
Fußballverein

Informationstechnische Systeme
4AHITM 2015/16

Antonio Pavic

Version 1.0

Inhaltsverzeichnis

| | |
|----------------------------|----|
| 1. Aufgabenstellung: | 3 |
| 2. Er-Diagramm | 7 |
| 3. SQL-Script | 7 |
| 3.1.Create - Script | 7 |
| 3.2. Drop - Script: | 7 |
| 4. Datenbank erstellen | 9 |
| 4.1. Postgres installieren | 9 |
| 4.2. Datenbank erstellen | 9 |
| 5. Datensätze generieren | 10 |
| 6. SQL-Abfragen | 12 |
| 7. Qt | 13 |
| 7.1.Einführung QT | 16 |
| 7.2.Qt Probleme | 19 |
| 8. Quellen und Referenzen | 20 |

1. Aufgabenstellung:

„Ein österreichischer Fußballverein braucht eine neue Datenbank, um zumindest die Personenverwaltung auf eine professionelle Basis zu stellen. In dieser Datenbank werden folgende Daten verwaltet:

Jede Person ist identifiziert durch eine eindeutige Personalnummer (kurz "persnr"). Außerdem werden zu jeder Person folgende Informationen verwaltet: Vorname ("vname"), Nachname ("nname"), Geschlecht ("geschlecht") und Geburtsdatum ("gebdat"). Für "geschlecht" sind einzig die Eingaben "W", "M" und "N" gültig.

Jede Person, die am Vereinsleben beteiligt ist, gehört zu genau einer der folgenden 4 Kategorien: Angestellter, Vereinsmitglied (kurz "Mitglied"), Spieler oder Trainer. Für all diese Kategorien von Personen müssen zusätzliche Informationen verwaltet werden:

Von jedem Angestellten werden das Gehalt ("gehalt"), die Überstunden ("ueberstunden") und die E-Mail-Adresse ("e_mail") vermerkt.

Für jedes Vereinsmitglied werden ausständige Beiträge ("beitrag") abgespeichert.

Jeder Spieler hat eine Position ("position") (z.B. Tormann, Stürmer, etc.). Außerdem sind Gehalt ("gehalt") sowie Beginn ("von") und Ende ("bis") der Vertragsdauer abzuspeichern. Jeder Spieler ist zumindest einer Mannschaft zugeordnet. Die Spieler innerhalb einer Mannschaft müssen jeweils eine eindeutige Trikot-Nummer ("nummer") zwischen 1 und 99 haben.

Jeder Trainer hat ein Gehalt ("gehalt") sowie Beginn ("von") und Ende ("bis") der Vertragsdauer. Jeder Trainer ist genau einer Mannschaft zugeordnet.

Der Verein hat mehrere Mannschaften. Für jede Mannschaft sind folgende Informationen zu verwalten: eine eindeutige Bezeichnung ("bezeichnung") (wie 1. Mannschaft, Junioren, A-Jugend, ...), die Klasse ("klasse") sowie das Datum des nächsten Spiels ("naechstes_spiel"). Jede Mannschaft hat mindestens 11 Spieler, genau einen Chef-Trainer und genau einen Trainer-Assistenten. Beide sind als Trainer des Vereins registriert. Außerdem hat jede Mannschaft einen Kapitän (der ein Spieler des Vereins ist).

Der Verein hat mehrere Standorte. Jeder Standort ist identifiziert durch eine eindeutige ID ("sid"). Außerdem sind zu jedem Standort folgende Informationen verfügbar: Land ("land"), Postleitzahl ("plz") und Ort ("ort").

An jedem Standort gibt es 1 oder mehrere Fan-Clubs. Jeder Fan-Club hat einen für den jeweiligen Standort eindeutigen Namen ("name"), ein Gründungsdatum ("gegrundet") und einen Obmann ("obmann"), der Vereinsmitglied sein muss. Ein Vereinsmitglied kann von höchstens einem Fan-Club der Obmann sein. Die Fan-Clubs werden von Angestellten des Vereins betreut: Und zwar kann jeder Angestellte einen Fan-Club jeweils für einen gewissen Zeitraum betreuen. Dieser Zeitraum ist durch Anfangsdatum ("anfang") und Endedatum ("ende") bestimmt. Jeder Angestellte kann 0 oder mehrere Fan-Clubs betreuen, und jeder Fanclub kann von einem oder mehreren Angestellten betreut werden.

Der Verein führt Aufzeichnungen über die absolvierten Spiele. Jedes Spiel ist gekennzeichnet durch die Bezeichnung der Mannschaft ("mannschaft") und das Datum ("datum"). Für jedes Spiel werden der Gegner ("gegner") und das Ergebnis ("ergebnis") eingetragen, das einen der Werte "Sieg", "Unentschieden" oder "Niederlage" haben kann. Außerdem werden zu jedem Spiel die beteiligten Spieler abgespeichert, wobei für jeden Spieler die Dauer Einsatzes ("dauer") als Wert zwischen 1 und 90 vermerkt wird.

Aufgabenstellung

1.) Schreiben Sie die nötigen CREATE-Befehle, um die vorgestellten Relationen mittels SQL zu realisieren. Dabei sind folgende Punkte zu beachten:

- * Die Datenbank soll keine NULL-Werte enthalten.
- * Realisieren Sie folgende Attribute mit fortlaufenden Nummern mit Hilfe von Sequences: "persnr" von Personen und "sid" von Standorten. Für das "persnr"-Attribut sollen nur gerade, 5-stellige Zahlen vergeben werden (d.h.: 10000, 10002, ..., 99998). Für das "sid"-Attribut sollen beliebige positive Zahlen (d.h. 1,2, ...) vergeben werden.
- * Falls zwischen 2 Tabellen zyklische FOREIGN KEY Beziehungen herrschen, dann sind diese FOREIGN KEYs auf eine Weise zu definieren, dass es möglich ist, immer weitere Datensätze mittels INSERT in diese Tabellen einzufügen.

2.) Schreiben Sie INSERT-Befehle, um Testdaten für die kreierten Tabellen einzurichten. Jede Tabelle soll zumindest 100000 Zeilen enthalten. Sie dürfen die Wahl der Namen, Bezeichnungen, etc. so einfach wie möglich gestalten, d.h.: Sie müssen nicht "real existierende" Fußballer-Namen, Länder, Städte, etc. wählen. Stattdessen können Sie ruhig 'Spieler 1', 'Spieler 2', 'Land 1', 'Land 2', 'Stadt 1', 'Stadt 2', etc. verwenden. Sie können für die Erstellung der Testdaten auch entsprechende Generatoren verwenden!

3.) Schreiben Sie die nötigen DROP-Befehle, um alle kreierten Datenbankobjekte wieder zu löschen.

Lösen Sie die folgenden Probleme mittels SQL:

S1.) (Fan-Club Betreuung) Wählen Sie "per Hand" die Personalnummer eines Angestellten aus Ihren Testdaten aus. Schreiben Sie eine SQL-Anfrage, die jene Fan-Clubs ermittelt, die dieser Angestellte im Moment nicht betreut. Geben Sie zu jedem derartigen Fan-Club die Standort-ID und den Namen des Fan-Clubs aus. Bemerkung: Ein Fan-Club wird von einem Angestellten im Moment nicht betreut, wenn entweder der Angestellte diesen Fan-Club überhaupt nie betreut hat oder wenn das heutige Datum (= sysdate) außerhalb des Betreuungszeitraums liegt. Vergessen Sie nicht, jene Fan-Clubs zu berücksichtigen, die von überhaupt keinem Angestellten betreut werden (dieser Fall sollte zwar laut Datenmodell nicht vorkommen. Die Einhaltung dieser Bedingung wird aber vermutlich vom Datenbanksystem nicht überprüft)!

S2.) (Die eifrigsten Angestellten) Schreiben Sie eine SQL-Anfrage, die den Nachnamen und die Personalnummer jener Angestellten ausgibt, die im Moment sämtliche Fan-Clubs betreuen. Ordnen Sie die Nachnamen alphabetisch.
Bemerkung: Passen Sie die Testdaten so an, dass diese Anfrage zumindest zwei Angestellte liefert.

S3.) (Spielereinsätze) Geben Sie für alle Spiele des Jahres 2015 jeweils alle Spieler und die Dauer ihres Einsatzes aus, d.h.: Gesucht sind alle Tupel (mannschaft, datum, vorname, nachname, dauer), mit folgender Eigenschaft:

"mannschaft" ist die Bezeichnung der Mannschaft, die gespielt hat.

"datum" ist das Datum, an dem das Spiel stattfand.

"vorname" und "nachname" beziehen sich auf einen Spieler, der bei diesem Spiel zum Einsatz kam.

"dauer" gibt die Dauer des Einsatzes (in Minuten) dieses Spielers bei diesem Spiel an.

S4.) (Spieler-Ranking) Geben Sie für jeden Spieler den Vornamen und Nachnamen sowie die Gesamtdauer ("gesamtdauer") der von ihm bei Spielen im Jahr 2015 geleisteten Einsätze aus. Vergessen Sie nicht, jene Spieler des Vereins zu berücksichtigen, die im Jahr 2015 bei keinem einzigen Spiel mitgespielt haben (d.h. gesamtdauer = 0). Ordnen Sie die Ausgabe in absteigender Gesamtdauer. Bei Gleichheit der Gesamtdauer sollen die Spieler in alphabetischer Reihenfolge (zuerst des Nachnamen, dann des Vornamen) sortiert werden.

S5.) (Der fleißigste Spieler) Geben Sie den Vornamen und Nachnamen jenes Spielers aus, von dem die unter b) berechnete Gesamtdauer am größten ist, d.h.: dieser Spieler ist bei Spielen im Jahr 2015 insgesamt am längsten im Einsatz gewesen. Falls sich mehrere Spieler den ersten Platz teilen (d.h. sie kommen auf die gleiche Gesamtdauer), dann sollen diese in alphabetischer Reihenfolge (zuerst des Nachnamen, dann des Vornamen) geordnet werden. Der Fall, dass im Jahr 2015 überhaupt kein Spiel stattfand, darf ignoriert werden.

Bemerkung: Berücksichtigen Sie bei Ihren Testdaten die Situation, dass sich zumindest 2 Spieler den ersten Platz teilen.

S6.) Schreiben Sie CREATE und DROP Befehle für eine View, die alle Informationen über Trainer aus der Personen- und Trainer-Tabelle zusammenfügt, d.h.: sowohl die allgemeinen Personendaten (Personalnummer, Vorname, Nachname, Geschlecht und Geburtsdatum) als auch die Trainer-spezifischen Informationen (Gehalt sowie Beginn und Ende der Vertragsdauer). In Summe ist also folgende View erforderlich:
Trainer_view (persnr, vname, nname, geschlecht, gebdat, gehalt, von, bis).

Datenbankclient (Java/C++) und DB-Connector (JDBC/libpqxx):

Schreiben Sie einen Client, der eine Datenbank-Verbindung herstellt. Realisieren Sie eine GUI (JavaFX/Qt), die das einfache Ändern (CRUD) der Spieler des Vereins erlaubt. Verwenden Sie dabei auf jeden Fall eine Tabelle (TableView, QTableView), die auch eine grafische Veränderung der Datensätze erlauben soll.

Ermöglichen Sie die gleichzeitige Verbindung von mehreren Clients auf die Datenbasis. Implementieren Sie dabei eine transaktionelle, gesicherte Erstellung und Änderung von Spielen. Beachten Sie dabei, dass der Spielstand und die Spielzeit der einzelnen Spieler laufend und von mehreren Clients gleichzeitig aktualisiert werden könnte. Stellen Sie für die Eingabe der Spielerzeit und Spielstand eine einfache grafische Möglichkeit zur Verfügung. Verwenden Sie dabei Transaktionen bzw. Locks und entsprechende programmtechnische Mittel um Inkonsistenzen zu vermeiden. Definieren Sie dabei für die einzelnen Informationen (Spielerzeit, Spielstand) eigene Threads.

Informationssysteme und Content Management:

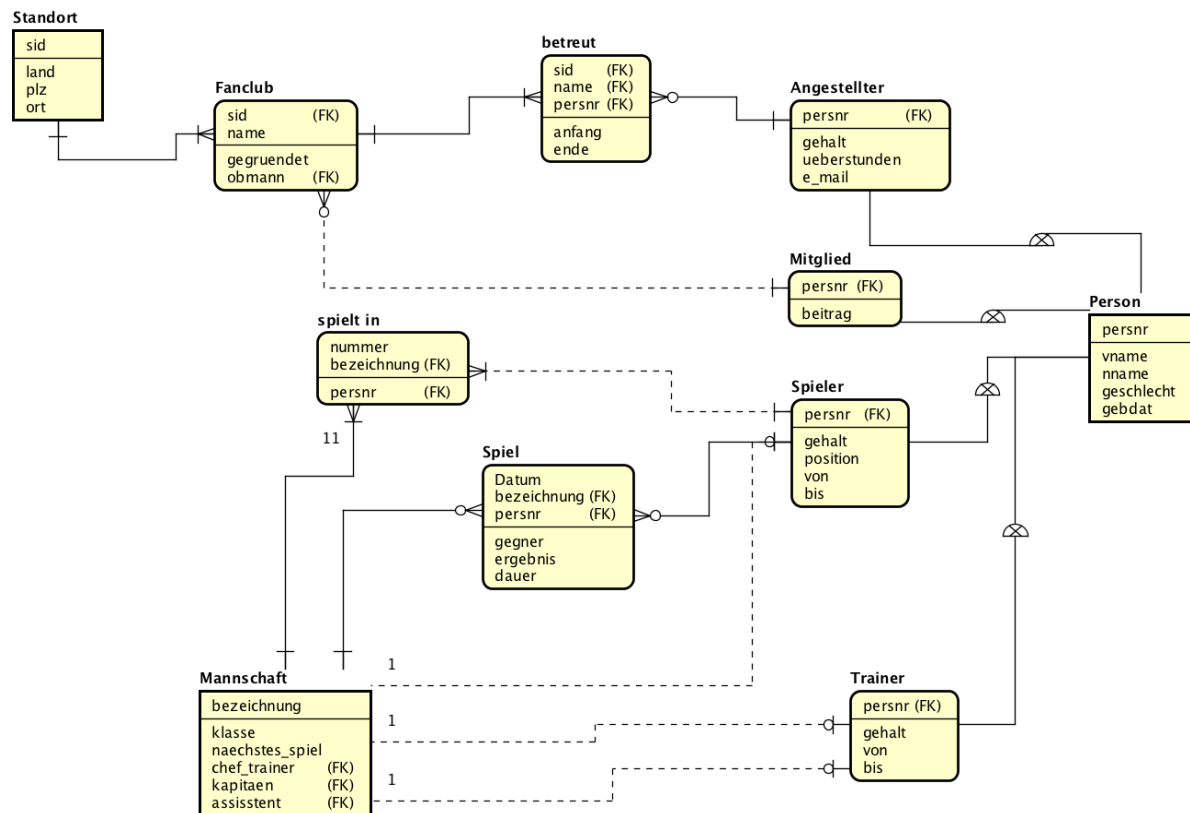
Versuchen Sie in einer kurzen schriftlichen Ausführung anzuführen, welche Informationssysteme im betrieblichen Umfeld genutzt werden und wie diese für die vorliegende Aufgabenstellung eingesetzt werden könnten. Im Bereich der Informationssysteme wurde im Unterricht im Speziellen auf die Thematik der Content Management Systeme eingegangen. Versuchen Sie auch hier kurz zu erläutern welche der kennengelernten CMS für die vorliegende Aufgabenstellung besonders gut eingesetzt werden könnten und warum.

Ausgehend von der vorangegangenen Analyse wählt ein entsprechendes Content Management aus, Implementiert ein passendes Template zu unserem Fußballverein und legt entsprechende Benutzer an die in eurem System erstellen, bearbeiten oder auch nur lesen können sollen. Holen Sie hierbei mittels angelegten Benutzer aus dem Content Management System Datensätze aus der Aufgabe Fußballverein (z.b. Spieler usw.) und zeigt diese entsprechend in eurem CMS an um das Design verdeutlichen sollen.

Setzen Sie das entwickelte Rahmenwerk "Fußballverein", das im Zuge der vorliegenden Arbeit generiert wurde, in Form eines Plug-Ins für das von Ihnen gewählte CMS um.“

[1]

2. Er-Diagramm



3. SQL-Script

3.1.Create - Script

Nachdem das Erd erfolgreich fertig war, wurde die Information in Astah realisiert.
Anschließend wurde das fertige File als SQL-File exportiert.(Ausschnitt SQL-File:)

3.2. Drop - Script:

Cascade = damit auch wirklich alles gelöscht wird.

```
4 DROP TABLE IF EXISTS Person CASCADE;  
5 DROP TABLE IF EXISTS Spieler CASCADE;  
6 DROP TABLE IF EXISTS Standort CASCADE;  
7 DROP TABLE IF EXISTS Trainer CASCADE;  
8 DROP TABLE IF EXISTS Angestellter CASCADE;  
9 DROP TABLE IF EXISTS Mannschaft CASCADE;  
10 DROP TABLE IF EXISTS Mitglied CASCADE;  
11 DROP TABLE IF EXISTS Spiel CASCADE;  
12 DROP TABLE IF EXISTS Spieltin CASCADE;  
13 DROP TABLE IF EXISTS Fanclub CASCADE;  
14 DROP TABLE IF EXISTS Betreuen CASCADE;
```


4. Datenbank erstellen

4.1. Postgres installieren

Zuerst wurde Postgres auf der VM erstellt, indem der

Befehl:

```
apt-get install postgres 9.5
```

einggegeben wurde.

4.2. Datenbank erstellen

Postgresql 4.5 installieren:

```
—> apt-get install postgresql 9.5
```

in Postgres einloggen:

```
—> su postgres
```

```
—> psql
```

Datenbank zeigen:

MySQL:

```
—> Show databases;
```

Postgres:

```
—> \l
```

Datenbank wechseln:

MySQL:

```
—> use dbname;
```

Postgres:

```
—> \c
```

Tabellen anzeigen:

MySQL:

```
—> show tables;
```

Postgres:

```
—> \dt;
```

Datenbank erstellen:

MySQL:

```
—> create database dbname;
```

Postgres:

—> create database dbname;

Datenbank löschen:

MySQL:

—> drop database dbname;

Postgres:

—> drop database dbname;

User erstellen:

Postgres:

—> create user username with password `password`;

Usern Rechte vergeben:

Postgres:

—> grant all privileges on database „dbname“ to username;

Mit User in eine Datenbank verbinden:

—> psql -h localhost -U fbuser -W -d fbverein

5. Datensätze generieren

Die Datensätze wurden mithilfe von Datafiller generiert.

Datafiller wird zum Generieren von Testdaten eingesetzt. Er basiert auf python.

[2]

Hie ein Beispiel:

```
CREATE TABLE Spieler ( -- df: mult=2000.00
  persnr SERIAL NOT NULL, -- df: offset=10000 shift=0 step=2 size=9999999999
  gehalt DECIMAL(10) NOT NULL, -- df: pattern='[1-9]{10}'
  position VARCHAR(30) NOT NULL, -- df: text=position length=1
  von DATE NOT NULL, -- df: start=1900-01-01 end=1950-01-01
  bis DATE NOT NULL -- df: start=1950-01-01 end=2006-01-01
);
```

Den Datafiller erkennt man durch das —df :

—df: mult=2000.00

```
CREATE TABLE Spieler ( -- df: mult=2000.00
```

Das mult=2000.00 sagt an wie viele Datensätze für die Tabelle Spieler generiert werden soll. Achtung der Wert nach = wird x100 gerechnet.

—df: offset=10000 shift=0 step=2 size=999999999

```
persnr SERIAL NOT NULL, -- df: offset=10000 shift=0 step=2 size=999999999
```

Das offset gibt an wo es anfängt, shift , step gibt an in wie vielen Schritten gezählt wird und size gibt das Ende an.

—df: pattern='[1-9]{10}'

```
gehalt DECIMAL(10) NOT NULL, -- df: pattern='[1-9]{10}'
```

Das [1-9] gibt an, dass die Ziffern von 1 bis 9 genommen werden dürfen. Das {10} gibt an wie viele Stellen es hat. In dem Beispiel kann so eine Zahl entstehen:

2451352353

—df: pattern= text=position length=1

```
position VARCHAR(30) NOT NULL, -- df: text=position length=1
```

Es wurde davor eine Liste mit Positionen erstellt. Da nimmt der Generator nur eine von den Positionen.

—df: start=1900-01-01 end=1950-01-01

```
von DATE NOT NULL, -- df: start=1900-01-01 end=1950-01-01
```

Start gibt an ab wann ein datum genommen werden soll und End gibt an bis wann das Datum genommen werden kann.

6. SQL-Abfragen

```
-- S1)
-- die benötigten Datensätze von der Tabelle betreut werden angezeigt, wenn das Datum stimmt
SELECT sid, name FROM betreut
WHERE persnr = 11
AND NOT CURRENT_DATE BETWEEN Betreuen.anfang AND Betreuen.ende;

-- S2)
-- mithilfe von NATURAL JOIN können auch Datensätze von einer anderen Tabelle abgerufen werden
SELECT Person.nname, Person.persnr FROM Betreuen Natural Join person
WHERE CURRENT_DATE BETWEEN Betreuen.anfang AND Betreuen.ende;

-- S3)
-- mithilfe von NATURAL JOIN können auch Datensätze von einer anderen Tabelle abgerufen werden
SELECT Spiel.mannschaft, Spiel.datum, Person.vname, Person.nname, Spiel.dauer
FROM Spiel NATURAL JOIN Person
WHERE (SELECT EXTRACT(YEAR FROM Spiel.datum))=2015;

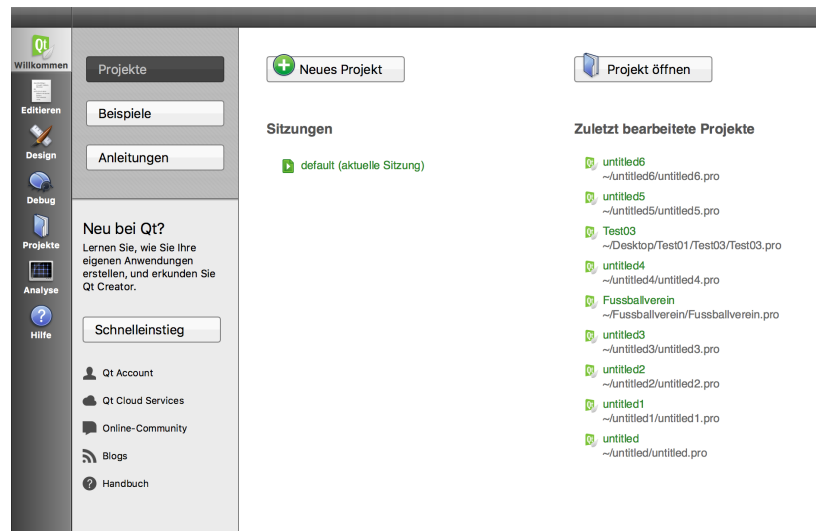
-- S6)
-- ein neuer View wird erstellt um Datensätze von Trainer und Person immer wieder einfach abrufen zu können
CREATE VIEW Trainer_view AS SELECT Person.persnr, Person.vname, Person.nname, Person.geschlecht, Person.gebdat, Trainer.gehalt, Trainer.von,
Trainer.bis FROM Trainer NATURAL JOIN Person;

-- die View wird nicht mehr benutzt und gelöscht
DROP VIEW Trainer_view;
```

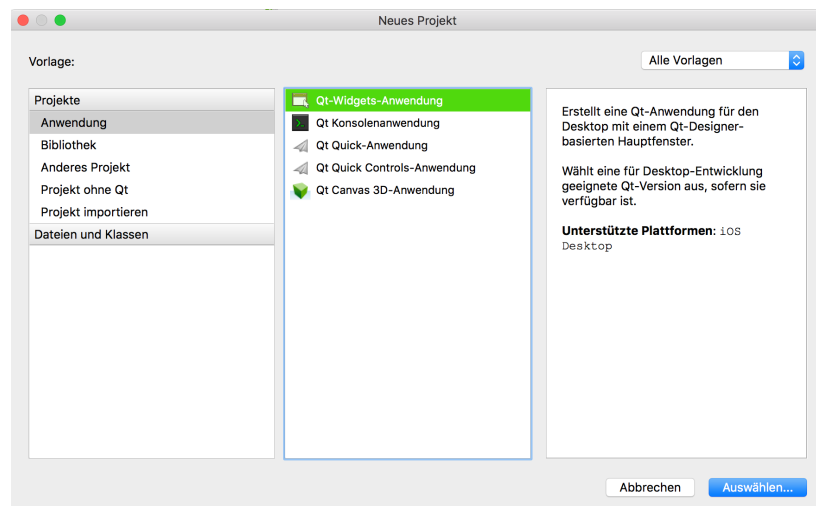
7. Qt

Zuerst wurde ein Qt Projekt durch die folgenden Schritte erstellt:

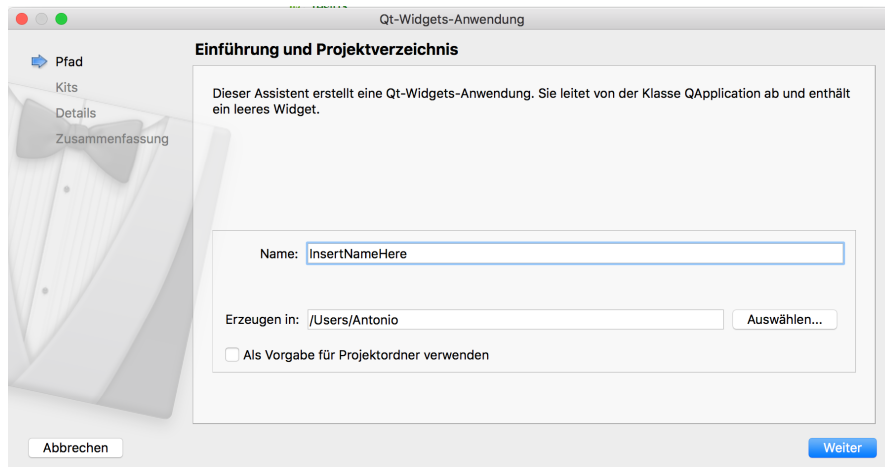
- Neues Projekt erstellen:



- QT-Widgets-Anwendung auswählen:



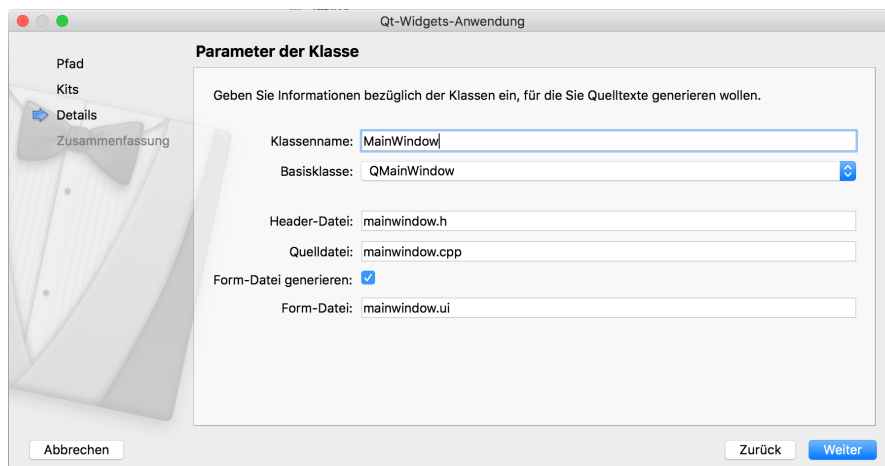
- Name und Pfad eingeben:



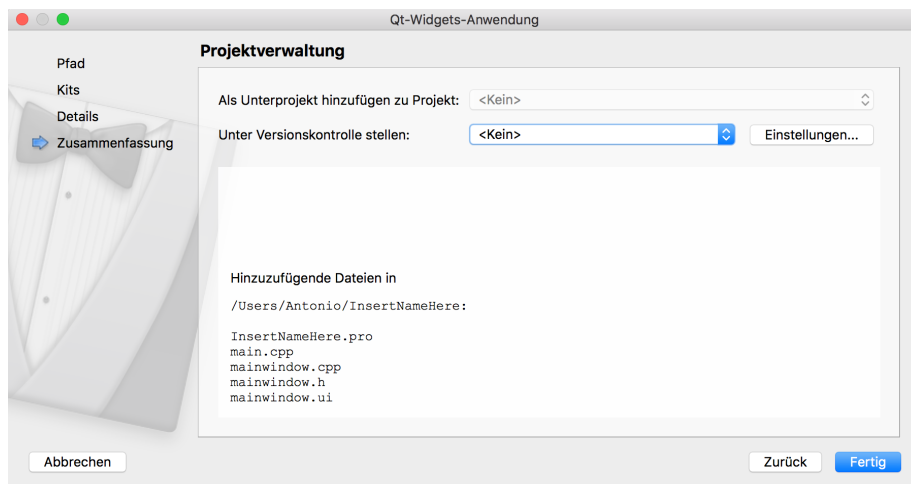
- Kit auswählen:



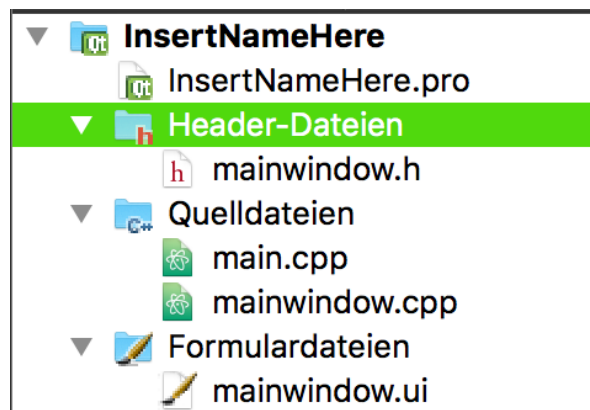
- Klassennamen eingeben:



- Zusammenfassung alles Dokumente:



- So sieht der Ordner aus:



7.1.Einführung QT

Nun zeige ich kurz wie QT Aufgebaut ist:

Als Erstes kommt das .pro File. Das ist wichtig, weil darin alle benötigten Libraries eingefügt werden.

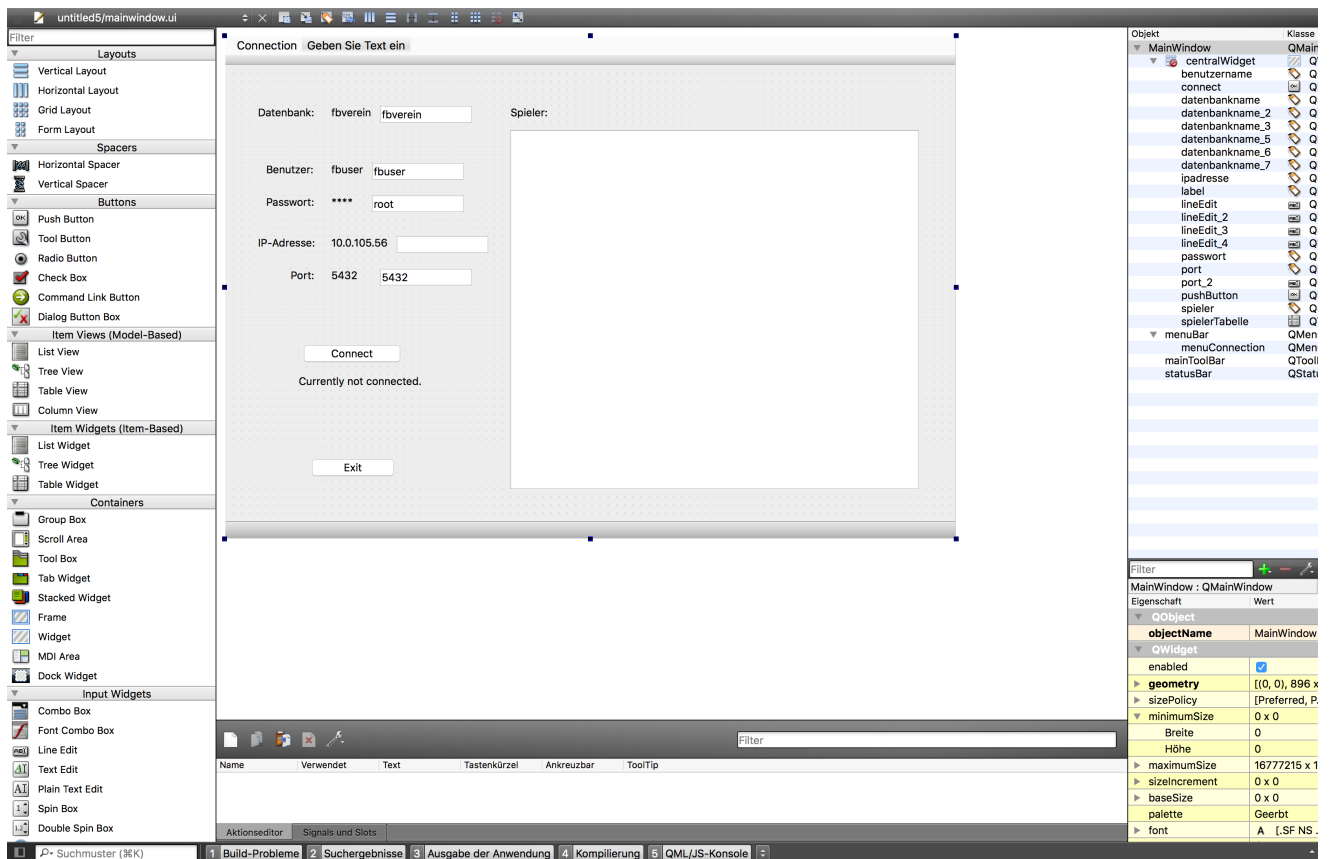
So sieht es bei mir aus:



```
1 #-----
2 #
3 # Project created by QtCreator 2016-04-17T22:30:37
4 #
5 #-----
6
7 QT      += core gui
8
9 greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
10
11 TARGET = Test03
12 TEMPLATE = app
13
14
15 SOURCES += main.cpp\
16           mainwindow.cpp
17
18 HEADERS += mainwindow.h
19
20 FORMS    += mainwindow.ui
21
22
23 #Libs
24 LIBS += -L"/usr/lib/"
25
26 unix|win32: LIBS += -lpqxx-4.0
27
28
29 win32:CONFIG(release, debug|release): LIBS += -L$$PWD/../../../../usr/local/Cellar/libpqxx/4.0.1/lib/release/ -lpqxx-4.0
30 else:win32:CONFIG(debug, debug|release): LIBS += -L$$PWD/../../../../usr/local/Cellar/libpqxx/4.0.1/lib/debug/ -lpqxx-4.0
31 else:unix: LIBS += -L$$PWD/../../../../usr/local/Cellar/libpqxx/4.0.1/lib/ -lpqxx-4.0
32
33 INCLUDEPATH += $$PWD/../../../../usr/local/Cellar/libpqxx/4.0.1/include
34 DEPENDPATH += $$PWD/../../../../usr/local/Cellar/libpqxx/4.0.1/include
35
36 INCLUDEPATH += $$PWD/../../../../usr/local/Cellar/libpqxx/4.0.2/include
37 DEPENDPATH += $$PWD/../../../../usr/local/Cellar/libpqxx/4.0.2/include
38 |
```


Ein weiteres wichtiges File ist das .ui File. Darin wird die GUI strukturiert.

So sieht es bei mir aus:



Zu letzt eines der wichtigsten Files: Das cpp File, wo die ganze Funktion stattfindet.

So sieht es bei mir aus:

```

untitled5/mainwindow.cpp

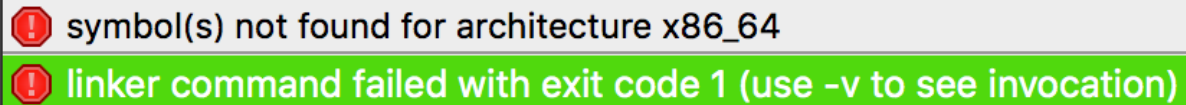
1  #include "mainwindow.h"
2  #include "ui_mainwindow.h"
3  #include <iostream>
4  #include <pqxx/pqxx>
5  #include <string>
6  #include <QTextEdit>
7
8  using namespace std;
9  using namespace pqxx;

33 void MainWindow::on_connect_clicked()
34 {
35     char * sql;
36     QString p2 = ui->port_2->text();
37     QString ip2 = ui->lineEdit_2->text();
38     QString pw2 = ui->lineEdit_3->text();
39     QString usr2 = ui->lineEdit_4->text();
40     QString dbn = ui->lineEdit->text();
41     QString s = "dbname="+dbn+" user="+usr2+" password="+pw2+" hostaddr="+ip2+" port="+p2;
42
43     // Either this if you use UTF-8 anywhere
44     std::string utf8_text = s.toUtf8().constData();
45
46     connection C(utf8_text);
47     if (C.is_open()) {
48         cout << "Opened database successfully: " << C.dbname() << endl;
49         ui->label->setText("Database connected!");
50     } else {
51         cout << "Can't open database" << endl;
52     }
53
54     /* Create SQL statement */
55     sql = "SELECT * from spieler";
56
57     /* Create a non-transactional object. */
58     nontransaction N(C);
59
60     /* Execute SQL query */
61     result R( N.exec( sql ));
62
63     /* List down all the records */
64     for (result::const_iterator c = R.begin(); c != R.end(); ++c) {
65         cout << "persnr = " << c[0].as<int>() << endl;
66         cout << "gehalt = " << c[1].as<double>() << endl;
67         cout << "position = " << c[2].as<string>() << endl;
68         cout << "von = " << c[3].as<string>() << endl;
69         cout << "bis = " << c[4].as<string>() << endl;
70     }
71 }
72

```

7.2.Qt Probleme

Leider ist beim Ausführen des Qt-Projekt der folgende Fehler aufgetreten:

A screenshot of a Qt error message. It consists of two lines. The first line has a red circular icon with a white exclamation mark, followed by the text "symbol(s) not found for architecture x86_64". The second line also has a red circular icon with a white exclamation mark, followed by the text "linker command failed with exit code 1 (use -v to see invocation)".

symbol(s) not found for architecture x86_64
linker command failed with exit code 1 (use -v to see invocation)

Auch durch suchen im Internet konnte das Problem nicht gelöst werden. Die folgenden Schritte habe ich ausprobiert um das Problem zu beheben:

- ein komplett neues Projekt erstellt und Daten übertragen
- Library geändert (hat funktioniert, doch bei mehrmaligen Ausführen hat es nicht mehr funktioniert)
- in Internet Foren wurde vorgeschlagen, die DEVELOPMENT-TARGET zu ändern was leider auch keinen Erfolg brachte
- Im Menü unter *Erstellen* -> *alles bereinigen* ausführen, qMake ausführen und anschließend das Projekt ausführen.

Da alle diese Versuche keinen Erfolg gebracht haben, habe ich beschlossen meine Fehler aufzulisten und meine Versuche dieses Problem zu beheben zu dokumentieren.

8. Quellen und Referenzen

[1] „Michael Borko,“ [Online]. Available: <https://elearning.tgm.ac.at/mod/assign/view.php?id=48203> [Zugriff am 30.Mai 2016]

[2] „Datafiller,“ [Online]. Available: <https://www.cri.ensmp.fr/people/coelho/datafiller.html> [Zugriff am 30.Mai 2016]