

**Sirajeddine Ben Zinab**

Entwicklung einer Anwendungsoberfläche für Datenbankmigration mit GuttenBase

Universität Bremen, März 2021

## **Selbstständigkeitserklärung**

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Bremen, den 28. Februar 2021

---

Sirajeddine Ben Zinab

## Zusammenfassung

Migration ist im wissenschaftlichen Bereich kein neues Thema. Es bieten sich viele Methoden und Frameworks zur Beschreibung, Analyse und Implementierung der Migration. Dies gilt auch für Datenbankverwaltungssysteme (DBMS).

In dieser Arbeit werden aktuelle Tools für Datenbank Migration vorgestellt. Dabei werden wichtige Eigenschaften der Open Source Bibliothek GuttenBase erläutert.

Außerdem befasst sich diese Arbeit hauptsächlich mit dem Entwurf, Implementierung und Evaluation eines Tools für Datenbank Migration zwischen verschiedenen Datenbanksystemen (DBMS) basierend auf GuttenBase. Um die Nutzung der GuttenBase Bibliothek für möglichst viele Nutzer zur Verfügung zu stellen, erfolgt die Umsetzung als ein IntelliJ (IDEA) Plugin.

Diese Bachelorarbeit wurde bei der Firma Akquinet AG in Bremen im Zeitraum von September 2020 bis Februar 2021 erstellt und stellt den Abschluss meines Bachelorstudiums an der Universität Bremen dar.

Diese Abschlussarbeit liegt in deutscher Sprache vor.



# Inhaltsverzeichnis

<b>Inhaltsverzeichnis</b>	<b>i</b>
<b>Abbildungsverzeichnis</b>	<b>iii</b>
<b>Tabellenverzeichnis</b>	<b>v</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Problemstellung und Motivation . . . . .	1
1.2 Zielsetzung . . . . .	1
1.3 Aufbau der Arbeit . . . . .	3
<b>2 Grundlagen</b>	<b>5</b>
2.1 Datenbankmanagementsysteme . . . . .	5
2.2 Datenbankmigration . . . . .	6
2.3 Stand der Dinge . . . . .	7
2.3.1 GutenBase . . . . .	7
2.3.2 IntelliJ . . . . .	7
<b>3 Anforderungsmanagement</b>	<b>11</b>
3.1 Anforderungsermittlung . . . . .	11
3.1.1 Funktionale Anforderungen . . . . .	11
3.1.2 Systemanforderungen . . . . .	11
3.2 Anforderungsanalyse . . . . .	11
3.3 Anforderungsbeschreibung . . . . .	11
<b>4 Konzeption und Implementierung</b>	<b>13</b>
4.1 Anwendungsfunktionalität . . . . .	13
4.2 Konzeptentwicklung . . . . .	13
4.2.1 Konzeptionelle Sicht . . . . .	13
4.2.1.1 GutenBase Plugin . . . . .	14
4.2.1.2 IntelliJ Plattform . . . . .	15
4.2.2 Modulsicht . . . . .	16
4.2.2.1 Modulsicht der Übersicht der Konfigurationsschritte . . . . .	16

4.2.2.2	Modulsicht der allgemeinen Übersicht . . . . .	18
4.2.2.3	Modulsicht der Konfigurationsübersicht . . . . .	19
4.2.2.4	Modulsicht der Ergebnisübersicht . . . . .	21
4.3	Technologieauswahl . . . . .	22
4.4	Umsetzungsform . . . . .	22
4.4.1	GuttenBase . . . . .	24
4.4.2	IntelliJ Platform . . . . .	26
4.5	PlugIn Implementierung . . . . .	28
4.5.1	Funktion: Liste der vorhandenen Migrationsoperationen . . . . .	28
4.5.2	Funktion: Hinzufügen einer neuen Migrationsoperation . . . . .	29
4.5.3	Funktion: Verbindungserstellung . . . . .	29
4.5.4	Funktion: Konfiguration . . . . .	30
4.5.5	Funktion: Migrationsdurchführung . . . . .	33
<b>5</b>	<b>Test und Evaluation</b>	<b>37</b>
5.1	Testplan . . . . .	37
5.1.1	Experten Interview . . . . .	37
5.1.2	Unit-Tests . . . . .	37
5.1.3	Integrationstest . . . . .	37
5.2	Evaluation . . . . .	37
5.3	Anpassungen . . . . .	37
<b>6</b>	<b>Fazit und Ausblick</b>	<b>39</b>
6.1	Fazit . . . . .	39
6.2	Ausblick . . . . .	39
	<b>Literatur</b>	<b>41</b>

# Abbildungsverzeichnis

4.1	Komponentendiagramm für die konzeptionelle Sicht . . . . .	14
4.2	Modulsicht der Übersicht der Konfigurationsschritte . . . . .	17
4.3	Modulsicht der allgemeinen Übersicht . . . . .	19
4.4	Modulsicht der Konfigurationsübersicht . . . . .	20
4.5	Modulsicht der Ergebnisübersicht . . . . .	21
4.6	Top IDE Index . . . . .	24
4.7	ConnectionInfo konfigurieren . . . . .	25
4.8	Connector Repository konfigurieren . . . . .	25
4.9	ColumnMapperHint hinzufügen . . . . .	26
4.10	Datenbank Migration durchführen . . . . .	26
4.11	Übersicht der Migrationsoperationen öffnen . . . . .	28
4.12	Übersicht der Migrationsoperationen . . . . .	29
4.13	Migrationsoperation Umbenennen erstellen . . . . .	30
4.14	Migrationsoperationen exportieren und importieren . . . . .	30
4.15	Allgemeine Übersicht der Datenbank Migration (generalView) . . . . .	31
4.16	Konfigurationsübersicht (overview) . . . . .	32
4.17	matches Methode der Migrationsoperation Umbenennen . . . . .	32
4.18	das Hinzufügen von der Migrationsoperation Umbenennen . . . . .	32
4.19	Ergebnisübersicht . . . . .	33
4.20	ConnectorHints hinzufügen . . . . .	34
4.21	Migration durchführen (MapperHelper Klasse) . . . . .	34
4.22	Fortschrittsübersicht . . . . .	35





# Tabellenverzeichnis

2.1	Database Migration Tools . . . . .	9
4.1	Umsetzungsmöglichkeiten . . . . .	23



# Einleitung

## 1.1 Problemstellung und Motivation

IT-Migration ist seit Anbeginn des Informationszeitalter ein wichtiger Bestandteil der Informationsverarbeitung [WZ15]. Wie die Hardware, Betriebssysteme und Programme, werden Datenbanken auch häufig migriert. Der Grund dafür könnte z. B. eine Änderung der Unternehmensrichtlinien sein.

Bei einer Datenbank Migration werden Daten von einer Quell-Datenbank zu einer Ziel-Datenbank verschoben.

Es gibt viele Tools zum Visualisieren oder Analysieren von Datenbanken. Ebenfalls gibt es einige Programme für Datenbank Migration. Dazu gehört die GuttenBase Bibliothek. Diese bietet durch das Hinzufügen von Migrationsoperationen eine gewisse Flexibilität während des Migrationsprozesses an. Um diese Migrationsoperationen am effizientesten auszunutzen und eine schnellere und anpassbare Migration durchzuführen, lässt sich GuttenBase stark optimieren.

Diese Arbeit beschäftigt sich mit der Frage, wie sich die von der Firma Akquinet AG entwickelte Bibliothek optimieren lässt.

## 1.2 Zielsetzung

Die GuttenBase Bibliothek lässt sich durch unterschiedliche Weiterentwicklungen optimieren. Im Rahmend dieser Arbeit sollte eine eigene Anwendungsoberfläche (GuttenBase Plugin) für Datenbank Migration basierend auf GuttenBase konzipiert, implementiert und anschließend evaluiert werden.

Das GuttenBase Plugin soll die wichtigsten Funktionalitäten von GuttenBase unterstützen. Diese werden bei der Anforderungsanalyse genauer erläutert (siehe Abschnitt ??).

Um ein benutzerfreundliches System zu erzielen, ist es wichtig dass die zu entwickelnde Anwen-

dungsoberfläche den Grundsätzen der Informationsdarstellung entsprechen. Diese wurden in der Norm DIN EN ISO 9241-112 vorgestellt und beinhalten folgende Grundsätze:

- Entdeckbarkeit: Informationen sollen bei der Darstellung erkennbar sein und als vorhanden wahrgenommen werden.
- Ablenkungsfreiheit: Erforderliche Informationen sollen wahrgenommen werden, ohne Störung von weiteren dargestellten Informationen.
- Unterscheidbarkeit: Elemente oder Gruppen von Elementen sollen voneinander unterschieden werden können. Die Darstellung sollte die Unterscheidung bzw. Zuordnung von Elementen und Gruppen unterstützen.
- Eindeutige Interpretierbarkeit: Informationen sollen verstanden werden, wie es vorgesehen ist.
- Kompaktheit: Nur notwendige Informationen sollen dargestellt werden.
- Konsistenz: Informationen mit ähnlicher Absicht sollen ähnlich dargestellt werden und Informationen mit unterschiedlicher Absicht sollen in unterschiedlicher Form dargestellt werden.

Die genannten Grundsätze sollen im Zusammenhang mit den Grundsätzen für die Benutzer-System-Interaktion („Dialogprinzipien“) angewendet werden. Diese beinhalten, nach der Norm DIN EN ISO 9241-11, folgende Grundsätze:

- Aufgabenangemessenheit: Schritte sollen nicht überflüssig sein und keine irreführende Informationen beinhalten.
- Selbstbeschreibungsfähigkeit: Es sollen nur genau die Informationen dargestellt werden, die für einen bestimmten Schritt erforderlich sind.
- Erwartungskonformität: Das System verhält sich nach Durchführung einer bestimmten Aufgabe wie erwartet.
- Lernförderlichkeit: Der Benutzer kann den entsprechenden Schritt durchführen, ohne ein Vorwissen bzw. eine Schulung zu haben.
- Steuerbarkeit: Der Benutzer kann konsequent und ohne Umwege in Richtungen gehen, die für die zu erledigende Aufgabe erforderlich sind.
- Fehlertoleranz: Das System soll den Benutzer vor Fehlern schützen, und wenn Fehler gemacht werden, sollen diese mit minimalen Aufwand behoben werden können.
- Individualisierbarkeit: Der Benutzer kann Anwendungsoberfläche durch individuelle Voreinstellungen anpassen.

Die oben genannten Grundsätze stellen sicher, dass das GuttenBase Plugin effektiv, effizient und zufriedenstellend ist. Diese sind die drei Ziele der Gebrauchstauglichkeit (Usability).

## 1.3 Aufbau der Arbeit

Zu Beginn der Arbeit werden einige Grundbegriffe für Datenbank Migration erläutert. Außerdem werden die Eigenschaften der GuttenBase Bibliothek vorgestellt.

Zusätzlich werden aktuelle Tools für Datenbank Migration erwähnt.

Der Hauptteil dieser Arbeit beschäftigt sich hauptsächlich mit der Umsetzung des Guttenbase Plugins. Dabei wird zuerst eine Anforderungsanalyse durchgeführt, um den Soll-Zustand zu definieren. Um die technische Machbarkeit zu prüfen und Zeit bei der Entwicklung zu sparen, werden bei der Analyse einige GUI-Prototypen erstellt. Somit wird am Anfang der Umsetzung klar sein, wie die zu entwickelnde Anwendungsoberfläche die Funktionalitäten von Guttenbase unterstützen würde. Außerdem wird die Umsetzungsform begründet.

Die Software Architektur erfolgt im darauffolgenden Abschnitt. Diese wird basierend auf den Siemens Blickwinkel erstellt. Zunächst werden die verwendeten Technologien sowie das Ergebnis vorgestellt.

Im darauffolgenden Kapitel wird das Ergebnis kurz evaluiert. Dabei wird ein Experten-Interview durchgeführt.

Anschließend gibt es eine Zusammenfassung sowie Ideen für Optimierungsmöglichkeiten.



# Grundlagen

Dieses Kapitel liefert einen allgemeinen Einblick über einige Grundaspekte der Datenbank Migration sowie der GuttenBase Bibliothek. Zusätzlich gibt es eine kleine Einführung in die IntelliJ Plugin Entwicklung.

Außerdem werden verwandte Arbeiten vorgestellt.

## 2.1 Datenbankmanagementsysteme

Damit Daten auf einem Computer verwaltet werden können, werden Datenbankmanagement Systeme (DBMS) benötigt. Diese sind leistungsfähige Programme für die flexible Speicherung und Abfrage strukturierter Daten.

Außerdem hilft ein DBMS bei der Organisation und Integrität von Daten und regelt den Zugriff auf Datengruppen [Gei14].

Ein DBMS kann aus einem einzelnen Programm bestehen. Dies ist z. B. bei einem Desktop-DBMS zu sehen. Es kann jedoch aus verschiedenen Programmen bestehen, die zusammenarbeiten und die Funktion des DBMS bereitstellen. Dies ist z. B. bei den servergestützten Datenbanksystemen der Fall.

Um eine Datenbank Anwendung zu implementieren, muss auf das Datenbankmodell geachtet werden. Dies stellt die Daten einer Datenbank und deren Beziehungen abstrakt dar. Meistens wird ein relationales Datenbankmodell eingesetzt. Dies hat, im Gegensatz zu den anderen Datenbankmodellen, keine strukturelle Abhängigkeit und versteckt die physikalische Komplexität der Datenbank komplett vor den Anwendern.

Es stehen zahlreiche Datenbankmanagementsysteme zur Verfügung. Folgendes sind einige der gängigsten DBMS:

- Microsoft SQL Server
- MS-Access
- MySQL

- PostgreSQL
- HSQLDB
- H2 Derby
- Oracle
- DB2
- Sybase

Um ein geeignetes DBMS auszuwählen, gibt es viele Kriterien wie die Ausführungszeit, CPU- und Speicher Nutzung. Der Artikel von Youssif Bassil, A Comparative Study on the Performance of the Top DBMS Systems, im Jahr 2011 vergleicht einige Datenbankmanagementsysteme anhand der genannten Kriterien.

## 2.2 Datenbankmigration

Datenbank Migration wird immer mehr von Unternehmen bzw. Organisationen gebraucht.

Die Migration von Datenbanken dient zum Verschieben der Daten von der Quell-Datenbank zur Ziel-Datenbank einschließlich die Schemaübersetzung und Datentransformation.

Mögliche Gründe für eine Datenbank Migration sind:

- Upgrade auf eine neue Software oder Hardware
- Änderung der Unternehmensrichtlinien
- Investition in IT-Dienstleistungen
- Integration von Datenquelle in ein System
- Zusammenführen mehrerer Datenbanken in einer Datenbank für eine einheitliche Datenansicht.
- Wartung des existierenden Systems ist schwer oder nicht möglich.

Außerdem gibt es unterschiedliche Strategien für Datenbank Migration. Diese können in drei Kategorien unterteilt werden:

1. Migration durch objekt orientierte Schnittstellen:

Bei dieser Strategie werden Daten in form von Objekten bzw. XML Dateien verarbeitet. Dafür wird ein bidirektionales Mapping benötigt, objektbasierte Schemas in Datenbank Schemas zu übersetzen.

2. Datenbank Integration:

Hier wird die Quell-Datenbank mit der Ziel-Datenbank verbunden, wodurch der Eindruck entsteht, als ob alle Daten in einer einzigen Datenbank gespeichert sind.

3. Datenbank Migration:

Die Quell-Datenbank wird in die Ziel-Datenbank kopiert. Dabei werden Schemas in ein



Zielschema semantisch übersetzt werden. Darauf basierend werden die enthaltenen Daten konvertiert.

## 2.3 Stand der Dinge

Eines der Hauptprobleme in der Softwareindustrie besteht darin, eine hochwertige Datenverwaltung sicherzustellen. Dies ist auch der Fall bei einer Datenbank Migration, wobei die mit dem Migrationsworkflow verbundenen Aufgaben vielfältig und kompliziert sind. Das manuelle Ausführen dieser Aufgaben erfordert viel Zeit und ein sehr erfahrenes Team. Um Zeit und Kosten bei der Migration zu sparen und um wiederholende Aufgaben zu automatisieren, bieten sich zahlreiche Tools bzw. Prototypen für Datenbank Migration (DBMT für Database Migration Tool). Einige dieser Tools werden in der Tabelle 2.1 vorgestellt. Diese basiert auf den Vorschlag von Jutta Hortsmann, J.

### 2.3.1 GutenBase

Viele Software Unternehmen haben sich dafür entschieden, ein eigenes Tool für Datenbankmigration zu entwickeln. Dies ist der Fall bei der Firma Akquinet AG, wo die Open Source Bibliothek GutenBase in 2012 entwickelt wurde. Da GutenBase open source ist, wurde sie in weiteren Schritten weiterentwickelt und um zusätzliche Funktionen erweitert.

Anderes als die in der Tabelle 2.1 vorgestellten Tools, bietet GutenBase eine gewisse Flexibilität bei der Migration. Für die Migration einer Datenbank ist häufig eine benutzerdefinierte Lösung erforderlich. Z. B. durch das Umbenennen von Tabellen bzw. Spalten in der Zieldatenbank, das Umwandeln von Spaltentypen, das Ausschließen von bestimmten Tabellen bzw. Spalten usw.

Dazu können die sogenannten Konfigurationshinweise hinzugefügt werden. Diese können durch das Überschreiben der Mapping Klassen spezifiziert werden. Standardmäßig wird eine Standardimplementierung der Hinweise nach dem Verbinden der Datenbanken hinzugefügt. Diese können jedoch von dem Nutzer überschrieben werden.

### 2.3.2 IntelliJ

Die von der Firma JetBrains und in Java entwickelte Entwicklungsumgebung (IDE), IntelliJ IDEA, ist ein Teil einer Reihe von ähnlich JetBrains Entwicklungsumgebungen wie Clion, PyCharm, PhpStorm, DataGrip usw. Diese basieren auf dem gleichen Kern, nämlich den IntelliJ

Plattform SDK, welcher eine freie Lizenz hat und kann von Dritten zum Erstellen von IDEs verwendet werden.

IntelliJ IDEA ist mit der Implementierung in Java, Kotlin, Groovy und Scala geeignet. Sie hat außerdem unterschiedliche Funktionalitäten. Dazu gehört ein GUI-Editor, ein Build-Management Tool (z. B. Gradle) und andere Frameworks z. B. für Versionskontrolle, Refactoring, und Test. Der Funktionsumfang dieser IDE kann mittels Plugins erweitert werden.

Name	Quell-DBMS	Ziel-DBMS	Lizenz	Betriebssysteme
OSDM Toolkit (Apptility)	Oracle, SyBase, Informix, DB2, MS Access, MS SQL	PostgreSQL, MySQL	Frei	Windows, Linux, Unix und Mac OS
DB Migration (Access)	Oracle und MS SQL	PostgreSQL und MySQL	Kommerziell	Windows
Mssql2 Pgsql (OS Project)	MS SQL	PostgreSQL	Frei	Windows
MySQL Migration Toolkit (MySQL AB)	MS Access und Oracle	MySQL	Frei	Windows
Open DBcopy (Puzzle ITC)	Alle RDBMS	Alle RDMS	Frei	Betriebssystem-unabhängig
Progression DB (Versora)	MS SQL	PostgreSQL, MySQL und Ingres	Frei	Linux und Windows
Shift2Ingres (OS Project)	Oracle und DB2	Ingres	Frei	Betriebssystem-unabhängig
SQLPorter (Real Soft Studio)	Oracle, MS SQL, DB2 und Sybase	MySQL	Kommerziell	Linux, Mac OS und Windows
SQLWays (Ispirer)	Alle RDMBS	PostgreSQL und MySQL	Kommerziell	Windows
SwisSQL Data Migration Tool (AdventNet)	Oracle, DB2, MS SQL, Sybase und MaxDB	MySQL	Kommerziell	Windows
SwisSQL SQLOne Console (AdventNet)	Oracle, MSSQL, DB2, Informix und Sybase	PostgreSQL und MySQL	Kommerziell	Windows
MapForce (Altova)	SQL Server, DB2, MS Access, MySQL und PostgreSQL	SQL Server, DB2, MS Access und Oracle	Kommerziell	Windows, Linux und Mac OS
Centerprise Data Integrator (Astera)	SQL Server, DB2, MS Access, MySQL und PostgreSQL	SQL Server, DB2, MS Access, MySQL und PostgreSQL	Kommerziell	Windows
DBConvert (DB Convert)	Oracle, DB2, SQLite, MySQL, PostgreSQL, MS Access und Foxpro	Oracle, DB2, SQLite, MySQL, PostgreSQL, MS Access und Foxpro	Kommerziell	Windows
Squirrel DBCopy Plugin (Sourceforge)	Alle RDBMS	Alle RDBMS	Frei	Alle Betriebssysteme

Tabelle 2.1 Database Migration Tools



# **Anforderungsmanagement**

## **3.1 Anforderungsermittlung**

### **3.1.1 Funktionale Anforderungen**

### **3.1.2 Systemanforderungen**

## **3.2 Anforderungsanalyse**

## **3.3 Anforderungsbeschreibung**



# Konzeption und Implementierung

## 4.1 Anwendungsfunktionalität

## 4.2 Konzeptentwicklung

In diesem Abschnitt werden grundlegende Aspekte der Softwarearchitektur vom GUTTENBase Plugin vorgestellt. Diese werden basierend auf der Empfehlung im IEEE-Standard IEEE STD 1471-2000 erstellt.

Die Architektur wird aus zwei Sichten (Views) betrachtet. Eine Sicht repräsentiert dabei das Softwaresystem aus der Perspektive einer verwandten Menge von Aspekten. Jede Sicht stellt spezifische Informationen bereit.

Es ist außerdem zu beachten, dass sich die Architektur auf die Ergebnisse der funktionalen Anforderungsanalyse sowie auf die Architektur der Zielplattform (IntelliJ Plattform) bezieht. Im Folgenden werden die einzelnen Sichten genauer vorgestellt.

### 4.2.1 Konzeptionelle Sicht

Im Folgenden wird die Konzeptionelle Sicht des Systems vorgestellt. Hier wird das System noch unabhängig von den Implementierungsentscheidungen betrachtet.

Die konzeptionelle Sicht wird als Komponentendiagramm in der Abbildung [4.1](#) dargestellt.

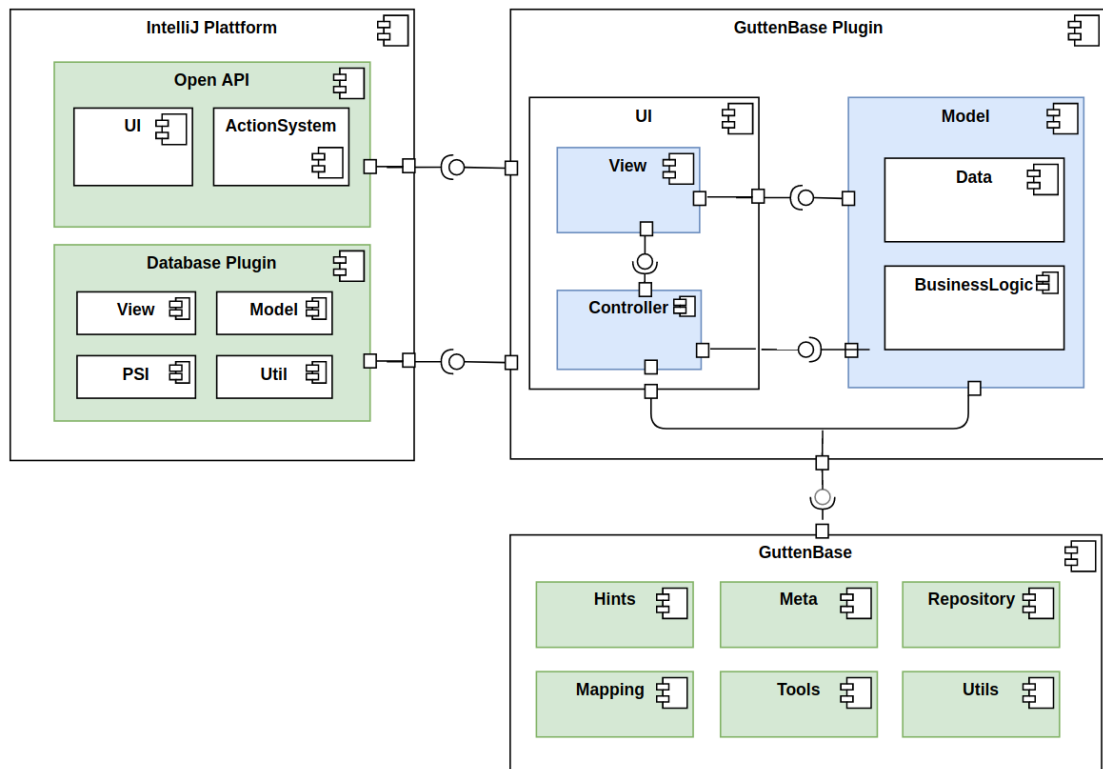


Abbildung 4.1 Komponentendiagramm für die konzeptionelle Sicht

Das Komponentendiagramm besteht aus der GuttenBase Plugin Komponente, welche das zu entwickelnde System darstellt, der IntelliJ Plattform Komponente und der GuttenBase Komponente. Es werden nur Komponenten veranschaulicht, die von unserem System benötigt sind. Diese werden im Folgenden genauer beschrieben.

#### 4.2.1.1 GuttenBase Plugin

Das GuttenBase Plugin wird nach dem MVC-Architekturstil konzipiert und besteht aus folgenden Komponenten:

##### Model

Das Modell enthält Daten, die von der View Komponente dargestellt werden. Hier liegt außerdem die Geschäftslogik(**BusinessLogik**), welche für die Änderung der Daten zuständig ist.



## View

Die View Komponente ist für die Darstellung der Daten für den Benutzer verantwortlich. Hier werden alle UI-abhängigen Aspekte wie Layout, Schriftart usw. behandelt.

## Controller

Die Controller Komponente ist für die Interaktion mit dem Benutzer verantwortlich. Sie wird von der View Komponente über Benutzerinteraktionen informiert und wertet sie aus. Anschließend können Änderungen an den Daten der Model Komponente sowie Anpassungen an der View Komponente angenommen werden.

### 4.2.1.2 IntelliJ Plattform

Wie in der Abbildung 4.1 zu sehen ist, interagiert das GuttenBase Plugin mit mehreren Komponenten der IntelliJ Plattform. Es wurden dabei nur die verwendeten Komponenten dargestellt. Diese sind in zwei Komponenten beinhaltet:

## Open API

Die IntelliJ Open API beinhaltet viele Komponenten, die auch von der IntelliJ IDEA Community Edition verwendet werden und für Plugin-Entwickler zur Verfügung stehen. Es können z. B. UI-Komponenten aus der Komponente **UI** für die Implementierung des Plugins benutzt werden. Außerdem wird **ActionSystem** Komponente benötigt, um bestimmte Aktionen, wie das Starten des Plugins, auszulösen.

## Database Plugin

Da das GuttenBase Plugin viel mit den Datenbanken umgeht, die in IntelliJ konfiguriert wurden, ist eine Interaktion mit dem Database Plugin sehr sinnvoll. Dazu bieten sich viele Komponenten für unterschiedliche Zwecke. Die für das GuttenBase Plugin benötigten Komponenten sind die **Model** Komponente (um evt. eine Datenbank Konfiguration zu bekommen), die **PSI** Komponente (um die Elemente der Datenbank zu bekommen), die **Util** Komponente (um ein Datenbank-Schema zu bekommen) und die **View** Komponente (um aus der Übersicht des Database Plugins Datenbank-Inhalte zu bekommen).

## 4.2.2 Modulsicht

Die Modulsicht zeigt die Struktur des GutenBase Plugins in Form von Modulen und deren Beziehungen zueinander. Hierbei werden die Komponenten und Konnektoren der konzeptionellen Sicht auf Module, Schichten und Subsysteme abgebildet. Diese werden in Paket- und Klassendiagramme verfeinert. Zur Wahrung der Übersichtlichkeit wird die Modulsicht nach den unterschiedlichen Übersichten unterteilt. Es gibt insgesamt vier Übersichten, die das gesamte System abdecken. Pro Übersicht werden nur Klassen bzw. Methoden beschrieben, die entsprechenden Anwendungsfälle realisieren.

### 4.2.2.1 Modulsicht der Übersicht der Konfigurationsschritte

Die Modulsicht der Abbildung 4.2 beschreibt alle beteiligten Klassen, die beim Erstellen und Verwalten der Konfigurationsschritte eingesetzt sind. Diese werden entsprechend der konzeptionellen Sicht aufgeteilt, nämlich in den View, Model und Controller Paketen.

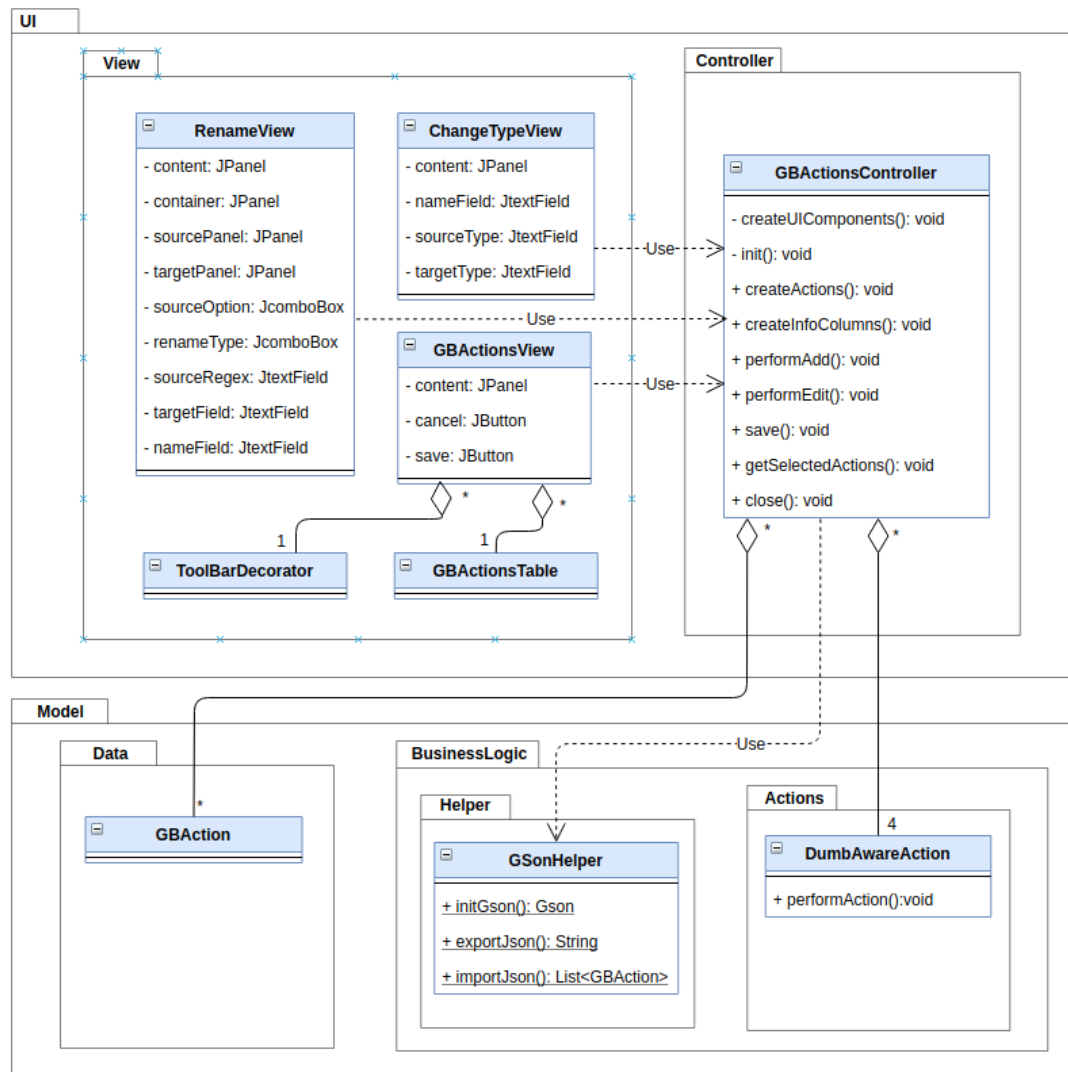


Abbildung 4.2 Modulsicht der Übersicht der Konfigurationsschritte

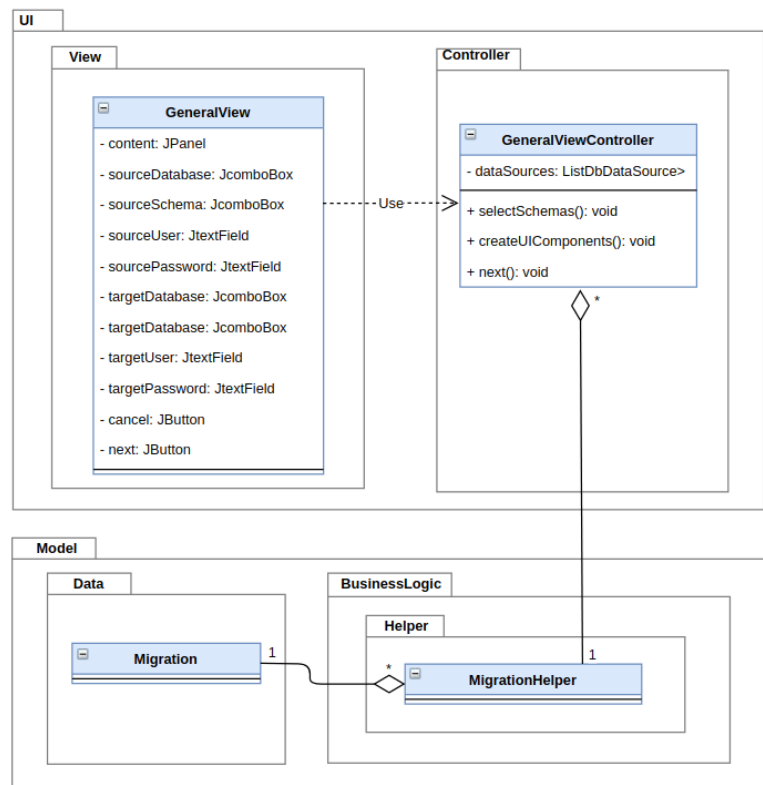
In dem View Paket werden alle Klassen dargestellt, die für die Darstellung der Konfigurationsschritte zuständig sind. Diese basieren auf unterschiedliche Swing Komponenten. Die Klasse `GBActionsView` repräsentiert dabei die Hauptübersicht der Konfigurationsschritte und beinhaltet die `GBActionsTable` Klasse, welche die Auflistung der Konfigurationsschritte übernimmt. Außerdem wird die `ToolBarDecorator` Klasse für die Darstellung der möglichen Aktionen benötigt.

Das Controller Paket enthält hierbei die `GBActionsController` Klasse. Diese hat folgende Aufgaben:

- UI-Komponenten vor dem Anzeigen vorbereiten, indem Daten aus dem Paket Model erzeugt bzw. geladen werden.
- Konfigurationsschritte erstellen. Dies passiert nach der Benutzer einen bestimmten Konfigurationsschritt hinzufügen möchte und diesen in der GBActionsView übersicht selektiert. Zunächst wird die entsprechende DumbAwareAction Klasse aufgerufen, um die entsprechende Aktion durchzuführen. Die DumbAwareAction Klasse ist im Paket BusinessLogic zu finden und könnte z. B. für das Anzeigen der Übersicht für das Erstellen des Umbenennen- oder Datentyp-Ändern-Konfigurationsschritt verwendet werden. Diese werden durch die RenameView und ChangeTypeView Klassen realisiert. Analog dazu erfolgt das Editieren der Konfigurationsschritte.
- Konfigurationsschritte speichern, indem die hinzugefügte Konfigurationsschritte in JSON konvertiert und dann exportiert werden. Dafür ist die GsonHelper Klasse des BusinessLogic Pakets zuständig.

#### 4.2.2.2 Modulsicht der allgemeinen Übersicht

Diese Modulsicht zeigt die beteiligten Klassen, die beim Verbinden der zu migrierenden Datenbanken genutzt werden. Diese wird in der Abbildung [4.3](#) dargestellt.

**Abbildung 4.3** Modulsicht der allgemeinen Übersicht

Die GeneralView Klasse ist für die Interaktion mit dem Benutzer verantwortlich und enthält alle Eingabefelder und Texte. Die GeneralViewController Klasse ist für das Laden der Datenbank-elemente zuständig. Klickt der Benutzer auf das „Next“-Button, werden alle Eingaben über die MigrationHelper Klasse des Pakets BusinessLogic übergeben. Diese Informationen werden dann in der Migration Klasse des Model Pakets gespeichert.

#### 4.2.2.3 Modulsicht der Konfigurationsübersicht

Die Modulsicht in der Abbildung 4.4 zeigt die für die Konfigurationsübersicht relevanten Klassen.

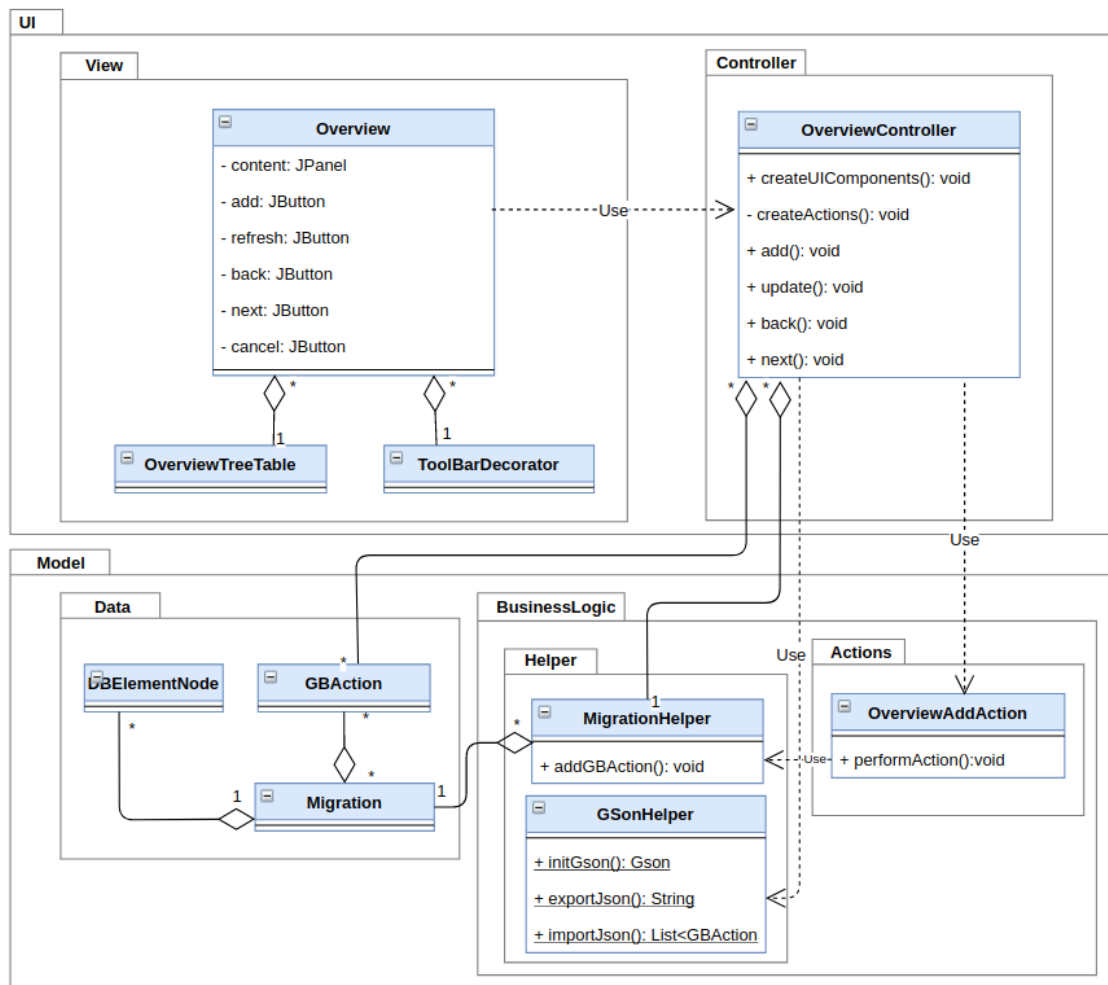


Abbildung 4.4 Modulsicht der Konfigurationsübersicht

In der Konfigurationsübersicht (Overview) enthält die OverviewTreeTable Klasse, welche alle Quell-Datenbankelemente auflistet und die ToolbarDecorator Klasse, die gespeicherten Konfigurationsschritte anzeigt. Außerdem stehen einige Buttons für die Benutzerinteraktion zur Verfügung.

Wie bei den vorherigen Modulsichten, stellt die OverviewController Klasse Methoden für folgende Zwecke bereit:

- Das Laden der Datenbankelemente: Hierbei werden die gespeicherten Konfigurationsschritte mithilfe der GsonHelper Klasse importiert und die Datenbankelement (DBElementNode) aus der Quell-Datenbank erstellt.
- das Anzeigen der Konfigurationsschritte: Konfigurationsschritte werden nach Kompatibilität mit den selektierten Datenbankelementen behandelt. Ist ein Konfigurationsschritt mit

einem Datenbankelement geeignet, wird dieser klickbar angezeigt, außerdem wird stattdessen ein ausgegrautes Button dargestellt.

- Hinzufügen von Konfigurationsschritten: Wenn der Benutzer ein Datenbankelement selektiert und dann einen entsprechenden Konfigurationsschritt hinzufügt, wird die OverviewAddAction Aktion von dem BusinessLogic Paket ausgelöst. Dabei wird der entsprechende Konfigurationsschritt durch die MigrationHelper Klasse zur Migration hinzugefügt.

#### 4.2.2.4 Modulsicht der Ergebnisübersicht

Die Modulsicht der Abbildung 4.5 stellt die für die Ergebnisübersicht (ResultView) zuständigen Klassen. Außerdem wird die Fortschritt Übersicht (ProgressView) miteingebunden, da diese vom selben Controller verwaltet wird.

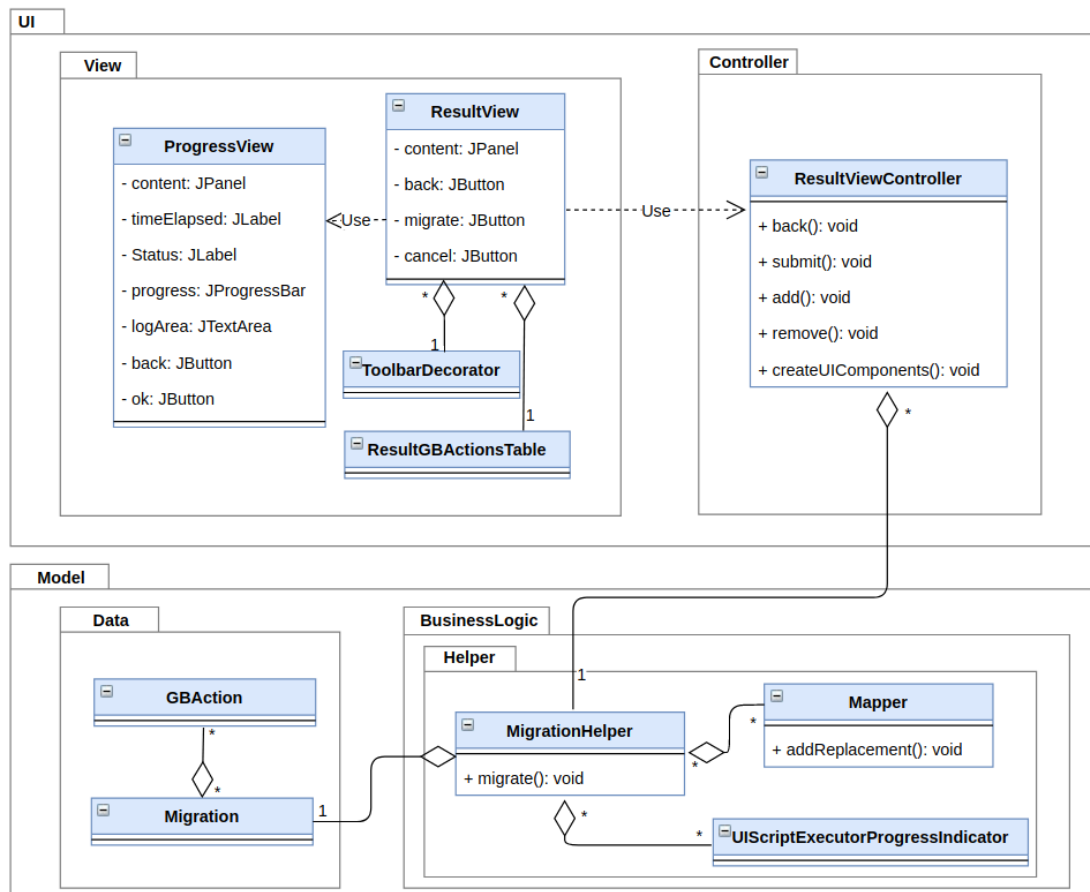


Abbildung 4.5 Modulsicht der Ergebnisübersicht

Ähnlich wie bei den anderen Übersichten, enthält die ResultView Klasse eine Tabelle (Result-

GBActionsTable), die alle hinzugefügten Konfigurationsschritte enthält und ein ToolbarDecorator, wo die Aktionen zum Löschen und Hinzufügen angezeigt werden.

Auf der anderen Seite stellt die ResultViewController Klasse Methoden für das Löschen und das Hinzufügen von Konfigurationsschritte sowie für das Starten des Migrationsprozesses bereit. Diese werden im Folgenden genauer erklärt:

- Hinzufügen: Wenn der Benutzer noch mehr Konfigurationsschritte zur Migration hinzufügen möchte, wird die aktuelle Übersicht zur Übersicht der Konfigurationsschritte (Overview) umgeleitet.
- Löschen: Wie das Hinzufügen, erfolgt das Löschen von Konfigurationsschritte (die schon zur Migration hinzugefügt wurden) durch die Entfernung von dem ausgewählten Konfigurationsschritt (GBAction) aus der Migration Klasse.
- Migration starten: Nach dem Klick auf das „Migrate“Button, wird der Migrationsprozess gestartet. Dieser erfolgt durch die MigrationHelper Klasse. Dabei wird das Connector Repository der GuttenBase Bibliothek entsprechend der Konfigurationsschritte konfiguriert (siehe 2.3.1) und anschließend das Kopieren durch das DefaultTableCopyTool gestartet.

Parallel dazu wird die Fortschritt Übersicht (ProgressView) angezeigt, um Informationen über den laufenden Prozess zu sehen. Dies ist durch die UIScriptExecutorProgressIndicator Klasse ermöglicht.

## 4.3 Technologieauswahl

## 4.4 Umsetzungsform

Um eine optimale Nutzung des GuttenBase Plugins zu erzielen, soll auf die Umsetzungsform geachtet werden.

Das zu entwickelnde Tool kann z. B. als eine Desktop Applikation, Web Applikation oder als Plugin einer anderen Anwendung realisiert werden.

In der Tabelle 4.1 werden einige Vor- und Nachteile jeder Alternative erläutert.

Alle drei Alternativen haben Pros und Contras allerdings ist die schnellere Erreichung von vielen Nutzern sowie die Einfache Installation bei der IDE Plugin Entwicklung entscheidend.

Zunächst soll für eine konkrete IDE entschieden werden. Um diese auszuwählen, muss auf die Anzahl der Nutzer, die Verfügbarkeit der Dokumentation für Plugin Entwicklung sowie die Unterstützung von Datenbanken geachtet werden.

Einer der bekanntesten Methoden, um die genaue Beliebtheit einer Programmiersprache bzw. eine IDE herauszufinden, ist der PYPL-Index. Er basiert auf Rohdaten aus Google Trends. PYPL enthält den TOP-IDE-Index, welches analysiert, wie oft IDEs bei Google durchgesucht werden. Die Suchanfragen spiegeln zwar nicht unbedingt die Beliebtheit der IDEs. Allerdings hilft einen



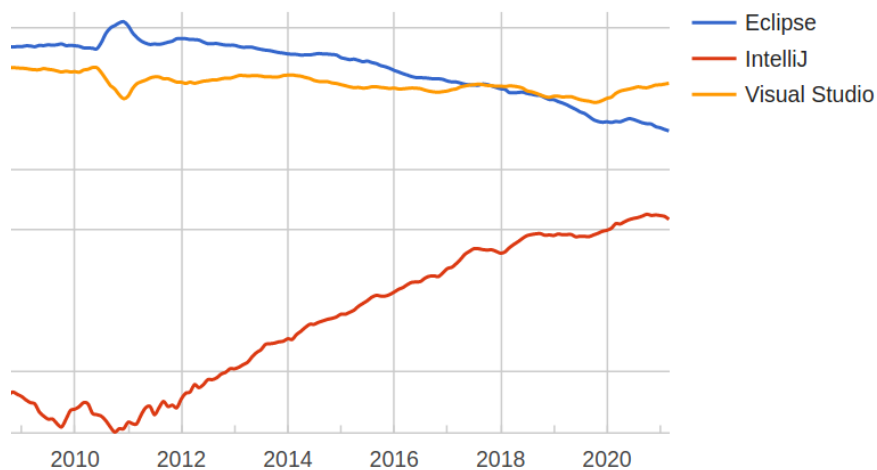
Alternative	Vorteile	Nachteile
Desktop App	<ul style="list-style-type: none"><li>• Offline immer verfügbar</li><li>• Volle Kontrolle über die Anwendung und die enthaltenen Daten.</li><li>• Bessere Leistung, da kein Browser als Zwischenschicht existiert.</li></ul>	<ul style="list-style-type: none"><li>• Plattformabhängig</li><li>• Hohe Entwicklungskosten</li><li>• Installation ist notwendig</li></ul>
Web App	<ul style="list-style-type: none"><li>• Installation oder manuelle Updates sind nicht notwendig.</li><li>• geringere Entwicklungs- und Wartungskosten, da die Anwendung unabhängig von lokalen Endgeräten ist.</li></ul>	<ul style="list-style-type: none"><li>• Offline meistens nicht verfügbar.</li><li>• Geringere Leistung.</li></ul>
IDE Plugin Entwicklung	<ul style="list-style-type: none"><li>• Für IntelliJ Benutzer ist das einfach und intuitiv zu benutzen.</li><li>• Manche Komponenten bzw. Funktionalitäten der zu erweiternden IDE können wiederverwendet werden, was die Entwicklungsdauer verkürzt.</li><li>• Intuitive Nutzung sowie eine einheitliche Benutzeroberfläche wie die benutzte IDE.</li></ul>	<ul style="list-style-type: none"><li>• Die Flexibilität beim Entwickeln ist durch die limitierte Erweiterbarkeit der IDE eingeschränkt.</li></ul>

**Tabelle 4.1** Umsetzungsmöglichkeiten

solchen Index enorm bei der Wahl einer Entwicklungsumgebung. Bei dieser Analyse sind die drei bekanntesten und für unseren Fall relevanten Entwicklungsumgebungen Visual Studio (erster Platz), Eclipse (zweiter Platz) und IntelliJ (sechster Platz). Außerdem hat sich der Index von IntelliJ IDE am stärksten erhöht (siehe Abbildung 4.6)

Bei einer anderen Umfrage (Jaxenter), mit welcher Entwicklungsumgebung am liebsten in Java programmiert wird, war IntelliJ sogar im ersten Platz mit 1660 Stimmen von 2934.

**Abbildung 4.6** Top IDE Index



Aus den oben erläuterten Daten und aufgrund der guten Dokumentation für Plugin Entwicklung wird das GuttenBase als ein IntelliJ Plugin umgesetzt.

#### 4.4.1 GuttenBase

Die Nutzung der GuttenBase Bibliothek wird in folgenden Schritten erklärt.

##### Schritt 1: Datenbankverbindung erstellen

Als erstes sollen Abhängigkeiten zu dem laufenden Projekt hinzugefügt werden. Die für GuttenBase benötigte Informationen sind:

- groupId: de.akquinet.jbosscg.guttenbase
- artifactId: GuttenBase
- version: 2.0.0

Zusätzlich sollen die Treiber-Klassen der Quell- und Ziel-DBMS hinzugefügt werden. Zunächst sollen die ConnectionInfo Klassen für erstellt werden. Diese beschreiben die Quell- und Ziel-Datenbanken und enthalten die für eine JDBC (Java Database Connectivity) Verbindung erforderlichen Attribute.

```
public class MySQLConnectionsInfo extends URLConnectorInfoImpl {
    public MySQLConnectionsInfo() {
        super("jdbc:mysql://localhost:3306/testdb", "user", "password",
            "com.mysql.jdbc.Driver", "aev", DatabaseType.MYSQL);
    }
}
```

**Abbildung 4.7** ConnectionInfo konfigurieren

Im nachhinein wird das Connector Repository konfiguriert werden. Dies enthält alle Konnektoren, die in der Datenbank Migration beteiligt sind.

```
public static final String SOURCE = "source";
public static final String TARGET = "target";
...
final ConnectorRepository connectorRepository = new
ConnectorRepositoryImpl();
connectorRepository.addConnectionInfo(SOURCE, new
PostgreSQLConnectionInfo());
connectorRepository.addConnectionInfo(TARGET, new
MySQLConnectionsInfo());
```

**Abbildung 4.8** Connector Repository konfigurieren

## Schritt 2: Hinweise hinzufügen

Meistens werden Konfigurationshinweise (hints) benötigt, um die Migration zu individualisieren. In der Dokumentation von GuttenBase<sup>1</sup> befindet sich eine Liste aller unterstützten Konfigurationshinweise.

Als Beispiel wird der Konfigurationshinweis ColumnMapperHint in der Abbildung dargestellt.

<sup>1</sup>Getting Started with GuttenBase (2018, 04.01)

<https://github.com/akquinet/GuttenBase/blob/master/Getting%20Started%20with%20GuttenBase.pdf>

```
connectorRepository.addConnectorHint(TARGET, new ColumnMapperHint() {  
    @Override  
    public ColumnMapper getValue() {  
        return new CustomColumnRenameName()  
            .addReplacement("name", "id_name")  
            .addReplacement("username", "id_username");  
    }  
})
```

Abbildung 4.9 ColumnMapperHint hinzufügen

### Schritt 3: Datenbank Migration durchführen

Anschließend kann die Migration mit den eventuell hinzugefügten Hinweisen durchgeführt werden. Dies passiert wie folgt:

- Das Datenbank Schema wird von der Quell-Datenbank in die Ziel-Datenbank kopiert. Dabei werden möglichst viele Unterschiede in Datentypdarstellung standardmäßig berücksichtigt.
- Die Kompatibilität von den Schemata wird geprüft. Hierbei wird nach gleichen Tabellen bzw. Spalten gesucht.
- Falls es keine Fehler beim Prüfen gibt, werden die Daten dann kopiert.

```
new CopySchemaTool(connectorRepository).copySchema(SOURCE, TARGET);  
final SchemaCompatibilityIssues schemaCompatibilityIssues = new  
    SchemaComparatorTool(connectorRepository).check(SOURCE, TARGET);  
if (schemaCompatibilityIssues.isSevere()) {  
    throw new SQLException(schemaCompatibilityIssues.toString());  
}  
new DefaultTableCopyTool(connectorRepository).copyTables(SOURCE,  
    TARGET);  
new CheckEqualTableDataTool(connectorRepository).checkTableData(SOURCE,  
    TARGET);
```

Abbildung 4.10 Datenbank Migration durchführen

#### 4.4.2 IntelliJ Platform

Für die erfolgreiche IntelliJ Plugin Entwicklung muss auf mehrere Aspekte geachtet werden wie die Projektstruktur und die häufig verwendeten Komponenten.

Die Plugin Entwicklung erfolgt in der IntelliJ IDE selbst. Deswegen kann das Plugin entweder in Java, Kotlin, Groovy oder Scala geschrieben werden.

Der von JetBrains empfohlene Weg für das Erstellen eines neuen Plugins ist das Gradle Projekt. Dabei muss die Option IntelliJ Platform Plugin ausgewählt werden, damit die Plugin Abhängigkeiten sowie die Basis-IDE automatisch konfiguriert werden. Zusätzlich muss die Datei plugin.xml

entsprechend des zu entwickelnden Plugins angepasst werden. Diese enthält wichtige Informationen, die in den folgenden Tags (Auszeichnungen) erklärt werden:

- **<name>**  
Der Name des Plugins. Er soll kurz und beschreibend sein.
- **<id>**  
Eine eindeutige Bezeichnung des Plugins. Diese kann nicht während der Entwicklung geändert werden.
- **<description>**  
Eine Kurze Beschreibung des Plugins.
- **<change-notes>**  
Eine Beschrei Beschreibung der Änderungen in der neusten Version des Plugins.
- **<version>**  
Die aktuelle Plugin Version.
- **<vendor>**  
Der Anbieter des Plugins. Hier kann zusätzlich eine Email Adresse angegeben werden.
- **<depends>**  
Abhängigkeiten zu Plugins oder Modulen.
- **<idea-version>**  
Die minimale und maximale Version der IDE, mit der das Plugin kompatibel ist.
- **<actions>**  
Definiert wie die Funktionalität des Plugins aufgerufen wird. Dies wird im folgenden Abschnitt behandelt.
- **<extensionPoints>**  
Die vom Plugin definierte Erweiterungspunkte. Diese können erlauben anderen Plugin-Entwicklern, auf bestimmte Daten zuzugreifen.
- **<extensions>**  
Erweiterungspunkte, die von IntelliJ-Plattform bzw. von anderen Plugins definiert sind und von dem zu entwickelnden Plugin verwendet werden.

## Action-System

Die am häufigsten verwendete Methode, um die Plugin Funktionalität aufzurufen, ist die Nutzung der sogenannten Actions vom Action-System der IntelliJ-Plattform.

Eine Aktion kann über ein Menüpunkt (menu item) oder einen Eintrag in der Symbolleiste ausgelöst werden. Dazu muss ein Eintrag in dem Actions Tag der plugin.xml Datei erfolgen. Dabei muss jede Action mindestens eine Id, eine Klasse und einen beschreibenden Text haben. I.d.R werden Menüpunkte nach Funktionaliät gruppiert. Um die Implementierte Action zu einer bestimmten Gruppe hinzufügen zu können, muss der Tag **<add-to-group>** verwendet werden.

Die Action Klasse muss von der AnAction Klasse abgeleitet werden und die actionPerformed() Methode überschreiben. Diese wird nach dem Klick auf das entsprechende Menüpunkt bzw. Symbolleiste aufgerufen.

## 4.5 Plugin Implementierung

### 4.5.1 Funktion: Liste der vorhandenen Migrationsoperationen

Um die Funktionalität des GuttenBase Plugins einfach und intuitiv für IntelliJ Nutzer zur Verfügung zu stellen, wurden Menüpunkte zum Database Plugin hinzugefügt. Diese wurden gemäß dem Grundsatz der Unterscheidbarkeit platziert. Somit kann die Übersicht der Migrationsoperationen nach dem Klick auf „Show Migration Actions“ geöffnet werden. Dies wird in der Abbildung 4.12 dargestellt. Diese enthält standardmäßig nur zwei Migrationsoperationen (Tabellen bzw. Spalten Ausschließen).

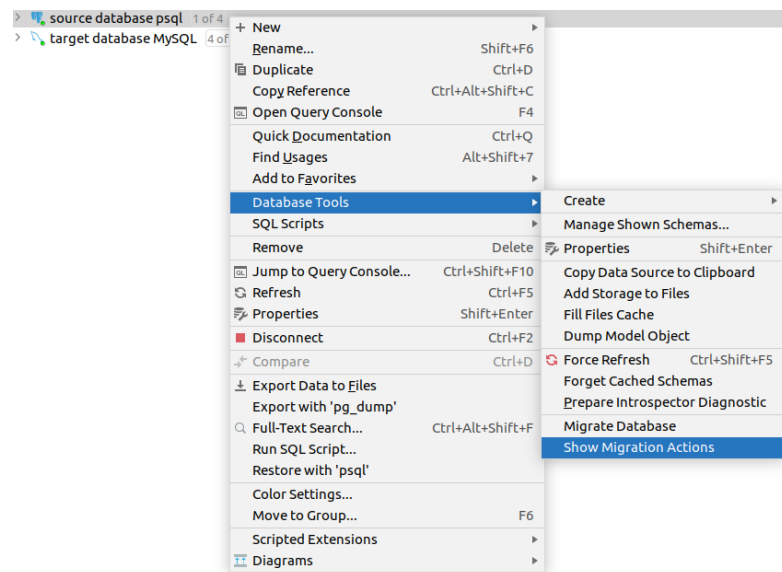


Abbildung 4.11 Übersicht der Migrationsoperationen öffnen

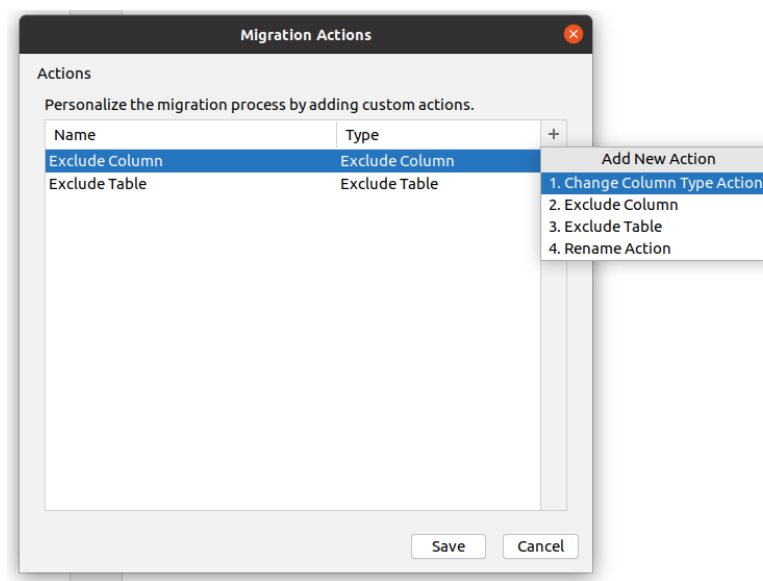


Abbildung 4.12 Übersicht der Migrationsoperationen

#### 4.5.2 Funktion: Hinzufügen einer neuen Migrationsoperation

Bei der Übersicht in der Abbildung 4.12 kann der Benutzer verschiedene Migrationsoperationen erstellen, wenn diese nicht bereits existieren. In diesem Szenario wird nur das Hinzufügen von der Migrationsoperation Umbenennen (Rename Action) dargestellt.

Nach dem Klick auf das entsprechende Button wird ein Dialog angezeigt, um die erforderliche Informationen einzugeben. Dabei wird der Name der Migrationsoperation festgelegt. Außerdem der Quell-Name durch einen regulären Ausdruck definiert. Anschließend wird die der Zeil-Name festgelegt.

Die in der Abbildung 4.13 angezeigte Migrationsoperation gilt für alle Datenbankelemente, deren Name die Zeichenkette „table“ enthält. Wenn diese an einem entsprechenden Datenbankelement angewendet wird, wird das Suffix „\_Test“ hinten hinzugefügt. Anschließend lässt sich die neu hinzugefügte Migrationsoperation durch das Klick auf das „Save“ Button speichern (siehe Abbildung 4.12). Dabei werden alle Migrationsoperationen nach JSON konvertiert und in einer externen Datei gespeichert. Dafür ist die Klasse GsonHelper verantwortlich (siehe Abbildung 4.14).

#### 4.5.3 Funktion: Verbindungserstellung

Wenn die Quell- und Ziel-Datenbank im Database Plugin eingerichtet sind, kann das Aktionsmenü nach Rechtsklick auf die Quell-Datenbank aktiviert werden. Wie in der Abbildung 4.11

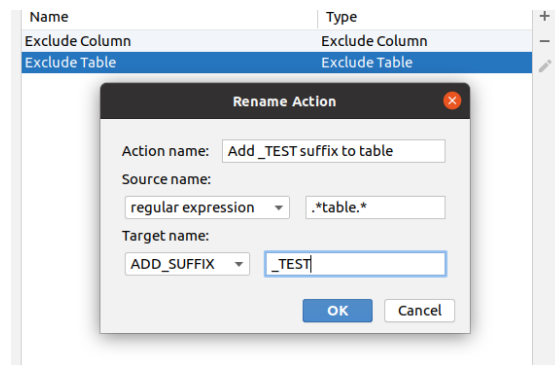


Abbildung 4.13 Migrationsoperation Umbenennen erstellen

```
public class GsonHelper {  
    public static String exportJSON(List<GBAction> gbActions, String fileName) throws IOException {  
        gbActions.forEach(gbAction -> System.out.println(gbAction.getName() + gbAction.getGBActionType()));  
        Gson gson = GsonHelper.initGson();  
        GBActionsJSON gbActionsJSON = new GBActionsJSON(gbActions);  
        String actionsString = gson.toJson(gbActionsJSON, GBActionsJSON.class);  
        FileWriter file = new FileWriter(fileName, append: false); //replace file  
        file.write(actionsString);  
        file.flush();  
        return new File(fileName).getAbsolutePath();  
    }  
  
    public static List<GBAction> importJSON(String fileName) throws FileNotFoundException {  
        Gson gson = GsonHelper.initGson();  
        JsonReader reader = new JsonReader(new FileReader(fileName));  
        GBActionsJSON gbActionsJSON = gson.fromJson(reader, GBActionsJSON.class);  
        gbActionsJSON.getGBActions().forEach(gbAction -> System.out.println(gbAction.getName() + gbAction.getGBActionType()));  
        return gbActionsJSON.getGBActions();  
    }  
}
```

Abbildung 4.14 Migrationsoperationen exportieren und importieren

angezeigt, kann der Benutzer auf das Button „Migrate Database“ klicken um die Übersicht der Datenbank Migration zu öffnen (siehe Abbildung 4.15). Dabei wird die Quell-Datenbank anhand der selektierten Datenbank automatisch selektiert. Außerdem muss das zu migrierende Schema der Quell-Datenbank sowie das Ziel-Schema ausgewählt werden. Zusätzlich müssen die Zugangsdaten jeder Datenbank angegeben werden, um die Datenbanken zu verbinden. Nach dem Klick auf das „Next“ Button, werden die Angaben erst geprüft. Wenn diese fehlerhaft sind, dann wird eine entsprechende Fehlermeldung ausgegeben. Ansonsten wird das ConnectorRepository (siehe 4.4.1) anhand der angegebenen Informationen erstellt und anschließend die nächste Übersicht (overview) angezeigt.

#### 4.5.4 Funktion: Konfiguration

Bei der Konfigurationsübersicht (siehe Abbildung 4.16) werden alle Elemente der Quell-Datenbank angezeigt. Hierbei wird Einzel- sowie Mehrfachauswahl ermöglicht.



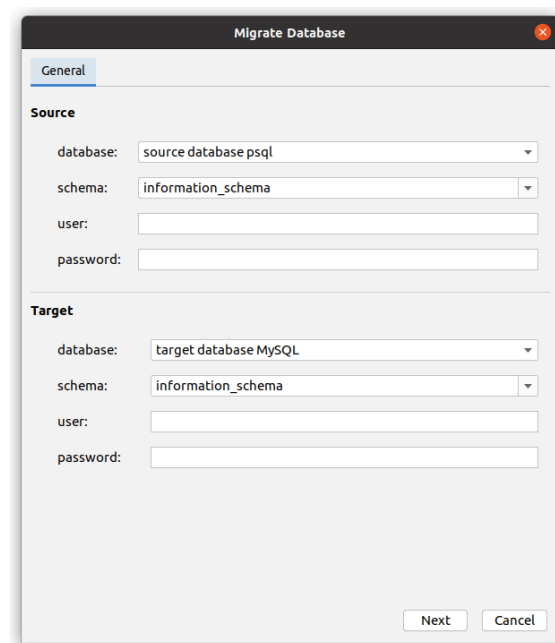


Abbildung 4.15 Allgemeine Übersicht der Datenbank Migration (generalView)

Außerdem werden alle Migrationsoperationen mithilfe der **GsonHelper** Klasse geladen werden. Diese werden abhängig von den selektierten Elementen unterschiedlich dargestellt. Wenn eine Migrationsoperation (GBAction) zu den ausgewählten Elementen passt, wird diese klickbar angezeigt, ansonsten wird diese deaktiviert und ausgegraut dargestellt. Dabei spielt die **matches** Methode der **GBAction** Klasse eine entscheidende Rolle. Diese wird entsprechend des Typen der Migrationsoperation. Für das Umbenennen wird z. B. geprüft, ob der gespeicherte reguläre Ausdruck zum Namen der selektierten Elemente passt (siehe Abbildung 4.17).

Bei jedem Hinzufügen wird die entsprechende Migrationsoperation zu der Liste aller Operationen hinzugefügt. Dabei wird eine neue Instanz erzeugt, die die benötigten Informationen des entsprechenden Datenbankelementes enthält (siehe Abbildung 4.18).

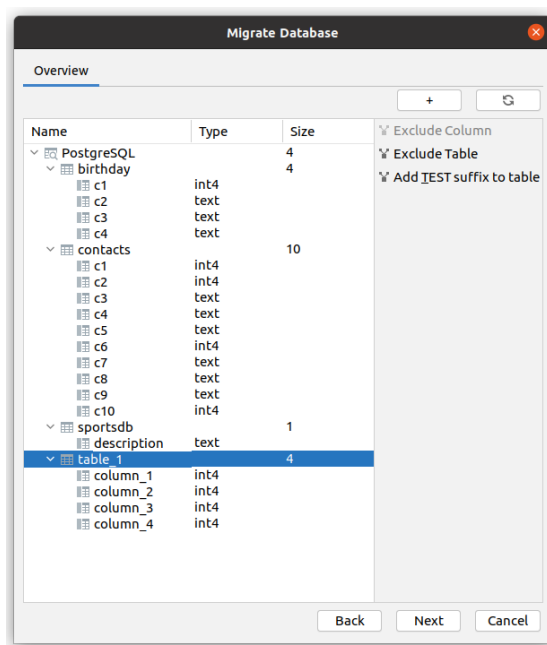


Abbildung 4.16 Konfigurationsübersicht (overview)

```
@Override
public boolean matches(MyDataNode node){
    return node.getName().matches(regExp);
}
```

Abbildung 4.17 matches Methode der Migrationsoperation Umbenennen

```
for (int row : rows) {
    MyDataNode node = (MyDataNode) overviewTreeTable.getValueAt(row, column: 0);
    GBAAction newGBAction;
    try {
        // create a new instance of the action (otherwise the same object will be overwritten).
        newGBAction = (GBAction) gbAction.clone();
    } catch (CloneNotSupportedException cloneNotSupportedException) {
        cloneNotSupportedException.printStackTrace();
        Messages.showErrorDialog(cloneNotSupportedException.getMessage(), title: "Error!");
        return;
    }
    newGBAction.setSource(node);
    if (node instanceof ColumnNode && newGBAction.getGBActionType().equals(GBActionType.RENAME)) {
        newGBAction.setGBActionType(GBActionType.RENAME_COLUMN);
    }
    else if (node instanceof TableNode && newGBAction.getGBActionType().equals(GBActionType.RENAME)) {
        newGBAction.setGBActionType(GBActionType.RENAME_TABLE);
    }
    migration.addGBAction(newGBAction);
}
```

Abbildung 4.18 das Hinzufügen von der Migrationsoperation Umbenennen

#### 4.5.5 Funktion: Migrationsdurchführung

Nach dem Klick auf das „Next“ Button, erhält der Benutzer eine Übersicht von allen hinzugefügten Migrationsoperationen (siehe Abbildung 4.19). Diese können nach Bedarf gelöscht werden.

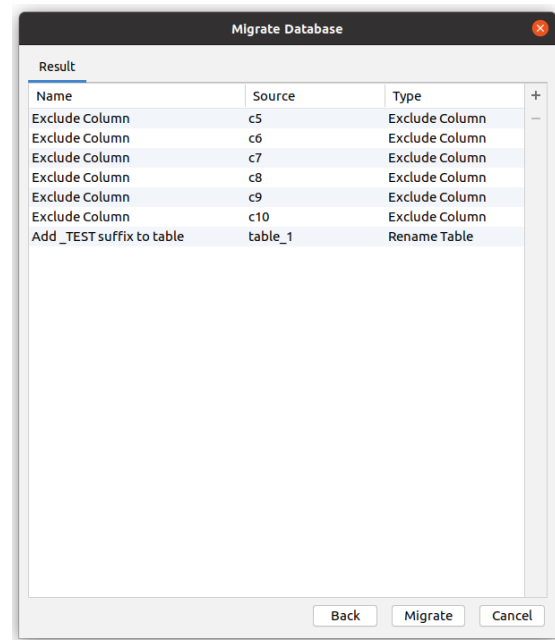


Abbildung 4.19 Ergebnisübersicht

Nach dem Klick auf das „Migrate“ Button, wird die Fortschrittsübersicht angezeigt (siehe Abbildung 4.22). Diese veranschaulicht den Migrationsprozess, welcher in einem neuen Thread ausgeführt wird. Bei der Migration werden die Mapper Klassen sowie die GuttenBase Connectors (siehe Abschnitt 4.4.1) entsprechend der hinzugefügten Migrationsoperationen zum Connector-Repository hinzugefügt. Danach werden die Daten von der Quell-Datenbank zur Zie-Datenbank kopiert (siehe Abbildung 4.20 bzw. Abbildung 4.21).

```
private void addConnectors() {
    connectorRepository.addConnectorHint(TARGET, (ColumnMapperHint) () -> {
        return columnRenameMapper;
    });
    connectorRepository.addConnectorHint(TARGET, (TableMapperHint) () -> {
        return tableRenameMapper;
    });
    connectorRepository.addConnectorHint(TARGET, (ColumnTypeMapperHint) () -> {
        return columnTypeMapper;
    });
    connectorRepository.addConnectorHint(SOURCE, (RepositoryColumnFilterHint) () -> {
        return column -> !excludedColumns.contains(column);
    });
    connectorRepository.addConnectorHint(SOURCE, (DefaultRepositoryTableFilterHint) () -> {
        return table -> !excludedTables.contains(table);
    });
    connectorRepository.addConnectorHint(SOURCE, (ScriptExecutorProgressIndicatorHint) () -> {
        return new UIScriptExecutorProgressIndicator(progressView);
    });
    connectorRepository.addConnectorHint(TARGET, (ScriptExecutorProgressIndicatorHint) () -> {
        return new UIScriptExecutorProgressIndicator(progressView);
    });
}
```

Abbildung 4.20 ConnectorHints hinzufügen

```
@Override
public void run() {
    updateMappers();
    addConnectors();
    try {
        new CopySchemaTool(connectorRepository).copySchema(SOURCE, TARGET);
        checkCompatibilityIssues();
        new DefaultTableCopyTool(connectorRepository).copyTables(SOURCE, TARGET);
        new CheckEqualTableDataTool(connectorRepository).checkTableData(SOURCE, TARGET);
    } catch (SQLException e) {
        e.printStackTrace();
        ApplicationManager.getApplication().invokeLater(() -> Messages.showErrorDialog(e.getMessage(), ERROR_TITLE));
        progressView.enableBack();
    }
}
```

Abbildung 4.21 Migration durchführen (MapperHelper Klasse)

Falls ein Fehler bei der Migration auftritt, wird eine entsprechende Fehlermeldung angezeigt und das „Back“ aktiviert, um Änderungen durchzuführen und die Migration nochmal zu starten. Der Benutzer bekommt außerdem einen Hinweis, Falls die Migration erfolgreich abgeschlossen ist.

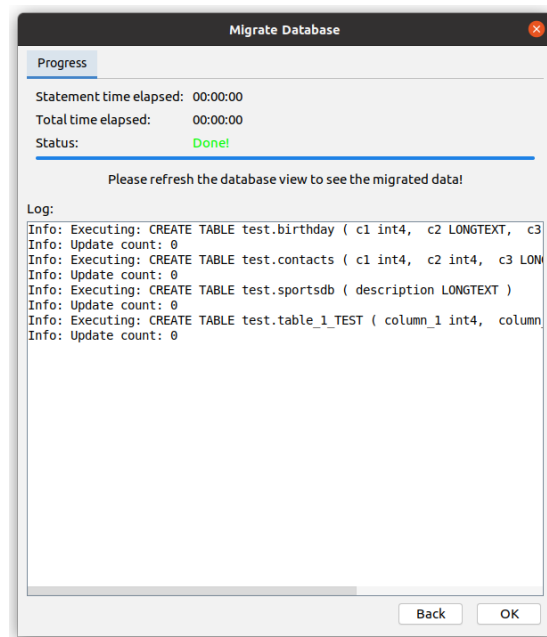


Abbildung 4.22 Fortschrittsübersicht



# **Test und Evaluation**

## **5.1 Testplan**

### **5.1.1 Experten Interview**

### **5.1.2 Unit-Tests**

### **5.1.3 Integrationstest**

## **5.2 Evaluation**

## **5.3 Anpassungen**





## Fazit und Ausblick

### 6.1 Fazit

Nach einer Einführung in die Grundlagen der Datenbank Migration, Guttenbase und die IntelliJ Plugin Entwicklung, wurde eine Anforderungsanalyse durchgeführt. basierend darauf wurde das GuttenBase Plugin entworfen, implementiert und evaluiert.

Mit dem GuttenBase Plugin lassen sich Datenbanken durch wenige Klicks migrieren. Während des Migrationsprozesses können Migrationsoperationen hinzugefügt werden wie das Umbenennen der Quell-Tabellen bzw. Spalten sowie das Ausschließen bestimmter Tabellen bzw. Spalten und das Ändern von Spalten-Datentypen. Diese können vor- oder in dem Migrationsprozess erstellt werden und werden gespeichert, um sie bei der nächsten Migration wiederverwenden zu können. Das GuttenBase Plugin läuft basierend auf der GuttenBase Bibliothek und interagiert mit dem Database Plugin von IntelliJ.

Mit den oben genannten Funktionalitäten erreicht das GuttenBase Plugin das zu Beginn der Bachelorarbeit gesetzte Ziel.

### 6.2 Ausblick



# Literatur

- [Gei14] Frank Geisler. *Datenbanken: Grundlagen und Design*. mitp Verlags GmbH & Co. KG, 2014.
- [WZ15] Sabine Wachter und Thomas Zaelke. *Systemkonsolidierung und Datenmigration als Erfolgsfaktoren: HMD Best Paper Award 2014*. Springer-Verlag, 2015.