Ergänzungsmaterial zum Lehrplannavigator

Relationale Datenbanken in Softwareprojekten

In den vorliegenden Materialien soll ein Projekt vorgestellt werden, das die Einbindung einer relationalen Datenbank in ein Softwareprodukt verdeutlicht. Zur Ansprache des Datenbanksystems kommen die didaktischen Klassen DatabaseConnector und QueryResult zum Einsatz.[[1]](#footnote-1)

# Projektskizze

Im Folgenden wird ein mögliches Endprodukt des Projekts in Form einer Modelllösung skizziert. Im Sinne eines schülerorientierten Unterrichts kann und sollte im Austausch mit der Lerngruppe davon abgewichen werden.

## Projektidee

Ziel des vorliegenden Projekts ist es, ein datenbankgestütztes *Quizspiel* zu entwickeln, bei dem Spieler Aufgaben aus einer zentralen Datenbank abrufen und mit Hilfe individueller Benutzerprofile ihren Spielerfolg langfristig verfolgen können.

Im Einzelnen sollen die folgenden Anforderungen erfüllt werden:

1. **Benutzerprofile**: Jeder Spieler soll sich über einen Benutzernamen und ein Passwort anmelden können. Eine Anmeldung kann lokal oder über ein Netzwerk erfolgen.
2. **Aufgabenformat**: Jeder Spieler kann für einen beliebigen Zeitraum Aufgaben spielen, die aus einer Frage und vier Antwortoptionen bestehen. Eine Antwortoption ist richtig.
3. **Erfolgsstatistik**: Jeder Spieler kann eine Erfolgsstatistik einsehen. Diese besteht aus zwei Teilen. Zum einen kann eingesehen werden, wie viele Aufgaben im aktuellen Spiel bereits gestellt wurden und wie viele bzw. wie viel Prozent davon korrekt beantwortet wurden. Zum anderen kann dieselbe Information bezogen auf alle Spiele des aktuellen Benutzers abgefragt werden.
4. **Intelligente Aufgabenauswahl**: Damit das Spiel möglichst lange motivierend bleibt, soll ein Spieler immer neue Aufgaben gestellt bekommen. Sind alle Aufgaben der Datenbank abgearbeitet, werden diejenigen gestellt, welche am seltensten von ihm bearbeitet wurden.

## Benutzungsoberfläche

Die Benutzungsoberfläche wird mit Hilfe von *SWING*-Komponenten und einem *GUI-Builder* realisiert. Anzuraten ist, den Schülerinnen und Schülern das Design der Benutzeroberfläche zu überlassen und anschließend einen Prototypen entsprechend dem Design der Schülerinnen und Schüler vorzugeben. Die vorliegende Modelllösung beinhaltet die folgende Oberfläche:



Abbildung 1: Anmeldebildschirm

Der *Anmeldebildschirm* (vgl. Abbildung 1) erscheint beim Start des Programms und lässt den Spieler seinen Benutzernamen und sein Passwort eingeben. Auf der rechten Seite ist ein *Steuer- bzw. Informationsbereich* zu sehen, der auch beim *Spielbildschirm* (vgl. Abbildung 2) vorhanden ist. In diesem Bereich kann das Spiel mit dem Knopf *Beenden* verlassen werden, es kann eine Information über das Programm mit dem Knopf *Info* abgerufen werden, und es sind Spieldaten abzulesen, die zunächst alle Nullwerte zeigen. Ist die Anmeldung erfolgreich, d.h. werden beim Druck auf den Knopf *Anmelden* entsprechende Anmeldeinformationen in der Datenbank gefunden, wechselt das Spiel zum *Spielbildschirm* (vgl. Abbildung 2).

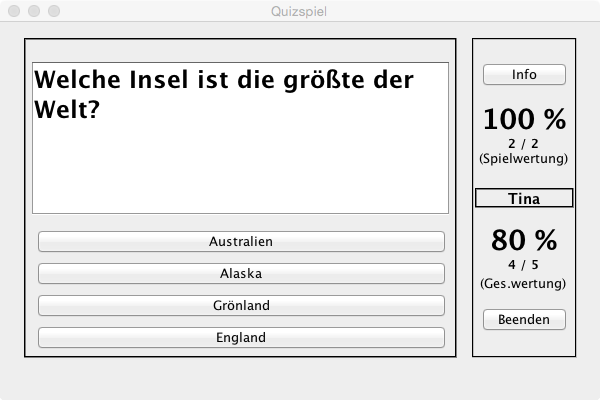


Abbildung 2: Spielbildschirm

Im *Spielbildschirm* wird am oberen Ende die aktuelle Frage angezeigt, darunter die vier Antwortoptionen auf jeweils einem Knopf. Wird eine Antwort angeklickt, wird der Knopf grün eingefärbt, wenn die Antwort korrekt ist, ansonsten rot. Bei einer falschen Antwort wird zugleich die richtige Antwort grün markiert. Nach einer Wartezeit von ca. zwei Sekunden wird automatisch zur nächsten Aufgabe gewechselt.

Im *Steuer- bzw. Informationsbereich* wird am oberen Ende angezeigt, wie viel Prozent der Fragen im aktuellen Spiel, d.h. seit Programmstart, richtig beantwortet wurden. Darunter ist die konkrete Anzahl der Fragen aufgeführt. In der Mitte ist der aktuelle Benutzer abzulesen. Darunter wird angegeben, wie viel Prozent der Fragen der Benutzer inklusive aller vergangenen Spiele richtig beantwortet hat. Auch hier werden darunter die konkreten Anzahlen an Aufgaben angegeben.

In Abbildung 2 spielt also *Tina* ein Spiel, in dem sie 100 %, d. h. von zwei Fragen zwei, korrekt beantwortet hat. Insgesamt hat sie 80 %, d. h. vier von fünf Aufgaben richtig bearbeitet. Da im aktuellen Spiel nur zwei Aufgaben gestellt wurden, entfallen drei dieser Fragen auf frühere Spiele.

## Modellierung der relationalen Datenbank

Um das Spiel umzusetzen, wird in der vorliegenden Modelllösung eine relationale Datenbank mit Hilfe von MySQL realisiert. Sie wird wie folgt modelliert:

Modellierung-Quizspiel

Abbildung 3: Modellierung der relationalen Datenbank

Die Modellierung aus Abbildung 3 wird mit dem folgenden Datenbankschema umgesetzt:

**Spieler**(SpielerID, Benutzername, Passwort)

**Aufgabe**(AufgabeID, Frage, KorrekteAntwort, FalscheAntwortA,

FalscheAntwortB, FalscheAntwortC)

**hatBearbeitet**(↑SpielerID, ↑AufgabeID, AnzahlBearbeitungen,

AnzahlKorrekteBearbeitungen)

Die Datenbank besteht aus drei Tabellen: Spieler, Aufgabe und der Verknüpfungstabelle hatBearbeitet.

In der Tabelle Spieler werden die Anmeldeinformationen jedes zugelassenen Spielers in Form von Benutzernamen und Passwörtern gespeichert. In der Tabelle Aufgabe sind die einzelnen Aufgaben mit Frage und vier Antwortoptionen zu finden, wobei eine Antwort die korrekte ist.

Die Verknüpfungstabelle hatBearbeitet ist beim ersten Spielstart zunächst leer. Bearbeitet ein Spieler eine Aufgabe, wird dort ein Eintrag gemacht, in dem gespeichert ist, dass die Aufgabe bearbeitet wurde und ob sie korrekt bearbeitet wurde. Da bei längeren Spielzeiten eine Aufgabe auch mehrmals bearbeitet werden kann, wird hier kein Boolescher Wert gespeichert, sondern jeweils die Anzahl der Bearbeitungen bzw. korrekten Bearbeitungen.

Ergänzungen oder Änderungen in den Tabellen Spieler und Aufgabe sind in der hier vorgestellten Modelllösung nicht vorgesehen. Neue Spieler und Fragen können z. B. mit Hilfe der Import-Funktion von MySQL aus einer Tabellenkalkulation oder einer CSV-Datei ergänzt werden.

## Modellierung des Programms

Bei der Modellierung des Programms wurde nach dem MVC-Prinzip vorgegangen. Die Klasse GUI stellt die Benutzunsgoberfläche entsprechend der in Abschnitt 1.2 gegebenen Beschreibung zur Verfügung. Der eigentliche Datenbestand des Programms befindet sich in der Datenbank, die mit Hilfe der Klassen DatabaseConnector und QueryResult angesprochen werden kann. Die Steuerung wird durch die Klassen Quizspiel und Aufgabe realisiert. Folgendes Implementationsdiagramm liegt der Modelllösung zu Grunde:



Abbildung 4: Teilmodellierung des Quizspiels

Die Klasse GUI verwendet beim Eintreten der entsprechenden GUI-Ereignisse die Methoden des Objekts quizspiel. Das Objekt aufgabenWechsler vom Typ Timer wird verwendet, um nach der Bearbeitung einer Aufgabe ca. 2 Sekunden zu warten, bis eine neue Aufgabe mit Hilfe der Methode ladeAufgabe vom Objekt quizspiel abgefragt wird. Auf die genaue Realisierung der Benutzeroberfläche soll hier nicht weiter eingegangen werden.

Die Klasse Quizspiel beinhaltet die eigentliche Programmfunktionalität. Im Konstruktor der Klasse wird die Verbindung zu Datenbank mit Hilfe der Klasse DatabaseConnector aufgebaut. Wird dann die Methode anmelden aufgerufen, so wird eine SQL-Anweisung an die Datenbank geschickt, mit der abgefragt wird, ob die Anmeldedaten gültig sind (vgl. Abschnitt 1.5). Ist das der Fall, wird die ID des Benutzers in aktuelleSpielerID gespeichert und die Anfrage istAngemeldet wird nun true liefern.

Die Methode gibAufgabe liefert eine Aufgabe vom Typ Aufgabe, die der Spieler als nächstes beantworten soll. Dazu wird zunächst geprüft, ob die Liste offeneAufgaben Aufgaben enthält. Ist das der Fall, wird eine zufällige Aufgabe aus dieser Liste genommen. Ist die Liste offeneAufgaben leer – wie z.B. beim Programmstart – dann werden mit Hilfe einer SQL-Anweisung (vgl. Abschnitt 1.5) die 20 vom aktuellen Benutzer am seltensten bzw. noch nie bearbeiteten Aufgaben aus der Datenbank abgerufen und in diese Liste übertragen. Anschließend wird aus ihnen eine Zufallsaufgabe gewählt.

Mit Hilfe der Methode abgebenAufgabe kann eine Aufgabe, die der Spieler beantwortet hat, weiter verarbeitet werden. In erster Linie wird in dieser Methode die Bearbeitung der Aufgabe mit Hilfe von zwei SQL-Anweisungen in die Datenbank übertragen, d.h. es wird vermerkt, dass die Aufgabe bearbeitet wurde und ob sie korrekt bearbeitet wurde. Dazu wird ein Eintrag in der Verknüpfungstabelle hatBearbeitet erstellt oder angepasst (vgl. Abschnitt 1.5).

Darüber hinaus können von einem Objekt der Klasse Quizspiel die statistischen Daten des Spielverlaufs abgefragt werden, worauf hier nicht weiter eingegangen werden soll.

Ein Objekt der Klasse Aufgabe speichert eine einzelne Aufgabe mit ID, Frage, Antworten und ob sie korrekt bearbeitet wurde.

## Verwendete SQL-Anweisungen

Um das Projekt zu realisieren, wurden die folgenden SQL-Anweisungen erstellt. Die in Winkelklammern ( < und > ) stehenden Ausdrücke werden von der jeweiligen Java-Methode abhängig vom Anwendungsfall durch zutreffende Zeichenfolgen ersetzt.

**SQL-Anweisung: Anmeldung**

SELECT SpielerID

FROM Spieler

WHERE Benutzername= <Benutzer> AND Passwort = <Passwort>

Die Anweisung wird von der Methode anmelden der Klasse Quizspiel verwendet, um die ID des Benutzers zu ermitteln, den der Spieler mit <Benutzer> und <Passwort> eingegeben hat. Liefert diese Anweisung ein nicht leeres Ergebnis, war die Anmeldung erfolgreich, da es offenbar einen passenden Eintrag gibt. Die ermittelte ID wird in diesem Fall als aktuelleSpielerID gespeichert. Ansonsten wird die Anmeldung mit einer Fehlermeldung abgebrochen.

**SQL-Anweisung: Statistikdaten**

SELECT SUM(hatBearbeitet.anzahlBearbeitungen),

SUM(hatBearbeitet.anzahlKorrekteBearbeitungen)

FROM hatBearbeitet

WHERE hatBearbeitet.SpielerID = <aktuelleSpielerID>

Die Anweisung wird ebenfalls von der Methode anmelden der Klasse Quizspiel verwendet. Sie ermittelt aus der Verknüpfungstabelle hatBearbeitet die Summe der bearbeiteten Aufgaben und die Summe der korrekt bearbeiteten Aufgaben des aktuellen Spielers mit der ID <aktuelleSpielerID>.

**SQL-Anweisung: Aufgaben ermitteln**

SELECT Aufgabe.AufgabeID, Aufgabe.Frage,

Aufgabe.KorrekteAntwort, Aufgabe.FalscheAntwortA,

Aufgabe.FalscheAntwortB, Aufgabe.FalscheAntwortC

FROM Aufgabe LEFT JOIN

(

SELECT hatBearbeitet.AufgabeID,

hatBearbeitet.anzahlBearbeitungen

FROM hatBearbeitet

WHERE hatBearbeitet.SpielerID = <aktuelleSpielerID>

) AS tmp ON Aufgabe.AufgabeID = tmp.AufgabeID

ORDER BY tmp.anzahlBearbeitungen

Die Anweisung wird von der Methode gibAufgabe der Klasse Quizspiel verwendet, um neue Aufgaben aus der Datenbank zu laden, wenn in der Liste offeneAufgaben keine mehr vorhanden sind.

Die Unterabfrage von Zeile 4 bis 6 erstellt eine Ergebnistabelle, in der die ID-Nummern aller Aufgaben und deren Anzahl an Bearbeitungen aufgeführt sind, die zum aktuellen Spieler gehören. Es werden nur Aufgaben aufgeführt, die vom aktuellen Spieler schon einmal bearbeitet wurden.

Die äußere SQL-Anweisung liefert AufgabeID, Frage und alle Antwortoptionen zu allen Fragen und ordnet ihnen mit einem LEFT JOIN die Anzahl der Bearbeitungen durch den aktuellen Spieler zu. Nicht vom aktuellen Spieler bearbeitete Aufgaben bekommen den Wert null zugewiesen. Die Liste wird nach der Anzahl der Bearbeitungen aufsteigend sortiert, d.h. diejenigen mit dem Eintrag null stehen am Anfang.

Die Methode gibAufgabe entnimmt nun die ersten 20 Aufgaben aus dieser Liste, d.h. die 20 am seltensten bzw. noch nie durch den aktuellen Spieler bearbeiteten Aufgaben. Diese Aufgaben werden in der Liste offeneAufgaben gespeichert und in einer zufälligen Reihenfolge dem Spieler präsentiert.

**SQL-Anweisung: Test auf einen Bearbeitungseintrag**

SELECT Count(hatBearbeitet.SpielerID)

FROM hatBearbeitet

WHERE hatBearbeitet.SpielerID = <aktuelleSpielerID> AND

hatBearbeitet.AufgabeID = <AufgabeID>

Die Anweisung wird von der Methode abgebenAufgabe der Klasse Quizspiel verwendet, um zu prüfen, ob ein Spieler eine Aufgabe schon einmal bearbeitet hat. Davon ist abhängig, ob in der Verknüpfungstabelle ein neuer Eintrag erstellt werden muss, oder ob ein existierender Eintrag geändert werden muss.

**SQL-Anweisung: Eintragen einer Aufgabenbearbeitung**

INSERT INTO hatBearbeitet(SpielerID, AufgabeID,

anzahlBearbeitungen, anzahlKorrekteBearbeitungen)

VALUES (<aktuelleSpielerID>, <AufgabeID>, 1, <1 oder 0>)

Die Anweisung wird von der Methode abgebenAufgabe der Klasse Quizspiel verwendet, um die Bearbeitung einer Aufgabe in die Verknüpfungstabelle hatBearbeitet einzutragen, sofern noch kein Bearbeitungseintrag für die Aufgabe <AufgabeID> und den aktuellen Spieler mit <aktuelleSpielerID> vorhanden ist. Die Anweisung besteht aus einem entsprechenden INSERT-Befehl. Der Eintrag der Spalte anzahlKorrekteBearbeitungen ist entweder 0 oder 1, je nach dem, ob die Aufgabe korrekt beantwortet wurde. Ein höherer Wert als 1 ist nicht möglich, da es sich zwangsläufig um die erste Bearbeitung der Aufgabe handeln muss.

**SQL-Anweisung: Abfragen der Bearbeitungsanzahlen**

SELECT hatBearbeitet.anzahlBearbeitungen,

hatBearbeitet.anzahlKorrekteBearbeitungen

FROM hatBearbeitet

WHERE hatBearbeitet.SpielerID = <aktuelleSpielerID> AND

hatBearbeitet.AufgabeID = <AufgabeID>

Die Anweisung wird von der Methode abgebenAufgabe der Klasse Quizspiel verwendet, wenn eine Bearbeitung zu einer Aufgabe eingetragen werden soll, die schon zuvor mindestens einmal vom aktuellen Spieler bearbeitet wurde. Zu diesem Zweck müssen die alten Bearbeitungszahlen ermitteln werden, damit sie korrigiert und wieder in die Datenbank überspielt werden können.

**SQL-Anweisung: Eintragen aktualisierter Bearbeitungsanzahlen**

UPDATE hatBearbeitet SET

anzahlBearbeitungen = <aktualisierteAnzahlBearbeitungen>,

anzahlKorrekteBearbeitungen =

<aktualisierteAnzahlKorrekteBearbeitungen>

WHERE hatBearbeitet.SpielerID = <aktuelleSpielerID> AND

hatBearbeitet.AufgabeID = <AufgabeID>

Die Anweisung wird von der Methode abgebenAufgabe der Klasse Quizspiel verwendet, um aktualisierte Bearbeitungsanzahlen zu einer Aufgabe in die Verknüpfungstabelle hatBearbeitet einzutragen. Dazu wird eine UPDATE-Anweisung verwendet.

# Materialien

1. Modellloesung\_Quizspiel
2. SQL-Beispieldatenbank

1. Schnittstellendokumentation und verschiedene Beispielimplementierungen zu diesen Klassen stehen auf den Seiten von QUA-LiS zur Verfügung. [↑](#footnote-ref-1)