

# Table of Contents

Praktische Aufgabenstellung: Jan Binder .....	1
Back End: Datenbank und Datenbankverbindung .....	1
Begin .....	1
Grundlagen .....	1
Getting Started .....	2

# Praktische Aufgabenstellung: Jan Binder

## Back End: Datenbank und Datenbankverbindung

### Begin

First of all the basics, like the technology we are using and the design are set. The database, in which all the tables and data sets are, is also appointed.

### Grundlagen

You need some foreknowledge to build a database and a database connectivity. Knowledge about SQL, JPA and Java is required. You need these languages to create queries or to understand how the program communicates with the database.

### SQL

SQL is short for "Structured Query Language". It is used to insert, change and delete data sets or create or delete tables.

*Beispielcode von SQL*

```
①
CREATE TABLE mitarbeiter (
    id integer,
    nachname text,
    vorname text,
    gehalt integer
);

②
INSERT INTO mitarbeiter (id, nachname, vorname, gehalt)
VALUES (1, Mustermann, Max, 2000);

③
SELECT * FROM mitarbeiter;

④
ALTER TABLE mitarbeiter ADD CONSTRAINT "MITARBEITER_pkey" PRIMARY KEY ("id");
```

- ① All this code does is it creates a new table called "mitarbeiter" which contains the columns "id", "nachname", "vorname" and "gehalt". Every column has its datatype next to it. Optionally there are constraints to the columns, such as "NOT NULL". which tells the database that this column cannot be empty or "UNIQUE", which means that a data set must be unique in this column.

- ② The next command is INSERT INTO. This code tells the database to write data into the table. First you need to set the table with all its columns and then the values like shown.
- ③ The "SELECT \* FROM mitarbeiter;" command picks out every data set from the table "mitarbeiter". This is done by the '\*' parameter. Further there is a possibility to only select a few specific data sets.
- ④ This command lets you change an already existing table. In this example I updated the tables constraints, in detail I set the primary key of the table.

## JPA

Applications are made up of business logic, interaction with other systems, user interfaces and data. Most of the data that our applications manipulate have to be stored in databases, retrieved, and analyzed. Databases are important: they store business data, act as a central point between applications, and process data through triggers or stored procedures. Persistent data are everywhere, and most of the time they use relational databases as the underlying persistence engine (as opposed to schemaless databases). Relational databases store data in tables made of rows and columns. Data are identified by primary keys, which are special columns with uniqueness constraints and, sometimes, indexes. The relationships between tables use foreign keys and join tables with integrity constraints.

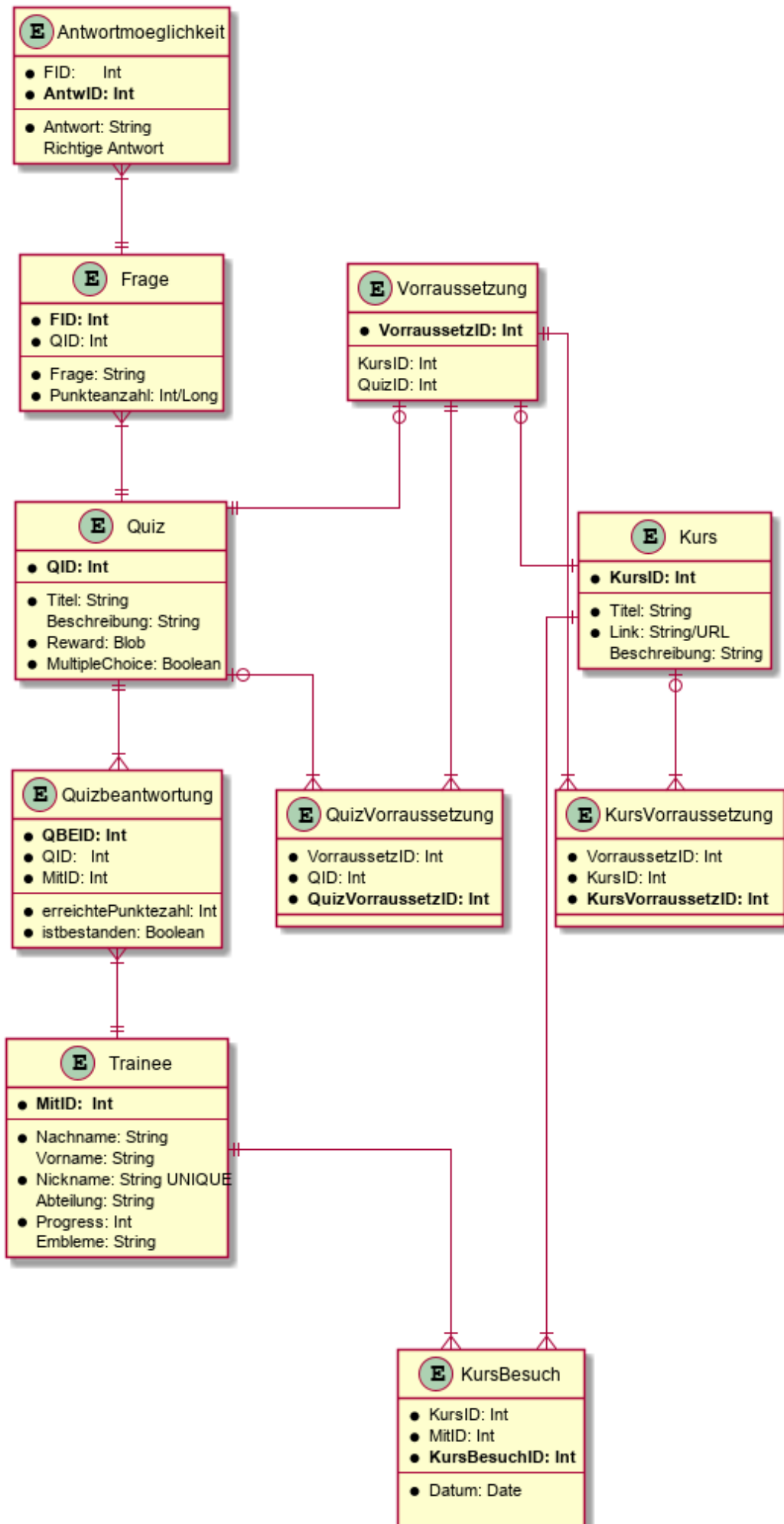
All this vocabulary is completely unknown in an object-oriented language such as Java. In Java, we manipulate objects that are instances of classes. Objects inherit from others, have references to collections of other objects, and sometimes point to themselves in a recursive manner. We have concrete classes, abstract classes, interfaces, enumerations, annotations, methods, attributes, and so on. Objects encapsulate state and behavior in a nice way, but this state is only accessible when the Java Virtual Machine (JVM) is running: if the JVM stops or the garbage collector cleans its memory content, objects disappear, as well as their state. Some objects need to be persistent. By persistent data, I mean data that are deliberately stored in a permanent form on magnetic media, flash memory, and so forth. An object that can store its state to get reused later is said to be persistent. The principle of object-relational mapping (ORM) is to bring the world of database and objects together. It involves delegating access to relational databases to external tools or frameworks, which in turn give an objectoriented view of relational data, and vice versa. Mapping tools have a bidirectional correspondence between the database and objects. Java Persistence API (JPA) is the preferred technology and is part of Java EE 7.

(vgl. *Beginning Java EE-Java Persistence API*, Antonio Goncalves, 2013)

## Getting Started

The first task was to visually showcase how all the tables needed, connect with each other, so we can understand what we need to do, and how everything should look and interact. Here we used the diagram called ER-Modell which does exactly that for us.

## ER-Modell



The ER-Model in general is a diagram, which shows the relationships between the entities of a database.

In this chapter I will showcase all the tables with their attributes and their respective meanings. Starting from the bottom to the top.

## **Trainee**

The first table is the table "Trainee". This table contains all trainees. It contains their first name, last name, the nickname they use on the website, their department in the company and their progress they have by doing quizzes. Also every Trainee can get emblems by completing quizzes. It has a One-To-Many relationship to the table "Quizbeantwortung" and to the table "KursBesuch".

## **KursBesuch**

The entity "KursBesuch" is a table, which splits the many-to-many relationship between "trainee" and "kurs" into two separate One-To-Many relationships therefore its relationships are Many-To-One to "Trainee" and to "Kurs". That means it has the PK (primary key) of both tables as a foreign key. It also provides a column named "datum" which indicates the date the user has visited the course. The function of this entity is to show which trainee has done which course and when.

## **KursVoraussetzung**

The function of "KursVoraussetzung" is that a course can have a course or a quiz as a requirement to take this course. This table contains courses and their required quizzes or courses. Therefore it contains the "KursVoraussetzID", the "KursID" and the "VoraussetzID" as columns.

## **QuizVoraussetzung**

This table has the same function as "KursVoraussetzung", but instead of courses with quizzes. So this table contains quizzes and the requirements to take this quiz, which can be courses or previous quizzes.

## **Quizbeantwortung**

This entity is essential for the implementation of the "Voraussetzungs-Logik". We need this table to check if a trainee has already done a quiz. This is done with the column "istbestanden". Also it stores the points a trainee has achieved in the quiz. It also splits up a Many-To-Many relationship between "Quiz" and "Trainee" into two One-To-Many relationships.

## **Kurs**

Here is the Kurs-Entity, it has the primary key "KursID", which helps to find a specific course from the table. The table also has a column "titel" which is basically the title of this course. It also has the constraint NOT NULL so if there is a new course there must be a title as well. The next column is the "link" it contains the links of the courses so the visitors can be redirected to the page with the explanation of the topic. This cannot be empty too. The last column is "beschreibung", which is the description of the course. This column can be empty, but it is not recommended since it helps the user specify which course contains what information.

## Quiz

This is the table for all quizzes. It contains the "QID", which is the primary key, a title, a description and a reward. The title has a constraint called NOT NULL because every quiz must have a specification which topic it has. There is also a option if the quiz is multiple choice. There are also so called queries to find data with specific parameters from the database. These queries are defined in the associated entity class.

## Voraussetzung

This table defines the requirement a course or a quiz can have. Therefor it contains a "KursId" and a "qid". The primary key of this table is then given to "KursVoraussetzung" or "QuizVoraussetzung" respectively.

## Frage

Every Quiz contains many questions this means that "Frage" and "Quiz" have a Many-To-One relationship. The questions are stored in the table "Frage". Each question has a "FID", a "QID", so it can be connected to a quiz. A question also has the questiontext itself and points you get for each question.

## Antwortmoeglichkeiten

Every question has 4 answers this means that there is a Many-To-One relationship between "Antwortmoeglichkeiten" and "Frage". These answers are stored in this table. Each answer is connected to its question. Also the answertext itself is stored here, as well as a boolean value if the answer is correct or not.

## EJBs

After knowing how to connect the tables with each other, the next step was to create so called EJBs for every table we needed one for.

EJBs are components that summarize the business logic and take care of transactions and security. It is basically a connection to the database.

They must contain an annotation called "Stateless", this means that after a methode is called it is terminated. This is the usual way to annotate a EJB.

Every EJB has a methode to find exactly one entity, persist/merge an entity or delete one entitiy from the database.

*KursEJB.java*

```
Unresolved directive in Praxis_Jan.adoc - include:../game-trainee-web/src/main
/java/org/game/trainee/kurs/KursEJB.java[lines=41..]
```

The main use of this EJB is to provide the courses page with all the courses within the database. This is done with the "findAll" methode. It is used on the courses page where all courses available are shown.

### *FrageEJB.java*

```
Unresolved directive in Praxis_Jan.adoc - include:../../../game-trainee-web/src/main
/java/org/game/trainee/quiz/FrageEJB.java[lines=21..]
```

The use of the "FrageEJB" is for example to find all questions connected to a qid. This is used to show the questions when you take a quiz. Also it is available to only find one question or all questions in form of a list.

### *QuizEJB.java*

```
Unresolved directive in Praxis_Jan.adoc - include:../../../game-trainee-web/src/main
/java/org/game/trainee/quiz/QuizEJB.java[lines=16..]
```

The function of this EJB is to find all quizzes from the database and return them in a list to the website. Also you can find single quizzes with the methode "find", you can delete and update quizzes.

### *TraineeEJB.java*

```
Unresolved directive in Praxis_Jan.adoc - include:../../../game-trainee-web/src/main
/java/org/game/trainee/trainee/TraineeEJB.java[lines=16..]
```

The use of this class is to provide all the trainees deposited on the database to the associated web page. As in every other EJB there is an opportunity to find, update or delete a trainee.

### *AntwortmoeglichkeitenEJB.java*

```
Unresolved directive in Praxis_Jan.adoc - include:../../../game-trainee-web/src/main
/java/org/game/trainee/quiz/AntwortmoeglichkeitenEJB.java[lines=16..]
```

This EJB provides the page where you take the quiz with all the corresponding answers to the questions. This is done by the methode "findAntwortenByFID" which starts a query when invoked. This query is defined in the "Antwortmoeglichkeiten" entity.

### *QuizVoraussetzungEJB.java*

```
Unresolved directive in Praxis_Jan.adoc - include:../../../game-trainee-web/src/main
/java/org/game/trainee/quiz/QuizVoraussetzungEJB.java[lines=13..]
```

The "QuizVoraussetzungEJB" allocates all "QuizVoraussetzungen" with the "findAllQuizVoraussetzungen" methode. It is used in the "Voraussetzungs-Logic".

### *VoraussetzungEJB.java*

```
Unresolved directive in Praxis_Jan.adoc - include:../../../game-trainee-web/src/main
/java/org/game/trainee/kurs/VoraussetzungEJB.java[lines=16..]
```

This EJB is used for the requirements logic. It contains a find methode where you can search for a requirement via a "VoraussetzID", which then returns a "Voraussetzung"-Entity.

*QuizbeantwortungEJB.java*

```
Unresolved directive in Praxis_Jan.adoc - include:../../../game-trainee-web/src/main
/java/org/game/trainee/quiz/QuizbeantwortungEJB.java[lines=13..]
```

The use of this bean is to provide information to all the quizzes a specific trainee has taken and when. This is done by a query, which is defined in the entity of this bean. It is required in the requirement logic.

## **Persistence.xml**

*persistence.xml*

The "persistence.xml" is used to configure many things, such as the source of the script used for dropping, if the database should always drop and then create all tables, and much more.

## **SQL-Files**

In this chapter I will introduce all SQL-files used in this project in the order they are runned when the program is started.

### **Dropping all tables**

This script has the function of dropping every table before creating so there are no errors.

### **Creating the tables**

This file is used to create all the tables we are using. It produces every table with its associated columns. This file is also used to give all the tables its associated primary keys and foreign keys.

### **Insert of data**

Last there is the "insertSQL" script, which is used to insert all the data we want into the belonging table. The order of the commands is very important because, if you insert "antwortmoeglichkeiten" at first there would be no matching "FNR" or with "frage " no fitting "QID".