bestandene Prüfungen betrachtet werden, sodass bei jeder Iteration geprüft wird, ob das jeweilige Fach bestanden wurde und zu der Summe hinzugefügt werden soll. Falls eine Prüfung nicht bestanden wurde, wird die Anzahl an ECTS nicht der Summe an ECTS hinzugefügt und der nächste Eintrag betrachtet. Die write_fillable_pdf-Funktion aus der fillpdfs-Bibliothek ermöglicht nun das Beschreiben einer PDF-Vorlage mit dem Pfad der Vorlage, dem Ablageort der finalen Datei und dem Datenverzeichnis als Eingabeparameter. Dadurch werden die zuvor definierten Textfelder in der PDF-Vorlage mit den eigenen Daten befüllt und die Datei abgespeichert. Ein letztes Problem besteht in der Überlagerung der Texteinträge und der Formfeldmasken, sodass der automatisch eingepflegte Text optisch hinter den Formfeldern erscheint. Ein möglicher Lösungsweg ist das Löschen der Formfelder, welches in einigen Fällen dazu führen kann, dass die Texteingaben auch gelöscht werden. In diesem Fall bietet sich eine zweite Variante an, bei der die ausgefüllte PDF-Vorlage mit einer flatten-Funktion von einer Vorlage in eine PDF-Datei umgewandelt wird. Hierdurch verliert die PDF-Vorlage die Bearbeitungsebene und die Möglichkeit Formfelder auszufüllen und es wird eine oberflächliche Kopie der ausgefüllten Datei gemacht. Dieser Schritt ergibt auch hinsichtlich der dadurch entfallenden Möglichkeit zur nachträglichen Bearbeitung der PDF-Datei Sinn, insofern eine nachträgliche Bearbeitung nicht gewünscht ist. Mit der Beendigung der flatten-Funktion ist ein Durchlauf für die Erstellung des ToR abgeschlossen. Die Möglichkeit mehrere ToRs gleichzeitig zu erstellen, wird durch die Ausführung des Prozesses innerhalb einer Schleife umgesetzt, die für jeden Benutzernamen in der Tabelle ausgeführt wird, sodass alle ToRs für jeden Benutzer aus der Datenbank auf einmal erstellt werden. Abschließend wird ein Infotext an den Anwender über die erfolgreiche Erstellung des ToR zurückgegeben.
Im Hinblick auf die Outgoings ist der Nominierungsprozess, wie er in Abschnitt 3.3 herausgearbeitet wird ein weiteres Optimierungsfeld. Die erste current-Ebene der Nominierungsfunktion ist in Abbildung 24 dargestellt, welche von der Struktur identisch zu der ersten current-Ebene der ToR-Erstellung ist, da sie dieselben Grundfunktionen teilen. Ebenso gibt es innerhalb dieser Funktion zwei current-Ebenen, sodass der Anwender durch Betätigung des Knopfes in der root-Ebene in die erste Ebene geleitet wird, welche in Abbildung 24 abgebildet ist. Auf dieser Ebene wird der Benutzer dazu aufgefordert ein akademisches Jahr und einen Ablageort für den Nominierungsbescheid auszuwählen. Ein Outgoing wird nur jeweils für das momentane akademische Jahr nominiert, sodass der Nominierungsbescheid nur für die Daten der Outgoings aus dem derzeitigen akademischen Jahr benötigt werden, wodurch sich eine Vorfilterung anbietet. Damit sich die Auswahl der akademischen Jahre entsprechend der Einträge in der Datenbank automatisch erneuert und keine Werte fest gesetzt werden müssen, wird die get_years_from_table-Funktion definiert, welche aus einer beliebigen Tabelle alle eindeutigen Einträge für das akademische Jahr zurückgibt. Dadurch werden in dem Dropdown-Menü für das akademische Jahr alle Jahre hinterlegt, welche genauso in der entsprechenden Tabelle in der Backend-Datenbank vorliegen. Somit wird eine automatische Erweiterung des Menüs gewährleistet, wenn weitere Einträge mit neuen akademischen Jahren in die Tabellen eingepflegt werden. Die zweite current-Ebene, wie sie in Abbildung 25 dargestellt ist, dient zur Auswahl des Outgoings innerhalb eines weiteren Dropdown-Menüs durch den Endanwender. Die Auswahl zwischen der Erstellung einer Datei und der vollumfänglichen Erstellung von Dateien zu allen vorhandenen Dateneinträgen wird auch hier implementiert, weshalb zwei Knöpfe in Abbildung 25 zur Erstellung des Nominierungsbescheides dargestellt sind. Beide Knöpfe führen bei Betätigung ein und dieselbe Funktion aus, dennoch werden bei der einfachen Erstellung nur der Name eines Outgoings und im Falle der massenhaften Erstellung die Namen aller Outgoings des ausgewählten akademischen Jahres als Liste an die Funktion übergeben. Der Ablauf innerhalb der dadurch ausgeführten fill_pdf_template-Funktion ist identisch zu derselben Funktion bei der ToR-Erstellung. Dennoch unterscheiden sich die beiden Datenverzeich-

nisse deutlich, da sich die PDF-Vorlagen vollkommen unterscheiden. Der Nominierungsbescheid beinhaltet keine Tabelle und besteht ausschließlich aus einem Fließtext, in den einige Textfelder eingebettet sind. Die Textfelder enthalten Informationen über die persönlichen Daten des Outgoings, den Namen der Partneruniversität und das akademische Jahr in dem das Auslandsemester geplant ist. Dadurch entfallen in dieser Funktion weitere Rechenschritte. Dennoch wird auch hier eine Zuordnung der Felder im Fließtext zu den Datenbankeinträgen benötigt. Die Komplexität der Zuordnung ist deutlich geringer, da für jeden Nominierungsbescheid nur eine Zeile aus der Backend-Datenbank benötigt wird und somit jede Spalte exakt einem Feld in der PDF-Vorlage zugeordnet werden kann.

Abbildung 24: Erste current-Ebene zur Erstellung des Nominierungsbescheides

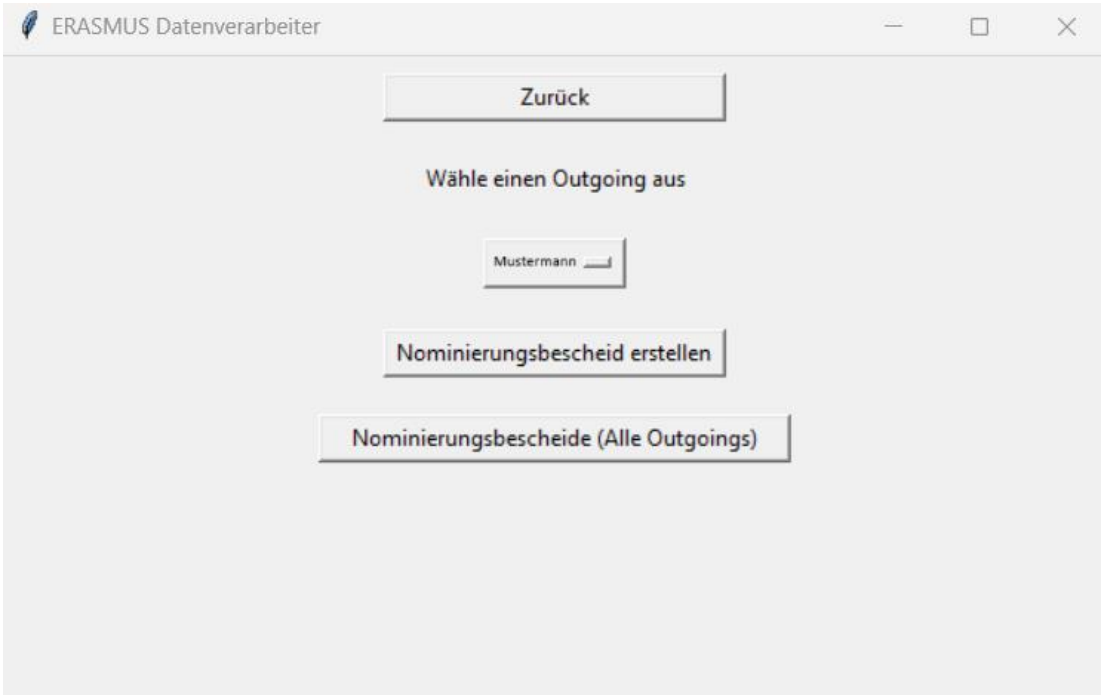


Abbildung 25: Zweite current-Ebene zur Erstellung des Nominierungsbescheides

Die dritte und letzte PDF-Vorlage, welche innerhalb des Projekts eingesetzt wird, ist die Vorlage für den Anerkennungsprozess. Im Kontrast zu dem ToR der Incomings ist die Anerkennungsdatei keine Leistungsbescheinigung, obwohl sie Daten zu den absolvierten Fächern der Outgoings enthält. Die Anerkennungsvereinbarung ist ein Austausch zwischen dem Outgoing, dem Projektteam und einem spezifischen Lehrstuhl an der TU Dortmund. Das Dokument ist die Voraussetzung der Anerkennung eines absolvierten Kurses innerhalb des Austauschsemesters. Das Dokument ist nicht tabellarisch aufgebaut und umfasst somit lediglich einen Kurs pro Dokument. Da ein Studierender im Schnitt drei bis vier Kurse an einer Partneruniversität belegt müssen auch drei bis vier Anerkennungsvereinbarungen pro Studierendem erstellt werden. Zudem übersteigt die Anzahl der Outgoings, die der Incomings in erheblichem Maße, sodass die Zeitersparnisse in diesem Prozessschritt groß sind. Die beiden current-Ebenen der Funktion für die Anerkennungsvereinbarung sind von der Struktur identisch mit den Ebenen der anderen beiden bereits aufgezeigten Funktionen. In der ersten current-Ebene müssen erneut ein gewünschtes akademisches Jahr und ein Ablageort ausgewählt werden. Die zweite current-Ebene ist in Abbildung 26 dargestellt und bietet die Möglichkeit auf Basis einer Auswahl eines Outgoing die zugehörige Anerkennungsvereinbarung für diesen Studenten zu erstellen. Da ein Outgoing drei bis vier Kurse im Durchschnitt belegt, wird pro Outgoing nicht ein einzelnes Dokument, sondern eine variable Menge an Dokumenten erstellt. Die Anzahl der Dokumente hängt von der Anzahl der Dateneinträge in der Tabelle für die Anerkennungsdaten des ausgewählten Studenten für das ausgewählte akademische Jahr ab. Die Dokumentenbezeichnung enthält neben dem Nachnamen des jeweiligen Outgoing auch die Kurzbezeichnung des absolvierten Kurses, der innerhalb des Dokumentes dokumentiert ist. Innerhalb der zweiten current-Ebene besteht zudem die Möglichkeit, Anerkennungsvereinbarungen für alle Outgoings eines bestimmten akademischen Jahres zu generieren. In diesem Fall werden für jeden einzelnen Outgoing des ausgewählten akademischen Jahres sowie für jeden individuellen Kurs entsprechende Anerkennungsvereinbarungen automatisch erstellt. Aufgrund der Vielzahl an Dateneinträgen und der Fähigkeit, diese Informationen in Echtzeit zu extrahieren und darauf basierend die benötigten Dokumente innerhalb weniger Sekunden zu erstellen, ist dieser Prozess erheblich effizienter und zeitsparender gestaltet.

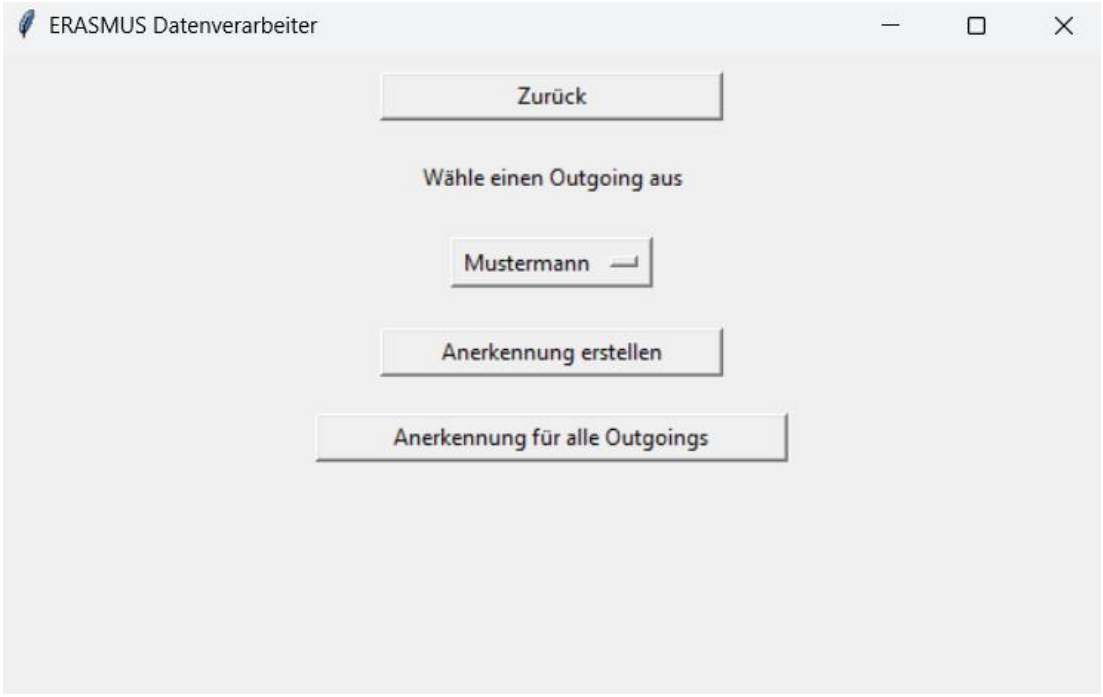


Abbildung 26: Zweite current-Ebene zur Erstellung der Anerkennungsvereinbarung

Im Rahmen des Projektes wird eine Konfigurationsdatei im INI-Format erstellt, die als zentrale Stelle zur Verwaltung aller relevanten Parameter und Einstellungen dient. Diese config.ini-Datei ermöglicht es, sämtliche im Python-Code verwendeten Zeichenketten und Pfade zentral zu definieren und zu steuern. Durch diese methodische Trennung von Code und Konfiguration wird nicht nur die Flexibilität des Codes erhöht, sondern auch die Wartbarkeit und Anpassungsfähigkeit des gesamten Systems erheblich verbessert. Die Konfigurationsdatei enthält Einträge für eine Vielzahl von Parametern, die für den Betrieb des Datenverarbeitungs-Tools essenziell sind. Dazu gehören insbesondere SQL-Statements, die in verschiedenen Funktionen innerhalb des Tools verwendet werden. Indem diese SQL-Statements in der Konfigurationsdatei hinterlegt werden, können sie bei Bedarf ohne Änderungen an dem Python-Code angepasst werden. Dies erleichtert die Anpassung an unterschiedliche Datenbankumgebungen oder spezifische Abfrageanforderungen, die sich im Verlauf des Projektes entwickeln können. Darüber hinaus werden in der config.ini-Datei auch die Namen der Formularfelder in den PDF-Vorlagen festgelegt. Diese Felder werden im Python-Code als Platzhalter referenziert, wodurch eine einfache Anpassung der Vorlagen ermöglicht wird. Sollten sich die Namen oder die Anzahl der Felder in den Vorlagen ändern, kann dies ohne Eingriffe in den Code über die Konfigurationsdatei erfolgen. Dies ist besonders vorteilhaft in Szenarien, in denen unterschiedliche Versionen einer Vorlage verwendet werden sollen oder wenn eine Vorlage an geänderte Anforderungen angepasst werden muss. Ein weiterer wichtiger Aspekt der Konfigurationsdatei ist die Verwaltung der Spaltennamen und Tabellennamen in den verwendeten Datenbanken. Auch hier trägt die zentrale Definition in der Konfigurationsdatei dazu bei, dass Anpassungen und Änderungen an der Datenstruktur ohne großen Aufwand im Code vorgenommen werden können. Dies bietet insbesondere in Entwicklungsprojekten, die auf unterschiedlichen Datenbankschemata basieren, einen erheblichen Vorteil, da die Konfigurationsdatei eine Abstraktionsschicht bildet, die es ermöglicht, Codeänderungen auf ein Minimum zu beschränken. Um die Nutzung der Konfigurationsdatei im Python-Code zu ermöglichen, wird eine spezielle Bibliothek verwendet, die das Einlesen und Verarbeiten der Konfigurationsdatei ermöglicht. Diese Bibliothek stellt sicher, dass die im Code verwendeten Platzhalter durch die entsprechenden Werte aus der Konfigurationsdatei ersetzt werden. Durch die Verwendung der Konfigurationsdatei in Verbindung mit der eingesetzten Bibliothek wird eine hohe Modularität und Flexibilität des Codes erreicht. Dies ermöglicht zukünftigen Entwicklern, auf sich ändernde Anforderungen schnell zu reagieren, ohne tiefgreifende Änderungen am bestehenden Code durchführen zu müssen. Die Konfigurationsdatei bildet somit eine zentrale Komponente des Projekts, die die Anpassbarkeit und Erweiterbarkeit der Lösung maßgeblich unterstützt. Diese Struktur trägt entscheidend zur Qualität und Zukunftsfähigkeit der entwickelten Applikation bei und stellt sicher, dass sie auch in komplexen und sich dynamisch entwickelnden Umgebungen erfolgreich eingesetzt werden kann.

5 Implementierung und Validierung von Datenbankfunktionalitäten in bestehender Systemlandschaft

Dieses Kapitel konzentriert sich auf die Implementierung und Validierung der entwickelten Datenbankfunktionalitäten in die bestehende Systemlandschaft. Zunächst wird ein Zugriffskonzept entwickelt und implementiert, um sicherzustellen, dass nur berechtigte Benutzer auf die Daten zugreifen können. Anschließend werden das Datenbanksystem und der automatisierte Datenverkehr umfassend validiert, um die Funktionalität und Integrität der Daten sicherzustellen. Abschließend werden die Ergebnisse diskutiert und interpretiert, um die Effektivität der Implementierung zu bewerten und mögliche Verbesserungspotenziale zu identifizieren.

5.1 Implementierung eines Zugriffskonzeptes

Ein wesentliches Problem besteht derzeit in dem nicht vorhandenen Zugriffskonzept, sodass die Gefahr einer Datenmanipulation oder dem unbeabsichtigten Löschen von Daten hoch ist. Die Planung der Implementierung eines Zugriffskonzeptes kann bereits helfen Schwachstellen innerhalb der Systemarchitektur und besonders der Zuweisung der Zugriffsberechtigungen zu erkennen. In der bisherigen Systemlandschaft werden den Benutzern alle Berechtigungen zugewiesen, sodass keine klare Hierarchie definiert ist und das Risiko des Datenverlustes erhöht wird. Deshalb wird in einem ersten Schritt definiert, welche Mitarbeiter die Aufgabenbereiche innerhalb des Projektteams übernehmen oder in Zukunft übernehmen sollen. Die Aufteilung und Darstellung der Aufgabenbereiche kann neben der Minderung des Datenverlustrisikos auch weitere Verbesserungen herbeiführen. Strukturelle Unterstützung bei der Arbeitsgestaltung kann die Arbeitsleistung und die Rollenergebnisse verbessern, insbesondere wenn die Mitarbeiter Klarheit über die Rollen der anderen haben (Parker et al. 2013). Eine gut durchdachte Organisationsstruktur erleichtert klare Autoritätsbeziehungen, Kommunikationsmuster und Entscheidungsprozesse (Dubey und Singhal 2016).

Bei der Planung eines Zugriffskonzeptes müssen vor allem Ausnahmesituationen beachtet und mit eingeplant werden. Die Auslegung eines Zugriffskonzeptes auf Basis einer Idealvorstellung ist nicht empfehlenswert, da ungeplante Situationen zu einem Stillstand des Projektes und der internen Prozesse führen können. Für Notfallsituation innerhalb großer Datenbanken und Systeme werden in der Industrie Notfallbenutzersysteme eingesetzt, welche in bestimmten Situationen einen weitgreifenden Zugriff auf die internen Systeme ermöglichen und so dem regulären Zugriffskonzept entgehen. Das Modell der rollenbasierten Zugangskontrolle für Notfälle beinhaltet eine „Break the Glass“-Richtlinie, die es den Nutzern ermöglicht, Zugangskontrollen zu umgehen und gleichzeitig die Rechenschaftspflicht zu wahren (Nazerian et al. 2019). So wird neben der Projektleiterin des Projektes auch ein Vertretungsbenutzer vorgesehen, sodass im Falle von einem geplanten Urlaub oder auch einer unbeabsichtigten Abwesenheit, die Zugriffe auf die Daten weiterhin möglich sind. Zudem ergeben sich regelmäßig Probleme, die ein fachlicher Mitarbeiter durch fehlende Expertise in dem Umgang mit Datenbanken und SQL-Abfragen nicht lösen kann, sodass ein IT-Mitarbeiter als weiterer Akteur im Projekt vorgesehen ist. Die fachlichen Mitarbeiter und studentischen Hilfskräfte werden die Datenbankabfragen am häufigsten Nutzen und stellen deshalb das größte Risiko einer ungewollten Datenbankmanipulation dar. Auf Basis der Prozessabläufe, wie sie bereits in Abschnitt 3.3 beleuchtet werden, können in einem nächsten Schritt die Berechtigungen der einzelnen Mitarbeiter definiert werden. In Tabelle 6 sind in der linken Spalte die einzelnen bereits genannten Akteure aufgelistet. Die Tabelle umfasst dabei nur die Berechtigungen für den Zugriff des Datenbanksystems, folglich der Frontend- und Backend-Datenbank und nicht die Zugriffe für die Datenapplikation, da diese von jedem Mitarbeitenden genutzt werden soll und darf. Ein Zugriffskonzept für die Datenapplikation kann bei einer Ausweitung des Projektes oder bei der

Verwendung der Applikation in mehreren Projekten Sinn ergeben, ist aber auf Basis der Größe des derzeitigen Projektstandes und der Mitarbeiteranzahl nicht vorgesehen.

Die Berechtigungen für die Frontend- und Backend-Datenbank sind jeweils in eine Zugriffs- und eine Bearbeitungskomponente aufgeteilt. Die ersten zwei Zeilen in Tabelle 6 beinhalten die Berechtigungen für die Projektleiterin und deren Vertretung. Die Projektleiterin muss in der Lage sein auf die Backend-Datenbank zuzugreifen und Tabellen in dieser zu bearbeiten. Dieser Umstand ist durch mögliche Notfallsituationen zu begründen. Im Falle von nicht funktionierenden Abfragen in der Frontend-Datenbank oder der Notwendigkeit einzelne Tabelleneinträge zeitnah anzupassen, benötigt die Projektleiterin die Möglichkeit dies zu jedem Zeitpunkt umzusetzen. Dasselbe gilt deshalb auch für die Vertretung der Projektleiterin, da diese dieselben Tätigkeiten ausführen muss. Zusätzlich sollen die beiden Mitarbeitenden den Zugriff auf die Frontend-Datenbank und somit auf die Abfragen erhalten, so wie alle weiteren Mitarbeiter auch. Die Abfragen stellen das zentrale Element des Datenverkehrs dar, da sie im Regelfall alle notwendigen Informationen ausgeben und in sämtlichen Prozessen innerhalb des Projekts verwendet werden. Die Bearbeitung der Abfragen soll der Projektleiterin untersagt werden, da die technische Expertise fehlt, um die SQL-Statements adäquat anzupassen. In Tabelle 6 wird in der dritten Zeile der IT-Mitarbeiter aufgeführt, der sowohl die genannten als auch alle weiteren Berechtigungen erhält. Dieser IT-Mitarbeiter muss in der Lage sein, akute Probleme zu beheben, insbesondere in den Backend-Tabellen, falls Anpassungen von Spalten und Tabellen aufgrund von Änderungen in der Datenapplikation erforderlich werden. Darüber hinaus muss er in der Lage sein, entsprechende Anpassungen in den Abfragen vorzunehmen. Da sich das Projekt stetig verändert und wächst werden zukünftig weitere Abfragen hinzukommen, welche von Mitarbeitenden mit einer gewissen Kenntnis in SQL und Datenbanken umgesetzt werden müssen. Die fachlichen Mitarbeiter und studentischen Hilfskräfte erhalten jeweils die Berechtigung zum Zugriff auf die Abfragen, da diese den Hauptbestandteil der Prozesse unterstützen. Die Berechtigungen für die Backend-Datenbank sind diesen beiden Akteuren verwehrt, damit die Basisdaten konsistent, sicher und dauerhaft abgespeichert werden können.

Tabelle 6: Zugriffsmatrix für die Backend- und Frontend-Datenbanken

Mitarbeiter	**Zugriff Backend-Datenbank**	**Bearbeitung Backend-Datenbank**	**Zugriff Frontend-Datenbank**	**Bearbeitung Frontend-Datenbank**
Projektleiter/in	Ja	Ja	Ja	Nein
Projektleiter/in-Vertretung	Ja	Ja	Ja	Nein
IT-Mitarbeiter	Ja	Ja	Ja	Ja
Fachliche/r Mitarbeiter/in	Nein	Nein	Ja	Nein
Studentische Hilfskraft	Nein	Nein	Ja	Nein

Aus der Zugriffsmatrix lassen sich die Berechtigungen auf Benutzerebene im System ableiten. Die Begriffe „Zugriff“ und „Bearbeitung“ können in die technischen Termini „Lesen“ und „Schreiben“ überführt werden. Dateiberechtigungen in Systemen sind grundlegend für die Kontrolle des Zugriffs auf Dateien und Verzeichnisse. Sie bestehen aus Lese-, Schreib- und Ausführungsberechtigungen (Purcell 2012). Leseberechtigungen ermöglichen die Anzeige von Dateiinhalten oder Verzeichnislisten, Schreibberechtigungen ermöglichen die Änderung

von Dateien oder die Erstellung von Verzeichnissen, und Ausführungsberechtigungen erlauben die Ausführung von Dateien oder die Suche in Verzeichnissen (Purcell 2012). Einige Systeme unterstützen eine feinkörnige Zugriffskontrolle durch partielle Verschlüsselung von baumstrukturierten Dokumenten, sodass verschiedene Benutzer unterschiedliche Ebenen von Lese- und Schreibberechtigungen haben können (Kollmann 2007). Der Übersichtlichkeit halber wird jedoch zunächst nur zwischen den beiden grundlegenden Begriffen unterschieden. Der IT-Mitarbeiter erhält daher umfassenden Zugriff auf sowohl die Backend- als auch die Frontend-Datenbank, da er in der Lage sein muss, grundlegende Komponenten beider Datenbanken anzupassen. Die Berechtigung eines Benutzers zur Durchführung einer Aktion gewährleistet jedoch nicht automatisch, dass diese korrekt ausgeführt wird. Daher empfiehlt es sich, regelmäßig Backups anzufertigen, um einen funktionierenden Zustand der Datenbanken an einem sicheren Speicherort zu bewahren. Auf diese Weise können Fehler, die während der Entwicklung oder Überarbeitung der Datenbanken auftreten, rückgängig gemacht werden. Backups sind für den Schutz wichtiger Daten vor verschiedenen Bedrohungen wie Hardwareausfällen oder menschlichen Fehlern unerlässlich (Fromm 2009). Die fachlichen Mitarbeiter und studentischen Hilfskräfte erhalten ausschließlich lesende Berechtigungen für die Frontend-Datenbank und können die Datensätze einsehen, ohne diese bearbeiten zu können. Um dennoch Ausnahmesituationen zu berücksichtigen und den einzelnen Mitarbeitern eine flexiblere Arbeitsweise mit der Frontend-Datenbank zu ermöglichen, gilt die Zugriffsberechtigung auf die Frontend-Datenbank in Tabelle 6 nur für die zentral abgelegte Datei. Obwohl die Mitarbeiter nicht befugt sind, die ursprüngliche Datenbankvorlage zu bearbeiten, haben sie die Möglichkeit, eine Kopie der Frontend-Datenbank auf ihrem eigenen Rechner zu speichern. Dadurch können sie alle Abfragen anpassen oder neue hinzufügen. Der Vorteil dieses Ansatzes besteht darin, dass nur die Abfragen in der eigenen gespeicherten Datei verändert werden, während die ursprüngliche Template-Datei unverändert bleibt. Dies stellt sicher, dass der aktuelle, funktionierende Stand der Frontend-Datenbank zentral und für alle zugänglich aufbewahrt wird. Eigene Versionen sind für andere nicht zugänglich, wodurch Verwechslungen vermieden werden. Sollte das Anpassen der Abfragen aufgrund fehlender technischer Kenntnisse fehlschlagen, kann der aktuelle Stand jederzeit zentral abgerufen und der lokale Stand überschrieben werden. Auf diese Weise existieren stets nur eine zentrale Backend- und eine zentrale Frontend-Datenbank.

5.2 Implementierung des Datenbanksystems und der Datenapplikation

Der Transport des Datenbanksystems aus der Entwicklungsumgebung in die Produktivumgebung ist durch die Verwendung von Microsoft Access als DBMS mit geringem Aufwand möglich. Die zentrale Version der Backend- und Frontend-Datenbank können direkt in die Systemlandschaft transportiert und dort in einem geteilten Ordner abgelegt werden, auf den alle Mitarbeitenden Zugriff haben. Die Beziehung zwischen der Frontend- und Backend-Datenbank muss erneut hergestellt werden, da sich die Dateipfade geändert haben. Bei diesem Schritt kann der Tabellenverknüpfungs-Manager von Microsoft Access unterstützen, welcher nach der Auswahl einer Backend-Datenbank und der benötigten Tabellen aus dieser, die Tabellen aus den Abfragen der Frontend-Datenbank auf diese verweisen lässt. Daraufhin können die Berechtigungen der einzelnen Mitarbeiter aus Abschnitt 5.1 für die beiden Datenbanken umgesetzt werden. Hierbei bietet Windows die Möglichkeit die Berechtigungen für einzelne Dateien auf Benutzerebene festzulegen.

Für den Transport und Einsatz der Datenapplikation in die bestehende Systemlandschaft müssen verschiedene Vorbereitungsschritte durchgeführt werden. In der Entwicklungsumgebung liegt die Datenapplikation als Python-Programm vor. Mehrere Funktionen innerhalb des Programmes basieren auf verschiedenen Bibliotheken, welche innerhalb der Entwicklungsumgebung installiert werden. Bei einem Transport des Programms in die Produktivumgebung, werden lediglich die Programmzeilen ohne die Inhalte der Bibliotheken und weiteren Dateien exportiert. Eine weitere Voraussetzung für die Ausführung des Programms im Ausgangszustand ist die Programmiersprache Python selbst. Ohne eine Installation von Python auf dem jeweiligen Rechner kann das Programm nicht ausgeführt werden. Da diese Programmiersprache

nicht in der Produktivumgebung installiert ist, soll eine ausführbare Anwendung erstellt werden, welche alle benötigten Bibliotheken und Funktionen mitgeliefert bekommt. Des Weiteren müssen auch zusätzliche Dateien, wie zum Beispiel die Konfigurationsdatei oder Bilder und Icons von der Applikation jederzeit aufgerufen werden. Für die Konvertierung einer Python-Applikation in eine ausführbare Datei mit dem Dateityp „.exe“ gibt es zahlreiche Bibliotheken, die diesen Prozess unterstützen können. Die Funktionen innerhalb der Bibliotheken bieten die Möglichkeit über eine Benutzeroberfläche das Python-Programm auszuwählen und weitere relevante Konfigurationen vorzunehmen. So können auch Bilder und Dateien ausgewählt werden, welche bei dem Export mit in den Ordnern der ausführbaren Applikation hinterlegt werden. Zusätzlich können die Kürzel der einzelnen Bibliotheken mit angegeben werden, sodass die Inhalte der Bibliotheken heruntergeladen und als Skripte den Ordnern beigefügt werden. Dadurch kann die Applikation auf die Inhalte der Bibliotheken zu jederzeit und auch ohne eine Installation der Bibliotheken auf dem Rechner zugreifen.

Für die Validation der Datenapplikation werden unterschiedliche Validationsansätze kombiniert. Unit-Tests sind eine wichtige Praxis in der Softwareentwicklung, insbesondere für Python-Anwendungen. Sie helfen bei der Validierung einzelner Komponenten und stellen sicher, dass sie wie vorgesehen funktionieren (Chandrasekaran et al. 2019). Unit-Tests können zu saubereren, flexibleren Programmen führen und sind für agile Entwicklungsmethoden unerlässlich (David Sale 2014). Innerhalb der Entwicklungsumgebung wird mit vereinfachten Unit-Testskripten gearbeitet, welche die Funktionalität jeder einzelnen Funktion auf Basis von Testdaten automatisiert testen und auf Fehler überprüfen können. Die Validation der Benutzeroberfläche wird manuell vollzogen, indem die einzelnen Prozessabläufe und unterschiedlichen Entscheidungswege mehrfach abgelaufen werden. Durch das automatisierte Testen der Funktionen beschränken sich die manuellen Tests nur auf die korrekte Navigation zwischen den verschiedenen Ebenen und die korrekte Anzeige der Fenster und Knöpfe. Die Überprüfung der erstellen Dokumente und der eingepflegten Daten in die Backend-Datenbank wird auch manuell durchgeführt. Hierbei werden für jeden Prozess zehn unterschiedliche Dokumente auf Basis zehn unterschiedlicher Testdaten erstellt. Ebenso werden jeweils zehn Datensätze in jede Basistabelle importiert. Der Validationsprozess erfolgt iterativ, sodass final eine Fehlerrate von null angestrebt wird und in die zweite Validationsphase innerhalb der Produktivumgebung fortgeschritten werden kann. Innerhalb der Produktivumgebung werden Benutzertests durchgeführt, indem die Applikation mithilfe realer Daten und in realen Prozessen innerhalb des Projektes von den Mitarbeitenden getestet wird. Mehrere Studien haben gezeigt, wie wichtig iteratives Design und Nutzerevaluierung für die Schaffung nutzerfreundlicher Schnittstellen sind (Ollenschläger et al. 2022). Diese Tests helfen dabei, Handhabungsprobleme zu erkennen und die Nutzerzufriedenheit zu verbessern (Nielsen et al. 2012). Indem sie sich auf Anpassbarkeit, Benutzerfreundlichkeit und Benutzererfahrung konzentrieren, können Entwickler GUIs erstellen, die die Implementierungszeit erheblich reduzieren und die allgemeine Benutzerzufriedenheit verbessern (Ollenschläger et al. 2022). Ein wesentlicher Punkt hierbei ist, dass die Tests von unterschiedlichen Mitarbeitenden durchgeführt werden sollen, da die Funktionalität der Applikation bereits in der Testumgebung überprüft wird und der Fokus hierbei auf der Handhabung der Applikation gesetzt wird.

Damit ein Grundwissen über die Einsatzmöglichkeiten und die korrekte Ausführung der Applikation bei den Mitarbeitenden besteht, werden Workshops durchgeführt. Innerhalb der Workshops werden Prozesse erstmalig mithilfe der Applikation und der neuen Struktur des Datenbanksystems durchlaufen. Des Weiteren werden verschiedene Anleitungsdokumente bereitgestellt, welche die Abläufe der Prozesse innerhalb der Applikation detailliert beschreiben und auf mögliche Fehlerquellen eingehen. Zudem werden auch Lösungswege für gängige Fehlermeldungen mitgeliefert, um die Fehlerbehebung zu vereinfachen. Diese Dokumentation bildet eine grundlegende Basis und kann bei der Erweiterung der Applikation ausgebaut werden. Der gesamte Validierungsprozess erfolgt ebenfalls iterativ. Nachdem ein Entwicklungsstand innerhalb der Entwicklungsumgebung ohne Fehler hervorzurufen überprüft wird, kann dieser Stand in die Produktivumgebung transportiert werden. Innerhalb der produktiven Prozesse muss die Applikation variierenden Anforderungen gerecht werden, sodass die Fehler innerhalb

der Applikation von der Entwicklungsumgebung stark abweichen. Zudem entstehen neue Anforderungen an die Applikation, welche auf Basis der Tests in der Produktivumgebung definiert und in den Entwicklungsprozess der Applikation übernommen werden. Deshalb muss der Stand, welcher bereits die Entwicklungsumgebung verlassen hat, erneut in dieser weiterentwickelt und getestet werden.

5.3 Diskussion der Forschungsfragen und Fazit

Im Rahmen dieser Arbeit wurden mehrere Forschungsfragen untersucht, die sich mit der Optimierung von Altsystemen in akademischen Einrichtungen durch moderne Technologien und gezielte Verbesserungen befassten. Um die zentrale Fragestellung zu beantworten, wie bestehende Datenbanksysteme durch eine Neustrukturierung und den Einsatz von Automatisierungen effizienter gestaltet werden können, wurde eine systematische Analyse entlang spezifischer Forschungsfragen durchgeführt. Zunächst wurde die Frage untersucht, welche konkreten Verbesserungen in der Datenqualität und -konsistenz durch die Neustrukturierung der Tabellen, die Festlegung von Namenskonventionen und die Implementierung von Primärschlüsseln erreicht werden können. Durch die gezielte Aufteilung der Tabellen und eine verbesserte inhaltliche Trennung der Daten konnte die Übersichtlichkeit der Daten erheblich gesteigert werden, insbesondere für die Benutzer. Laut Nutzerfeedback gaben alle befragten Testbenutzer an, dass die Übersichtlichkeit der Daten deutlich verbessert wurde, was die Effizienz in der Nutzung signifikant erhöhte. Darüber hinaus führte der korrekte Einsatz von Primärschlüsseln dazu, dass Doppelungen in den Stammdaten vollständig eliminiert wurden. Vor der Implementierung dieser Maßnahme wiesen ungefähr 30 Prozent der Datensätze Inkonsistenzen bezüglich der Primärschlüssel und doppelte Einträge auf, wie eine Stichprobe von 300 Dateneinträgen zeigte, von denen 98 Tupel Fehler enthielten. Nach der Einführung der Primärschlüssel konnten diese Inkonsistenzen vollständig behoben werden. Auch die Festlegung von Namenskonventionen trug wesentlich zur Verbesserung der Datenkonsistenz bei. Laut Nutzerfeedback half die Standardisierung der Namensgebung dabei, Inkonsistenzen innerhalb der Tabellen und Abfragen zu vermeiden. Vor der Implementierung der Namenskonventionen existierten Spalten mit denselben Inhalten und unterschiedlichen Bezeichnungen innerhalb der Datenbanken. Durch die Integration einheitlicher Namenskonventionen für Attribute, Tabellen und Abfragen wurde ein klar definierter Leitfaden geschaffen, der Unstimmigkeiten nahezu vollständig eliminierte. Es muss jedoch darauf hingewiesen werden, dass die Einhaltung der Namenskonventionen nicht automatisch überprüft wird und daher von der korrekten Umsetzung durch die einzelnen Mitarbeitenden abhängt.

In einem weiteren Schritt wurde analysiert, inwieweit die bestehende Infrastruktur, insbesondere Microsoft Access, durch die Integration von Python-Code und einer grafischen Benutzeroberfläche effizient erweitert und optimiert werden kann. Die Einführung einer grafischen Benutzeroberfläche trug erheblich zur Verbesserung der Dateneingabeprozesse und der allgemeinen Benutzerfreundlichkeit bei. Sie ermöglichte es, klare Prozesse zu definieren, wie und in welcher Reihenfolge Daten eingepflegt werden sollten. Ein schrittweiser Aufbau, bei dem zunächst die Datenquelle ausgewählt und anschließend die entsprechende Tabelle spezifiziert wird, führte zu einer grundlegenden Restrukturierung der internen Abläufe. Diese neue Struktur ermöglichte es, Dateien nach vorher nichtexistierenden Ordnerstrukturen und nach neu definierten Schemata abzulegen. Zuvor wurden Daten manuell und oft fehleranfällig direkt aus E-Mail-Texten in die Datenbank eingepflegt. Dieser manuelle Prozess führte zu einem hohen Fehleranteil. Wie bereits erwähnt, waren fast 30 Prozent der Datensätze aufgrund dieser Vorgehensweise fehlerhaft. Die manuelle Dateneingabe war zudem äußerst zeitintensiv, da das Einpflegen eines Tupels, einschließlich der notwendigen Überprüfungen der einzelnen Attribute, laut Aussagen der Mitarbeitenden mehrere Minuten in Anspruch nahm. Mit zunehmender Datenmenge verschlechterte sich die Situation weiter, da die Fehlerquote stieg und die Konzentration der Mitarbeitenden nachließ. Durch die Implementierung der Python-Applikation konnte die Zeit für die Einpflegung eines Tupels in der Produktivumgebung bei unterschiedlichen Anwendern jedoch auf durchschnittlich 10 Sekunden reduziert werden. Ein weiterer erheblicher Vorteil besteht darin, dass die Dauer für die Eintragung auch bei mehreren Hundert

Tupeln nur minimal ansteigt. Nach Benutzertests dauert das Eintragen von 100 Tupeln beispielsweise nur etwa 11 Sekunden. Diese Effizienzsteigerung wächst somit proportional zur Menge der zu verarbeitenden Daten. Darüber hinaus wurde die Datenkonsistenz weiter verbessert. Dies gelang, indem durch die Anpassungen der Primärschlüssel innerhalb der Datenbank Doppelungen der Stammdaten vollständig vermieden wurden. Die Python-Applikation verfügt über integrierte Prüfmechanismen, die bereits bei der Einpflegung überprüfen, ob alle erforderlichen Spalten vorhanden sind und ob Doppelungen in anderen Tabellen bestehen. Diese Mechanismen tragen entscheidend dazu bei, die Datenkonsistenz in der gesamten Datenbank zu steigern. Es ist jedoch zu beachten, dass die beschriebenen Effizienzsteigerungen stark von der Verfügbarkeit der Daten in Excel-Dateien abhängen. Während die Daten der Outgoings über Formulare vorliegen und somit direkt verarbeitet werden können, müssen die Daten anderer Akteure derzeit noch manuell in Excel-Dateien übertragen werden. Um die volle Effizienz der Applikation auszuschöpfen, sollte daher als logischer nächster Schritt die Datensammlung auch für diese Akteure über Formulare erfolgen, ähnlich wie es bereits bei den Outgoings der Fall ist.

Eine weitere zentrale Forschungsfrage bezog sich auf die Optimierung der Systemleistung durch die Aufteilung der Datenbank in Backend- und Frontend-Komponenten. Stichproben- und Mehrbenutzertests in der Produktivumgebung zeigen, dass diese Aufteilung nicht zu einer signifikanten Steigerung der Systemperformance oder Abfragegeschwindigkeit führt. Dies liegt vor allem daran, dass die Datenmenge im Vergleich zu industriellen Datenbanken gering ist und die Anzahl der Mitarbeitenden, die gleichzeitig auf die Datenbanken zugreifen, ebenfalls begrenzt ist. Dennoch entstehen durch die Auftrennung des Datenbanksystems in Frontend- und Backend-Komponenten andere bedeutende Vorteile. Insbesondere ermöglicht diese Struktur die Integration eines Zugriffskonzeptes, da Abfragen und Daten getrennt voneinander verwaltet werden. Dies trägt maßgeblich zur Verbesserung der Datensicherheit und zum Schutz sensibler Informationen bei. Zudem führte die Trennung laut Nutzerfeedback zu einer Optimierung der internen Prozesse, da Aufgaben innerhalb des Projekts klarer definiert und gezielter auf die Mitarbeitenden verteilt werden können. Diese verbesserte Aufgabenverteilung und die klarere Strukturierung der Prozesse werden von den Nutzern positiv hervorgehoben und tragen zu einer effizienteren Arbeitsweise im Umgang mit der Datenbank bei.

Ein besonderer Fokus lag zudem auf der Automatisierung von Datenimporten, -exporten und der Dokumentenerstellung, um den manuellen Aufwand zu reduzieren und die Genauigkeit administrativer Prozesse zu steigern. Die Implementierung von Python-Skripten führte zu einer deutlichen Zeitersparnis und einer Reduktion menschlicher Fehler, was insbesondere in einem internationalen Kontext von großer Bedeutung ist. Durch die Automatisierung konnte der manuelle Aufwand auf das Betätigen einzelner Knöpfe reduziert werden, sodass alle nachfolgenden Schritte nun automatisiert ablaufen. Beispielsweise werden Exportdateien für Studierende innerhalb von nur 7 Sekunden erstellt, wobei dieser Wert alle erforderlichen Schritte, wie das Betätigen der Knöpfe, einschließt. Selbst bei Tests mit 100 Dateien lag die benötigte Zeit in der Entwicklungsumgebung bei durchschnittlich 17 Sekunden. Vor der Automatisierung nahmen diese Aufgaben nach Einschätzung der Mitarbeitenden durchschnittlich drei bis vier Minuten in Anspruch, da die Datenbank und die Datei gleichzeitig geöffnet werden mussten, was häufig zu Problemen führte, wie dem versehentlichen Verrutschen von Zeilen oder anderen manuellen Fehlern. Ein weiterer bedeutender Vorteil dieser Automatisierung ist die erhebliche Verringerung des Risikos von Datenmanipulationen. Die Mitarbeitenden können nun die erforderlichen Dateien erstellen, ohne die Datenbank selbst öffnen zu müssen. Dies reduziert die Notwendigkeit, Daten manuell zu kopieren, was laut den Mitarbeitenden oft zu unbeabsichtigtem Löschen von Daten oder zur Manipulation von Einträgen führte. Zudem gibt es positive Rückkopplungseffekte, die die Datenqualität weiter verbessern. In den automatisch erstellten Dateien werden fehlende Dateneinträge durch leere Formfelder sofort sichtbar, sodass die Mitarbeitenden diese Lücken schneller erkennen können. Dies zwingt die Nutzer, die fehlenden Daten in die Datenbank einzupflegen und die Dateien anschließend erneut zu erstellen, wodurch die Datenbank kontinuierlich aktualisiert und vervollständigt wird. Zuvor war es möglich, die Dateien manuell zu befüllen, wodurch Lücken in der Datenbank weiterhin bestehen

bleiben konnten. Diese Automatisierung sorgt somit nicht nur für eine Effizienzsteigerung, sondern trägt auch zur kontinuierlichen Verbesserung der Datenqualität bei.

Schließlich wurde untersucht, wie die Implementierung datengetriebener Prozesse und Automatisierungen zur Verringerung der Abhängigkeit von finanziellen Mitteln, Zeitressourcen und Personal beitragen kann, um die Wettbewerbsfähigkeit der Hochschule zu verbessern. Die Ergebnisse zeigen, dass durch die Automatisierung und Optimierung der Prozesse erhebliche Kosteneinsparungen erzielt werden können. Der Einsatz von Python und die kostengünstige Umsetzung einer datengetriebenen Applikation mit einer benutzerfreundlichen Oberfläche führten zu einer signifikanten Steigerung der Effizienz im gesamten Projekt, ohne die Notwendigkeit einer umfassenden und kostspieligen Neuentwicklung eines Datenbanksystems. Diese Lösung ermöglicht es den Mitarbeitenden, weniger Zeit für Verwaltungstätigkeiten aufzuwenden und ihren Fokus stattdessen auf die Weiterentwicklung und Ausweitung des Projekts zu legen. Dadurch sind sie in der Lage, eine größere Anzahl an Studierenden zu betreuen, wodurch das Projekt wachsen kann. Die eingesparte Zeit führt nicht nur zu einer Reduktion der benötigten Arbeitsstunden im Umgang mit den Daten und bei der Erstellung der Dokumente, sondern auch zu direkten Kosteneinsparungen auf Basis der Arbeitsstunden. Um die Wettbewerbsfähigkeit der Hochschule langfristig zu gewährleisten, ist es jedoch entscheidend, dass das Grundsystem der Automatisierung kontinuierlich weiterentwickelt und durch zusätzliche Module ergänzt wird. Zudem müssen die derzeit definierten Strukturen konsequent beibehalten und strikt umgesetzt werden, um die erreichte Effizienz und Datenintegrität dauerhaft zu sichern. Zusammenfassend lässt sich festhalten, dass die in dieser Arbeit entwickelten und implementierten Maßnahmen zu einer signifikanten Verbesserung der Effizienz, Benutzerfreundlichkeit und Sicherheit in der Verwaltung von Bildungsdaten geführt haben. Die spezifischen Forschungsfragen wurden allesamt positiv beantwortet, was die zentrale Fragestellung dieser Arbeit klar bestätigt. Die Einführung von Automatisierungen, die Aufteilung in Backend- und Frontend-Komponenten sowie die Implementierung einer kostengünstigen, datengetriebenen Applikation haben zu einer deutlichen Optimierung der Prozesse geführt, wodurch nicht nur die Verwaltung effizienter, sondern auch die Datenqualität erheblich gesteigert wurde. Diese Erkenntnisse bieten wertvolle praktische Implikationen für die Optimierung von Datenbanksystemen im Bildungsbereich und tragen gleichzeitig zur theoretischen Diskussion über den Einsatz moderner Technologien zur Effizienzsteigerung bei. Um die langfristige Wettbewerbsfähigkeit der Hochschule zu sichern, ist es jedoch entscheidend, dass das entwickelte Automatisierungssystem kontinuierlich weiter verfeinert und durch neue Technologien ergänzt wird. Ebenso wichtig ist es, die definierten Strukturen konsequent beizubehalten und strikt umzusetzen, um die erreichten Verbesserungen dauerhaft zu sichern. Zukünftige Forschungen könnten sich darauf konzentrieren, diese Automatisierungen weiter zu optimieren und neue, innovative Ansätze zu integrieren, um den sich stetig wandelnden Anforderungen an Datenbanksysteme gerecht zu werden. Ein möglicher Ansatz für zukünftige Erweiterungen könnte die automatische Datensichtung auf Basis definierter Schwellenwerte sein. So können Bewerbungen von Studierenden automatisch überprüft und entsprechend der mitgelieferten Daten eine sofortige Bestätigung oder Ablehnung generiert werden. Dies würde den Prozess der Bewerbungsbewertung weiter beschleunigen und den administrativen Aufwand reduzieren.

6 Zusammenfassung und Ausblick

Innerhalb dieser Arbeit wurde die Entwicklung und Implementierung eines automatisierten Datenbanksystems zur Unterstützung des internationalen Bildungsaustauschs untersucht, mit dem Ziel, die Effizienz der Datenverwaltung zu maximieren und gleichzeitig die Sicherheit sensibler Bildungsdaten zu gewährleisten. Akademische Einrichtungen stehen bei der Verwaltung ihrer Datenbanken vor zahlreichen Herausforderungen. Begrenzte Ressourcen, einschließlich eines unzureichenden Budgets und Personals, stellen ein erhebliches Hindernis für die Entwicklung und den Einsatz von institutionellen Verwaltungssystemen dar (Joo et al., 2019). Während renommierte Universitäten von einem besseren Zugang zu Technologien profitieren, ist die Implementierung dieser Systeme dennoch oft zeitaufwendig (Kustitskaya et al., 2023). Zudem sind Probleme mit der Datenqualität weit verbreitet, was die Aufrechterhaltung genauer und aktueller Informationen zu einer zentralen Herausforderung macht (Astuti et al., 2024). Angesichts der zunehmenden Datenmengen und der Sensibilität bestimmter Informationen werden diese Herausforderungen noch verstärkt (Joo et al., 2019). Vor diesem Hintergrund wird in dieser Arbeit untersucht, ob und in welchem Maße die grundlegende Neustrukturierung einer bestehenden Datenbank, kombiniert mit der Implementierung einer Software-Applikation zur datengetriebenen Automatisierung von Hochschulprozessen, zu einer signifikanten Effizienzsteigerung und Stabilisierung der Projektprozesse beitragen kann. Um diese zentrale Forschungsfrage zu beantworten, wurden mehrere spezifische Unterforschungsfragen definiert, die die Grundlage für die systematische Vorgehensweise der Untersuchung bilden. (siehe Abschnitt 1) Die methodischen Grundlagen zur Entwicklung und Implementierung des automatisierten Datenbanksystems wurden detailliert beschrieben, wobei der Fokus auf dem relationalen Datenmodell und der Nutzung von SQL als zentralem Werkzeug zur Datenmanipulation und -abfrage lag. Die Integration von Python zur Entwicklung automatisierter Prozesse, wie etwa dem Datenimport und -export und die Erstellung einer GUI mithilfe von Tkinter werden ebenfalls umfassend behandelt (siehe Abschnitt 2).

Ein wesentlicher Aspekt der Arbeit war die Restrukturierung der Datentabellen, die zur Verbesserung der Übersichtlichkeit, der Datenqualität und der Datenkonsistenz führte. Konkret wurde die bestehende Datenbank in eine Frontend- und eine Backend-Komponente aufgeteilt, was eine klare Trennung von Daten und Benutzeroberfläche ermöglichte. Zudem wurden die Tabellen in spezifischere Untertabellen unterteilt, wobei klare Primär- und Fremdschlüssel definiert wurden, um die Datenintegrität weiter zu erhöhen. Im Zuge dieser Maßnahmen wurden alle bestehenden Abfragen überarbeitet und zahlreiche neue Abfragen hinzugefügt, um die Funktionalität und Flexibilität des Systems zu erweitern. Um Automatisierungspotenziale zu identifizieren, wurden zudem Prozessdiagramme erstellt, die die bestehenden Arbeitsabläufe analysierten und als Grundlage für die Entwicklung entsprechender Automatisierungen dienten, die anschließend in Abschnitt 4 umgesetzt wurden. (siehe Abschnitt 3) Danach wurde die Entwicklung der Datenapplikation innerhalb von Python detailliert beschrieben. Zunächst wurde eine GUI mithilfe der Tkinter-Bibliothek aufgesetzt, deren Funktionsweise unter Hinzunahme verschiedener Abbildungen ausführlich erläutert wurde. Diese Benutzeroberfläche ermöglicht den Nutzern eine intuitive Interaktion mit der Datenbank und stellt sicher, dass die Prozesse klar und effizient ausgeführt werden können. Anschließend wurden die automatisierte Dateneinpflegung sowie der Datenimport programmiert und umfassend erklärt. Diese Automatisierungen tragen erheblich zur Effizienzsteigerung bei, indem sie manuelle Eingaben minimieren und Fehler reduzieren. Im letzten Schritt wurden die Datenexporte und die damit verbundene Dokumentenerstellung implementiert, einschließlich des automatisierten Ausfüllens von PDF-Vorlagen für drei spezifische Dokumente, die im Rahmen der Projektprozesse benötigt werden. Diese Automatisierungen erleichtern den Arbeitsablauf und gewährleisten eine konsistente und fehlerfreie Dokumentenerstellung. (siehe Abschnitt 4)

Die erfolgreiche Implementierung und Validierung des entwickelten Systems in der bestehenden Systemlandschaft wird detailliert erläutert, wobei ein besonderes Augenmerk auf das Zugriffskonzept gelegt wird, das unbefugte Zugriffe verhindert und gleichzeitig autorisierten Benutzern den notwendigen Zugang gewährt. In diesem Kontext wird auch die Integration der Automatisierungslösungen beschrieben, die die Effizienz und Sicherheit der Datenverwaltung

erheblich verbessert haben. In der abschließenden Diskussion werden die in der Arbeit formulierten Forschungsfragen und Ziele erneut aufgegriffen und anhand der Ergebnisse der vorherigen Abschnitte diskutiert und bewertet. Dabei wird deutlich, wie die entwickelten Lösungen zur Erreichung der gesetzten Ziele beigetragen haben. Zusätzlich wird ein Ausblick auf zukünftige Entwicklungen gegeben, der aufzeigt, dass durch Erweiterungen wie beispielsweise die automatische Sichtung von Bewerbungen auf Basis definierter Schwellenwerte, bei denen Studierende automatisch eine Bestätigung oder Ablehnung erhalten, die Automatisierung noch weiter optimiert werden kann. (siehe Abschnitt 5)
Diese Arbeit leistet einen wesentlichen Beitrag zur Optimierung der Datenverwaltung im internationalen Bildungsaustausch und bietet gleichzeitig ein Modell, das auf andere Anwendungsbereiche und Projekte innerhalb von Hochschulen übertragen werden kann. Die Ergebnisse unterstreichen die Bedeutung einer durchdachten Datenbankentwicklung und die Notwendigkeit, moderne Programmiersprachen wie Python in die Entwicklung solcher Systeme zu integrieren. Dieser Ansatz ist besonders wichtig, um als Hochschule wettbewerbsfähig zu bleiben und die vorhandenen Ressourcen sowie das begrenzte Budget effizient einzusetzen. Durch die Automatisierung von Prozessen wird nicht nur der manuelle Aufwand reduziert, sondern auch die Fehleranfälligkeit durch Mitarbeitende minimiert, was langfristig zu einer stabileren und effektiveren Datenverwaltung führt.

Literaturverzeichnis

Agarwal, S.; Keller, A. M.; Wiederhold, G.; Saraswat, K. (1995): Flexible relation: an approach for integrating data from multiple, possibly inconsistent databases. In: Proceedings of the Eleventh International Conference on Data Engineering. Eleventh International Conference on Data Engineering. Taipei, Taiwan, 6-10 March 1995: IEEE Comput. Soc. Press, S. 495–504.

Ahad, Rafiul; Bapa, K. V.; McLeod, Dennis (1989): On estimating the cardinality of the projection of a database relation. In: *ACM Trans. Database Syst.* 14 (1), S. 28–40. DOI: 10.1145/62032.62034.

Ahmad, Baraani-Dastjerdi (1996): Access control object-oriented databases. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:63353092.

Aljarallah, Nasser Ali; Dutta, Ashit Kumar (2022): Developing a Quality Automation Framework to Assess Specifications for Academic Accreditation in Saudi Arabian Universities. In: *TEM Journal*, S. 667–674. DOI: 10.18421/tem112-21.

Astuti, Hanim Maria; Wibowo, Radityo Prasetianto; Herdiyanti, Anisah (2024): Towards the National Higher Education Database in Indonesia: Challenges to Data Governance Implementation from The Perspective of a Public University. In: *Procedia Computer Science* 234, S. 1322–1331. DOI: 10.1016/j.procs.2024.03.130.

Bachman, Charles W.; R. Batchelor; I. Marvin Beriss; Charles R. Blose; Turgut I. Burakreis; Vincent Delia Valle et al. (1969): Data base task group report to the CODASYL programming language committee. In: *Data Base* 2, S. 11–18. Online verfügbar unter https://api.semanticscholar.org/CorpusID:20969082.

Badia, Antonio (2020): SQL FOR DATA SCIENCE. Data cleaning, wrangling and analytics with relational databases. [S.l.]: Springer.

Basak, Abhinav; Roy, Shatarupa T. (2019): Reassessing User-Friendliness of Evolving Graphical Interface Design from Social Perspective. In: Amaresh Chakrabarti (Hg.): Research into Design for a Connected World, Bd. 135. Singapore: Springer Singapore (Smart Innovation, Systems and Technologies), S. 327–341.

Bauer, Thomas (2017): Vormodellierte Flexibilität in Prozess-Management-Systemen:Anforderungen, Vorgehensweisen, Lösungsansätze. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:64561204.

Benedikt, Michael; Senellart, Pierre (2012): Introduction : Two Views of Database Research. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:13753917.

Bercich, Nancy Hartline (2002): The Evolution of the Computerized Database. In: *ArXiv* cs.DB/0305038. Online verfügbar unter https://api.semanticscholar.org/CorpusID:5385001.

Berg Hansen, Kim (2020): Pitfalls of Set Operations. In: Kim Berg Hansen (Hg.): Practical Oracle SQL. Berkeley, CA: APRESS, S. 17–38.

Bernick, Jonathan P. (2003): A translation of the one-to-one relationship for introductory relational database courses. In: *SIGCSE Bull.* 35 (4), S. 66–67. DOI: 10.1145/960492.960529.

Bertino, E.; Ferrari, E. (1998): Data security. In: Proceedings. The Twenty-Second Annual International Computer Software and Applications Conference (Compsac '98) (Cat. No.98CB 36241). Proceedings. The Twenty-Second Annual International Computer Software and Applications Conference (Compsac '98) (Cat. No.98CB 36241). Vienna, Austria, 19-21 Aug. 1998: IEEE Comput. Soc, S. 228–237.

Biswas, Debmalya; Jiwane, Ashwin; Genest, Blaise (2009): Atomicity for XML Databases. In: Zohra Bellahsène, Ela Hunt, Michael Rys und Rainer Unland (Hg.): Database and XML Technologies, Bd. 5679. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), S. 180–187.

Blaser, Albrecht (1995): Die BTW im Wandel der Datenbank-Zeiten. In: W. Brauer und Georg Lausen (Hg.): Datenbanksysteme in Büro, Technik und Wissenschaft. Berlin, Heidelberg: Springer Berlin Heidelberg (Informatik aktuell), S. 48–50.

Blaustein, B. T.; Jajodia, S.; McCollum, C. D.; Notargiacomo, L. (1993): A model of atomicity for multilevel transactions. In: Proceedings 1993 IEEE Computer Society Symposium on Research in Security and Privacy. 1993 IEEE Computer Society Symposium on Research in Security and Privacy. Oakland, CA, USA, 24-26 May 1993: IEEE Comput. Soc. Press, S. 120–134.

Bogdanchikov, A.; Zhaparov, M.; Suliyev, R. (2013): Python to learn programming. In: *J. Phys.: Conf. Ser.* 423, S. 12027. DOI: 10.1088/1742-6596/423/1/012027.

Böhm, Alexander; Golombek, Mathias; Heinz, Christoph; Löser, Henrik; Schlaucher, Alfred; Ruf, Thomas (2015): Panel: ,,Big Data - Evolution oder Revolution in der Datenverarbeitung?“. In: Datenbanksysteme für Business, Technologie und Web (BTW 2015). Bonn: Gesellschaft für Informatik e.V, S. 647–648.

Boonlit, Adipat; Dongsong, Zhang (2005): Interface Design for Mobile Applications. In: Americas Conference on Information Systems. Online verfügbar unter https://api.semanticscholar.org/CorpusID:38097528.

Botha, Reinhardt A. (2002): Towards Semantic Integrity in Relational Databases. In: M. Adeeb Ghonaimy, Mahmoud T. El-Hadidi und Heba K. Aslan (Hg.): Security in the Information Society, Bd. 86. Boston, MA: Springer US (IFIP Advances in Information and Communication Technology), S. 287–297.

Brock, Bert de (2023): Developing Information Systems Accurately. A Wholistic Approach. 1st ed. 2023. Cham: Springer International Publishing; Imprint: Springer.

Bronselaer, Antoon; van Britsom, Daan; Tre, Guy de (2015): Propagation of Data Fusion. In: *IEEE Trans. Knowl. Data Eng.* 27 (5), S. 1330–1342. DOI: 10.1109/TKDE.2014.2365807.

Brunel, Robert; Finis, Jan (2013): Eine effiziente Indexstruktur für dynamische hierarchische Daten. In: Gunter Saake (Hg.): Datenbanksysteme für Business, Technologie und Web (BTW) 2013 - Workshopband. Tagung vom 11. - 12. März 2013 in Magdeburg. Bonn: Ges. für Informatik (GI-Edition : Proceedings, 216), S. 267–276. Online verfügbar unter https://subs.emis.de/LNI/Proceedings/Proceedings216/P-216.pdf.

Bühler, Peter (2019): Datenmanagement. Daten - Datenbanken - Datensicherheit. Berlin: Springer Vieweg (Bibliothek der Mediengestaltung).

Butnaru, Daniel (2006): Concurrency in XML. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:57298043.

Candace Kamm (1994): User Interfaces for voice applications. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:17056314.

Chandrasekaran, G.; Neethidevan, V.; Murugachandravel, J. (2019): Impact of Unit Testing in Web Automation Testing. In: *IJRTE* 8 (3), S. 1011–1013. DOI: 10.35940/ijrte.c4064.098319.

Charatan, Quentin; Kans, Aaron (2022): Programming in two semesters. Using Python and Java. Cham: Springer (Texts in computer science).

Collins, Jr.David (2021): Accessing Microsoft Access Databases Using ODBC and RODBC.

Csongor Nyulas; Martin J. O'Connor; Samson W. Tu (2007): DataMaster – a Plug-in for Importing Schemas and Data from Relational Databases into Protégé. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:53827388.

Dadam, Peter (1986): Forschung und Entwicklung im Datenbank-Management-System-Bereich. In: W. Brauer, G. Hommel und S. Schindler (Hg.): Informatik-Anwendungen — Trends und Perspektiven, Bd. 126. Berlin, Heidelberg: Springer Berlin Heidelberg (Informatik-Fachberichte), S. 63–89.

David Sale (2014): Testing Python: Applying Unit Testing, TDD, BDD and Acceptance Testing. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:62462563.

Deen, S. M. (Hg.) (1977): Fundamentals of Data Base Systems. London: Macmillan Education UK.

Deppisch, U.; Obermeit, V.; Paul, H.-B.; Schek, H.-J.; Scholl, M.; Weikum, G. (1985): Ein Subsystem zur Stabilen Speicherung Versionenbehafteter, Hierarchisch Strukturierter Tupel. In: W. Brauer, A. Blaser und P. Pistor (Hg.): Datenbank-Systeme für Büro, Technik und Wissenschaft, Bd. 94. Berlin, Heidelberg: Springer Berlin Heidelberg (Informatik-Fachberichte), S. 421–440.

Dippold, Rolf; Meier, Andreas; Schnider, Walter; Schwinn, Klaus (2005): Aufbau der Datenbankadministration. In: Rolf Dippold, Andreas Meier, Walter Schnider und Klaus Schwinn (Hg.): Unternehmensweites Datenmanagement. Wiesbaden: Vieweg+Teubner Verlag, S. 49–68.

Dorendorf, Stefan; Küspert, Klaus (2000): Datenbank-Reorganisation bei relationalen Datenbank-Management-Systemen / Reorganization of Relational Databases. In: *it - Information Technology* 42 (3), S. 18–25. DOI: 10.1524/itit.2000.42.3.18.

Driscoll, James R. (1978): An implementation technique for data base management systems. In: Unknown (Hg.): Proceedings of the 16th annual Southeast regional conference on - ACM-SE 16. the 16th annual Southeast regional conference. Atlanta, Georgia, 13.04.1978 - 15.04.1978. New York, New York, USA: ACM Press, S. 16–21.

Dubey, Anita; Singhal, Anil Kumar (2016): Role of Organisational Structure in Empolyee's Empowerment. In: *International Journal of Education and Management Studies* 6, S. 110. Online verfügbar unter https://api.semanticscholar.org/CorpusID:148769219.

Dunn, J.; Davey, S.; Descour, A.; Snodgrass, R. T. (2002): Sequenced subset operators: definition and implementation. In: Proceedings 18th International Conference on Data Engineering. 18th International Conference on Data Engineering. San Jose, CA, USA, 26 Feb.-1 March 2002: IEEE Comput. Soc, S. 81–92.

Eckstein, Jonathan; Schultz, Bonnie R. (Hg.) (2017): Introductory Relational Database Design for Business, with Microsoft Access: Wiley.

Ehrich, H.-D. (1973): Datenstrukturen und Q-Systeme — eine mathematische Studie. In: G. Goos, J. Hartmanis, P. B. Hansen, G. Seegmüller, N. Wirth und Wilfried Brauer (Hg.): GI Gesellschaft für Informatik e. V. 3. Jahrestagung Hamburg, 8.–10. Oktober 1973, Bd. 1. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), S. 363–371.

Elmagarmid, Ahmed K. (1991): Transaction Models for Advanced Database Applications. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:60874504.

Elmasri, Ramez; Navathe, Sham (2016): Fundamentals of database systems. 7. ed. Hoboken: Pearson.

Elumalai, Aarthi (2021): Project: Tic-tac-toe Game with Tkinter. In: Aarthi Elumalai (Hg.): Introduction to Python for Kids. Berkeley, CA: APRESS, S. 369–390.

Engles, Robert W. (1972): A tutorial on data-base organization. In: *Annual Review in Automatic Programming* 7, S. 1–64. DOI: 10.1016/0066-4138(72)90003-1.

Fong, Elizabeth; Kimbleton, Stephen R. (1980): Database semantic integrity for a network data manager. In: Unknown (Hg.): Proceedings of the May 19-22, 1980, national computer conference on - AFIPS '80. the May 19-22, 1980, national computer conference. Anaheim, California, 19.05.1980 - 22.05.1980. New York, New York, USA: ACM Press, S. 261.

Frank, Lars (2010): Design of Distributed Integrated Heterogeneous or Mobile Databases: System Integration by using relaxed ACID properties. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:63160074.

Fromm, Niels (2009): Backup-Strategie für den Dokumentenserver. Unter Mitarbeit von Humboldt-Universität zu Berlin und Peter Schirmbacher.

Guynes, Carl S. (1987): Monitoring database performance—a control issue. In: *SIGSAC Rev.* 5 (2), S. 7–11. DOI: 10.1145/27944.27946.

Hammer, Michael M.; McLeod, Dennis J. (1975): Semantic integrity in a relational data base system. In: R. Stockton Gaines und David K. Hsiao (Hg.): Proceedings of the 1st International Conference on Very Large Data Bases - VLDB '75. the 1st International Conference. Framingham, Massachusetts, 22.09.1975 - 24.09.1975. New York, New York, USA: ACM Press, S. 25.

Harpreet Kaur; Jaspreet Kaur; Kamaljit Kaur (2013): A Review Of Non Relational Databases, Their Types, Advantages And Disadvantages. In: *International journal of engineering research and technology* 2. Online verfügbar unter https://api.semanticscholar.org/CorpusID:60701926.

Hellerstein, Joseph M.; Michael Stonebraker; Rick Caccia (1999): Independent, Open Enterprise Data Integration. In: *IEEE Data Eng. Bull.* 22, S. 43–49. Online verfügbar unter https://api.semanticscholar.org/CorpusID:6664269.

Herman, Susan K.; Aburdene, Maurice F. (1991): Design of a Graphical User Interface for Laboratory Instruments. In: *Journal of Engineering Design* 2 (3), S. 197–218. DOI: 10.1080/09544829108901681.

Hunt, John (2019): Introduction. In: John Hunt (Hg.): Advanced Guide to Python 3 Programming. Cham: Springer International Publishing (Undergraduate Topics in Computer Science), S. 1–2.

Hunt, John (2023): Advanced Guide to Python 3 Programming. 2nd edition 2023. Cham: Springer Nature Switzerland (Undergraduate Topics in Computer Science).

Ivongbe, Matthew Ihaza; Lucky T.*, Abdulsalami; Harry O., Omorogbe (2021): Application of Database Management System in the Library of Federal University, Lafia. In: *NIJBMR* (52), S. 39–45. DOI: 10.51550/NIJBMR.52.39.45.

Joo, Soohyung; Hofman, Darra; Kim, Youngseek (2019): Investigation of challenges in academic institutional repositories. In: *LHT* 37 (3), S. 525–548. DOI: 10.1108/lht-12-2017-0266.

Kamra, Ashish; Terzi, Evimaria; Bertino, Elisa (2008): Detecting anomalous access patterns in relational databases. In: *The VLDB Journal* 17 (5), S. 1063–1077. DOI: 10.1007/s00778-007-0051-4.

Kao, Ben; Garcia-Molina, Hector (1994): An Overview of Real-Time Database Systems. In: Wolfgang A. Halang und Alexander D. Stoyenko (Hg.): Real Time Computing, Bd. 127. Berlin, Heidelberg: Springer Berlin Heidelberg (NATO ASI Series), S. 261–282.

Kaufmann, Michael; Meier, Andreas (2023): SQL- et NoSQL-Datenbanken. 9. erweiterte und aktualisierte Auflage. 9th ed. 2023. Berlin, Heidelberg: Springer Berlin Heidelberg; Springer Vieweg.

Kaur, Husandeep (2017): Review on Overview of Oracle Operators. In: *IJRASET* V (IV), S. 1288–1290. DOI: 10.22214/IJRASET.2017.4230.

Kellenberger, Kathi; Everest, Lee (2021): Manipulating Data. In: Kathi Kellenberger und Lee Everest (Hg.): Beginning T-SQL. Berkeley, CA: APRESS, S. 331–372.

Kline, Kevin E.; Kline, Daniel (2004): SQL in a nutshell. 2nd ed. Sebastopol, Calif: O'Reilly.

Knackstedt, Ralf; Dahlke, Beate (2002): Prozessmodellierung und Kundenintegration. In: Matthias Meyer (Hg.): CRM-Systeme mit EAI. Wiesbaden: Vieweg+Teubner Verlag, S. 89–115.

Kollmann, Franz (2007): Realizing fine-granular Read andWrite Rights on Tree Structured Documents. In: The Second International Conference on Availability, Reliability and Security (ARES'07). The Second International Conference on Availability, Reliability and Security (ARES'07). Vienna, Austria, 10.04.2007 - 13.04.2007: IEEE, S. 517–523.

Künzle, Vera; Reichert, Manfred (2009): Herausforderungen auf dem Weg zu datenorientierten Prozess-Management-Systemen. In: *EMISA Forum* 29, S. 9–24. Online verfügbar unter https://api.semanticscholar.org/CorpusID:44783819.

Kustitskaya, Tatiana A.; Esin, Roman V.; Kytmanov, Alexey A.; Zykova, Tatiana V. (2023): Designing an Education Database in a Higher Education Institution for the Data-Driven Management of the Educational Process. In: *Education Sciences* 13 (9), S. 947. DOI: 10.3390/educsci13090947.

Kvet, Michal; Matiasko, Karol (2019): Efficiency of the relational database tuple access. In: 2019 IEEE 15th International Scientific Conference on Informatics. 2019 IEEE 15th International Scientific Conference on Informatics. Poprad, Slovakia, 20.11.2019 - 22.11.2019: IEEE, S. 231–236.

Lam, Kam-yiu; Kuo, Tei-Wei (2013): Real-Time Database Systems: Architecture and Techniques. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:60930461.

Leonid Libkin; Liat Peterfreund (2020): Handling SQL Nulls with Two-Valued Logic. In: *ArXiv* abs/2012.13198. Online verfügbar unter https://api.semanticscholar.org/CorpusID:229371252.

Levene, Mark; Loizou, George (2001): A Generalisation of Entity and Referential Integrity in Relational Databases. In: *RAIRO-Theor. Inf. Appl.* 35 (2), S. 113–127. DOI: 10.1051/ita:2001111.

Lindström, Jan (2007): Real Time Database Systems. In: Benjamin W. Wah (Hg.): Wiley Encyclopedia of Computer Science and Engineering: Wiley, S. 1–13.

Mannila, Linda; Peltomäki, Mia; Salakoski, Tapio (2006): What about a simple language? Analyzing the difficulties in learning to program. In: *Computer Science Education* 16 (3), S. 211–227. DOI: 10.1080/08993400600912384.

Markowitz, Victor M. (1990): Referential Integrity Revisited: An Object-Oriented Perspective. In: Very Large Data Bases Conference. Online verfügbar unter https://api.semanticscholar.org/CorpusID:18517587.

McFadyen, Ron; Kanabar, Vijay (1991): An Introduction to Structured Query Language. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:61945064.

Meier, Andreas.; Kaufmann, Michael. (2019): SQL & NoSQL Databases. Models, Languages, Consistency Options and Architectures for Big Data Management. 1st ed. 2019. Wiesbaden: Springer Fachmedien Wiesbaden; Imprint; Springer Vieweg.

Melton, Jim (1996): SQL language summary. In: *ACM Comput. Surv.* 28 (1), S. 141–143. DOI: 10.1145/234313.234374.

Meyer, Robert; Helmich, Martin (2011): Praxiswissen TYPO3. TYPO3-Version 4.5 ; der praxisnahe TYPO3-Einstieg ; komplette Beispielanwendung auf CD ; mit Tipps aus dem Support. 5. Aufl. Beijing, Köln: O'Reilly (O'Reillys basics). Online verfügbar unter https://swbplus.bsz-bw.de/bsz338137904cov.htm.

Meyer-Wegener, Klaus (1988): Transaktionssysteme. Funktionsumfang, Realisierungsmöglichkeiten, Leistungsverhalten. Wiesbaden: Vieweg+Teubner Verlag (Leitfäden der angewandten Informatik).

Michael R. Blaha (2005): Referential Integrity Is Important For Databases. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:6831872.

Michael Smith; Analyst; A. Lee; Mech. (2021): Appendix A Overview of Relational DBMS. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:236961287.

Mielebacher, Jörg (2024): Datenbanken für Nichtinformatiker. Eine praxisnahe Einführung. Wiesbaden, Heidelberg: Springer Vieweg (Lehrbuch). Online verfügbar unter https://link.springer.com/978-3-658-42662-0.

Mitschang, Bernhard (1984): Überlegungen zur Architektur von Datenbanksystemen für Ingenieuranwendungen. In: W. Brauer und Hans-Dieter Ehrich (Hg.): GI — 14. Jahrestagung, Bd. 88. Berlin, Heidelberg: Springer Berlin Heidelberg (Informatik-Fachberichte), S. 318–334.

Mohamed, Aya; Auer, Dagmar; Hofer, Daniel; Küng, Josef (2022): Authorization and Access Control for Different Database Models: Requirements and Current State of the Art. In: Tran Khanh Dang, Josef Küng und Tai M. Chung (Hg.): Future Data and Security Engineering. Big Data, Security and Privacy, Smart City and Industry 4.0 Applications, Bd. 1688. Singapore: Springer Nature Singapore (Communications in Computer and Information Science), S. 225–239.

Moruzzi, Giovanni (2020): ESSENTIAL PYTHON FOR THE PHYSICIST. [Place of publication not identified]: SPRINGER NATURE.

Mullins, Craig (2007): Database administration. The complete guide to practices and procedures. 5th ed. Boston: Addison-Wesley.

Mutti, Simone; Bacis, Enrico; Paraboschi, Stefano (2015): SeSQLite: Security Enhanced SQLite. In: Proceedings of the 31st Annual Computer Security Applications Conference. ACSAC 2015: 2015 Annual Computer Security Applications Conference. Los Angeles CA USA, 07 12 2015 11 12 2015. New York, NY, USA: ACM, S. 411–420.

Nakkeeran, R.; Reddy, Ailuri Narmada; Lavanya, Kanaparthi; Sandeep, Sunka (2023): Automation of Seating Plan for Examinations using Round-Robin Policy. In: *IJRASET* 11 (11), S. 634–640. DOI: 10.22214/ijraset.2023.56579.

Nazerian, Fatemeh; Motameni, Homayun; Nematzadeh, Hossein (2019): Emergency role-based access control (E-RBAC) and analysis of model specifications with alloy. In: *Journal of Information Security and Applications* 45, S. 131–142. DOI: 10.1016/j.jisa.2019.01.008.

Nelli, Fabio (2015): Introduction to the Python's World. In: Fabio Nelli (Hg.): Python Data Analytics. Berkeley, CA: APRESS, S. 13–34.

Nielsen, J.; Tscherning, C. C.; Jansson, T. R. N.; Forsberg, R. (2012): Development and User Testing of a Python Interface to the GRAVSOFT Gravity Field Programs. In: Steve Kenyon, Maria Christina Pacino und Urs Marti (Hg.): Geodesy for Planet Earth, Bd. 136. Berlin, Heidelberg: Springer Berlin Heidelberg (International Association of Geodesy Symposia), S. 443–449.

Oberweis, Andreas; Paulzen, Oliver; J. Sexauer, Hagen (2001): Ein wissensbasiertes Vorgehensmodell zur Gestaltung von CRM-Systemen. In: GI Jahrestagung. Online verfügbar unter https://api.semanticscholar.org/CorpusID:38707790.

Ollenschläger, Malte; Küderle, Arne; Mehringer, Wolfgang; Seifer, Ann-Kristin; Winkler, Jürgen; Gaßner, Heiko et al. (2022): MaD GUI: An Open-Source Python Package for Annotation and Analysis of Time-Series Data. In: *Sensors (Basel, Switzerland)* 22 (15). DOI: 10.3390/s22155849.

Pahle, H.; Hübel, C. (1992): Ansätze einer adaptierbaren Datenorganisation in einer integrierten Entwurfsumgebung. In: W. Brauer, Frank-Lothar Krause, Helmut Jansen und Detlev Ruland (Hg.): CAD ’92. Berlin, Heidelberg: Springer Berlin Heidelberg (Informatik aktuell), S. 233–253.

Panse, Fabian (2009): Datenunvollständigkeit aufgrund der mangelnden Modellierungsmächtigkeit aktuell dominierender Datenmodelle. In: Grundlagen von Datenbanken. Online verfügbar unter https://api.semanticscholar.org/CorpusID:33736073.

Parker, Sharon K.; Johnson, Anya; Collins, Catherine; Nguyen, Helena (2013): Making the Most of Structural Support: Moderating Influence of Employees' Clarity and Negative Affect. In: *AMJ* 56 (3), S. 867–892. DOI: 10.5465/AMJ.2010.0927.

Pernul, Günther; Unland, Rainer (2003): Datenbanken im Unternehmen: DE GRUYTER.

Purcell, Joseph D. (2012): Unix File Permissions. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:60363306.

Ramamritham, Krithi (1996): Advances in Real-Time Database Systems Research Special Section on RTDBS of ACM SIGMOD Record. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:8266111.

Rastogi, Rajeev; Mehrotra, Sharad; Breitbart, Yuri; Korth, Henry F.; Silberschatz, Avi (1993): On correctness of non-serializable executions. In: Catriel Beeri (Hg.): Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems - PODS '93. the twelfth ACM SIGACT-SIGMOD-SIGART symposium. Washington, D.C., United States, 25.05.1993 - 28.05.1993. New York, New York, USA: ACM Press, S. 97–108.

Ray, Kisor; Ghosh, Santanu; Das, Dr.Mridul; Ray, Dr.Bhaswati (2015): Design and Implementation Approach for Error Free Clinical Data Repository for the Medical Practitioners. In: *IJCTT* 21 (2), S. 71–74. DOI: 10.14445/22312803/IJCTT-V21P113.

Remus, Horst (1976): Überlegungen zur Entwicklung von Datenbanksystemen. In: G. Goos, J. Hartmanis, P. Brinch Hansen, D. Gries, C. Moler, G. Seegmüller et al. (Hg.): Data Base Systems, Bd. 39. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), S. 1–20.

Rivero, Laura C.; Doorn, Jorge Horacio; Ferraggine, Viviana E. (2005): Encyclopedia of Database Technologies and Applications: IGI Global.

Sallam, Asmaa; Bertino, Elisa (2018): Detection of Temporal Data Ex-Filtration Threats to Relational Databases. In: 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC). 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC). Philadelphia, PA, 18.10.2018 - 20.10.2018: IEEE, S. 146–155.

Schicker, Edwin (2017): Datenbanken und SQL. Eine praxisorientierte Einführung mit Anwendungen in Oracle, SQL Server und MySQL. 5. Aufl. 2017. Wiesbaden: Springer Vieweg (Informatik & Praxis).

Setiyadi, Didik (2021): Database System Development Life Cycle (DSDLC) on System Libraries for Data Manipulation Language (DML) Using SQL Server 2008. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:239664040.

Shoval, Peretz (1993): On Non-Constrained, Constrained and Mandatory Many-to-Many Relationship Types. In: *Journal of Database Management* 4 (1), S. 3–15. DOI: 10.4018/JDM.1993010101.

Simon, Mark (2023): Levellin up with SQL. Advanced techniques for transforming data into insights. [S.l.]: APRESS.

Snuggs, Mary E.; Popek, Gerald J.; Peterson, Ronald J. (1974): Data base system objectives as design constraints. In: *SIGMIS Database* 6 (3), S. 11–20. DOI: 10.1145/1017558.1017561.

Starbuck, Craig (2023): The Fundamentals of People Analytics. With Applications in R. Cham: SPRINGER NATURE.

Staud, Josef L. (2005): Datenmodellierung und Datenbankentwurf. [New York]: Springer Berlin Heidelberg.

Steiner, René (2017): Grundkurs relationale Datenbanken. Einführung in die Praxis der Datenbankentwicklung für Ausbildung, Studium und IT-Beruf. 9., erweiterte und aktualisierte Auflage. Wiesbaden: Springer Vieweg (Lehrbuch).

Thanasis Stamos (2007): ThanCad: a 2dimensional CAD for engineers. In:. Online verfügbar unter https://api.semanticscholar.org/CorpusID:59035036.

Thuraisingham, Bhavani (2000): Database Management and the Internet: Developments and Challenges. In: Sanjiv Purba (Hg.): High-Performance Web Databases: Auerbach Publications, S. 549–554.

Tsolkas, Alexander (2017): Rollen und Berechtigungskonzepte. Identity- und Access-Management im Unternehmen. Wiesbaden: Springer Fachmedien Wiesbaden (Edition Ser). Online verfügbar unter https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=4900831.

Unterstein, Michael; Matthiessen, Günter (2012): Relationale Datenbanken und SQL in Theorie und Praxis. 5. Aufl. Berlin: Springer Vieweg (eXamen.press).

van den Bercken, Jochen; Bernhard Seeger; Peter Widmayer (1997): A Generic Approach to Bulk Loading Multidimensional Index Structures. In: Very Large Data Bases Conference. Online verfügbar unter https://api.semanticscholar.org/CorpusID:6337438.

van der Lans, R. F. (1986): Data security in a relational database environment. In: *Computers & Security* 5 (2), S. 128–134. DOI: 10.1016/0167-4048(86)90135-5.

van Evert, Frist K.; Spaans, Egbert J. A.; Krieger, Scott D.; Carlis, John V.; Baker, John M. (1999): A Database for Agroecological Research Data: II. A Relational Implementation. In: *Agronomy Journal* 91 (1), S. 62–71. DOI: 10.2134/AGRONJ1999.00021962009100010010X.

van Hee, Kees M.; Sidorova, Natalia; Voorhoeve, Marc; van derWerf, Jan Martijn (2009): Generation of Database Transactions with Petri Nets. In: *Fundamenta Informaticae* 93 (1-3), S. 171–184. DOI: 10.3233/FI-2009-0095.

Voitovych, Olesia; Kupershtein, Leonid; Lukichov, Vitalii; Mikityuk, Ivan (2018): Multilayer Access for Database Protection. In: 2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T). 2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T). Kharkiv, Ukraine, 09.10.2018 - 12.10.2018: IEEE, S. 474–478.

Vossen, Gottfried (1999): Relationale Datenbanken im Wandel der Zeiten. In: Patrick Horster (Hg.): Angewandte Mathematik, insbesondere Informatik. Wiesbaden: Vieweg+Teubner Verlag, S. 301–320.

Wang, Liqiang; Stoller, Scott D. (2006): Accurate and efficient runtime detection of atomicity errors in concurrent programs. In: Josep Torrellas und Siddhartha Chatterjee (Hg.): Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming. PPoPP06: ACM SIGPLAN 2006 Symposium on Principles and Practice of Parallel Programming 2006. New York New York USA, 29 03 2006 31 03 2006. New York, NY, USA: ACM, S. 137–146.

Zauner, Martin; Schrempf, Andreas (2009): Datenbanken. In: Martin Zauner und Andreas Schrempf (Hg.): Informatik in der Medizintechnik. Vienna: Springer Vienna, S. 135–153.