

Datenbanken

Ein Skript für das Berufskolleg

Hermann Maier

21. Mai 2024

©2024 Maier, Hermann, maier@privatemail.com

Aktuelle Version inklusive Quelldateien unter

https://github.com/hoerm007/DatenbankenSkript_KaufmBK_BW

Die verwendeten Datenbanken finden sich unter

https://github.com/hoerm007/DatenbankenSkript_KaufmBK_BW/tree/main/Datenbanken

Dieses Werk unterliegt der CC BY-NC-SA 4.0 Lizenz

<https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode.de>.

Sie dürfen:

- Teilen — das Material in jedwedem Format oder Medium vervielfältigen und weiterverbreiten
- Bearbeiten — das Material remixen, verändern und darauf aufbauen

Unter folgenden Bedingungen:

- Namensnennung - Sie müssen angemessene Urheber- und Rechteangaben machen , einen Link zur Lizenz beifügen und angeben, ob Änderungen vorgenommen wurden. Diese Angaben dürfen in jeder angemessenen Art und Weise gemacht werden, allerdings nicht so, dass der Eindruck entsteht, der Lizenzgeber unterstütze gerade Sie oder Ihre Nutzung besonders.
- Nicht kommerziell - Sie dürfen das Material nicht für kommerzielle Zwecke nutzen.
- Weitergabe unter gleichen Bedingungen - Wenn Sie das Material remixen, verändern oder anderweitig direkt darauf aufbauen, dürfen Sie Ihre Beiträge nur unter derselben Lizenz wie das Original verbreiten.

Inhaltsverzeichnis

1	Entity-Relationship-Modell	4
1.1	Datenbanken	4
1.2	Grundlagen	4
1.3	Primärschlüssel und Kardinalitäten	6
1.4	Beispiel eines ERMs einer Firma	8
1.5	Darstellung auf der DB	10
1.6	Anomalien	12
1.7	Normalisieren	14
1.8	Optimierung	16
1.9	Datentypen	18
2	DBMS und SQL	19
2.1	Begriffe	19
2.2	Anlegen/Befüllen einer DB	20
2.3	Erstellen bzw. Öffnen einer Datenbank	20
2.4	Erstellen bzw. löschen von Tabellen/Entitätstypen	20
2.5	Befüllen einer Tabelle mit Daten	21
2.6	SELECT-Statement	23
2.7	WHERE-Klausel	24
2.8	DELETE-Statement	27
2.9	UPDATE-Statement	27
2.10	Funktionen	28
2.11	JOIN-Statement	29
3	Lösungen der Aufgaben	31

1 Entity-Relationship-Modell

1.1 Datenbanken - Einführung

Datenbanken enthalten, wie der Name schon sagt, große Mengen an Daten, z.B. eine Firma, die ihre Kunden mit Anschrift und die zugehörigen Bestellungen speichern muss. Das Verwalten und Durchsuchen von großen Mengen an Daten ist nicht trivial, z.B. würde Excel schnell an seine Grenzen stoßen, wenn z.B. eine Firma ihre Produkte, Kunden, Bestellungen, usw. speichern will. Datenbanken wurden genau zu diesem Zweck, dem Speichern und Bearbeiten von großen Mengen an Daten entwickelt.

Die Daten sollen übersichtlich gespeichert werden und so, dass man sie bearbeiten und durchsuchen kann. Dafür benötigt man eine Struktur. Stellen wir uns vor, eine Firma würde alle Daten, also Bestellungen, Kundendaten, Produktbeschreibungen, Preise, Rechnungen, Daten zu Angestellten, usw. einfach ausdrucken und in einen riesigen Container zusammen werfen. Die Daten wären zwar vorhanden, aber sucht man nun nach der Bestellung von John Wick, weil er sich beschwert, einen Toaster statt eines Eierkochers bekommen zu haben, so würde dies exorbitant viel Zeit in Anspruch nehmen. Würde man jedoch alle Bestellungen zusammen in einem Ordner sammeln, wäre die Suche deutlich einfacher. Die Daten sind dann strukturiert und damit übersichtlicher und einfacher zu bearbeiten. Etwas Ähnliches macht man auch mit den Daten in einer Datenbank. Sehr häufig wird das Entity-Relationship-Modell (ERM) verwendet, um die Daten zu strukturieren.

1.2 Entity-Relationship-Modell

Das Entity-Relationship-Modell (ERM) dient dazu die Struktur von Daten darzustellen, z.B., dass ein Kunde über einen Namen, Vornamen und eine Adresse verfügt. Wie genau ein bestimmter Kunde heißt oder wo er wohnt spielt für das ERM keine Rolle. Dazu wird eine Grafik, das ER-Diagramm angefertigt sowie eine Beschreibung der Elemente dieser Grafik. In unseren Beispielen werden die Elemente selbsterklärend sein und wir werden uns die Beschreibung sparen (Im obigen Beispiel ist klar, was Name, Vorname und Adresse des Kunden sind. Es ist keine zusätzliche Beschreibung notwendig).

Für die grafische Darstellungen werden wir die **Chen-Notation** verwenden. Die wesentlichen Elemente eines ERMs sind:

Entitätstypen und Entitäten

Darstellung eines meistens in der Realität vorhandenen Objekts auf der Datenbank, z.B. Kunde, Schüler oder Rechnung. Der Entitätstyp ist die abstrakte Darstellung, z.B. Schüler, während eine Entität eine konkrete Ausprägung, also ein Beispiel ist. So wäre der Schüler Momen Subotic eine Entität in der Datenbank. Weitere Beispiele für Entitäten sind:

- Individuen: Person Heinrich Müller, Firma GehtganzGut, Kunde Maria Meyer
- Konkreter Gegenstand: Raum A-308, Abteilung Lohn&Gehalt, Wohnort Berlin
- Ereignis: Buchung, Mahnung, Vermietung
- Abstraktes: Unterricht, Klasse, Zahlungsart, Tagesplan

Eine Entität ist immer Mitglied einer Gruppe, des Entitätstyps. Diese kategorisiert also Entitäten mit gleichen Eigenschaften. So sind z.B. die Schülerin Christine Adler und der Schüler Christopher Jäger konkrete individuell identifizierbare Objekte, zu denen Informationen gespeichert werden. Da sie aber die gleichen Eigenschaften haben, gehören sie zum Entitätstyp Schüler. Welche Objekte so wichtig sind, dass sie als Entitätstyp in das Datenbankmodell aufgenommen werden sollen, muss sich an den funktionellen und informatorischen Zusammenhängen der zu speichernden Daten orientieren. Der Entitätstyp, also die Menge der Entitäten, wird in der grafischen Darstellung des ER-Modells, dem ER-Diagramm, als Rechteck dargestellt und die Bezeichnung eines Entitätstyps ist immer ein Substantiv.

Beziehungen

Verknüpfungen von Entitäten, z.B. ist eine Rechnung immer einem bestimmten Kunden zugeordnet. Oft kann ein Entitätstyp Beziehungen zu vielen anderen Entitätstypen haben. So könnte die Rechnung nicht nur mit einem Kunden, sondern auch mit dem Entitätstyp Produkt verknüpft sein, der die bestellten Waren angibt.

Durch Beziehungen werden die Wechselwirkungen oder Abhängigkeiten von Entitäten ausgedrückt. Beziehungen können ebenfalls Attribute (Eigenschaften) besitzen. Ein Beziehungstyp ist, analog zum Entitätstyp, die Abstraktion gleichartiger Beziehungen. Die Beziehung wird dabei meist durch Verben beschrieben und soll in Beziehungsrichtung einen vollständigen Satz ergeben.

Beispiele für Beziehungen:

- Kind gehört zu Eltern
- Team verfügt über Betreuer
- Team besteht aus Teammitglied

Der Beziehungstyp wird grafisch durch eine Raute dargestellt, die durch zwei Kanten mit den Entitätstypen verbunden ist. In der Raute steht der Name des Beziehungstyps.

Attribute

Attribute, auch als Eigenschaft oder Merkmal bezeichnet, beschreiben die Entitäten näher. Alle Entitäten eines Entitätstyps besitzen dieselben Attribute, jedoch sind die Attributswerte unterschiedlich. Attribute charakterisieren also eine Entität, einen Entitätstyp, eine Beziehung bzw. einen Beziehungstyp.

Beispiele für Attribute:

- Name, Vorname oder Adresse einer Person
- Betrag einer Rechnung oder Bestellung
- Klassengröße oder Klassenzimmer einer Klasse

In der grafischen Darstellung werden Attribute als Ellipsen oder Kreise dargestellt. Diese sind über ungerichtete Kanten mit dem Entitätstyp verbunden.



Grafische Darstellung eines Entitätstyps.



Grafische Darstellung einer Beziehung.



Grafische Darstellung eines Attributs.

Übung 1 Beantworte folgende Fragen.

1. Worin liegt der Unterschied zwischen Entitäten und Entitätstypen?
2. Worin liegt der Unterschied zwischen Attributen und Attributswerten?

Übung 2 Erstelle jeweils ein ERM

1. Ein Fahrradverleih am Bodensee verleiht Damen-, Herren- und Kinderfahrräder. Dabei wird für jedes Fahrrad ein eigener Mietvertrag abgeschlossen. Eine Person kann mehrere Fahrräder mieten. Der Fahrradverleih möchte eine Datenbank aufbauen. Helfen Sie dabei.
2. Ein befreundeter Autohändler bittet uns beim Aufbau einer Kundendatenbank zu helfen. Zuerst soll diese in einem ERM modelliert werden. Darin erscheinen sollen Kunde, Auto, Karosserietyp und Reifen. Ein Auto gehört dabei zu einem Kunden, ein Kunde kann aber mehrere Autos haben.
3. Ein DVD-Verleiher betreibt mehrere Filialen (id, strasse, plz), wo es jeweils mehrere Medien (DVDs, BluRays, Spiele) zu leihen gibt. Jeder Kunde kann nur einer Filiale zugeordnet sein. Jeder Kunde kann mehrere Medien ausleihen. Ein Mitarbeiter kann nur in einer Filiale arbeiten.

1.3 Primärschlüssel und Kardinalitäten

Der Primärschlüssel löst das Problem der Eindeutigkeit. Jede Entität, die auf der Datenbank gespeichert wird, muss eindeutig identifizierbar sein. Speichert z.B. eine Firma die Entität Kunde, so muss jeder Kunde eindeutig identifizierbar sein. Würde man als Attribute nur den

Namen und Vornamen anhängen, so könnte man Diego Maradonna aus Bremen nicht von Diego Maradonna aus Stuttgart unterscheiden. Man kann natürlich einfach zusätzliche Attribute hinzufügen, wie z.B. das Geburtsdatum oder die Adresse, bis man sich sicher ist, dass es nicht zu Verwechslungen kommen kann, aber es gibt einen eleganteren Weg. Im Normalfall hängt man eine Nummer an (z.B. die Kundennummer), die für jede Entität eine andere sein muss. So kann man jede Entität eindeutig an Hand der Nummer identifizieren. Diego mit der Kundennummer 44445 ist dann eine andere Person als Diego mit der Kundennummer 85417. Diese Nummer bezeichnet man Primärschlüssel.

Primärschlüssel

Attribut, das eine Entität eindeutig identifizierbar macht. Im Normalfall eine laufende Nummer, d.h. bei jeder neu hinzukommenden Entität wird die Nummer einfach um eins größer gemacht. Der Primärschlüssel wird im ERM durch Unterstreichen kenntlich gemacht.

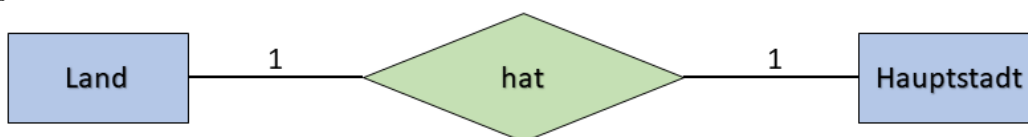
Fremdschlüssel

Wird der Primärschlüssel eines Entitätstyps an einen anderen Entitätstyp als Attribut hinzugefügt, so bezeichnet man dieses Attribut als Fremdschlüssel.

Die Kardinalität gehört zu Beziehungen und gibt an, wie viele Entitäten jeweils in Beziehung zueinander stehen können. Diesen Angaben schreibt man auf die Kanten zwischen den jeweiligen Entitätstypen und der Beziehung. Man unterscheidet im Wesentlichen folgende Kardinalitäten:

- 1:1 Beziehung

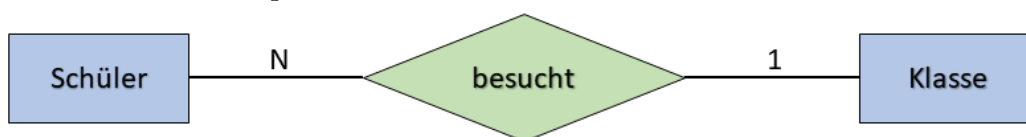
Jede Entität des einen Typs E_1 ist maximal einer Entität des anderen Typs E_2 zugeordnet und umgekehrt, z.B. hat jedes Land genau eine Hauptstadt und jede Hauptstadt liegt in genau einem Land.



Grafische Darstellung einer 1:1 Beziehung.

- 1:N Beziehung

Jeder Entität des einen Typs E_1 sind beliebig viele Entitäten des zweiten Typs E_2 zugeordnet. Umgekehrt sind jedoch jeder Entität vom Typ E_2 maximal eine Entität vom Typ E_1 zugeordnet, z.B. gehen mehrere Schüler in eine Klasse, umgekehrt geht aber ein einzelner Schüler in genau eine Klasse.



Grafische Darstellung einer 1:1 Beziehung.

- N:M Beziehung

Jeder Entität des einen Typs E_1 sind beliebig viele Entitäten des zweiten Typs E_2

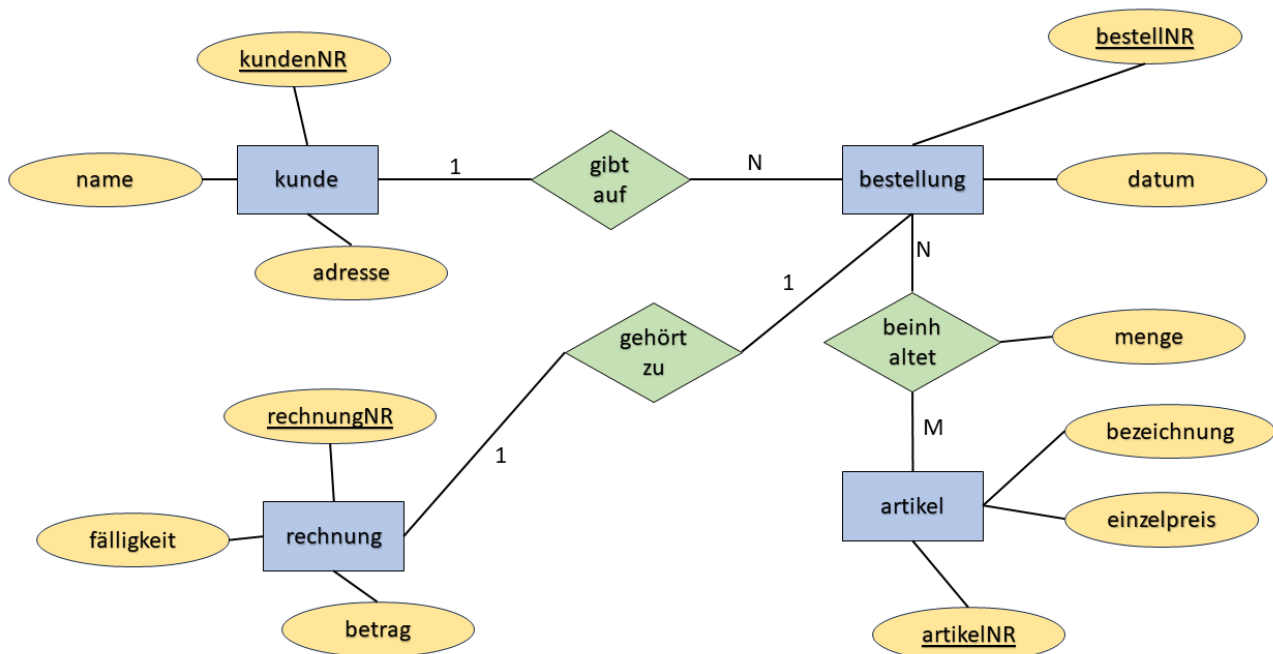
zugeordnet und umgekehrt, z.B. kann ein Mitarbeiter an mehreren Projekten gleichzeitig arbeiten und umgekehrt können an einem Projekt mehrere Mitarbeiter gleichzeitig arbeiten.



Grafische Darstellung einer 1:1 Beziehung.

1.4 Beispiel eines ERMs einer Firma

Die Kunden einer Firma können Bestellungen aufgeben, die ein oder mehrere Artikel enthalten. Zu jeder Bestellung erhält der Kunde eine Rechnung. Für die Firma wichtig sind Name und Adresse der Kunden, das Datum der Bestellung, welche Artikel enthalten sind und wie teuer diese sind sowie der Betrag und das Fälligkeitsdatum der Rechnung:



Beispiel für ein ERM. Die Primärschlüssel sind jeweils unterstrichen. Die Kardinalitäten der Beziehungen ergeben sich aus folgenden Überlegungen:

- Kunde zu Bestellung eine 1:N Beziehung. Ein Kunde kann mehrere Bestellungen aufgeben, aber jede Bestellung ist genau einem Kunden zugeordnet.
- Rechnung zu Bestellung eine 1:1 Beziehung. Hinter jeder Rechnung verbirgt sich genau eine Bestellung und zu jeder Bestellung wird genau eine Rechnung erstellt.
- Bestellung zu Artikel eine N:M Beziehung. In jeder Bestellung können mehrere (M verschiedene) Artikel vorkommen und umgekehrt kann ein Artikel in verschiedenen Bestellungen (N verschiedene) vorkommen.

Übung 3 Vervollständige die ERMs aus Aufgabe 2. Jeder Entitätstyp muss einen Primärschlüssel haben und ergänze die Kardinalitäten.

Übung 4 Erstelle ein ERM. In der Oberstufe eines Gymnasiums wird nicht mehr in Klassen, sondern in Kursen unterrichtet. Sie erhalten von der Schulleitung den Auftrag, eine Kursverwaltung mittels des Entity-Relationship-Modells zu modellieren. Mit Hilfe dieser Kursverwaltung soll festgehalten werden, welche Schüler welche Kurse besuchen. Als Schülerdaten soll neben dem Vornamen und Nachnamen der Schüler auch die individuelle Schülernummer, das Geburtsdatum, Geschlecht sowie die Postadresse festgehalten werden. Jeder Kurs hat eine eigene Kursnummer. Außerdem sind der Kurstyp (5stündig / 2stündig), das Fach (z.B. D, M, E, ...) und die Jahrgangsstufe (K1 / K2) zu speichern. In dem neuen System sollen auch die Fehlstunden und Kursnoten jedes Schülers dokumentiert werden. Jeder Kurs ist einem Lehrer zugeordnet. Als lehrerspezifische Daten sollen dessen Vor- und Nachname, das Kürzel und seine Fächer (max. 2) mit in das Kursverwaltungsprogramm aufgenommen werden.

1.5 Darstellung auf der Datenbank

Einerseits ist die Darstellung eines ERMs auf der Datenbank simpel. Jeder Entitätstyp und jede Beziehung kann als Tabelle dargestellt werden. Andererseits ergeben sich aus der Darstellung bestimmte Regeln zum Erstellen eines guten ERMs, die wir unter dem Kapitel Normalisierung besprechen werden.

Entitäten/-typen und Beziehungen auf der Datenbank

Entitäten bzw. Entitätstypen und Beziehungen werden als Tabellen auf der Datenbank vorgestellt. Eine Tabelle besteht aus einer Überschrift, die den Entitätstyp bzw. die Beziehung angibt, den Spaltenüberschriften, die die Attribute darstellen und den Zeilen mit konkreten Werten. Die Werte einzelner Zellen stehen dabei für die Attributswerte während eine ganze Zeile eine Entität repräsentiert.

Beispiel für die Entitätstypen Schüler, Lehrer und Abteilung. Für die Namen auf der Datenbank wird im Skript folgende Notation verwendet: **schueler**

schueler

schuelerNR	name	klasse	klassenlehrer
15	Waylon Smithers	BK22	Krabappel
24	Moe Szyslak	BK13	Skinner
102	Homer Simpson	BK22	Krabappel
9	Apu Nahasapeemapetilon	WG32	Krabappel
47	Carl Carlson	BK12	Skinner

abteilung

abteilungNR	bezeichnung	abteilungsleiter
1	Berufskolleg	Krabappel
2	Wirtschaftsgymnasium	Hoover

lehrer

lehrerNR	name	kuerzel
24	Krabappel	KRB
30	Skinner	SKI
31	Hoover	HOO

Wie wir später sehen werden, ist das gewählte ERM noch stark verbesserungsfähig. Vorerst wollen wir aber jeden Lehrer genau einer Abteilung zuordnen. Diese Beziehung kann ebenfalls über eine Tabelle dargestellt werden:

abteilung_lehrer

lfdNR	abteilungNR	lehrerNR
1	1	24
2	1	30
3	2	31

In der Tabelle **abteilung_lehrer** könnte man die **lfdNR** als Primärschlüssel auch streichen und

stattdessen die **abteilungNR** und **lehrerNR** zusammen als Primärschlüssel verwenden. Außer dem geringfügig größeren Speicherbedarf spricht aber für unsere Zwecke nichts gegen das Hinzufügen einer laufenden Nummer als Primärschlüssel. Die erste Entität bzw. Zeile sagt nun aus, dass der Abteilung mit der **abteilungNR** 1, also dem Berufskolleg, der Lehrer mit der **lehrerNR** 24, also Krabappel, angehört. In der zweiten Zeile bzw. Entität steht, dass dem Berufskolleg außerdem auch Skinner angehört. Und die dritte Zeile besagt, dass dem Wirtschaftsgymnasium Hoover angehört.

Beachte, dass der Primärschlüssel auch hier unterstrichen ist und aus Gründen der Übersichtlichkeit immer an erster Stelle steht.

Übung 5 **Erstelle zu den ERMs aus Aufgabe 2 passende Tabellen.**

1.6 Anomalien

Von Anomalien spricht man, wenn die Daten auf der Datenbank inkonsistent sind, also fehlerhaft. Wir unterscheiden folgende Anomalien:

Anomalien

1. **Änderungs-Anomalien:** Diese können auftreten, wenn Attributswerte an mehreren Stellen geändert werden sollen, jedoch nicht alle Stellen geändert werden.
2. **Einfüge-Anomalien:** Diese können auftreten, wenn das Einfügen eines Attributswertes zum zwingenden Einfügen weiterer Attributswerte führt.
3. **Lösch-Anomalien:** Diese können auftreten, wenn das Löschen einer Entität das ungewollte Löschen wichtiger Infos (Attributswerte) mit sich bringt.

Zudem entspricht es dem Best Practice Redundanzen (das Speichern der gleichen Information an mehreren Stellen in der Datenbank) zu vermeiden. Betrachten wir folgendes Beispiel:

Unsere Schüler engagieren sich in unterschiedlichen Schulprojekten. Der Verbindungslehrer Herr KeinDBProfi hat zu Verwaltungszwecken folgende Tabelle angelegt:

projektinfos							
<u>schNR</u>	name	vorname	klasse	klassenbez	projektNR	probez	prostd
1	Müller	Marius	BK22	Kaufm. BK2	1	Homepage	30
2	Kryof	Yuri	BK14	Kaufm. BK1	2	Foyergestaltung	25
3	Abadi	Ali	BK14	Kaufm. BK1	1,2	Homepage, Foyergestaltung	10, 15
4	Sanbei	Sarah	BK22	Kaufm. BK2	1,3	Homepage, Schulfest	15, 35

Nun will man folgende Änderungen vornehmen:

1. Da es sich um die ÜFA-Homepage handelt, soll die Projekt-Bezeichnung entsprechend geändert werden.
2. Das neue Projekt Abschlussfeier soll in die Tabelle aufgenommen werden.
3. Die Schüler der BK22 machen ihren Abschluss und verlassen die Schule

Die geänderte Tabelle sieht nun wie folgt aus:

projektinfos							
<u>schNR</u>	name	vorname	klasse	klassenbez	projektNR	probez	prostd
1	Müller	Marius	BK22	Kaufm. BK2	1	ÜFA-Homepage	30
2	Kryof	Yuri	BK14	Kaufm. BK1	2	Foyergestaltung	25
3	Abadi	Ali	BK14	Kaufm. BK1	1,2	Homepage, Foyergestaltung	10, 15
4	Sanbei	Sarah	BK22	Kaufm. BK2	1,3	ÜFA-Homepage, Schulfest	15 35
?					4	Abschlussfeier	

Es sind drei verschiedene Anomlien aufgetreten

1. Die Projektbezeichnung wurde nicht an allen Stellen von Homepage zu ÜFA-Homepage geändert. Es ist eine Änderungs-Anomalie aufgetreten.
2. Das Einfügen des Projekts Abschlussfeier hat zu einer Einfüge-Anomlie geführt. Da noch kein Schüler an dem Projekt arbeitet, muss der Primärschlüssel **schNR** leer bleiben, was verboten ist. Die Entität bzw. Zeile wäre dann nicht mehr an Hand des Primärschlüssels eindeutig identifizierbar.
3. Es ist eine Lösch-Anomalie aufgetreten. Löscht man die Schüler aus der BK22 aus der Tabelle, so wird auch das Projekt **Schulfest** gelöscht.

Zudem verfügt die Tabelle über Redundanzen, z.B. werden die Klassenbezeichnungen und Projektbezeichnungen mehrfach gespeichert.

1.7 Normalisieren von Datenbanken

Um Anomalien sowie Redundanzen möglichst zu vermeiden und die Datenbank einfach zu verwalten, muss diese normalisiert werden:

Erste Normalform

Eine Tabelle liegt in der ersten Normalform vor, wenn jeder Attributwert atomar vorliegt, d.h. jeder Wert ist nicht (sinnvoll) weiter zerlegbar. Zudem muss jeder Entitätstyp über einen Primärschlüssel verfügen (Ob als einzelnes Schlüsselattribut oder als Kombination mehrerer Attribute, ist zweitrangig).

Betrachten wir folgenden Sachverhalt. Ein DVD-Verleih legt folgende Tabelle an. Der Primärschlüssel besteht hier aus der kNR und filmID.

dvdVerleih

<u>kNR</u>	<u>filmID</u>	name	plz	ort	filmname	ausg
5	1002	Keanu Reeves	70180	Stuttgart	Rambo	01.02.2023
5	1003	Keanu Reeves	70180	Stuttgart	Rambo2	06.02.2023
7	2018	Will Smith	72070	Tübingen	LotR	04.02.2023

Die Tabelle liegt nicht in der ersten Hauptform vor, da man den Namen noch in Vor- und Nachname aufteilen kann. Der Vorteil ist, dass man die Tabelle dann leichter nach dem Vor- oder Nachnamen sortieren bzw. durchsuchen kann:

dvdVerleih

<u>kNR</u>	<u>filmID</u>	vorname	nachname	plz	ort	filmname	ausg
5	1002	Keanu	Reeves	70180	Stuttgart	Rambo	01.02.2023
5	1003	Keanu	Reeves	70180	Stuttgart	Rambo2	06.02.2023
7	2018	Will	Smith	72070	Tübingen	LotR	04.02.2023

Zweite Normalform

Eine Tabelle liegt in der zweiten Normalform vor, wenn sie in der ersten Normalform ist und zusätzlich keine Attribute enthält, die bereits von einem Teil eines Schlüsselkandidaten eindeutig bestimmt werden. Somit muss jedes Nichtschlüsselattribut voll funktional (d.h. ausschließlich) abhängig vom Primärschlüssel sein.

In diesem Fall ist der filmname nicht von kNR und filmID abhängig, sondern nur von einem Teil des Primärschlüssels, nämlich der filmID. Selbiges gilt für vorname und nachname, die nur von kNR abhängig sind. Um dieses Problem zu lösen, legen wir zwei zusätzliche Tabelle an:

leiht

filme

kunden

<u>kNR</u>	<u>filmID</u>	ausg	<u>filmID</u>	filmname	<u>kNR</u>	vorname	nachname	plz	ort
5	1002	01.02.2023	1002	Rambo	5	Keanu	Reeves	70180	Stuttgart
5	1003	06.02.2023	1003	Rambo2	7	Will	Smith	72070	Tübingen
7	2018	04.02.2023	2018	LotR					

Dritte Normalform

Eine Tabelle liegt in der dritten Normalform vor, wenn sie in der zweiten Normalform ist und zusätzlich keine Attribute enthält, die transitiv abhängig sind, d.h. Attribute, die nicht direkt vom Primärschlüssel abhängen.

In diesem Fall ist in der Tabelle **kunden** der **ort** nicht von der **kNR**, sondern von der **plz** abhängig. Auch hier schafft eine Aufteilung in zwei Tabellen Abhilfe. Die vollständig normalisierte Datenbank sieht dann wie folgt aus:

kunden				ort	
<u>kNR</u>	vorname	nachname	plz	plz	ort
5	Keanu	Reeves	70180	70180	Stuttgart
7	Will	Smith	72070	72070	Tübingen

filme		leiht		
<u>filmID</u>	filmname	<u>kNR</u>	<u>filmID</u>	ausg
1002	Rambo	5	1002	01.02.2023
1003	Rambo2	5	1003	06.02.2023
2018	LotR	7	2018	04.02.2023

Übung 6 Normalisiere die Tabelle **projektinfos** aus dem Kapitel 1.6.

Übung 7 Normalisiere folgende Tabelle und markiere die Primärschlüssel in deinem Ergebnis. Ein Bauunternehmer hat die folgende Tabelle erstellt, in der Daten über Bauaufträge und Daten zu den beteiligten Mitarbeitern gespeichert sind:

bauunternehmer								
aNR	auftrag	baust	PNR	mitarbeiter	plz	wohntort	kkasse	kkbeitrag
A1	Garage	Stuttgart	13	Cem Özdemir	72070	Tübingen	AOKBW	16,2
A2	Haus	Esslingen	13	Cem Özdemir	72070	Tübingen	AOKBW	16,2
A2	Haus	Esslingen	17	Christian Lindner	70794	Filderstadt	DAK	16,3

Übung 8 Normalisiere folgende Tabelle und markiere die Primärschlüssel in deinem Ergebnis.

bestellungen				
<u>kundeNR</u>	name	<u>artNR</u>	artBez	anzahl
5001	Volker Finke e.K.	8001	Schraubendreher 5mm	10
		8005	Schraubendreher 8mm	15
		8007	Schraubendreher-Set	15
5004	Hubert Hase GmbH	8001	Schraubendreher 5mm	20
		8006	Schraubendreher 10mm	20
		8007	Schraubendreher-Set	10
5007	Rudi Rüssel KG	8007	Schraubendreher-Set	25

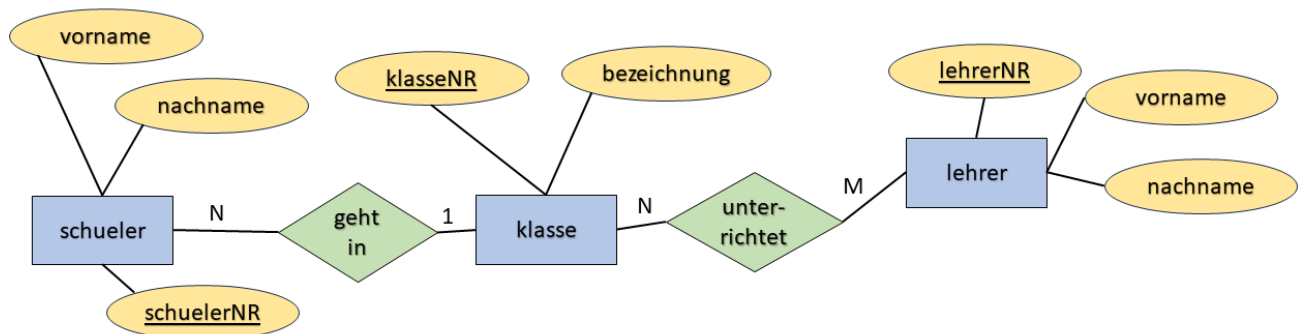
1.8 Optimierung von Datenbanken

Bisher wurde jeder Entitätstyp und jede Beziehung als eigene Tabelle auf der Datenbank dargestellt. Für alle Beziehungen, die von der Kardinalität 1:X oder X:1 sind, kann man die Informationen der Beziehungstabelle in die Tabelle eines der beiden Entitätstypen verschieben.

Optimierung

Für 1:X oder X:1 Beziehungen kann die Beziehungstabelle in einer der Entitätstyp-Tabellen integriert werden, indem man den Primärschlüssel des einen Entitätstyps in die Tabelle des anderen Entitätstyps als Fremdschlüssel hinzufügt.

Betrachten wir folgendes Beispiel:



Bildet man jeden Entitätstyp und jede Beziehung als eine eigene Tabelle ab, so erhält man z.B.:

schueler

<u>schuelerNR</u>	vorname	nachname
105	Max	Verstappen
106	Charles	Leclerc
110	Lewis	Hamilton

lehrer

<u>lehrerNR</u>	vorname	nachname
12	Christian	Horner
13	Frederic	Vasseur

schueler_klasse

<u>schuelerNR</u>	<u>klasseNR</u>
105	1
106	1
110	2

klasse

<u>klasseNR</u>	bezeichnung
1	BK22
2	BK13

lehrer_klasse

<u>klasseNR</u>	<u>lehrerNR</u>
1	12
1	13
2	12
2	13

Die Tabelle **schueler_klasse** beschreibt die N:1 Beziehung zwischen den Entitätstypen **schueler** und **klasse**. Da jedem Schüler genau eine Klasse zugeordnet wird, kann man diese Tabelle einsparen, indem man in der Tabelle Schüler als zusätzliches Attribut den Fremdschlüssel **klasseNR** einfügt, die direkt die Klasse angibt, in die der Schüler geht:

schueler				klasse	
<u>schuelerNR</u>	vorname	nachname	klasseNR	<u>klasseNR</u>	bezeichnung
105	Max	Verstappen	1	1	BK22
106	Charles	Leclerc	1	2	BK13
110	Lewis	Hamilton	2		

lehrer			lehrer_klasse	
<u>lehrerNR</u>	vorname	nachname	<u>klasseNR</u>	<u>lehrerNR</u>
			1	12
12	Christian	Horner	1	13
13	Frederic	Vasseur	2	12
			2	13

Die N:M Beziehung zwischen **klasse** und **lehrer** kann so nicht eingespart werden. Ein Lehrer unterrichtet im Normalfall mehrere Klassen, also müsste man in der Tabelle Klasse mehrere Einträge hinzufügen und umgekehrt wird eine Klasse von mehreren verschiedenen Lehrern unterrichtet.

Übung 9 Überlege dir welche Beziehungstabellen man in Aufgabe 2 wegoptimieren kann und gib an, welches Attribut man an welchen Entitätstyp als Fremdschlüssel hinzufügen muss.

1.9 Datentypen

Eine relationale Datenbank besteht aus verschiedenen **Entitätstypen** und **Beziehungen** zwischen diesen, die jeweils als Tabellen abgebildet werden. Die **Entitätstypen** und **Beziehungen** sind dabei sozusagen die Überschriften der Tabellen, die einzelnen Spalten bezeichnet man als **Attribute** und die Zeilen als Entitäten:

Schüler			
<u>schNr</u>	vorname	nachname	alter
23	Heinz	Huber	15
24	Max	Power	NULL

Schüler ist hier der **Entitätstyp** mit den **Attributen** **schNr**, **vorname**, **nachname** und **alter**. Der Schüler mit der **schNr** 23, Heinz, Huber, 15 ist eine Entität. Datenbanken benötigen meist bereits beim Erstellen eines **Entitätstyps**/Tabelle eine Angabe zum Datentyp der jeweiligen Attribute. Wir beschränken uns hier auf wenige "große" Datentypen. Je nach Datenbankmanagementsystem lassen sich die Datentypen nochmals in mehrere kleinere Untertypen aufspalten. SQLite ist ein beliebtes DBMS, da es klein und relativ simpel ist. Wir werden selbst mit SQLite arbeiten, weil außerdem eine portable Version gibt. SQLite beschränkt sich darüber hinaus bereits selbst auf wenige Datentypen:

Datentypen

- INTEGER (ganze Zahl)
- REAL (oft auch float genannt, Fließkommazahl)
- TEXT (oft auch char oder string genannt)
- BLOB
- DATE/DATETIME (Anmerkung: SQLite hat hierfür keinen eigenen Datentyp, die meisten DBMS jedoch schon. SQLite speichert ein Datum als Text oder Zahl ab)

Übung 10 Beantworte folgende Fragen; Du kannst das Internet zu Rate ziehen.

1. Informiere dich über den NULL-Wert, der oben in der Datenbank vorkommt. Für was steht dieser Wert? Was ist der Unterschied zwischen Null bzw. 0 und NULL?
2. Was ist ein Byte?
3. Wie werden INTEGER auf der Datenbank gespeichert?

2 Datenbankmanagementsystem und Structured Query Language

2.1 Begriffe

In einer Datenbank werden die Daten oder auch Informationen abgelegt. Für das Erstellen oder Ändern der Daten oder auch für die Abfrage von Daten benötigt man ein Programm, das sogenannte Datenbankmanagementsystem. So wie es verschiedene

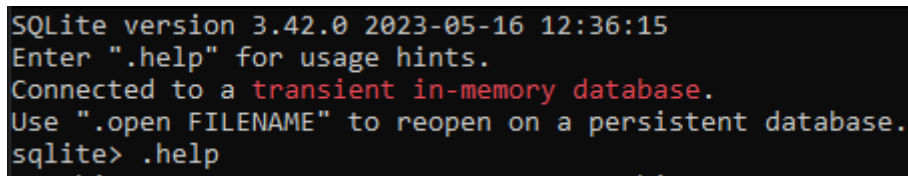
Tabellenkalkulationsprogramme wie z.B. Excel, Calc oder Numbers gibt, gibt es auch verschiedene DBMS, z.B. Oracle, Postgres oder DB2.

Obwohl es kleinere Unterschiede vor allem im Design der Oberfläche gibt, funktionieren alle Tabellenkalkulationsprogramme in großen Teilen gleich, z.B. summiert `SUM(A1; B3; C5)` die Werte der angegebenen Zellen zusammen. Etwas ähnliches gibt es für DBMS ebenfalls. Fast alle DBMS verwenden SQL bzw. Structured Query Language für das Verwalten der Datenbank. Will man z.B. eine neue Tabelle in der Datenbank erstellen, so kann man nicht einfach auf *Neu* in einer Oberfläche klicken, so wie man z.B. in Word ein neues Dokument erstellen kann, sondern muss dem DBMS eine Anweisung in Textform als SQL-Befehl geben, wie z.B.

```
CREATE TABLE farben(laufendeNR INT PRIMARY KEY, bezeichnung TEXT,  
rotanteil INT NOT NULL, blauanteil INT NOT NULL, gruenanteil NOT NULL);
```

Ein sehr entfernt ähnliches Konzept haben bei HTML mit den verschiedenen Tags kennengelernt. SQL selbst ist nicht case sensitive, d.h. die Groß- und Kleinschreibung spielt keine Rolle. Jedoch hat es sich eingebürgert die SQL-Schlüsselwörter komplett in Großbuchstaben zu schreiben und die Bezeichnungen von Tabellen/Entitäten und Attributen klein zuschreiben. Jeder SQL-Befehl endet mit einem Strichpunkt.

Wir werden als DBMS SQLite verwenden, da es nicht viel Speicherplatz braucht und es eine portable Version gibt, d.h. es ist keine Installation notwendig. SQLite kann man starten, indem man die `sqlite3.exe` startet.



```
SQLite version 3.42.0 2023-05-16 12:36:15  
Enter ".help" for usage hints.  
Connected to a transient in-memory database.  
Use ".open FILENAME" to reopen on a persistent database.  
sqlite> .help
```

Kommandozeile nach dem Starten von `sqlite3.exe`.

Alle SQLite-Befehle beginnen mit einem Punkt. Für Interessierte: Der Befehl `.help` gibt eine Übersicht der möglichen Befehle. Wir werden aber nur wenige Befehle, wie z.B. `.open bsp.db` verwenden.

2.2 SQL - Anlegen und Befüllen einer Datenbank

Erinnerung: Wir werden das Programm SQLite als Datenbankmanagementsystem (DBMS) verwenden. Nach dem Starten von SQLite sind nur noch SQLite-spezifische Befehle, die immer mit einem Punkt beginnen oder SQL-Anweisungen erlaubt.

2.3 Erstellen bzw. Öffnen einer Datenbank

Starte die `sqlite3.exe`. Mit dem Befehl

`.open #Name#.db` (z.B. `.open meineDatenbank.db`)

lässt sich eine bestehende DB öffnen bzw.

neu erstellen. Im Verzeichnis, in dem auch `sqlite3.exe`

liegt, sollte nun (falls zuvor noch nicht vorhanden)

eine neue Datei `#Name#.db` erstellt worden sein.

Mit dem Befehl `.tables` kann man sich alle in der DB

vorhandenen Tabellen/Entitätstypen anzeigen lassen.

Mit dem Befehl `.schema` kann man sich die Tabellen mit einer Liste der Attribute anzeigen lassen.

```
sqlite> .open meineDatenbank.db
sqlite> .tables
sqlite> .schema
sqlite>
```

Da noch keine Tabellen angelegt sind, geben die Befehle `.tables` und `.schema` keine Ausgabe zurück.

Datenbank öffnen

Starte `sqlite3.exe` und gib dann `.open datenbankname.db` ein.

2.4 Erstellen bzw. löschen von Tabellen/Entitätstypen

Neue Tabellen/Entitätstypen lassen sich mit dem SQL-Befehl `CREATE TABLE` erzeugen:

Tabellen erstellen

```
CREATE TABLE name_tabelle (name_attribut1 datentyp1 einschränkung1,
name_attribut2 datentyp2 einschränkung2, ...);
```

ACHTUNG: Bei SQL-Befehlen den Strichpunkt am Ende der Zeile nicht vergessen! Man kann SQL-Befehle auch auf mehrere Zeilen aufteilen.

Als Datentypen werden wir `INT` für ganze Zahlen und `TEXT` für Texte oder Geburtsdaten verwenden. Dem interessierten Leser seien die restlichen Datentypen zum Selbststudium ans Herz gelegt.

Als Einschränkungen werden wir uns auf `PRIMARY KEY` und `NOT NULL` beschränken.

`PRIMARY KEY` markiert das Attribut als Primärschlüssel und sollte als erstes Attribut angelegt werden. `NOT NULL` zeigt an, dass dieses Attribut beim Füllen der Tabelle mit Daten nicht leer bleiben darf. Die Angabe von Einschränkungen ist optional.

Bestehende Tabellen lassen sich mit dem Befehl `DROP TABLE name_tabelle` wieder löschen:

Tabellen löschen

```
DROP TABLE name_tabelle
```

```
sqlite> CREATE TABLE schueler(  
(x1...> schuelerNR INT PRIMARY KEY NOT NULL,  
(x1...> vorname TEXT,  
(x1...> geburtsdatum DATE,  
(x1...> klasse TEXT  
(x1...> );  
sqlite> .tables  
schueler  
sqlite> .schema  
CREATE TABLE schueler(  
schuelerNR INT PRIMARY KEY NOT NULL,  
vorname TEXT,  
geburtsdatum DATE,  
klasse TEXT  
);  
sqlite>
```

Die Tabelle **schueler** wurde angelegt. Die Befehle **.tables** und **.schema** haben nun einen Rückgabewert.

```
sqlite> .tables  
schueler  
sqlite> DROP TABLE schueler;  
sqlite> .tables  
sqlite>
```

Die Tabelle **schueler** wurde gelöscht.

2.5 Befüllen einer Tabelle mit Daten

Um einer Tabelle eine neue Entität/Zeile hinzuzufügen, verwendet man den **INSERT INTO** Befehl:

Befüllen einer Tabelle

```
INSERT INTO name_tabelle VALUES(wert1, wert2, wert3, ...);
```

- Alle Attribute angegeben: Sollen für alle Attribute Werte angegeben werden, so kann man einfach die Werte als kommaseparierte Liste angeben:
INSERT INTO schueler VALUES(1,'Heinz','Huber','01.01.2000','BKFH');
Beachte, dass Texte mit Anführungszeichen (neben dem #-Zeichen) angegeben werden müssen.
- Manche Attribute ohne Wert: Hat man für manche Attribute keinen Wert zur Hand, so kann man die Entität trotzdem anlegen, indem man hinter den Namen der Entität die Liste der Attribute angibt, für die man Werte zur Hand hat:

```
INSERT INTO schueler(schuelerNR, vorname, nachname, klasse) VALUES (20,  
'Anton', 'Atonovich', 'BK11');
```

Übung 11 Beantworte folgende Fragen mit Hilfe deiner Datenbank.

1. Erzeuge eine Tabelle **schueler** mit den Attributen **schuelerNR** als Primärschlüssel, der nicht NULL sein darf, **name**, **plz** und **klasse**.
2. Füge 2 verschiedene Schüler hinzu, die aus den Klassen BK13 und BK21 stammen.
3. Was passiert, wenn man einen weiteren Schüler mit einer bereits vergebenen **schuelerNR** hinzufügen will?
4. Was passiert, wenn man einen weiteren Schüler einfügen will ohne eine **schuelerNR** anzugeben?
5. Wie sehen die Ausgaben von **.tables** und **.schema** aus?

2.6 SQL - Das SELECT-Statement

Speichere die Datei `vieleSchueler.db` in deinem Verzeichnis und öffne die DB mit `sqlite3.exe` (`.open vieleSchueler.db`). Mit dem Befehl `.tables` oder `.schema` kannst du dir die Tabellen (mit Attributen) anzeigen lassen.

Wie erwartet gibt es eine Tabelle `schueler`. Um den Inhalt anzuzeigen, kannst du das SELECT-Statement von SQL verwenden:

SELECT-Statement

```
SELECT * FROM schueler;
```

Mit den Zusätzen `ORDER BY nachname ASC` bzw. `ORDER BY nachname DESC` kannst du die Ausgabe nach einem Attribut aufsteigend (ascending) bzw. absteigend (descending) sortieren lassen. Ohne die Angabe von `ASC` bzw. `DESC` erfolgt die Ausgabe standardmäßig aufsteigend.

Übung 12 Beantworte folgende Fragen mit Hilfe deiner Datenbank und dem Internet.

1. Welche Ausgabe erzeugt das Statement `SELECT * FROM schueler;`?
2. Wofür steht der Stern (*) in obigem Statement?
3. Welche Ausgabe erzeugt `SELECT vorname, nachname FROM schueler;`?
4. Finde ein Statement, um dir `nachname`, `plz` und `klasse` anzeigen zu lassen.

2.7 SQL - Die WHERE-Klausel

Im letzten Abschnitt haben wir gelernt, wie wir alle Einträge einer ganzen Tabelle oder eines/mehrerer Attribute anzeigen lassen können. Oft sind die Tabellen jedoch so groß, dass es sehr mühsam wäre, alle Einträge von Hand zu durchsuchen. Daher kann man das SELECT-Statement mit Hilfe der WHERE-Klausel einschränken. Die WHERE-Klausel kann auch an viele andere Statements angehängt werden, wie z.B. beim später noch einzuführenden DELETE-Statement.

Der grundsätzliche Aufbau ist denkbar einfach:

WHERE-Klausel

```
SQL-STATEMENT WHERE bedingungen;
```

Der Teufel steckt wie so oft im Detail, hier im Aufbau der Bedingungen. Einige wichtige Beispiele:

- = gleich, < kleiner, > größer, <= kleiner gleich, >= größer gleich, != ungleich

Beispiel: `SELECT * FROM schueler where schuelerNR<=10;` gibt alle Schueler, deren `schuelerNR` kleiner oder gleich 10 ist aus.

- `IS NULL` bzw. `IS NOT NULL`

Testet, ob ein Attribut den Wert NULL hat oder eben nicht. Ein Test mit `= NULL` funktioniert oft nicht so wie erhofft.

Beispiele: `SELECT * FROM schueler WHERE vorname IS NOT NULL;` gibt alle Schueler aus, für die beim Vornamen ein beliebiger Wert angegeben wurde. Auch Schüler, deren Vorname aus einem Text mit der Länge 0 Zeichen besteht, würden ausgegeben werden.

`SELECT * FROM schueler WHERE geburtsdatum IS NULL;` gibt alle Schüler aus, die beim Attribut `geburtsdatum` keinen Wert eingetragen haben. (Es sollten fünf Schüler ohne Geburtsdatum vorhanden sein.)

- `LIKE muster`

Wird normalerweise für den Vergleich für Attributen mit dem Datentyp `TEXT` verwendet. Für das Muster gibt man einen Text mit Platzhaltern an. Das Prozentzeichen steht für beliebig viele (Null oder mehr Zeichen), der Unterstrich für genau ein Zeichen. Beispiele:

`SELECT * FROM schueler WHERE vorname LIKE 'A%';`

Gibt alle Schüler aus, deren `vorname` mit einem A beginnt.

`SELECT * FROM schueler WHERE vorname LIKE 'A%i';`

Gibt alle Schüler aus, deren `vorname` mit einem A beginnt und einem i endet (eine Schülerin). `SELECT * FROM schueler WHERE vorname LIKE 'A___';`

Gibt alle Schüler aus, deren `vorname` mit einem A beginnt und insgesamt genau vier Zeichen lang ist (2 Schüler).

`SELECT * FROM schueler WHERE nachname LIKE 'Sch___';`

Gib alle Schüler aus, deren `nachname` mit Sch beginnt und genau 6 Zeichen hat (ein Schüler).

- **IN**

Funktioniert wie mehrere Tests auf Gleichheit (= oder **IS**). Die Vergleichswerte werden als kommaseparierte Liste angegeben:

```
SELECT * FROM schueler WHERE schuelerNR in (1,2,5);
```

Gibt die Schüler mit den **schuelerNR** 1, 2 und 5 aus.

```
SELECT * FROM schueler WHERE nachname IN ('Mayer', 'Maier');
```

Gibt alle Schüler aus, deren **nachname** Mayer oder Maier lautet.

- **BETWEEN**

Dem **IN** sehr ähnlich, diesmal kann man jedoch gleich auf einen ganzen Bereich prüfen:

```
SELECT * FROM schueler WHERE schuelerNR BETWEEN 10 AND 20;
```

Gibt alle Schüler aus, deren **schuelerNR** zwischen 10 und 20 liegen (10 und 20 jeweils eingeschlossen).

ACHTUNG: Beim Test auf NULL muss immer **IS** statt = verwendet werden, z.B. liefert

```
SELECT * FROM schueler WHERE plz is NULL;
```

 einen Schüler, während

```
SELECT * FROM schueler WHERE plz = NULL;
```

 kein Ergebnis aber auch keine Fehlermeldung liefert.

Es lassen sich mehrere Bedingungen mit einem **AND** bzw. **OR** verknüpfen:

```
SELECT * FROM schueler WHERE schuelerNR BETWEEN 10 AND 20
```

```
OR nachname IN ('Mayer', 'Maier');
```

Gibt alle Schüler aus, deren **schuelerNR** zwischen 10 und 20 liegen oder deren **nachname** Mayer oder Maier lautet.

```
SELECT * FROM schueler WHERE schuelerNR BETWEEN 10 AND 20
```

```
AND nachname IN ('Mayer', 'Maier');
```

Gibt alle Schüler aus, deren **schuelerNR** zwischen 10 und 20 liegen und deren **nachname** Mayer oder Maier lautet. (Nur eine Schülerin erfüllt beide Bedingungen gleichzeitig.)

Übung 13 Erstelle ein SELECT-Statement mit WHERE-Klausel, das

1. den Schüler mit der **schuelerNR** 31 zurück gibt.
2. alle Schüler mit der **schuelerNR** 10, 23, 50 und 65 findet.
3. alle Schüler findet, deren **nachname** auf hein endet.
4. alle Schüler findet, deren **nachname** nicht auf hein endet.
5. den Schüler findet, für den keine **klasse** angegeben worden ist.
6. alle Schüler findet, deren **schuelerNR** kleiner 18 ist.
7. alle Schüler findet, deren **vorname** aus genau 4 Buchstaben besteht.
8. Bonus-Frage: Warum gibt das Statement

```
SELECT * FROM schueler WHERE geburtsdatum BETWEEN '01.01.2000' AND '31.12.2000';
```

viel zu viele Schüler zurück?

Tipp: Probiere das Statement

```
SELECT * FROM schueler WHERE geburtsdatum BETWEEN '31.01.2000' AND '31.12.2000';
```

oder das Statement

```
SELECT geburtsdatum FROM schueler ORDER BY geburtsdatum;
```

aus.

2.8 SQL - Das DELETE-Statement

Mit Hilfe des DELETE-Statements lassen sich Einträge aus einer Tabelle löschen (Das DROP-Statement war zum Löschen der kompletten Tabelle):

DELETE-Statement

```
DELETE FROM name_der_Tabelle WHERE bedingungen;
```

ACHTUNG: Vergisst man die WHERE-Klausel oder wählt mit der WHERE-Klausel mehr Einträge aus als beabsichtigt, werden ALLE oder mehr Einträge als beabsichtigt gelöscht.

Übung 14 Lösche folgende Einträge aus der Datenbank:

1. Den Schüler mit der **schuelerNR** 40.
2. Alle Schüler mit der **schuelerNR** 10, 20, 50 und 60.
3. Alle Schüler, deren **vorname** auf ia endet.

2.9 SQL - Das UPDATE-Statement

Mit Hilfe des UPDATE-Statements lassen sich einzelne Werte eines Eintrags in einer Tabelle nachträglich ändern:

UPDATE-Statement

```
UPDATE name_der_Tabelle SET attribut1=wert1, attribut2=wert2, ... WHERE  
bedingungen;
```

ACHTUNG: Vergisst man die WHERE-Klausel oder wählt mit der WHERE-Klausel mehr Einträge aus als beabsichtigt, werden ALLE oder mehr Einträge als beabsichtigt geändert.

Übung 15 Ändere folgende Einträge aus der Datenbank:

1. Der Schüler mit der **schuelerNR** 19 soll mit **vornamen** Hans Gustav Adalbert heißen.
2. Die Bezeichnung der **klasse** soll von BK2 auf BK22 geändert werden.
3. Bei einigen Schülern ist ein Problem beim Eintragen der **plz** aufgetreten. Bei allen Schülern mit einer 4-stelligen **plz** soll diese auf NULL geändert werden.

2.10 Funktionen im SELECT-Statement

Öffne die Datenbank `schule.db` mit dem Datenbank-Browser oder `sqlite3.exe`.

SQL bietet einige einfache Funktionen an, die man innerhalb des SELECT-Statements verwenden kann:

Funktionen in SQL

```
SELECT FUNKTIONSNAME(name_attribut) FROM name_tabelle (WHERE bedingungen);
```

- **COUNT** Anzahl der Werte zählen
- **MAX** Maximalwert bestimmen
- **MIN** Minimalwert bestimmen
- **AVG** Durchschnittswert bestimmen
- **SUM** Werte summieren

Offensichtlich kann man bis auf die COUNT-Funktion als Argument sinnvoll nur Attribute mit einer Zahl als Datentyp anwenden.

Verwendung als SQL-Statements:

- **SELECT COUNT(schuelerNR) FROM schueler;**
Gibt die Anzahl der Einträge in der Tabelle `schueler` zurück.
- **SELECT COUNT(schuelerNR) FROM schueler WHERE plz=70806;**
Gibt die Anzahl der Schüler zurück, die die plz 70806 haben.
- **SELECT COUNT(lehrerNR) FROM lehrer WHERE nachname LIKE 'S%';**
Gibt die Anzahl an Lehrern zurück, deren Nachname mit einem S beginnt.

Mit dem Zusatz **GROUP BY** kann man Datensätze, die in einem Attribut übereinstimmen, zusammenfassen:

- **SELECT COUNT(lehrerNR) FROM unterrichtet;** Gibt die Anzahl der Einträge in der Tabelle `unterrichtet` zurück.
- **SELECT klasseNR, COUNT(lehrerNR) FROM unterrichtet GROUP BY klasseNR;** Gibt die Anzahl verschiedener Lehrer (eigentlich die Anzahl von Einträgen pro Klasse) für jede Klasse bzw. `klasseNR` zurück.

Übung 16 Bearbeite folgende Aufgaben

1. Erstelle ein zur Datenbank passendes ERM. Tipp: `.schema` könnte hilfreich sein.
2. Wie viele verschiedene Lehrer unterrichten an der Schule?
3. Wie viele Schüler kommen aus einer Stadt, deren PLZ mit einer 7 beginnt?
4. Wie viele Lehrer sind den einzelnen Abteilungen jeweils zugeordnet?

2.11 SQL - Das JOIN-Statement

Öffne die Datenbank `schuleOptimiert.db` mit dem Datenbank-Browser oder `sqlite3.exe`. Die Datenbank beinhaltet die gleichen Informationen wie `schule.db`, jedoch wurde die Datenbank dahingehend optimiert, dass alle zu-1-Beziehungen keine eigene Beziehungstabelle mehr haben. Dies wird das Erstellen der JOIN-Statements erleichtern. Bisher haben sich unsere SQL-Abfragen immer auf eine einzelne Tabelle bezogen. Viele Fragen lassen sich damit aber nicht oder nicht zufriedenstellend beantworten, z.B. gibt die Tabelle `lehrer` indirekt über die `abteilungNR` an, welche Lehrer welcher Abteilung zugeordnet sind. Der Anwender möchte jedoch gerne die Zuordnung der Lehrer-Namen zu den Abteilungsbezeichnungen haben, statt der Zuordnung zur `abteilungNR`. Dies kann man mit Hilfe des JOIN-Zusatzes erreichen:

JOIN-Statement

```
SELECT tabelle1.attribut1, tabelle2.attribut2, usw. FROM tabelle1  
INNER JOIN tabelle2 ON tabelle1.attributx=tabelle2.attributy, usw.;
```

Nach dem `SELECT` gibt man entweder `*` für alle Attribute oder eine Liste von Attributen an. Neu ist, dass man zwischen den Tabellen unterscheiden muss, z.B. steht `lehrer.vorname` für das Attribut `vorname` aus der Tabelle `lehrer`, während `schueler.vorname` für das Attribut `vorname` aus der Tabelle `schueler` steht.

`FROM tabelle1 INNER JOIN tabelle2` gibt die beiden Tabellen an, die man verknüpfen will. `ON bedingungen` gibt an, welche Zeilen der beiden Tabellen als eine ausgegeben werden sollen. **ACHTUNG:** Ohne `ON bedingungen` wird die Potenzmenge gebildet, d.h. jede Zeile der ersten Tabelle wird jeweils mit jeder Zeile der zweiten Tabelle zu jeweils einer Zeile zusammengefasst und ausgegeben, was bei großen Datenbanken zu langen Bearbeitungszeiten führt.

Beispiel:

`SELECT vorname, nachname, abteilungNR FROM lehrer;` Gibt die Zuordnung von Lehrern zu Abteilungen an Hand der `abteilungNR` an. Diese Darstellung ist abstrakt. Um die Namen der Lehrer und die Bezeichnungen der Abteilungen zu kennen, müssten wir in einer weiteren Tabellen nachschauen. Mit `SELECT abteilungNR, bezeichnung FROM abteilung;` können wir der `abteilungNR` die Bezeichnung zuordnen und händisch prüfen, welcher Lehrer in welcher Abteilung ist. Bei großen Datensätzen wird dies schnell mühsam.

Stattdessen können wir folgendes JOIN-Statement verwenden:

```
SELECT * FROM lehrer INNER JOIN abteilung  
ON lehrer.abteilungNR = abteilung.abteilungNR;
```

Gibt jeweils in einer Zeile einen Eintrag aus `lehrer` und `abteilung` aus, so dass die `abteilungNR` von beiden Einträgen übereinstimmen. Das DBMS nimmt also die erste Zeile aus `lehrer` in die Hand, liest die `abteilungNR` aus und geht dann in die Tabelle `abteilung`. Es prüft für jede Zeile aus `abteilung`, ob die beiden Nummern gleich sind und falls ja, klebt es die Zeilen aus `lehrer` und `abteilung` nebeneinander, z.B. lautet eine Zeile des Ergebnisses:

```
5|Olaf|Scholz|3|3|WG|Scholz
```

Der erste Teil 5|Olaf|Scholz|3 stammt aus `lehrer` mit den Attributen `lehrer.lehrerNR=5`, `lehrer.vorname=Olaf`, `lehrer.nachname=Scholz` und `lehrer.abteilungNR=3`. Der zweite Teil 3|WG|Scholz stammt aus `abteilung` mit den Attributen `abteilung.abteilungNR=3`, `abteilung.bezeichnung=WG` und `abteilung.abteilungsleiter=Scholz`. Diese beiden Teile wurden zu einer Zeile zusammengefasst, weil die `abteilungNR` bei beiden 3 ist.

Übersichtlicher ist es, wenn man sich nur die Informationen ausgeben lässt, die relevant sind:

```
SELECT lehrer.vorname, lehrer.nachname, abteilung.bezeichnung FROM lehrer  
INNER JOIN abteilung ON lehrer.abteilungNR = abteilung.abteilungNR;
```

liefert als erste Zeile Olaf|Scholz|WG.

Übung 17 Bearbeite folgende Aufgaben

1. Erzeuge eine Ausgabe, die dem Vor- und Nachnamen der Lehrer jeweils die passenden Abteilungsbezeichnungen zuordnet.
2. Erzeuge eine Ausgabe, die dem Vor- und Nachnamen aller Schüler jeweils die Bezeichnung der passenden Klasse zuordnet.
3. Erzeuge eine Ausgabe, die jeder Klassenbezeichnung die Anzahl der Schüler der Klasse zuordnet. Tipp: COUNT-Funktion verwenden.
4. Erzeuge eine Ausgabe, die dem Vor- und Nachnamen aller Schüler jeweils den passenden Schultyp zuordnet. Tipp: Man muss zwei JOIN-Statements verwenden.

3 Lösungen der Aufgaben

Lösung zu Übung 1

1. Worin liegt der Unterschied zwischen Entitäten und Entitätstypen?

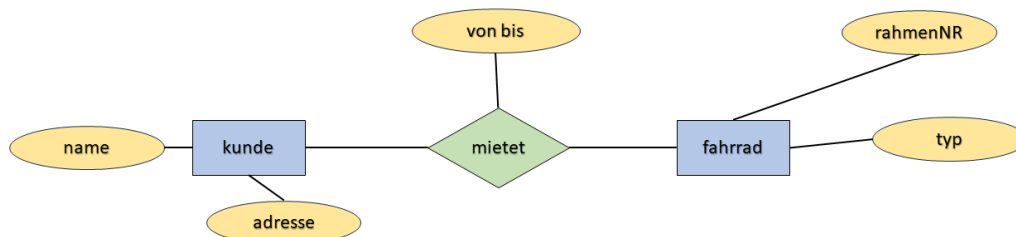
Ein Entitätstyp ist der abstrakte Übergriff, z.B. Schüler, während eine Entität ein konkretes Beispiel ist, z.B. der Schüler Noah Schimek ist eine Entität vom Entitätstyp Schüler.

2. Worin liegt der Unterschied zwischen Attributen und Attributswerten?

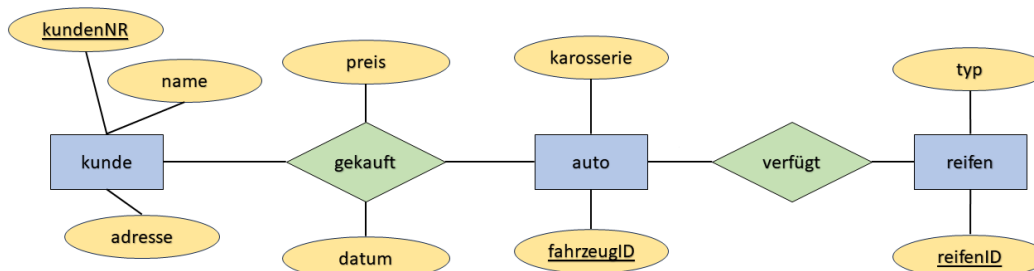
Auch hier ist das Attribut die abstrakte Eigenschaft, z.B. hat der Entitätstyp Schüler ein Attribut Namen. Den Wert eines Attributs erhält man, wenn man eine bestimmte Entität betrachtet, z.B. hat das Attribut Namen beim obigen Schüler den Wert Noah Schimek.

Lösung zu Übung 2 Anmerkung: Die ERMs sind nur Lösungsvorschläge. Man kann auch zusätzliche Attribute hinzufügen. Zudem sind die ERMs nicht vollständig, wir werden später noch neue Begriffe kennen lernen, die hier noch fehlen.

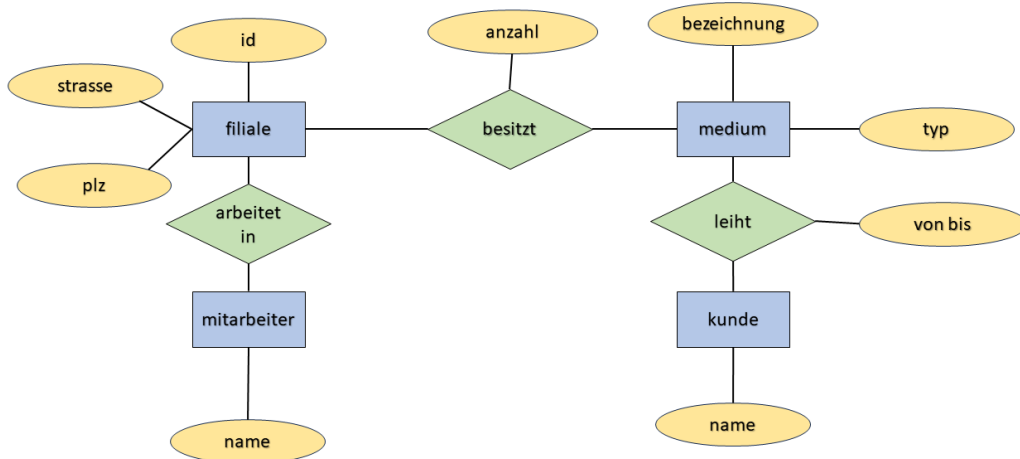
1. Ein Fahrradverleih am Bodensee verleiht Damen-, Herren- und Kinderfahrräder. Dabei wird für jedes Fahrrad ein eigener Mietvertrag abgeschlossen. Eine Person kann mehrere Fahrräder mieten. Der Fahrradverleih möchte eine Datenbank aufbauen. Helfen Sie dabei.



2. Ein befreundeter Autohändler bittet uns beim Aufbau einer Kundendatenbank zu helfen. Zuerst soll diese in einem ERM modelliert werden. Darin erscheinen sollen Kunde, Auto, Karosserietyp und Reifen. Ein Auto gehört dabei zu einem Kunden, ein Kunde kann aber mehrere Autos haben.

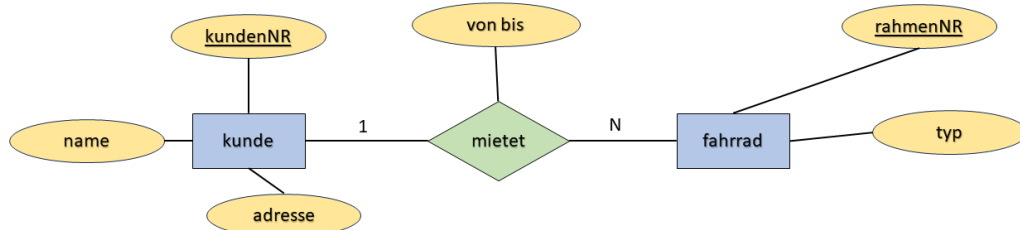


3. Ein DVD-Verleiher betreibt mehrere Filialen (id, strasse, plz), wo es jeweils mehrere Medien (DVDs, BluRays, Spiele) zu leihen gibt. Jeder Kunde kann nur einer Filiale zugeordnet sein. Jeder Kunde kann mehrere Medien ausleihen. Ein Mitarbeiter kann nur in einer Filiale arbeiten.

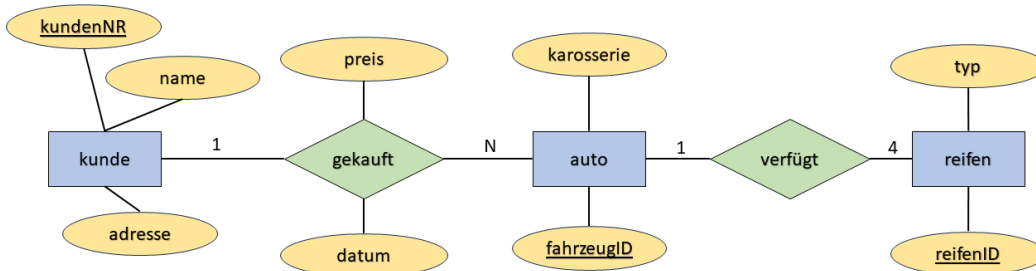


Lösung zu Übung 3

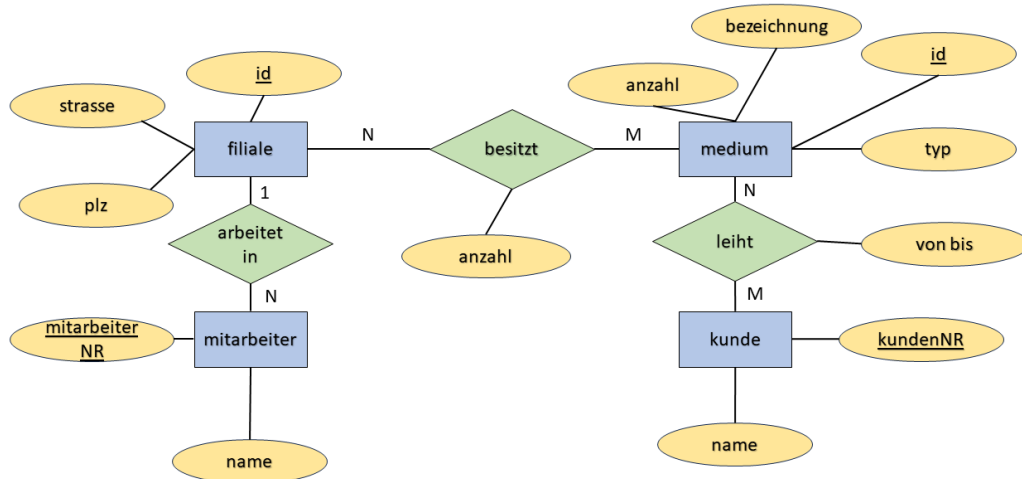
1. Ein Fahrradverleih am Bodensee verleiht Damen-, Herren- und Kinderfahrräder. Dabei wird für jedes Fahrrad ein eigener Mietvertrag abgeschlossen. Eine Person kann mehrere Fahrräder mieten. Der Fahrradverleih möchte eine Datenbank aufbauen. Helfen Sie dabei.



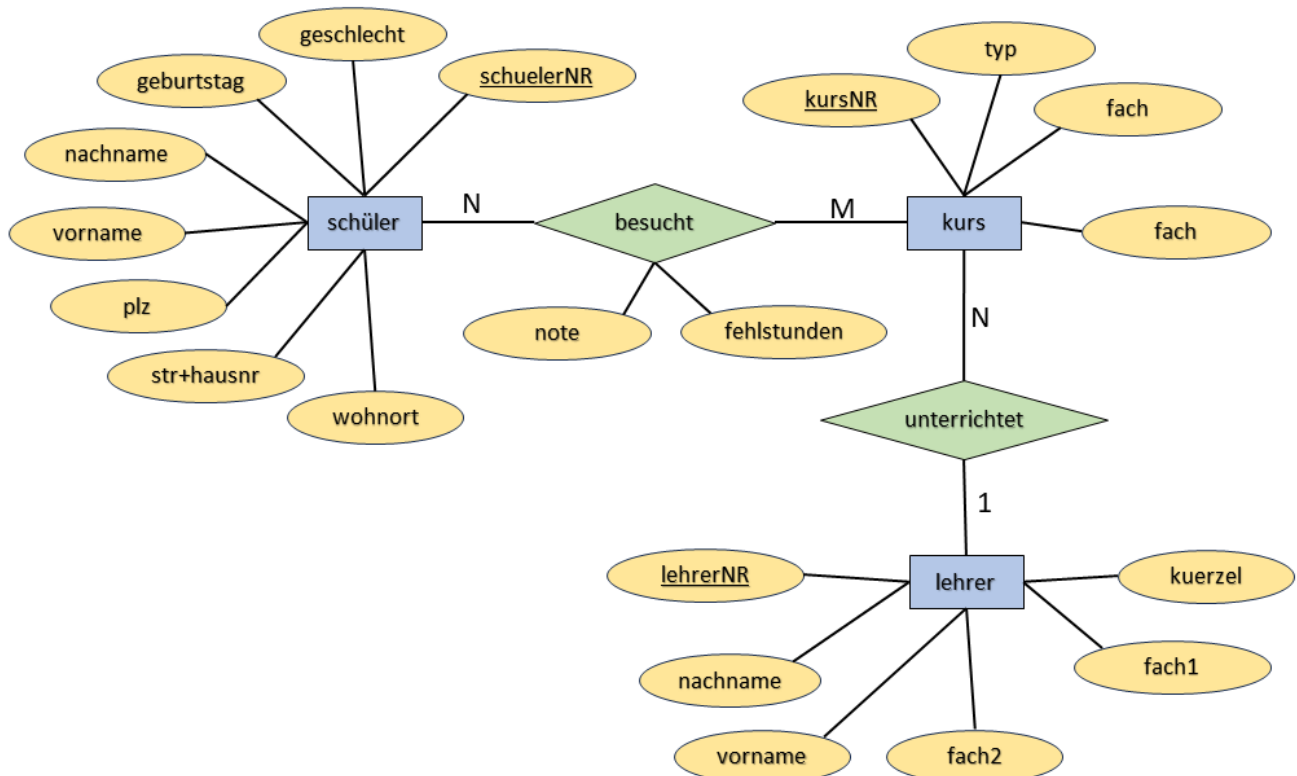
2. Ein befreundeter Autohändler bittet uns beim Aufbau einer Kundendatenbank zu helfen. Zuerst soll diese in einem ERM modelliert werden. Darin erscheinen sollen Kunde, Auto, Karosserietyp und Reifen. Ein Auto gehört dabei zu einem Kunden, ein Kunde kann aber mehrere Autos haben.



3. Ein DVD-Verleiher betreibt mehrere Filialen (id, strasse, plz), wo es jeweils mehrere Medien (DVDs, BluRays, Spiele) zu leihen gibt. Jeder Kunde kann nur einer Filiale zugeordnet sein. Jeder Kunde kann mehrere Medien ausleihen. Ein Mitarbeiter kann nur in einer Filiale arbeiten.



Lösung zu Übung 4 In der Oberstufe eines Gymnasiums wird nicht mehr in Klassen, sondern in Kursen unterrichtet. Sie erhalten von der Schulleitung den Auftrag, eine Kursverwaltung mittels des Entity-Relationship-Modells zu modellieren. Mit Hilfe dieser Kursverwaltung soll festgehalten werden, welche Schüler welche Kurse besuchen. Als Schülerdaten soll neben dem Vornamen und Nachnamen der Schüler auch die individuelle Schülernummer, das Geburtsdatum, Geschlecht sowie die Postadresse festgehalten werden. Jeder Kurs hat eine eigene Kursnummer. Außerdem sind der Kurstyp (5stündig / 2stündig), das Fach (z.B. D, M, E, ...) und die Jahrgangsstufe (K1 / K2) zu speichern. In dem neuen System sollen auch die Fehlstunden und Kursnoten jedes Schülers dokumentiert werden. Jeder Kurs ist einem Lehrer zugeordnet. Als lehrerspezifische Daten sollen dessen Vor- und Nachname, das Kürzel und seine Fächer (max. 2) mit in das Kursverwaltungsprogramm aufgenommen werden.



Lösung zu Übung 5

- Ein Fahrradverleih am Bodensee verleiht Damen-, Herren- und Kinderfahrräder. Dabei wird für jedes Fahrrad ein eigener Mietvertrag abgeschlossen. Eine Person kann mehrere Fahrräder mieten. Der Fahrradverleih möchte eine Datenbank aufbauen. Helfen Sie dabei.

kunde				
<u>kundeNR</u>	name	adresse		
15	Fathi Russdi	Rotgasse 12, 55555 Bielefeld		
fahrrad				
<u>rahmenNR</u>	typ			
18498451	Rennrad			
89415456	Kinderrad			
mietet				
<u>laufendeNR</u>	kundenNR	rahmenNR	beginn	ende
1	15	18498451	01.09.2023	03.09.2023
2	15	89415456	01.09.2023	03.09.2023

- Ein befreundeter Autohändler bittet uns beim Aufbau einer Kundendatenbank zu helfen. Zuerst soll diese in einem ERM modelliert werden. Darin erscheinen sollen Kunde, Auto, Karosserietyp und Reifen. Ein Auto gehört dabei zu einem Kunden, ein Kunde kann aber mehrere Autos haben.

kunde				
<u>kundeNR</u>	name	adresse		
15	Fathi Russdi	Rotgasse 12, 55555 Bielefeld		
auto				
<u>fahrzeugID</u>	karosserie			
189765465451	Kübelwagen			
reifen				
<u>reifenID</u>	typ			
89454115	Winterreifen			
auto_reifen				
<u>laufendeNR</u>	fahrzeugID	reifenID		
4	189765465451	89454115		
auto_kunde				
<u>laufendeNR</u>	fahrzeugID	kundenNR	preis	datum
119	189765465451	15	43000	10.08.2023

- Ein DVD-Verleiher betreibt mehrere Filialen (id, strasse, plz), wo es jeweils mehrere Medien (DVDs, BluRays, Spiele) zu leihen gibt. Jeder Kunde kann nur einer Filiale zugeordnet sein. Jeder Kunde kann mehrere Medien ausleihen. Ein Mitarbeiter kann nur in einer Filiale arbeiten.

filiale				
<u>filialID</u>	strasse	plz		
3	Königsbau 4	70174		
mitarbeiter				
<u>mitarbeiterNR</u>	name			
47	Agent 47			
medium				
<u>medienID</u>	bezeichnung	anzahl	typ	
1002	The matrix	5	DVD	
kunde				
<u>kundenNR</u>	name			
50	Lucky Luke			
filiale_mitarbeiter				
<u>laufendeNR</u>	filialID	mitarbeiterNR		
1	3	47		
filiale_medium				
<u>laufendeNR</u>	filialID	medienID	anzahl	
1	3	1002	2	
kunde_medium				
<u>laufendeNR</u>	kundenNR	medienID	beginn	beginn
99	50	1002	01.01.2023	14.01.2023

Lösung zu Übung 6

schueler			projekte		klasse		
<u>schNR</u>	name	vorname	<u>projektNR</u>	probez	<u>klasseNR</u>	klasse	klassenbez
1	Müller	Marius	1	Homepage	1	BK14	Kaufm. BK1
2	Kryof	Yuri	2	Foyergestaltung	2	BK22	Kaufm. BK2
3	Abadi	Ali	3	Schulfest			
4	Sanbei	Sarah					
klassenzug		projektteilnahme					
<u>schNR</u>	klasseNR	<u>schNR</u>	<u>projektNR</u>	prostd			
1	2	1	1	30			
2	1	2	2	25			
3	1	3	1	10			
4	2	3	2	15			
		4	1	15			
		4	3	35			

Lösung zu Übung 7

auftrag			personal				
<u>aNR</u>	auftrag	baust	<u>pNR</u>	vorname	nachname	plz	kkasse
A1	Garage	Stuttgart	13	Cem	Özdemir	72070	AOKBW
A2	Haus	Esslingen	17	Christian	Lindner	70794	DAK

ort		krankenkasse	
<u>plz</u>	ort	<u>kkasse</u>	kkbeitrag
70794	Filderstadt	AOKBW	16,2
72070	Tübingen	DAK	16,3

Lösung zu Übung 8

kunden		artikel		bestellung		
<u>kundeNR</u>	name	<u>artNR</u>	artBez	<u>kundeNR</u>	<u>artNR</u>	anzahl
5001	Volker Finke e.K.	8001	Schraubendreher 5mm	5001	8001	10
5004	Hubert Hase GmbH	8005	Schraubendreher 8mm	5001	8005	15
5007	Rudi Rüssel KG	8007	Schraubendreher-Set	5001	8007	15
				5004	8001	20
				5004	8006	20
				5004	8007	10
				5007	8007	25

Lösung zu Übung 9

- Fahrradverleih:
Die Beziehungstabelle zur Beziehung **mietet** kann wegoptimiert werden, indem man das Attribut **kundenNR** als Fremdschlüssel zur Tabelle **fahrrad** hinzufügt.
- Autohändler:
Die Beziehungstabelle zur Beziehung **gekauft** kann wegoptimiert werden, indem man das Attribut **kundenNR** als Fremdschlüssel zur Tabelle **auto** hinzufügt.
Die Beziehungstabelle zur Beziehung **verfuegt** kann wegoptimiert werden, indem man das Attribut **fahrzeugID** als Fremdschlüssel zur Tabelle **reifen** hinzufügt.
- DVD-Verleih:
Die Beziehungstabelle zur Beziehung **arbeitet_in** kann wegoptimiert werden, indem man das Attribut **filiale.id** als Fremdschlüssel zur Tabelle **mitarbeiter** hinzufügt.

Lösung zu Übung 10

1. Informiere dich über den NULL-Wert, der oben in der Datenbank vorkommt. Für was steht dieser Wert? Was ist der Unterschied zwischen Null bzw. 0 und NULL?
Der Wert NULL bedeutet, dass kein Wert vorhanden ist. Ein ähnliches Konzept kennen wir aus der Mathematik. Die Gleichung $x^2 = 0$ hat die Lösung $x = 0$, während die Gleichung $x^2 = -1$ keine Lösung hat, was wir durch das Blitzsymbol \nexists anzeigen. Im obigen Beispiel steht ein Wert von 0 für das Alter für eine Person, die ihren ersten Geburtstag

noch nicht hatte. Ein Wert von NULL bedeutet, dass das Alter unbekannt ist.

2. Was ist ein Byte?

Ein Byte ist eine Informationseinheit, die normalerweise aus 8 Bit besteht. Ein Bit kann die beiden Zustände 1 oder 0 annehmen. Ein Byte kann also $2^8 = 256$ verschiedene Zustände annehmen. Ältere Zeichensätze haben jeweils ein Zeichen in ein Byte gespeichert. So konnten also 256 verschiedene Zeichen (z.B. a, b, c, A, B, C, §, +, usw.) unterschieden werden.

3. Wie werden INTEGER auf der Datenbank gespeichert?

Im Alltag verwenden wir das Dezimalsystem, d.h. jede Zahl wird in Form von Potenzen von 10 dargestellt:

$$123 = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 = 100 + 20 + 3$$

INTEGER werden einfach vom Dezimalsystem auf das Binärsystem übertragen:

$$\begin{aligned} 123 &= 1111011_{BIN} = 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 64 + 32 + 16 + 8 + 2 + 1 = 123. \end{aligned}$$

Das erste Bit kann als Vorzeichen verwendet werden. Dann kann man in einem Byte Zahlen von -128 bis 127 speichern. Je mehr Byte man für eine Zahl verwendet, desto mehr Speicherplatz benötigt man. Jedoch lassen sich dann auch größere Zahlen speichern.

Lösung zu Übung 11

1. Erzeuge eine Tabelle `schueler` mit den Attributen `schuelerNR` als Primärschlüssel, der nicht NULL sein darf, `name`, `plz` und `klasse`.

```
CREATE TABLE schueler(schuelerNR INT PRIMARY KEY NOT NULL, name TEXT, plz INT, klasse TEXT);
```

2. Füge 2 verschiedene Schüler hinzu, die aus den Klassen BK13 und BK21 stammen., z.B.:

```
INSERT INTO schueler VALUES (1, 'Heinz Huber', 70180, 'BK13');
INSERT INTO schueler VALUES (2, 'Dasan Ilhan', 70567, 'BK21');
```

3. Was passiert, wenn man einen weiteren Schüler mit einer bereits vergebenen `schuelerNR` hinzufügen will?

Z.B. folgender Befehl:

```
INSERT INTO schueler VALUES (1, 'Alina Lutz', 70874, 'BK21');
```

Es wird ein Fehler ausgegeben:

```
Runtime error: UNIQUE constraint failed: schueler.schuelerNR
```

Unique bedeutet einzigartig und constraint steht für Einschränkung. Der Fehler besagt also, dass beim Hinzufügen eines Schülers in der Tabelle `schueler` der Wert des Attributs `schuelerNR` nicht einzigartig war.

4. Was passiert, wenn man einen weiteren Schüler mit der `schuelerNR` NULL einfügen will?

Z.B. folgender Befehl:

```
INSERT INTO schueler(name) VALUES ('Vanessa Oranbay');
```

.

Dieser Befehl würde gerne eine Zeile in der Tabelle `schueler` anlegen, bei der alle Einträge bis auf den `name` den Wert NULL haben. Es wird ein Fehler ausgegeben:

```
Runtime error: NOT NULL constraint failed: schueler.schuelerNR
```

Der Fehler besagt also, dass beim Hinzufügen eines Schülers in der Tabelle **schueler** der Wert des Attributs **schuelerNR** NULL war, was nicht erlaubt ist.

5. Wie sehen die Ausgaben von `.tables` und `.schema` aus?

```
sqlite> .table
schueler
sqlite> .schema
CREATE TABLE schueler(schuelerNR INT PRIMARY KEY NOT NULL,
name TEXT, plz INT, klasse TEXT);
```

Lösung zu Übung 12

1. Welche Ausgabe erzeugt das Statement `SELECT * FROM schueler;`?

Das Statement gibt alle in der Tabelle vorhandenen Schüler aus:

```
1 1|Anica|Nosudohein|6268|06.11.1998|BKFH
2 2|Marlies|Gavofu|25361|06.01.2002|BK2
3 3|Franz|Rotagateson|71296|13.01.1998|BK1
4 4|Elisabeth|Kotibodoweiner|14798|20.11.2003|BK1
5 5|Henni|Kitavare|22926|21.07.1999|BK2
6 6|Mariana|Hewalode|23879|19.05.2004|BK2
7 7|Henry|Zütuschatthein|94405|31.12.2004|BK1
8 8|Fatma|Varobason|19370|08.01.2005|BK1
9 9|Gundel|Culufledemeiner|97896|12.04.1996|BKFH
10 10|Reinhold|Tulimattson|25821|08.08.1997|BK1
11 11|Silvia|Cüwiwattmüller|88339|09.11.2001|BK2
12 ...
```

Anmerkung: Es wurden aus Platzgründen nicht alle Schüler hier aufgelistet.

2. Wofür steht der Stern (*) in obigem Statement?

Der Stern ist eine sogenannte Wildcard. Das SELECT-Statement muss wissen, welche Attribute angezeigt werden sollen. Der Stern bedeutet, dass die Werte aller an der Tabelle vorhandenen Attribute ausgegeben werden.

3. Welche Ausgabe erzeugt `SELECT vorname, nachname FROM schueler;`?

```
1 Anica|Nosudohein
2 Marlies|Gavofu
3 Franz|Rotagateson
4 Elisabeth|Kotibodoweiner
5 Henni|Kitavare
6 Mariana|Hewalode
7 Henry|Zütuschatthein
8 Fatma|Varobason
9 Gundel|Culufledemeiner
10 Reinhold|Tulimattson
```

```
11      Silvia|Cüwiwattemüller
12      ...
```

Da nun nicht mehr der Stern verwendet wurde, um die Werte aller Attribute anzuzeigen, werden nur die Werte von **vorname** und **nachname** angezeigt.

4. Finde ein Statement, um dir **nachname**, **plz** und **klasse** anzeigen zu lassen.

```
SELECT nachname, plz, klasse FROM schueler;
```

Lösung zu Übung 13

1. den Schüler mit der **schuelerNR** 31 zurück gibt.

```
SELECT * FROM schueler WHERE schuelerNR = 31;
```

2. alle Schüler mit der **schuelerNR** 10, 23, 50 und 65 findet.

```
SELECT * FROM schueler WHERE schuelerNR in (10, 23, 50, 65);
```

3. alle Schüler findet, deren **nachname** auf hein endet.

```
SELECT * FROM schueler WHERE nachname LIKE '%hein';
```

4. alle Schüler findet, deren **nachname** nicht auf hein endet.

```
SELECT * FROM schueler WHERE nachname NOT LIKE '%hein';
```

5. alle Schüler findet, für die keine **klasse** angegeben worden ist.

```
SELECT * FROM schueler WHERE klasse IS NULL;
```

6. alle Schüler findet, deren **schuelerNR** kleiner 18 ist.

```
SELECT * FROM schueler WHERE schuelerNR < 18;
```

7. alle Schüler findet, deren **vorname** aus genau 4 Buchstaben besteht.

```
SELECT * FROM schueler WHERE vorname LIKE '____';
```

8. Bonus-Frage: Warum gibt das Statement

```
SELECT * FROM schueler WHERE geburtsdatum BETWEEN '01.01.2000' AND '31.12.2000';
```

viel zu viele Schüler zurück?

Tipp: Probiere das Statement

```
SELECT * FROM schueler WHERE geburtsdatum BETWEEN '31.01.2000' AND '31.12.2000';
```

oder das Statement `SELECT geburtsdatum FROM schueler ORDER BY geburtsdatum;`

aus.

Das zweite Statement aus dem Tipp zeigt, dass die Geburtsdaten nicht wie erwartet sortiert werden, sondern Zeichen für Zeichen. Da die Punkte bei allen Geburtsdaten an der gleichen Stelle stehen, können wir diese ignorieren. Man kann sich die Geburtsdaten dann einfach als Zahlen vorstellen, z.B. wäre 01.04.1995 die Zahl 1.041.995. Nun werden alle **schueler** mit Geburtsdaten, deren Zahl zw. 1.012.000 und 31.122.000 liegen, zurückgegeben. Z.B. entspricht das Geburtsdatum der Schülerin mit **schuelerNR** 100 10.07.1997 der Zahl 10.071.997. Da diese Zahl zwischen den beiden Grenzen 1.012.000 und 31.122.000 liegt, wird sie *fälschlicherweise* ausgegeben.

Lösung zu Übung 14

1. Den Schüler mit der **schuelerNR** 40.

```
DELETE FROM schueler WHERE schuelerNR = 40;
```

2. Alle Schüler mit der **schuelerNR** 10, 20, 50 und 60.

```
DELETE FROM schueler WHERE schuelerNR IN (10, 20, 50, 60);
```

3. Alle Schüler, deren **vorname** auf ia endet.

```
DELETE FROM schueler WHERE vorname LIKE '%ia';
```

Lösung zu Übung 15

1. Der Schüler mit der **schuelerNR** 19 soll mit **vornamen** Hans Gustav Adalbert heißen.

```
UPDATE schueler SET vorname = 'Hans Gustav Adalbert' WHERE schuelerNR = 19;
```

2. Die Bezeichnung der **klasse** soll von BK2 auf BK22 geändert werden.

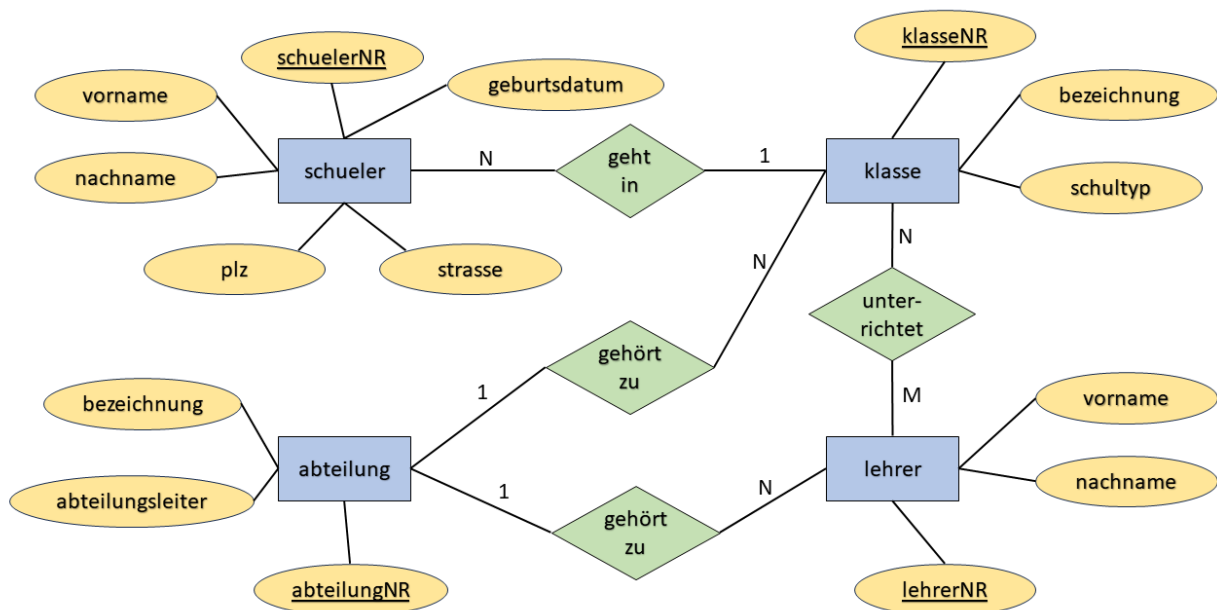
```
UPDATE schueler SET klasse = 'BK22' WHERE klasse IS 'BK2';
```

3. Bei einigen Schülern ist ein Problem beim Eintragen der **plz** aufgetreten. Bei allen Schülern mit einer 4-stelligen **plz** soll diese auf NULL geändert werden.

```
UPDATE schueler SET plz = NULL WHERE plz < 10000;
```

Lösung zu Übung 16

1. Erstelle ein zur Datenbank passendes ERM. Tipp: **.schema** könnte hilfreich sein.



2. Wie viele verschiedene Lehrer unterrichten an der Schule?

```
SELECT COUNT(lehrerNR) FROM lehrer;
```

Es sind 17 Lehrer.

3. Wie viele Schüler kommen aus einer Stadt, deren PLZ mit einer 7 beginnt?

```
SELECT COUNT(schuelerNR) FROM schueler WHERE plz between 70000 AND 79999;
```

Es sind 142 Schüler.

4. Wie viele Lehrer sind den einzelnen Abteilungen jeweils zugeordnet?

```
SELECT abteilungNR, COUNT(lehrerNR) FROM lehrer_abteilung GROUP BY abteilungNR;
```



```
1      3|6
2      6|7
3      8|3
```

```
SELECT abteilungNR, bezeichnung FROM abteilung;
```

```
1      3|WG
2      6|Berufskolleg
3      8|Berufsschule
```

Das bedeutet also, dass im WG 6 Lehrer, im Berufskolleg 7 Lehrer und in der Berufsschule 3 Lehrer unterrichten.

Lösung zu Übung 17

1. Erzeuge eine Ausgabe, die dem Vor- und Nachnamen der Lehrer jeweils die passenden Abteilungsbezeichnungen zuordnet.

```
SELECT lehrer.vorname, lehrer.nachname, abteilung.bezeichnung FROM lehrer INNER JOIN abteilung ON lehrer.abteilungNR = abteilung.abteilungNR;
```

2. Erzeuge eine Ausgabe, die dem Vor- und Nachnamen aller Schüler jeweils die Bezeichnung der passenden Klasse zuordnet.

```
SELECT schueler.vorname, schueler.nachname, klasse.bezeichnung FROM schueler INNER JOIN klasse ON schueler.klasseNR = klasse.klasseNR;
```

3. Erzeuge eine Ausgabe, die jeder Klassenbezeichnung die Anzahl der Schüler der Klasse zuordnet. Tipp: COUNT-Funktion verwenden.

```
SELECT klasse.bezeichnung, COUNT(klasse.bezeichnung) FROM schueler INNER JOIN klasse ON schueler.klasseNR = klasse.klasseNR GROUP BY klasse.bezeichnung;
```

4. Erzeuge eine Ausgabe, die dem Vor- und Nachnamen aller Schüler jeweils den passenden Schultyp zuordnet. Tipp: Man muss zwei JOIN-Statements verwenden.

```
SELECT schueler.vorname, schueler.nachname, abteilung.bezeichnung FROM schueler INNER JOIN klasse ON schueler.klasseNR = klasse.klasseNR INNER JOIN abteilung ON klasse.abteilungNR = abteilung.abteilungNR;
```