Verwaltung von Daten – Teil1

Herausforderungen bei der Verwaltung großer Datenbestände

Eine Sammlung von Daten lässt sich übersichtlich in Tabellenform darstellen. Ein Beispiel wäre eine Kursliste mit allen Schülern[1] des Kurses, ihrer Anschrift, Telefonnummer und Geburtsdatum.

Aufgabe 1: Nennen Sie weitere Beispiele für Daten, die sich übersichtlich in Tabellen darstellen lassen.

Schulen, Behörden, Vereine, aber auch Online-Dienste, wie Online-Shops oder soziale Netzwerke müssen so viele Daten verwalten, dass eine einzelne Tabelle nicht ausreicht, um die Daten übersichtlich und effizient zu speichern. Wir betrachten als Beispiel die Daten, die ein Sportverein verwaltet.

Aufgabe 2:

a) Sammeln Sie in Kleingruppen Daten, die ein Sportverein vermutlich speichern möchte. Erstellen Sie eine Mindmap.

b) Im Sportverein TSV Fitstadt wird für jeden Sportkurs eine Tabelle angelegt. Abbildung 1 zeigt das Schema, nach dem die Tabellen aufgebaut sind. Jeder Trainer erhält die Tabelle zu seinem Kurs. Diskutieren Sie Vor- und Nachteile der Tabellenstruktur. Gehen Sie dabei auch auf folgende Aspekte ein.
   - Können alle Daten, die Sie in Teil a) gesammelt haben, in dem Tabellenschema untergebracht werden?
   - Welche Probleme treten ggf. auf, wenn
     - ein Mitglied, das mehrere Kurse belegt, umzieht.
     - jemand nach einer Übersicht über alle Handballkurse fragt.
     - nachgeschaut werden soll, ob die Trainingszeit Donnerstag von 16:00 bis 17:00 Uhr in Sporthalle 2 noch frei ist.
     - ein Mitglied vorübergehend keinen Sport ausüben kann und sich deshalb von allen Kursen abmeldet, aber passives Mitglied des Vereins bleibt.

Kurs:	Handball HHE
Trainer:	Rudi Hüpfer
Ort:	Sporthalle 2
Trainingszeit:	Mittwoch, 17:30 Uhr bis 19:00 Uhr

Nachname	Vorname	Anschrift	Telefon	Geb.-Datum	Kontonr.
Ahorn	Rainer	Uferweg 3, 23413 Fitstadt	08234-123412	12.03.2000	DE12 2654 6879 5587 5316 75
Bauer	Armin	Seestr. 56, 23413 Fitstadt	0141-3265987	15.08.2001	DE34 7852 6487 3254 1279 58
Lang	Stefan	Hauptstr. 90, 23411 Neudorf	0179-6523125789	18.01.1999	DE83 7654 3495 7612 9458 87
…	…	…	…	…	…

Abbildung 1: Beispiel einer Kurstabelle des TSV Fitstadt

[1] Zur besseren Lesbarkeit und Umsetzbarkeit im Zusammenhang mit Datenbanken wird das generische Maskulinum und exemplarisch die männliche Form verwendet. Diese Formulierung umfasst hier gleichermaßen und gleichberechtigt alle Geschlechter.

Am Beispiel der Kurstabellen des Sportvereins TSV Fitstadt wird deutlich, dass es unpraktisch ist, wenn man Daten mehrfach speichert. Hier wird für ein Mitglied, das mehrere Kurse belegt, in jeder Kurstabelle die vollständige Adresse gespeichert. Wenn ein Mitglied umzieht, kann es schnell passieren, dass in einer Tabelle vergessen wird, die Adresse zu ändern. Abgesehen davon wird unnötiger Speicherplatz belegt.

Während die Tabelle die Daten zu einem Kurs für die Trainer übersichtlich darstellt, müssen die angebotenen Handballzeiten oder die Belegungen der Sporthalle erst mühsam aus mehreren Tabellen herausgesucht werden. Für Mitglieder und Trainer, die aktuell keinem Kurs zugeordnet werden können, stellt sich die Frage, wo die Daten überhaupt gespeichert werden. Außerdem benötigt der Sportverein vermutlich auch noch zusätzliche Daten, z. B. über die Trainer, für die in den Kurstabellen keine Einträge vorgesehen sind.

Der Sportverein sollte die Struktur seiner Tabellen also noch einmal überdenken.

Es kommen jedoch noch weitere Probleme hinzu. Nehmen wir einmal an, die Tabellen werden in einer Tabellenkalkulationssoftware (TKS) erstellt und alle in einer Mappe, d. h. in einer Datei abgelegt, um die Daten der einzelnen Tabellen leichter miteinander verknüpfen zu können. Wer erhält nun Zugriff auf die Datei? Es ist bei der ursprünglichen Tabelle in Abbildung 1 schon problematisch, dass die Trainer die Kontodaten sämtlicher Kursteilnehmer erhalten, eine Einsicht in die Daten sämtlicher Mitglieder wäre nicht zu rechtfertigen. Die Trainer dürften also jeweils nur einen Auszug der relevanten Daten erhalten.

Im Verein gibt es vermutlich mehrere Personen, die die Möglichkeit haben müssen, Daten zu aktualisieren, wenn sich z. B. Trainingszeiten, Kursbelegungen, Adressen usw. ändern. Wenn jeder die Datei lokal speichert, ist es sehr schwierig und aufwändig, sicherzustellen, dass alle immer die aktuelle Version der Datei verwenden. Es gibt die Möglichkeit TKS-Dateien in einem Cloud-Speicher abzulegen und kollaborativ zu bearbeiten. Das heißt mehrere Personen können die Datei an ihrem Rechner gleichzeitig öffnen und bearbeiten. Aber auch das läuft nicht immer reibungslos.

Aufgabe 3: Wenn Sie an Ihrer Schule ein System verwenden, dass das kollaborative Arbeiten ermöglicht, testen Sie einmal, was passiert, wenn alle Schüler Ihres Kurses gleichzeitig Änderungen in dem Dokument vornehmen. Dazu kann einer von Ihnen eine Tabelle nach dem Schema in Abbildung 1 anlegen. Jeder im Kurs trägt nun ein fiktives Mitglied in die Tabelle ein und versucht die Daten eines anderen Mitgliedes zu ändern.

Wenn Sie den Versuch in Aufgabe 3 durchgeführt haben, sind vermutlich technische Probleme aufgetreten. Möglicherweise wurde das System durch die vielen Eingaben überlastet oder Sie haben Ihre Änderungen gegenseitig überschrieben, so dass einige Daten verloren gegangen sind. Die Tools zum kollaborativen Arbeiten an einem Text- oder TKS-Dokument sind also häufig eher darauf ausgelegt, dass einer schreibt und die anderen nur lesen oder mehrere Personen das Dokument zu unterschiedlichen Zeiten bearbeiten.

Aufgabe 4: Erstellen Sie eine Liste von Anforderungen, die eine Software zur Verwaltung großer Datenbestände in Unternehmen, Schulen, Behörden, Online-Diensten usw. erfüllen sollte.

Datenbanksysteme

Für die Verwaltung größerer Datenmengen, verwendet man statt einer TKS meist leistungsstärkere Datenbanksysteme. Wir schauen uns im Folgenden exemplarisch die weit verbreiteten **relationalen Datenbanksysteme** an. Ein Datenbanksystem besteht aus der **Datenbasis** und dem **Datenbankmanagementsystem** (**DBMS**). Während in der Datenbasis die zu verwaltenden Daten einheitlich strukturiert abgelegt sind, organisiert das DBMS den Zugriff auf die Daten[2]. Wir schauen uns zunächst an, was das DBMS leistet, bevor wir uns ausführlicher mit der Struktur einer relationalen Datenbank beschäftigen.

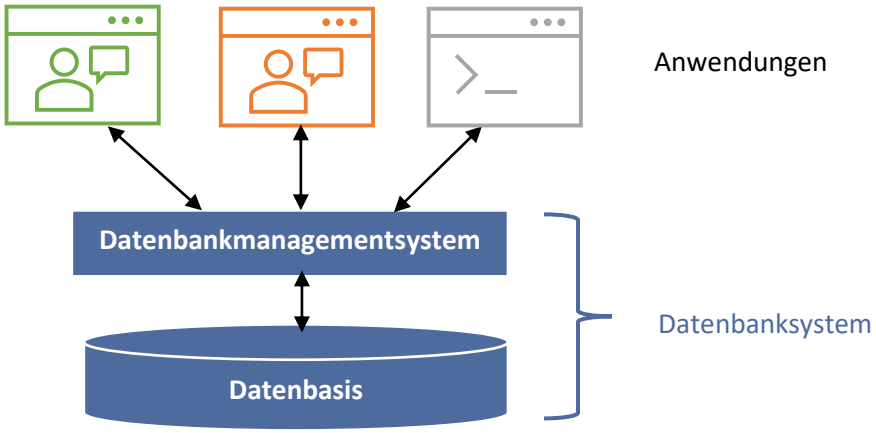

Abbildung 2: Übersicht Datenbanksystem

Das Datenbankmanagementsystem

Wenn wir Online-Dienste wie Online-Shops oder soziale Netzwerke nutzen, merken wir gar nicht, dass wir Daten in eine Datenbank schreiben oder daraus auslesen. Wir wissen auch nichts über die Struktur der Datenbasis. Das liegt daran, dass wir weder auf die Daten noch auf das DBMS direkt zugreifen. Das DBMS ist die Schnittstelle, über die Programmierer oder Datenbankadministratorinnen die Daten auslesen oder verändern können. Dazu stellt das DBMS eine **Datenmanipulationssprache** zur Verfügung. Eine solche Sprache werden wir uns am Beispiel von **SQL** (Struktured Query Language) noch genauer anschauen. SQL-Befehle können nicht nur einzeln an eine Datenbank gesendet werden, sondern auch in Anwendungen, die in anderen Programmiersprachen geschrieben wurden, eingebettet sein. So können beispielsweise auf Webseiten Daten aus einer Datenbank eingebunden werden, ohne dass der Anwender die technischen Zusammenhänge kennt.

Das DBMS koordiniert auch den **Mehrbenutzerbetrieb**. Jede Bestellung, die ein Kunde in einem Online-Shop aufgibt, oder ein Eintrag, der in einem sozialen Netzwerk gemacht wird, führt zu einer Änderung im Datenbestand. Das DBMS regelt daher in der **Transaktionsverwaltung**, dass die Anfragen der Reihe nach abgearbeitet werden und keine Änderung verloren geht. Das gilt auch für Anfragen einzelner Nutzer, die aus mehreren Teilen bestehen und aufeinander aufbauen. Außerdem wird sichergestellt, dass beim Ausführen der SQL-Befehle zum Ändern, Einfügen und Löschen keine **inkonsistenten**, d. h. keine widersprüchlichen, Daten gespeichert werden.

[2] Sprachlich werden Datenbasis und DBMS häufig nicht scharf getrennt. Wenn von Datenbanken die Rede ist, sind meistens sowohl die Datenbasis als auch das zugehörige DBMS gemeint.

Über das DBMS können auch **Zugriffsrechte** vergeben werden, so dass in einem Unternehmen zwar alle Daten in einer Datenbank gespeichert sind, jeder Mitarbeiter aber nur Zugriff auf den für ihn relevanten und zulässigen Ausschnitt der Datenbank erhält.

Um den geregelten Zugriff mehrerer Personen auf die Datenbasis zu ermöglichen, werden die Daten nicht lokal bei den Anwendern gespeichert, sondern auf einem zentralen **Server**, auf den jeder Client über das Anwendungsprogramm zugreift. Auf dem Server werden regelmäßig **Sicherungskopien** des Datenbestands erstellt.

Aufgabe 5: Vergleichen Sie Ihre Anforderungen aus Aufgabe 4 mit den Funktionen, die ein DBMS zur Verfügung stellt.

Die Datenbasis

Strukturierung der Daten

In einem Datenbanksystem werden die Daten logisch so strukturiert, dass einerseits Redundanzen vermieden werden, die Daten andererseits aber beliebig miteinander verknüpft und so z. B. verschiedene Übersichten erstellt werden können. Die Struktur einer relationalen Datenbank besteht aus einer Vielzahl von **Tabellen**[3], die miteinander in Beziehung gesetzt werden können. Dabei wird versucht für jede Kategorie von konkreten oder abstrakten Objekten der realen Welt, zu denen Daten gespeichert werden sollen, eine Tabelle anzulegen. Diese Objektkategorien bezeichnet man als **Entitäten**, die zugehörigen Daten als **Attribute**. Außerdem können die Entitäten in **Beziehungen** zueinander stehen, die ebenfalls in der Tabellenstruktur der Datenbasis abgebildet werden müssen.

Wir schauen uns das am Beispiel des Sportvereins an. Als Entitäten finden wir hier die *Mitglieder*, die *Trainier* und die *Kurse*. Mitglieder *belegen* einen oder mehrere Kurse, Trainer *leiten* Kurse. Diese Zusammenhänge lassen sich übersichtlich in einem **Entity-Relationship-Diagramm** (ER-Diagramm) darstellen (s. Abb. 3):

Im ER-Diagramm werden die Entitäten in Rechtecken dargestellt. Alle Eigenschaften (Attribute) einer Entität werden in Ovalen notiert und mit der Entität verbunden. Die Beziehungen zwischen den Entitäten werden als Rauten dargestellt. Auch Beziehungen können Attribute zugeordnet werden, wenn sich eine Eigenschaft nicht unabhängig von der Beziehung einer der beiden Entitäten zuordnen lässt. Im Beispiel wird das Attribut *Teilnahme Wettbewerbe* der Beziehung *belegt* zugeordnet, da davon ausgegangen wird, dass ein Mitglied für jeden Kurs, den es belegt, individuell entscheiden kann, ob er oder sie auch an Wettkämpfen teilnimmt.

Bei den Attributen fällt auf, dass sie so weit wie möglich in einzelne Eigenschaften zerlegt wurden: die Adresse in Hausnummer, Straße, PLZ und Ort und die Trainingszeit in Wochentag, Uhrzeit und Dauer. Man sagt, dass die Wertebereiche der Attribute **atomar** sind. Das hat später den Vorteil, dass man gezielter auf einzelne Daten zugreifen kann.

[3] Wir beschäftigen uns hier nur mit der Struktur der Daten auf der logischen Ebene, mit der Programmierer arbeiten. Die Abbildung der Tabellen auf den physischen Speicher übernimmt das Datenbanksystem und wird von Datenbankadministratoren optimiert.

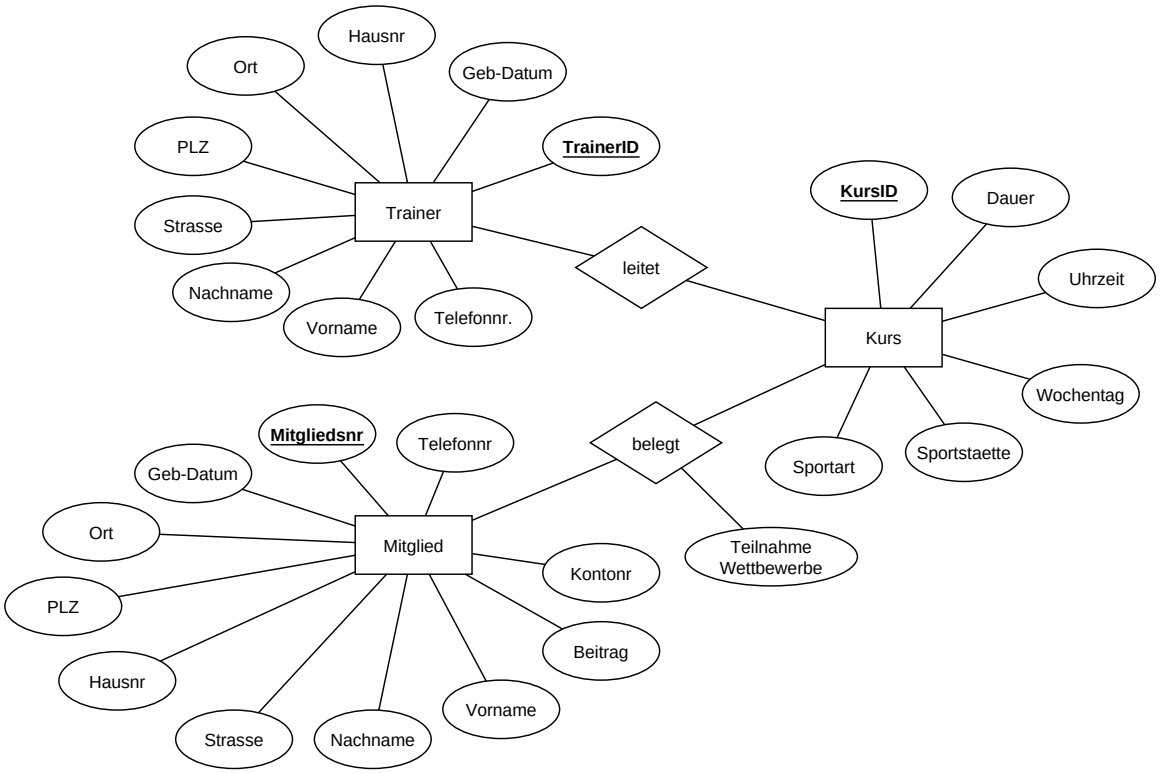

Abbildung 3: Enitity-Relationship-Diagramm des Sportvereins TSV Fitstadt

Abbildung der Datenstrukturierung auf Tabellen

Die Entitäten mit ihren Attributen und Beziehungen sind nun die Grundlage für die Struktur der Tabellen (s. Abbildung 4).

Aufgabe 6:

a) Beschreiben Sie am Beispiel der Tabellen in Abbildung 4, wie die Tabellen einer relationalen Datenbank aufgebaut sind. Stellen Sie eine Vermutung auf, wie die Daten verschiedener Tabellen miteinander verknüpft werden.

b) Beschriften Sie Abbildung 4 mithilfe des folgenden Textes mit den Begriffen *Tabelle*, *Attribut*, *Datensatz*, *Primärschlüssel* und *Fremdschlüssel*.

Für jede Entität wird eine Tabelle angelegt. In der Tabelle erhält jedes Attribut eine Spalte. Die Daten zu einem konkreten Mitglied oder einem einzelnen Kurs werden in eine Zeile der Tabelle eingetragen. Eine solche Zeile wird als **Datensatz** bezeichnet. Ein Attribut ist im ER-Diagramm jeweils dick gedruckt und unterstrichen. Dieses Attribut ist der **Primärschlüssel.** Anders als bei den anderen Attributen darf jeder Wert dieses Attributs in der Tabelle nur einmal vorkommen, so dass man anhand dieses Wertes einen Datensatz **eindeutig** identifizieren kann. Wenn es kein einzelnes Attribut gibt, dessen Werte einen Datensatz eindeutig identifiziert, können auch mehrere Attribute zu einem Primärschlüssel kombiniert werden. Dann muss die Kombination der Attributwerte eindeutig sein. Es gibt jedoch die Anforderung, dass die Menge der Schlüsselattribute **minimal** sein muss, das heißt wenn es z. B. zwei Attribute gibt, deren Werte jeden Datensatz eindeutig identifizieren, darf kein weiteres zum Primärschlüssel hinzugenommen werden. Um sich auf ein Attribut beschränken zu

können und sicher zu sein, dass dieses eindeutig ist, werden in der Praxis häufig künstliche Schlüsselattribute in Form einer ID in den Tabellen ergänzt.

Die Primärschlüssel werden nun auch verwendet, um die Tabellen miteinander zu verknüpfen und die Beziehungen abzubilden. Ein Kurs wird von nur einem Trainer geleitet. Um diese Beziehung in den Tabellen darzustellen, erhält die Tabelle *Kurs* daher eine weitere Spalte *Trainer*, in die der Wert des Primärschlüssels der jeweiligen Trainers eingetragen wird, in diesem Fall also die *TrainerID*. Über die *TrainerID* würde man in der Tabelle *Trainer* nun alle weiteren Daten, die zu dem Trainer gehören, finden. Man bezeichnet ein Attribut, das den Primärschlüssel einer anderen Tabelle bildet und als Verweis auf einen Datensatz in dieser Tabelle aufgenommen wird, als **Fremdschlüssel**.

Da die Mitglieder unterschiedlich viele Kurse belegen und die Kurse unterschiedlich viele Teilnehmer haben, lässt sich die Beziehung *belegt* nicht in einer zusätzlichen Spalte der Tabellen Mitglied oder Kurs ablegen. Listen wollten wir ja als Werte vermeiden, da sie keine atomaren Werte darstellen. Wir bräuchten also pro Mitglied bzw. pro Kurs in der jeweils anderen Tabelle eine Spalte, wüssten aber nicht, wie viele Spalten wir als Maximum anlegen müssten.

Für die Beziehung *belegt* legen wir daher eine zusätzliche Tabelle *Kursbelegungen* an, in die wir für jeden Kurs, den ein Mitglied belegt, einen Datensatz eintragen. Die Attribute der Tabelle ergeben sich aus den Primärschlüsseln der beteiligten Entitäten. In diesem Fall die *Mitgliedsnr* und die *KursID*. Diese Attribute werden hier als Fremdschlüssel eingetragen. Die Kombination dieser beiden Attribute bildet aber gleichzeitig auch den Primärschlüssel der Tabelle *Kursbelegung*.

Aufgabe 7: Ergänzen Sie zu der folgenden Beschreibung geeignete Datensätze in den Tabellen in Abbildung 4.
Trainerin Getrud Wiesel leitet dienstags von 19:00 bis 20:00 Uhr den Kurs Kunstturnen mit der ID KTG in Trainingsraum 1. An dem Kurs nehmen Armin Bauer, Amalia Liebig und Sara Meier teil. Der Beitrag für den Kurs beträgt 20 €.

Aufgabe 8:

a) Begründen Sie, dass es nicht ausreicht, eines der Attribute der Tabelle *Kursbelegung* als Primärschlüssel festzulegen.
b) Untersuchen Sie, welche Attribute sich als Primärschlüssel für die Tabelle *Trainer* eignen würden, wenn man auf das Attribut *TrainerID* verzichten möchte.

Aufgabe 9:

a) Beschreiben Sie, wie man vorgehen müsste, um
   - eine Liste aller Zeiten zu erstellen, zu denen Sporthalle 2 belegt ist.
   - eine Liste aller Mitglieder mit Vor- und Nachname zu erstellen, die am Handballkurs HHE teilnehmen.
   - eine Liste aller Mitglieder zu erstellen, die Handball spielen.
b) Nennen Sie Beispiele für weitere Informationen, die man aus den Daten gewinnen könnte.

Trainer

TrainerID	Nachname	Vorname	Strasse	Hausnr	PLZ	Ort	Telefon	Geb-Datum
…	…	…	…	…	…	…	…	…
25	Hüpfer	Rudi	Walstr.	7	23413	Fitstadt	0171-830218432	1993-01-23
26	Wiesel	Gertrud	Turmstr.	67	23411	Neudorf	08234 - 578612349	1980-06-24
…	…	…	…	…	…	…	…	…

Mitglied

Mitgliedsnr	Nachname	Vorname	Strasse	Hausnr	PLZ	Ort	Telefon	Geb-Datum	Kontonr	Beitrag
…	…	…	…	…	…	…	…	…	…	…
124	Ahorn	Rainer	Uferweg	3	23413	Fitstadt	08234-123412	2000-02002	DE12 2654 6879 5587 5316 75	40
632	Bauer	Armin	Seestr.	56	23413	Fitstadt	0141-3265987	2001-08-15	DE34 7852 6487 3254 1279 58	40
701	Lang	Stefan	Hauptstr.	90	23411	Neudorf	0179 - 6523125789	1999-01-18	DE83 7654 3495 7612 9458 87	40
…	…	…	…	…	…	…	…	…	…	…

Kurs

KursID	Sportart	Sportstaette	Wochentag	Uhrzeit	Dauer	Trainer
…	…	…	…	…	…	…
HHE	Handball	Sporthalle 2	Mittwoch	17:30	90	25
HHJ	Handball	Sporthalle 2	Donnerstag	17:00	60	25
…	…	…	…	…	…	…

Kursbelegung

Mitgliedsnr	KursID	Teilnahme Wettbewerbe
…	…	…
124	HHE	Ja
632	HHE	Nein
701	HHE	Ja
…	…	…

Abbildung 4

Kurzschreibweise für Tabellen

Die Tabellenstruktur einer relationalen Datenbank bezeichnet man als **(Relationen-)schema**. Um das Relationenschema einer Datenbank kompakt zu beschreiben, wird für die Tabellen häufig eine Kurzschreibweise ohne exemplarische Daten verwendet. Dazu wird der Tabellenname mit einer Aufzählung aller Attribute in Klammern angegeben. Die Attribute, die den Primärschlüssel bilden, werden unterstrichen. Fremdschlüsselattributen wird ein Pfeil vorangestellt. Für die Tabellen in Abbildung 4 ergibt sich somit die folgende Kurzschreibweise:

Trainer (TrainerID, Nachname, Vorname, Strasse, Hausnr, PLZ, Ort, Telefon, Geb-Datum)

Mitglied (Mitgliedsnr, Nachname, Vorname, Strasse, Hausnr, PLZ, Ort, Telefon, Geb-Datum, Kontonr, Beitrag)

Kurs (KursID, Sportart, Sportstaette, Wochentag, Uhrzeit, Dauer, ↑Trainer)

Kursbelegung (↑Mitgliedsnr, ↑KursID, Teilnahme Wettbewerbe)

Aufgabe 10: Die Datenbank eines Oberstufensekretariats umfasst die folgenden Tabellen.

Schueler (ID_nummer, Name, Vorname, Geburtstag, Geburtsort, Staatsang, Geschlecht, Konfession, PLZ, Ort, Ortsteil, von_Schulform, Tutor_in_11, Tutor_in_12_13, WDH_11, WDH_12, WDH_13)

Kurse (Kursnummer, KHJ, Kursthema, Kurslehrerkurz, Kursart)

Hatkurs (↑ID_Nummer, ↑Kursnummer, Punkte)

a) Beschreiben Sie die in der Datenbank gespeicherten Daten unter Verwendung der folgenden Begriffe: *Tabelle, Datensatz, Attribut, Primärschlüssel, Fremdschlüssel* und *Beziehung*
b) Stellen Sie das Datenbankschema, das durch die Tabellen in Kurzschreibweise gegeben ist, in einem ER-Diagramm dar.

Exkurs: Entwurf von Datenbanken

Die Strukturierung der Daten mithilfe von Tabellen, Attributen, Primär- und Fremdschlüssel ist für relationale Datenbanken festgelegt. Die Struktur der Tabellen lässt sich dabei nach festen Regeln aus dem ER-Diagramm ableiten. Der Aufbau des ER-Diagramms ist jedoch eine Modellierung und damit keineswegs eindeutig. In einem ER-Diagramm kann immer nur ein Ausschnitt der realen Welt abgebildet werden. Welche Entitäten, Attribute und Beziehungen dabei relevant sind, hängt davon ab, zu welchem Zweck Daten gespeichert und verarbeitet werden sollen. In diesem Kapitel verwenden wir ER-Diagramme nur zur Veranschaulichung der Struktur einer konkreten Datenbasis. Den Entwurf eines geeigneten Modells in Form eines ER-Diagramms stellen wir hinten an. In Aufgabe 11 können Sie sich aber bereits erste entsprechende Gedanken machen.

Aufgabe 11:

a) Vergleichen Sie das ER-Diagramm, mit den Daten, die Sie in Ihrer Mindmap in Aufgabe 2a erstellt haben. Welche Attribute würden Sie noch ergänzen. Gibt es auch weitere Entitäten oder Beziehungen, die Ihnen relevant erscheinen?
b) Beurteilen Sie die Modellierungsentscheidung, die Sportstätte als Attribut der Entität *Kurs* abzubilden.
c) Beurteilen Sie die Modellierungsentscheidung, die beteiligten Personen in zwei Entitäten, *Trainer* und *Mitglieder*, aufzuteilen.

Die Structured Query Language (SQL)

Die Structured Query Language ist ein Standard für eine **Datenmanipulationssprache** (DML) für relationale Datenbanken. Je nachdem welches relationale Datenbanksystem man verwendet, kann die Implementierung des SQL-Standards leicht variieren. Die grundlegenden Befehle sind jedoch durch den Standard festgelegt. Neben der Möglichkeit Daten zu verändern, einzufügen und zu löschen, enthält eine DML auch eine **Anfragesprache**, mit der Daten gezielt aus der Datenbank ausgelesen werden können. Wir schauen uns hier nur den Teil von SQL an, der der Bezeichnung Anfragesprache (Query Language) gerecht wird. SQL bietet aber auch Befehle zum Einfügen, Löschen und Ändern von Datensätzen an.

Auch wenn es sich bei SQL um eine Art Programmiersprache handelt, ist der Aufbau nicht direkt mit anderen Sprachen vergleichbar, die Sie aus dem Unterricht kennen. Während es sich bei Processing, Java, Snap*!* usw. um objektorientierte Sprachen mit einem imperativen Kern handelt, zählt SQL zu den deklarativen Sprachen. Wenn wir ein Programm in einer **imperativen Sprache** erstellen, müssen wir uns genau überlegen, **wie** das gegebene Problem Schritt für Schritt mithilfe der Anweisungen, die die Programmiersprache zur Verfügung stellt, gelöst werden kann. Bei einer **deklarativen Sprache** müssen wir uns über das Wie hingegen keine Gedanken machen, sondern nur exakt beschreiben, **was** ausgegeben werden soll. Das hat den Vorteil, dass wir uns weiter auf der logischen Ebene bewegen können und nicht wissen müssen, wie die Daten im Speicher abgelegt sind.

Um das Was zu beschreiben, genauer welche Daten ausgegeben werden sollen, stellt SQL eine feste Befehlsstruktur zur Verfügung. Im Datenbanksystem ist ein Algorithmus implementiert, der die Anfrage dann optimiert und herausfindet, wie die entsprechenden Daten möglichst effizient aus der Datenbasis ausgelesen werden.

Übungen

Um die Anwendung der eingeführten SQL-Befehle kleinschrittig üben zu können, enthält jeder Abschnitt eine Aufgabe, die sich auf die beiliegende Datenbank einer Oberstufe[4] (*Oberstufe.db*) bezieht. Die Datenbank enthält die Tabellen aus Aufgabe 10. Es handelt sich um eine SQLite-Datenbank, die mit entsprechenden frei verfügbaren Tools[5] geöffnet und bearbeitet werden kann.

Für eine vertiefte Übung können die Angebote der Universität Bayreuth und der Lichtenbergschule Darmstadt ergänzend bearbeitet werden, die im Folgenden kurz vorgestellt werden.

Die Universität Bayreuth stellt auf ihrem Webserver zwei relationale Datenbanken zur Übung von SQL-Anfragen zur Verfügung. Die eine Datenbank enthält Daten zur aktuellen Saison der Fußball-Bundesliga, die andere enthält Daten des deutschen Wetterdienstes. SQL-Anfragen können über die Webseiten https://dbup2date.uni-bayreuth.de/bundesliga.html bzw. https://dbup2date.uni-bayreuth.de/wetterdaten.html gestellt werden. Sowohl zu der Fußball-Bundesliga-Datenbank als

[4] Die Datenbank basiert auf den Daten einer Datenbank, die im Original unter dem Namen vlinDB2 als MySQL-Datenbank auf der Webseite der VLIN zur Verfügung gestellt wurden: http://vlin.de/zusatz/mysql/ [Datum des Zugriffs: 18.05.2018]. Die Daten wurden von Georg Beckmann anonymisiert und bereitgestellt und von der Autorin für diesen Leitfaden in Teilen angepasst.

[5] Ein entsprechendes Programm, das online im Browser verwendet werden kann, ist z. B. https://sqliteonline.com/ [Datum des Zugriffs: 09.06.2021]
Der *DB Browser for SQLite* steht als Download zur Verfügung und kann auch als portable Version installiert werden: https://sqlitebrowser.org/ [Datum des Zugriffs: 09.06.2021]

auch zur Wetter-Datenbank stehen auf den angegebenen Webseiten Aufgaben zur Verfügung, mit denen die hier vorgestellten Konzepte von SQL-Anfragen geübt werden können. Die Aufgaben zu den SQL-Anfragen sind jeweils in zwei Blöcke eingeteilt: Der erste Block bezieht sich auf Anfragen an eine Tabelle, der zweite Block auf Anfragen an mehrere Tabellen. Es bietet sich daher an, den jeweils ersten Block nach dem Abschnitt *Berechnungen mithilfe von Aggregatfunktionen* zu bearbeiten (s. Aufgabe 16) und den jeweils zweiten Block nach dem Abschnitt *Gruppen auswählen*. Einzelne Aufgaben im jeweils zweiten Block sind mit (!) gekennzeichnet, da sie Unterabfragen erfordern[6]. Dieser Aspekt wird im Folgenden nicht thematisiert.

Ergänzende Erläuterungen und Übungen finden Sie im *SQL-Tutorial*, das unter der Leitung von Gerhard Röhner an der Lichtenbergschule Darmstadt entstanden ist: https://luo-darmstadt.de/sqltut24/start.php[7].

Ausgewählte Spalten einer Tabelle ausgeben

Wir schauen uns zunächst SQL-Befehle an, die nach dem folgenden Muster aufgebaut sind:

```
SELECT ...
FROM ...
```

Hinter **FROM** geben wir den Namen der Tabelle an, aus der wir Daten auslesen möchten. Hinter **SELECT** stehen die Namen der gewünschten Attribute (Spalten). Ein Sternchen anstelle einzelner Spaltennamen bedeutet, dass alle Spalten ausgegeben werden sollen.

Zur Veranschaulichung exemplarischer SQL-Anfragen greifen wir wieder auf die Datenbank des TSV Fitstadt mit dem Relationenschema aus Abbildung 4 zurück. Beispiel 1 liefert alle Daten der Tabelle *Trainer*, während mit Beispiel 2 nur die Vor- und Nachnamen der Trainer ausgeben werden.

```
1  SELECT *
2  FROM Trainer;
```
Beispiel 1

```
1  SELECT Vorname, Nachname
2  FROM Trainer;
```
Beispiel 2

Wenn es in dem Sportverein zwei Trainer mit dem gleichen Vor- und Nachnamen gibt, würde die Reduktion der Tabelle auf die beiden Spalten *Vorname* und *Nachname* dazu führen, dass im Ergebnis zwei gleiche Datensätze angezeigt werden. Da es sich im Fall der Trainer um zwei verschiedene Personen handelt, kann eine doppelte Anzeige hier durchaus sinnvoll sein. In anderen Fällen wird es übersichtlicher, wenn doppelte Datensätze nur einmal angezeigt werden. Das erreicht man durch Angabe des Schlüsselwortes **DISTINCT** hinter SELECT. Mit der SQL-Anfrage in Beispiel 3 werden alle IDs von Trainern ausgegeben, die aktuell mindestens einen Kurs leiten. Auch wenn eine Person mehrere Kurse leitet, wird die ID nur einmal angezeigt.

```
1  SELECT DISTINCT Trainer
2  FROM Kurs;
```
Beispiel 3

[6] Die Ausführungen beziehen sich auf die Version der Webseite https://dbup2date.uni-bayreuth.de/index.html vom 11.05.2021. Der Dienst steht zum Zeitpunkt der Überarbeitung des Leitfadens nicht zur Verfügung. Es sollte also immer aktuell geprüft werden, ob die Webseite online ist.
[7] Alternativ ist das SQL-Tutorial über die Adressen https://sql-tutorial.de/home/start.php oder https://imoodle.de/ erreichbar [Datum des Zugriffs: 05.04.2024]

Aufgabe 12: Formulieren Sie geeignete SQL-Anfragen für die Datenbank *Oberstufe*.

a) Gesucht sind die Vor- und Nachnamen aller Schüler.
b) Gesucht sind alle Orte, in denen Schüler der Schule wohnen.

Datensätze einer Tabelle auswählen und sortieren

Das Ergebnis einer SQL-Anfrage ist wieder eine Tabelle, die auf die angefragten Daten reduziert wurde. Bislang können wir mithilfe des SELECT-Befehls nur die angezeigten Spalten reduzieren. Wir ergänzen nun Befehle, mit denen wir die angezeigten Datensätze selektieren und die Sortierung steuern können.

Hinter **`WHERE`** können wir Bedingungen angeben, die ein Datensatz erfüllen muss, damit er in die Ergebnistabelle aufgenommen wird. Eine Bedingung kann z. B. sein, dass ein Attribut einen bestimmten Wert hat oder in einem bestimmten Wertebereich liegt.

In Beispiel 4 werden nur die Kurse ausgegeben, die in Sporthalle 2 stattfinden und länger als 60 Minuten dauern.

```
1  SELECT *
2  FROM Kurs
3  WHERE Sportstaette = 'Sporthalle 2' AND Dauer > 60;
```

Beispiel 4

In Beispiel 4 sehen wir, dass Zeichenketten in Anführungszeichen stehen müssen. Je nach SQL-Implementierung werden einfache oder doppelte Anführungszeichen akzeptiert.

Attributwerte können auch mit einem Muster verglichen werden. Dazu stehen der Vergleichsoperator **`like`** und die Platzhalter Unterstrich `_` für genau ein beliebiges Zeichen und Prozent `%` für kein oder beliebig viele Zeichen zur Verfügung. Die Bedingung `WHERE Nachname like '%mann'` filtert alle Datensätze mit Nachnamen, die auf „mann“ enden, heraus. Die Bedingung `WHERE kurslehrerkurz like '___'` selektiert alle Lehrerkürzel mit genau drei Zeichen, da das Muster aus drei Unterstrichen besteht.

Mit **`ORDER BY`** können wir angeben, nach welcher Spalte oder welchen Spalten die Daten sortiert werden sollen. Dabei kann durch Angabe von **`asc`** bzw. **`desc`** zwischen aufsteigend und absteigend unterschieden werden. Entfällt die Angabe, wird aufsteigend sortiert.

Hinter **`LIMIT`** kann angegeben werden, wie viele Zeilen die Ergebnistabelle enthalten soll. Für eine Ergebnistabelle mit beliebig vielen Datensätzen würde z. B. die Angabe `limit 5` dazu führen, dass nur die ersten fünf Datensätze ausgegeben werden. Sinnvoll ist diese Reduzierung häufig im Zusammenhang mit einer Sortierung. So könnten z. B. die fünf ältesten Trainer gefunden werden:

```
1  SELECT Vorname, Nachname, Geb-Datum
2  FROM Trainer
3  ORDER BY Geb-Datum asc
4  LIMIT 5
```

Beispiel 5

Daten vom Datentyp `date` werden nach dem Muster JJJJ-MM-TT gespeichert. Aus dem Datum 18.01.2021 wird also der Wert 2021-01-18. Je nach Datenbanksystem ist die interne Verarbeitung unterschiedlich. SQLite-Datenbanken bilden das Datum wahlweise auf entsprechende Zeichenketten oder Ganzzahlen ab.

Je größer die einem Datum zugeordnete Zahl ist, desto länger liegt das Datum zurück. Da die Geburtsdaten in Beispiel 5 aufsteigend sortiert werden, enthalten die ersten 5 Datensätze die Personen mit dem am längsten zurückliegenden Geburtsdatum. Eine entsprechende Sortierung ergibt sich auch bei der internen Abbildung auf eine Zeichenkette.

Die Befehle `WHERE`, `ORDER BY` und `LIMIT` können auch kombiniert werden. Wichtig ist, dass immer die im SQL-Standard festgelegte Reihenfolge eingehalten wird:

```
SELECT ...
FROM ...
WHERE ...
ORDER BY ...
LIMIT ...
```

Aufgabe 13: Formulieren Sie geeignete SQL-Anfragen für die Datenbank *Oberstufe*.

a) Gesucht sind alle Schüler katholischer Konfession, sortiert nach den Nachnamen.
b) Gesucht sind alle Schüler, die nach dem 31.12.2000 geboren wurden. (Die Datenbank *Oberstufe* stellt die Datumsangaben als Zeichenketten dar.)
c) Gesucht sind die Kursnummern und Themen aller Deutschkurse. Die Kursnummern der Deutschkurse beginnen alle mit 'dt' bzw. 'Dt'.
d) Gesucht sind alle Kursthemen, die den Begriff „Gesellschaft“ enthalten und im Grundkurs (kursart 'g') angeboten werden.
e) Gesucht sind die drei jüngsten Schüler.

Aufgabe 14: Begründen Sie, warum es für das in Abbildung 4 dargestellte Relationenschema der Datenbank des TSV Fitstadt schwierig ist, eine Liste aller weiblichen Mitglieder auszugeben.

Berechnungen mithilfe von Aggregatfunktionen

Hinter SELECT können nicht nur Attribute, sondern auch Rechenoperationen angegeben werden, die auf den Werten einer Spalte ausgeführt werden sollen. Diese Operationen werden als **Aggregatfunktionen** bezeichnet. Die folgenden Aggregatfunktionen stehen zur Verfügung:

Funktion	Bedeutung
`AVG()`	Arithmetisches Mittel (Durchschnitt) der Werte
`COUNT()`	Anzahl der Werte
`MAX()`	größter Wert
`MIN()`	kleinster Wert
`SUM()`	Summe der Werte

Tabelle 1: Aggregatfunktionen

Beispiel 6 gibt die Gesamtdauer aller Kurse aus, die mittwochs in Sporthalle 2 angeboten werden. Dabei erhält die Spalte durch die Angabe eines **Alias** mithilfe von **`AS`** die Überschrift *Gesamtdauer*.

```
1  SELECT SUM(Dauer) AS Gesamtdauer
2  FROM Kurse
3  WHERE Sportstatte = 'Sporthalle 2' AND Tag = 'Mittwoch';
```

Beispiel 6

Um die Datensätze einer Tabelle zu zählen, kann auch der Ausdruck `COUNT(*)` verwendet werden. Beispiel 7 gibt die Anzahl der gespeicherten Kurse aus.

```
1  SELECT COUNT(*)
2  FROM Kurse
```

Beispiel 7

Aufgabe 15: Formulieren Sie geeignete SQL-Anfragen für die Datenbank *Oberstufe*.

a) Wie viele Lehrer unterrichten Kurse in der Oberstufe? (Vorsicht: Passen Sie auf, dass niemand doppelt gezählt wird!)
b) Wie viele Deutschkurse gibt es?
c) Welches ist die höchste, welches die niedrigste Punktzahl, die in einem Mathekurs vergeben wurde?
d) Wie viele Punkte wurden durchschnittlich in Mathematik erzielt.

Aufgabe 16[8]: Die Tabellen der Datenbanken *Fußball-Bundesliga* und *Wetter in Deutschland* sind auf den Webseiten https://dbup2date.uni-bayreuth.de/bundesliga.html bzw. https://dbup2date.uni-bayreuth.de/wetterdaten.html jeweils in Kurzschreibweise angegeben. Fremdschlüssel erhalten hier keinen vorangestellten Pfeil, sondern werden überstrichen. Neben den Namen der Attribute ist dazu auch jeweils der Datentyp angegeben. Bei den Attributen unterscheidet man ähnlich wie bei den Variablen verschiedene Datentypen wie Ganzzahl (int), Gleitkommazahl (double), Zeichenkette (char, varchar) oder Datum (date, time). Je nach Datentyp sind unterschiedliche Operationen auf den Daten erlaubt. Im Folgenden wurde die Angabe der Tabellen in Kurzschreibweise auf die in Niedersachen übliche Darstellung[9] reduziert:

Wetter in Deutschland

Wetterstation (S_ID, Standort, Geo_Breite, Geo_Laenge, Hoehe, Betreiber)

Wettermessung (↑Stations_ID, Datum, Qualitaet, Min_5cm, Min_2m, Mittel_2m, Max_2m, Relative_Feuchte, Mittel_Windstaerke, Max_Windgeschwindigkeit, Sonnenscheindauer, Mittel_Bedeckungsgrad, Niederschlagshoehe, Mittel_Luftdruck)

Fußball-Bundesliga

Verein (V_ID, Name, ↑Liga)

Spiel (Spiel_ID, Spieltag, Datum, Uhrzeit, ↑Heim, ↑Gast, Tore_Heim, Tore_Gast)

Spieler (Spieler_ID, ↑Vereins_ID, Trikot_Nr, Spieler_Name, Land, Spiele, Tore, Vorlagen)

Liga (Liga_Nr, Verband, Erstaustragung, ↑Meister, Rekordspieler, Spiele_Rekordspieler)

Bearbeiten Sie die Teilaufgaben a) bis c) in der Reihenfolge, die für Sie am hilfreichsten ist, um einen Überblick über die Struktur der Datenbanken zu bekommen.

a) Beschreiben Sie das Relationenschema der Datenbanken *Fußball-Bundesliga* und *deutsche Wetterdaten* mithilfe geeigneter Fachbegriffe.
b) Stellen Sie die Struktur der Datenbanken jeweils in einem ER-Diagramm dar.
c) Lassen Sie sich für jede Tabelle den Inhalt mithilfe eines SELECT-FROM SQL-Befehls anzeigen.
d) Bearbeiten Sie zu den Datenbanken jeweils den ersten Aufgabenblock.

[8] Wenn die Uni Bayreuth den Dienst aktuell nicht anbietet, kann alternativ Lektion 1 und Lektion 2 des SQL-Tutorials der Lichtenbergschule Darmstadt bearbeitet werden: https://luo-darmstadt.de/sqltut24/start.php
[9] Ergänzende Hinweise zum Kerncurriculum INFORMATIK für die gymnasiale Oberstufe am Gymnasium, an der Gesamtschule sowie für das Kolleg: https://www.cuvo.nibis.de/cuvo.php?p=download&upload=174 [Datum des Zugriffs: 10.06.2021]

Tabellen in Abfragen verknüpfen

Bislang haben wir nur Daten angefragt, die sich in einer Tabelle befinden. Wenn wir z. B. Name und Telefonnummer des Trainers der Handballmannschaft HHE benötigen, müssen wie die Daten von zwei Tabellen zusammenführen.

Aufgabe 17: Hinter `FROM` können durch Komma getrennt, mehrere Tabellen angegeben werden.

a) Stellen Sie eine Vermutung auf, welches Ergebnis die Datenbank *Oberstufe* für folgende SQL-Anweisung ausgibt:

```
1  SELECT schueler.ID_Nummer, schueler.Name, schueler.Vorname,
   hatkurs.ID_Nummer, hatkurs.Kursnummer, hatkurs.Punkte
2  FROM schueler, hatkurs
```

b) Führen Sie die Anfrage durch und vergleichen Sie das Ergebnis mit Ihrer Vermutung. Erläutern Sie.

Aufgabe 18:

a) Beschreiben Sie, wie Sie von Hand vorgehen, um in den Tabellen in Abbildung 4 Name und Telefonnummer des Trainers der Handballmannschaft HHE herauszusuchen.

b) Stellen Sie eine Vermutung auf, wie man dieses Vorgehen in einer SQL-Anfrage abbilden kann. Testen Sie Ihre Vermutung, indem Sie die SQL-Anfrage in Aufgabe 17a) für die Datenbank *Oberstufe* entsprechend erweitern.

Wenn wir zwei Tabellen hinter `FROM` angeben, wird das **Kreuzprodukt** der beiden Tabellen gebildet. Das heißt, es entsteht eine neue Tabelle, in der jede Zeile der ersten Tabelle mit jeder Zeile der zweiten Tabelle kombiniert wird. Dadurch ergeben sich viele Datensätze, deren Daten logisch gar nicht sinnvoll zusammenpassen. Wenn wir Tabellen verknüpfen, enthält eine der Tabellen meist den Primärschlüssel der anderen Tabelle als Fremdschlüssel. Wenn wir z. B. die Tabelle *Trainer* mit der Tabelle *Kurs* verknüpfen, so interessiert uns vermutlich zu jedem Kurs der passende Trainer, so dass wir in der Gesamttabelle nur die Datensätze haben möchten, bei denen das Attribut *TrainerID* der Tabelle *Trainer* und das Attribut *Trainer* der Tabelle *Kurs* den gleichen Wert enthalten. Damit das Kreuzprodukt der Tabellen auf diese sinnvollen Datensätze reduziert wird, müssen wir mithilfe des **WHERE**-Befehls eine geeignete Bedingung angeben. In unserem Beispiel müsste die Anfrage daher so aussehen:

```
1  SELECT Vorname, Nachname, Telefon
2  FROM Trainer, Kurs
3  WHERE Trainer.TrainerID = Kurs.Trainer AND KursID = 'HHE'
```

Beispiel 8

Um bei mehreren Tabellen deutlich zu machen, zu welcher Tabelle ein Attribut gehört, ist es sinnvoll den Tabellennamen voranzustellen. Haben zwei Tabellen ein gleichnamiges Attribut, ist das sogar zwingend notwendig. Bei langen Tabellennamen kann den Tabellen im `FROM`-Teil ein Kürzel zugewiesen werden. Dies ist vergleichbar mit einem Variablennamen.

```
1  SELECT T.Vorname, T.Nachname, T.Telefon
2  FROM Trainer T, Kurs K
3  WHERE T.TrainerID = K.Trainer and K.KursID = 'HHE'
```

Beispiel 9

Aufgabe 19:

a) Ergänzen Sie in Aufgabe 17a) eine geeignete Bedingung, so dass jedem Schüler die Kurse zugeordnet werden, die er belegt hat.
b) Geben Sie für das Relationenschema des Sportvereins in Abbildung 4 Beispiele für sinnvolle Verknüpfungen der Tabellen an.
c) Erstellen Sie eine SQL-Anfrage, die jedem Schüler die Lehrer zuordnet, bei denen er Unterricht hat.

Datensätze gruppieren

Die Aggregatfunktionen sollen manchmal nicht auf alle Datensätze, sondern auf unterschiedliche Teilmengen der Tabelle angewendet werden. Beispielsweise könnte statt der Gesamtdauer aller Kurse die Gesamtdauer der Kurse pro Trainer interessieren. Dazu besteht die Möglichkeit die Datensätze zunächst nach einem Attribut oder mehreren Attributen, die hinter **`GROUP BY`** angegeben werden, zu gruppieren. Das heißt, es werden alle Datensätze zu einer Gruppe zusammengefasst, die für die angegebenen Attribute den gleichen Wert haben. Eine Aggregatfunktion kann dann auf jede dieser Gruppen angewendet werden:

```
1  SELECT TrainerID, Vorname, Nachname, SUM(Dauer)
2  FROM Trainer T, Kurs K
3  WHERE T.TrainerID = K.Trainer
4  GROUP BY TrainerID, Vorname, Nachname
```

Beispiel 10

Da jede Gruppe in der Ausgabe auf eine Zeile reduziert wird, dürfen hinter `SELECT` nur Aggregatfunktionen oder Attribute stehen, nach den gruppiert wird und die somit den gleichen Wert haben. Deshalb müssen die Attribute *TrainerID*, *Vorname* und *Nachname* auch alle drei hinter `GROUP BY` angegeben werden, auch wenn Vorname und Nachname für eine TrainerID immer gleich sind und damit keinen zusätzlichen Einfluss auf die Gruppenbildung haben. Manche SQL-Implementierungen akzeptieren die Anfrage auch, wenn hinter `GROUP BY` nur das Attribut *TrainerID* angegeben wird.

Aufgabe 20: Formulieren Sie geeignete SQL-Anfragen für die Datenbank *Oberstufe*.

a) Gesucht sind für jeden Schüler die durchschnittlich erreichten Punkte. Die Schüler mit den meisten Punkten sollen zuerst angezeigt werden.
b) Gesucht sind für jeden Schüler die Anzahl der Kurse, die er belegt hat. Benennen Sie die Spalte mit der Anzahl der Kurse geeignet.

Gruppen auswählen

Hinter `WHERE` können wir Bedingungen angeben, die für jeden einzelnen Datensatz festlegen, ob er im Ergebnis berücksichtigt werden soll. Möchten wir hingegen für ganze Gruppen angeben, ob diese im Ergebnis aufgenommen werden sollen, müssen wir hinter `GROUP BY` eine Bedingung mit **`HAVING`** einfügen, die sich auf die Gruppen bezieht. Ergänzen wir in Beispiel 10 die Zeile `HAVING SUM(Dauer) > 120`, werden nur die Trainer in der Ausgabe berücksichtigt, die Kurse im Gesamtumfang von mehr als 2 Stunden anbieten:

```
1  SELECT TrainerID, Vorname, Nachname, SUM(Dauer)
2  FROM Trainer T, Kurs K
3  WHERE T.TrainerID = K.Trainer
4  GROUP BY TrainerID, Vorname, Nachname
5  HAVING SUM(Dauer) > 120
```

Beispiel 11

Merke: Bedingungen hinter `WHERE` beziehen sich auf Attribute der einzelnen Datensätze. Bedingungen hinter `HAVING` beziehen sich auf Eigenschaften der einzelnen Gruppen.

Aufgabe 21: Formulieren Sie geeignete SQL-Anfragen für die Datenbank *Oberstufe*.

a) Gesucht sind die Schüler, die weniger als 34 Kurse belegt haben.
b) Gesucht sind die Schüler, deren Punktedurchschnitt im Fach Mathematik unter fünf Punkten liegt.

Weitere Übungsaufgaben finden Sie im jeweils zweiten Aufgabenblock zu den Datenbanken *Fußball-Bundesliga* und *Wetter in Deutschland* und im SQL-Tutorial der Lichtenbergschule Darmstadt in den Lektionen 3 und 4.

Zusammenfassung

Zum Zeitpunkt der Entstehung dieses Leitfadens wird in den zentralen Prüfungsaufgaben in Niedersachen die Abfragesprache SQL mit dem folgenden Sprachumfang unter Berücksichtigung der angegebenen Operatoren und Gruppenfunktionen verwendet. Verschachtelte SQL-Anweisungen werden im Rahmen der zentralen Prüfungsaufgaben nicht thematisiert[10]. Teile in eckigen Klammern sind optional. Der senkrechte Strich trennt Alternativen. Die runden Klammern dienen hier der Strukturierung und sind nicht Teil der SQL-Anweisung. Runde Klammern können aber zum Zusammenfassen von Bedingungen verwendet werden. Spalten können Attribute, Aggregatfunktionen oder Berechnungen sein. Bei `GROUP BY` und `ORDER BY` ist auch die Angabe eines Alias möglich.

SELECT-Anweisung:

```
SELECT [DISTINCT | ALL] * | spalte1 [AS alias1], spalte2 [AS alias2],
…, spaltenn [AS aliasn]
FROM tabelle1, tabelle2, …, tabellem
[WHERE bedingung1 (AND | OR) bedingung2 … (AND|OR) bedingungk]
[GROUP BY spalte1 , spalte2 , … , spalte1
 HAVING gruppenBedingung1 (AND | OR) gruppenBedingung2 … (AND | OR)
 gruppenBedingungs] ]
[ORDER BY spalte1 [ASC | DESC], spalte2 [ASC | DESC], ..., spaltet
[ASC | DESC]]
[LIMIT anzahl]
```

Operatoren für Berechnungen: +, −, *, /
Operatoren für Vergleiche in Bedingungen: =, != (ungleich), >, <, >=, <=, `NOT`, `LIKE` (mit den Platzhaltern _ und %) , `BETWEEN`, `IN`, `IS NULL`
Aggregatfunktionen: `AVG( )`, `COUNT( )`, `MAX( )`, `MIN( )`, `SUM( )`

[10] Die Aussage bezieht sich auf Ergänzende Hinweise zum Kerncurriculum INFORMATIK
für die gymnasiale Oberstufe am Gymnasium, an der Gesamtschule sowie
für das Kolleg: https://www.cuvo.nibis.de/cuvo.php?p=download&upload=174 [Datum des Zugriffs: 10.06.2021]

Interpretation von Daten

Bislang haben wir uns mit Anfragen beschäftigt, bei denen man die umgangssprachliche Formulierung der Aufgabenstellung direkt in entsprechende SQL-Anweisungen übersetzen konnte. Manchmal mussten Sie dabei sicher einige Zeit nachdenken und probieren. Aber es war immer möglich eine SQL-Anfrage zu finden, die auf Grundlage der Daten in der Datenbank die gewünschte Information liefert. Denkbar sind aber auch Fragestellungen, bei denen das nicht so einfach möglich ist.

Aufgabe 22: Diskutieren Sie, inwieweit sich die folgenden Fragen mithilfe der Daten der Datenbank *Oberstufe* und geeigneter SQL-Anfragen beantworten lassen.

a) In Deutsch ist es leichter, gute Noten zu bekommen, als in Mathe.
b) Lehrer AHG bevorzugt Mädchen in Physik.
c) Mädchen sind schlauer als Jungen.

Rechner arbeiten mit Daten. Auch in einer Datenbank sind Daten gespeichert, die wir mithilfe von SQL-Anweisungen auslesen können. Erst durch die Interpretation dieser Daten entsteht eine Information. In der Datenbank *Oberstufe* sind beispielsweise zu jedem Schüler und jedem Kurs die Noten in Punkten gespeichert. Mithilfe von SQL-Anfragen können wir einzelne Noten auslesen oder den Durchschnittswert für alle Notenpunkte des Schülers in Deutsch, Mathe oder insgesamt berechnen. Als Ergebnis erhalten wir jeweils eine Zahl. Erst wenn wir diese Zahlen in die Notenskala von 0 bis 15 einordnen und miteinander vergleichen, wird daraus die Information, in welchen Fächern der Schüler seine Stärken hat und ob er gute oder schlechte Bewertungen erzielt hat. Der Interpretation sind aber Grenzen gesetzt. An einer guten Note können wir beispielsweise nicht ablesen, ob der Schüler besonders schlau oder besonders fleißig ist. Während uns für den Fleiß die entsprechenden Daten fehlen, z. B. die Anzahl der Arbeitsstunden, ist „schlau“ eine Eigenschaft, die sich sehr schwer messen und in Zahlen ausdrücken lässt.

Aufgabe 23: Geben Sie in Bezug auf die Datenbanken *Oberstufe*, *Fußball-Bundesliga* bzw. *Wetter in Deutschland* Beispiele für Fragestellungen an, die sich nicht vollständig formalisieren lassen. Das heißt, dass eine direkte Übersetzung in eine SQL-Anfrage nicht möglich bzw. die Interpretation des Ergebnisses nicht eindeutig wäre.

Aufgabe 24: In einem fiktiven Land ohne Datenschutzgesetzte möchte eine Krankenkasse ein möglichst genaues Bild über den Gesundheitszustand ihrer Versicherten bekommen. Die Krankenkasse bittet daher eine große Online-Apotheke um eine Kooperation. Die Krankenkasse möchte Zugriff auf die Daten zu sämtlichen Bestellungen haben. Im Gegenzug verteilt sie Vorteilscoupons für diese Online-Apotheke an ihre Versicherten und übernimmt die anfallenden Kosten.

a) Diskutieren Sie folgenden Punkte
   - Ist es technisch möglich, die Daten aus dem Datenbestand der Krankenkasse und der Online-Apotheke miteinander zu verknüpfen? Gehen Sie davon aus, dass die Tabellen der Datenbank der Online-Apotheke in die Datenbank der Krankenkasse importiert werden.
   - Welche Vorteile könnten die Versicherung und die Online-Apotheke aus der Kooperation ziehen?
b) Wäre eine solche Kooperation in Deutschland möglich? Beurteilen Sie das Vorhaben der Versicherung hinsichtlich der Regelungen der DSGVO.

Hinweis

Die Materialien erheben keinen Anspruch auf Vollständigkeit hinsichtlich der für die Abiturprüfung erwarteten Kompetenzen. Verbindlich für das Abitur in Niedersachsen sind allein das niedersächsische Kerncurriculum für die gymnasiale Oberstufe sowie die ergänzenden Hinweise in der jeweils aktuellen Fassung.