

## Inhalt Datenbanken programmieren

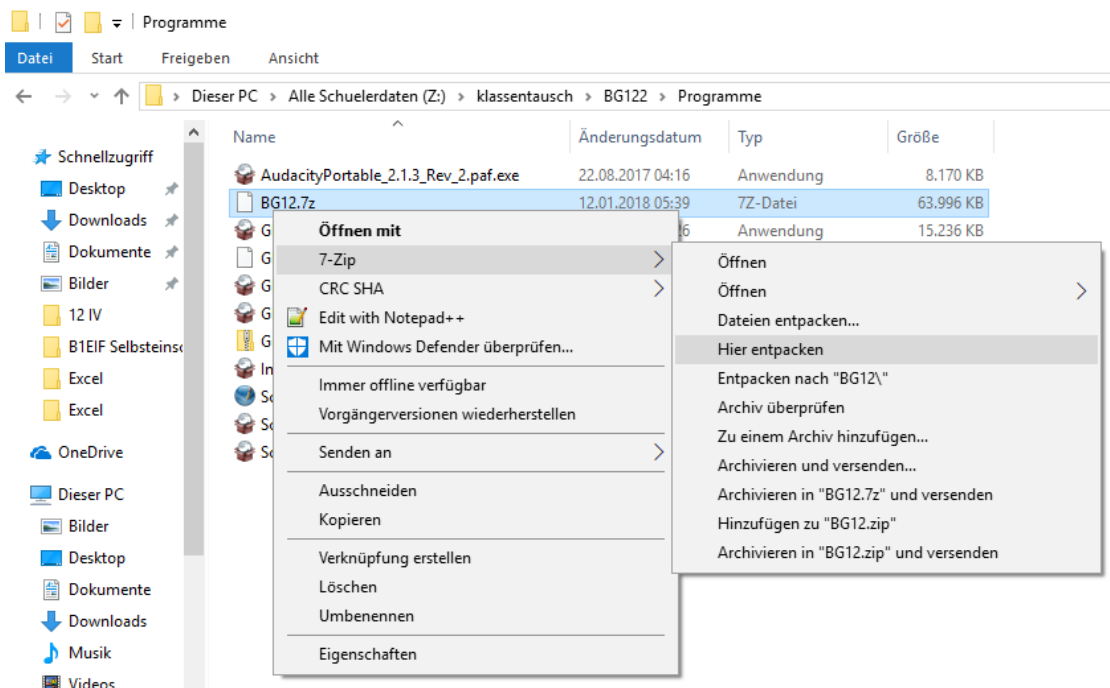
Einrichtung der Datenbank-Software .....	2
Anleitung XAMPP-Installation .....	2
SQL-Programm.....	4
Datenbanktheorie .....	6
SQL-Sprachelemente .....	6
Informationsblatt Datenbank erstellen .....	7
1. Datenbank erstellen .....	7
2. Tabellen erstellen .....	8
3. Einfügen von Datensätzen.....	9
4. Abfrage der Tabellendaten .....	10
5. Besondere Abfragen der Tabellendaten .....	10
6. Tabellenänderung, Datenmanipulation .....	12
7. Abfragen über mehrere Tabellen.....	13
8. Gruppierungen und Bedingungen.....	14
9. Hinweise für die BG13 (Abitur 2019) .....	15

## Einrichtung der Datenbank-Software

### Anleitung XAMPP-Installation

Die Erstellung einer Datenbank benötigt einen MySQL-Server. Mit dem Programm „XAMPP“ ist es möglich, einen MySQL-Server zu nutzen, damit Datenbanken programmiert werden können.

Im Klassentausch (BG12X/Programm) ist ein Archiv hinterlegt, welches XAMPP beinhaltet. Das Archiv „BG12.7z“ soll auf das Laufwerk „C:“ gemäß nachfolgendem Bild entpackt werden.



In dem entpackten XAMPP-Ordner befindet sich die Datei „setup\_xampp.bat“, die ausgeführt werden soll. Beim geöffneten Fenster (nachfolgendes Bild), soll die Taste „1“ und nach dem Durchlauf „ENTER“ gedrückt werden, damit das Fenster geschlossen wird.

```

C:\Windows\system32\cmd.exe
##### START XAMPP TEST SECTION #####
[XAMPP]: Test php.exe with php\php.exe -n -d output_buffering=0 --version ...
PHP 7.1.8 (cli) (built: Aug 1 2017 21:10:46) ( ZTS MSVC14 (Visual C++ 2015) x86 )
Copyright (c) 1997-2017 The PHP Group
Zend Engine v3.1.0, Copyright (c) 1998-2017 Zend Technologies
[XAMPP]: Test for the php.exe successfully passed. Good!
##### END XAMPP TEST SECTION #####

#####
# ApacheFriends XAMPP setup win32 Version
#-----#
# Copyright (c) 2002-2018 ApacheFriends 7.1.8
#
#-----#
# Authors: Kay Vogelgesang <kvo@apacheFriends.org>
#          Carsten Wiedmann <webmaster@wiedmann-online.de>
#####

Do you want to refresh the XAMPP installation?
Soll die XAMPP Installation jetzt aktualisiert werden?

1) Refresh now! (Jetzt aktualisieren!)
x) Exit (Beenden)
  
```

```

C:\Windows\system32\cmd.exe
# ApacheFriends XAMPP setup win32 Version
#-----#
# Copyright (c) 2002-2018 ApacheFriends 7.1.8
#-----#
# Authors: Kay Vogelgesang <kvo@apachefriends.org>
#          Carsten Wiedmann <webmaster@wiedmann-online.de>
#####

Do you want to refresh the XAMPP installation?
Soll die XAMPP Installation jetzt aktualisiert werden?

1) Refresh now! (Jetzt aktualisieren!)
x) Exit (Beenden)
1

XAMPP is refreshing now ...
XAMPP wird nun aktualisiert ...

Refreshing all paths in config files ...

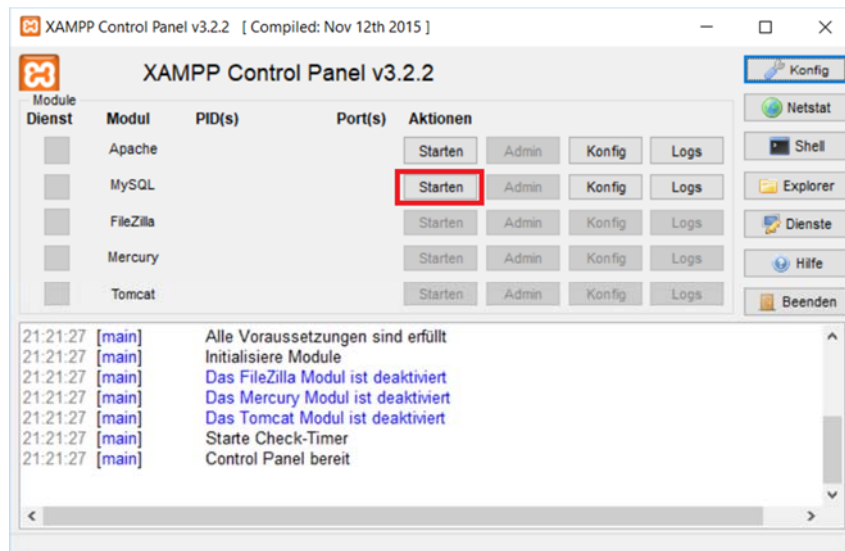
Configure XAMPP with awk for 'Windows_NT'
Updating configuration files ... please wait ... DONE!

##### Have fun with ApacheFriends XAMPP! #####

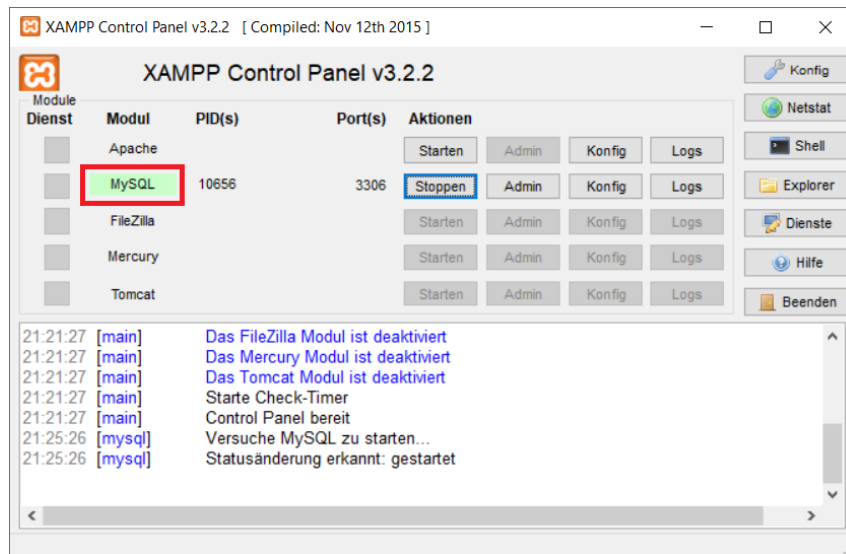
Drücken Sie eine beliebige Taste . . .

```

Im Ordner XAMPP-Ordner befindet sich die Datei „xampp-control.exe“, die zum Starten eines MySQL-Servers als auch die Programmierung von SQL-Datenbanken notwendig ist.



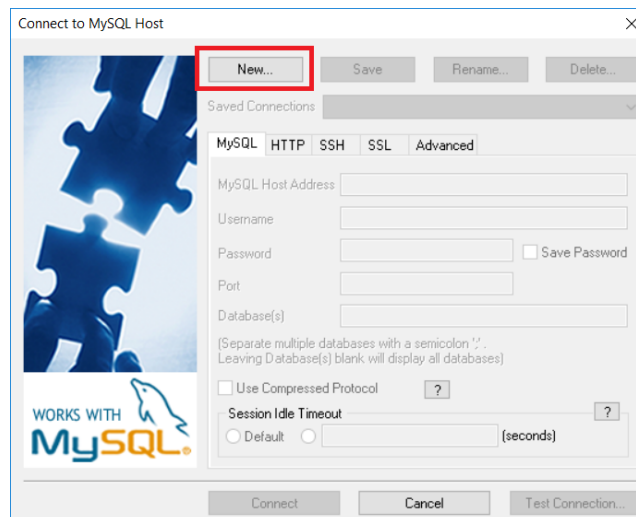
Nun auf den zweiten „Starten“-Button von oben für MySQL drücken. Leuchtet „MySQL“ anschließend grün (siehe nachfolgendes Bild), ist der MySQL-Server online. Das jeweilige SQL-Programmierungstool kann gestartet werden.



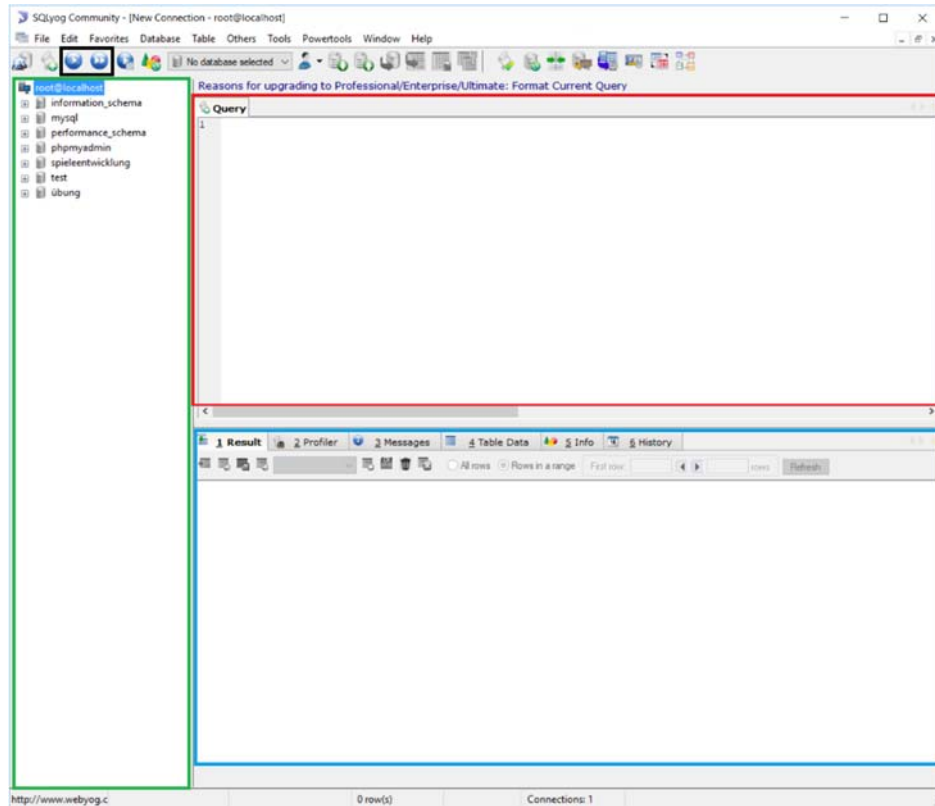
**Hinweis:** Anschließend XAMPP minimieren oder offenlassen. Auch wenn oben rechts „schließen“ gedrückt wird, bleibt XAMPP weiterhin im Hintergrund (System Tray) offen. Soll XAMPP beendet werden, so muss vorher der MySQL-Server gestoppt werden und dann der Button „Beenden“ gedrückt werden. Ansonsten werden mehrere XAMPP-Instanzen geöffnet und ausgeführt.

## SQL-Programm

Nach dem Öffnen der Datei „SQLyogCommunity.exe“ muss der „Continue...“-Button gedrückt werden. Nun auf den „New“-Button drücken, Namen übernehmen und dann auf „Connect“.



## SQLyog-Interface



**Schwarzer Kasten:** Ausführen einzelner SQL-Befehle (F9), Doppelpfeile für das Ausführen aller SQL-Befehle auf einmal. Empfehlenswert ist bei der Fehlersuche die Ausführung einzelner Befehle, damit die Fehlerzuordnung einfacher ist.

**Roter Kasten:** Fenster für das Eintippen des SQL-Programmcodes. Kommentare werden mit einem „/\*“ eingeleitet und mit „\*/“ beendet (ohne Anführungszeichen). Ist für die Strukturierung des SQL-Codes notwendig.

**Blauer Kasten:** Fenster für die Ausgabe der Abfragen, Rückmeldung über erfolgreiche Ausführung der Befehle, Ausgabe der Fehlermeldung

**Grüner Kasten:** Struktur der Datenbanken; Nach der Datenbankerstellung dieses Fenster aktualisieren (F5), damit eine Übersicht der Tabellen und Spalten angezeigt wird.

## **Datenbanktheorie**

### **SQL-Sprachelemente**

#### Data Definition Language (DDL)

Darunter fallen die Befehle zur Definition von Datenbanken und Tabellen als auch anderen Datenstrukturen.

#### Data Control Language (DCL)

Hierzu gehören die Befehle, die zur Kontrolle der Zugriffsberechtigungen gehören. Den Benutzern können Abfragerechte, Aktualisierungsrechte, .... Gewährt als auch wieder entzogen werden. Zum Beispiel ergibt es Sinn, den Sekretärinnen Aktualisierungsrechte zu geben, reinen Fachlehrern sollten diese Rechte nicht haben.

#### Data Manipulation Language (DML)

Dazu gehören, die zur Manipulation und Abfrage von Daten verwendet werden. Manipulation bedeutet, dass gelöscht, geändert oder eingefügt werden. Hierbei geht es um die Inhalte von Datenbanken.

Nachfolgend steht eine Übersicht zu den jeweiligen Befehlen. In grauer Farbe sind die Befehle, die nicht angesprochen werden.

<b>DDL</b>	<b>DCL</b>	<b>DML</b>
CREATE DATABASE Datenbankerstellung	GRANT Zugriffsrechte gewähren	SELECT Tabellenabfrage
CREATE TABLE Tabellenerstellung	REVOKE Zugriffsrechte entziehen	DELETE Datensatzlöschung
ALTER TABLE Tabellenänderung		INSERT Datensatzeingfügung
DROP TABLE Tabellenlöschung		UPDATE Datensatzänderung
RENAME Tabellenumbenennung		

## Informationsblatt Datenbank erstellen

### Arbeitsauftrag:

1. Tippen Sie den Programmcode von der ersten Seite ab.
2. Führen Sie einzeln die SQL-Befehle aus.
3. Lesen Sie sich die einzelnen vier Schritte durch.

### Hinweise:

- Jeder SQL-Befehl wird mit einem Semikolon geschlossen.
- Teste Sie schrittweise jeden Befehl. Das bedeutet, jedes Semikolon nach der Erstellung ausführen. Dies erleichtert die Fehlersuche.
- Das mehrmalige Ausführen von SQL-Befehlen kann zu Fehlermeldungen führen, z. B. weil eine Datenbank mit dem gleichen Namen bereits verfügbar ist.

### SQL-Programmcode:

```
ers.sql*
1  /*Datenbankerstellung*/
2  DROP DATABASE ers;
3  CREATE DATABASE ers;
4  USE ers;
5  DROP TABLE Schueler;
6  /*Tabellenerstellung*/
7  CREATE TABLE Schueler (
8      s_nr INT(4) NOT NULL AUTO_INCREMENT,
9      s_nachname VARCHAR(40),
10     s_vorname VARCHAR(40),
11     s_str VARCHAR(60),
12     s_plz VARCHAR(5),
13     s_ort VARCHAR(40),
14     s_tel VARCHAR(15),
15     PRIMARY KEY(s_nr)
16 ) ENGINE = INNODB;
17 /*Datensatz einfügen*/
18 INSERT INTO Schueler (s_nr, s_nachname, s_vorname, s_str, s_plz, s_ort, s_tel)
19     VALUES (NULL, 'Schmidt', 'Wendelin', 'Bahnhofstraße 2', '31785', 'Hameln', '051519999999999');
20 /*Datensatz abfragen*/
21 SELECT * FROM Schueler;
22 SELECT s_nr, s_nachname, s_vorname FROM Schueler;
```

### 1. Datenbank erstellen

Wir legen eine Datenbank für die Schule ERS an. Der Befehl zur Erstellung einer Datenbank lautet:

Syntax: **create database** *datenbankname*;

Beispiel: *create database ers*;

Wir wählen nun die erstellte Datenbank aus.

Syntax: **use** *datenbankname*;

Beispiel: *use ers*;

Wir löschen Datenbanken mit dem Befehl *drop*.

Syntax: **drop database** *datenbankname*;

Beispiel: *drop database ers*;

## 2. Tabellen erstellen

Wir legen eine Schülertabelle innerhalb der Datenbank ERS an.

Syntax:	<pre>create table <i>tabellenname</i> (     <i>spaltenname1</i> <i>datentyp</i>(<i>stellenanzahl</i>) not null auto_increment,     <i>spaltenname2</i> <i>datentyp</i>(<i>stellenanzahl</i>),     ...     <i>spaltennameX</i> <i>datentyp</i>(<i>stellenanzahl</i>),     primary key(<i>spalte</i>) ) engine = innodb;</pre>
Beispiel:	<pre>create table <i>Schueler</i> (     <i>s_nr</i> <i>int</i>(4) not null auto_increment,     <i>s_nachname</i> <i>varchar</i>(40),     ...,     <i>s_tel</i> <i>varchar</i>(40),     primary key(<i>s_nr</i>) ) engine = innodb;</pre>

### Informationen zu Datentypen (Besonderheiten in Kapitel 3 beachten!):

- Hinter den Datentypen steht in runden Klammern die maximale Anzahl der Stellen des Datentyps. *int*(3) sagt aus, dass eine Spalte höchstens bis zu drei Stellen haben kann.
- varchar**: Wird verwendet, wenn der Spalteninhalt alphanummerisch ist (Buchstaben oder Kombination aus Buchstaben und Zahlen)  
 Syntax: *spaltenname varchar*(*stellenanzahl*)  
 Beispiel: *s\_plz varchar*(5)
- int**: Wird verwendet, wenn der Spalteninhalt aus absoluten Zahlen besteht.  
 Syntax: *spaltenname int*(*stellenanzahl*)  
 Beispiel: *s\_alter int*(3)
- dec**: Wird verwendet, wenn der Spalteninhalt aus Kommazahlen besteht.  
 Syntax: *spaltenname dec*(*Gesamtlänge Kommazahl*,*Nachkommastellen*)  
 Beispiel: *s\_entfernung dec*(4,2) entspricht dem maximalen Wert 99,99
- date**: Wird für das Datum verwendet.  
 Syntax: *spaltenname date*  
 Beispiel: *s\_gebdatum date*
- year**: Wird für das Jahr verwendet.  
 Syntax: *spaltenname year*  
 Beispiel: *s\_gebdatum year*
- time**: Wird für die Zeitangaben verwendet.  
 Syntax: *spaltenname time*  
 Beispiel: *s\_zeit time*



**Information zum Primärschlüssel:**

- Datensätze sind einzigartig. Das heißt es gibt einen Datensatz in Datenbanken nur ein einziges Mal.
- Das Beispiel bei einer Schülertabelle ist die Schülernummer (abgekürzt *s\_nr*).
- Die Spalte *s\_nr* wird zum Primärschlüssel durch den Befehl *primary key*.
- Der Befehl *not null* gibt an, dass dieser Spalteninhalt nicht NULL sein darf, also er immer ausgefüllt werden muss. Deshalb eignet er sich für den Primärschlüssel.
- Der Befehl *auto\_increment* bedeutet, dass die Spalte automatisch hochgezählt wird, wenn ein neuer Schüler (Datensatz) hinzugefügt wird.

Beispiel:        1. Datensatz: *s\_nr=1*  
                  2. Datensatz: *s\_nr=2...*

- *engine = innodb* muss nach der letzten Klammer und vor dem Semikolon stehen. Es dient zur Interpretation des späteren Fremdschlüssels.

**Hinweise:**

- Der erste Buchstabe vor dem Spaltennamen trägt immer den ersten Buchstaben des Tabellennamens.
- Der Datentyp *int* lässt keine 0 an erster Stelle einer Zahl darstellen. Deswegen wird bei Postleitzahlen der Datentyp *varchar* verwendet. Beispiel: *PLZ* ist *09482*.
- Wichtig sind die runden Klammern nach dem Tabellennamen und vor *engine = innodb*; Ansonsten gibt es eine Fehlermeldung.
- Werden in einer Tabelle mehrere Spalten definiert, so werden diese mit einem Komma getrennt. Die letzte Spaltendefinition enthält kein Komma.
- Einrücken der nachfolgenden Zeilen nach *create table*, damit der Programmcode übersichtlicher wird.

**3. Einfügen von Datensätzen**

Wir füllen die Mitarbeitertabelle mit einem Datensatz. Ein Mitarbeiter entspricht einem Datensatz. Ein anderer Mitarbeiter ist ein anderer Datensatz.

Syntax:        ***insert into tabellenname (spalte1,spalte2,spalte3)***  
                  ***values(wert1,wert2,wert3);***

Beim Datentyp *varchar* werden die Werte in Hochkommata gesetzt. Beim Datentyp *int*, *dec* und *date* werden keine Hochkommata gesetzt.

Beispiel:        ***insert into mitarbeiter (s\_nr,s\_name,s\_vorname)***  
                  ***values(NULL,'Schmidt','Wendelin');***

Die Angabe der Spalten nach dem Tabellennamen kann entfallen, wenn die Spaltenanzahl = Anzahl der Values (Werte) entspricht.

Beispiel:        ***insert into mitarbeiter***  
                  ***values(NULL,'Schmidt','Wendelin');***

Werden mehrere Datensätze auf einmal eingefügt, dann entfällt *values* beim zweiten Datensatz. Die Datensätze werden mit einem Komma getrennt.

Beispiel: ***insert into*** *mitarbeiter*  
***values***(NULL, 'Schmidt', 'Wendelin'),  
(NULL, 'Ballauf', 'Magda');

- Der Datentyp *date* hat die Besonderheit, dass das Format 'JJJJ-MM-TT' (Jahr-Monat-Tag) lautet. Also lautet dies in der *value*-Zeile beispielsweise '2017-12-31'.
- Der Datentyp *time* hat die Besonderheit, dass das Format 'HH:MM:SS' (Stunden-Minuten-Sekunden) lautet. Also lautet dies in der *value*-Zeile beispielsweise '00:10:30' für 10 Minuten und 30 Sekunden.

**Hinweis:** Wichtig ist die Reihenfolge beim Einfügen in die Spalten, ansonsten werden die Daten in die falschen Spalten eingepflegt.

#### 4. Abfrage der Tabellendaten

Wir fragen die Schülerdaten der Schülertabelle mit dem *select*-Befehl ab. Mit einem Sternchen nach dem *select*-Befehl werden alle Spalten einer Tabelle abgefragt.

Syntax: ***select \* from*** *tabellenname*;

Beispiel: *select \* from schueler*;

Sind bestimmte Spalten gefragt, werden die Spaltennamen der Tabellen angegeben. Mehrere Spaltennamen werden mit einem Komma getrennt.

Syntax 2: *select spalte1, spalte 2 from Tabellenname*;

Beispiel 2: *select s\_nr, s\_name from schueler*;

#### 5. Besondere Abfragen der Tabellendaten

Abfragen können mit Bedingungen erstellt werden, wenn nach bestimmten Daten gesucht wird. Dies erfolgt mit dem Zusatz *where* bei dem *select*-Befehl.

Syntax: *select \* from Tabellenname where Spaltenname=suchwort/zahl*;

Beispielaufgabe: Es soll eine *select*-Abfrage erstellt werden, bei der alle Schüler mit dem Vornamen „Wendelin“ ausgegeben werden.

Beispiel: *select \* from schueler where s\_vorname='wendelin'*;

**Hinweise:**

- Beim Datentyp *varchar* und *date* werden Hochkommata verwendet
- Beim Datentyp *int* und *dec* werden keine Hochkommata verwendet

Das Gleichzeichen nach dem Spaltennamen kann durch andere mathematische Operatoren ersetzt werden, wie z. B. >, <, >=, <=

Beispiel: *select \* from schueler where s\_alter >= 18*;  
*select \* from schueler where s\_alter <= 18*;

### Anpassung der Suchbedingung

Ähnlich wie bei der Suche in jedem Dateisystem können bestimmte Zeichen einzelne oder mehrere Buchstaben bei der *WHERE*-Bedingung ersetzen. Notwendig ist hierfür der Befehl *like*. Ein Unterstrich steht genau für ein beliebiges Zeichen. Das bedeutet gleichzeitig, wenn z. B. drei Unterstriche verwendet werden, werden bei der Suche drei Buchstaben bzw. Zahlen ersetzt. Hiermit trifft die Bedingung für den Altersbereich von 10 bis 19 Jahren zu.

Beispiel: `select * from schueler where s_alter like 1_;`

Eine weitere Möglichkeit ist die Ersetzung von alphanumerischen Zeichen durch das Prozentzeichen. Das Prozentzeichen wird für eine beliebige Anzahl von Zeichen verwendet. Hiermit werden alle Vornamen, die mit dem Buchstaben „a“ enden, abgefragt. Es spielt keine Rolle, wie viele Zeichen vor dem Buchstaben „a“ sind.

Beispiel: `select * from schueler where s_vorname like 'a%';`

Es ist zudem möglich, mehrere Unterstriche und Prozentzeichen miteinander zu kombinieren. Eine Begrenzung gibt es hierbei nicht.

### Verknüpfungen von Bedingungen

Mehrere Bedingungen bei der Erstellung von *select*-Abfragen sind möglich. Dies erfolgt mit dem *and* (UND-Verknüpfung) zwischen zwei Bedingungen. Hierfür müssen beide Bedingungen zutreffen, damit die Abfrage durchgeführt wird.

Beispiel: `select * from schueler where s_vorname='wendelin' and s_alter>=18;`

Zudem ist es möglich, eine *or* (ODER-Verknüpfung) zu verwenden. Hierfür reicht lediglich das Zutreffen mindestens einer der Bedingung, damit die Abfrage durchgeführt wird.

Beispiel: `select * from schueler where s_vorname='wendelin' or s_alter>=18;`

### Vermeidung von Dopplungen

Es können auch Mehrfachnennungen verhindert werden, indem der Zusatz *distinct* verwendet wird. Durch die nachfolgende Abfrage werden nach unterschiedlichen Nachnamen der Schülerinnen und Schüler abgefragt.

Beispiel: `select distinct s_name from schueler;`

### Aggregatsfunktion

Bei Abfragen besteht auch die Möglichkeit durch Aggregatsfunktionen, Durchschnittswerte oder Summen einer bestimmten Spalte zu berechnen. Nachfolgend wird zuerst das Durchschnittsalter der Schülerinnen und Schüler abgefragt und im zweiten Befehl die Summe des bisher bezahlten Kopiergeldes.

Beispiel: `select avg(s_alter) from schueler;`

Beispiel: `select sum(s_kopiergeld) from schueler;`

Weitere Aggregatsfunktionen, wie Minimum, Maximum und Count, sind von der Syntax ähnlich AVG und SUM.

Beispiel: *select **max(spalte)** from schueler;*  
*select **min(spalte)** from schueler;*  
*select **count(spalte)** from schueler;*

### Sortierung

Zudem ist es möglich, eine *or* (ODER-Verknüpfung) zu verwenden. Hierfür reicht lediglich das Zutreffen mindestens einer der Bedingung, damit die Abfrage durchgeführt wird.

Beispiel: *select \* from schueler where s\_vorname='wendelin' **or** s\_alter>=18;*

Abfragen können nach einer bestimmten Spalte sortiert werden.

Beispiel: *select \* from schueler **order by** spalte;*  
*select \* from schueler order by s\_alter;*

Soll die Sortierung zusätzlich absteigend sein, wird ein DESC am Ende des SQL-Befehls hinzugefügt.

Beispiel: *select \* from schueler order by spalte **desc**;*  
*select \* from schueler order by s\_alter desc;*

## 6. Tabellenänderung, Datenmanipulation

Es ist bei Tabellen möglich, nachträglich die Spalten zu editieren, sowohl eine Umbenennung als auch eine Änderung des Datentyps.

Beispiel: ***alter table** tabellenname*  
***change** spaltenname\_alt spaltenname\_neu datentyp;*  
*alter table schueler*  
*change s\_kopiergeld\_dm s\_kopiergeld\_eur dec(4,2);*

Es ist zudem bei Tabellen möglich, nachträglich bestimmte Spalten zu löschen.

Beispiel: ***alter table** tabellenname*  
***drop** spaltenname[, spaltenname2];*  
*alter table schueler*  
*drop s\_email\_privat, s\_kfz;*

Auch das Hinzufügen von Spalten in Tabellen ist möglich.

Beispiel: ***alter table** tabellenname*  
***add** spaltenname datentyp;*  
*alter table schueler add s\_staatsangehoerigkeit varchar(30);*

Nachträglich können auch bestimmte Datensätze innerhalb einer Tabelle gelöscht werden. Geschickter ist es, bei der Bedingung den Primärschlüssel zu verwenden, weil hiermit Verwechslungen (zwei Schüler, die Hans Meier heißen) ausgeschlossen werden. Hierzu wird vorher eine SELECT-Abfrage zur Ermittlung der richtigen s\_nr verwendet werden.

Beispiel: **delete from** tabellenname **where** spaltenname=Bedingung;  
**select** s\_nr, s\_vorname, s\_name **from** schueler **where**  
s\_vorname='Hans' and s\_name='Meier';  
**delete from** schueler **where** s\_nr=100;

Datensätze können aktualisiert werden, z. B. bei einer neuen Telefonnummer. Sollen alle Datensätze aktualisiert werden, wird die where-Bedingung weggelassen.

Beispiel: **update** tabellenname **set** spaltenname=Neuer Wert[, spaltenname2  
neuer Wert] **where** spaltenname=Bedingung;  
**update** schueler **set** s\_tel='05151123456' **where** s\_nr=10;

**Hinweis** zum UPDATE-Befehl: Bei der Verwendung ist Vorsicht geboten! Wird die Bedingung falsch formuliert und trifft somit auf ungewollte Datensätze zu, werden diese auch aktualisiert. Einen Befehl wie „rückgängig machen“ gibt es hierfür nicht. Somit wäre eine Änderungsanomalie entstanden.

Zusätzlich ermöglicht **update**, dass damit mathematische Aktualisierungen/Rechnungen erledigt werden. Beispielsweise wird ein bestehender Preis um 20 % erhöht oder eine bestehende Menge um 50 Einheiten erhöht. Wichtig ist, dass bei Dezimalzahlen aufgrund der amerikanischen Schreibweise ein Punkt verwendet wird.

Beispiel: **update** schueler **set** s\_kopiergeld=s\_kopiergeld\*1.2;  
**update** schueler **set** s\_fehltage=s\_fehltage+50 **where** s\_nr=20;

## 7. Abfragen über mehrere Tabellen

Abfragen sind über mehrere Tabellen möglich. Es gibt zwei Möglichkeiten, eine technisch bessere und eine technische schlechtere Möglichkeit. Für eine Abfrage über mehreren Tabellen ist eine Verknüpfung notwendig. Die Verknüpfung erfolgt über den Primärschlüssel der ersten Tabelle und dem Fremdschlüssel der zweiten (oder weiteren) Tabellen.

### 1. Möglichkeit: Verknüpfung über WHERE (technisch schlechtere Möglichkeit)

Die Verknüpfung über die Dozentennummer. Die Dozentennummer wurde in der Tabelle „Kurse“ als Fremdschlüssel aufgenommen. Wichtig ist, dass nach FROM alle Tabellen angegeben werden und mit einem Komma getrennt sind (ähnlich wie bei der Spaltenangabe). Zudem müssen bei der Dozentennummer die Tabellennamen angegeben werden, da sonst die Fehlermeldung „ambiguous“ erscheint. Dies steht für eine

zweideutige Angabe der Spalte, weil „dnr“ in den beiden Tabellen „Dozenten“ und „Kurse“ vorkommt.

Beispiel: *select spalte1, spalte2 from tabelle1, tabelle2*  
***where tabelle1.pk=tabelle2.fk;***  
*select kname,dname from kurse,dozenten*  
*where kurse.d\_nr=dozenten.d\_nr;*

## 2. Möglichkeit: Verknüpfung über INNER JOIN (technisch bessere Möglichkeit)

Die Syntax bei INNER JOIN ist im Vergleich zu der Verknüpfung über WHERE anders aufgebaut. Im Beispiel ist zu sehen, dass die zweite erst nach dem INNER JOIN angegeben wird. Anschließend erfolgt der ON, bei dem der Vergleich ähnlich wie bei der 1. Möglichkeit steht.

Beispiel: *select spalte1, spalte2 from tabelle1*  
***inner join tabelle2 on tabelle1.pk=tabelle2.fk***  
***inner join tabelle3 on tabelleX.pk=tabelle3.fk;***  
*select kname, dname from kurse*  
***inner join dozenten on kurse.d\_nr=dozenten.d\_nr;***

INNER JOIN mit einer Bedingung:

Die Bedingung zur Einschränkung der Abfrage steht nach dem ON-Vergleich.

Beispiel: *select spalte1, spalte2 from tabelle1*  
***inner join tabelle2 on tabelle1.pk=tabelle2.fk **where** ...;***  
*select kname, dname from kurse*  
***inner join dozenten on kurse.d\_nr=dozenten.d\_nr where ...;***

## 8. Gruppierungen und Bedingungen

Abfragen können zusätzlich gruppiert werden, wenn z. B. öfters der Ort „Hameln“ oder „Bad Münder“ vorkommt. Zusätzlich wurden beim konkreten Beispiel die Gruppierung nach dem Buchstaben sortiert.

Beispiel: *select spaltennamen from tabellennamen [where=Bedingung] **group by spaltennamen;***  
*select s\_ort from schueler group by s\_ort order by s\_ort;*

**Hinweis:** Möchte zusätzlich die Gruppierung mit einer Bedingung eingeschränkt werden, muss **having** verwendet werden! Nur having wirkt auf **group by**.

Das bedeutet: select-Abfrage → Einschränkung der select-Abfrage durch where-Bedingung  
→ Erstellung Gruppierung durch group by → Einschränkung von group by durch having

Beispiel: *select spaltennamne from tabellennamen where=Bedingung group by*  
*spaltennamen **having=Bedingung;***

```
select s_ort from schueler where s_nr>10 group by s_ort having  
count(s_nr)>5 order by s_ort;
```

Hier werden bei der ersten Einschränkung nur die Schülernummern größer 10 betrachtet, nach der Gruppierung der Orte, werden durch having lediglich die Orte ausgegeben, deren Anzahl höher als 5 ist.

## 9. Hinweise für die BG13 (Abitur 2019)

Innerhalb des Skriptes gibt es einige Änderungen:

1. PRIMARY KEY hat sich bei der Tabellendefinition leicht geändert (siehe Kapitel 2)
2. VARCHAR hat CHAR bei alphanummerischen Spaltendefinitionen ersetzt (siehe Kapitel 2)
3. DEC (dezimal) hat FLOAT bei der Spaltendefinition von Kommazahlen ersetzt (siehe Kapitel 2)
4. Bei der Verknüpfung von Bedingungen wurde OR (ODER-Verknüpfung) hinzugefügt (siehe Kapitel 5)
5. Die weitere Suchanpassung bei der WHERE-Bedingung wurde erweitert (siehe Kapitel 5)

## Normalisierung

Normalisierung bei Datenbanken ist ein zentraler Punkt von Datenbanken, den wir parallel in ERD und bei der Programmierung der Datenbanken gemacht haben. Das ERM und die Normalisierung erfüllen den gleichen Zweck, Daten für eine relationale Datenbank vorzubereiten. Zusätzlich verhindert die Normalisierung Widersprüche und führt zu einer besseren Strukturierung von Tabellen. Die Widersprüche werden in der Fachsprache „**Anomalien**“ genannt, die in drei Arten unterschieden werden: Bei **Änderungen**, **Einfügungen** und **Löschungen** innerhalb der Datenbank entstehen diese.

**Redundanzen** entstehen dadurch, dass beispielsweise derselbe Schüler „Hans Meyer“ zweimal in einer Schülertabelle vorkommt. Tritt nun z. B. eine Adressänderung ein, so müssen beide Datensätze angepasst werden. Somit ergibt die Verwendung von Fremdschlüsseln bei Datenbanken einen Sinn, weil die Datensätze deswegen nur noch einmal hinterlegt werden. Eine Zuordnung erfolgt dann nur noch über die Primär- bzw. Fremdschlüssel.

Die schrittweise Optimierung von Tabellen nennt sich Normalisierung, die bis zur dritten Normalform vollzogen wird. Es ist zu beachten, dass je nach Quelle oder lehrenden Personen die Interpretation der Normalform nicht einheitlich ist. So verschieben sich gelegentlich Regeln der zweiten Normalform auch in die erste Normalform. Die dritte Normalform macht die Datenbank nutzbar, ohne dass Redundanzen auftreten. Es gibt höhere Normalformen, diese sind für diesen Unterricht nicht relevant.

### Erste Normalform:

Jedes Attribut der Relation muss einen atomaren (einzeln) Wertebereich haben. Wir kennen dies aus dem Unterricht mit der Aufteilung der Adresse. Hier in diesem Beispiel enthalten die Spalten „Album“ und „Titelliste“ eine Verletzung der ersten Normalform.


### Negativbeispiel: 1. NF verletzt

#### CDs

CD-ID	Album	Erscheinungsjahr	Titelleiste
4711	Anastacia – Not That Kind	2000	{1. Not That Kind, 2. I'm Outta Love, 3. Cowboys & Kisses }
4712	Pink Floyd – Wish You Were Here	1975	{1. Shine On You Crazy Diamond}
4713	Anastacia – Freak of Nature	2001	{1. Paid My Dues}

Die Lösung ist die Aufteilung der Spalte „Album“ in „Albumtitel“ und „Interpret“ und bezüglich der Spalte „Titelliste“ in „Track“ und „Titel“.



 EUGEN REINTJES SCHULE HAMELN	BG12 IV	Datenbanken
--	---------	-------------

## Lösung

### CDs

cd_id	cd_albumtitel	cd_interpret	cd_erscheinungsjahr	cd_track	cd_titel
4711	Not That Kind	Anastacia	2000	1	Not That Kind
4711	Not That Kind	Anastacia	2000	2	I'm Outta Love
4711	Not That Kind	Anastacia	2000	3	Cowboys & Kisses
4712	Wish You Were Here	Pink Floyd	1975	1	Shine On You Crazy Diamond
4713	Freak of Nature	Anastacia	2001	1	Paid My Dues

### Zweite Normalform:

Die Zweite Normalform liegt vor, wenn die Tabellen in der ersten Normalform vorliegen und Attribute, die nicht Teil des Primärschlüssels sind (Nichtprimärattribut; mehrere Attribute können einen zusammengesetzten Primärschlüssel ergeben! → siehe „CD-ID“ und „Track“). Die Attribute dürfen auch nicht von einer echten Teilmenge eines möglichen Schlüsselkandidaten abhängig sein.

Also sollen nicht-primäre Attribute jeweils nur von ganzen Schlüsseln abhängig sein, sprich nicht nur von einem Teil eines zusammengesetzten Primärschlüssel. So sind hier in diesem Beispiel die Felder „Albumtitel“, „Interpret“ und „Erscheinungsjahr“ von „CD-ID“ abhängig, es besteht jedoch keine Abhängigkeit dieser drei Spalten zu „Track“. Das bedeutet, wir gliedern z. B. bei einem zusammengesetzten Primärschlüssel, der aus zwei Teilen besteht, beide Attribute zu einer neuen Tabelle aus (siehe „CD-ID“ und „Track“[nummer] in der Lösung).

### Negativbeispiel: 2. NF verletzt

#### CDs

cd_id	cd_albumtitel	cd_interpret	cd_erscheinungsjahr	cd_track	cd_titel
4711	Not That Kind	Anastacia	2000	1	Not That Kind
4711	Not That Kind	Anastacia	2000	2	I'm Outta Love
4711	Not That Kind	Anastacia	2000	3	Cowboys & Kisses
4712	Wish You Were Here	Pink Floyd	1975	1	Shine On You Crazy Diamond
4713	Freak of Nature	Anastacia	2001	1	Paid My Dues

**Hinweis zur Tabellenstruktur bezüglich Primärschlüssel:** Wurde eine neue Tabelle erstellt, steht der (zusammengesetzte) Primärschlüssel am Anfang ganz links von der Tabelle. Fremdschlüssel werden am Ende ganz rechts hinzugefügt. Ergeben sich mehrere Fremdschlüssel, werden diese nacheinander am Ende der Tabelle stehen.

## Lösung

### CDs

cd_id	cd_albumtitel	cd_interpret	cd_erscheinungsjahr
4711	Not That Kind	Anastacia	2000
4712	Wish You Were Here	Pink Floyd	1975
4713	Freak of Nature	Anastacia	2001

### Lieder

cd_id	cd_track	cd_titel
4711	1	Not That Kind
4711	2	I'm Outta Love
4711	3	Cowboys & Kisses
4712	1	Shine On You Crazy Diamond
4713	1	Paid My Dues

**Hinweis:** Ähnlich wie in diesem Beispiel 1:n-Beziehungen per Fremdschlüssel in verschiedene Tabellen aufgeteilt wird, werden in der 2. NF m:n-Beziehungen in Hilfs- bzw. Verknüpfungstabellen (Tabelle „Lieder“) aufgeteilt, erkennbar am zusammengesetzten Primärschlüssel.

### Dritte Normalform:

Die dritte Normalform liegt vor, wenn die Datenbank in der zweiten Normalform ist und kein Nichtschlüsselattribut (hellgraue Zellen in der Tabelle) von einem Schlüsselkandidaten transitiv abhängt. In diesem Beispiel lässt sich der „Interpret“ aus „CD-ID“ bestimmen. Das Gründungsjahr hängt jedoch vom „Interpreten“ ab. Somit entsteht die transitive Abhängigkeit.

*Nachfolgend eine Erweiterung der Tabelle durch das Gründungsjahr des Künstlers. Damit unterscheidet es sich von der Ausgangstabelle!*

### Negativbeispiel: 3. NF verletzt

#### CDs

cd_id	cd_albumtitel	cd_interpret	cd_interpret_gründungsjahr
4711	Not That Kind	Anastacia	1999
4712	Wish You Were Here	Pink Floyd	1965
4713	Freak of Nature	Anastacia	1999

Damit eine Datenredundanz verhindert wird, werden die transitiv abhängigen Spalten zu einer neuen Tabelle ausgegliedert. Um eine Zuordnung nach der Ausgliederung wieder zu ermöglichen, wird der Primärschlüssel der neuen Tabelle als Fremdschlüssel in der Tabelle CD aufgenommen.

## Lösung

### CDs

cd_id	cd_albumtitel	k_id
4711	Not That Kind	311
4712	Wish You Were Here	312
4713	Freak of Nature	311

### Künstler

k_id	k_interpret	k_gründungsjahr
311	Anastacia	1999
312	Pink Floyd	1965

### Lieder

cd_id	cd_track	cd_titel
4711	1	Not That Kind
4711	2	I'm Outta Love
4711	3	Cowboys & Kisses
4712	1	Shine On You Crazy Diamond
4713	1	Paid My Dues

Quelle angepasstes Beispiel: Wikipedia - Normalisierung

Weitere Quelle zum Nachlesen:

<https://www.tinohempel.de/info/info/datenbank/normalisierung.htm>

## Übung

Ein Bezirksleiter für mehrere Autohändler verwendet für seine Datenverwaltung, in der die redundanten Daten lediglich zusammengefasst sind. Wenden Sie auf diese Tabelle die drei Schritte bis zur dritten Normalisierung an.

Hersteller	Modell	Eigenschaften	Händler	Adresse	Preis
VW	Golf	blau, Benzin	Auto Müller	Seestraße 1, Hameln	34000
		blau, Diesel			32000
		blau, Benzin	Auto Werner	Oststraße 3, Holzminden	33500
		blau, Diesel			33000
Skoda	Fabia	rot, Benzin	Auto Müller	Seestraße 1, Hameln	14000
			Auto Werner	Oststraße 3, Holzminden	16000
BMW	5er	schwarz, Benzin	Auto Burkhardt	Jägerstraße 4, Bremen	45000
		weiß, Benzin			44000
		schwarz, Benzin	Auto Burkhardt	Am Tor 6, Güstrow	43000
		weiß, Benzin			43000


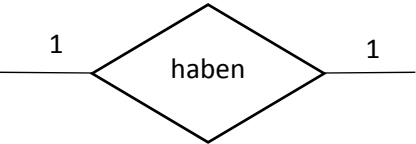
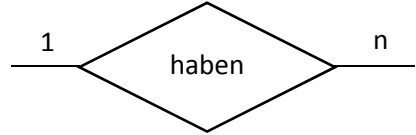
### Theorie von Datenbankmodellierung

Wir kommen im Alltag täglich mit mehreren verschiedenen Datenbanken in Berührung, ohne davon Kenntnis zu nehmen. Sei es beim Abruf von Schülerdaten, bei der Onlinebestellung auf Amazon, die Verwendung von Facebook oder beim Spielen von MMO/MOBAS. Die Daten werden in Tabellen gespeichert. Bevor jedoch eine Datenbank in Tabellenform vorliegt, wird dieses modelliert. Ein Modell ist ein Abbild der Wirklichkeit, gleichzeitig wird auf eine detailreiche aufgrund der hohen Komplexität der Realität verzichtet.

Das Entity-Relationship-Modell (ERM) enthält eine Beschreibung von Daten und eine Darstellung der Objekte und deren jeweilige Beziehungen zueinander in der Form einer Grafik, genannt Entity-Relationship-Diagramm (ERD). Nun beschäftigen wir uns mit der Erstellung von ERD. Situationstexte erläutern die Form der Beziehungen der Objekte.

Beispiel: VW und Bosch sind zwei verschiedene Unternehmen, die gemeinsam durch Teilfertigung den VW Golf in verschiedenen Farben produzieren. Eindeutig identifizierbar ist das produzierte Auto an der Fahrgestellnummer (Primärschlüssel).

Form	Begriff	Erklärung	Beispiel
Objekte	Entität (Entity) <b>[1]</b>	Reales Objekt der realen Welt (Datensatz oder Tupel)	VW Golf
<div style="border: 1px solid black; padding: 5px; display: inline-block;">Autos</div>	Entitätstyp	Gleichartige Entitäten mit denselben Eigenschaften zusammengefasst (Datei)	Autos
<div style="border: 1px solid black; border-radius: 50%; padding: 5px; display: inline-block;">a_farbe</div>	Attribut <b>[2, 3]</b>	Eigenschaft der Entität (Datenfeld)	Farbe, Modell
	Attributsausprägung, Attributswert	Tatsächlicher Wert des Attributs (Feldinhalt)	rot, schwarz
<div style="border: 1px solid black; border-radius: 50%; padding: 5px; display: inline-block;"><u>a_nr</u></div>	Primärschlüssel (PK)	Eindeutige Identifizierung einer Entität	a_nr (Autonummer)
<div style="border: 1px solid black; border-radius: 50%; padding: 5px; display: inline-block;">h_nr</div>	Fremdschlüssel (FK)	Zuordnung eines fremden PK bei einer anderen Entitätsmenge	Hersteller (h_nr) -produzieren - Autos (a_nr, h_nr)

	Domäne	Zulässige Wertebereich eines Attributs (Datentyp)	Buchstaben, Zahlen (Farben vorgeben, z. B. nur rot, gelb)
	Relation (Beziehung) [4]	Beziehung zwischen zwei Entitäten	Hersteller produziert Auto
Chen-Notation	Kardinalität	Gibt an, wie viele Entitäten einer Entitätsmenge mit den Entitäten einer anderen Entitätsmenge in Relation stehen	(Beispiele siehe unten)
	Kardinalität: 1:1-Beziehung [5]	Jede Entität einer Entitätsmenge ist mit einer Entität aus einer anderen Entitätsmenge in Relation	Ein Hersteller produziert ein Auto
	Kardinalität: 1:n-Beziehung [6]	Jede Entität einer Entitätsmenge ist mit mehreren Entitäten aus der anderen Entitätsmenge in Relation	Ein Hersteller produziert ein oder mehrere Autos
	Kardinalität: m:n-Beziehung [7]	Mehrere Entitäten einer Entitätsmenge stehen mit mehreren Entitäten aus der anderen Entitätsmenge in Relation	Ein oder mehrere Hersteller produzieren ein oder mehrere Autos

1. Nomen/Objekte verwenden
2. Gerade Linien bei Relationen verwenden
3. Primärschlüssel unterstreichen
4. Verben für Relationen verwenden
5. Gerade Linien bei Relationen verwenden (90 Grad Winkel ist erlaubt)
6. Fremdschlüssel bei 1:n-Beziehung
7. Hilfstabelle bei m:n-Beziehung; Gestrichelte Umrandung
8. Übertragung ERD -> Tabelle, Tabelle -> ERD; In Zusammenhang bringen

## Transformationsregeln

Vom ER-Modell in Tabellen einer Datenbank  
(Vorbereitung auf die SQL-Programmierung)

## Transformationsregeln

1. Jede Entität bekommt eine eigene Tabelle
  - Jede Tabelle erhält einen eigenen Primärschlüssel
2. Verknüpfung von 1:n-Beziehung
  - n-Seite erhält den Primärschlüssel der 1-Seite als Fremdschlüssel
3. Jede m:n-Beziehung bekommt eine eigene Tabelle
  - Primärschlüssel wird aus den beiden verbundenen Primärschlüsseln der Entitätsmengen zusammengesetzt
4. Jede Relation einer 1:n-Beziehung oder 1:1-Beziehung mit Attributen erhält eine eigene Tabelle

## Tupelschreibweise

- Tabellenname(Primärschlüssel, Attribut 1, Attribut 2, ..., Fremdschlüssel (1), Fremdschlüssel (2), ...);
- Lehrer(l\_nr, l\_name, l\_telefonnummer, ...);
- Hinweis: Jeden Entitätstypen bzw. Tabelle mit einem Semikolon beenden (Programmierungsgründe)

## Aufgabe

Erstellen Sie Transformation der Tabellen für die vergangene ERD-Situation „Musikwelt“, indem Sie die Tupelschreibweise verwenden.