Universal Frontend

1.Voraussetzung:

A. Sytem

MySql-Server

Qt-Designer

optional Office-Writer oder MS-Word

B. Persönlich

Grundkenntnisse in Datenbank-Design :

- 1. - 3. Normalform
- Primär- u.Fremdschlüssel

Kenntnisse in Formular-Design:

- Widget-Klassen(LineEdit, ComboBox, CheckBox etc.)
- Widget-Eigenschaften
- Ereignis-Behandlung

2.Einleitung

Haben Sie schon einmal versucht eine Datenbank mit den üblichen Office-Produkten(MS-Access, Libre-Office Base etc.) zu erstellen, und waren Sie über das Ergebnis enttäuscht ? Sie können zwar Daten ein- und ausgeben, aber es mangelt doch an dem richtigen „Pepp“ um wirklich professionell damit zu arbeiten. Unter anderem mangelt es an einer Unterstützung für Benutzerrechte; Datenbankabfragen sind nicht sehr flexibel; der Standardfilter ist quasi unbrauchbar, da er nur „exact-Match“ kann; das automatische Formularlayout ist ein Witz; es ist nur ein Unterformular erlaubt; Metadaten wie Table-Table oder Foreign-Keys sind nicht oder sehr umständlich zugreifbar(nicht als Abfrage!).

Wer diesen „Pepp“ in seiner Datenbank-Anwendung haben will, hat dann keine andere Wahl als mit der integrierten Makro-Sprache(Basic, Java-Script o.ä.) den notwendigen Programm-Code zu ergänzen. Das multipliziert aber schnell den Zeitaufwand für die Entwicklung um dem Faktor 10 oder mehr. Dadurch ist es nicht mehr möglich für einen kleinen (Einzel-)Unternehmer eine individuelle Datenbank zu erstellen, wenn er evtl. nur sagen wir mal 1000 € bezahlen kann.
Ziel dieser Software ist u.A. genau dies zu ermöglichen.

Aber auch größere Unternehmen können von dieser Software profitieren. Da alle Formulare über die Datenbank verwaltet werden, ist es möglich Änderungen im Datenbankmodell vorzunehmen,

alle daraus resultierende Änderungen in den Formularen zu ermitteln und diese dann anzupassen. Dadurch kann die Datenbank mit dem Unternehmen mitwachsen; organisatorische Änderungen im Unternehmen können leicht in die Datenbank übernommen werden. Es ist dann nicht mehr nötig, von Zeit zu Zeit neue Software zu kaufen.

3.Vorteile gegenüber Office-Produkten:

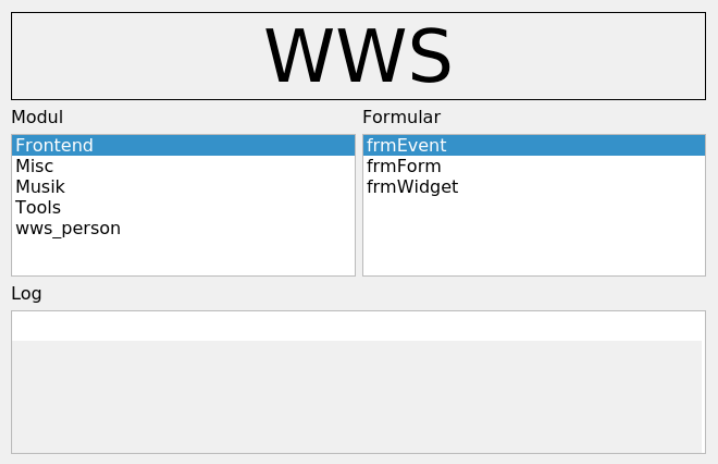
- Ausgefeilter Standardfilter
- beliebig viele Unterformulare
- ansprechendes Autolayout, generisches Formular
- flexibles Design von Formularen und Toolbars
- Formulare, Widget-Eigenschaften und Events transparent über Datenbank-Tabellen gesteuert
- unterschiedliche Arbeitsmodi der Formulare für Anzeigen, Ändern und Hinzufügen von Datensätzen
- Unterstützung von User-Rechten
- Unterstützung von Änderungen im Tabellenmodell

4.Installation

1. Installieren Sie sich den MySql-Server
2. Führen Sie die Skripte Frontend.sql und Frontend_user.sql aus.
   Frontend.sql erzeugt das Datenbankschema "Frontend" mit allen notwendigen Tabellen, die es zu seiner Verwaltung braucht. Frontend_user.sql erzeugt den User "Frontend" der nur Lese-Berechtigung auf dem Schema "Frontend" hat.
   Um dieses Handbuch durcharbeiten zu können, führen Sie noch die Skripte Demo.sql und wws_person.sql aus. Diese legen gleichnamige Schemata mit einigen Übungs-Tabellen an.
3. Installieren Sie sich den QT5-Mysql-Treiber (Paket libqt5sql5-mysql). Dieser ist nicht im Lieferumfang enthalten.
4. Führen sie das Skript install_modules.sh aus. Dieses installiert alle notwendigen Module mittels apt-get.
   Falls Sie das nicht möchten oder können, laden Sie sich das Paket "libraries.tar.gz" von der WebSite BerndsDatenbanken.de herunter. Dieses enthält alle notwendigen shared-libraries und einen generischen App-Starter. Kopieren Sie die Anwendung Frontend und Frontend.conf in den bin/ - Ordner. Die Anwendung wird dann über das Skript Frontend.sh gestartet.

5.Start

A. Das Startformular

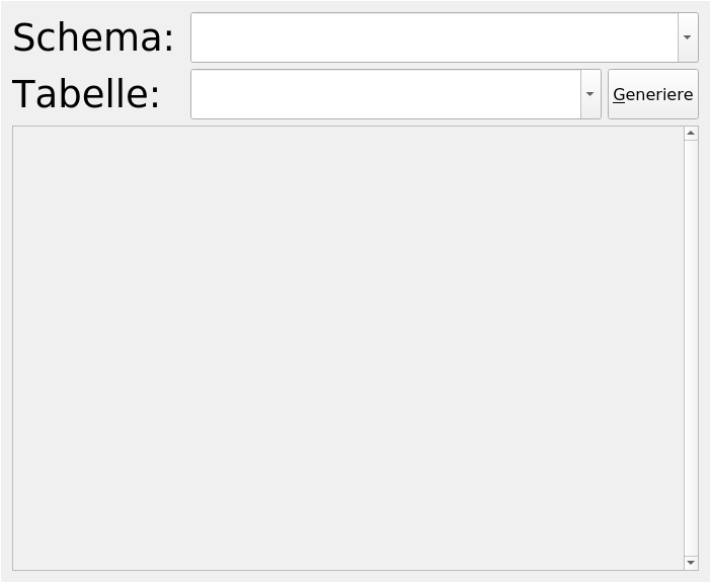


Das Startformular ist einfach aufgebaut. Es enthält die Listboxen Modul und Formular. Klicken Sie auf ein Modul in der linken Listbox „Modul“ , dann erscheint auf der rechten Seite in der Listbox „Formular“ die diesem Modul zugeordneten Formulare. Klicken Sie ein Formular an dann startet das Formular.

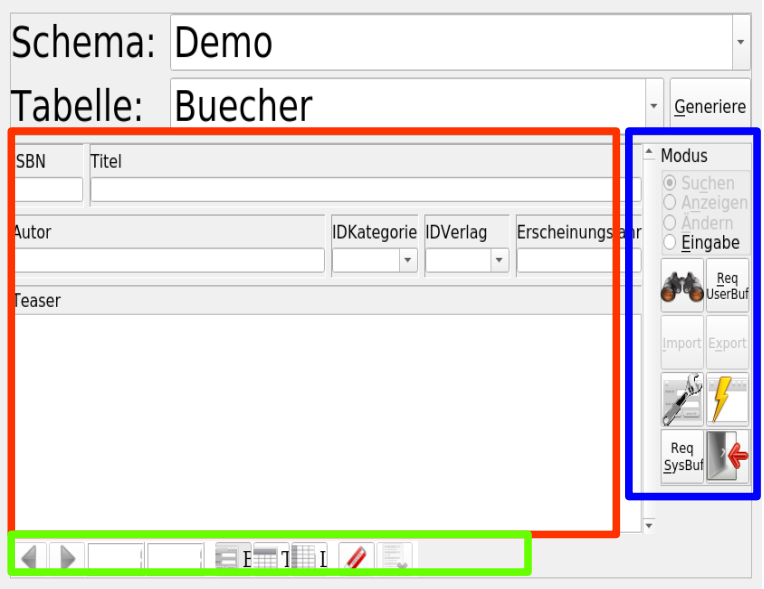
Darüber hinaus enthält das StartFormular noch ein Textfeld „Log“ für Fehler-Nachrichten und Status-Informationen.

B. Einfacher Weg: Generisches Formular

Falls Sie keine besonderen Ansprüche haben und nur relativ schlichte Tabellen bearbeiten wollen, dann legen Sie doch einfach los mit dem generischen Formular. Sie finden es im Modul *Misc* unter dem Namen *frmGeneric*.



Wählen Sie in den Comboboxen Schema und Tabelle die gewünschte Tabelle eines Datenbankschemas aus, die Sie bearbeiten wollen. Nachdem Sie den Button *generiere* betätigen, wird ein Standard-Formular für diese Tabelle erzeugt.



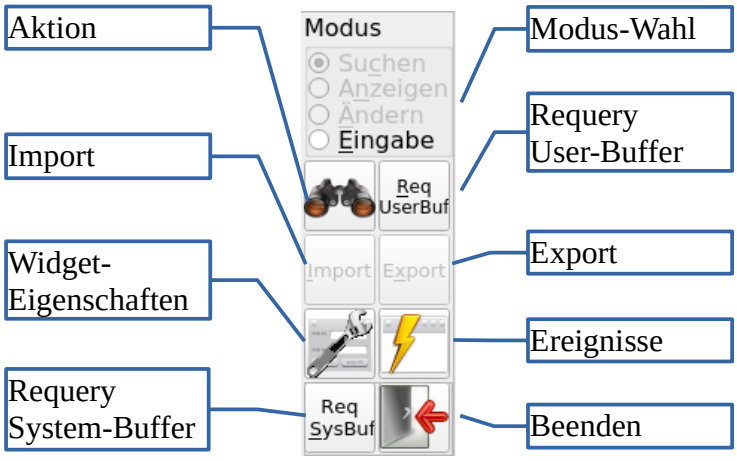
Das Standard-Formular besteht aus 3 Bereichen:

- Rot: Eingabefelder
- Blau: Sidebar
- Grün: Navigator

I. Eingabefelder:

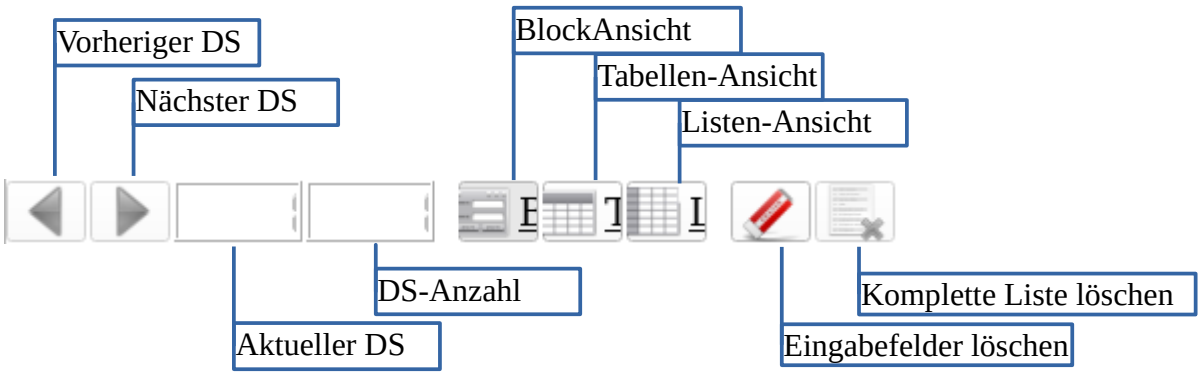
Es werden drei verschiedene Arten von Widgets erzeugt. Standard ist ein **QLineEdit**-Widget . Ist das Tabellenfeld Bestandteil eines Foreign-Keys, wird eine **QCombobox** erzeugt, mit den Werten des Zielfeldes. Ist die Textlänge des Datenbankfeldes sehr groß, wird ein mehrzeiliges **QPlainTextEdit**-Widget generiert.

II. Sidebar:



Sie ermöglicht die Auswahl des Arbeitsmodus, sowie einige Modusabhängige Standard-Aktionen.
Beim Start befindet sich das Formular im Modus ***Suchen*** .
Der 1. Button unterhalb der Modusauswahl, ist der ***Aktionsbutton*** für den gewählten Modus.
Im Modus *Suchen* startet er die Suche, in den Modi Ändern und Eingabe werden die Daten an den DB-Server geschickt.
Mit dem 2.Button können den ***User-Buffer aktualisieren***. Dies benötigen Sie vor allem wenn sich in der Tabelle, auf die die Liste einer ComboBox verweist, Einträge geändert haben.
Mit den Button darunter (***Import/Export)*** können sie die Datensätze im in eine lokale Textdatei speichern bzw. aus lokaler Datei einlesen. ***Export*** dient als Schnittstelle für das erstellen von Dokumenten, wie z.B. Rechnungen. Mit ***Import*** können Sie größere Datenmengen einlesen, die erst noch vorbereitet werden müssen.
Darunter befinden sich zwei Button um die Formulare frmWidget und frmEvent zu starten, mit denen Sie die ***Widget-Eigenschaften*** und ***Ereignisse*** für die Widgets des aktuellen Formulars bearbeiten können.
Darunter sind noch die Button ***RequerySystem-Buffer*** um die System-Tabellen und Views aus dem Schema *Frontend* neu abzufragen(Auch diese werden lokal gepuffert) und der Button ***Beenden*** um das Formular zu verlassen.
Im Augenblick interessiert uns nur der ***Aktionsbutton***. Auf die weiteren Button gehen wir in späteren Kapiteln ein.

III. Navigator



Die ersten vier Widgets(vorheriger/nächster/ aktueller Datensatz und Anzahl) verstehen sich von selbst.
Mit den nächsten 3 Button können Sie die verschiedenen Ansichtsarten ein- und ausschalten. Was sie in der obigen Abbildung sehen ist die ***Blockansicht***. In der ***Tabellenansicht*** sehen Sie alle Datensätze in der gewohnten tabellarischen Anordnung. In der ***Listenansicht*** ist nur der aktuelle Datensatz zu sehen. Hier sind alle Felder als Liste untereinander angeordnet.

IV. Die 4 verschieden Arbeitsmodi

a. Suchen

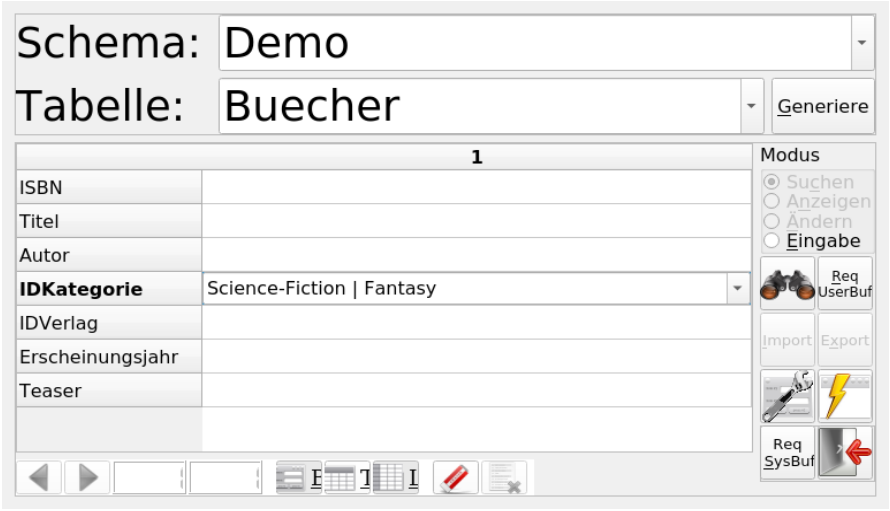
Hier können Sie in allen Eingabefeldern, geeignete Werte zum Filtern der Abfrage eingeben.
In jedem Feld können Sie Operatoren zur Suchabfrage eingeben. Dies sind die Standard-Vergleichsoperatoren <,<=,>,>=
außerdem != Für ungleich
Bereichsoperator […;...] -> eckige Klammern ,die zwei durch Semikolon getrennte Werte enthalten.
Pattern-Match ~ (Tilde) ->für Textsuche mit Jokerzeichen.
Logik-Operatoren &, | für mehrere Angaben in einen Feld.

Alle Operatoren können auch auf Text angewandt werden.

Beispiele für die Tabelle *Demo.Buecher*:

1. Sie wählen in der Combobox **IDKategorie** den Eintrag ***Fantasy*** aus. Nachdem Sie den Aktionsbutton betätigt haben, springt der Modus automatisch auf Anzeigen um und Sie erhalten die Einträge für alle Bücher aus dieser Kategorie(„Rincewind, der Zauberer“, „Nebel von Avalon“ ...).
2. Sie geben im Feld **Erscheinungsjahr** ***>2000*** ein . Sie erhalten alle Bücher, die nach dem Jahr 2000 erschienen sind .
3. Eingabe ***~%Asimov%*** im Feld **Autor** ergibt alle Bücher von Isaac Asimov.

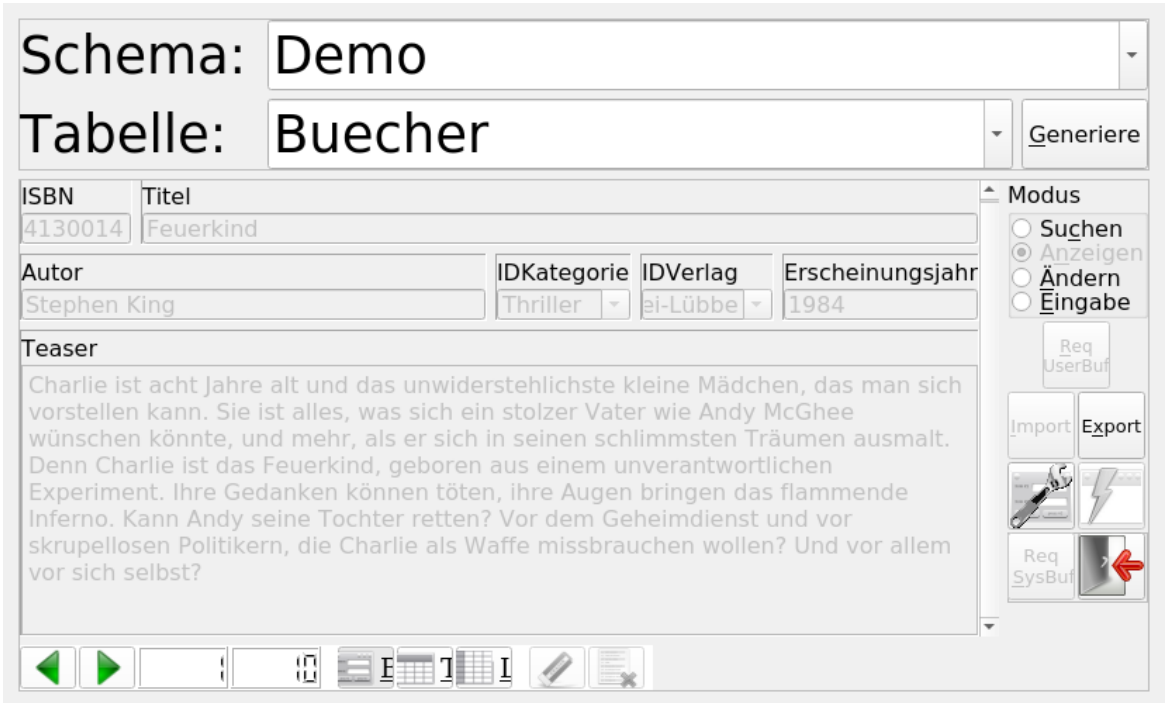
4. Eingabe ***Science-Fiction | Fantasy*** im Feld IDKategorie ergibt alle Bücher dieser beiden Kategorien.
   Falls Ihnen die Eingabefelder in der Blockansicht zu klein sind, wechseln Sie doch einfach zur Listenansicht



5. Wenn Sie ***alle Eingabefelder leer*** lassen, erhalten Sie alle Datensätze aus dieser Tabelle.

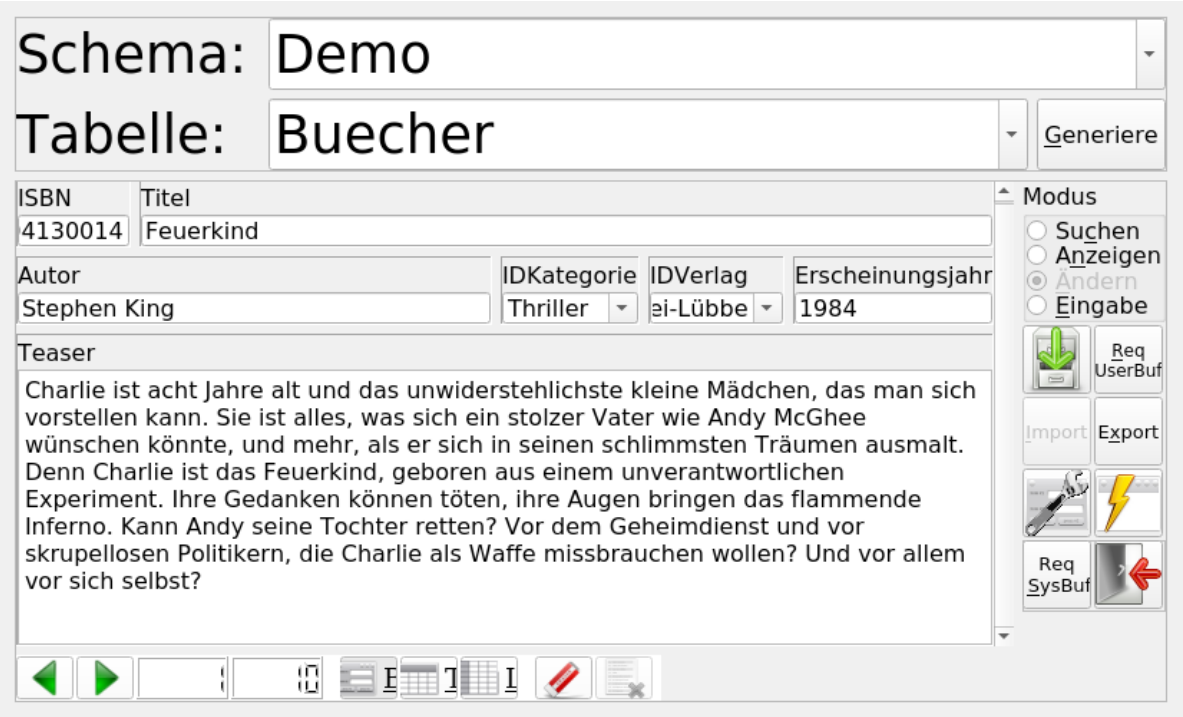
b. Der Modus Anzeigen

Nachdem Sie die Aktion Suchen ausgelöst haben, wechselt das Formular automatisch in den Modus ***Anzeigen***. In der Navigationsleiste sind jetzt die Button Vorheriger und Nächster Datensatz aktiv. Sie können Sie benutzen um sich alle Datensätze anzusehen. Die Eingabefelder sind deaktiviert. Der Aktionsbutton wurde unsichtbar gemacht, da in diesem Modus keine Aktion ausgeführt werden kann.



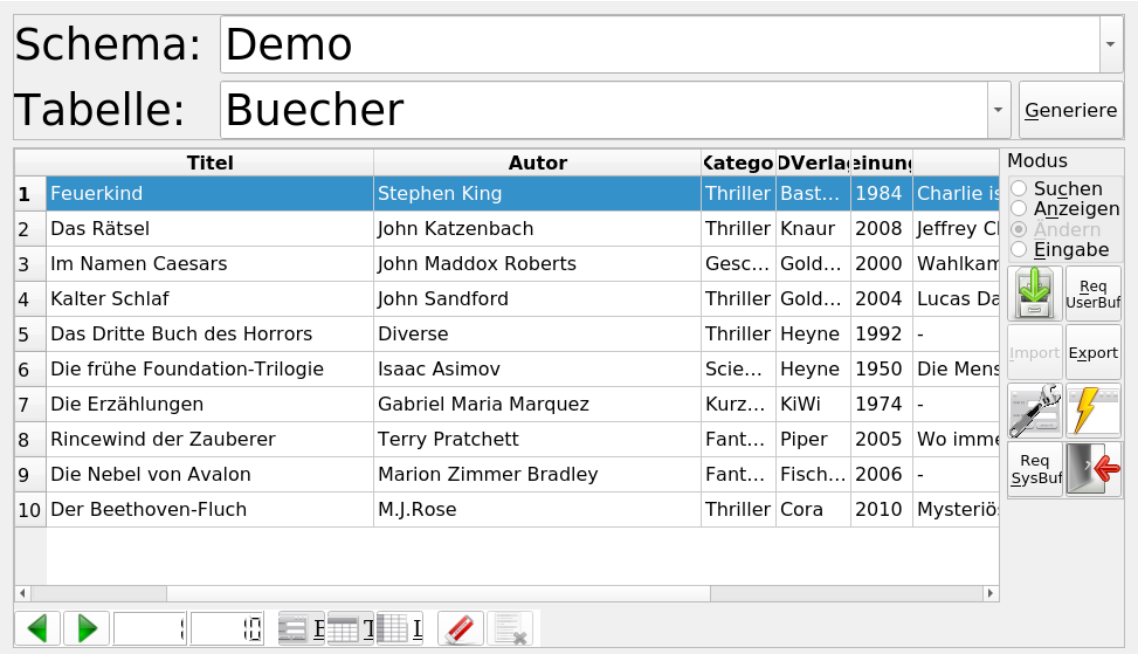
c. Der Modus Ändern

Wenn Sie die ausgewählten Datensätze bearbeiten wollen, wählen Sie den Modus ***Ändern***. Nun sind alle Felder aktiv, und der Aktionsbutton ist wieder sichtbar, mit einem Icon, das ***Daten Speichern*** andeuten soll.



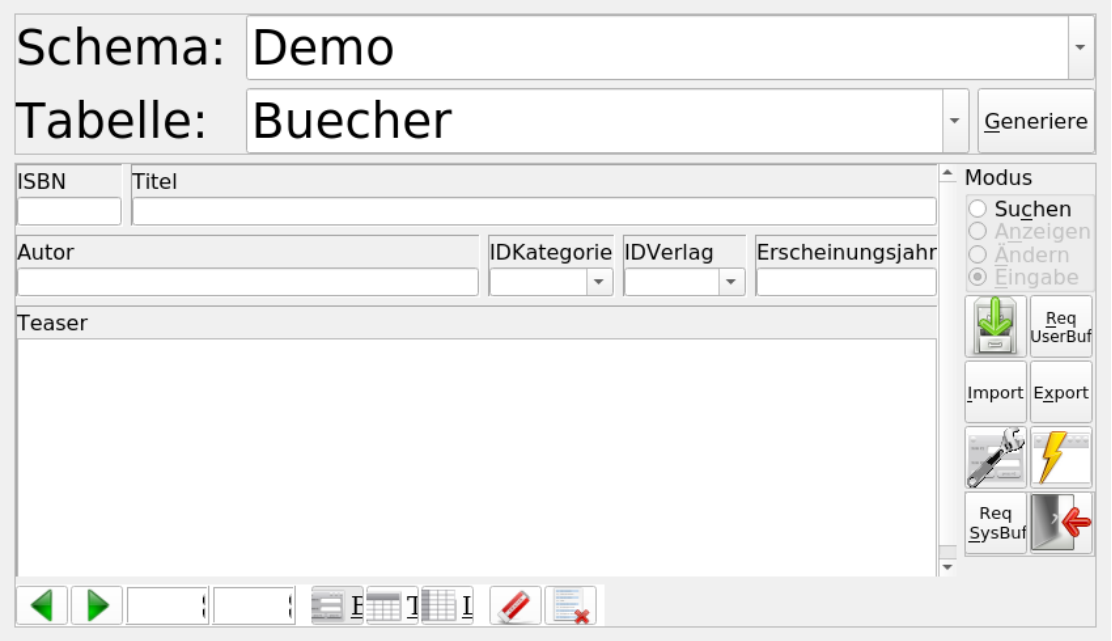
Navigieren Sie durch die Datensätze und machen Sie alle gewünschten Änderungen. Sie können problemlos beliebig viele Datensätze bearbeiten. Anschließend betätigen Sie den Aktionsbutton. Alle Änderungen werden gespeichert. Sie können danach weiterhin Datensätze ändern, falls Sie das möchten. Sie können also, wenn Sie viele Datensätze zu bearbeiten haben, zwischendurch beliebig oft speichern.

Falls sie beispielsweise nur eine Spalte für viele Datensätze bearbeiten wollen, wäre evtl. die Tabellenansicht vorteilhafter.



d. Der Modus Hinzufügen

Wenn Sie neue Datensätze eingeben möchten, wählen Sie den Modus hinzufügen. Auch hier können Sie beliebig viele Datensätze eingeben und dann speichern.



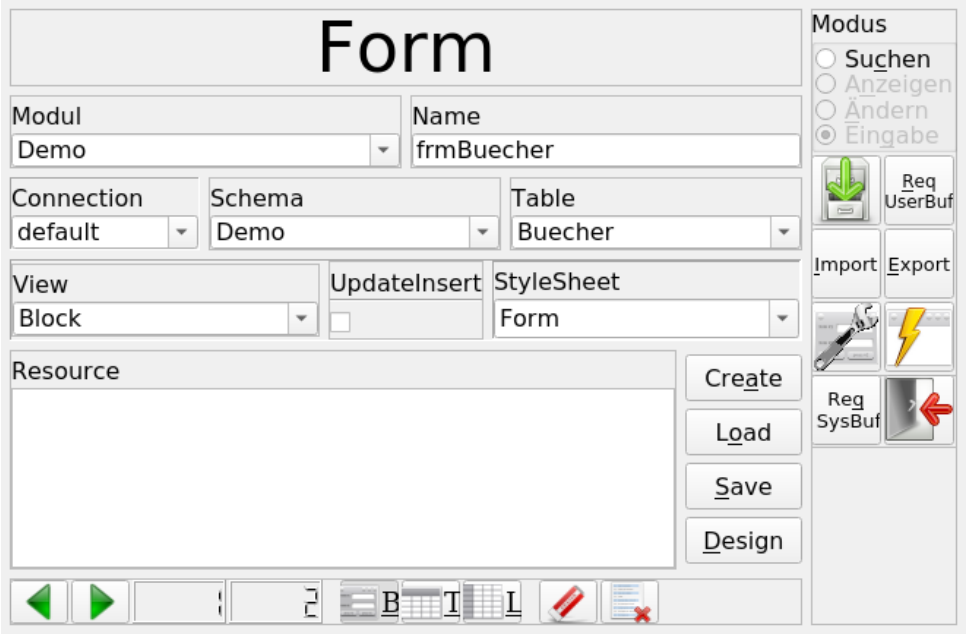
6.Formular-Design

Das Generische Formular ist zu beschränkt um Dinge zu tun die ein wenig anspruchsvoller sind. Diese sind:

- andere Widgets mit Datenbankfeldern verknüpfen wie z.B. CheckBox, ListBox, Time-/DateEdit, Slider etc.
- Widget-Eigenschaften einstellen z.B. Eingabeprüfung, Fehlermeldung, Datenquelle bei Container-Widgets (Checkbox, ListBox, TreeWidget)
- Das Verhalten des Formular abhängig von Eingabewerten steuern.
- Zusätzliche Widgets für Steuerung und Information einfügen

A. Formular Form Autolayout, Designer, lokal Speichern und laden

Auch hier fangen wir wieder möglichst einfach an.
Öffnen Sie da Formular Frontend.Form. Hier sieht man bereits ein elaborierteres Formular. Es enthält 4 zusätzlichen Aktionsbutton. Außerdem enthält es eine Ereignisverarbeitung. Jedesmal, wenn Sie den Wert in der ComboBox **Schema** abändern, wird die Liste in der ComboBox **Table** aktualisiert, d.h. es werden nur die Tabellen angezeigt, die in dem gewählten Schema vorhanden sind. [1]



Die Formulare/ Widgets sind Analog zu der Datenbankstruktur organisiert.
Auf dem DB-Server : Schema, Table, Column
Frontend-seitig : Modul, Form, Widget

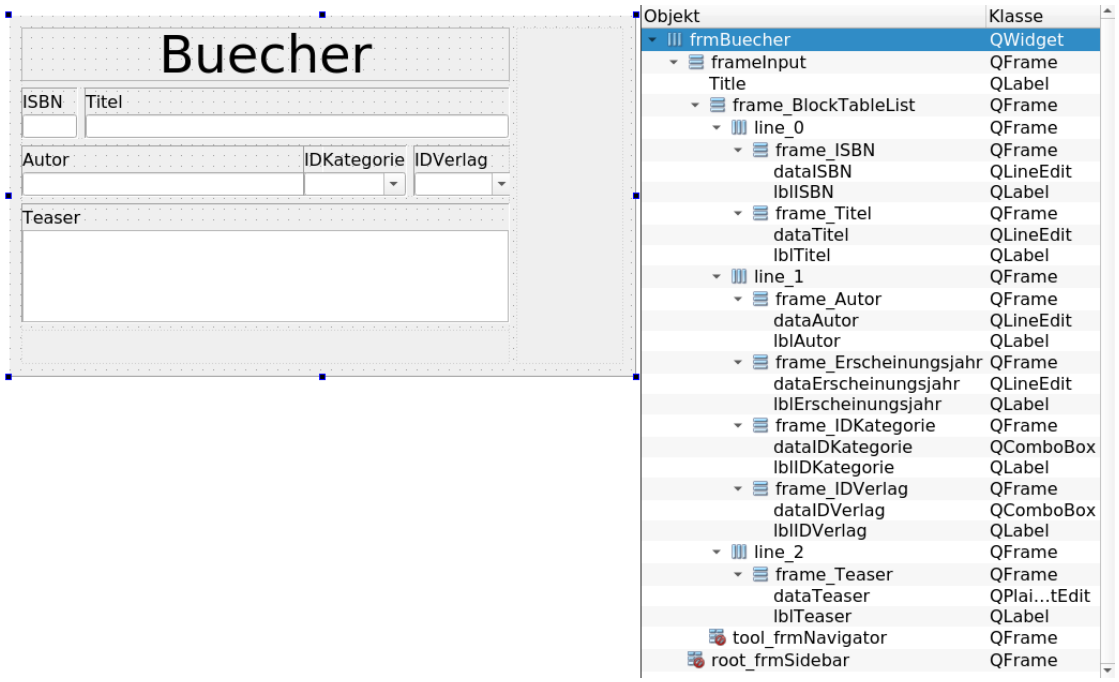
Wir wollen ein Formular zur Tabelle Buecher im Schema Demo erstellen. Modul und Name für das Formular sind frei wählbar, allerdings **muss** der Formularname mit dem Präfix ***frm*** beginnen.

1 Ursprünglich wurde dies durch einen Eintrag in der Tabelle *Event* bewerkstelligt. Inzwischen wird dies durch Analyse des Eintrages im Feld DataSource und anschließendem Setzen von Abhängigkeiten geregelt. Ähnlich wie es Spreadsheets(Excel) tun.

Meistens vergebe ich die Werte wie folgt: **Modul** = **Datenbankschema**, und Name = ***frm*+Tabelle** – dies ist aber nicht zwingend.
In der Combobox **Connection** geben Sie die ID aus der Tabelle Connection ein. Dort können Verbindungen zu beliebigen MySQL-Servern hinterlegt werden, zu denen sie Zugriff haben. Standardmäßig sind dort die Verbindungen **default** und **Frontend** eingetragen. Verwenden Sie hier bitte die Connection-ID ***default;*** dies ist die Verbindung mit der Sie sich zu Beginn einloggen. Mehr zur Tabelle Connection im Kapitel **Connections**.
Mit der ComboBox **View** wählen Sie aus, in welcher Ansicht das Formular starten soll. Standard ist Blockansicht. Mit der CheckBox UpdateInsert erlauben Sie, dass im Modus Ändern auch Datensätze hinzugefügt werden dürfen, falls das Formular als Unterformular verwendet wird. Mehr dazu im Kapitel **Unterformulare**.
In der ComboBox StyleSheet können Sie die ID eines Style-Sheets auswählen.
Im Widget **Resource** geben Sie nun den XML-Code für die Formular-Resource ein …
…
Ha ha, - kleiner Scherz. Natürlich sollten Sie das nicht tun, es sei denn, Sie sind ein Masochist, oder echt, echt gut…
Wir machen es uns natürlich weiterhin so einfach wie möglich. Dazu hat das Formular neben dem Widget **Resource** vier weiter Buttons.
Pressen Sie den Button **Generate**, und es wird die Autolayout-Funktion benutzt um ein Formular zu erzeugen. Auf dem Bildschirm poppt eine Vorschau des Formulars auf. Diese können Sie wieder wegklicken. Falls Sie sich wundern warum sie in der Vorschau die **Sidebar** und den **Navigator** nicht sehen – keine Sorge, das ist schon in Ordnung. Die Erklärung folgt gleich im nächsten Abschnitt.
Das Widget **Resource** enthält nun den XML-Code für das Formular. Um den Resource-Code manuell weiter zu bearbeiten, speichern Sie ihn zunächst mit dem Button **Save** ab. Achten Sie darauf, dass der Dateiname die Endung ***.ui*** hat. Falls Sie Qt-Designer installiert haben, und die Kommandozeile in Tabelle **Variable** angegeben haben, können Sie mit dem Button **Design** den Designer starten, und die Resource bearbeiten. Mit dem Aktionsbutton der Sidebar können Sie zur Sicherheit schon einmal den Datensatz für das Formular in der Datenbank speichern.

B. Steuerung über Präfixe, Konventionen

Wenn Sie das Formular mit Qt-Designer öffnen, sehen Sie folgenden Objektbaum:



Wenn Sie sich die Namen der Widgets ansehen, wird Ihnen auffallen , dass fast alle Widgets mit irgendeinem Präfix beginnen. Dies ist beabsichtigt, denn:

Die Rolle die ein Widget spielt, wird durch das Präfix festgelegt.

Es gelten folgende Regeln:

Präfix	Rest	Widget-Typ	Rolle
data	Spaltenname	Input-Widget	Eingabefeld, das mit der entsprechenden Spalte verknüpft ist.
lbl	Spaltenname	Label-Widget	Das dazugehörige Bezeichnungsfeld. Dieses muss vorhanden sein, wenn das entsprechende Input-Widget existiert
frame_	Spaltenname	Container	Frame in dem das dazugehörigen Input- und Label-Widget eingerahmt sind. Dieses ist optional, aber zu empfehlen
ctrl	beliebig	Input-Widget	Steuerelement um Ereignisse auszulösen, die zur Ausführung von Kommandos führt. Beispielsweise die Buttons in der Sidebar und im Navigator.
info	beliebig	Display-Widget	Anzeige von Zusatzinformationen. Formel, die in den Widget-Eigenschaften hinterlegt ist wird ausgewertet. Es kann dabei auf eine Vielzahl von Variablen zugegriffen werden. Mehr dazu im Kapitel Widget-Eigenschaften.

tool_	Formularname	Container	Das entsprechende Formular wird nachgeladen und in diesen Container eingefügt. Dieses muss sich im Modul **Tools** befinden.
root_	Formularname	Container	Wie ***tool_*,** wird aber nur geladen, wenn es Bestandteil des obersten Hauptformulars, also nicht eines Unterformulars ist. → **frmNavigator** bekommt das Präfix ***tool_***, weil es für jedes Unterformular benötigt wird; **frmSidebar** bekommt das Präfix ***root_***, weil es auf oberster Ebene für die Gesamtsteuerung verantwortlich ist.
sub_	Formularname	Container	Fügt das entsprechende Formular als Unterformular hinzu. Jedes Datenbank-Formular(kein Tool-Form) kann auch als Unterformular verwendet werden. Alle mit ***root_*** beginnenden Container werden ignoriert und das Formular entsprechend verkleinert. Anhand der **Foreign-Keys** wird automatisch die Verknüpfung zum Hauptformular hergestellt.
main_	Formularname	Container	Das oberste Hauptformular selbst wird in diesen Container geladen. Aus Ästhetik-/Designgründen werden auch hier alle mit root_ beginnenden Container ignoriert. D.h. haben sie zu einer Datenbanktabelle das elementare Formular ***frmXYZ*** erstellt, dann ist ein weiteres Formular das lediglich die leeren Container ***main_frmXYZ*** und ***root_frmSidebar*** enthält **funktional identisch**, ist also das neutrale Eins-Element. Der Sinn erschließt sich wenn man sich z.B. das Formular frmPersonPlus aus dem Modul wws_person anschaut. Dies enthält lediglich ein Tab-Widget, in dem die erste Seite das Hauptformular ist und die weiteren Seiten die Unterformulare hierzu.
frame_	Ansichtsart[en] + beliebig	Container	Container für die entsprechende Ansichtsart(***Block***, ***Table*** oder ***List*** -View. Jedes Formular unterstützt zwingend alle 3 Ansichtsarten.Werden mehrere Ansichtsarten im Namen angegeben wird automatisch ein Stack-Widget erzeugt. Fehlt eine der Ansichtsarten ***List*** oder ***Table***-View dann wird es dem **Stack** der ***Block***-Ansicht zugeordnet. Existiert kein Container mit ***Block***-Ansicht, dann wird ein Frame erzeugt, der alle ***data*** und ***lbl***-Widgets umfasst, und diese dem Frame untergeordnet.

Die bequemste Design-Strategie besteht also darin, zunächst die elementaren Formulare, die nur eine Tabelle bedienen zu erstellen und anschließend komplexere Formulare zu entwerfen, die beliebig viele Subform-Container enthalten. Achten sie unbedingt darauf, dass die Foreign-Keys in der Datenbank richtig gesetzt sind.

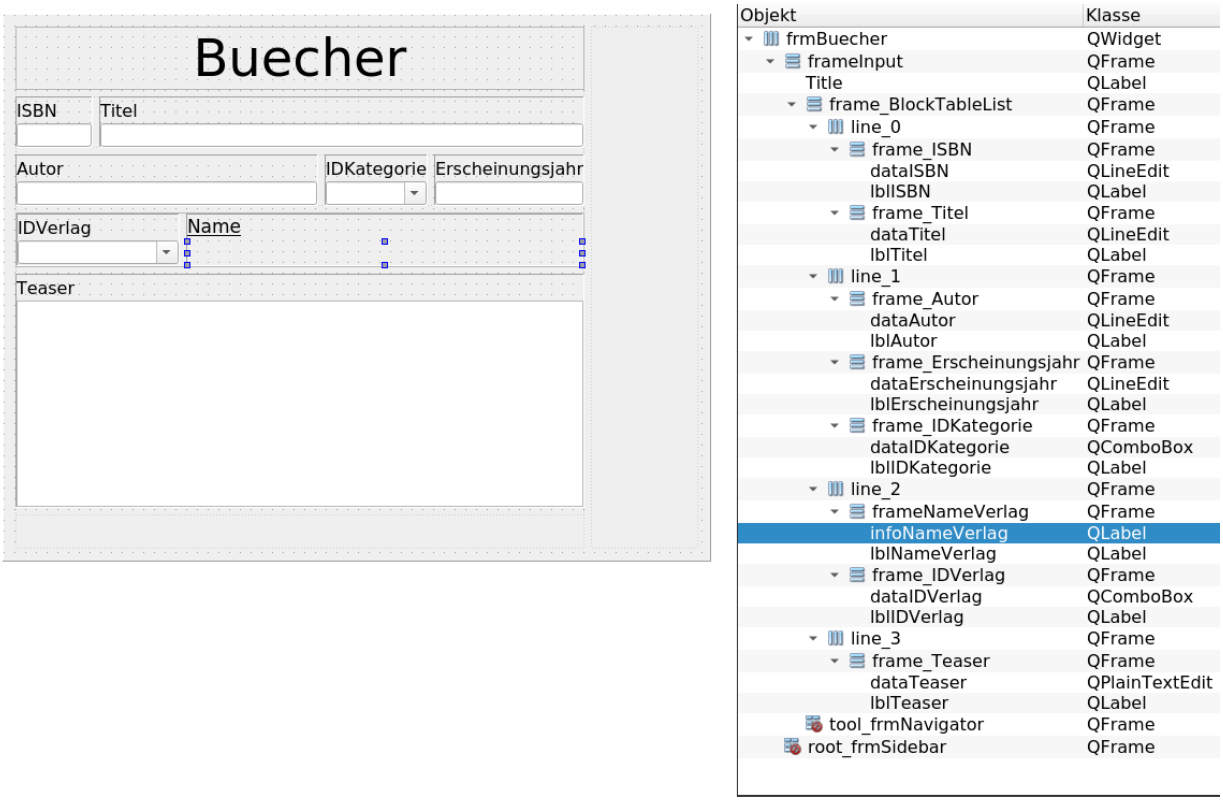
C. Unterstützte Widgets

Unterstützt werden derzeit folgende Widget-Klassen,

Da dieses Programm auf Qt basiert, beginnen alle Klassennamen mit einem ***Q***.

Input-Widgets	QComboBox, QLineEdit, QPlainTextEdit, QSpinBox, QDoubleSpinBox, QTimeEdit, QDateEdit, QDateTimeEdit, QDial, QScrollbar, QSlider, QPushButton, QRadioButton, QCheckBox
Container	alle (hoffentlich…), also: QGroupBox, QScrollArea, QToolBox, QTabWidget, QStackedWidget, QFrame, QWidget, QMdiArea, QDockWidget
Display-Widgets	QLabel, QLCDNumber,QTextBrowser

Nachdem ich einige Anpassungen vorgenommen habe, sieht das Formular folgendermaßen aus:

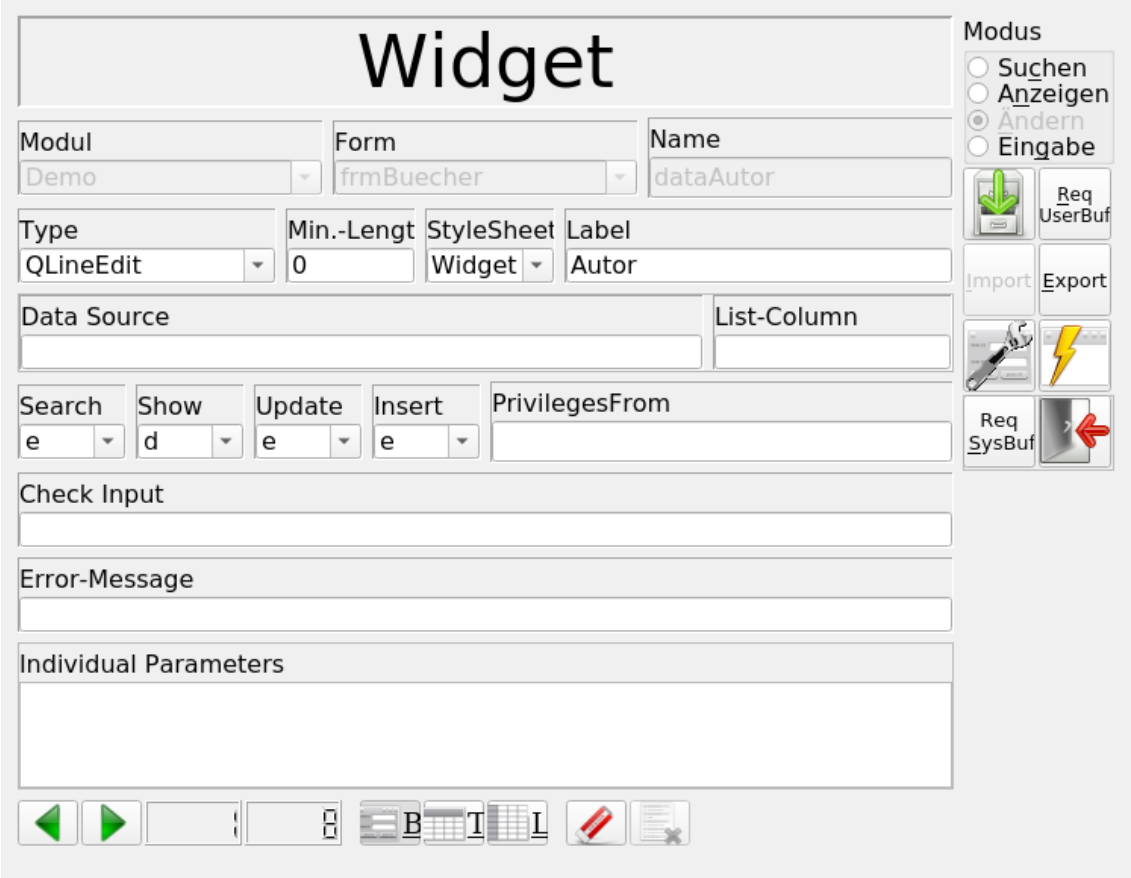


Hinzugekommen ist das Info-Widget **infoNameVerlag**, in der Name und Ort des Verlages zu der ID angezeigt werden soll. Wie das gemacht wird, kommt im nächsten Kapitel.
Außerdem habe ich bei einigen Widgets *MinimumSize*, sowie *horizontalStretch* angepasst.

7.Widget-Eigenschaften in Tabelle Widget

Sobald Sie das Formular fertig designt haben, sollten Sie nochmal das Formular ***Frontend.frmForm*** öffnen, falls Sie es zwischenzeitlich geschlossen haben, und sich den Eintrag für das bearbeitete Formular heraussuchen. Wählen Sie den Update-Modus, und laden Sie mit dem Button **Load**, den Inhalt der lokal gespeicherten Formular-Resource in das Feld **Resource** und speichern Sie den Datensatz in der Datenbank. Nun ist Ihr Formular zur Verwendung verfügbar.
Wenn Sie ein Formular zum ersten mal starten, werden für alle Steuerelemente mit den Präfixen data, ctrl und info automatisch Einträge in der Tabelle Frontend.Widget eingefügt. Diese enthält die

notwendigen Eigenschaften der Widgets, die von Ihnen angepasst werden können. Wenn Sie ein Formular geöffnet haben und den Button Widget-Eigenschaften (der mit dem Schraubenschlüssel) anklicken, öffnet sich automatisch das Formular Frontend.frmWidget, mit den Einträgen für dieses Formular.



Erlauterung der Felder:

Modul, Form, Name	sollte inzwischen klar sein, sind in dieser Tabelle der zusammengesetzte Primärschlüssel, deshalb im Update-Modus deaktiviert.
Type	die Qt-Widget-Klasse. Wird nur von Autolayout verwendet.
Min.-Length	die Mindestbreite des Widgets in Anzahl an Zeichen. Wird nur von Autolayout verwendet.
StyleSheet[i]	Hier kann dem Widget ein individuelle ID aus derTabelle StyleSheet zugewiesen werden
Label	Text des Labelfeldes. Wird nur von Autolayout verwendet.

Data Source	Datenquelle der Listenelemente für die Widget-Klassen **QCombobox**,**QListWidget** und **QTreeWidget**. Es existieren folgende Optionen: • **SQL:** *SQL-Statement* • **Table:** *Tabellenname* • **List:** *Wert1;Wert2;Wert3…* • **auto** Die Option **auto** gilt nur für Datenbank-InputWidgets(Präfix ***data***). Hier wird anhand des Foreign-Keys, die Datenquelle automatisch bestimmt. (Erwähnte ich eigentlich schon, dass es enorm wichtig ist, alle Foreign-Keys in Ihrer Datenbank zu definieren?) Für Info-Widgets(Präfix ***info***) kann hier eine Formel eingegeben werden, die bei jedem Datensatz-Wechsel ausgewertet wird. Verfügbare Variablen sind: • alle Datensätze auf die die Foreign-Keys verweisen(Erwähnte ich eigentlich schon…?) • die augenblicklichen Werte aller Widgets(***ctrl*** und ***data***) • einige spezielle Variablen wie z.B. ParentForm, ParentModul, InputMode
List-Column	Wird zusammen mit Data-Source verwenden. Gibt an Aus welcher Spalte die in **QCombobox**,**QListWidget** oder **QTreeWidget** angezeigten Listenelemente stammen. Diese Widget-Klassen können mit zwei Spalten aus der Daten-Quelle arbeiten, eine für die gespeicherten Daten im aktuellen Datensatz(**Data-Column**: üblicherweise die ID der Datenquelle) und eine für die Anzeige in der Liste Widgets. Beispiel: Combobox zu Speicherung des Wochentages. Gespeichert wird eine **Zahl** von ***1-7***, angezeigt wird der **Text *Montag,Dienstag,*…** Datenquelle ist die Tabelle **Wochentage** mit den Spalten **ID** und **Text**. Dann wäre die Angabe in Data-Source z.B. ***Table: Wochentag*** und der Eintrag in List-Column: ***Text*** Auch bei Data-Source-Option ***auto*** muss List-Column noch explizit angegeben werden Wird in **List-Column nichts** angegeben, dann erscheinen die Werte aus der **Data-Column**
Search, Show, Update, Insert	Hier können, die Widgets abhängig vom Eingabe-Modus aktiviert, deaktiviert oder versteckt werden. Werte sind: e = enabled, d=disabled, h=hidden Für Primärschlüssel-Felder sollten Sie den Update-Modus auf disabled setzen.
Privileges From[ii]	Für Nicht- ***data*** -Widgets kann hier ein Verweis auf eine Tabelle oder Spalte angegeben werden von dem die User-Privilegien übernommen werden.

Check-Input	Eingabeprüfung, wird vor dem Speichern (Aktionsbutton gedrückt) ausgewertet. Optionen sind: • **RE: *Regular-Expression*** • **Num: *Minimum;Maximum*** • **Len: [Min-Length*;] Max-Length*** Zeichenkette-Länge darf Maximal diese Anzahl an Zeichen enthalten. Min-Length ist optional • ***WC: Wildcard-Expression*** Zeichenkette entspricht diesem Muster mit den Wildcards **?** für ein Zeichen und ***** für beliebig viele Zeichen
Error-Message	Fehler-Meldung die angezeigt wird
Individual[iii] Parameters	Hier können komplexe Display-Widgets mit individuellen Parametern Initialisiert werden. Zum Beispiel Minimum und Maximum für **QDial**, **QSlider**, **QScrollBar**. Syntax ist : ***ParameterName1 = Wert1 ; ParameterName2 = Wert2 ;*** …

Unser Beispiel Demo.Buecher hat folgende Tabellenstruktur:

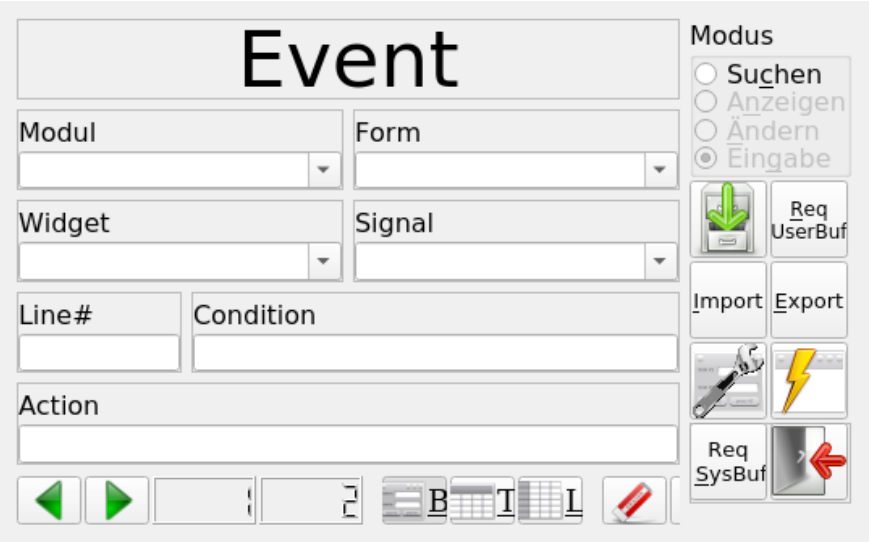
ISBN char(10) Primary-Key
Titel varchar(100)
Autor varchar(100)
IDKategorie varchar(20) , Foreign-Key → Demo.Kategorie.ID
IDVerlag varchar(20), Foreign-Key → Demo.Verlag.ID
Erscheinungsjahr int(11)
Teaser text

Deshalb nehme ich folgende Anpassungen in den Widget-Eigenschaften vor(Fehlertext wurde hier weggelassen; dieser ist identisch mit der Erklärung zu CheckInput):

dataISBN	Update: ***d*** CheckInput: ***RE: ^\d{10}$***	Primary-Key vor Änderungen schützen ISBN muss aus 10 Ziffern bestehen
dataTitel	CheckInput: ***Len: 100***	Titel darf maximal 100 Zeichen haben
dataAutor	CheckInput: ***Len: 100***	Autor darf maximal 100 Zeichen haben
dataIDKategorie	Data-Source: ***auto***	In der Datenbank ist Foreign-Key gesetzt, deshalb funktioniert auto
dataIDVerlag	Data-Source: ***auto***	- dito -
dataErscheinungsjahr	CheckInput: ***NUM:-4000;2200***	Erscheinungsjahr muss zwischen -4000 und 2200 liegen
dataTeaser	CheckInput: LEN: 1,65535	Teaser muss zwische 1 und 65535 Zeichen haben
infoNameVerlag	Demo.Verlag.Name + „“ +Demo.Verlag.Ort	Wird automatisch ausgewertet wenn sich die Werte der Variablen ändern

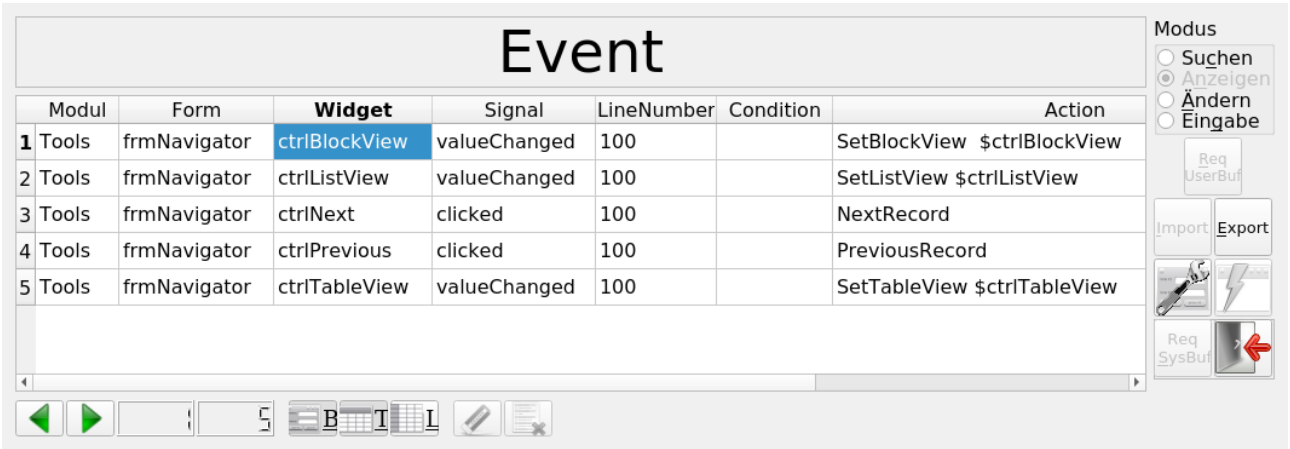
8.Ereignisbehandlung in Tabelle Event

Als nächstes können Sie noch Ereignisbehandlungen für die Widgets Ihres Formulars definieren, falls notwendig. Dies machen Sie mit dem Formular Frontend.frmEvent:



Die Felder **Modul**, **Form** und **Widget** sollten inzwischen klar sein. Im Feld **Signal** können Sie das auswählen welches Ereignis behandelt wird. Im Feld **Line#** können Sie eine Zeilennummer eingeben mit der Sie die Reihenfolge der Aktionen für dieses Ereignis festlegen. Im Feld **Condition**, können Sie eine Bedingung eingeben unter der die nachfolgend angegebene Aktion(Feld **Action**) ausgeführt wird.
Unser Beispiel **Frontend.frmBuecher** ist so simpel, dass keine weitere Ereignisbehandlung angegeben werden muss. Stattdessen schauen wir uns als Beispiele die Ereignisse für den Navigator und die Sidebar an.

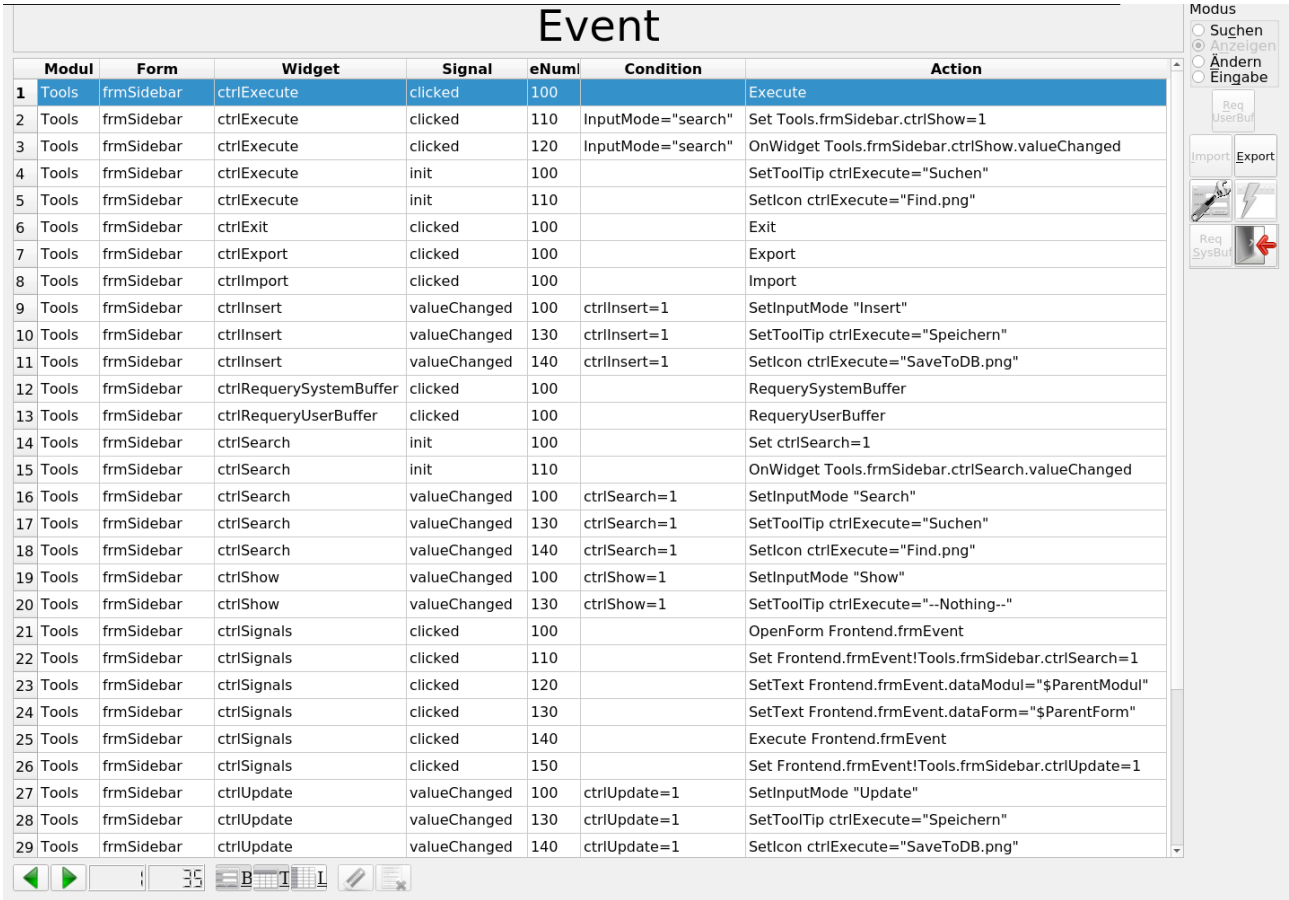
Navigator:



Erklärung zu Zeile 1:
Das Widget **ctrlBlockView** ist ein Button, der *checkable* ist, d.h. er funktioniert wie eine Checkbox. Er verbleibt im gedrückten Zustand, wenn man ihn betätigt, dadurch repräsentiert er einen binären Wert. Dieser wird im Feld Action ausgewertet.

Zeile 3: **ctrlNext** ist **nicht *checkable***, deshalb wird hier jetzt als Signal ***clicked*** gewählt.

Sidebar:



	Modul	Form	Widget	Signal	eNum	Condition	Action
1	Tools	frmSidebar	ctrlExecute	clicked	100		Execute
2	Tools	frmSidebar	ctrlExecute	clicked	110	InputMode="search"	Set Tools.frmSidebar.ctrlShow=1
3	Tools	frmSidebar	ctrlExecute	clicked	120	InputMode="search"	OnWidget Tools.frmSidebar.ctrlShow.valueChanged
4	Tools	frmSidebar	ctrlExecute	init	100		SetToolTip ctrlExecute="Suchen"
5	Tools	frmSidebar	ctrlExecute	init	110		SetIcon ctrlExecute="Find.png"
6	Tools	frmSidebar	ctrlExit	clicked	100		Exit
7	Tools	frmSidebar	ctrlExport	clicked	100		Export
8	Tools	frmSidebar	ctrlImport	clicked	100		Import
9	Tools	frmSidebar	ctrlInsert	valueChanged	100	ctrlInsert=1	SetInputMode "Insert"
10	Tools	frmSidebar	ctrlInsert	valueChanged	130	ctrlInsert=1	SetToolTip ctrlExecute="Speichern"
11	Tools	frmSidebar	ctrlInsert	valueChanged	140	ctrlInsert=1	SetIcon ctrlExecute="SaveToDB.png"
12	Tools	frmSidebar	ctrlRequerySystemBuffer	clicked	100		RequerySystemBuffer
13	Tools	frmSidebar	ctrlRequeryUserBuffer	clicked	100		RequeryUserBuffer
14	Tools	frmSidebar	ctrlSearch	init	100		Set ctrlSearch=1
15	Tools	frmSidebar	ctrlSearch	init	110		OnWidget Tools.frmSidebar.ctrlSearch.valueChanged
16	Tools	frmSidebar	ctrlSearch	valueChanged	100	ctrlSearch=1	SetInputMode "Search"
17	Tools	frmSidebar	ctrlSearch	valueChanged	130	ctrlSearch=1	SetToolTip ctrlExecute="Suchen"
18	Tools	frmSidebar	ctrlSearch	valueChanged	140	ctrlSearch=1	SetIcon ctrlExecute="Find.png"
19	Tools	frmSidebar	ctrlShow	valueChanged	100	ctrlShow=1	SetInputMode "Show"
20	Tools	frmSidebar	ctrlShow	valueChanged	130	ctrlShow=1	SetToolTip ctrlExecute="--Nothing--"
21	Tools	frmSidebar	ctrlSignals	clicked	100		OpenForm Frontend.frmEvent
22	Tools	frmSidebar	ctrlSignals	clicked	110		Set Frontend.frmEvent!Tools.frmSidebar.ctrlSearch=1
23	Tools	frmSidebar	ctrlSignals	clicked	120		SetText Frontend.frmEvent.dataModul="$ParentModul"
24	Tools	frmSidebar	ctrlSignals	clicked	130		SetText Frontend.frmEvent.dataForm="$ParentForm"
25	Tools	frmSidebar	ctrlSignals	clicked	140		Execute Frontend.frmEvent
26	Tools	frmSidebar	ctrlSignals	clicked	150		Set Frontend.frmEvent!Tools.frmSidebar.ctrlUpdate=1
27	Tools	frmSidebar	ctrlUpdate	valueChanged	100	ctrlUpdate=1	SetInputMode "Update"
28	Tools	frmSidebar	ctrlUpdate	valueChanged	130	ctrlUpdate=1	SetToolTip ctrlExecute="Speichern"
29	Tools	frmSidebar	ctrlUpdate	valueChanged	140	ctrlUpdate=1	SetIcon ctrlExecute="SaveToDB.png"

hier wird es schon interessanter:

Zeile 1-3: **ctrlExecute** ist das was ich bisher als Aktionsbutton bezeichnet habe, also der Button der je nach Modus Suche/ Speichern auslöst. Dank der Zeilennummern werden die Aktionen in der angegebenen Reihenfolge abgearbeitet. Der Befehl **ExecuteSQL** führt die entsprechende Datenbank-Aktion aus. Falls wir uns im Modus ***Suchen*** befinden(Feld **Condition**) , müssen wir noch automatisch auf Modus ***Anzeigen*** umschalten. Zeile 2 schalten den RadioButton *ctrlShow* ein. Zeile 3 verweist auf die Ereignisbehandlung für das Einschalten von *ctrlShow*. Das Kommando **OnWidget** stellt den Aufruf der Ereignisbehandlung selbst zur Verfügung. Das hat zur Folge, das die Zeilen 19 und 20 ebenfalls abgearbeitet werden.[2]

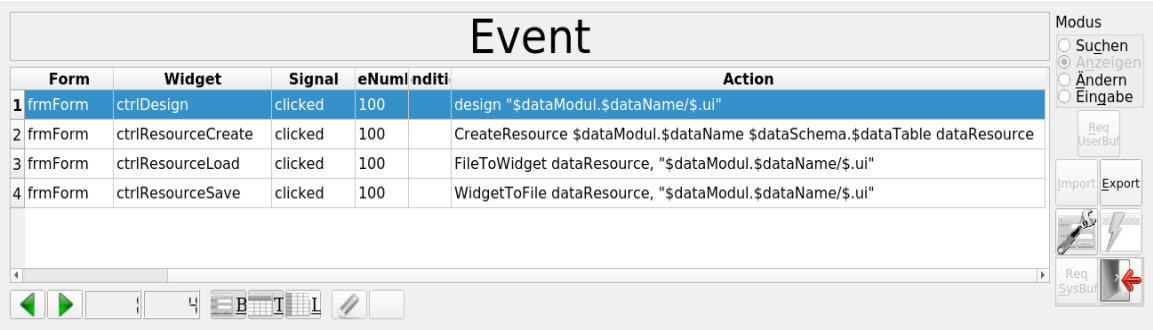
Zeile 4 und 5: Das Signal **Init** ist eigentlich kein Ereignis im eigentlichen Sinne. Es dient dazu Beim Start des Formulars individuelle Initialisierungen vorzunehmen.

Zeilen 21-26: Falls Sie dieses Formular (**Frontend.frmEvent**) aufrufen, indem Sie in der Sidebar eines Formulars, wie z.B. unser Beispiel-Formular **Demo.frmBuecher** den Button mit dem Blitz-Icon drücken(**ctrlSignals**) , dann sehen in diesen Zeilen die Befehle dafür.
Zeile 21 öffnet das Formular **Frontend.frmEvent**.
Zeile 22:ab jetzt wird **Frontend.frmEvent** ferngesteuert. Zunächst schalten wir den Modus Suchen ein. wir befinden uns immer noch in **Demo.frmBuecher**. Da beide eine Sidebar haben müssen wir den Namen des Hauptformulars voranstellen also
Frontend.frmEvent!Tools.frmSidebar.ctrlSearch
Das Setzen auf den Wert 1 löst automatisch bei **Frontend.frmEvent** die entsprechende

2 Falls Sie sich fragen, warum der Aufruf von *OnWidget* nötig ist, da ja die Änderung des Wertes von ctrlShow ein Folge-Ereignis auslösen sollte, dann ist die Antwort ganz einfach: Dies habe ich ganz bewusst unterbunden. **Eine Ereignisbehandlung löst keine weiteren Ereignisse aus !** Die Widgets werden währenddessen gesperrt.

Ereignisbehandlung, als ob es manuell angeklickt wurde.

Als letztes sei hier noch das „**Preprocessing**“ bei der Auswertung einer Formel erwähnt. Als Beispiel nehmen wir das Formular **Frontend.frmForm**, das bei der Erstellung von Formularen kennengelernt haben:



Die Action ***design $dataModul.$dataName/$.ui*** bewirkt, dass der externe Formulardesigner aufgerufen wird mit Formularname + Endung .ui als Dateiname. Das $-Zeichen bewirkt, dass der Wert der Widgets **dataModul** und **dataName** ohne Rücksicht auf jegliche Syntax eingefügt werden und erst dann der Befehl ausgewertet wird. Bei unserem Beispiel führt also das Preprocessing zu der Anweisung ***design Demo.frmBuecher.ui*** , die anschließend ausgeführt wird. Das abschließende **/$** hinter **dataName** ist notwendig um Mehrdeutigkeiten bei der Angaben von Variablennamen zu vermeiden. Ohne es, würde der Präprozessor versuchen die Variable **dataName.ui** zu finden, die aber nicht existiert. Mehr dazu in Kapitel 11 – Variablen und Ausdrücke im Detail.

Als nächstes folgt eine komplettübersicht über die implementierten Ereignissen und Befehle.

A. Implementierte Ereignisse

Im wesentlichen interessieren nur Ereignisse/Signale bei denen sich der Wert des Widgets geändert hat. Aus Gründen der **Vereinheitlichung** wird dieses Signal einfach ***changed*** genannt, egal ob eine Checkbox markiert ,aus einer Listbox etwas ausgewählt oder ein Slider bewegt wurde. Wer mag kann statt dessen die Klassenspezifischen Signale ***valueChanged, TextChanged, EditTextChanged, toggled*** oder ***stateChanged*** angeben; diese sind alle synonym zueinander.

Bei den Widget-Klassen **QComboBox**, **QDateEdit**, **QDial**, **QSpinBox**, **QDoubleSpinbox**, **QLineEdit**, **QScrollBar** und **QSlider** wird unterschieden ob man gerade dabei ist einen Wert zu ändern oder ob man mit der Eingabe fertig ist. Das Erstere wird einheitlich ***editing*** genannt, das letztere ***editingFinished***. Z.B. wird in einem **QLineEdit**-Feld bei j**edem Buchstaben** den man eintippt ein ***editing*** Signal erzeugt. Wird das **QLineEdit**-Feld verlassen wird ein ***editingFinished*** Signal gesendet.
Es gilt ***changed***=***editing*** + ***editingFinished,*** d.h. Geben Sie z.B. in unserem Beispiel **Demo.frmBuecher** für das Feld **dataISBN** ein ***changed***-Ereignis ein, dann wird die Aktion sowohl beim Tippen jeden Buchstabens, als auch beim Verlassen des Feldes ausgeführt. Möchten Sie das nicht können bei diesen Klassen auch ***editing*** bzw. ***editingFinished*** als Signale verwenden.

Darüber hinaus können Sie noch bei den Klassen **QPushButton** und **QListWidget** das Signal ***clicked*** verwenden.

B. Implementierte Befehle

Zunächst möchte ich darauf hinweisen, dass es zwei Arten von Formularen gibt. Die Formulare **frmStart**, **frmNavigator** und **frmSidebar** sind einfache Formulare aus der Klasse **Form**. Ihre Widgets sind nicht an Datenbank-Felder gebunden, werden aber auch über die Datenbank verwaltet wie wir im vorherigen Kapitel gesehen haben.

Datenbankformulare (Klasse **dbForm**) sind an eine Datenbanktabelle gebunden und bieten die üblichen Datenbankfunktionen an. Da Sie von Klasse **Form** abgeleitet sind können sie alles was auch ein einfaches Formular kann.

Syntax ist immer: ***Kommando*** **ArgumentList**
wobei die Argumente in der ArgumentList nach belieben durch Leerzeichen, Komma oder Gleichheitszeichen getrennt werden dürfen.
Ein Argument hat einen von folgenden Typen: Widgetname, Formname, Signalname, Tablename,Value

Die Namen sollten möglichst voll qualifiziert angegeben werden, also Tools.frmNavigator.ctrlNext usw. Werden die Name unqualifiziert angegeben, dann wird, das Formular des Sender-Widgets als Default verwendet. qualifizierte Namen sind folgendermaßen definiert. Angaben in eckigen Klammern sind optional:
qual_Formname = Modulname.Formname[!Modulname.Formname]
qual_Widgetname =qual_Formname.Widgetname
qual_Tablename = Schemaname.Tablename
qual_Columnname=qual_Tablename.Columnname
qual_Signalname=qual_Widgetname.Signalname
Vartype = *Data*, *Text* oder *Label*
Variablename = qual_Widgetname.Vartype oder qual_Columnname oder SpecialVariablename
Single_Field = qual_Columnname#Primarykey
Value = Literal-Value oder Variablename oder Formel, die diese enthält.

Folgende Kommandos können für die Ereignisbehandlung genutzt werden:

Form:

design *Value*	Formular-Designer aufrufen, Value ist ein Dateiname der als Vorschlag im Open-File-Dialog verwendet wird.
disable *Widgetname,Value*	Eingabe für dieses Feld verbieten
enable *Widgetname,Value*	Eingabe für dieses Feld ermöglichen
filetowidget *Widgetname, Value*	Inhalt einer Datei in ein Widget übertragen. Value = Dateiname als Vorschlag für Open-File-Dialog
hide *Widgetname*	Widget unsichtbar machen
hideframe *Widgetname*	Frame eines Unterformulars/ Tool-Formulars unsichtbar machen
loadwidgettext Widgetname, Value	wie **filetowdget**, nur ohne Dialogbox. Dateiinhalt wird ohne Interaktion direkt in den Widgettext geladen.
onwidget *Signalname*	Signalverarbeitung auslösen
openform *Formname*	Formular öffnen

replacemacros Widgetname, qual_Columnname	Ersetzt im angegebenen Widget alle Makros welche die Form [$qual_Columnname]#key[(Arg0,Arg1…)] durch den Inhalt dieses Datenbankfeldes. Weitere Erklärungen folgen nach dieser Tabelle
requerysystembuffer	Den Inhalt der gepufferten Systemtabellen Modul, Form, Event und Widget neu abfragen
requeryuserbuffer	Den Inhalt aller gepufferten Tabellen, auf die vom User verwendete Foreign-Keys verweisen, neu abfragen.
savewidgettext Widgetname, Value	wie **widgettofile**, nur ohne Dialogbox. Text wird ohne Interaktion direkt in die angegebene Datei gespeichert.
set *Widgetname, Value*	Synonym mit Settext
setdata *Widgetname, Value*	den Datenbankseitigen Wert des Widgets ändern
seticon *Widgetname, Value*	Das angezeigte Icon des Widgets ändern
setlabeltext *Widgetname, Value*	Den Label-Text des Widgets ändern
sethtml Widgetname, Value	Setzt den in Value angegebenen Html-Code in Widgets wie z.B. QTextbrowser die Html darstellen können.
settext *Widgetname, Value*	den angezeigten Wert des Widgets ändern.
settooltip *Widgetname, Value*	den Tooltip des Widgets ändern
show *Widgetname*	das Widget wieder sichtbar machen
showframe *Widgetname*	Frame eines Unterformulars/ Tool-Formulars sichtbar machen
widgettofile *Widgetname, Value*	den Text eines Widgets in eine Datei speichern, Value = Dateiname als Vorschlag für Open-File-Dialog

dbForm:
alle Kommandos von Form, plus

autolayout *Tablename*	erzeugt automatisches Layout für die Eingabefelder eines Datenbankformulars, wird von Misc.frmGeneric verwendet
createresource *Formularname, Tablename, Widgetname*	erzeugt ein Formular für die angegebene Tabelle zum Speichern und weiter bearbeiten. Das angegebene Widget erhält den generierten Formularcode.
executesql	Den vom Eingabemodus abhängigen Datenbankbefehl(Select, Update, Insert) ausführen
exit	Das Formular schließen
exportlist *Value*	Den Inhalt des Formulars in eine oder mehrere Dateien speichern, Value=Dateiname
foreach	Alle diesem Schlüsselwort nachfolgenden Zeilen für ein Signal werden für jeden Datensatz im Formular ausgeführt
importlist *Value*	Den Inhalt einer Datei in das Formular laden, Value=Dateiname
nextrecord	Den nächsten Datensatz anzeigen
previousrecord	Den vorherigen Datensatz anzeigen
print *Value1, Value2*	Eine Datei Drucken, Value1=Drucker, Value2=Dateiname
setblockview *Value*	Blockansicht ein-/ausschalten ,Value: 0=aus 1=ein

setinputmode Value	Eingabemodus wählen, Value aus {„search“, „show“, „update“, „insert“}
setlistview *Value*	Listenansicht ein-/ausschalten, Value: 0=aus 1=ein
settableview *Value*	Tabellenansicht ein-/ausschalten, Value: 0=aus 1=ein

Erläuterungen zum Kommando **replacemacros** :
Ziel ist es einen möglichst bequemen und flexiblen Textersatz für Textinhalte in einem Widget zu schaffen.
Die Syntax lautet: **replacemacros** Widgetname, qual_Columnname
Widgetname gibt das Widget an in den der Textersatz stattfindet. Qual_Columnname ist die default-Angabe falls im Text die Makros in verkürzter Form angegeben sind. Um die Quelle des Ersatztextes zu bestimmen ist die exakte Angabe eines einzelnen Datenbank-Feldes nötig. Deshalb hat das Makro die Form $Qual_Columnname#key. key ist der Wert des Primärschlüssels .Falls Sie den vorderen Teil weglassen und nur #key schreiben wird die default-Angabe aus dem Kommando verwendet.
Der aus der Datenbank geholte Text darf darüber hinaus noch die Platzhalter $0, $1 … enthalten. Diese werden durch die Argumente die in Klammern direkt hinter dem Makro angeführt sind ersetzt.
Beispiele:
In dem Schema ***Test*** haben wir die Tabelle ***Schnipsel*** angelegt. Diese hat das numerische Primärschlüsselfeld ***ID*** und die Spalte ***Blabla*** mit den Textinhalten.
Die Tabelle enthält folgende Einträge:

1	Hallo allerseits,
2	Ich wünsche allen einen $0
3	bis bald

Das Formular, das wir gerade bearbeiten enthält das TextWidget ***dataBeispiel*** . Das Kommando würde dann lauten: *ReplaceMacros dataBeispiel, Test.Schnipsel.Blabla*
Angenommen Sie tippen in das Widget folgenden Text ein:
#1
#2(guten Morgen)
Ich habe gerade das Buch „$Demo.Buecher.Titel#3453196570“ von $Demo.Buecher.Autor#3453196570 gelesen
#3
nach auslösen des Kommandos, befindet sich in dem Widget nun der Text:
Hallo allerseits
Ich wünsche allen einen guten Morgen
Ich habe gerade das Buch „Die Foundation-Trilogie“ von Isaac Asimov gelesen
bis bald

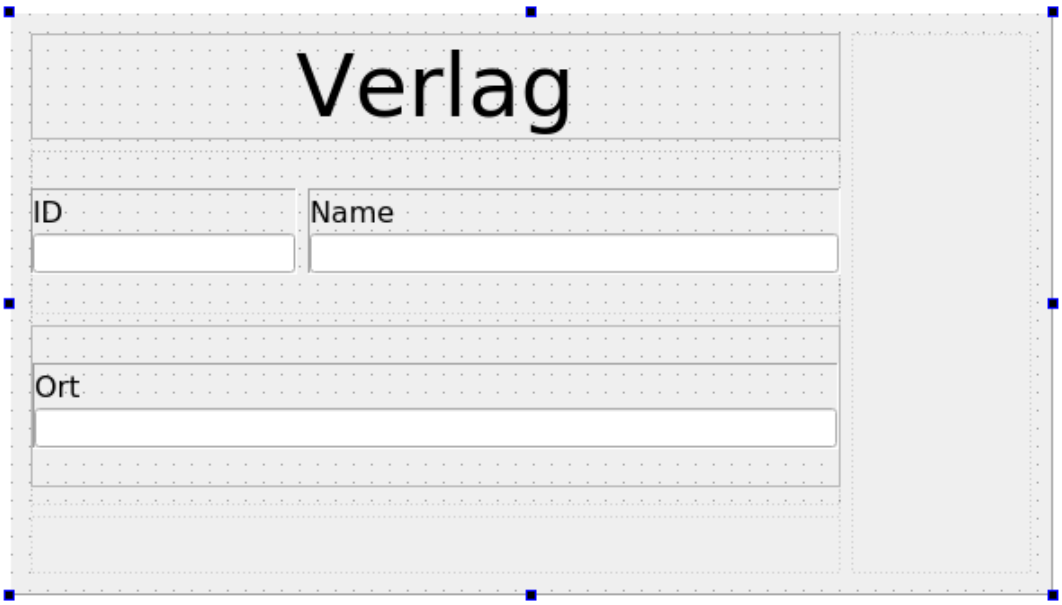
Falls Sie, wie in Zeile 2, Argumente angeben, beachten Sie bitte, dass Kommata die Trennzeichen sind um mehrere Argumente anzugeben. Verwenden Sie dann Anführungszeichen.
Argumente können selbst auch wieder Makros sein.

9.Unterformulare

A. einfaches Beispiel

Hat man in einer Tabelle Foreign-Keys, dann lässt sich das dazugehörige Formular als Unterformular eines Master-Datensatzes verwenden. In unserem Beispiel hat die Tabelle Demo.Buecher Foreign-Keys auf die Tabellen Verlag und Kategorie. Die Tabelle Kategorie ist trivial, sie besteht nur aus einer Spalte, deshalb verwenden wir die Tabelle Verlag und kreieren ein Formular für diese Tabelle mit Unterformular für die Tabelle Buecher.

Zunächst öffnen wir das Formular Frontend.frmForm und generieren ein Formular für die Tabelle Demo.Verlag, das wir am besten frmVerlag nennen. Mit dem Qt-Designer gestalten wir das Layout etwas angenehmer (zwei Zeilen, statt einer).



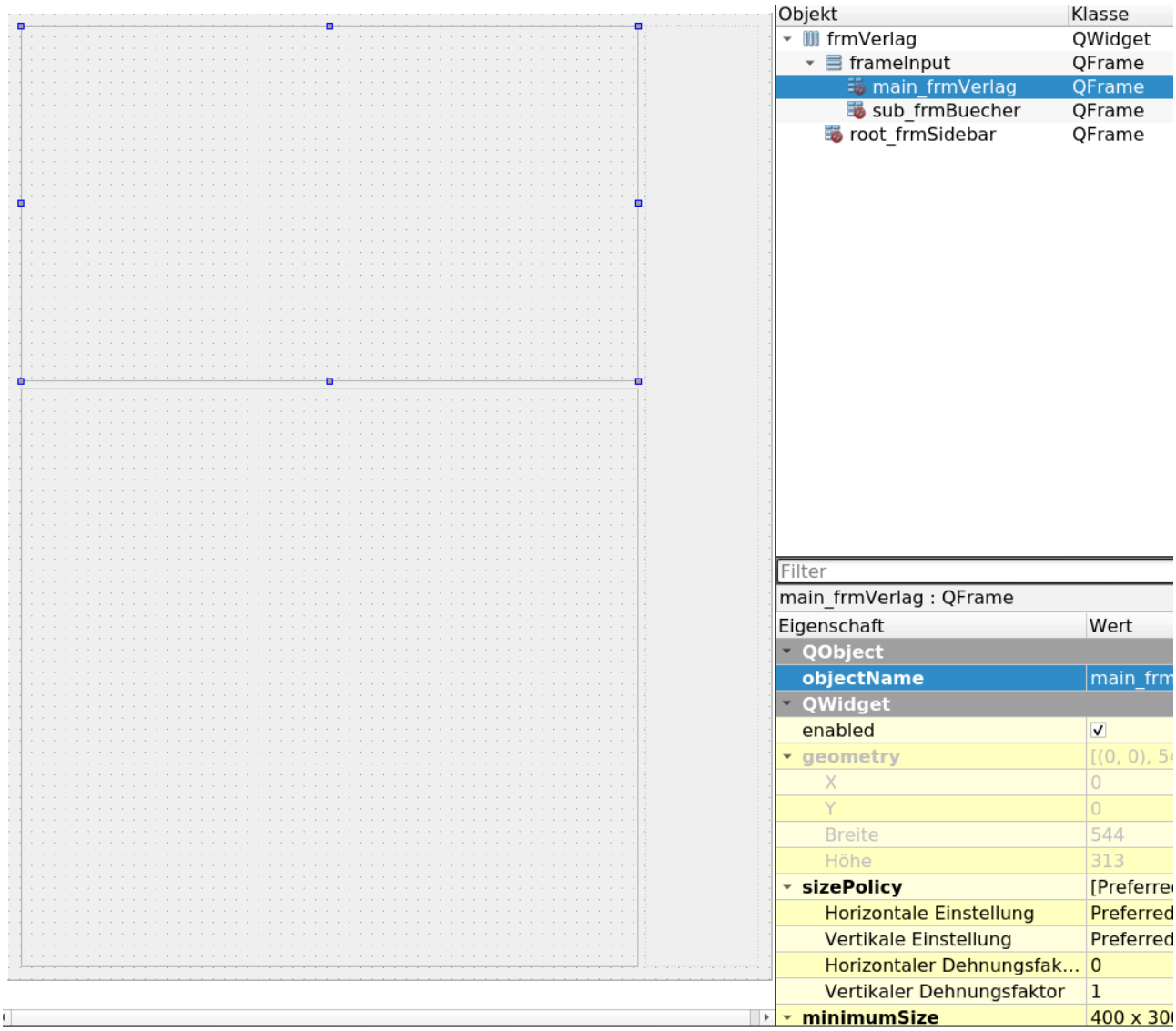
Nach dem Design nicht vergessen es wieder mit Frontend.frmForm zu laden und in Datenbank speichern !

Jetzt haben wir die „elementaren“ Formulare beieinander. Das zusammengesetzte Formular ist jetzt sehr einfach zu erstellen.

Mit dem Designer erstellen wir ein neues Formular, das wir z.B. frmVerlagPlusBuecher oder so nennen .Es enthält nur drei(wesentliche) leere Frames:

- main_frmVerlag:
  lädt die Resource von frmVerlag nach. Alle Widgeteigenschaften und Events werden von frmVerlag übernommen. Das Präfix main_ sorgt dafür, dass das frmVerlag als Root-/TopLevel-/Master- Formular (Tabelle) gilt.
- sub_frmBuecher:
  lädt die Resource von frmBuecher nach. Auch hier werden alle Widgeteigenschaften und Events übernommen. Durch das Präfix sub_ bekommt es die Rolle als Unterformular zugewiesen, das von frmVerlag abhängig ist.
- root_Sidebar:
  Aus den nachgeladenen Formularen, werden alle Frames mit dem Präfix root_ ignoriert(bei

sub_-Frames aus logischen Gründen, bei main_ um der Flexibilität beim Design willen)
Deshalb müssen wir jetzt diesen Frame extra einfügen.



Anmerkung: Der Frame **frameInput** hat für die Programm-Logik keine Bedeutung. Er enthält nur ein Vertical-Layout, das die beiden anderen Frames zusammenhält.

Tja, das war‘s schon. Jetzt müssen Sie nur noch dieses Formular mit Frontend.frmForm der Datenbank hinzufügen und können es verwenden.

Außerdem wird jetzt, das Flag ***UpdateInsert*** wichtig. Mit diesem Formular wollen wir Bücher hinzufügen können auch wenn der Verlag bereits schon existiert. Da der Verlag bereits existiert, müssen wir für die Eingabe den Update-Modus wählen, und können dazu wahlweise Bücher hinzufügen oder ändern. Deswegen müssen wird noch für das Formular Demo.Buecher das Häkchen in der CheckBox setzen.
Falls Ihnen die Notwendigkeit dieses Flags noch nicht ganz einleuchtet, denken Sie an etwas praktischere Beispiele aus einem Buchhaltungssystem:

1. Kundenstammdaten und Telefonnummern zu diesem Kunden(falls dies eine extra Tabelle mit 1:n-Relation ist). Hier ist es sinnvoll, später noch Telefonnummern hinzuzufügen.
2. Rechnungen und Rechnungspositionen. Hier wäre es absolut falsch, später noch Rechnungspositionen hinzuzufügen.

Falls Ihnen das Design des Formulars nicht gefällt, können Sie es auch anders gestalten. Statt das Formular **frmVerlag** nachzuladen können Sie auch die notwendigen Steuerelemente manuell hinzufügen und den Frame main_frmVerlag weglassen.
Der Frame für das Unterformular ist aber zwingend notwendig. Allerdings können Sie ein spezielles Unterformular für diesen Zweck, nennen wir es einmal frmBuecherSub, entwerfen. Der Name des Frames muss dann natürlich sub_frmBuecherSub lauten.

B. fortgeschrittenes Beispiel

Im Modul wws_person befindet sich u.A. das Formular frmPersonPlus. Dieses repräsentiert meine Vorstellungen von der Modellierung der verschiedenen Personengruppen (Kreditoren, Debitoren, Mitarbeiter etc.) in einem professionellen ERP-/WWS- System.

Die erste Registerkarte ***Person*** ist das Hauptformular(Master-Tabelle), gekennzeichnet durch Benennung der Registerkarte als ***main_frmPerson***. Alle weiteren Registerkarten sind Unterformulare zu ***main_frmPerson***. Wie üblich greifen wir auf die elementaren Formulare ***frmPerson***, ***frmRolle, frmKommunikation, frmAdresse, frmGeldverkehr*** und ***frmMitarbeiter*** zurück, die auch separat verwendet werden können.

kurze Erläuterung zu den Unterformularen:

Rolle: hier kann der Person eine oder mehrere Rollen zugeordnet werden. Eine Person kann z.B. Mitarbeiter und Kunde sein.

Kommunikation: alle Kommunikationskanäle zu dieser Person z.B. Festnetztelefon, Mobiltelefon, email etc.

Adresse: alle relevanten Adressen z.B. Lieferanschrift und Rechnungsanschrift.

Geldverkehr: alle Zahlungsverbindungen wie z.B. Bank, Kryptowährungskonto etc.

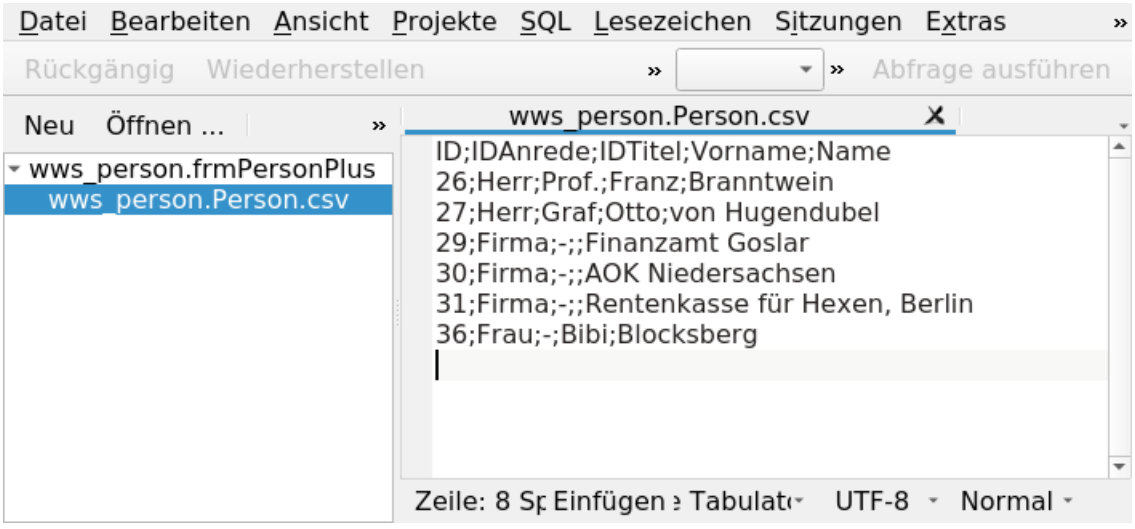
Mitarbeiter: Angaben für die Mitarbeiterverwaltung wie Lohn, Kranken und Rentenversicherung etc.

Z.Zt. fehlt leider noch die Möglichkeit die Unterformulare abhängig von der Rolle ein-/und auszublenden. Wenn z.B. eine Person kein Mitarbeiter ist, dann sollte auch die Registerkarte **Mitarbeiter** ausgeblendet werden.

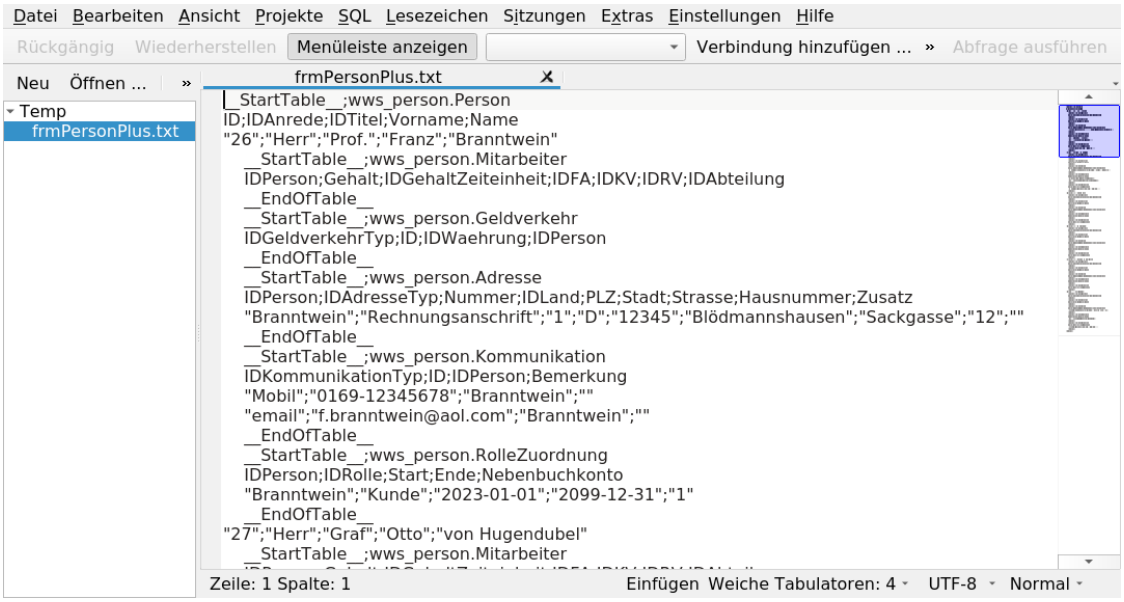
10. Import und Export von Formularinhalten

Sie können den augenblicklichen Inhalt eines Formulars exportieren, um die Daten z.B. zum Drucken von Dokumenten zu verwenden. Dieses Programm hat (noch) keine eigene Routinen zum Dokumentendruck, also müssen Sie die entsprechenden Dokumente z.B. in Libre-Office Writer entwerfen, welche die exportierten Daten einlesen. In einer späteren Version dieser Software werden entsprechende Beispiel-Dokumente beigefügt.
Umgekehrt können Sie große Datenmengen importieren insofern sie sich exakt an die Schnittstellenspezifikation halten.
Es gibt zwei verschiedene Formate für Import und Export von Daten.

1. ImportCSV [Pfad], ExportCSV [Pfad]
   Falls Sie keinen Pfad angeben, erscheint eine FileDialogBox mit der Sie einen Pfad auswählen können. Es wird ein Ordner in diesem Pfad angelegt mit dem Namen des Formulars. In diesem Ordner wird für jede Tabelle (Haupt-/Unterformular) eine eigene Datei angelegt mit dem Namen ***Tabelle*.csv** . Diese Dateien entsprechen den Spezifikationen für csv-Dateien, d.h. die erste Zeile enthält die Spalten-Namen, alle weiteren Zeilen enthalten die Daten, wobei die einzelnen Werte in Anführungszeichen gesetzt, und durch den **Value-Separator** (Semikolon/Komma) voneinander getrennt werden. Sie können den Separator in den globalen Einstellungen festlegen. Standard ist das **Semikolon**. LibreOffice Calc bzw. MS-Excel können dieses Format lesen.



2. ImportStructured [Dateiname], ExportStructured [Dateiname]
   Falls Sie keinen Dateinamen angeben, Erscheint als ersten eine FileDialogBox mit der sie einen Dateinamen angeben können.
   Hier werden zu jedem (Haupt-)Datensatz erst alle Unterdatensätze rekursiv abgespeichert bevor der nächste (Haupt-)Datensatz gespeichert wird, analog zu einer voll aufgeklappten Baumansicht des Datei-Explorers o.Ä.



11. Variablen und Ausdrücke im Detail

- Ein Variablenname besteht aus einer Folge von Alphanumerischen Zeichen inklusive Unterstrich _ , wobei das erste Zeichen ein Buchstabe sein muss. Es dürfen außerdem maximal 3 Punkte enthalten sein als Trennzeichen für die Namenskomponenten.
- Ein Variablenname ist entweder
  - ein Widget-Name mit angefügtem „Wertselektor“(data , text, oder label) hat dann also die Struktur: ***Modul.Form.Widget.Selector***
    Beispiel: *Demo.frmBuecher.dataIDKategorie.Text*
    wird die Angabe des Moduls und/oder des Formulars weggelassen, dann wird
    - bei Ereignissen(Tabelle Event) das Formular, in dem sich das Ereignis-auslösende Widget befindet als default-Angabe verwendet.
    - Wenn in den Widgeteigenschaften(Tabelle Widget) eines Widgets auf ein anderes Widget verwiesen wird, das Formular des ersteren als default verwendet, also davon ausgegangen, dass sich beide im selben Formular befinden.
    - Wird der Wertselektor weggelassen, dann wird *Text* als default-Angabe verwendet.
  - oder ein Spalten-Name(Columnname). Dann lautet also die Struktur:
    Schema.Tabelle.Spalte

Beispiel: *Demo.Buecher.IDKategorie*
Wird die Angabe von Schema und/oder Tabelle weggelassen, dann wird die Tabelle mit der das Formular verknüpft ist, als default-Angabe verwendet.

- Es werden folgende Variablen automatisch angelegt:
  - für jedes Widgets in dem Formular jeweils data, text und label, auch wenn es zu Überschneidungen kommt(in den meisten Fällen z.B. im einfachen LineEdit-Widget ist Widget.data=Widget.text)
  - Die „Columnname“-Variablen für den derzeitigen Datensatz und für alle Datensätze auf die Foreign-Keys verweisen.
  - einige spezielle Variable z.B. ParentForm, ParentModul, CurrentIndex, RecordCount
- aktualisiert werden die Variablen
  - alle bei Datensatzwechsel
  - wenn sich der Wert eines Widget durch Eingabe oder Anweisung ändert. Dies löst automatisch die Aktualisierung der Widgets aus, die auf ersteres in Ausdrücken verweisen(ähnlich wie es Spreadsheets z.B. Excel tun). Sollte es zu zirkulären Abhängigkeiten kommen, dann wird die Aktualisierung nach einer Runde abgebrochen. Zirkuläre Abhängigkeiten können sinnvoll sein, z.B. wenn man zwei Widgets synchronisieren will, deshalb erzeugen sie keine Fehlermeldung.
- Ein Ausdruck ist entweder
  - ein literaler Wert also z.B. 5, 3.14 oder „Banane“
  - eine Variable
  - eine vordefinierte Funktion z.B. sin(…), sqrt(…) log(…)
  - eine Formel, die die oben genannten Komponenten mit Operatoren verknüpft. Implementiert sind die
    - arithmetischen Operatoren +, -, *, /, % (modulo)
    - die Vergleichsoperatoren =, != (ungleich), <, <=, >,>=
    - die logische Operatoren ! (nicht), | (oder), &(und)

12. Globale Variablen

Mit der Tabelle Frontend.Variable haben Sie umfassende Möglichkeiten, das Frontend an Ihre Bedürfnisse anzupassen.
Die Tabelle besteht aus den Spalten Group,Name,Value, Comment.
Die Werte in den Spalten Group und Name dürfen nicht abgeändert werden.

Im Folgenden werden die Variablen für jede Gruppe erklärt.

1. Command

Command	Design	designer %1	%1 = Filename
Command	Print	libreoffice --pt %1 %2	%1 = Printer %2 = Filename

Hier werden die Kommandozeilen für den Aufruf des Designers bzw. Dokumentendruck festgelegt.

2. DynProp

DynProp	Category	Category	NULL
DynProp	InputMode	InputMode	NULL
DynProp	Mandatory	Mandatory	NULL

DynProp steht für Dynamic Properties. Diese können Sie bei der Gestaltung von Stylesheets abfragen.
Category ist die Rolle des Widgets in Dem Formular. Werte sind die Präfixe der Widgets, also: data, ctrl, info und lbl, insofern Sie nicht andere Präfixe festlegen.
InputMode enthält die Werte Search, Show, Update, Insert.
Mandatory zeigt an, dass dieses Feld ein Pflichteingabefeld ist. Werte sind true, false

3. Export:

Export	EndOfTable	__EndOfTable__	NULL
Export	StartTable	__StartTable__	NULL
Export	ValueSeparator	;	NULL

EndOfTable/StartTable markieren Anfang und Ende einer (Unter-) Tabelle in ExportStructured
ValueSeparator trennt die einzelnen Werte eines Datensatzes.

4. File:

File	ErrorLog	error.log	NULL

Name der Error-Log-Datei. Pfad befindet sich unter Path,Log

5. Form:

Form	InputModeInsert	insert	NULL
Form	InputModeSearch	search	NULL
Form	InputModeShow	show	NULL
Form	InputModeUpdate	update	NULL

String-Werte der Eingabemodi. Diese finden Anwendung in Event-Bedingungen und -Aktionen, sowie in Stylesheets.
Ändern Sie diese Werte, falls sie die Werte z.B. in einer anderen Sprache haben möchten.

6. FormBuilder:

FormBuilder	FontDataWidget	Sans Serif;9;50	NULL
FormBuilder	FontLabel	Sans Serif;9;50	NULL
FormBuilder	LengthContractionFactor	0.4	NULL
FormBuilder	MaxFrameWidth	800	NULL
FormBuilder	MaxLineEditLength	80	NULL
FormBuilder	MinFrameWidth	400	NULL
FormBuilder	ResourceTemplate	Misc.frmTemplateStandardForm	NULL
FormBuilder	Spacing	6	NULL
FormBuilder	TargetAspectRatio	1.8	NULL
FormBuilder	TitleWidget	Title	NULL
FormBuilder	WidgetHeightFactor	1.2	NULL

Hier können Sie einige Parameter für das AutoLayout von Formularen anpassen.
FontDataWidget u. FontLabelWidget: Verwendung erklärt sich von selbst. Die Fontangabe besteht aus Font-Familie; Font-Größe in Punkt; Gewicht von 0 bis 99, wobei 0 sehr dünn und 99 extrem fett ist. Die Angaben müssen durch Semikolon voneinander getrennt werden.
WidgetHeightFactor: Höhe des Font multipliziert mit diesem Faktor ergibt die Höhe des

Widgets.
MaxLineEditLength und LengthContractionFactor:
Es wäre übertrieben, die nominelle(in der Datenbank festgelegte) Anzahl an Zeichen eines Feldes als Breite der Eingabefelder zu verwenden. Deshalb wird die real darstellbare Zeichenanzahl mit folgender Formel aus der nominellen Feldbreite berechnet.
**int(MaxLineEditLength *
(1.0-exp(-LengthContractionFactor*length/MaxLineEditLength))) +1**
wobei ***length*** die nominelle Zeichenanzahl ist. Dadurch konvergiert die reale Feldbreite gegen MaxLineEditLength für große Werte von length. Der LengthContractionFactor verursacht eine weitere Verkürzung für kleine Werte von length.Ohne diesen Faktor würde die Formel für ein Zahlenfeld(Breite 11 Zeichen) ein Ergebnis von ebenfalls 11 liefern. Reichen würde aber meistens eine Breite von 4 oder 5.
TargetAspectRatio, MinFrameWidth, MaxFrameWidth:
Mit **TargetAspectRatio** können Sie ein gewünschtes Seitenverhältnis(Breite/Höhe) festlegen, das die angeordneten Felder einnehmen, wobei der Frame in denen sich die Felder befinden mindestens **MinFrameWidth** und maximal **MaxFrameWidth** breit ist. Der Vorgabewert für TargetAspectRatio kann allerdings nur sehr grob erfüllt werden. Der erzeugte Frame hat vielleicht ein Seitenverhältnis von 1,5 statt 1,8. Es wird aber diejenige Anordnung ermittelt, die dem Vorgabewert am nächsten kommt.
Spacing: horizontaler und vertikaler Abstand zwischen den Widgets.
ResourceTemplate: Modul und Name der verwendeten leeren Formular-Schablone, in welche die Felder für die Tabelle eingefügt werden.
TitleWidget: Name des Titel-Widgets. Dieses erhält den Tabellennamen als Text.

7. Interpreter:

Interpreter	ValueOf	$	NULL

ValueOf ist der Operator der beim Preprocessing dafür sorgt, den Variablennamen durch seinen Wert zu ersetzen.

8. Layout:

Layout	SubformAllIndicator	all	NULL
Layout	SubformNameSeparator	__	NULL

Diese Gruppe enthält Ergänzungen zur Gruppe Prefix, mit weiteren Steuerungen für den Namen eines Widgets. Beide Variablen dienen dazu mehrere Unterformulare mit einem Frame zu generieren.
SubformNameSeparator dient dazu mehrere Unterformularnamen voneinander zu trennen.
SubformAllIndicator dient zur Erzeugung aller möglichen Unterformulare zu einem Hauptformular. Beispiel Hauptformular: frmPerson
Ein Frame mit dem Namen **sub_frmAdresse__frmKommunikation** erzeugt ein TabWidget mit diesen beiden Unterformularen.
hat der Frame den Namen **sub_all** wird ein TabWidget mit allen Unterformularen erzeugt. (Achtung: dieses Feature ist noch nicht getestet)

9. NameC,NameF,NameM,NameS,NameT,NameV,NameW:
Auf eine Auflistung der Variablen dieser Gruppen verzichte ich an dieser Stelle. Diese Gruppen enthalten die Namen, die für
Schemata(NameS),Tabellen(NameT),Spalten(NameC),
Module(NameM),Formulare(NameF), Widgets(NameW) und sonstige Variablen(NameV) für das Modul Frontend verwendet werden. Standardmäßig ist der Wert identisch mit dem Variablennamen. Auch hier können Sie den Wert abändern, falls Sie die Tabellen-,Spaltennamen etc. zum Beispiel in einer anderen Sprache darstellen wollen. Falls Sie diese Variablen ändern wollen, müssen Sie es tun bevor Sie die Datenbank **Frontend**

generieren. Falls Sie dies später tun, müssen Sie darauf achten, es konsistent zu tun.
Beispiel: Sie wollen den Spaltennamen ***Widget*** auf den deutschen Namen ***Steuerelement*** abändern. Zunächst ändern Sie den Wert für die Variable NameC,Widget auf Steuerelement ab.
Danach müssen Sie für alle Tabellen, die die Spalte ***Widget*** enthalten (Tabelle Event) den Spaltennamen auf ***Steuerelement*** abändern.
Anschließend müssen Sie noch in allen Formeln und Statements und Formular-Resourcen in denen die Spalte verwendet wird entsprechend abändern.
Diese Änderungen zu automatisieren ist auf der ToDo-Liste für zukünftige Programmerweiterungen.

10. Prefix:

Prefix	BlockView	block	NULL
Prefix	DataWidget	data	NULL
Prefix	Form	frm	NULL
Prefix	FrameMainForm	main_	NULL
Prefix	FrameRootOnlyForm	root_	NULL
Prefix	FrameSubForm	sub_	NULL
Prefix	FrameToolForm	tool_	NULL
Prefix	Frame	frame_	NULL
Prefix	InfoWidget	info	NULL
Prefix	LabelWidget	lbl	NULL
Prefix	ListView	list	NULL
Prefix	SimpleWidget	ctrl	NULL
Prefix	TableView	table	NULL

In dieser Gruppe befinden sich alle Präfixe zur Steuerung des Formularlayouts, wie in Kapitel 6 – Formular-Design beschrieben.

11. Syntax:

Syntax	CILenIndicator	len	NULL
Syntax	CINumIndicator	num	NULL
Syntax	CIRegExpIndicator	re	NULL
Syntax	CISeparator	:	NULL
Syntax	CIWildCardIndicator	wc	NULL
Syntax	DSSeparator	:	NULL
Syntax	DSTAuto	auto	NULL
Syntax	DSTColumn	column	NULL
Syntax	DSTextNoSource	- no Source -	NULL
Syntax	DSTList	list	NULL
Syntax	DSTSql	sql	NULL
Syntax	DSTTable	table	NULL

Hier sind die Parameter enthalten, für die Syntax-Analyse bei den Feldern CheckInput und DataSource in der Tabelle Widget.

12. Widget-State:

WidgetState	Disabled	d	NULL
WidgetState	Enabled	e	NULL
WidgetState	Hidden	h	NULL

13. Connections

Während des Betriebes werden mindestens 2 Datenbankverbindungen verwendet. Wenn Sie das Programm starten, wird zunächst eine Verbindung zum Datenbank-Schema ***Frontend*** benötigt. Die Verbindungsinformationen hierfür sind in der Datei ***Frontend.ini*** eingetragen. Der User der dort eingetragen ist(Name ist ebenfalls ***Frontend***) hat lediglich Leserechte in diesem Datenbankschema

und sonst nichts. Er besorgt alle notwendigen Informationen über die verwendeten Formulare, Widgets, Events usw. Außerdem wird er für die Ausführung des Start-Formulars verwendet.
Wenn Sie ein weiteres Formular starten, wird die Verbindung, deren ID für dieses Formular hinterlegt ist verwendet. Standardmäßig ist dies die Verbindung ***default***.
Sie können in der Tabelle beliebig weitere Verbindungen für den Zugriff auf entfernte Server anlegen.
In der Regel sollten Sie die Felder User und Passwort leer lassen. Die führt dazu, dass bei der ersten Verwendung dieser Verbindung ein Login-Dialog erscheint, der danach fragt.
Eine Verbindung wird immer nur bei Bedarf geöffnet, und nach Ausführung der Datenbank-Kommandos wieder geschlossen.

Im Folgenden werden Features beschrieben, die auf der ToDo-Liste für eine zukünftige Version, stehen, aber noch nicht implementiert oder getestet sind.

14. StyleSheets

Stylesheets sind bereits vorgesehen. Die Tabelle Stylesheet enthält 3 Einträge mit den IDs Application, Form und Widget.
Beim Programmstart wird für die Anwendung als ganzes das Stylesheet mit der ID Application verwendet. In den Tabellen Form und Widget ist jeweils eine Spalte IDStylesheet vorgesehen, mit der individuell Stylesheets zugeordnet werden können.
Dieses Feature ist noch nicht getestet, die Stylesheets werden aber schon geladen.

15. User-Rechte

Die Verwaltung von User-Rechten erledigt der MySql-Server, Sie brauchen also keine Sorgen haben, dass ein User Dinge tut, die er nicht darf.
Was in der Software fehlt, ist der Abgleich mit den im Server hinterlegten Rechten. Ein Anwender kann also zunächst ein Formular starten, für dessen Tabelle oder einzelne Felder er keine Berechtigung hat, und würde erst beim Versuch auf diese Daten zuzugreifen eine Fehlermeldung erhalten.

16. Unterstützung von Änderungen im Datenbankmodell

Dieser Punkt ist eigentlich der wichtigste und wertvollste von allen. Zur Zeit ist es in allen Unternehmen immer noch so, dass das Datenbankmodell des verwendeten ERP-Systems völlig starr ist. Wenn das Unternehmen wächst und/oder sich organisatorisch ändert, passiert es schnell, dass das ERP-System nicht mehr die Abläufe transparent wiederspiegelt. Dann wird bei der Datenhaltung getrickst(„*Wenn ich in Feld xyz den Wert abc eintrage, dann bedeutet das in Wahrheit blablubb...*“) , bis keiner mehr durchblickt. Das ist furchtbar und treibt die Anwender in den Wahnsinn. Ein einsichtiger Unternehmer wird an dieser Stelle ein neues ERP-System kaufen, das an die derzeitige Organisation angepasst ist. Diese Geldausgabe ist aber überflüssig, wenn man ein System hat, das sich flexibel an neue Gegebenheiten anpassen kann.
Da dieses Frontend alle Formulare, Widgets usw, über Datenbanktabellen steuert ist es jetzt nicht mehr schwer Änderungen zumindest halbautomatisch mit einer ToDo-Liste zu unterstüzen.

A. Einfügen einer Tabellenfeldes

Dies ist der einfachste Fall. Es genügt alle Formulare aufzulisten, die die betroffene Tabelle verwenden.

B. Generierte To-Do -Liste bei Änderungen eines Tabellenfeldes

Wenn ein Tabellenfeld geändert(anderer Name) oder gelöscht wird, können viele verschiedene Teile betroffen sein. Die ToDo-Liste muss folgende Dinge enthalten:

- Alle Formulare die auf dieses Feld zugreifen
- In Tabelle Widget, alle Einträge die in dem Feld DataSource Formeln oder SQL-Statements enthalten, die das betroffene Tabellenfeld verwenden.
- In der Tabelle Event, alle Einträge, die in den Feldern Condition und Action Formeln enthalten, die das betroffene Tabellenfeld verwenden
- Handelt es sich um ein (Teil-)Schlüsselfeld, müssen noch alle betroffenen Fremdschlüssel abgeändert werden.

C. Zerlegen einer Tabelle

Dies ist die heikelste, aber auch wichtigste Operation am Datenbankmodell. Es kommt sicherlich sehr häufig vor, dass man eine 1:n-Relation von einer Tabelle abspalten muss, da man Sie beim Design des Datenbankmodells noch nicht kannte. Nehmen wir als Beispiel ein Personenverzeichnis. Vor 30 Jahren, genügte noch ein Feld für die Telefonnummer. Oder besser 2 Felder für Telefon dienstlich und Telefon privat. Später kamen dann noch das Mobiltelefon hinzu , also hat man in Tabelle ein entsprechendes Feld hinzugefügt. Mit dem Internet kam dann noch email hinzu, außerdem ist man heute auch noch über Snapchat, WhatsApp und weiß der Geier was noch alles erreichbar. Jetzt wird es Zeit all diese Felder, die irgendwelche Kommunikationskanäle kennzeichnen, abzuspalten, eine untergeordnete Tabelle Kommunikation zu erzeugen und mit der Master-Tabelle zu verknüpfen.

Die ToDo-Liste hierfür sieht folgendermaßen aus:

- Die Vereinigung der ToDo-Listen aller abgespaltenen Felder, gemäß Punkt B.
- Erstellen eines Formulars für die neue(abgespaltene) Tabelle.
- In den betroffenen Formularen, werden die betroffenen Felder gelöscht und stattdessen ein Unterformular für die neue Tabelle eingefügt.
- Die eigentliche Zerlegung der Tabelle in der Datenbank; am besten mit Backup des vorherigen Zustands.

Da die Formulare komplett über die Datenbank gesteuert werden, lassen sich alle notwendigen Änderungen in einem SQL-Script zusammenfassen.

Das bedeutet, man geht bei einer Änderung am Datenbankmodell folgendermaßen vor:

1. To-Do-Liste erstellen
2. Anhand der ToDo-Liste SQL-Script mit allen notwendigen Änderungen erstellen
3. System zur Wartung herunterfahren, alle User beenden Ihr Programm
4. SQL-Script ausführen

5. System wieder hochfahren

Die Stillstandszeit für das System beträgt also nur wenige Minuten.

17. Web-Grabber

Was ist wenn man Informationen aus dem Internet z.B. von Wikipedia in seine Datenbank übernehmen will ? Für Jedes Feld die Werte von Hand in die eigene Datenbank zu übertragen, ist evtl. zu umständlich. Diese Arbeit ließe sich „skripten“ mit Hilfe einer Tabelle die folgende Spalten enthält: Table, Column,Website-URL, TextPreceding,Textfollowing.

Ein Datensatz in dieser Tabelle ist also so zu verstehen, dass von der Website-URL der Text, der sich zwischen TextPreceding und TextFollowing befindet, in die Spalte Column der Tabelle Table gespeichert wird. Evtl. wird dieses Feature so gestaltet dass eine .csv erzeugt wird, die man aus dem passenden Formular heraus importieren kann, um das Ergebnis manuell zu überprüfen. (Ich fürchte, es kann dabei viel schiefgehen, da Websites möglicherweise nicht so homogen sind, wie man es erwartet, damit alles glattgeht. Das Internet ist nun mal eine allwissende Müllkippe.)

i StyleSheets sind implementiert aber noch nicht getestet
ii User-Privilegien sind noch nicht implementiert
iii Wird noch umfasend erweitert. Geplant sind viele komplexe Display-Widget-Klassen für alle möglichen Diagramm-Arten.