DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

Studiengang Informationstechnik

Bericht zum Modul Praxis I

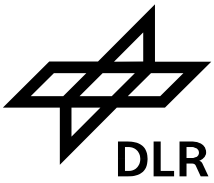
Praxisphase 2: 19. Juni 2017 - 30. September 2017

Thema: Konzeption, Implementierung und Inbetriebnahme einer postgreSQL-Datenbank

von **Beitler, Daniel**

- Matrikelnr.: 9247761 -

- Kurs: TINF16ITIN -



Deutsches Zentrum für
Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft

Standort: Oberpfaffenhofen

Institut für Hochfrequenztechnik und
Radarsysteme
Abteilung: SAR-Technologie

Dr. Rolf Scheiber

Abstract

Dieser Bericht befasst sich mit dem Einrichten und der Inbetriebnahme einer Datenbank. Zu diesem Zweck werden zuerst allgemeine Grundlagen im Bereich der Datenbanken dargelegt. Dabei erfolgt auch eine Erläuterung einiger grundlegender SQL-Befehle, die bei der Arbeit auf einer Datenbank unverzichtbar sind. Anschließend wird ein Pythonprogramm, welches die Arbeit auf der Datenbank übernimmt, entworfen und die Einrichtung einer speziellen Tabelle, die die Log-Daten der Prozessierung flugzeuggestützter Radardaten enthält, beschrieben. Der Schwerpunkt liegt hierbei auf dem Entwurf des Tools und der konkreten Implementierung desselben. Dabei wird die innere Struktur des Programms und deren praktische Umsetzung beschrieben.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich
die vorliegende Arbeit selbstständig und nur unter
Verwendung der angegebenen Quellen und
Hilfsmittel angefertigt habe.

Oberpfaffenhofen, der 22. September 2017

Inhaltsverzeichnis

Abbildungsverzeichnis **V**

Tabellenverzeichnis **VI**

Abkürzungsverzeichnis **VII**

1 Einleitung **1**

2 Konzeption, Implementierung und Inbetriebnahme einer postgreSQL-Datenbank **2**

2.1 Aufgabenstellung 2
2.2 Datenbanken und SQL 3
2.3 Programmstruktur 9
2.4 Implementierung von SQLog 11
2.4.1 Verwaltung von Datenbankverbindungen und Anfragen 14
2.4.2 Methodenbeschreibung 14
2.4.3 Implementierung der graphischen Oberfläche 18
2.5 Konfiguration einer Tabelle für Log-Einträge der flugzeuggestützen SAR-Prozessierung 20

Literaturverzeichnis **IX**

Abbildungsverzeichnis

1 Das Hauptfenster von pgAdmin3 4
2 Strukturierung der Pakete in SQLog 10
3 Aufbau einer Verbindung zu einer Datenbank 11
4 Genereller Ablauf beim Starten einer Operation im Controller 12
5 Ablauf einer Anfrage and die Funktion für spezielle Daten 13
6 Grober Aufbau der graphischen Oberfläche aus Elementen 13
7 Leere simulierte Tabelle 15
8 Tabelle nach dem Ausführen der Funktion `addEntry` 16
9 Tabelle nach dem Ausführen der Funktion `editEntry` 16
10 Tabelle nach dem Ausführen der Funktion `addTime` 17
11 Die graphische Oberfläche des Programms *SQLog* 20
12 Hauptfenster in der Ansicht von *command_history* 23
13 Metadatenansicht in der Tabelle *command_history* 24
14 Ansicht eines Eintrags für *project__textfile* in *command_history* 25

Tabellenverzeichnis

1 Häufig benutzte Datentypen in PostgreSQL 6
2 Operationen für `addSpecialData` 18
3 Spalten in *command_history* 22

Abkürzungsverzeichnis

DLR	**D**eutsches Zentrum für **L**uft- und **R**aumfahrt e.V.
HR	**H**ochfrequenztechnik und **R**adarsysteme
GUI	**G**raphical **U**ser **I**nterface
SAR	**S**ynthetic **A**perture**R**adar
DBFSAR / DBF-SAR	**D**igital **B**eam **F**orming **SAR**
F-SAR	**F**lugzeug-**SAR**
XML	Ex(**X**)tensive **M**arkup **L**anguage
HTML	**H**yper**t**ext **M**odeling **L**anguage
DB	**D**aten**b**ank
DBVS	**D**aten**b**ank**v**erwaltungs**s**ystem
DBS	**D**aten**b**anks**s**ystem
SQL	**S**tructured **Q**uery **L**anguage
psql	**P**ostgre**SQL**
CSV	**C**omma **S**eparated **V**alues

1 Einleitung

Das Deutsche Zentrum für Luft- und Raumfahrt: Die achttausend Angestellten des Deutschen Zentrum für Luft und Raumfahrt (DLR) betätigen sich in der Luft- und Raumfahrtforschung für nationale Projekte der Bundesrepublik Deutschland, stehen aber auch in enger Zusammenarbeit mit der europäischen Luft- und Raumfahrtindustrie, sowie den internationalen Raumfahrtagenturen In diesem Rahmen werden Forschungsprojekte aus den beispielhaften Themenkomplexen des Umweltschutzes, der Astronomie und der Erdbeobachtung realisiert[1].

Aktuell ist der Standort Oberpfaffenhofen mit fast zweitausend Mitarbeitern der größte des DLR. Hier wird unter anderem auf den Gebieten der Robotik, der Erdbeobachtung und der Kommunikation geforscht und entwickelt[2].

Das Institut für Hochfrequenztechnik und Radarsysteme: Das von Prof. Dr.-Ing. habil. Alberto Moreira geleitete Institut für Hochfrequenztechnik und Radarsysteme (HR) befasst sich mit seinen zirka 150 Mitarbeitern mit Projekten zu den Themen Radartechnik und auch Aufklärung und Sicherheit. Dabei liegt der Fokus auf der Entwicklung fortschrittlicher Methoden im Bereich der Radaraufnahme mit synthetischer Apertur (SAR - Synthetic Aperture Radar)[3] [4].

Die Abteilung für SAR Technologie: Innerhalb des Instituts HR werden in der Abteilung SAR Technologie unter der Leitung von Prof. Dr. Andreas Reigber flugzeuggestützte Radarsysteme entwickelt. Dabei wird auch die Ver- und Bearbeitung von Radardaten behandelt und verbessert. Derzeit wird das Flugzeugradar F-SAR verwendet[5].

Um einen reibungslosen Ablauf innerhalb der Forschung im Bereich des Radars mit synthetischer Apertur zu gewährleisten werden Anwendungen zur Verwaltung und Verarbeitung von Daten benötigt. In diesem Bericht wird die Entwicklung und Inbetriebnahme einer Schnittstelle zu einer Datenbank zur Verwaltung von Prozessierungsdurchläufen beschrieben.

2 Konzeption, Implementierung und Inbetriebnahme einer postgreSQL-Datenbank

2.1 Aufgabenstellung

Bei vielen Anwendungen in der Informatik ist es üblich ein sogenanntes Log zu erstellen, welches die durchgeführten Arbeitsschritte protokolliert und dem Nutzer somit auch Informationen liefert, sollte ein Fehler während eines Prozesses aufgetreten sein.

Bei der Prozessierung von SAR-Daten ist eine solche Funktion auch wünschenswert, da so Informationen über den genauen Prozessbefehl, die Anfangs- und die Endzeit des Prozesses und zum Beispiel der ausführende Benutzer gespeichert werden und bei Bedarf betrachtet und für weitere Schritte verwendet werden können.

Für die Prozessierung von flugzeuggestützten SAR-Dateien ist ein seperates Tool, das diese Aufgaben erledigt, nötig geworden. Dieses Programm sollte sehr universell einsetzbar sein, da es verschiedene Prozessarten in der SAR-Prozessierug gibt. Es ist geplant dieses Tool eventuell institutsweit verfügbar zu machen.

Für die Speicherung der Daten wurde eine Datenbank vorgeschlagen, da diese den Vorteil bietet, dass verschiedene Rechner in dasselbe Log schreiben können, was sinnvoll ist, da SAR-Dateien nicht nur auf einem Rechner bearbeitet werden.

Zur Anzeige der Daten sollte außerdem eine graphische Oberfläche gestaltet werden, die es dem Nutzer erlaubt die Daten zu betrachten, in der Datenbank zu suchen, bestimmte überflüssige Einträge auszublenden und zu sortieren.

Das Tool soll auf Kommandozeilenebene die Möglichkeit bieten, sich mit einer Datenbank zu verbinden und diese Verbindung zu schließen. Außerdem soll der Nutzer Einträge hinzufügen und editieren können. Auch wurde eine Möglichkeit zur Speicherung von Zeiten und speziellen Daten, sowie deren Anzeige für sinnvoll erachtet.

Das ganze Tool, mit dem Namen SQLog sollte in Python geschrieben sein.

2.2 Datenbanken und SQL

Allgemeines Als Datenbanken werden logische Einheiten von großen, dauerhaft auf externen Speichern verwalteten Datenbeständen bezeichnet (Definition nach [6, S. 747]). Heinz Peter Gumm und Manfred Sommer führen in ihrer *Einführung in die Informatik*[6] die folgenden Abkürzungen, die auch in diesem Text verwendet werden ein:

> Als Datenbankverwaltungssystem (DBVS) wird die Software zur Verwaltung einer Datenbank bezeichnet. Außerdem wird der Komplex aus Datenbank und Datenbankverwaltungssystem Datenbanksystem (DBS) genannt (nach [6, S. 747]).

Bis in die 1960er Jahre wurden anfallende Daten normalerweise durch die Programme selbständig verwaltet, was zu verschiedenen Komplikationen führte. So waren zum Beispiel nicht alle Datenformate miteinander kompatibel und mussten vom Benutzer oder von speziellen Programmen konvertiert werden. Auch konnte auf die meisten Datenbanken nur ein einziger Nutzer zeitgleich zugreifen, da sie auch oft nicht von außen zugänglich waren. (nach [6, S. 747])

Dieser Umstand machte die Einführung einer eigenen DBVS als Schnittstelle zwischen den Anwenderprogrammen (AWP) und den Datensätzen nötig.

Nach Gumm und Sommer hat diese Lösung mehrere Vorteile, unter anderem müssen so bei Änderungen in der Datenbank (Software und Definitionen) nur noch minimale Änderungen in den Anwenderprogrammen unternommen werden, wenn überhaupt. Auch kann die Datenbank nun so gegliedert werden, dass kein Nutzer einen vollständigen Zugriff auf alle Daten erhält und dadurch nicht wissen muss, wie die Daten organisiert sind. Außerdem kann solch eine DBVS als interaktive Umbegung - wie zum Beispiel Python - dienen und verfügt meistens über die Möglichkeit zeitgleich mehreren (verschiedenen) Benutzern Zugriff zu gewähren. (nach [6, S. 748f.])

Aktuell werden bei relationalen Datenbanken bevorzugt die Sprache SQL (Structured Query Language) und darauf aufbauende Systeme verwendet. Diese ermöglichen es dem Nutzer verschiedene Operationen, wie das Erstellen, Editieren, Auswerten und Löschen von Datensätzen und auch Tabellen (relationale Datenbanken bauen auf Tabellen auf) durchzuführen.

Verwendetes Datenbankverwaltungssystem Für diese Projekt wurde das DBVS PostgreSQL (Internetseite: https://www. postgresql.org/) ausgewählt, da das Institut für Hoch-

frequenztechnik und Radarsysteme des DLR schon über ein aktiv administriertes Datenbanksystem mit dieser SQL-Distribution verfügt.

Laut der PostgreSQL Global Development Group ist PostgreSQL ein Open Source-Projekt, für welches es Schnittstellen in vielen gängigen Programmiersprachen gibt. Auch unterstützt PostgreSQL laut Hersteller große Datenmengen, so soll die maximale Speicherkapazität für eine ganze Datenbank unlimitiert sein. Tabellen, Zeilen und einzelne Felder können Größen von 32 Terrabyte, 1,6 Terrabyte, beziehungsweise einem Gigabyte erreichen. (nach [7])

Verwaltung von PostgreSQL-Datenbanken: pgAdmin3 Zur Verwaltung von PostgreSQL-Datenbanken wird im Institut für Hochfrequenztechnik und Radarsysteme des DLR hauptsächlich das Programm pgAdmin3 [8] verwendet. Dieses Tool ist Open Source und für die gängigen Betriebssysteme verfügbar [8]. Abbildung 1 zeigt das Hauptfenster von pgAdmin3. Im

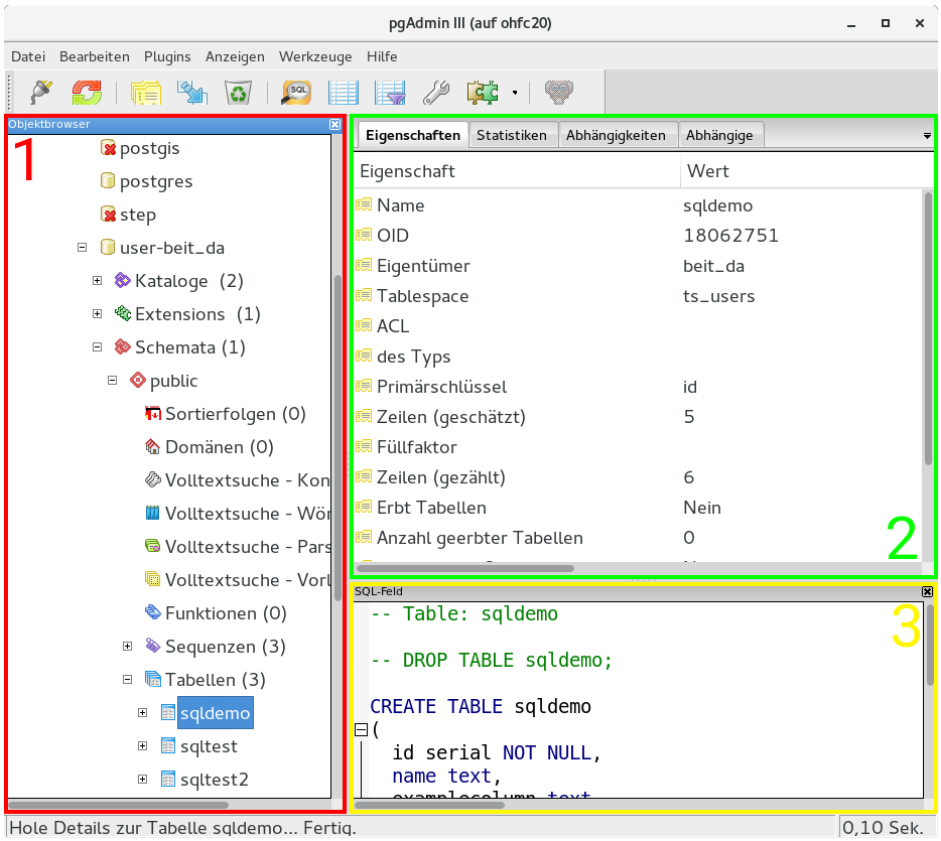


Abbildung 1: Das Hauptfenster von pgAdmin3

sogenannten *Objektbaum* (mit der Ziffer 1 markiert) werden alle Servergruppen (beinhalten verschiedene Serververbindungen) und ihre Unterobjekte, wie zum Beispiel Datenbanken und Tabellen innerhalb dieser Datenbanken als Baumstruktur angezeigt. Details zu Objekten, die im Objektbaum ausgewählt werden, werden im oberen Fenster auf der rechten Seite (in Abbildung 1 mit der Ziffer 2 markiert) angezeigt und bieten dem Nutzer eine erste Übersicht. Das Fenster darunter (3), *SQL-Feld* genannt, zeigt - sofern vorhanden und möglich - eine SQL-Sequenz, die das in (1) ausgewählte Objekt beschreibt.

Um einen neuen Server hinzuzufügen muss der Nutzer im Menüreiter *Datei* den Menüpunkt *Server hinzufügen* auswählen.

Überblick über grundlegende Befehle in SQL und PostgreSQL (auch als psql bezeichnet) [1]

PostgreSQL unterscheidet nicht zwischen Groß- und Kleinbuchstaben in Befehlen und Namen - sofern diese nicht von Anführungszeichen umgeben werden -, was bedeutet, dass die folgenden beiden Sequenzen als äquivalent behandelt werden:

```
1 create table meine_tabelle(name text);
```

```
1 CreAte taBle MeiNe_TABeLLe(NaMe tEXt);
```

Beide Befehle erzeugen eine Tabelle mit dem Namen *„meine_tabelle"*.

Häufig werden daher Befehlssequenzen mittels Großbuchstaben und Variablennamen und Typen mittels Kleinbuchstaben dargestellt:

```
1 CREATE TABLE meine_tabelle(name text);
```

Eine Möglichkeit, Namen mit Großbuchstaben zu verwenden, sind Namen mit Anführungszeichen, auf die hier nicht weiter eingegangen wird. (nach [9])

Notiz: Das Semikolon zeigt dem Interpreter (in diesem Fall psql), dass der Befehl zu Ende ist und ausgeführt werden soll.

Erstellen einer Tabelle In SQL werden neue Tabellen normalerweise mithilfe des Befehls

[1] Die in diesem und den folgenden Paragraphen beschriebenen Sequenzen stellen zu einem großen Teil Elemente der standardmäßig definierten SQL-Sprache dar. Daher können diese Sequenzen auf nahezu jeder SQL-Konsole ausgeführt werden.

```
1 CREATE TABLE name_der_tabelle(erste_spalte typ_erste_spalte,
    weitere_spalte typ_weitere_spalte);
```

erzeugt. PostgreSQL bildet hierbei keine Ausnahme. [10] [6, S. 755f.]

Im Folgenden werden einige häufig genutzte Datentypen tabellarisch aufgelistet:

Typ	Beschreibung
text	Dieser, nicht durch den SQL-Standard festgelegte Datentyp kann variable große Strings mit einer maximalen Größe von einem Gigabyte (limitiert durch die maximale Feldgröße) speichern [7] [11].
smallint	Speichert kleine, ganze Zahlen im Bereich von -32768 bis +32767 (Speichergröße: 2 Bytes) (nach [12, Tabelle 8-2.] [12]).
integer	Speichert ganze Zahlen im Bereich von -2147483648 bis +2147483647 (Speichergröße: 4 Bytes) (nach [12, Tabelle 8-2.] [12]).
longint	Speichert große, ganze Zahlen im Bereich von -9223372036854775808 bis +9223372036854775807 (Speichergröße: 8 Bytes) (nach [12, Tabelle 8-2.] [12]).
timestamp	Dieser Datentyp speichert das Datum und die Zeit im gregorianischen System zwischen 4713 vor Christus und 294276 nach Christus mit einer Genauigkeit von einer Mikrosekunde in 8 Byte. Dieser Typ kann wahlweise mit oder ohne Zeitzone verwendet werden [13].

Tabelle 1: Häufig benutzte Datentypen in PostgreSQL

Einfügen von Datensätzen Die Sprache SQL stellt dem Benutzer die Schlüsselworte *„INSERT INTO"* zum Einfügen von Daten zur Verfügung. PostgreSQL verwendet, da es nur eine Erweiterung von SQL ist, auch diese Sequenz nach dem Muster

```
1 -- Einfuegen aller Werte in eine Tabelle
```

```
2 INSERT INTO tabellenname VALUES (wert1, wert2, wert3);
3
4 -- Einfuegen bestimmter Werte
5 INSERT INTO tabellenname (spaltenname1, spaltenname3) VALUES (wert1,
      wert3);
6
7 -- Einfuegen von Werten in einer anderen Reihenfolge als in der
      Tabellendeklaration vorgegeben
8 INSERT INTO tabellenname (spaltenname3, spaltenname1) VALUES (wert3,
      wert1);
```

aufgebaut. [6, S. 759f.] [14]

Editieren von Datensätzen Einträge in einer SQL-Tabelle können mithilfe der Sequenz

```
1 UPDATE tabellenname SET zu_aendernde_spalte_1=neuer_wert_1,
      zu_aendernde_spalte_2=neuer_wert_2 WHERE
      name_der_spalte_zur_identifikation=vergleichswert;
```

verändert werden.

Die Variable *,,tabellenname"* legt die Tabelle innerhalb der Datenbank, auf der die Operation erfolgen, soll fest, die Sequenz zwischen *,,SET"* und *,,WHERE"* weist die neuen Werte ihrer jeweiligen Spalte, beziehungsweise ihrem jeweiligen Variablennamen zu. Der letzte Teil des Befehls, der nach *,,WHERE"* folgt, wird im Zusammenhang mit der Ausgabe von Daten noch näher behandelt. [6, S. 760] [15]

Löschen von Daten und Tabellen Einträge in einer Datenbank können mithilfe des SQL-Befehls

```
1 -- Loeschen einer (oder mehrerer) bestimmten Zeile (Zeilen)
2 DELETE FROM tabellenname WHERE name_der_spalte_zur_identifikation=
      vergleichswert;
3
4 -- Loeschen des gesammten Inhalts einer Tabelle
5 DELETE FROM tabellenname;
```

entfernt werden.

Die Bezeichnung *,,tabellenname"* ist hierbei wieder die Definition der zu bearbeitenden Tabelle und auf den Abschnitt, der auf *,,WHERE"* folgt, wird, wie zuvor schon erwähnt, im

Verlauf dieses Textes noch weiter eingegangen.[6, S. 760] [16]

Tabellen können mithilfe des *DROP TABLE*-Befehls komplett aus der Datenbank entfernt werden.

```
1 DROP TABLE tabellenname;
```

Die Variable *,,tabellenname"* bestimmt, wie in jeder anderen SQL-Sequenz, auf welche Tabelle diese Operation angewandt werden soll. [17]

Einfache Ausgabe von Daten aus einer Datenbank In einer Datenbank gespeicherte Daten können durch den *SELECT*-Befehl (Anfrage) ausgegeben werden. Dieser Befehl ist nach dem folgendem Schema aufgebaut:

```
1 SELECT liste_der_auszugebenden_spalten FROM tabellenname WHERE
      schluesselsequenz;
```

Die Spalten, für die die Datenbank Werte ausgeben soll, werden durch den Parameter *,,liste_der_auszugebenden_spalten"* beschrieben. Dies ist eine durch einfache Kommas getrennte Aufzählung der verlangten Spalten.

```
1 spaltenname1, spaltenname3
```

Wie zuvor schon definiert *,,tabellenname"* die Tabelle, für die die Operation ausgeführt wird.

Die *,,schluesselsequenz"*, die auf *,,WHERE"* folgt, definiert, welche Einträge (entsprechen den Zeilen) genau ausgegeben werden sollen. Diese Sequenz ist eine Aneinanderkettung von Vergleichen, die mittels logischen Operatoren - ähnlich wie bei *if*-Sequenzen in den meisten Programmiersprachen - miteinander verknüpft werden.

Vergleiche bestehen hierbei aus dem Spaltennamen (Variablenname), einem Vergleichsoperator und aus einem Vergleichswert. Vergleichsoperatoren können hierbei die allgemein bekannten Operatoren ,,größer" ($>$), ,,größer gleich" ($>=$), ,,kleiner" ($<$), ,,kleiner gleich" ($<=$), ,,ist gleich" ($=$) und ,,ist nicht gleich" ($!=$ oder $<>$) sein. Es ist aber auch möglich nach sogenannten Wildcards (unbekannte Zeichen innerhalb eines Worts werden durch _ (**ein** Zeichen) oder % (**mehrere Zeichen**) dargestellt) zu suchen. Dafür gibt es die Operatoren *LIKE* und *SIMILAR TO*. Für dieses Projekt wird aber nur *LIKE* benötigt.

Logische Operatoren, die die Vergleiche verbinden sind hierbei *OR* und *AND*, wobei *AND* eine größere Priorität hat als *OR*. Sequenzen können zudem durch Klammern gruppiert werden, um präzisere Aussagen zu treffen.

```
1 WHERE name!='Daniel' OR alter<50 AND geschlecht='m'
```

bedeutet zum Beispiel, dass alle Datensätze aufgelistet werden, bei denen der Wert in der Spalte,,name" nicht dem Wert 'Daniel' entspricht **oder** alle Datensätze, bei denen der Wert in ,,alter" kleiner als 50 und der Wert in ,,geschlecht" dem Buchstaben 'm' entspricht. Im Gegensatz dazu gibt die Sequenz

```
1 WHERE (name!='Daniel' OR alter<50) AND geschlecht='m'
```

alle Datensätze zurück, die einen Buchstaben 'm' in der Variablen ,,geschlecht" haben **und** bei denen der Wert in ,,name" nicht der Wert 'Daniel' oder der Wert in ,,alter" kleiner als 50 ist (beides kann natürlich zutreffen, da es sich bei diesem *OR* nicht um ein *XOR* handelt). [6, S. 757f.] [18] [19] [20] [21]

2.3 Programmstruktur

Um eine Trennung der einzelnen Aufgaben in diesem Tool zu erreichen, und um den Nutzern eine bessere Übersicht und Flexibilität zur Verfügung zu stellen, baut SQLog auf einem objektorientierten Programmieransatz auf. Aus dieser Überlegung resultiert die Trennung in Datenbankverbindung, spezielle Werkzeuge und Oberfläche, die in Abbildung 2 dargestellt ist.

In dieser Struktur übernimmt das Paket *DBAdm* das Lesen und Schreiben in der Datenbank, *GUI* stellt die Nutzeroberfläche zur Verfügung und Module in *SpecialTools* ermöglichen es dem Programm, spezielle Funktionen, die auch der Nutzer definieren kann, auszuführen. Dabei werden Anfragen des Anwenders durch das Modul *Controller*, welches in dem Hauptordner *SQLog* liegt, verwaltet.

Da Datenbankverbindungen so lange geöffnet bleiben, bis sie geschlossen werden, ist, um den Nutzer eine bessere Übersichtlichkeit zu bieten, das Öffnen mehrerer Verbindungen möglich. Diese werden in einem Array zwischengespeichert. Damit eine Unterscheidung zwischen den Verbindungen möglich ist, werden Datenbankverbindungen immer als Tupel mit einer Liste von Zugriffsdaten (Nutzername, Hostname, Datenbankname und Tabellenname) zusammen gespeichert. Um dem Nutzer eine Angabe der Zugangsdaten zu ermöglichen, werden diese schon beim Eröffnen einer Tabelle zurückgegeben (Abbildung 3). Die erhaltenen Daten können zwischengespeichert und später beim Ausführen einer Operation auf der Datenbank

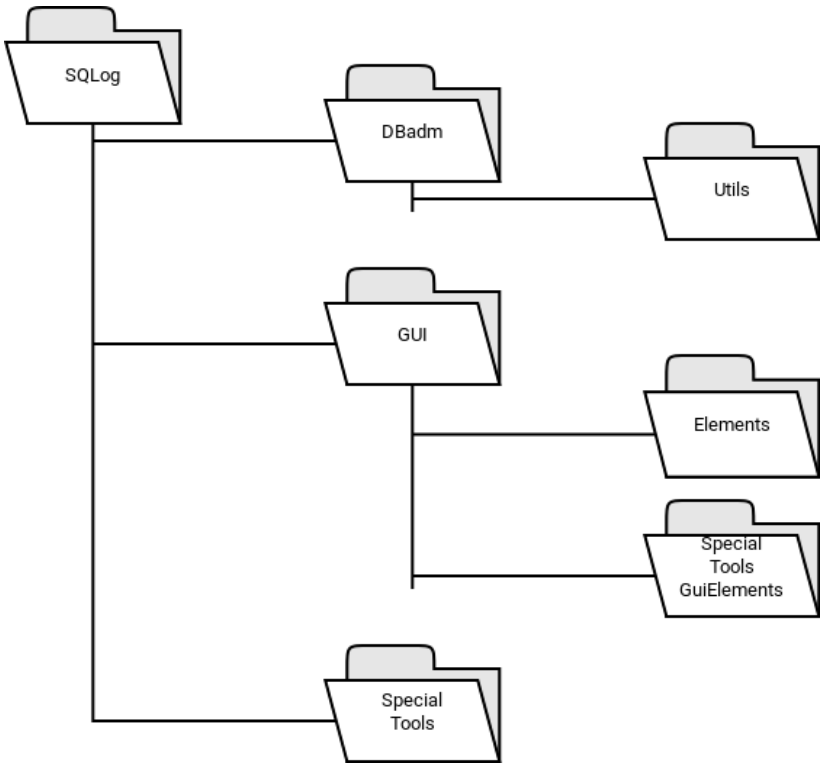


Abbildung 2: Strukturierung der Pakete in SQLog

angegeben werden. Abbildung 4 zeigt den generellen Ablauf einer solchen Operation.

Um spezielle Daten, die nicht über eine Edit-Funktion eingefügt werden können, hinzuzufügen, ist es möglich, eine Operation aufzurufen, bei der anhand eines übergebenen Strings eine Bestimmung der auszuführenden Schritte erfolgen kann. Der eigentliche Algorithmus dieser speziellen Funktionen wird dabei separat abgelegt und die Methode in *Controller* übernimmt hierbei nur eine unterscheidende Funktion. Aufgrund der geforderten Modularität in diesem Bereich ist dies die aktuell vorteilhafteste Lösung, da auf diese Weise eine ähnliche Bestimmung auch in der graphischen Oberfläche erfolgen kann. Dabei muss aber beachtet werden , dass in dem entsprechendem Modul für die Anwendung in der GUI ein eigenes Fenster definiert wird. Hierhin wird automatisch ein Knopf an der Stelle dieser speziellen Daten generiert , um eine allzu große Ausdehnung der entsprechenden Spalte zu vermeiden. Abbildung 5 zeigt den Ablauf einer solchen Operation, wobei zu beachten ist, dass das Operieren auf eine Datenbank nur innerhalb des Moduls *Controller* erfolgen darf. Daten für diese Operation können über

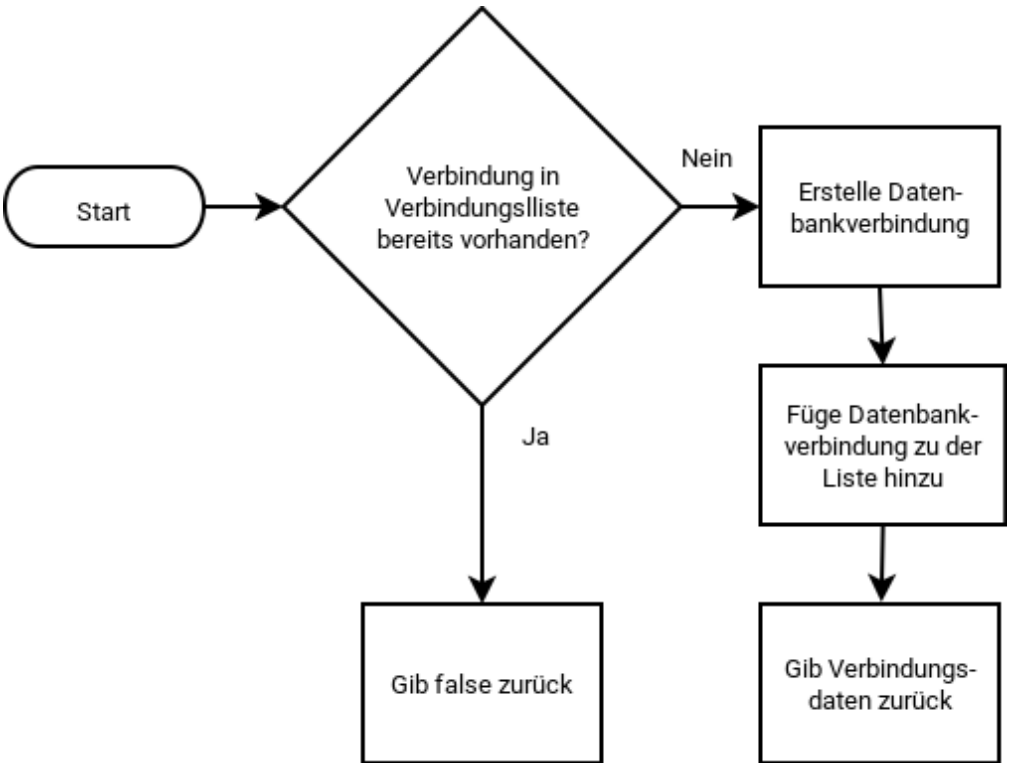


Abbildung 3: Aufbau einer Verbindung zu einer Datenbank

die externe Funktion erzeugt werden.

Abbildung 6 zeigt den groben Aufbau der graphischen Oberfläche, wobei nahezu alle gezeigten Elemente zusätzlich noch aus den Grundelementen des GUI-Pakets bestehen, sofern sie nicht schon welche sind (in diesem Fall der Knopf). Als Paket für den Aufbau der Oberfläche wurde zunächst *TkInter* und später *PyQt5* gewählt, da *TkInter* im Gegensatz zu *PyQt5* keine Möglichkeiten zum Verwalten von Tabs und Knöpfen innerhalb einer vorgefertigten Tabelle (beziehungsweise Baumstruktur) liefert.

2.4 Implementierung von SQLog

Bei der Implementierung der zuvor genannten Ideen wurde schnell klar, dass ein anfangs gewählter Ansatz nicht zufriedenstellend war und somit eine komplette Überarbeitung des

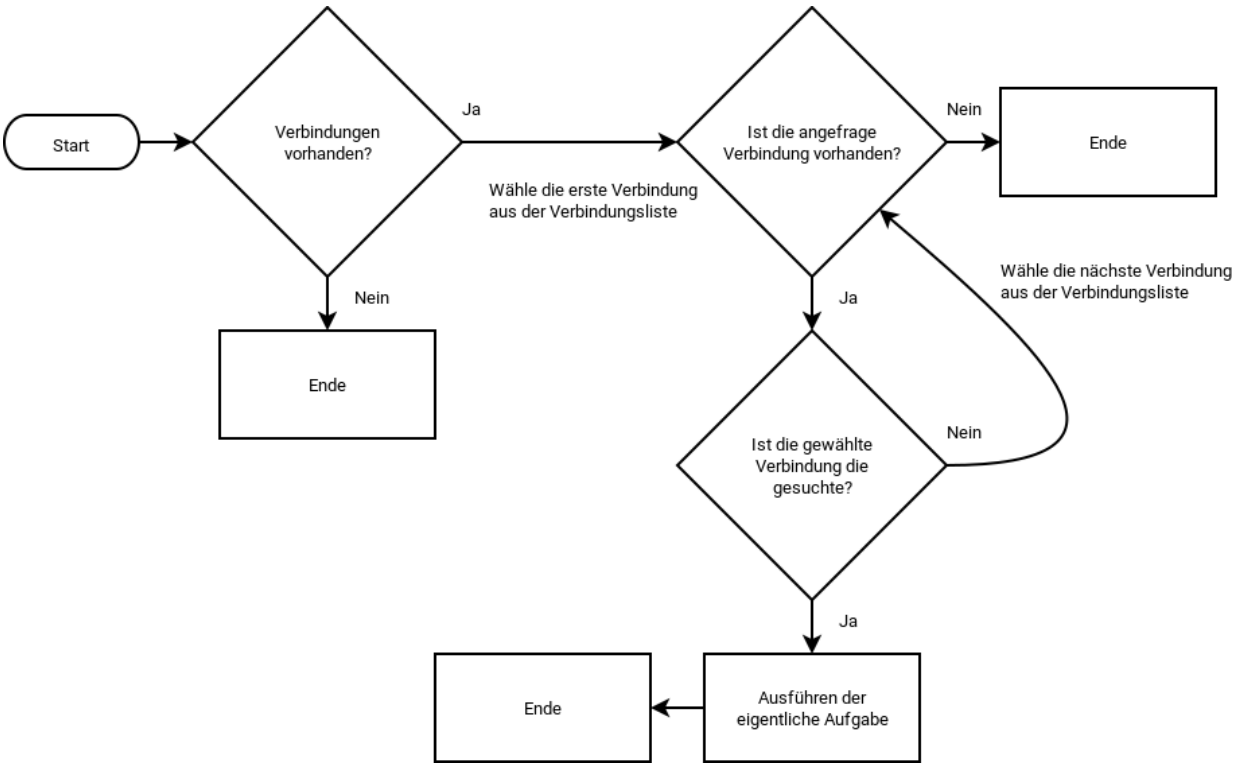


Abbildung 4: Genereller Ablauf beim Starten einer Operation im Controller

Tools erforderte, bei der unter anderem die Möglichkeit zum Öffnen mehrerer Verbindungen und die Einbeziehung von *PyQt5* realisiert wurde.

Die Implementierung des schon bereits erwähnten Arrays zum Ablegen der Verbindungen im Modul *Controller* erfolgte mithilfe einer globalen Variable, die eine Liste von Tupeln enthält. Der Inhalt dieser Tupel ist, wie zuvor schon erwähnt, die Verbindung zur Datenbank und eine bestimmende Liste.

```
1 # Definition der Variable
2 connections_list = []
3
4 # Deklaration der globalen Variable in jeder Funktion, um sie nutzen zu
      koennen
5 global connections_list
```

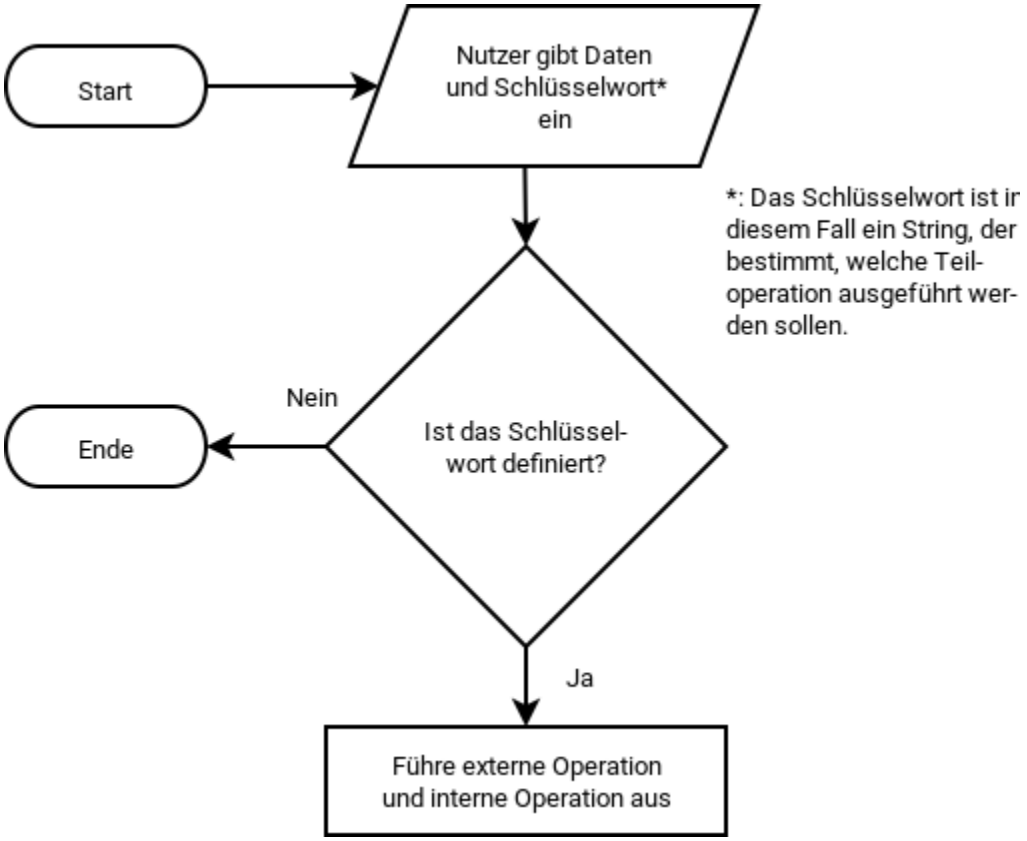


Abbildung 5: Ablauf einer Anfrage and die Funktion für spezielle Daten

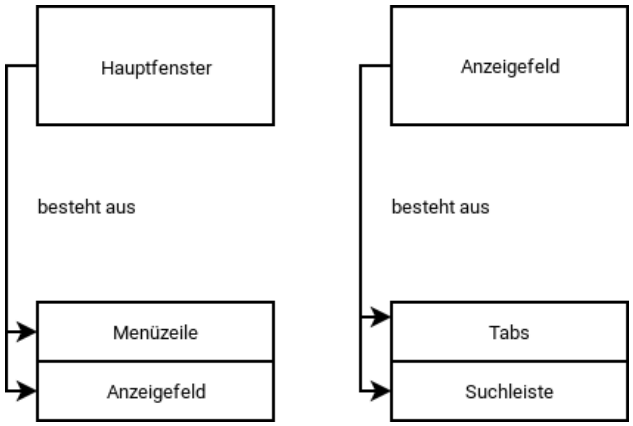


Abbildung 6: Grober Aufbau der graphischen Oberfläche aus Elementen

2.4.1 Verwaltung von Datenbankverbindungen und Anfragen

Nachdem die globale Variable als eine solche innerhalb einer Funktion in *Controller* definiert wurde, werden Algorithmen meistens innerhalb einer Struktur platziert, die überprüft, ob die gewünschte Verbindung vorhanden ist.

```
1 # Algorithm for selecting the requested conection
2 if len(connections_list) > 0:
3     for el in connections_list:
4             if el[0] == define_vals:
5
6               # Codeblock
```

Dazu muss erwähnt werden, dass die Definition der Verbindung vor dem eigentlichen Objekt platziert wird. Dies erfolgt nach dem folgenden Schema:

```
1 [[[Liste, mit, vier, Definitionswerten], Verbindungsobjekt], Weitere
      Verbindungen]
```

Die Bearbeitung von Anfragen an die Datenbank erfolgt innerhalb der Module *DBReader*, beziehungsweise *DBWriter* innerhalb des Paketes *Utils* in *DBAdm*, wobei *DBConnect* lediglich die Verbindung zur Datenbank aufbaut, diese Verbindung speichert und alle Anfragen an die Module zum Schreiben und Lesen weiterleitet, wobei jedes Mal auch der Tabellenname, sowie das Verbindungsobjekt übergeben wird.

Die Zugriffe auf die Datenbank erfolgen mithilfe des PostGreSQL-Pakets *psycopg2*. Dieses Paket stellt Funktionen zum Ausführen einer SQL-Anweisung und zum Auslesen der Rückgabe zur Verfügung.

```
1 # SQL-Anweisung
2 cursor.execute('SELECT DISTINCT %s FROM %s ORDER BY %s ASC'
3                 %(names[i], table, names[i]))
4
5 # Auslesen der Rueckgabe
6 col_hlp = cursor.fetchall()
```

2.4.2 Methodenbeschreibung

Jede der im Folgenden beschriebenen Methoden verfügt über eine Struktur mit dem Namen *defin_vals*. In dieser Struktur wird die Definition der Tabelle, die beim Eröffnen einer Daten-

bankverbindung zurückgegeben wird, eingefügt, um die Tabelle, auf der gearbeitet wird zu definieren (→ 2.4.1, S. 14). Auch wird jedes mal die Identifikationsnummer der veränderten Zeile zurückgegeben.[2]

Für die folgenden Beispiele wird die Tabelle *command_history* simuliert, die sich an ein reales Beispiel anlehnt, in Abbildung 7 wird die leere Tabelle gezeigt.

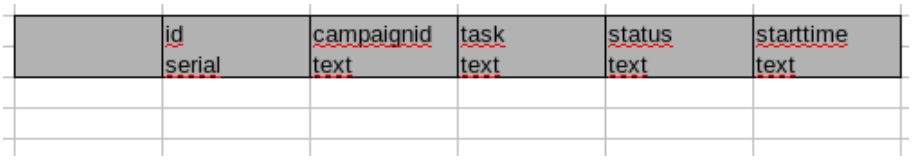


Abbildung 7: Leere simulierte Tabelle

addEntry Die Methode

`addEntry(define_vals, **kwargs)`

in *Controller* ermöglicht dem Anwender, einen neuen Datensatz (eine Zeile) zu einer angegebenen Tabelle hinzuzufügen. Die Eingabe erfolgt hier über ***kwargs* (**k***ey***w***ord* **arg***uments*), das sind Schlüsselwortparameter mit einer variablen Länge. Das bedeutet, dass in ***kwargs* Schlüsselwörter verwendet werden, um Werte zu übergeben. In `addEntry` werden zu verändernde Spalten durch die "Schlüsselwörter" und die einzufügenden Werte durch die assoziierten Werte dargestellt.

```
1 define_vals = ["f_step_sql", "hr-svl005.intra.dlr.de", "step", "
    command_history"]
2 identifier = addEntry(define_vals, campaignid="17OP17AF", task="update")
3
4 print(identifier)
5 -> 1
```

Dieser Methodenaufruf fügt eine neue Zeile in die Tabelle *command_history* auf der Datenbank *step* für den Nutzer *f_step_sql* auf dem Server *hr-svl005.intra.dlr.de* ein. Dabei wird der Wert der Spalte *campaignid* auf "17OP17AF" und der Wert der Spalte *task* auf "update" gesetzt. Daraus ergibt sich die Abbildung 8.

[2] Die folgenden Auszüge zeigen lediglich das Schema, folglich wird auf Zusätze, wie Importe und Modulbezeichner, verzichtet.

	id serial	campaignid text	task text	status text	starttime text
1	1	17OP17AF	update		

Abbildung 8: Tabelle nach dem Ausführen der Funktion `addEntry`

editEntry Um Datensätze zu Editieren wird von SQLog die Methode

`editEntry(define_vals, where, **kwargs)`

zur Verfügung gestellt. Hierbei funktionieren *define_vals* und ***kwargs* wie schon zuvor beschrieben. Neu ist hierbei der Parameter *where*. Hier werden die zu verändernden Einträge definiert. SQLog verlangt, dass *where* eine Liste von Strings ist. Diese Strings werden intern mit `AND` verknüpft und als SQL-Bedingung verwendet. Jeder String besteht aus zwei Teilen, die durch einen Vergleichsoperator ($=$, $>$, $<$, usw.) verbunden sind: Links steht die Spalte, mit der verglichen werden soll, rechts der Vergleichswert, folglich sieht solch ein String folgendermaßen aus: "`spalte = 'Vergleichsstring'`". Die Sequenz

```
1 editEntry(define_vals, ["id = 1"], status="running")
```

fügt in alle Datensätze, bei denen die Spalte *id* den Wert 1 hat, den String "running" in die Spalte *status* ein. Dadurch ergibt sich Abbildung 9.

	id serial	campaignid text	task text	status text	starttime text
1	1	17OP17AF	update	running	

Abbildung 9: Tabelle nach dem Ausführen der Funktion `editEntry`

addTime Einfacher zu handhaben ist die Funktion

`addTime(define_vals, identifier, column_list)`

. Hier ist lediglich zu beachten, dass die Identifikationsnummer der zu bearbeitenden Zeile bekannt ist. Diese Nummer wird für den Parameter *identifier* benötigt, um den zu bearbeitenden Datensatz zu bestimmen. Auch muss beachtet werden, dass *column_list* als eine Liste von

Strings interpretiert wird. Dort werden alle Spalten bestimmt, in denen die aktuelle UTC-Zeit eingefügt werden soll. Diese wird von dem Programm SQLog aus der aktuellen Systemzeit über `datetime.utcnow()`. Folglich verändert die Sequenz

```
1 addTime(define_vals, identifier, ["starttime"])
```

die Tabelle dahingehend, dass die Zeit hinzugefügt wird (→ Abbildung 10). Dabei werden bei jedem Schreiben in der Datenbank zuerst die Datentypen für die zu editierenden Spalten abgefragt. Auf der Basis dieser Typen werden dann die Eingabewerte entsprechend konvertiert.

	id serial	campaignid text	task text	status text	starttime text
1	1	17OP17AF	update	running	2017-09-07 09:43:55.81059

Abbildung 10: Tabelle nach dem Ausführen der Funktion `addTime`

addSpecialData Die letzte Funktion zum Bearbeiten von Daten ist `addSpecialData(define_vals, identifier, column_name, operation_string, [path='/', comp_cols=[]])`. Diese Methode kann spezielle Daten abfragen, die dann auch von spezialisierten Funktionen angezeigt werden können. Hierbei werden *define_vals* und *identifier* wie zuvor verwendet, auch *column_name* als String für den Spaltenname funktioniert ähnlich wie *column_list*, mit dem Unterschied, dass diesmal eine einzelne Spalte definiert wird[3]. Der Parameter *operation_string* bestimmt hierbei, welche Funktionalität ausgeführt wird. Alle Parameter innerhalb der eckigen Klammern (in diesem Fall nur *path* und *comp_cols*[4]) sind vorbesetzt und optional. Ob sie benötigt werden hängt von *operation_string* ab. Tabelle 2 listet die aktuell[5] vorhandenen Operationen auf, und beschreibt ihre zugehörigen Parameter, und die zu nutzenden Datentyperweiterungen[6].

[3] Es können auch mehrere Spalten definiert werden: "Spalte1, Spalte2". Dies ist auch in der Liste so möglich, wird aber nicht speziell behandelt
[4] Stand: 22. September 2017
[5] Stand: 19. September 2017
[6] Als Datentyp bezeichnet, werden durch zwei Unterstriche vom Spaltennamen getrennt und ermöglichen es SQLog diese Spalten gesondert zu behandeln. Beispiel: auxdata__dirmeta

Operation	Parameter	Datentyp	Anmerkungen
directory.getMeta	path	dirmeta	Liest Metadaten aus einem angegebenen Pfad und Unterobjekten (Unterpfade, Links).
file.saveText	path	textfile	Speichert den Inhalt einer angegebenen Datei als Text.
date.setDiff	comp_cols	Kein spezieller Datentyp	Berechnet die Differenz zwischen zwei Zeiten

Tabelle 2: Operationen für `addSpecialData`

Daraus lässt sich der Befehl

```
addSpecialData(define_vals, identifier, 'auxdata__dirmeta', 'directory.
    getMeta', path='/bin')
```

bilden. Dabei werden der Inhalt und die Metadaten des Ordners */bin* in der Spalte *auxdata__dirmeta* in der Zeile mit der Identifikationsnummer *identifier* gespeichert.

2.4.3 Implementierung der graphischen Oberfläche

Um Daten, die in einer Datenbank abgelegt sind, anzuzeigen, ist in vielen Fällen eine graphische Oberfläche eine für den Nutzer komfortable Darstellungsweise.

In diesem Fall besteht die GUI aus einem Hauptfenster (*GUIMain*), welches die Menüzeile und Funktionen zum Anzeigen einer neuen Datenbank enthält, und den Klassen zum Anzeigen der Tabellen innerhalb dieser Datenbanken.

Hauptfenster Wie zuvor schon erwähnt ist die Klasse *GUIMain* die Hauptklasse der Oberfläche. Sie erweitert die Klasse *QMainWindow* aus dem Paket *QtWidgets*. Hier erfolgt die Erstellung der Menüleiste und des Elements, welches die Tabs mit den Inhalten verwaltet. Auch wird hier schon unter Zuhilfenahme einer Erweiterung von *QDialog* ein Hinzufügen von Datenbankverbindungen möglich.

Verwaltung der Tabs Um mehrere Tabellen darstellen zu können (eine Datenbank in diesem Sinne enthält Tabellen) wird auf das Prinzip der Tabs zurückgegriffen, die in der

Informatik eine breite Anwendung finden, sei es als Tabs in einem Internetbrowser oder als Tabs (verschiedene Tabellen) in einem Tabellenkalkulationsprogramm. Solche Elemente bestehen meist aus Objekten verschiedener Klassen. So wird fast immer ein Objekt, das die Verwaltung der Tabs übernimmt, benötigt. Der Inhalt dieser Tabs wird durch andere Objekte dargestellt, die durchaus auch wieder Tabs enthalten können. In diesem Fall erfolgt die Verwaltung über die, das Modul *QTabWidget* erweiternde Klasse, *GUITabManager*. Der Inhalt wird durch *GUITab*, einer Erweiterung von *QWidget*, dargestellt.

Bei der Erstellung eines Objekts der Klasse *GUITabManager* wird überprüft, ob schon Verbindungen zu einer oder mehreren Datenbanken vorhanden sind. Sollte dies der Fall sein, so werden schon beim Öffnen der graphischen Oberfläche einige Seiten mit Inhalt erstellt.

```
1  # Check, if the controller has any connections
2  if SQLog.getStatus():
3
4          # If so, get the data for the connections
5          data_list = SQLog.returnAccessData()
6
7          # Add tabs
8          for el in data_list:
9              tab = GuiTb.GUITab([el[0], el[1], el[2], el[3]])
10
11             self.addTab(tab , '%s@%s, %s/%s'%(el[0], el[1], el[2], el[3])
                   )
```

Die Klasse *GUITab* erstellt bei ihrem Aufruf ein Widget. Dieses beinhaltet eine Zeile mit deren Hilfe man innerhalb einer Tabelle suchen kann. Darunter wird die Tabelle mit dem eigentlichem Inhalt dargestellt. Dies erfolgt durch eine Baumansicht, die von *PyQt5* bereitgestellt wird und die Möglichkeit bietet, innerhalb einer Spalte zu sortieren. Die Suchzeile wird durch ein Element, welches ein Areal mit Scrollbalken und Elementen für die Auswahl der Option enthält, und einen Knopf zum Suchen gebildet.

Aus diesen Teilen setzt sich dann schließlich die Ansicht in Abbildung 11 zusammen.

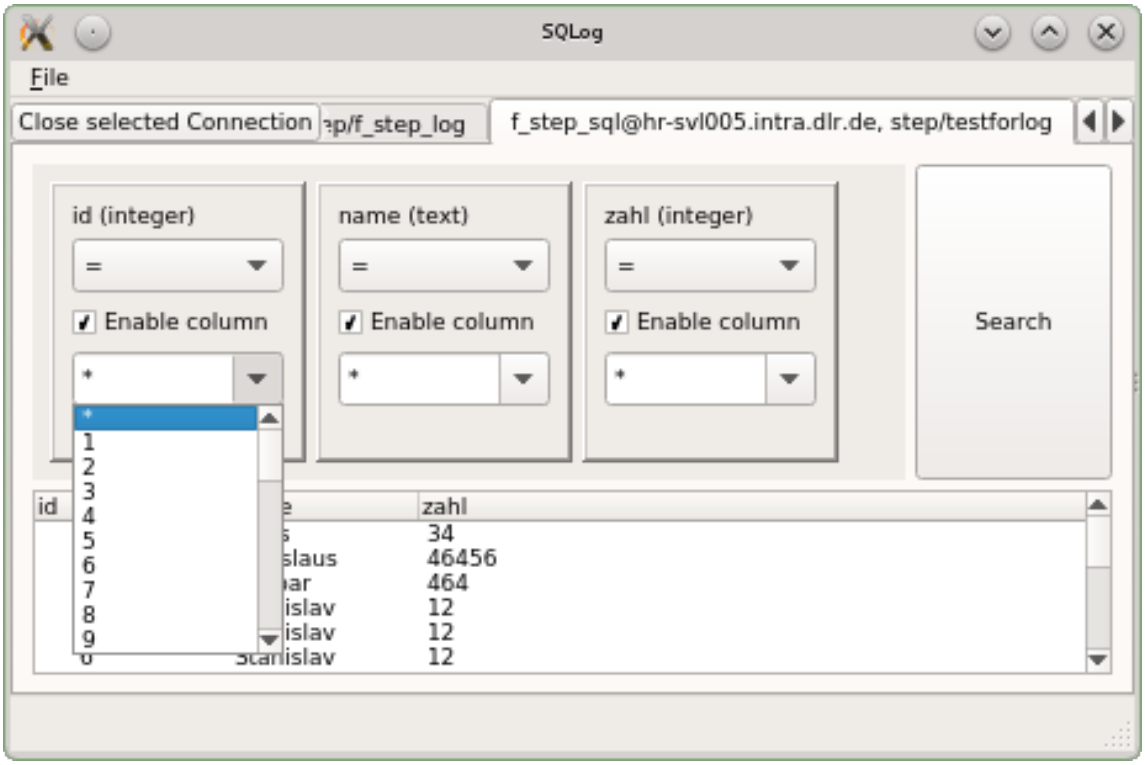


Abbildung 11: Die graphische Oberfläche des Programms *SQLog*

2.5 Konfiguration einer Tabelle für Log-Einträge der flugzeuggestützen SAR-Prozessierung

Um das Tool *SQLog* zur Anwendung zu bringen, wurde es in einen Testprozess für die Prozessierung von Radardaten integriert. Dafür steht auf dem Datenbankserver `hr-slv005.intra.dlr.de` (Stand: 19. September 2017, die Datenbank wird verschoben werden) der funktionale Nutzer *f_step_sql* zur Verfügung. Für diesen Nutzer wurde auf der Datenbank *step* die Tabelle *command_history* mit dem Befehl

```
1 CREATE TABLE command_history
2 (
3   id serial NOT NULL,
4   campaignid text,
5   flightid text,
6   passid text,
```

```
7   tryid text,
8   processor text,
9   task text,
10  projectfile text,
11  status text,
12  starttime text,
13  stoptime text,
14  totaltime text,
15  username text,
16  hostname text,
17  command text,
18  auxdata__dirmeta text,
19  project__textfile text
20 )
```

, der auf jeder beliebigen Konsole für PostgreSQL, in diesem Fall in *pgAdmin3*, ausgeführt werden kann, angelegt. Auf die Bedeutung der einzelnen Spalten wird in Tabelle 3 weiter eingegangen.

Name	Typ	Anmerkungen
id	serial NOT NULL	Die *id* ist eine Identifikationsnummer innerhalb der Tabelle, um zu bearbeitende Datensätze (Zeilen) eindeutig identifizieren zu können.
campaignid	text	Die *campaignid* ist eine Zeichenfolge, die die zugehörige Mission festlegt.
flightid	text	Die *flightid* beschreibt, welche Nummer der Flug innerhalb der Mission hat.
passid	text	Die *passid* beinhaltet die Nummer des Passes (oder Überflugs) innerhalb des Flugs.
tryid	text	Kennung des Prozessierungsversuchs.
processor	text	Prozessbeschreibung (Kurzform).
task	text	Auszuführende Aufgabe.
projectfile	text	Projektdatei zur Konfiguration des Prozessors.
status	text	Aktueller Status des Prozess (running/completed/error).

Name	Typ	Anmerkungen
starttime	text	Die *starttime* wird am Anfang eines Prozesses gesetzt und beschreibt die Startzeit.[7]
stoptime	text	Die Endzeit eines Prozesses wird in *stoptime* beschrieben.[8]
totaltime	text	Die Differenz zwischen *stoptime* und *starttime* steht in *totaltime*.
username	text	Benutzer.
hostname	text	Verwendeter Server/Rechnername.
command	text	Der Wert in *command* ist eine Kopie der Kommandozeile und beschreibt den Aufruf des Prozesses.
auxdata__dirmeta	text	In *auxdata* wird eine Ordnerstruktur als Text gespeichert (erkennbar durch *dirmeta*).
project__textfile	text	In *project* kann der Inhalt einer Textdatei als Text abgelegt werden. Durch *textfile* wird für SQLog ein spezieller Datentyp definiert.

Tabelle 3: Spalten in *command_history*

Bei der Ansicht der Tabelle mit der GUI von SQLog ergibt sich dann das in Abbildung 12 gezeigte Bild. Durch das Doppelklicken auf eine entsprechend beschriftete Zelle ("Doubleclick here") öffnen sich die in Abbildung 13 und Abbildung 14 dargestellten Fenster. Dabei ist Abbildung 13 ein Eintrag für die Spalte *auxdata__dirmeta* aus *command_history*. Abbildung 14 gibt einen Eintrag für die Spalte *project__textfile* wider, der zur Konfiguration eines "FUSAR"-Prozessors genutzt wurde.

[7] Die Zeit wird durch SQLog eingefügt (addTime)
[8] Die Zeit wird durch SQLog eingefügt (addTime)

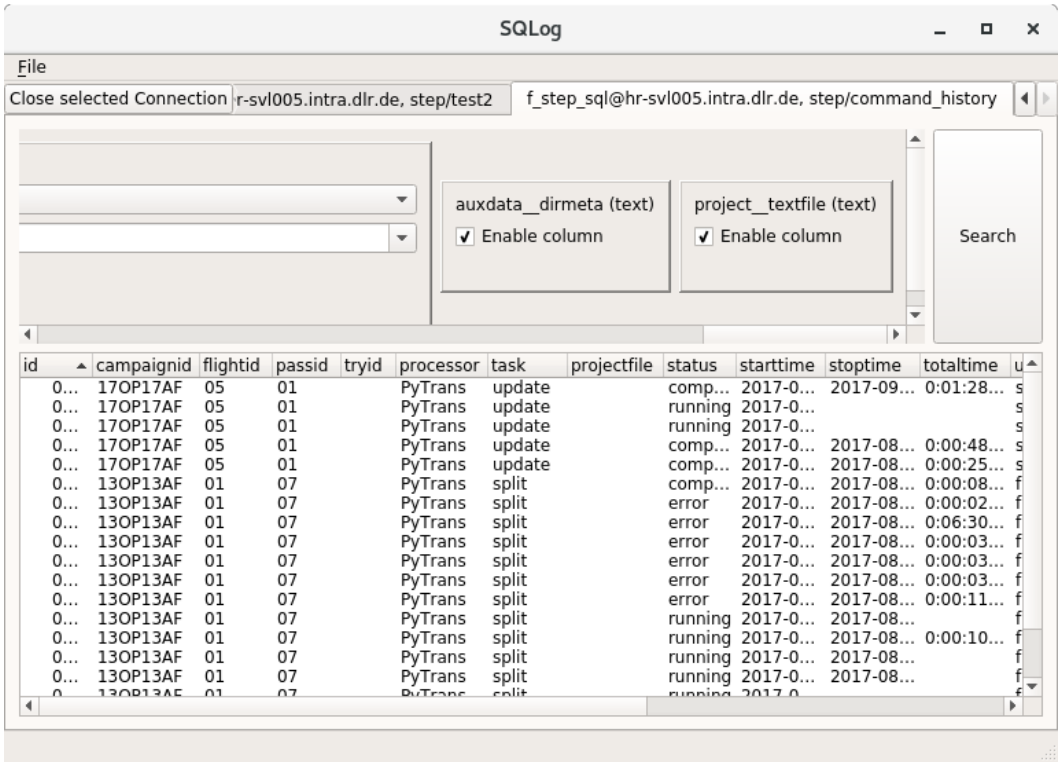


Abbildung 12: Hauptfenster in der Ansicht von *command_history*

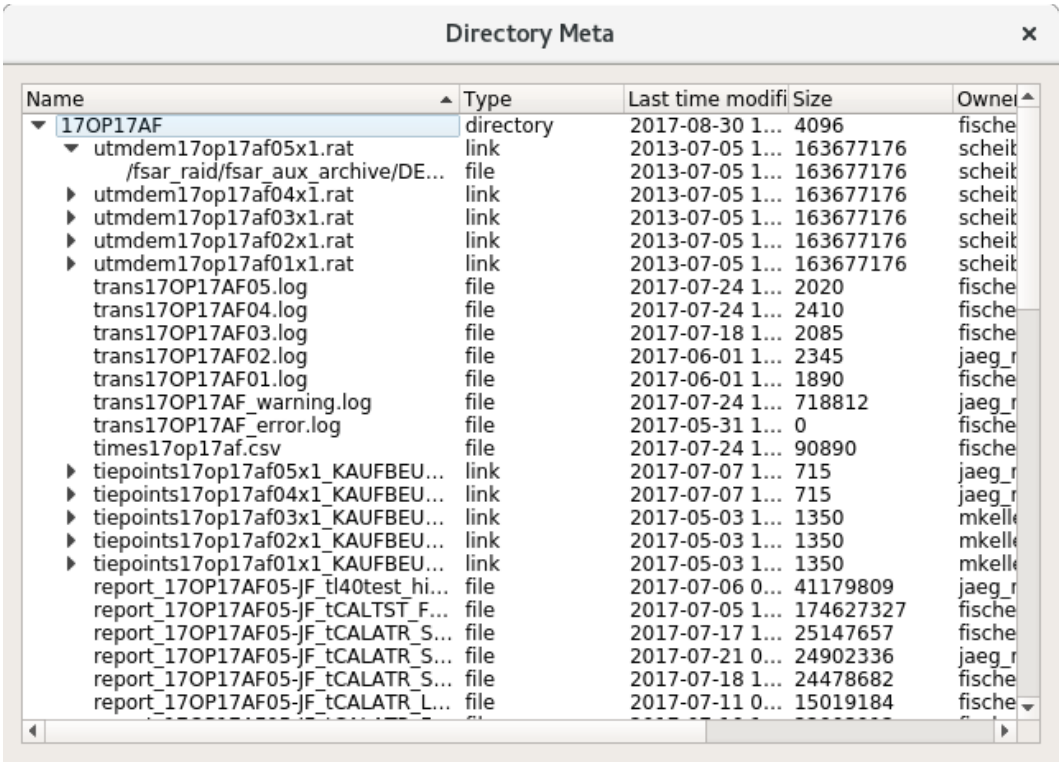


Abbildung 13: Metadatenansicht in der Tabelle *command_history*

Abbildung 14: Ansicht eines Eintrags für *project__textfile* in *command_history*

Literaturverzeichnis

[1] Deutsches Zentrum für Luft- und Raumfahrt e.V.: *Das DLR im Überblick*. Version: 2017. http://www.dlr.de/dlr/de/desktopdefault.aspx/tabid-10443/637_read-251/#/gallery/8570. – zuletzt aufgerufen: 12.09.2017

[2] Deutsches Zentrum für Luft- und Raumfahrt e.V.: *Der Standort Oberpfaffenhofen des DLR*. Version: 2017. http://www.dlr.de/dlr/de/desktopdefault.aspx/tabid-10261/320_read-214#/gallery/701. – zuletzt aufgerufen: 13.09.2017

[3] Deutsches Zentrum für Luft- und Raumfahrt e.V.: *Institut für Hochfrequenztechnik und Radarsysteme*. Version: 2017. http://www.dlr.de/hr/desktopdefault.aspx/tabid-2304/3442_read-39287/. – zuletzt aufgerufen: 13.09.2017

[4] Deutsches Zentrum für Luft- und Raumfahrt e.V.: *Institut für Hochfrequenztechnik und Radarsysteme - Organisation*. Version: 2017. http://www.dlr.de/hr/desktopdefault.aspx/tabid-2312/3439_read-39331. – zuletzt aufgerufen: 13.09.2017

[5] Deutsches Zentrum für Luft- und Raumfahrt e.V.: *Institut für Hochfrequenztechnik und Radarsysteme - Abteilung SAR Technologie (Prof. Dr. Andreas Reigber)*. Version: 2017. http://www.dlr.de/hr/desktopdefault.aspx/tabid-2326/3776_read-44251. – zuletzt aufgerufen: 13.09.2017

[6] Gumm, H.P. ; Sommer, M.: *Einführung in die Informatik*. Oldenbourg, 2006 https://books.google.de/books?id=xYNET5Prj6oC. – ISBN 9783486581157

[7] PGD Group and others: *Postgresql*. Version: 2017. http://www.postgresql.org/about/. – zuletzt aufgerufen: 21.08.2017

[8] The pgAdmin Development Team: *pgAdmin*. Version: 2017. https://www.pgadmin.org/. – zuletzt aufgerufen: 23.08.2017

[9] PGD Group and others: *Postgresql*. Version: 2017. https://www.postgresql.org/docs/9.6/static/sql-syntax-lexical.html. – zuletzt aufgerufen: 22.08.2017

[10] PGD Group and others: *Postgresql*. Version: 2017. https://www.postgresql.org/docs/9.6/static/tutorial-table.html. – zuletzt aufgerufen: 22.08.2017

[11] PGD Group and others: *Postgresql*. Version: 2017. https://www.postgresql.

org/docs/9.6/static/datatype-character.html. – zuletzt aufgerufen: 22.08.2017

[12] PGD Group and others: *Postgresql*. Version: 2017. https://www.postgresql.org/docs/9.6/static/datatype-numeric.html. – zuletzt aufgerufen: 22.08.2017

[13] PGD Group and others: *Postgresql*. Version: 2017. https://www.postgresql.org/docs/9.6/static/datatype-datetime.html. – zuletzt aufgerufen: 22.08.2017

[14] PGD Group and others: *Postgresql*. Version: 2017. https://www.postgresql.org/docs/9.6/static/tutorial-populate.html. – zuletzt aufgerufen: 22.08.2017

[15] PGD Group and others: *Postgresql*. Version: 2017. https://www.postgresql.org/docs/9.6/static/tutorial-update.html. – zuletzt aufgerufen: 22.08.2017

[16] PGD Group and others: *Postgresql*. Version: 2017. https://www.postgresql.org/docs/9.6/static/tutorial-delete.html. – zuletzt aufgerufen: 22.08.2017

[17] PGD Group and others: *Postgresql*. Version: 2017. https://www.postgresql.org/docs/9.6/static/sql-droptable.html. – zuletzt aufgerufen: 22.08.2017

[18] PGD Group and others: *Postgresql*. Version: 2017. https://www.postgresql.org/docs/9.6/static/tutorial-select.html. – zuletzt aufgerufen: 23.08.2017

[19] PGD Group and others: *Postgresql*. Version: 2017. https://www.postgresql.org/docs/9.6/static/queries-table-expressions.html. – zuletzt aufgerufen: 23.08.2017

[20] PGD Group and others: *Postgresql*. Version: 2017. https://www.postgresql.org/docs/9.6/static/functions-matching.html. – zuletzt aufgerufen: 23.08.2017

[21] Inc, Stack E.: *Stackoverflow*. Version: 2017. https://stackoverflow.com/questions/16122695/are-brackets-in-the-where-clause-standard-sql. – zuletzt aufgerufen: 23.08.2017, Antwort durch den Nutzer ,,PaulProgrammer"