Facharbeit zum Thema:

Entwicklung und Implementation einer Datenbank anhand einer Schulkursbelegung

Schule:	Gymnasium Verl
Schuljahr:	2021/2022
Kurs:	Grundkurs Informatik
Betreuender Lehrer:	Herr Jansen

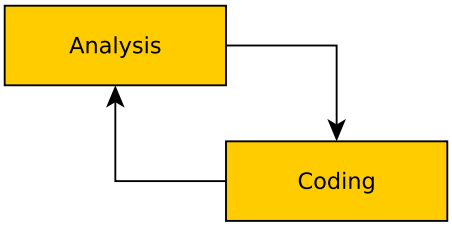
Vorgelegt von:
Frederik Folkers

Inhaltsverzeichnis

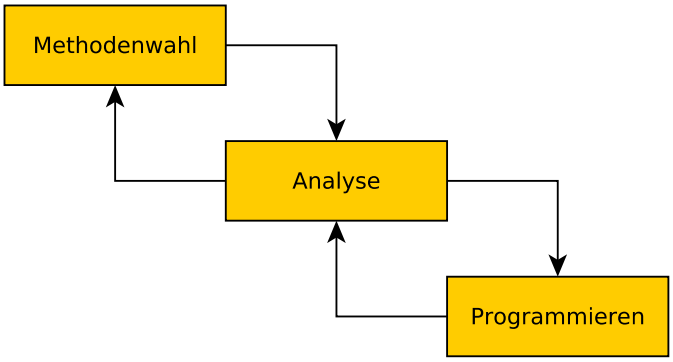
Anhang liegt im USB-Stick bei (UI-Script, Datenbank, Vorbereitungsdateien)

1 Einleitung

Im heutigen Informationszeitalter ist es wichtig, große und komplexe Datenmengen effizient abzuspeichern und die gewünschten Informationen strukturiert abrufen zu können. Daher habe ich für diese Facharbeit das Thema Datenbanken gewählt.
Zunächst werde ich verschiedene Möglichkeiten zur Sicherung von Daten vorstellen, dann dieses Wissen exemplarisch am Beispiel einer Schulkursbelegung für die Oberstufe darstellen. Bei diesem Projekt folge ich dem einfachen Wasserfallmodell[1]. Dieses sieht folgendermaßen aus:



Es ist sehr einfach, da es nur aus zwei Schritten besteht: Der Analyse und dem Programmieren. Diese habe ich gewählt, da es ein relativ kleines Projekt ist und die Anzahl der Nutzer höchstwahrscheinlich nicht in den zweistelligen Bereich gehen wird. Allerdings muss noch einen weiterer Schritt hinzugefügt werden:



Die Methodenwahl ist in diesem Fall wichtig, da ich mir am Anfang der Projektes noch nicht sicher war, welche Methoden ich verwenden werde. Dieser Schritt wird die Wahl der Datenbankstruktur, die Wahl des Datenbankmanagementsystems und die Wahl der Programmiersprache beschreiben.

2 Problem

Für diese Facharbeit habe ich mir überlegt, dass ich eine Stundenplanverwaltungssoftware erstellen will. An diesem Beispiel werde ich den Entwicklungsprozess der Datenbankstruk-

[1] [Roy70]

tur erklären. Das Problem, welches diese Software lösen könnte, ist die Verwaltung der Individualpläne der Oberstufe. Bei diesen ist, im Gegensatz zu der Unterstufe, das Problem, dass sie, wie der Name schon sagt, individuell sind. Das bedeutet, dass man für jeden einzelnen Schüler der Oberstufe einen eigenen Stundenplan abspeichern muss. Um dies möglichst effizient zu tun, werde ich die Datenbankstruktur auf dieses Problem optimieren. Weitere Funktionen sollten sein, dass man die Raumbelegung, die Schienenzeiten sowie die Namen der Schüler und Lehrer allgemein ändern kann.

3 Methodenwahl

3.1 Datenbankmodelle

Es gibt verschiedene Datenbankmodelle. Diese unterscheiden sich in ihrer Struktur und Funktionsweise. Die drei Wichtigsten werden in den nächten Abschnitten vorgestellt.

3.1.1 Hierarchisch

Ein normales Ordnersystem kann als Datenbankstruktur genutzt werden. Dabei handelt es sich um eine näherungsweise hierarchische Datenbank. Die Informationen werden sehr leicht in Gruppen (durch Unterordner) eingeteilt. Beispiel für die Anwendung einer solchen Struktur ist mein Vertretungsplan. Diesen habe ich als Alternative zu dem normalen Vertretungsplan programmiert. Ich konnte so einige stilistische Verbesserungen einbauen. In der neuen Struktur hat jede Klasse bzw. Stufe ein eigenes Verzeichnis. Dadurch müssen sich die Schüler nicht mehr alle anderen Vertretungen angucken. Abbildung 1 zeigt die Datenbankstruktur, welche in Form eines Dateisystems angelegt wurde. In jedem Ordner ist eine csv-Datei, in welcher die Daten von der entsprechenden Klasse oder Stufe gespeichert sind. Beiliegend zu jeder csv-Datei ist auch noch eine HTML-Seite. Die beiden Dateien enthalten also grundsätzlich die gleichen Informationen, aber die HTML-Seite ist für Menschen und das csv-Dokument für Maschinen. Ein Vorteil in diesem Fall ist, dass dadurch Daten und Anzeige getrennt sind. So eine Datenbank entspricht aber, wie im ersten Satz angedeutet, nicht in allen Punkten einer hierarchischen Datenbankstruktur. Es gibt zwei große Unterschiede. Wenn in sich in einem Ordnersystem ein leerer Ordner befindet, wird dieser nicht zu einem Blatt (einer Datei). Dies wäre in einer hierarchischen Datenbankstruktur der Fall. Ein weiterer Unterschied ist, dass es keine Beziehungen zwischen den Dateien geben kann.[2]

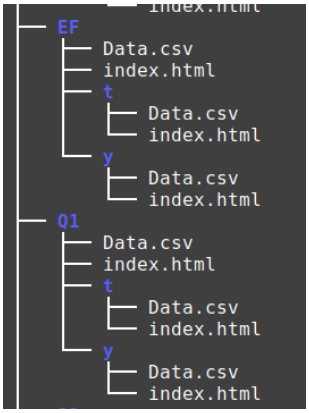


Abbildung 1: Vertretungsplan Struktur

Dieses Modell hat allerdings auch einige Nachteile, weswegen die Nutzung dieser Struktur

[2] vgl. [Mop07]

bei großen Datenmengen nur eingeschränkt möglich ist. Ein Nachteil ist, dass die Datenbank nur auf einen Anwendung optimiert werden kann[3]. Wenn z.B. mein Vertretungsplan nach Tag und nicht nach Klasse sortiert werden soll, müsste die Datenverwaltungssoftware in jeden Ordner reingehen und diesen die entsprechenden Informationen auslesen und abspeichern. Dies ist langsam und leistungsintensiv. Ich habe dieses Problem gelöst, indem ich einen zweiten Ordner für alle eingebaut habe. Das ist allerdings durch die Dopplung nicht sehr effizient.
Ein weiteres Problem ist, dass nach einer kleinen Änderung in der Struktur das Programm, ohne Prüfung der Struktur, diese nicht mehr lesen kann[4]. Aktuell ist mein Vertretungsplan nach Klassen sortiert. Wenn ich nun eine Sortierung nach Tagen wollte, müsste das gesamte Verwaltungsprogramm neu geschrieben werden.

3.1.2 Netzwerk

Die Netzwerkstruktur löst einige Probleme des hierarchischem Datenbankmodells. Bei einer Netzwerkstruktur kann prinzipiell jeder Punkt als Einstiegspunkt verwendet werden. Diese müssen allerdings bei dem Erstellen des Entwurfs festgelegt werden. Das bedeutet, dass man es nicht auf eine bestimmte Sortierung optimieren muss. Beim Erstellen dieses Modells werden die Brücken bzw. Beziehungen schon vordefiniert. Dabei ist es wichtig, dass die Beziehungen in beide Richtungen verwendbar sind. Ein Nachteil dieses Systems ist, dass nur 1:n bzw. n:1 Beziehungen dargestellt werden können[5].

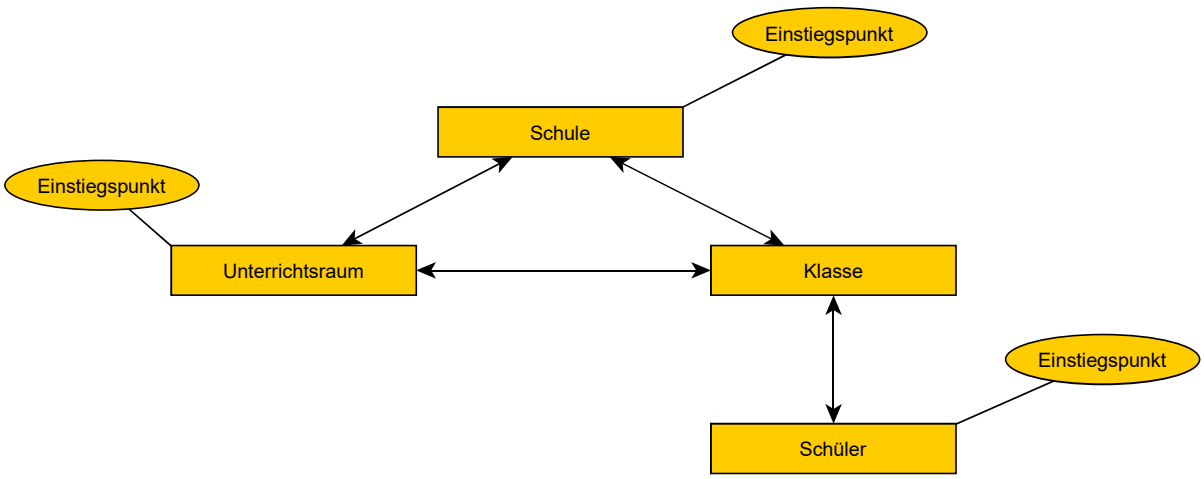


Abbildung 2: Netzwerkmodell

In Abbildung 2 sind vier Quadrate zu erkennen. Diese stehen für Entitätsmengen. An drei der vier Entitätsmengen sind Ellipsen in denen Einstiegspunkt steht. Ein Einstiegspunkt kennzeichnet im Grunde die Information, die als Ausgangsinformation verwendet

[3] vgl. [Jar10] 3.2 Das hierarchische Datenbank-Modell
[4] vgl. [Cod70] S.378
[5] vgl. [Jar10] 3.3 Das Netzwerk-Datenbank-Modell

werden kann.
Wenn nun ermittelt werden soll, in welcher Klasse ein bestimmter Schüler ist, kann der Einstiegspunkt bei Schüler benutzt und von dort aus auf Klasse zugegriffen werden. Dies gilt genauso für Unterrichtsraum und Schule. Wenn erfragt werden soll, welche Klasse einen bestimmten Raum benutzt, oder welche Klassen eine bestimmte Schule hat, so ist das möglich. Wenn aber erfragt werden soll, welcher Klassenraum von einer bestimmten Klasse benutzt wird, ist das genauso wenig möglich, wie herauszufinden welche Schüler in dieser Klasse sind. Der Grund dafür ist, dass Klasse nicht als Einstiegspunkt markiert wurde und somit auch nicht als ein solcher benutzbar ist. Mit anderen Worten: In diesem Beispiel ist es nicht möglich Informationen über Schüler, Unterrichtsräume oder Schulen abzurufen, wenn die einzige gegebene Information die Klasse ist.

3.1.3 Relational

Dieses Datenbankmodell ist das populärste. Die Grundlage dafür bildet das mir vorliegende Artikel „A Relational Model of Data for Large Shared Data Banks" [Cod70]. Dieses wurde von Edgar F. Codd verfasst und im Juni 1970 veröffentlicht. In diesem Artikel wird eine neue Art der Datenspeicherung vorgestellt. Durch dieses Modell werden einige der Probleme gelöst, welche bei primitiveren Modellen vorhanden waren. Dieser Artikel bildet nebenbei auch noch die Grundlage für die moderne und meistgenutzte Datenbanksprache SQL.
Der Artikel beginnt damit, dass der Autor die damaligen Probleme an Datenstrukturen erklärt. Dabei geht er auf das Hierarchische (3.1.1) und das Netzwerk-Modell (3.1.2) ein. Daraufhin bringt er ein neues verbessertes Datenbankmodell ins Spiel: Das relationale Datenbankmodell. Dieses Modell basiert darauf, dass die Informationen in zweidimensionalen Tabellen, also Tabellen mit nur einer Spalten und Zeilen Ebene, gespeichert werden. Das hat den Vorteil, dass alle Informationen in einem 2D Array gespeichert werden können. Bei einer dreidimensionalen Tabelle, also wenn mehrere Ebenen existieren, müsste dies durch ein 3D Array dargestellt werden. Das macht die Datenspeicherung unübersichtlich und unnötig kompliziert.
Diese Tabellen fassen verschiedenen Entitäts-Typen zusammen wie z.B. „User", „Bücher" oder „Verkäufer". Für jede Eigenschaft der Entitäten wird eine Spalte angelegt. Jetzt kann man immer, wenn man einen neuen Datensatz bzw. Entität in die Tabelle einspeichern will, eine weitere Zeile zu der Tabelle hinzufügen. Jede dieser Tabellen hat einen sogenannten Primary Key. Dieser kann jede Zeile individuell identifizieren. Er kann auch aus mehreren Spalten bestehen. Damit man diese Spalte erkennen kann, ist sie meistens gekennzeichnet. Somit muss Primary Key bei jeder Entität anders sein. Wenn es jetzt Sinn macht einige Informationen in einer anderen Tabelle zu speichern z.B. wenn man ein Bibliotheksverwaltungsprogramm hat, kann man den Inhalt getrennt von dem Regal abspeichern. Für diesen Fall hatte E. F. Codd eine Idee. Man kann eine zweite Tabelle

erstellen, in der man alle Informationen zu seinem zweiten Datensatz abspeichert. Um erkennen zu können, welche Zeile zu welcher Zeile in der zweiten Tabelle gehört, fügt man der zweiten Tabelle den Primary Key des ersten hinzu. In der zweiten Tabelle wird dieser dann Foreign Key genannt. Wenn man z.B. nach einem Buch mit einem bestimmten Inhalt sucht, kann man diesen in der Inhaltstabelle suchen und dann in der Regalpositionsliste schauen, wo es steht. Andersherum kann man ein bestimmtes Buch aus der Regalpositionsliste heraussuchen und dann nachschauen, wie der Inhalt ist. Man kann hier einen großen Unterschied zur Netzwerkstruktur erkennen, da man jede Tabelle als Eintrittspunkt nutzen kann. Diese Beziehung wird 1:1 Beziehung genannt, da jede Zeile nur auf eine Zeile verweist. Im Beispiel wird dadurch beschrieben, dass jedes Buch nur einen Inhalt haben kann. Bei dieser Beziehung ist es egal, in welcher Tabelle den Foreign Key steht.
Eine 1:n Beziehung ist sehr ähnlich aufgebaut. Bei dieser wird der Primary Key genauso in die andere Tabelle geschrieben. Es ist aber möglich, dass der Primary Key auf mehrere Foreign Keys verweist. In unserem Beispiel würde diese Beziehung genutzt werden, wenn man eine statt einer Regalpositionsliste nur eine Regalliste hätte. In dieser würde dann nur das Regal abgespeichert werden. Damit kann jedes Regal mehrere Bücher haben, jedes Buch aber nur ein Regal. In diesem Fall würde man den Foreign Key in die Bücherliste schreiben.
Die letzte Beziehung, mit der ich mich beschäftigen werde, ist eine n:m Beziehung. Diese ist etwas komplizierter, da es nötig wird eine neue Tabelle zu erstellen. Diese Tabelle setzt sich aus den beiden Foreign Keys der Tabelle zusammen, welche beide als Primary Key der neuen Tabelle dienen. Wenn man nun herausfinden will, welche Zeilen der einen Tabelle zur anderen gehören, guckt man einfach den entsprechenden Primary Key in der Beziehungstabelle nach. Dadurch findet man die Primary Keys der entsprechenden Zeilen der zweiten Tabelle und kann diese auslesen. Dieser Vorgang kann auch andersherum benutzt werden. Mit dieser Struktur ist es möglich, große Datenmengen relativ einfach abzuspeichern und diese in Verbindung zu setzten.

3.1.4 Entscheidung

Ich habe mich für das relationale Datenbankmodell entschieden, da es das modernste und flexibelste ist. Für die Implementierung der Schulkursbelegung ist diese Flexibilität nötig, da ich keine Begrenzung in den Abfragemöglichkeiten haben will. Zudem möchte ich die Programmierung in SQL nutzen.

3.2 Datenbankmanagementsysteme

In den 70er Jahren wurde von IBM ein neues Datenbankmanagementsysteme entwickelt, das System R. Um die Daten zu verwalten wurde SQL auf Basis der Erkenntnisse von E. F. Codd entwickelt. SQL steht für „Structured Query Language“ und wird dazu genutzt,

Datenbankstrukturen in relationalen Datenbanken zu erstellen[6]. Zudem können die Daten mit SQL bearbeitet und abgefragt werden[7].

3.2.1 MySQL

MySQL ist ein Server basiertes Datenbankmanagementsystem. Das bedeutet, es ist möglich mit verschiedenen Benutzern auf dieses System zuzugreifen. Das hat den Vorteil, dass es von mehreren Programmen gleichzeitig genutzt werden kann, diese müssen sich nicht einmal auf dem gleichen Computer befinden. MySQL kann sehr große Mengen an Daten abspeichern, ohne sich dabei zu verlangsamen.

3.2.2 SQLite

SQLite ist ein einfaches und kleineres Datenbanksystem. „Lite“ bezieht sich auf die Menge des Speicherplatzes, es wird deutlich weniger benötigt. Um mit der Datenbank zu kommunizieren, wird SQL genutzt. Pro Zeile kann 1 Gigabyte an Daten gespeichert werden und es kann maximal 281 Terabyte an Daten speichern. Ein großer Vorteil dieses Datenbanksystems ist, dass es keinen Server benötigt. Die gesamte Datenbank kann einfach als Datei gespeichert werden[8]. Aus diesem Grund wird es oft in kleinen Anwendungen benutzt. Am weitesten ist diese Datenbank auf dem Smartphone Markt verbreitet. SQLite schätzt die Anzahl der aktiv benutzten Datenbanken auf über 1 Billionen[9]. Des Weiteren ist SQLite Open Source.

3.2.3 Entscheidung

Für dieses Projekt ist meiner Meinung nach SQLite klar die bessere Wahl. Ein Aspekt ist, dass ich dass ich die Datenbank abgeben muss, deswegen sollte diese auch unabhängig von dem Status irgendeines Servers sein. Allerdings ist es sehr einfach zwischen den Datenbanken zu wechseln, da sie beide SQL benutzen. Dadurch müssen die Befehle nur einmal konstruiert werden und funktionieren dann für beide Datenbanken. Deswegen habe ich beide implementiert.

3.3 Programmiersprache

Bei der Programmiersprache habe ich mir folgende mögliche Kanidaten angesehen.

[6] vgl. [Lau19] S.19
[7] vgl. [ntc22]
[8] vgl. [SQL22b]
[9] [SQL22a]

3.3.1 Python

Pyhton ist eine einfache Scriptingsprache, mit der ich schon Erfahrung sammeln konnte[10]. ist sie gut strukturiert und perfekt für schnelle Programme[11].

3.3.2 PHP

PHP ist eine relativ einfache Programmiersprache zum Programmieren des Backends von Websites. Schon im ersten Jahr wurde eine SQL Funktion eingebaut[12]. Dazu hat es mehrere nützliche Funktionen wie 'associative array' bei denen den verschiedenen Objekten statt Nummern Namen zugewiesen werden. Dies ist insbesondere beim Auslesen von Tabellenzeilen nützlich.

3.3.3 Entscheidung

Ich habe mich in diesem Fall für Python entschieden, da es im Gegensatz zu PHP keinen Webserver benötigt, um zu funktionieren. Außerdem wäre es in PHP nötig mehrere html-Seiten zu designen.

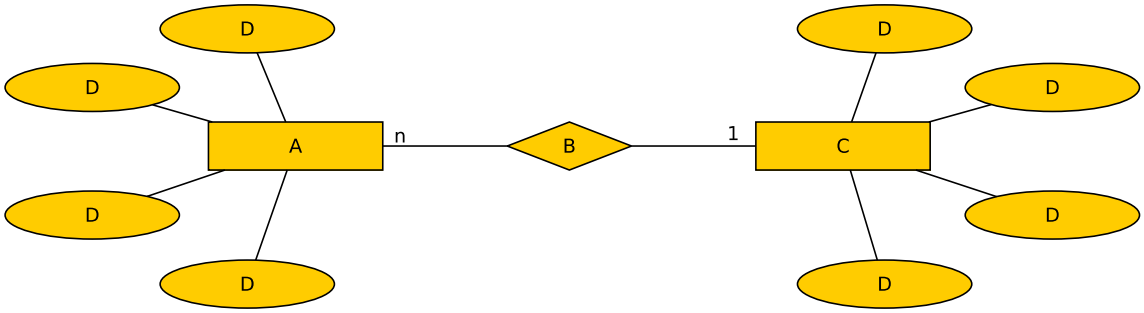
4 Analyse

Zu der Analyse gehört das Beschreiben eines Programmes. Dies geschieht in diesem Fall durch ein Entity-Relationship-Diagram. Die Syntax eines solchen werde ich im folgendem Kapitel erklären. Danach werde ich mein eigenes Diagramm erklären.

4.1 Entity-Relationship-Diagramm erstellen

4.1.1 Syntax

Nach der Idee für die Datenbank wird zuerst eine Entity-Relationship-Diagramm erstellt. Dieses wird genutzt um die Datenbank genau zu planen. Bevor wir uns mein Diagramm ansehen könnnen, werde ich zuerst an Abbildung 4.1.1 die Syntax erklären.



[10] siehe Vertretungsplan: 3.1.1
[11] vgl. [Kuh13] S.13
[12] siehe: [Gro22]

Attribut (D) In dieser Grafik werden die Attribute bzw. Spalten durch Ellipsen dargestellt. In einer bildlichen Darstellung durch Tabellen würden Domänen die Spaltenüberschriften repräsentieren. Eine Domäne kann als Primary Key für eine Entitätsmenge gekennzeichnet werden. Dies geschieht, indem man den Spaltennamen unterstreicht[13].

Entitätsmenge (A, B) Die Entitätsmengen werden durch Vierecke dargestellt. Im übertragenen Sinne sind dies Tabellen. Entitätsmengen sind von Attributen umringt. Davon ist normaler Weise mindestens eines ein Primary Key[14].

Beziehungsmenge (B) Eine Raute beschreibt eine Beziehungsmenge1. In der Mitte steht die reale Beziehung zwischen den Entitätsmengen z.B. „besitzt“ oder „ist in“. Die Raute ist mit zwei Strichen mit den zugehörigen Entitätsmengen verbunden. Auf diesen Pfeilen werden die Kardinalitäten notiert. Dabei steht die Menge der Beziehungen einer einzelnen Entität auf der anderen Seite der Raute. In dem abgebildeten Beispiel kann eine Zeile in A nur auf einen Zeile in C verweisen, aber eine Zeile in C kann auf beliebig viele Zeilen in A verweisen[15].

4.1.2 ER-Diagramm

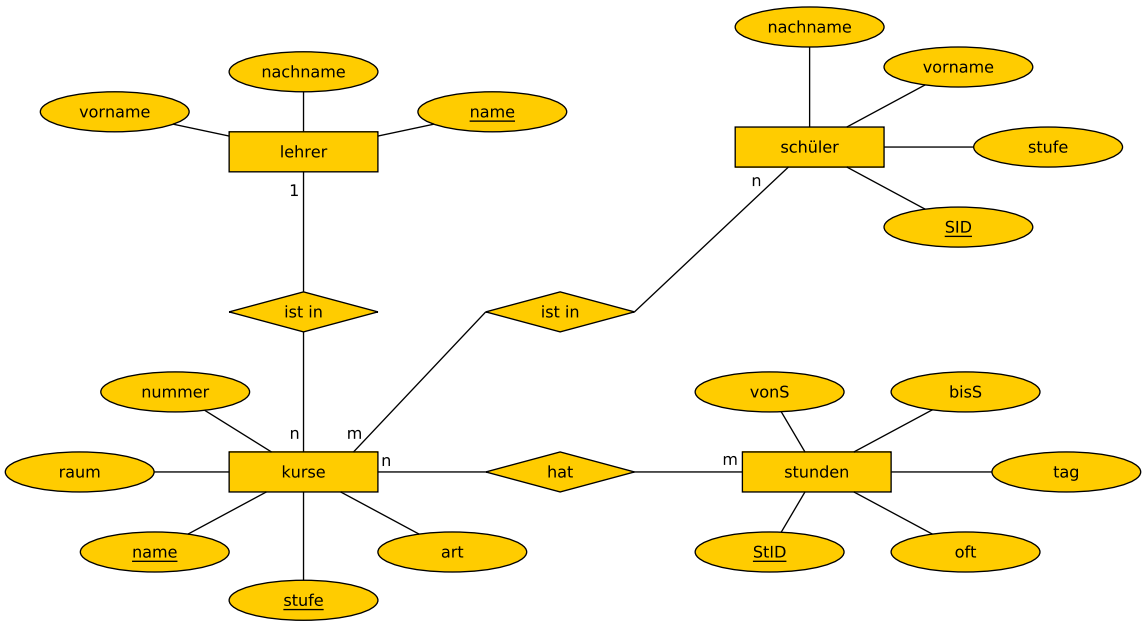


Abbildung 3: Stundenplan ER-Diagramm

In Abbildung 3 wird das Stundenplan ER-Diagramm dargestellt. Die folgenden Abschnitte erklären die Einzelheiten.

[13] vgl. [Jar10] 7.5 Datenbankentwurf: Attribut
[14] vgl. [Jar10] 7.5 Datenbankentwurf: Entität
[15] vgl. [Jar10] 7.5 Datenbankentwurf: Beziehung

Schüler Hier wird für jeden Schüler eine Entität hinzugefügt. Von jedem Schüler wird der Vorname, der Nachname und die Stufe abgespeichert. Des Weiteren wird jedem Schüler eine „SID“(Schüler Identitätsnummer) zugewiesen. Diese dient als Primary Key. Dieser Schritt ermöglicht, dass es mehrere Schüler mit dem gleichen Namen und Nachnamen, welche in derselben Stufe sind, unabhängig voneinander zu verwalten. Der Name stufe ist etwas irreführend, da dieses Attribute die Klasse speichert dabei sind Oberstufe und Unterstufe gleichberechtigt. Also kann hier ”8bünd ËFßtehen.

Kurse In der Entitätsmenge Kurse werden alle besetzbaren Kurse zusammengefasst. Dabei werden die Werte „stufe“, „name“, „fach“ und „art“ gespeichert. „stufe“ und „fach“ sind dabei selbsterklärend. In „name“ wird die Abkürzung des Kurses abgespeichert. Dies geschieht in dem oft von der Schule benutzten Format[16]. In „art“ wird die Art des Kurses abgespeichert. Damit ist gemeint, ob es ein Grundkurs, ein Leistungskurs oder ein Zusatzkurs ist. In der letzten ungeklärten Domäne „nummer“ wird abgespeichert der wievielte Kurs es ist. Wenn ein Kurs nicht ausreicht, um alle Schüler zu unterrichten werden diese aufgeteilt. Die Kurse werden dann durchnummeriert. Damit man diese unterscheiden kann muss ich diese Information auch abspeichern. Jetzt könnte man anmerken, dass durch diese Aufteilung eine Art Informationsdopplung entsteht. Ich habe mich allerdings für diese Aufteilung entschieden, da so die Informationen individuell abrufbar sind. Ein weiterer Vorteil ist, dass so die Informationen in einer erweiterten Form abgespeichert werden können. Man kann zum Beispiel statt nur der Fach-Abkürzung den gesamten Fachnamen abspeichern. Dies erleichtert im Nachhinein das Ausgeben der Informationen, da man die Abkürzung nicht mehr im Programm in den ausgeschriebenen Namen umwandeln muss. Diese Tabelle besitzt einen Foreign Key in welchem das Kürzel des Kurslehrers gespeichert ist. Dieses kann mit der Tabelle Lehrer abgeglichen werden.

Lehrer Diese Entitätsmenge enthält eine Liste aller Lehrer der Schule inklusive ihren Vornamen, Nachname und Kürzel.

Stunden In der Tabelle 'stunden' werden alle möglichen Kurszeiten abgespeichert. Alle Attribute dieser Tabelle sind Zahlen. In 'tag' wird der Wochentag abgespeichert. In 'vonS' und 'bisS' wird die Startstunden und die Endstunde abgespeichert. In 'oft' steht in welchem Rhythmus die entsprechende Stunde stattfindet, bei den meisten wäre das einmal pro Woche, also steht dort eine 1. Wenn eine Stunde nur jede zweite Woche stattfindet steht in 'oft' eine 2. 'StId' ist eine automatisch zugewiesene Identitätsnummer.
Nun zu den Beziehungen, die eine eigene Tabelle benötigen:

Schüler-Kurse(Oberstufe) Diese Verbindung beschreibt eine Tabelle in der die Kurs belegung der verschiedenen Schüler abgespeichert ist. Eine Zeile besteht aus der ID des

[16]Fach KursartNummer (z.B. CH G1)

Schülers, der Stufe, welche zum Primary Key von Kurse gehört, und der Name des zubelegenden Kurses.

Kurse-Stunden Diese Beziehung beschreibt eine Tabelle, in der die Stunden ihren Zeiten zugeordnet werden. Dies geschieht, indem der Kursname und die Stufe zusammen mit der entsprechenden Stunden ID in eine Zeile geschrieben werden.

4.2 Datenbank erstellen

Die in 3.2 besprochenen Unterschiede sprechen klar dafür, dass ich eine SQLite Datenbank verwenden sollte. Aber, da ich mehr Erfahrung in MySQL habe, habe ich beides gemacht. Die SQL-Befehle habe ich in der offiziellen MySQL-Dokumentation nachgeschaut[17]. Da SQLite auch mit SQL funktioniert konnte ich die Befehle im Grunde übernehmen.

Zunächst habe ist eine neue Datenbank erstellt. In SQLite ist dies nicht nötig, da es nur einen Datenbank gibt. Damit es keine Zeichenfehler gibt habe ich ü gegen ue ersetzt.

```
01 | CREATE DATABASE schueler;
```

Danach konnte ich mit der ersten Tabelle beginnen. Für die Länge des Vornamens und des Nachnamens habe ich 100 Zeichen gewählt. Der Stufenname kann nur aus 2 Buchstaben bestehen, also habe ich ihn darauf auch begrenzt. Die SID wird in MySQL automatisch definiert. In SQLite funktioniert das leider nicht. Also habe ich das Problem gelöst indem ich die SID in SQLite manuel definiere. Somit konnte ich das 'AUTO INCREMENT' Attribute weglassen.

```
01 | CREATE TABLE schueler(
02 |         SID INT PRIMARY KEY AUTO_INCREMENT,
03 |         vorname VARCHAR(100),
04 |         nachname VARCHAR(100),
05 |         stufe VARCHAR(2)
06 | );
```

Das Erstellen der anderen Tabellen erfolgte mit dem gleichem Befehl. Nur die Arttribute und der Name mussten verändert werden.

```
01 | CREATE TABLE stunden(
02 |         StId INT AUTO_INCREMENT PRIMARY KEY,
03 |         vonS INT,
04 |         bisS INT,
05 |         tag INT,
06 |         oft INT
07 | );
08 |
```

[17] vgl. [Tea22]

```
09 | CREATE TABLE lehrer(
10 |         short VARCHAR(3) PRIMARY KEY,
11 |         vorname VARCHAR(100),
12 |         nachname VARCHAR(100),
13 | );
14 |
15 | CREATE TABLE kurse(
16 |         name VARCHAR(10),
17 |         stufe VARCHAR(2),
18 |         fach VARCHAR(50),
19 |         art VARCHAR(30),
20 |         nummer INT,
21 |         raum VARCHAR(10),
22 |         lShort VARCHAR(3),
23 |         PRIMARY KEY(name, stufe),
24 |         FOREIGN KEY (lShort) REFERENCES lehrer(short)
25 | );
```

Die Tabellen, die jetzt noch fehlen sind die Beziehungstabellen. In diesen Tabellen sind alle Foreign Keys und gleichzeitig Primary Keys. Da es mehrere Primary Keys gibt, müssen diese, genau wie die Foreign Keys, am Ende definiert werden.

```
01 | CREATE TABLE stundenKurs(
02 |         name VARCHAR(10),
03 |         stufe VARCHAR(2),
04 |         StId INT,
05 |         PRIMARY KEY (name, stufe, StId),
06 |         FOREIGN KEY (name, stufe) REFERENCES kurse(name, stufe),
07 |         FOREIGN KEY (StId) REFERENCES stunden(StId)
08 | );
09 |
10 | CREATE TABLE schuelerKurs(
11 |         SID INT,
12 |         name VARCHAR(10),
13 |         stufe VARCHAR(2),
14 |         PRIMARY KEY (SID, name, stufe),
15 |         FOREIGN KEY (name, stufe) REFERENCES kurse(name, stufe),
16 |         FOREIGN KEY (SID) REFERENCES schueler(SID)
17 | );
```

Damit ein Programm von meinem Computer auf den SQL-Datenbankserver zugreifen kann, habe ich einen neuen Beutzer erstellt. Diesem musste ich dann die Benutzungsrechte für die meine Datenbank geben. Das ging mit dem folgendem Befehl:

```
01 | CREATE USER "schueler"@"%" IDENTIFIED BY "stundenplan#1Pas";
02 |
03 | GRANT ALL PRIVILEGES ON schueler.* TO schueler;
```

In SQLite gibt es nur einen User, deswegen war dies bei der SQLite-Datenbank nicht nötig.

4.3 Datenbank füllen

Um die Datenbank zu füllen habe ich für jede Tabelle ein Script programmiert. Diese liest eine csv-Datei aus und überträgt jede Zeile in die entsprechende Datenbank. Die csv-Dateien habe ich folgendermassen erstellen:

Kurse Ich habe mir aus unserem Kursplan ein paar Kurse ausgesucht und diese per Hand in das Dokument geschrieben.

Lehrer Hier konnte ich die Kürzelliste von der Schulwebsite kopieren und dann mit einem Script in die gewünschte Form bringen.

Schüler Die Namen in dieser Tabelle sind frei erfunden.

SchülerKurs Da die Namen schon frei erfunden waren war es schwer für diese eine Kursbelegung zu finden. Also habe ich mir auch diese ausgedacht und versucht keine Zeiten doppelt zu belegen.

Stunden Ich habe einfach alle möglichen Kurszeiten per Hand abgeschrieben.

StundenKurs Hier habe ich per Hand die entsprechden Zeiten aus dem Kursplan abgeschrieben.

4.4 Datenbank auslesen

Um die Tabelle auszulesen, habe ich für die verschiedenen Anwendungen folgende Befehle konstruiert:

Kurslisten Damit man alle Teilnehmer eines Kurses anzeigen kann, muss man zuerst die Schülertabelle mit der Schüler-Kurstabelle verbinden. Dies geschieht durch den Befehl 'JOIN'. Der Parameter 'ON' beschreibt, wann die Tabellen verbunden werden sollen, so wie in diesem Fall, wenn die SID in der Schülertabelle gleich mit der SID in der Schüler-Kurstabelle ist. Dieser Befehl würde eine Tabelle mit allen Kursnamen, ihren Stufen und allen Schülern ausgeben. Da wir aber genauere bzw. ausgeschriebene Angaben haben wollen, habe ich diese Tabelle mit der Kurstabelle verbunden. Dies geschieht, wenn der Name und die Stufe gleich sind. In der letzten Zeile wird bestimmt, welcher Kurs ausgegeben werden soll. Dies geschieht, indem eine Ausgabebedingung hinzugefügt wird.

Somit werden nur die Zeilen ausgegeben, in denen der Name und die Stufe mit den gegebenen Werten übereinstimmt.

```
01 | SELECT schueler.vorname, schueler.nachname
02 | FROM schueler
03 |         JOIN schuelerKurs
04 |         ON schueler.SID = schuelerKurs.SID
05 |         JOIN kurse
06 |         ON schuelerKurs.name = kurse.name AND schuelerKurs.stufe
       = kurse.stufe
07 | WHERE kurse.name = <Kursname> AND kurse.stufe = <stufe>;
```

Zeiten Um die Zeiten eines Kurses ausgeben zu können, wird die Kurstabelle mit der Stunden-Kurstabelle verglichen und die Zeilen zusammengefügt, bei denen der Name des Kurses übereinstimmt. Diese Tabelle wird dann mit der Stundentabelle fusioniert, bei denen die StId's übereinstimmen. Daraufhin werden nur die Zeilen ausgegeben, bei denen der Name und die Stufe mit der gegebenen Stufe und dem gegebenen Namen übereinstimmt.

```
01 | SELECT stunden.tag, stunden.vonS, stunden.bisS
02 |     FROM kurse
03 |         JOIN stundenKurs
04 |         ON kurse.name = stundenKurs.name
05 |         JOIN stunden
06 |         ON stundenKurs.StId = stunden.StId
07 |     WHERE kurse.name = <name> AND kurse.stufe = <stufe>;
```

Stundenpläne Die Stundenplanabfrage war mit Abstand die komplizierteste, da fünf Tabellen miteinander verbunden werden müssen. Ich habe mit der Schülertabelle begonnen. Diese habe ich wie in Kurslisten mit der Schüler-Kurstabelle verbunden. Die daraus hervorgehende Tabelle konnte ich dann wie in Zeiten mit der Stunden-Kurstabelle und der Stundentabelle zusammenführen. Aus dieser Tabelle musste ich dann nur noch die Zeilen, in denen der Vorname und Nachname mit den gegebenen übereinstimmt, ausgegeben werden.

```
01 | SELECT kurse.name, stunden.tag, stunden.vonS, stunden.bisS
02 | FROM schueler
03 |     JOIN schuelerKurs
04 |     ON schuelerKurs.SID = schueler.SID
05 |     JOIN kurse
06 |     ON schuelerKurs.name = kurse.name AND schuelerKurs.stufe =
       kurse.stufe
07 |     JOIN stundenKurs
08 |     ON kurse.name = stundenKurs.name
09 |     JOIN stunden
10 |     ON stundenKurs.StId = stunden.StId
```

```
11 |  WHERE schueler.vorname = <Vorname> AND schueler.nachname = <
       Nachname> ORDER BY  stunden.tag;
```

4.5 Mein Script

Zur einfachen Anwendung dieser Befehle habe ich ein Script programmiert welches eine einfache UI[18] bietet. Es heißt select.py und ist auf dem USB-Stick[19].

5 Zusammenfassung und Ausblick

Ich habe es geschafft, dieses Projekt mit einem funktionierenden Prototypen zu beenden. Allerdings bin ich noch nicht ganz zufrieden mit der jetzigen Datenbank. Diese hat das Problem, dass sie nicht alle möglichen Stundenpläne abspeichern kann. Wenn sich z.B. ein Kurs zu verschiedenen Zeiten in verschiedenen Räumen unterrichtet wird, ist es nicht möglich, dies abzubilden. Eine Lösung wäre in 'stunden' den Raum mit abzuspeichern. Ich hatte allerdings keine Zeit mehr, diese Änderung zu implementieren. Eine weitere mögliche Erweiterung wäre eine grafische UI und eine verbesserte Stundenplananzeige. Letzteres lässt in ihrer jetzigen Form wirklich zu wünschen übrig. Aber mit der aktuellen Grundlage bin ich zufrieden.
Bei diesem Projekt habe ich viel über Datenbanken und ihre Struktur, sowie SQL lernen können. Ich habe mein Wissen auch in der Recherche nach guten Quellen erweitern können. Besonders beeindruckt hat mich der Artikel von E. F. Codd, in dem er bereits vor 50 Jahren exakt beschreibt, was ich heute programmiert habe.

[18] UI - User Interface
[19] Denken Sie daran README.txt zu lesen

Literaturverzeichnis

[Cod70] Edgar F. Codd. „A Relational Model of Data for Large Shared Data Banks“. In: *Communications of the ACM* (1970).

[Roy70] Dr. Winston W. Royce. „Managing the development of large software systems“. In: 1970.

[Mop07] Mopskatze. *Hierarchisches Datenbankmodell. Dateisysteme.* Der Wikipedia-Artikel ist die älteste Referenz, die ich dazu finden kann. Es scheint als ob alle anderen Autoren, die so etwas geschrieben haben, Wikipedia kopiert haben. 2007. URL: `https://de.wikipedia.org/wiki/Hierarchisches_Datenbankmodell` (besucht am 27. 02. 2022).

[Jar10] Helmut Jarosch. *Grundkurs Datanbankentwurf.* 2010.

[Kuh13] Dave Kuhlman. *A Python Book: Beginning Python, Advanced Python, and Python Exercises.* 2013. URL: `https://www.davekuhlman.org/python_book_01.pdf`.

[Lau19] Michael Laube. *Einstieg in SQL.* 2019.

[Gro22] The PHP Group. *Die Geschichte von PHP.* 2022. URL: `https://www.php.net/manual/de/history.php.php#history.php`.

[ntc22] ntcHosting. *SQL (Structured Query Language).* 2022. URL: `https://www.ntchosting.com/encyclopedia/databases/structured-query-language/`.

[SQL22a] SQLite. *Most Widely Deployed and Used Database Engine.* 2022. URL: `https://sqlite.org/mostdeployed.html`.

[SQL22b] SQLite. *What Is SQLite?* 2022. URL: `https://sqlite.org/index.html`.

[Tea22] SQL Entwickler Team. *MySQL 8.0 Reference Manual.* 2022. URL: `https://dev.mysql.com/doc/refman/8.0/en/sql-statements.html`.

Schlusserklärung

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig angefertigt, keine weiteren als die angegeben Hilfsmittel benutzt und die Stellen der Facharbeit, die im Wortlaut oder im wesentlichen Inhalt aus anderen Werken entnommen sind, mit genauer Quellenangabe kenntlich gemacht habe.

Verl, den 6. März 2022