Datenbank

Teil 1 SQL

Inhalt

1 Einführung

Wo werden Datenbanken überall eingesetzt?

Für welche Berufsgruppen sind Datenbankkenntnisse Vorteilhaft?

Was für Datenbanken kennen wir?

Was sind Anforderungen an Datenbanken?

2 Datenbanken Typen

Es wird heute häufig zwischen SQL und NoSQL Datenbanken unterschieden. Bei genauerer Betrachtung gibt es aber mehr Datenbank Typen, die man unterscheiden kann:

- Relationale Datenbanken
  z.B. Oracle, MySQL, Postgress
- Dokumentenorientierte Datenbanken
  z.B. MongoDB
- Key Value Datenbanken
  z.B. Redis
- Graph-Datenbanken
  z.B. ArangoDB

2.1 Datenbank Begriffe

- DB - Datenbank
- DBS - Datenbanksystem
- DBMS – Datenbank Management System
- CRUD – Create, Read, Update, Delete

2.2 Datenbankmodelle

2.2.1 Relationales Modell

relationales Modell		informeller Begriff		Erklärung
Deutsch	**Englisch**	**Deutsch**	**Englisch**	
Relation, Entitätstyp, Entitätsklasse	relation	Tabelle	table	Eine Tabelle in einer Datenbank
Tupel, Entität	tuple, entity	Zeile	row	Ein horizontaler Datensatz einer Tabelle in der Datenbank
Beziehung	relationship			Beziehung einzelner Tupel zueinander
Kardinalität	cardinality			Mengenangabe zur Beziehung einzelner Tupel (z. B. 1:1, 1:n, n:m)
Attribut	attribute	Spalte	column	vertikaler Spaltenindex einer Tabelle
Grad	degree			Anzahl der eindeutig identifizierenden Attribute
Primärschlüssel	primary key			eindeutiger Identifikator
Fremdschlüssel	foreign key			Schlüssel aus einer anderen Tabelle, um eine Beziehung herstellen zu können
Wertebereich	domain	Typ	type	Werte, die ein Attribut annehmen kann
Skalar	scalar	Wert	value	Wert eines Attributes in einem Tupel

Quelle: https://de.wikipedia.org/wiki/Relation_(Datenbank)

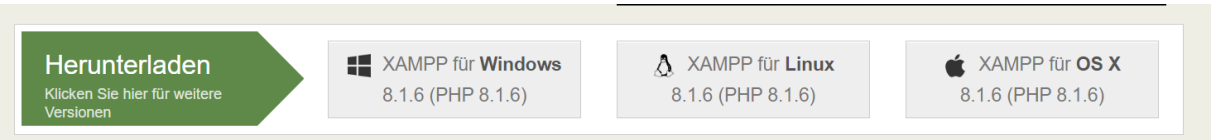
3 Die Relationale Datenbank MySQL/ MariaDB

MySQL und MariaDB basieren auf der gleichen Implementierung. Das heißt beide Datenbankensysteme sind weitgehendst kompatibel. MySQL gehört zu Oracle und unterliegt verschiedenen Lizenzen. MariaDB ist komplett Open Source und wird von einer Community weiterentwickelt. MariaDB hat in einigen Fällen bessere Performance und auch einige geschlossene Sicherheitslücken.

Obwohl wir von MySQL sprechen, verwenden wir eigentlich genau genommen MariaDB (auch im XAMPP enthalten). Von der Benutzung sind beide Systeme zum großen Teil identisch.

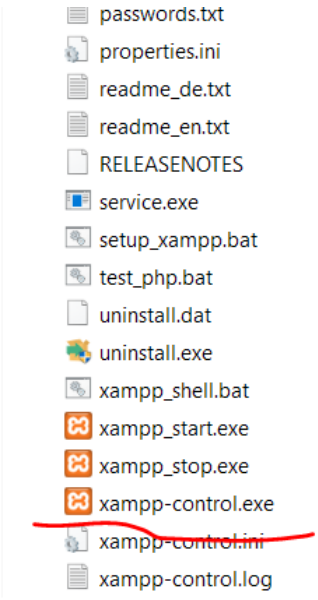
3.1 MySQL Installation

Komplettpaket XAMPP

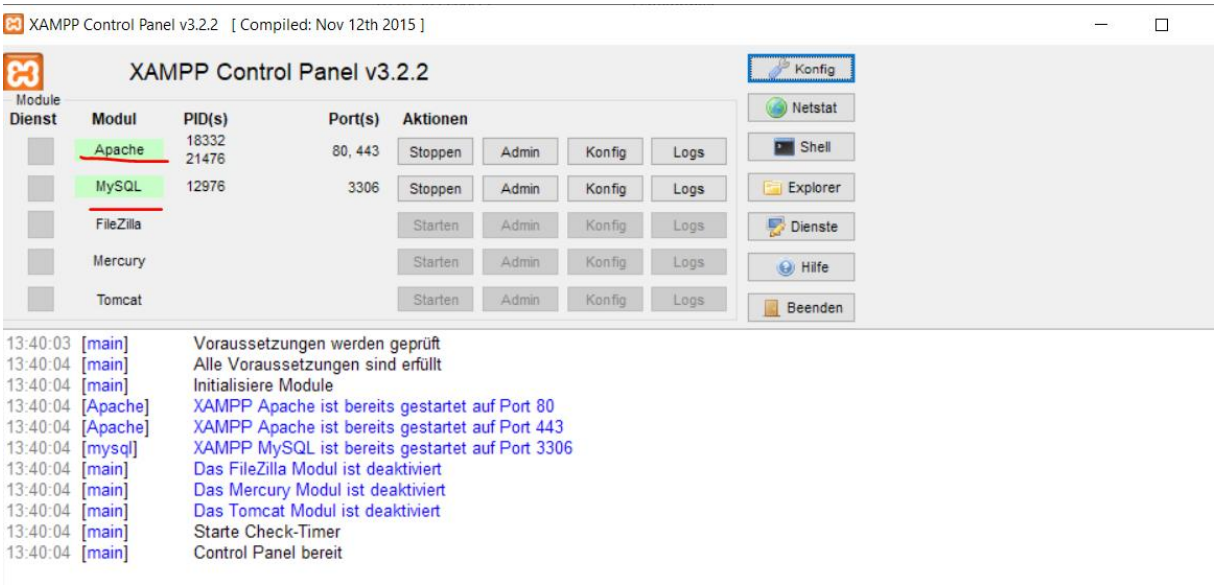


https://www.apachefriends.org/de/index.html

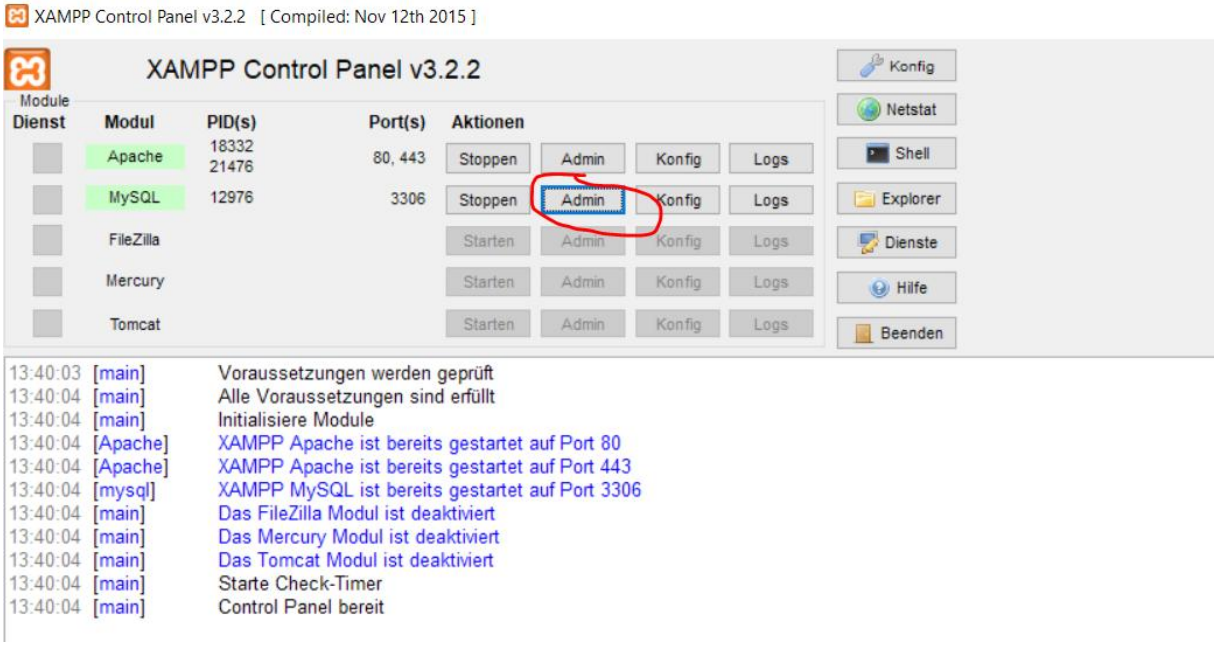
Wichtig: unter Windows direkt unter c:/xampp installieren!



Im XAMPP Control Panel muss der Webserver (Apache) und MySQL gestartet werden.



Mit dem Admin Button von MySQL kommen wir zum MySQL Client PHPMyAdmin:

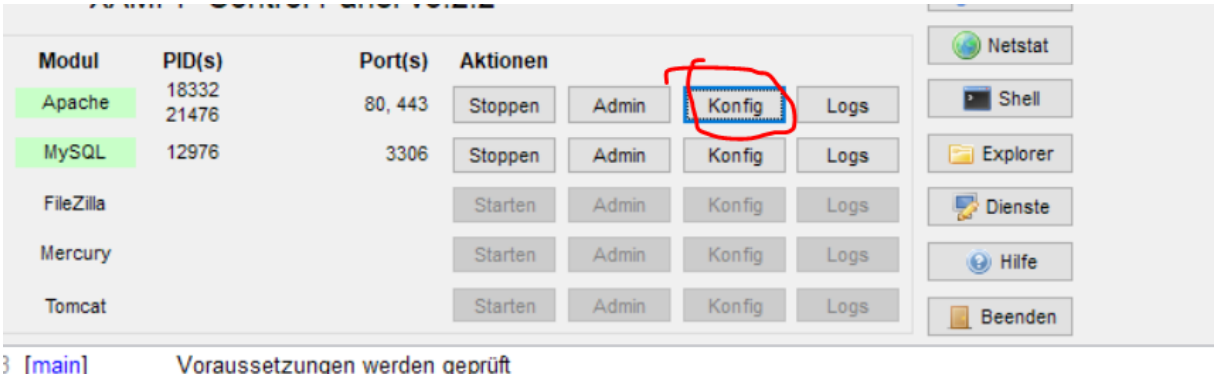


3.2 PHPMyAdmin Client

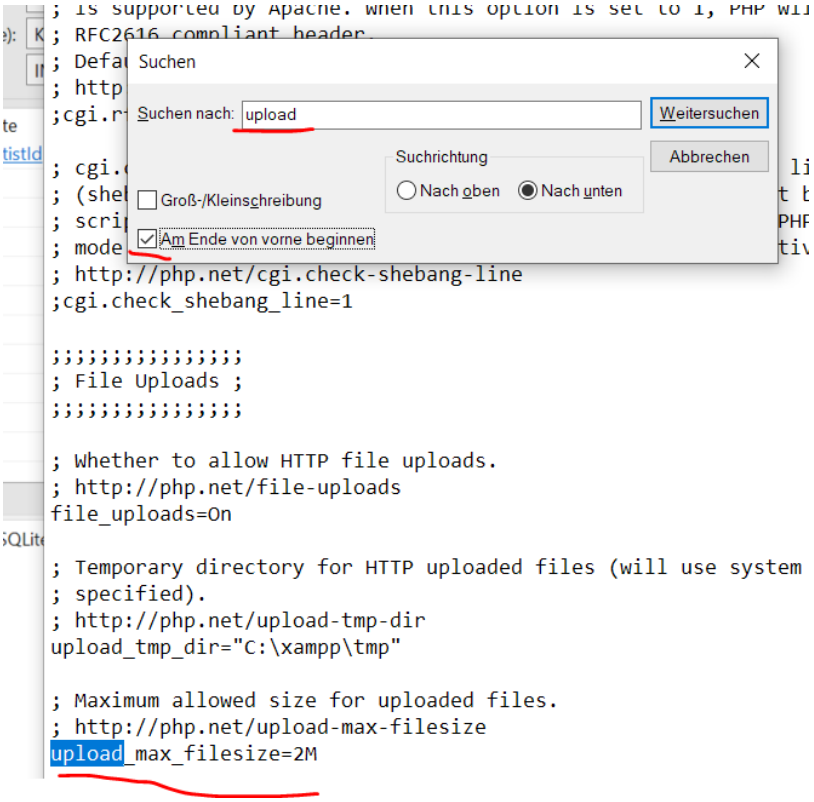
3.2.1 XAMPP Konfiguration für den Daten Import

Das Importieren von Daten erfolgt in der Regel über *.sql Skripte. Da diese sehr groß sein können, müssen wir XAMPP so konfigurieren, dass er auch mit großen Dateien keine Probleme hat.

Unter Konfig PHP (php.ini) auswählen.

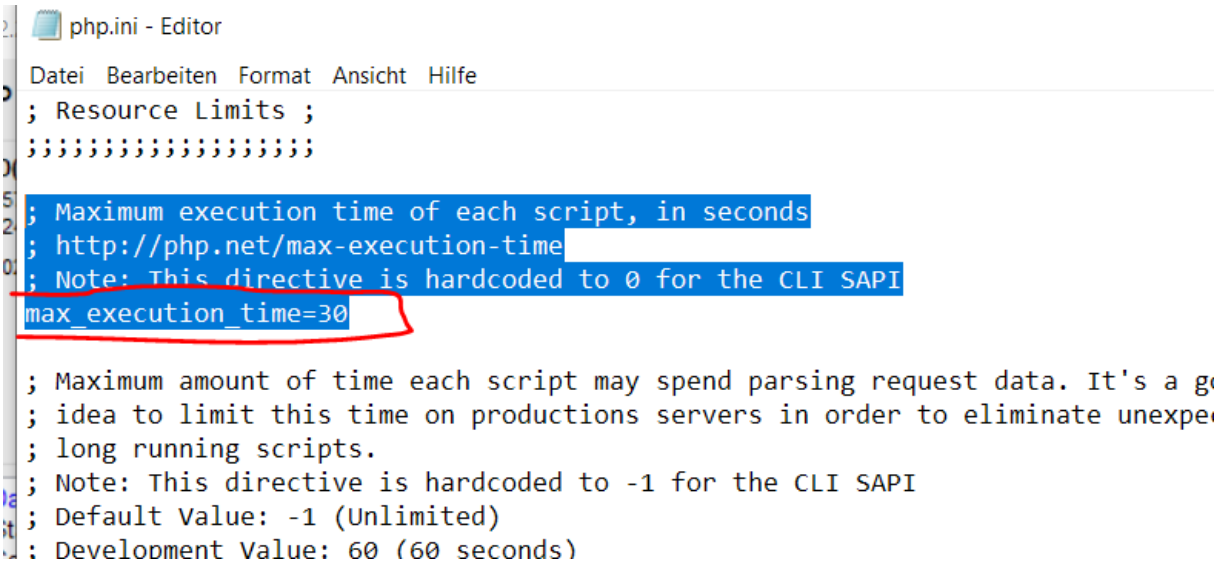


In der Konfigurationsdatei suchen wir nach *upload*:



Kann man z.B. auf 50M (50 MB) erhöhen und Datei speichern.

Des Weiteren empfiehlt es sich die **max_execution_time** zu erhöhen (z.B. 500):

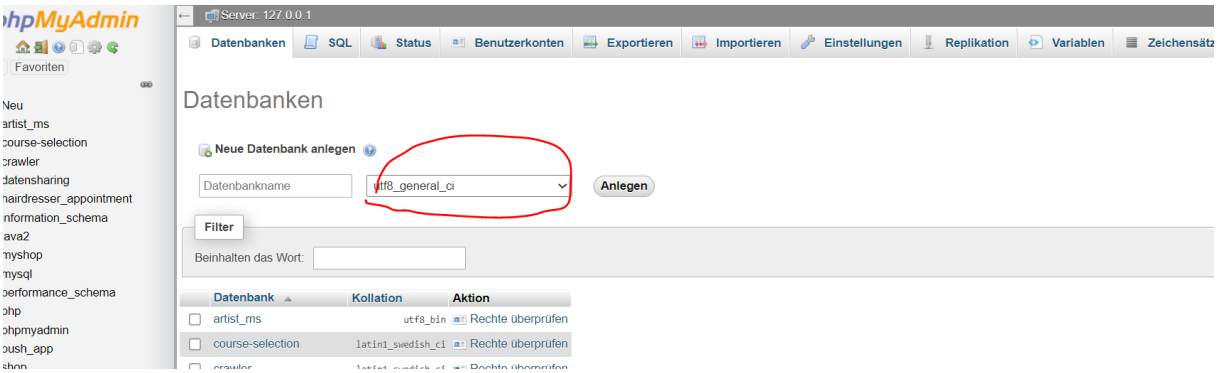


Achtung, danach muss Apache nochmal gestartet werden, damit die Konfiguration neu eingelesen wird.

3.3 Datenbank anlegen

Unter dem Punkt Datenbanken können wir diese anlegen. Wichtige ist UTF8-Zeichensatz einzustellen:

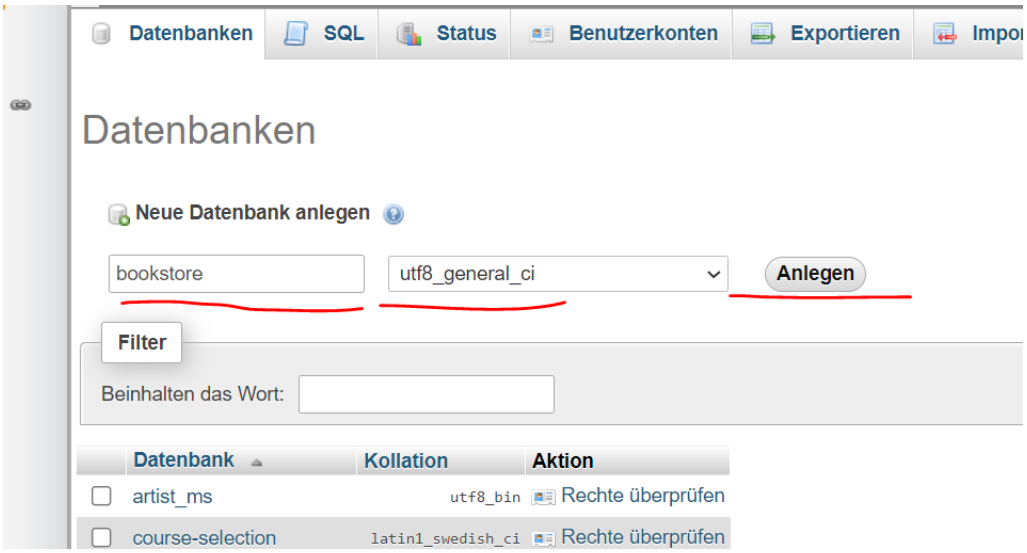
Achtung! MySQL hat mehrere UTF-8 Zeichensatz Varianten. Wenn für uns deutsche Umlaute relevant sind, sollten wir utf8_german2_ci einstellen. Das kann auch für jede Spalte der Tabelle einzeln später



Vor dem Anlegen einer Datenbank sollte als Zeichensatz sollte **utf8_general_ci** gewählt werden.

Das ist wichtig für die Darstellung von Sonderzeichen und Umlauten bzw. auch für die Sortierung.

Die Datenbank *bookstore* anlegen.



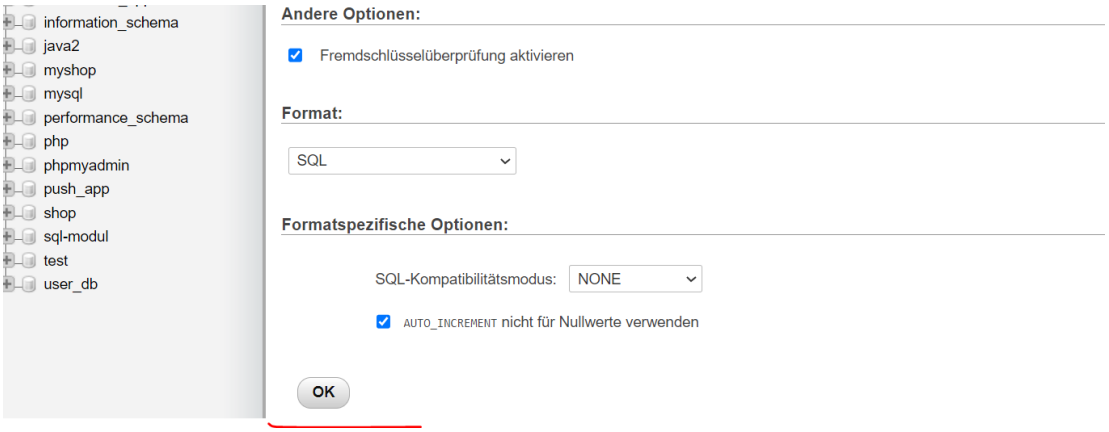
3.4 Importieren von Tabellen

Vor dem Import muss die Datenbank ausgewählt sein.

Danach gehen wir auf Datei auswählen und wählen die Datei bookstore.sql.bz2.

➔ Es können komprimierte und unkomprimierte Formate benutzt werden.

Links unten mit Ok bestätigen:



Der Import kann mehrere Minuten dauern! Falls der Import abbricht, sollte vorher die Konfiguration (siehe oben) vorgenommen werden.

3.5 Installation DBeaver

DBeaver ist ein ähnlich wie PHPMyAdmin ein weiterer Client für MySQL und andere relationale Datenbanken. Vor allem das ausführen und verwalten funktioniert mit diesem Programm sehr gut.

Letztendlich ist DBeaver für uns aber nur eine weitere praktische Alternative zu PHPMyAdmin.

Download: https://dbeaver.io/

Bitte die Community Edition wählen.

4 Werkzeuge und Editoren

4.1 DBEaver

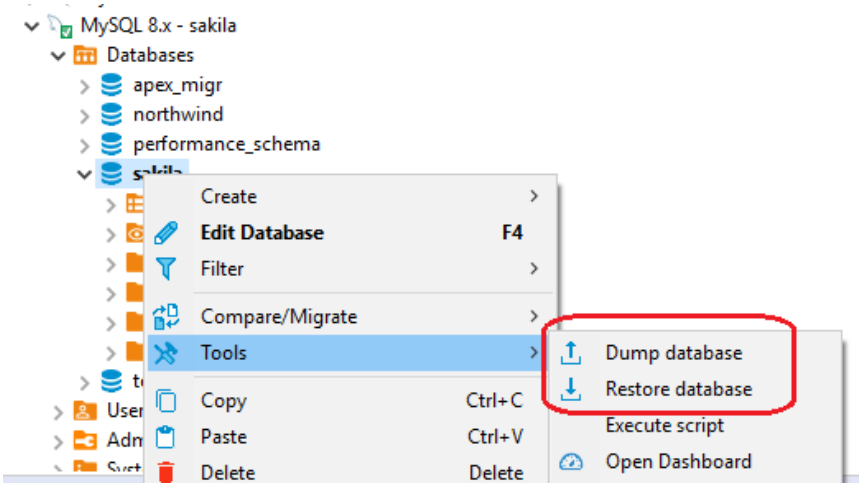
4.1.1 Dokumentation

https://dbeaver.com/docs/wiki

4.1.2 Backup Restore

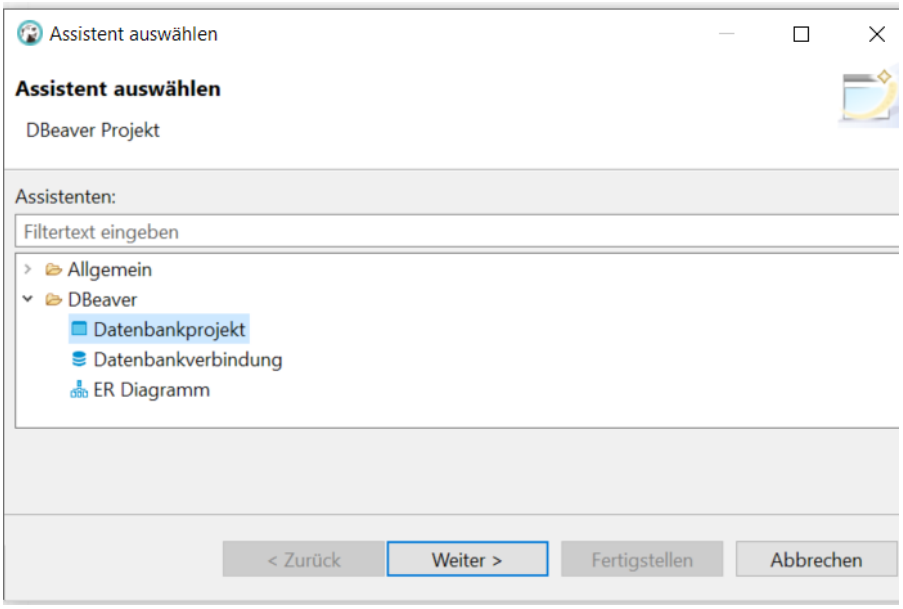
Für Backups werden in der Regel als sql-Datei gespeichert. Speichern im Binärformat ist allerdings auch möglich.

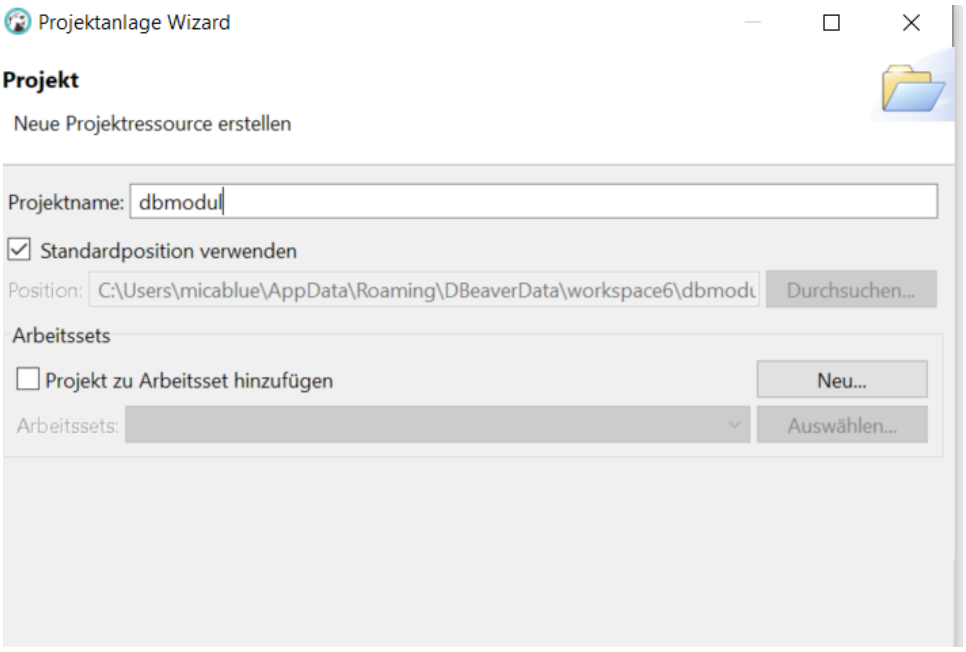
https://dbeaver.com/docs/wiki/Backup-Restore/

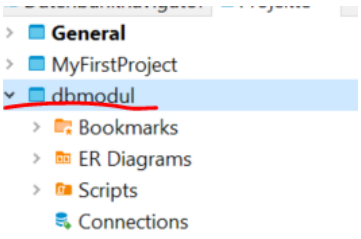


4.1.3 Neues DBEaver Projekt

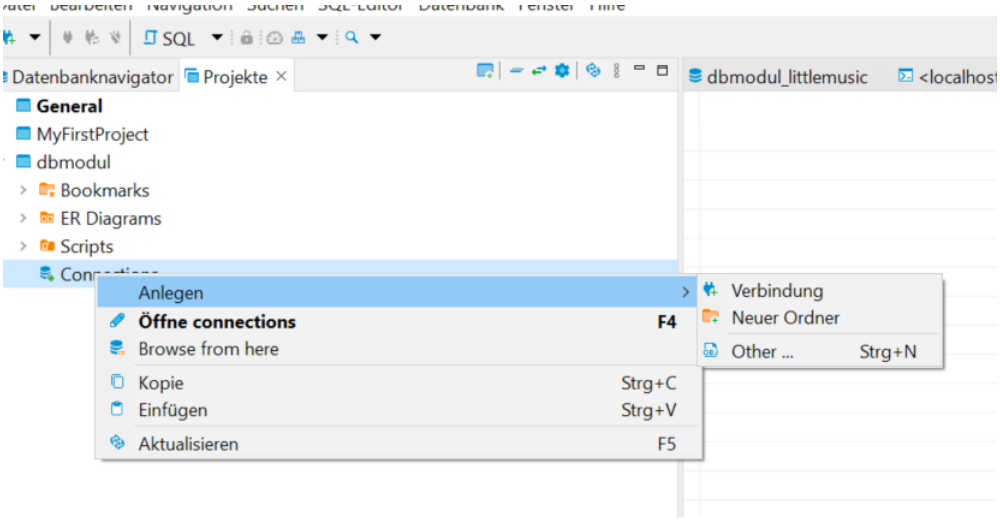
Menü/ Datei/Neu



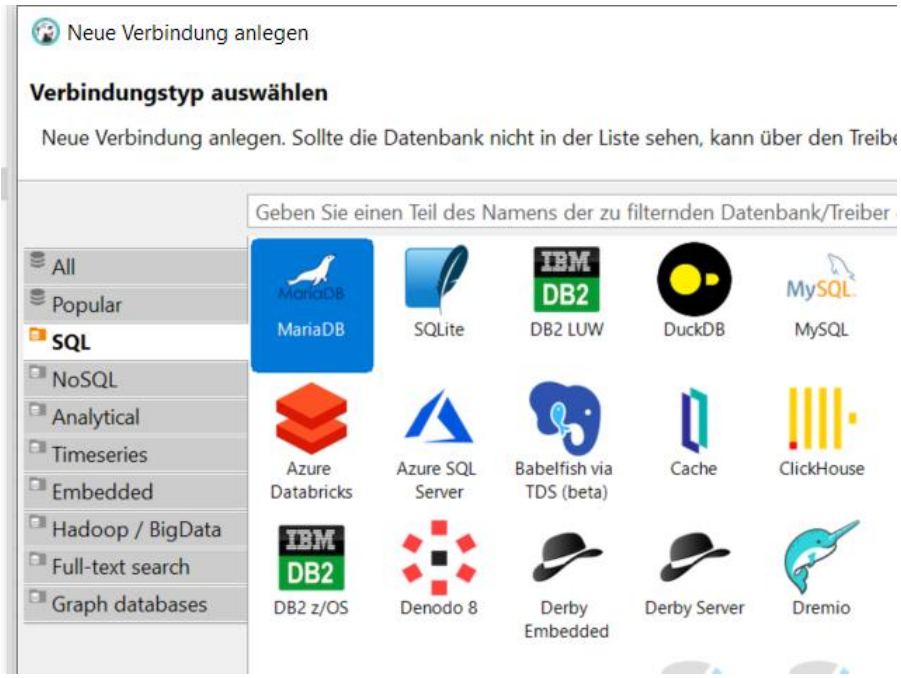


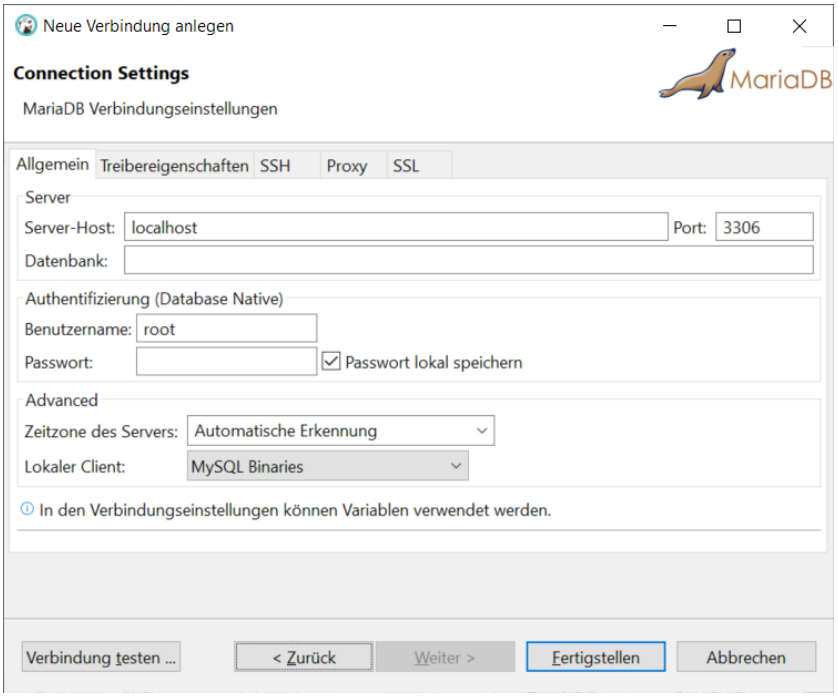


Jetzt kann eine Connection hergestellt werden. Dazu muss die Datenbank vorher gestartet sein.

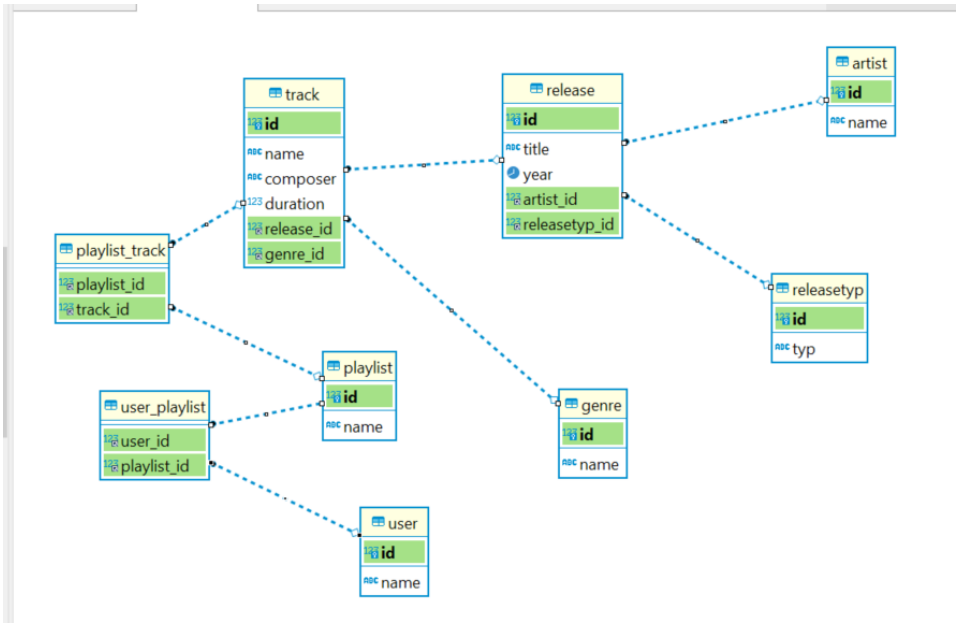


MariaDB oder MySQL aussuchen


Neue Verbindung anlegen
Verbindungstyp auswählen
Neue Verbindung anlegen. Sollte die Datenbank nicht in der Liste sehen, kann über den Treibe
Geben Sie einen Teil des Namens der zu filternden Datenbank/Treiber
All
Popular
SQL
NoSQL
Analytical
Timeseries
Embedded
Hadoop / BigData
Full-text search
Graph databases
MariaDB
SQLite
DB2 LUW
DuckDB
MySQL
Azure Databricks
Azure SQL Server
Babelfish via TDS (beta)
Cache
ClickHouse
DB2 z/OS
Denodo 8
Derby Embedded
Derby Server
Dremio

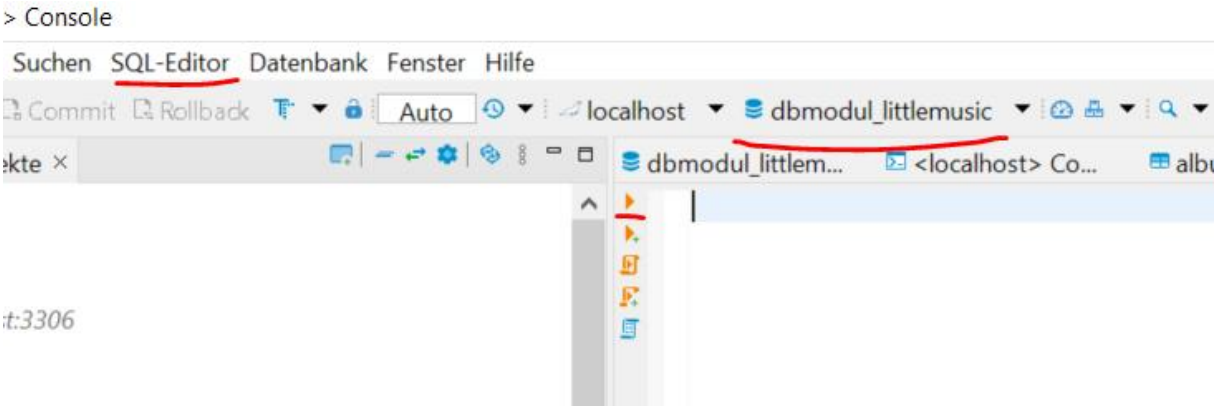

Neue Verbindung anlegen
Connection Settings
MariaDB Verbindungseinstellungen
MariaDB
Allgemein
Treibereigenschaften
SSH
Proxy
SSL
Server
Server-Host:
localhost
Port:
3306
Datenbank:
Authentifizierung (Database Native)
Benutzername:
root
Passwort:
Passwort lokal speichern
Advanced
Zeitzone des Servers:
Automatische Erkennung
Lokaler Client:
MySQL Binaries
In den Verbindungseinstellungen können Variablen verwendet werden.
Verbindung testen ...
< Zurück
Weiter >
Fertigstellen
Abbrechen

4.2 Die Beispieldatenbank: dbmodul_musicdb



4.2.1 DBEaver SQL Editor

Der SQL Editor ist über das Menü zu erreichen. Man muss sicherstellen, dass die richtige Tabelle ausgewählt ist, siehe Bild:

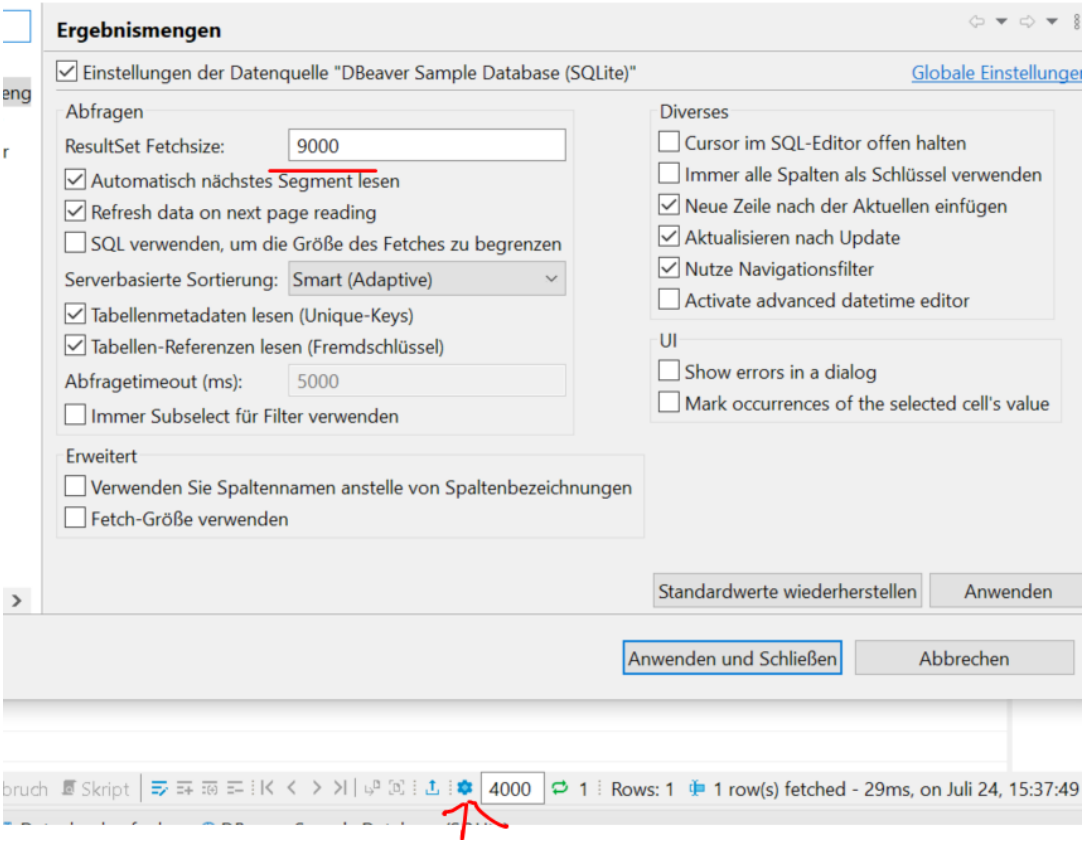


Die Anweisungen lassen sich über die Icons (links im Editor) oder rechte Maustaste ausführen.

- ➔ Bei mehreren Anweisungen aus das Icon *Skript ausführen*
- ➔ Über der Tabelle kann auch SQL angegeben werden, um die Ergebnisse vorher zu filtern.

4.2.2 Ändern der Ergebnismenge

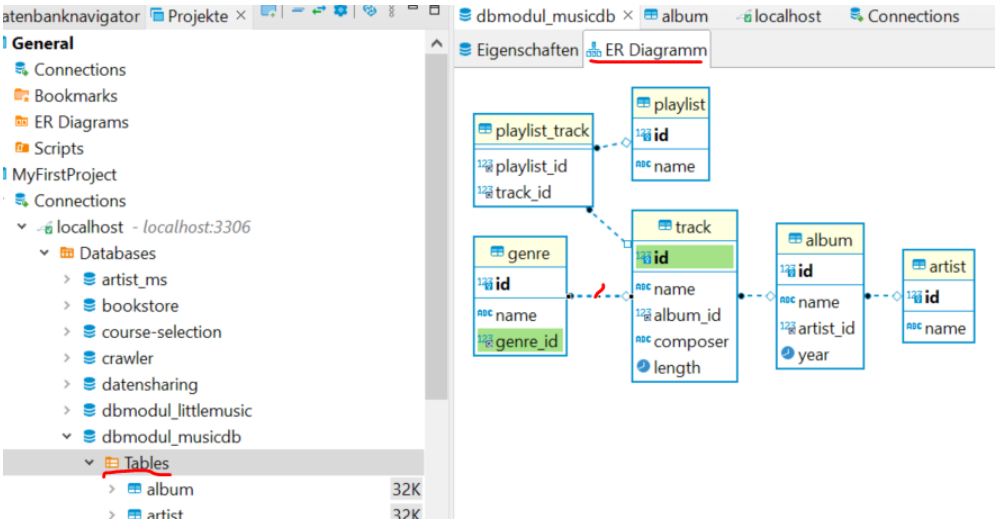
Standardmäßig werden maximal bis 200 gefundene Datensätze angezeigt. Den Defaultwert kann man hier ändern:



Achtung, eine größere Zahl belastet den Hauptspeicher des Rechners!

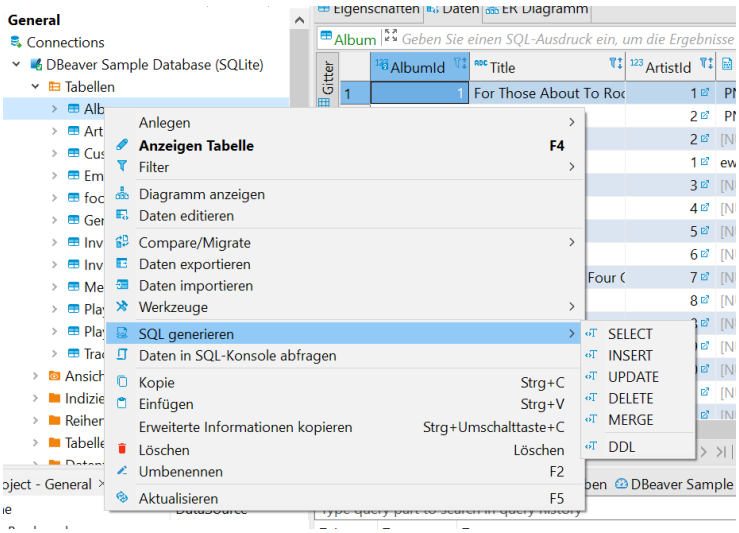
4.2.3 DBeaver ER-Diagramm

Die ER Diagramm Ansicht erscheint, wenn man im Project Explorer auf der linken Seite das Tables-Icon markiert. Beim Klick auf die Verbindungslinie erscheint die verbundene Fremdschlüssel Beziehung.



4.2.4 DBeaver SQL generieren

In der Datenansicht Zeilen markieren und rechte Maustaste SQL exportieren:



5 SQL -SELECT

SQL ist die Abkürzung für Structured Query Language, die Abfragesprache für relationale Datenbanken. SQL ist für relationale Datenbanken allgemein gültig. Jede Datenbank hat aber ihre Besonderheiten und Erweiterungen des Standards. Um SQL für verschiedene Datenbanken einzusetzen, muss es häufig angepasst werden.

Wir unterscheiden in SQL drei Gruppen:

DDL → Data Definition Language:

- Zum Anlegen und Löschen von Datenbanken und Tabellen

DML → Data Manipulation Language

- Zum Abfragen, Einfügen und Verändern von Daten

DCL → Data Control Language

- Um Benutzer zu verwalten und Rechte zu setzen

Zeichen

*	Alle Spalten /Felder bei Select-Anweisungen
`...`	Backtick um, Felder, Tabellen- und Datenbanknamen einzuschließen -ist optional, aber bei der Verwendung von Schlüsselwörtern in Bezeichnern Pflicht
--	Kommentar

;	Semikolon, Ende der Anweisung (kann bei einigen Clients weggelassen werden)
'…'	Einfache Hochkomma für Zeichenketten

5.1 Select Anweisungen

MySQL unterscheidet standardmäßig nicht zwischen Klein -und Großschreibung. Dennoch ist es eine einheitliche Konvention zu benutzen.

Zum Beispiel: Namen von Tabellen, Datenbanken, Feldern immer klein.

5.2 Erste Beispiele für Select

Alle Datensätze und alle Felder

```
SELECT * FROM mydb.student;    -- DB ist häufig optional
```

Die die id aber alle Datensätze:

```
SELECT id FROM student;
```

Diese Abfrage gibt bestimmte Attribute einer Tabelle auf dem Constraint Employee ID =0000 aus.

```
SELECT EMP_ID, NAME FROM EMPLOYEE_TBL WHERE EMP_ID = '0000';
```

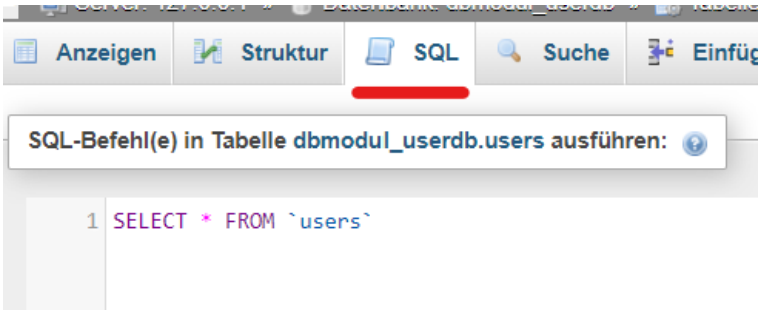
Mit Sortierung nach EMP_ID

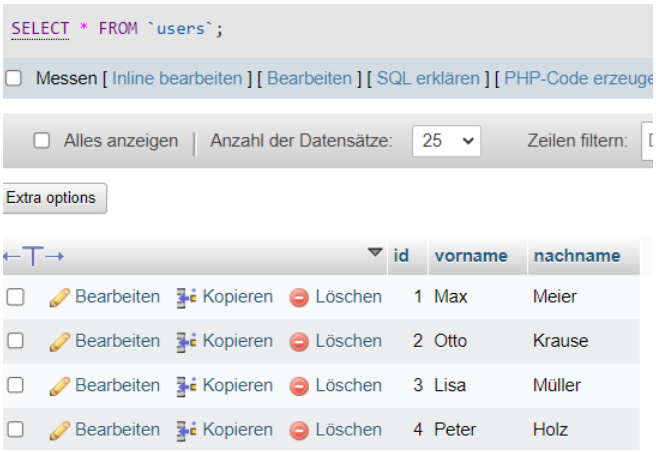
```
SELECT EMP_ID, LAST_NAME FROM EMPLOYEE
WHERE CITY = 'Seattle' ORDER BY EMP_ID;
```

ASC Aufsteigende Sortierung, *DESC* Absteigende Sortierung

```
SELECT EMP_ID, LAST_NAME FROM EMPLOYEE_TBL
WHERE CITY = 'INDIANAPOLIS' ORDER BY EMP_ID asc;
```

Select im PHPMyAdmin





5.3 Select mit Beispieldatenbank *customers*

Anzahl ermitteln mit count()

```
SELECT count(*) FROM `customers`;
```

Limit auf 10 Datensätze begrenzen

```
SELECT * FROM `customers` LIMIT 10;
```

Sortieren: absteigend (ASC=aufsteigend, default)

```
SELECT title, lastname FROM `customers` ORDER BY lastname DESC;
```

Mit *where*: Nur Customer, die in Berlin wohnen

```
SELECT id, lastname,city FROM `customers` WHERE city='Berlin';
```

5.4 Select mit Verknüpfung und Vergleich

5.4.1 Vergleichsoperatoren

>	größer

>=	Größer gleich
<=	Kleiner gleich
<	kleiner
=	gleich
<>, !=	ungleich
IN	Ist in Menge
IS / IS NOT	Vergleicht einen Wert mit einem booleschen Wert, wobei boolean_value TRUE, FALSE oder UNKNOWN sein kann.
BETWEEN	zwischen

Logische Operatoren

OR, ||	oder
AND, &&	und
XOR	Entweder oder
NOT, !	Nicht

```
SELECT title, lastname,city FROM `customers` WHERE title='Frau
Dr.' OR title='Herr Dr.';
```

AND hat mehr Priorität als OR. OR muss also in diesemFall in Klammern gesetzt werden.

```
SELECT title, lastname,city FROM `customers` WHERE (title='Frau
Dr.' OR title='Herr Dr.') AND age < 30;
```

Kombination mit count()

```
SELECT count(*) FROM `customers` WHERE age < 50;
```

Längere Befehle können im Editor auch zur besseren Lesbarkeit eingerückt werden

```
1 SELECT count(*) FROM `customers`
2 WHERE age < 50
```

5.5 Eindeutigkeit mit Distinct

Gibt nur unterschiedliche (eindeutige) Nachnamen aus

```
SELECT DISTINCT lastname FROM customers;
```

Die Kombination von *lastname* und *firstname* wird nur einmal ausgegeben

```
SELECT DISTINCT firstname, lastname FROM customers;
```

5.6 Filtern mit dem LIKE -Operator

Der LIKE-Operator wird in einer WHERE-Klausel verwendet, um nach einem bestimmten Muster in einer Spalte zu suchen.

Es gibt zwei Platzhalter, die häufig in Verbindung mit dem LIKE-Operator verwendet werden:

- Das Prozentzeichen (%) steht für null, ein oder mehrere Zeichen
- Der Unterstrich (_) steht für ein einzelnes Zeichen

Das Prozentzeichen und der Unterstrich können auch kombiniert verwendet werden!

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

5.6.1 Like Beispiele

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

5.6.2 Beispiele mit der Tabelle *customers*

Alle Vornamen mit „A“ beginnend

```
SELECT * FROM `customers` WHERE firstname LIKE 'A%';
```

id	title	firstname	lastname	street	city	email	age
19	Frau Prof. Dr.	Alwin	Brunner	Larissa-Bartsch-Gasse 6/3	Ditzingen	hess.arnd@gmx.de	
30	Herr Prof. Dr.	Anne	Bruns	Ulla-Neumann-Platz 286	Tübingen	irmtraut50@gmx.de	
39	Frau	Albert	Fleischer	Westphalallee 9/5	Staßfurt	anneliese07@yahoo.de	
48	Herr	Anke	Hesse	Dietmar-Kiefer-Allee 967	Meißen	margitta.heim@t-online.de	

Beginnend mit „A“ gefolgt von 3 beliebigen Zeichen

```
SELECT * FROM `customers` WHERE firstname LIKE 'A___';
```

id	title	firstname	lastname	street	city	email	age
30	Herr Prof. Dr.	Anne	Bruns	Ulla-Neumann-Platz 286	Tübingen	irmtraut50@gmx.de	50
48	Herr	Anke	Hesse	Dietmar-Kiefer-Allee 967	Meißen	margitta.heim@t-online.de	51
99	Herr Prof. Dr.	Arno	Zander	Sylke-Kruse-Allee 00c	Flensburg	johanna.schade@posteo.de	47
102	Herr	Arno	Voß	Ria-Graf-Platz 59	Puchheim	grete39@hotmail.de	45
124	Herr Dr.	Alma	Scheffler	Rosi-John-Weg 7b	Bottrop	johanne.kluge@yahoo.de	57

Alle E-Mails, die gmail.com sind.

```
SELECT * FROM `customers` WHERE email LIKE '%@gmail.com';
```

Hier werden auch die googlemail-Adressen gefunden:

```
SELECT * FROM `customers` WHERE email LIKE '%@g%mail.com';
```

	email	age
	otto77@gmail.com	58
	rsauer@googlemail.com	79
	christa41@googlemail.com	31
)	bkoster@gmail.com	23
	fleischmann.silvio@googlemail.com	62

5.7 Filtern mit BETWEEN und IN

5.7.1 Abfrage mit dem IN-Operator

Der IN-Operator ist vereinfacht bestimmte Abfragen. Er funktioniert ähnlich wie ein „oder“.

```
SELECT * FROM customers WHERE firstname IN('Lilo','Nico','Arno');
```

id	title	firstname	lastname	street	city	email
12	Frau Prof. Dr.	Lilo	Niemann	Mayallee 9/3	Husum	christa41@googlemail.com
32	Frau Dr.	Nico	Grimm	Bärgasse 91b	Seevetal	fleischmann.silvio@googlemail.com
80	Herr Dr.	Nico	Hiller	Wilhelmine-Fleischmann-Platz 37a	Kaltenkirchen	ebert.elfriede@freenet.de
99	Herr Prof. Dr.	Arno	Zander	Sylke-Kruse-Allee 00c	Flensburg	johanna.schade@posteo.de
102	Herr	Arno	Voß	Ria-Graf-Platz 59	Puchheim	grete39@hotmail.de

5.7.2 Abfrage mit dem BETWEEN-Operator

Wir können mit BETWEEN nach Wertebereichen filtern.

Alle Personen mit dem Alter zwischen 20 und 60 (einschließlich 20 und 60).

```
SELECT * FROM `customers` WHERE age BETWEEN 20 AND 60;
```

In MySQL kann der BETWEEN – Operator auch für Text verwendet werden. Die Werte müssen allerdings zu dem „Text“ – Datentyp passen.

Es genügt auch nur der Anfangsbuchstabe.

Beispiel:

```
SELECT * FROM `customers` WHERE lastname BETWEEN 'A' AND 'C';
```

Achtung! Das „C“ wird NICHT eingeschlossen → exklusiv

Wir erhalten Nachnamen mit A bis B:

id	title	firstname	lastname ▲ 1	street	city
28	Frau	Renata	Albers	Elmar-Seidl-Platz 9	Sankt Ingbert
113	Herr Dr.	Hasan	Albers	Helena-Bühler-Platz 4b	Jülich
156	Frau	Leonid	Auer	Sylvia-Harms-Weg 61b	Stendal
16	Frau Prof. Dr.	Hanspeter	Bader	Hoffmannallee 7	Heide
189	Herr Prof. Dr.	Robert	Bader	Gebhard-Nowak-Allee 66b	Leonberg
183	Herr Prof.	Wendelin	Bartels	Henrik-Herold-Straße 2	Hanau
176	Frau Dr.	Giesela	Barthel	Ralph-Jahn-Platz 5/2	Kehl
81	Frau	Gerda	Bartsch	Pieperallee 55	Wolfsburg

5.8 Prioritäten und Operatoren

Die verschiedenen Operatoren haben verschiedene Prioritäten. Der AND-Operator hat zum Beispiel höhere Priorität als der OR-Operator. Um korrekte Ergebnisse zu bekommen, müssen evtl. Klammern gesetzt werden.

```
1.  INTERVAL
2.  BINARY, COLLATE
3.  !
        -   (unary minus), ~ (unary bit inversion)
4.  ^
5.  *, /, DIV, %, MOD
6.  -, +
7.  <<, >>
8.  &
9.  |
10. = (comparison), <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN, MEMBER OF
11. BETWEEN, CASE, WHEN, THEN, ELSE
12. NOT
13. AND, &&
14. XOR
15. OR, ||
16. = (assignment), :=
```

5.9 Spalten für Select Abfragen umbenennen mit „as“

lastname wird in diesem Beispiel in *lname* für die Ausgabe umbenannt. Die Umbenennung ist **nur** für die Ausgabe temporär gültig.

```
SELECT lastname as lname FROM `customers`;
```

	lname
Kopieren Löschen	Mai
Kopieren Löschen	Heinrich

5.10 Funktionen

5.10.1 max und min Funktion

```
SELECT max(age) FROM `customers`;
```

Gibt 80 zurück

Oder kombiniert mit min

```
SELECT max(age) as max_age, min(age) as min_age FROM `customers`
```

Zusätzlich mit Mittelwert

```
SELECT
max(age) as max_age,
min(age) as min_age,
avg(age) as avg
FROM `customers`;
```

max_age	min_age	avg
80	18	47.1150

Achtung: max()/min() sind *Gruppierungsfunktionen* und dürfen nicht in der WHERE Klausel der selben Select-Anweisung eingesetzt werden:

SELECT * FROM `baby_names` WHERE `year` < max(year); -- geht nicht

Falls erforderlich, können dafür Subselects eingesetzt werden:

```
SELECT * FROM `baby_names`
WHERE `year` < (SELECT max(year) FROM baby_names);
```

Siehe Seite 35 Subselect/ Sub-Query

5.10.2 concat()

```
SELECT concat(firstname,' ',lastname) as Name FROM `customers`;
```

Name
Dana Mai
Gaby Heinrich
Simon Behrens
Meinolf Kolb
Waldemar Gottschalk

5.11 Weitere nützliche Funktionen

Referenz MySQL Funktionen:

https://dev.mysql.com/doc/refman/8.0/en/functions.html

upper()	Großbuchstaben
lower()	Kleinbuchstaben
sum()	summieren
curdate()	Aktuelle Datun
date_format()	Formatiert das Datum mit einem Pattern
length()	Die Länge der Zeichenkette
trim()	Entfernt führende und abschließende http://localhost/phpmyadmin/index.phpLeerzeichen bzw. Whitespace (auch Steuerzeichen)
now()	Aktuelle Datum und Zeit
replace()	Ersetzt alten Text durch neuen Text

5.11.1 length()

```
SELECT length(lastname) FROM `customers`;
```

Alle Nachnamen, deren Länge gleich 4 ist

```
SELECT * FROM `customers` WHERE length(lastname)=4;
```

5.11.2 Datumsfunktionen

CURDATE, CURTIME

Geben das aktuelle Datum oder den aktuellen Tag zurück. Lassen sich sehr schön in WHERE-Blöcken einsetzen.

```
SELECT * FROM tabelle WHERE datum < CURDATE();
```

DATEDIFF

Ermittelt die Anzahl von Tagen, die zwischen zwei Zeitangaben liegt.
In diesem Beispiel also die heutigen Termine.

```
# Ermittelt alle Geburstagskinder
SELECT name FROM Tabelle WHERE DATEDIFF(CURDATE(), geburtstag) = 0;
```

Beispiel 2:

```
SELECT DATEDIFF(CURDATE(), datum), datum FROM termine;
```

DATEFORMAT

Wandelt das Standarddatumsformat von MySQL um. Die Parameter findet ihr unter der vollständigen Liste.

https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html#function_date-format

```
SELECT DATE_FORMAT(datum,'%d.%m.%Y') FROM test;
```

NOW

Ermittelt die aktuelle Uhrzeit inklusive Datum. Eignet sich zum Beispiel hervorragend bei Bestellungen in einem Online-Shop, wo man eine genaue Zeitangabe benötigt.

```
INSERT INTO bestellung (datum) VALUES(NOW());
```

6 SQL – Manipulieren von Datensätzen- Insert, Update, Delete

6.1 Insert zum Einfügen von Datensätzen

Syntax

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Achtung, die Anzahl der Felder muss mit der Anzahl der Werte übereinstimmen. Id mit Auto-Increment darf nicht, oder nur mit null gesetzt werden. Achte bei den Werten auf die korrekten Datentypen!

Beispiel 1:

INSERT INTO tasks(title,priority)

VALUES('Learn MySQL INSERT Statement',1);

Beispiel 2:

```
INSERT INTO `farbe` (`id`,`name`, `hexwert`)
VALUES (NULL, 'grau', '#555555');
```

Beispiel 3:

```
INSERT INTO `farbe` (`name`, `hexwert`)
VALUES ('schwarz', '#000000'),
       ('blau', '#0000ff');
```

6.1.1 Automatische Konvertierung bei INSERT

In einigen Fällen findet beim Setzen eines Wertes eine automatische Konvertierung statt.

Das ist vor allem für Date und Date-Time Felder relevant. Hier können wir die Werte im String/ Zeichenketten-Format angeben.

Beispiel:

```
INSERT INTO farbe (name,hexwert,date_time)
VALUES ('black','#000000','2022-08-14 13:49:31');
```

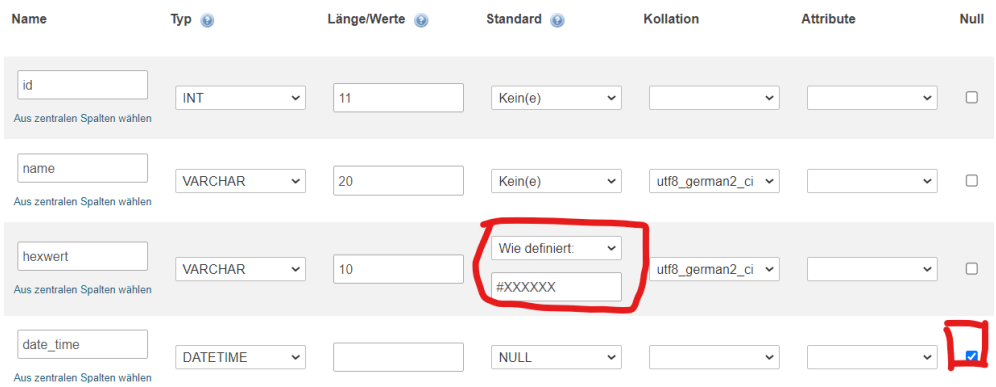
In diesem Beispiel ist der MySQL-Datentyp date-time

Welche Felder müssen beim INSERT Werte bekommen?

Alle Felder bei denen kein Default Wert, oder zulässige Null-Werte konfiguriert sind.

Beispiel: Hier bekommt das Feld *hexwert* einen Standardwert und das Feld *date_time* ein Häkchen bei *Null.*

Siehe NULL Werte 7.4



6.2 UPDATE – um Ändern von Datensätzen

Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Nach dem Schlüsselwort UPDATE kommt der Tabellenname, nach dem Schlüsselwort SET kommt der neue Wert.

Beispiel 1

```
UPDATE kontakte SET nachname = 'Meier' WHERE id = 5
```

Beispiel 2 (mehrere Felder)

```
UPDATE kontakte
SET nachname = 'Dite',
    email = 'viktor-dite@musterdomain.de'
WHERE id = 6;
```

Beispiel 3

Hier wird keine WHERE-Klausel verwendet, sondern der neue Wert #XXXXXX für alle Datensätze gesetzt

```
UPDATE `farbe` SET `hexwert`='#XXXXXX';
```

Bei UPDATE können genau wie bei SELECT auch Funktionen bei SET verwendet werden.

Beispiel: Hier werden alle Leerzeichen aus dem Feld *hexwert* entfernt:

```
UPDATE `farbe` SET `hexwert`= REPLACE(hexwert,' ','');
```

6.3 Daten löschen

Syntax:

DELETE FROM Tabelle **WHERE** Bedingung

Beispiel 1 (Löscht alle Daten aus allen Feldern)

```
DELETE FROM `farbe`;
```

Beispiel 2(Löscht alle Datensätzen, die nur einen Leerstring bei name haben)

```
DELETE FROM `farbe` WHERE name='';
```

Beispiel 3

```
DELETE FROM `farbe` WHERE id=1;
```

7 Verwalten von Tabellen

7.1 Anlegen von Tabellen

https://dev.mysql.com/doc/refman/8.0/en/create-table.html

Beim Anlegen von Tabellen werden Tabellenname, Feldnamen, Datentypen und weitere Eigenschaften für Felder festgelegt. Die Schlüsselwörter sind *CREATE TABLE.*

Syntax

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

Beispiel 1

CREATE TABLE `baby_names` (

`id` bigint(20) UNSIGNED NOT NULL,

`name` varchar(127) NOT NULL,

`year` int(11) NOT NULL,

`gender` varchar(10) NOT NULL,

`count` bigint(20) NOT NULL

)

Beispiel 2

CREATE TABLE IF NOT EXISTS tasks (

task_id INT AUTO_INCREMENT PRIMARY KEY,

title VARCHAR(255) NOT NULL,

start_date DATE,

due_date DATE,

status TINYINT NOT NULL,

priority TINYINT NOT NULL,

description TEXT,

created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP

) ;

Beispiel 3

```
CREATE TABLE persons (
    id int NOT NULL,
    lastname varchar(255) NOT NULL,
    firstname varchar(255),
    age int,
    PRIMARY KEY (id)
);
```

7.1.1 Änderungen an der Tabellenstruktur mit ALTER TABLE

Mit ALTER TABLE kann die Tabellenstruktur im Nachhinein geändert werden.

Beispiel: Hier wird eine Spalte *email* hinzugefügt.

```
ALTER TABLE customers
ADD email varchar(255);
```

Hier wird die Spalte *email* entfernt:

```
ALTER TABLE customers
DROP COLUMN email;
```

Hier wird der Datentyp auf year (zum Beispiel von date) geändert:

```
ALTER TABLE persons
MODIFY COLUMN DateOfBirth year;
```

7.2 Datentypen

Ganzzahlen

Beim Erstellen einer Tabelle kann man mit der Option UNSIGNED dafür sorgen, dass eine Spalte vom Typ Zahl vorzeichenlos sein soll. Dann wäre also der kleinstmögliche Wert eine 0.

Typ	vorzeichenbehaftet		vorzeichenlos	
	Minimalwert	Maximalwert	Minimalwert	Maximalwert
TINYINT	-128	+127	0	255
SMALLINT	-32.768	+32.767	0	65.535
MEDIUMINT	-8.388.608	+8.388.607	0	16.777.215
INT/INTEGER	-2.147.483.647	+2.147.483.646	0	4.294.967.295
BIGINT (*)	-2^{63}	$+2^{63} - 1$	0	$2^{64} - 1$

Fließkommazahlen

Typ	vorzeichenbehaftet		vorzeichenlos	
	Minimalwert	Maximalwert	Minimalwert	Maximalwert
FLOAT	$-3{,}402823466\times10^{38}$	$-1{,}175494351\times10^{-38}$	$1{,}175494351\times10^{-38}$	$3{,}402823466\times10^{38}$
DOUBLE	$-1{,}798\times10^{308}$	$-2{,}225\times10^{-308}$	$2{,}225\times10^{-308}$	$1{,}798\times10^{308}$

https://www.schmager.de/mysql.php

Datumsangaben

Typ	Format	Beispiel
DATE	YYYY-MM-DD	2008-07-10
TIME	hh:mm:ss	12:35:17
DATETIME/TIMESTAMP	YYYY-MM-DD hh:mm:ss	2008-07-10 12:35:17
YEAR	YYYY	1966

Zeichenketten

Typ	Wertebereich	Anmerkung
CHAR	0 - 255 Zeichen	feste Länge *
VARCHAR	0 - 255 Zeichen (**)	variable Länge
TINYTEXT	0 - 255	feste Länge *
TEXT	0 - 65535	feste Länge *
MEDIUMTEXT	0 - 16.777.215	feste Länge *
LONGTEXT	0 - 4.294.967.295	feste Länge *

Binärdaten

Typ	Wertebereich	Anmerkung
TINYBLOB	bis 2^8 Byte	feste Länge *
BLOB	bis 2^{16} Byte	feste Länge *
MEDIUMBLOB	bis 2^{24} Byte	feste Länge *
LONGBLOB	bis 2^{32} Byte	feste Länge *

7.3 Besondere Datentypen

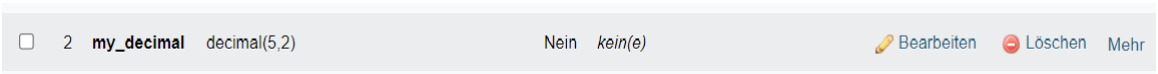
7.3.1 DECIMAL

Bei der Arbeit mit Fließkommazahlen gibt es häufiger Probleme mit Rundungsfehlern.

Neben den Fließkommazahlen gibt es in MySQL den Typ DECIMAL, der sich für feste Nachkommastellen gut eignet.

DECIMAL hat die Möglichkeit feste Nachkommastellen zu definieren.

Einstellung im *PHPMyAdamin*:

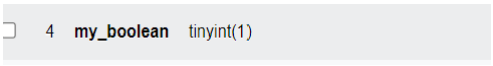


Hier hat das Feld my_decimal 5 Vorkomma -und 2 Nachkommastellen bekommen.

7.3.2 BOOLEAN

Boolean steht üblicherweise für einen Wahrheitswert, der true oder false ergibt. In MySQL

Wird dieser Typ als Zahl (tinyint) 1/0 abgespeichert.



7.3.3 BLOB

Binary Large Object (BLOB) dient zum Speichern von binären Dateien, wie z.B. Bilder oder Videos.

Häufig werden aber auch noch Pfade oder Namen als **varchar** gespeichert.

7.4 Der Wert NULL

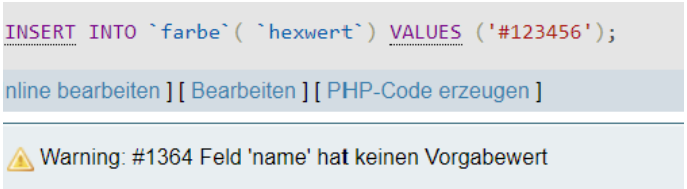
NULL-Werte sind nicht definierte/ nicht belegte Werte. Für können für jede Spalte festlegen, ob NULL-Werte zulässig sind oder nicht. Häufig werden NULL-Werte zur Initialisierung als Standardwert verwendet. Ein Primärschlüssel darf nie NULL sein!

NULL-Werte spielen bei INSERT eine Rolle.
Wenn wir für eine Spalte keine NULL-Werte erlauben (kein Häkchen bei NULL) und auch keinen anderen Defaultwert definiert haben, bekommen wir zumindest eine Warnung, falls wir dies Felder beim INSERT nicht berücksichtigen.

Beispiel:

Name	Typ	Kollation	Attribute	Null
id	int(11)			Nein
name	varchar(20)	utf8_german2_ci		Nein
hexwert	varchar(10)	utf8_german2_ci		Nein

Wir fügen nur bei *hexwert* Daten ein.



Wir erhalten eine Warnung, dass für *name* kein Standardwert gesetzt ist. Wir sollten uns also Standardwerte für alle Felder überlegen bzw. NULL als Standardwert einstellen.

7.5 Defaultwerte

Neben NULL können wir auch selbst definierte Werte als Standartwerte angeben. Im folgenden Beispiel setzen für das Feld my_int

```
ALTER TABLE `test_table`
ADD `my_int` INT NOT NULL DEFAULT '0' AFTER `my_text`;
```

7.6 Der Primärschlüssel

Als Primärschlüssel wird häufig eine Id verwendet. Ein Primärschlüssel darf nicht NULL und muss eindeutig sein. Häufig wird die Id auf AUTO INCREMENT gesetzt, das ist aber nicht in jedem Fall notwendig. Sobald wir in MySQL ein Feld als AUTO_INCREMENT einstellen, wird dieses Feld automatisch zum Primärschlüssel.

1	**id** 🔑	int(11)		Nein	*kein(e)*	AUTO_INCREMENT
2	**name**	varchar(20)	utf8_german2_ci	Nein	*kein(e)*	
3	**hexwert**	varchar(10)	utf8_german2_ci	Nein	*kein(e)*	

7.7 Tabellen löschen

Eine komplette Tabelle wird mit dem Befehl DROP TABLE <tabellenname> gelöscht.

Beispiel:

DROP TABLE *farben*

7.8 Berechnete Felder (mit Funktionen)

Wir können in SELECT-Abfragen den ursprünglichen Datentyp über Funktionen für die Ausgabe verändern.

Spaltenname	#	Data Type	Not Null	Auto Increment	Key	Default
id	1	int(11)	[v]	[v]	PRI	
date	2	date	[ ]	[ ]		curdate()
timestamp	3	timestamp	[ ]	[ ]		current_timestamp()

```
select DATE_FORMAT(date, '%y'), year(timestamp)
from berechnete_felder;
```

```
select DATE_FORMAT(date, '%d.%m.%Y') as Datum,
year(timestamp)  from berechnete_felder;
```

https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html

https://www.w3schools.com/sql/func_mysql_date_format.asp

8 Komplexe Abfragen

Komplexe Abfragen gehen häufig über mehrere Tabellen. In der Regel ergeben sich dafür mindestens 3 Möglichkeiten:

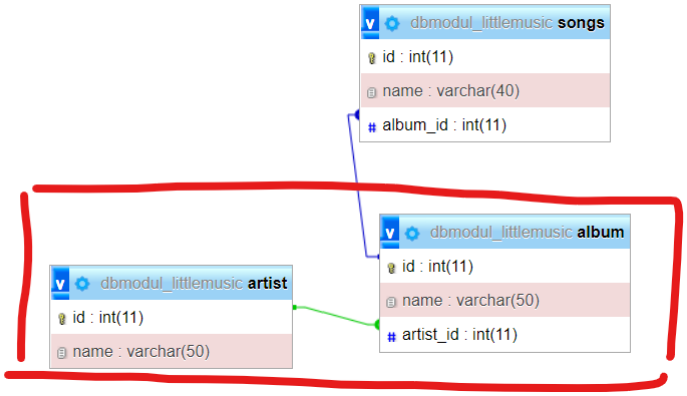
1. SUBSELECT/ Sub-Query (eine oder mehrere Tabellen)
2. JOIN (mehrere Tabellen)
3. GROUP BY (eine oder mehrere Tabellen)

8.1 Subselect/ Sub-Query

Beispiel-Datenbank: *dbmodul_littlemusic*

Subselects können sowohl in der Select-Anweisung als auch in der Where-Klausel benutzt werden.

Wir betrachten zunächst die Tabelle *artist* und *album. Die Tabelle a*lbum besitzt einen Fremdschlüssel, der angibt zu welchem Artisten das konkrete Album gehört.



Eine Abfrage zu den Namen des Albums würde so aussehen:

```
SELECT name FROM ALBUM;
```

name
Lights
Lungs
Deluxe Edition

Jetzt möchten wir aber zusätzlich die Namen der Artists erhalten.

→Subselects sollten in runden Klammern geschrieben werden.

Erster Versuch:

SELECT album.name, **(SELECT artist.name FROM artist)** FROM ALBUM;

MySQL meldet:

#1242 - Unterabfrage lieferte mehr als einen Datensatz zurück

Wir müssen bei der Abfrage die Id's (Primärschlüssel und Fremdschlüssel) berücksichtigen:

```
SELECT album.name,
(SELECT artist.name FROM artist WHERE artist.id = album.artist_id)
FROM ALBUM;
```

Das Ergebnis:

name	(SELECT artist.name FROM artist WHERE artist.id = album.artist_id)
Lights	Ellie Goulding
Lungs	Florence + The Machine
Deluxe Edition	Ed Sheeran

Die Überschrift kann jetzt noch verbessert werden:

```
SELECT album.name as Albumname,
(SELECT artist.name FROM artist WHERE artist.id = album.artist_id)
as Artistname FROM ALBUM;
```

Albumname	Artistname
Lights	Ellie Goulding
Lungs	Florence + The Machine
Deluxe Edition	Ed Sheeran

Bei Subselects darf nur ein Feld angegeben werden. Benötigen wir mehrere Felder, müssen wir mehrere Subselects verwenden.

8.1.1 Subselect nach WHERE innerhalb einer Tabelle

Syntax

```
SELECT ...

FROM   ...

WHERE  Feld Operator (SELECT ... FROM ... WHERE ...)
```

Bei Vergleichen mit „=" darf die Unterabfrage nur **einen** Wert zurückliefern!

Beispiel 1

Wir suchen den/die ältesten Kunden (es kann mehrere gleichalte Kunden geben). Das SELECT wird diesmal nur auf eine Tabelle angewendet.

```
SELECT age, firstname, lastname FROM `customers`
WHERE age = (SELECT MAX(age) FROM customers );
```

Wir vergleichen das Alter mit einem Subselect nach der Where Klausel. Dabei wird zuerst das Subselect ausgewertet.

	123 age	ABC firstname	ABC lastname
1	80	Olga	Brand
2	80	Irina	Gebhardt

Die Aufgabe könnte man auch mit Sortierung lösen. Das wäre jedoch für die Datenbank aufwendiger.

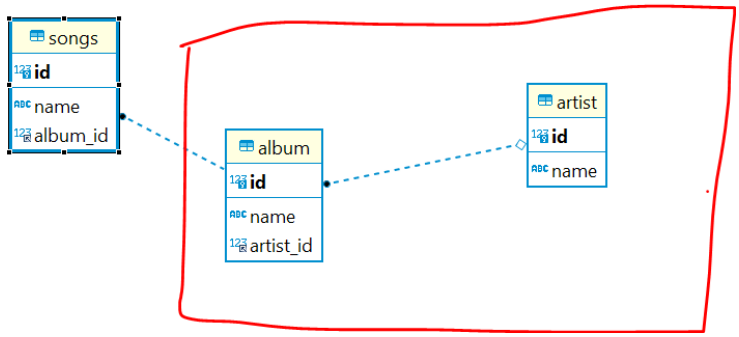
Beispiel 2

Wir wollen alle Kunden ermitteln, die genau eine Bestellung haben.
(Die Tabellennamen erhalten in diesem Beispiel einen Alias *customers c*)

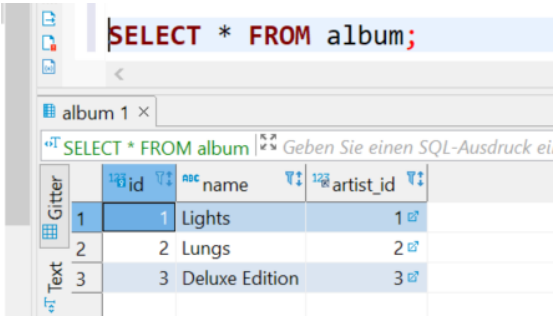
```
SELECT * FROM customers c WHERE 1 = total_amount_view
```

8.2 SELECT über mehrere Tabellen mit Join

Ressource: dbmodul_littlemusic



Ein einfaches Select auf eine Tabelle mit Fremdschlüssel gibt das folgende Ergebnis:



Wir möchten nun aber an Stelle der artist_id, die Daten der anderen Tabelle abfragen.

Das geschieht mit dem Befehl *join (Inner)*:

```
SELECT *
    FROM album
    JOIN artist ON (artist.id = album.artist_id);*
```

album(+) 1 ×

SELECT * FROM album JOIN artist ON (artist.id = Geben Sie einen SQL-Ausdruck ein, um die Ergebnisse

	id	name	artist_id	id	name
1	1	Lights	1	1	Ellie Goulding
2	2	Lungs	2	2	Florence + The Machine
3	3	Deluxe Edition	3	3	Ed Sheeran

Im MySQL ist *join* default für ***Inner Join.***

Nun wollen wir aber die zusätzlichen Fremdschlüssel nicht ausgeben und passen den Befehl etwas an. Das Schlüsselwort *as* hilft zusätzlich:

```
SELECT album.id AS 'Album ID',
    album.name AS 'Album Name',
    artist.name AS 'Artist Name'
    FROM album
    JOIN artist ON (artist.id = album.artist_id);
```

um(+) 1 ×

LECT album.id AS 'Album ID', album.name A Geben Sie einen SQL-Ausdruck ein, um die Erge

Album ID	Album Name	Artist Name
1	Lights	Ellie Goulding
2	Lungs	Florence + The Machine
3	Deluxe Edition	Ed Sheeran

SQL mit Tabellen Alias:

```
select  a1.id,  a1.name, a2.name
from album a1
join artist  a2 on (a1.artist_id = a2.id  );
```

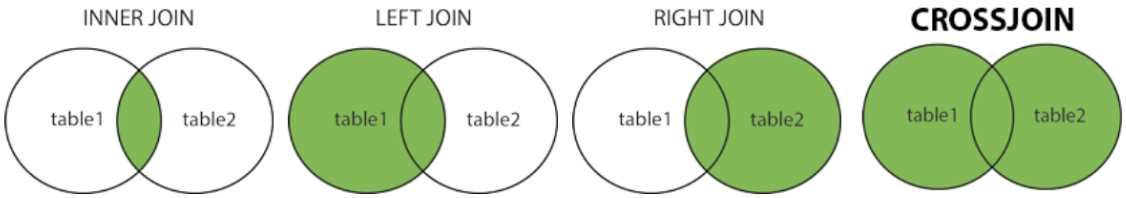
8.2.1 Verschiedene Join-Typen

INNER JOIN (Default in MySQL) und LEFT JOIN werden in den meisten Fällen genügen.

CROSS JOIN ist im Vergleich zu anderen JOIN-Varianten sehr ineffektiv, da erst alle Datensätze geholt werden und dann gefiltert werden müssen. In der Regel können alle Aufgaben über INNER JOIN und LEFT JOIN gelöst werden.

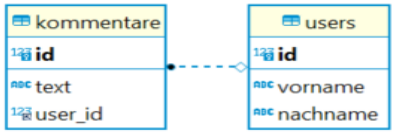
Supported Types of Joins in MySQL

- `INNER JOIN`: Returns records that have matching values in both tables
- `LEFT JOIN`: Returns all records from the left table, and the matched records from the right table
- `RIGHT JOIN`: Returns all records from the right table, and the matched records from the left table
- `CROSS JOIN`: Returns all records from both tables

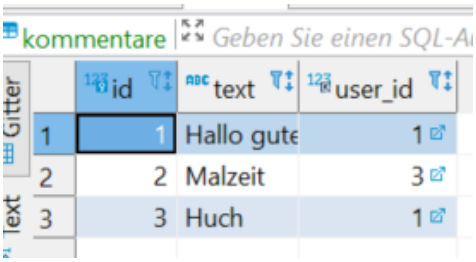


8.2.2 INNER JOIN(JOIN)

Tabellen:



kommentare



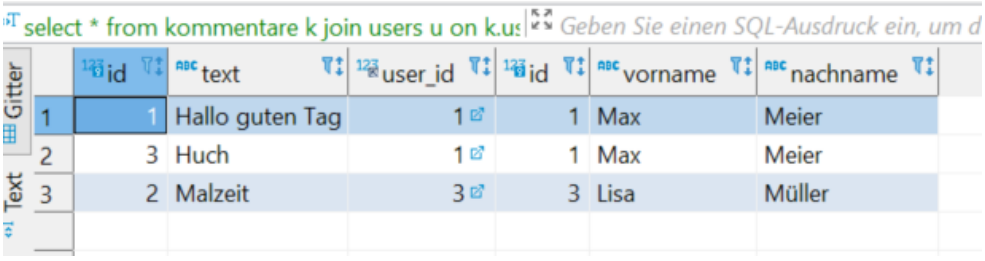
users

8.2.3 INNER JOIN

Join (Inner Join) liefert immer die Schnittmenge zweier Tabellen. Das heißt

```
select * from kommentare k
```

```
join users u on k.user_id  = u.id;
```

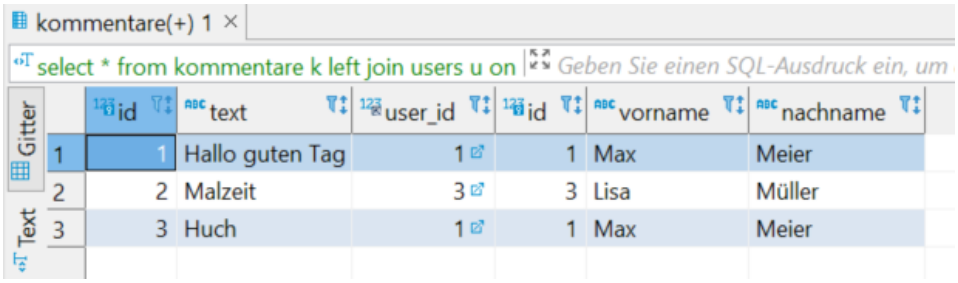


	id	text	user_id	id	vorname	nachname
1	1	Hallo guten Tag	1	1	Max	Meier
2	3	Huch	1	1	Max	Meier
3	2	Malzeit	3	3	Lisa	Müller

INNER JOIN liefert die Schnittmenge beider Tabellen

8.2.4 LEFT JOIN

```
select * from kommentare k
left join users u on k.user_id  = u.id;
```



	id	text	user_id	id	vorname	nachname
1	1	Hallo guten Tag	1	1	Max	Meier
2	2	Malzeit	3	3	Lisa	Müller
3	3	Huch	1	1	Max	Meier

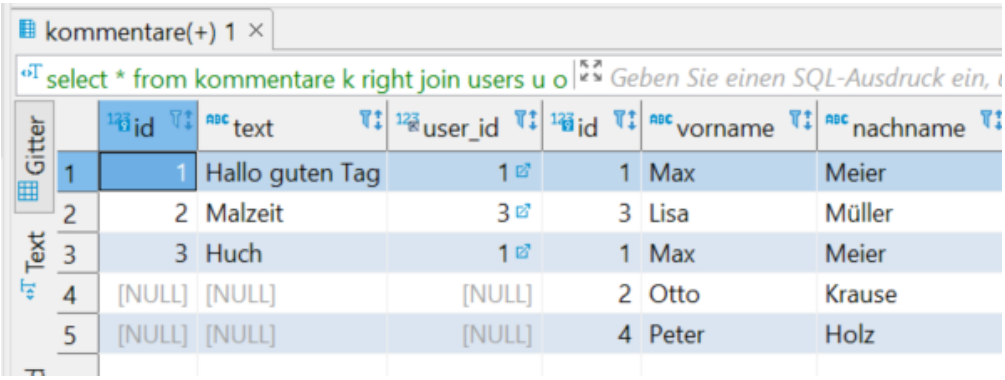
Left Join liefert hier die gleiche Menge (anderer Reihenfolge, da „Linke“ FROM-Tabelle *kommentare* - zuerst betrachtet wird).

Weitere User werden nicht ausgegeben, wenn sie keine Kommentare haben.

Falls wir aber alle User erhalten möchten, unabhängig ob sie Kommentare haben, können wir das mit right join erreichen. Hierbei wird von der rechten Tabelle (user) ausgegangen und danach Kommentare oder null-Werte zurückgegeben.

8.2.5 RIGHT JOIN

```
select * from kommentare k
right join users u on k.user_id  = u.id;
```



	id	text	user_id	id	vorname	nachname
1	1	Hallo guten Tag	1	1	Max	Meier
2	2	Malzeit	3	3	Lisa	Müller
3	3	Huch	1	1	Max	Meier
4	[NULL]	[NULL]	[NULL]	2	Otto	Krause
5	[NULL]	[NULL]	[NULL]	4	Peter	Holz

Bei **right join** wird diesmal die Tabelle mit dem **join** (users) als Basistabelle betrachtet. Nun werden auch die User ohne Kommentare angezeigt.

In den meisten Fällen kommt man mit **left join** oder **join** (inner join) aus, da Reihenfolge der Tabellen in der Abfrage verändert werden kann.

Aufgabe: Füge in der Tabelle Kommentare in der Spalte Text ein Kommentar ein user_id.

Führe eine Abfrage mit LEFT JOIN durch und vergleich die Ausgabe.

8.3 GROUP BY

Gruppieren nach bestimmten Feldern.

Ein einfaches SELECT liefert bei der Tabelle track (musicdb) das folgende Ergebnis:

13	Angus Young, Malcolm Young, Brian Johr
14	Angus Young, Malcolm Young, Brian Johr
15	AC/DC
16	AC/DC
17	AC/DC
18	AC/DC
19	AC/DC
20	AC/DC
21	AC/DC
22	AC/DC
23	Steven Tyler, Joe Perry, Jack Blades, Tom

Nun wollen wir die Tabelle nach composer gruppieren.

```
SELECT composer FROM track group by composer ;
```

Jetzt erhalten wir im Ergebnis nur noch eindeutige Ergebnisse.#:

2	A. F. Iommi, W. Ward, T. Butler, J. Osbourne
3	A. Jamal
4	A.Bouchard/J.Bouchard/S.Pearlman
5	Aaron Copland
6	Aaron Goldberg
7	AC/DC
8	Ace Frehley
9	Acyi Marques/Arlindo Bruz/Braço, Beto Sem/Zeca Pagodinho
10	Acyr Marques/Arlindo Cruz/Franco

Beispiel 2: Die Composer und die Anzahl des jeweiligen Composers

```
SELECT composer, count(*) as `Wie oft?` FROM track
group by composer;
```

Count bezieht sich jetzt auf die Gruppe.

1	[NULL]	948
2	A. F. Iommi, W. Ward, T. Butler, J. Osbourne	3
3	A. Jamal	2
4	A.Bouchard/J.Bouchard/S.Pearlman	1
5	Aaron Copland	1
6	Aaron Goldberg	1
7	AC/DC	8
8	Ace Frehley	2
9	Acyi Marques/Arlindo Bruz/Braço, Beto Sem/Zeca Pagodinho	1
10	Acyr Marques/Arlindo Cruz/Franco	1
11	Adalto Magalha/Lourenco	1
12	Adam Clayton, Bono, Larry Mullen & The Edge	11
13	Adam Clayton, Bono, Larry Mullen, The Edge	11

GROUP BY wirkt ähnlich wie ein DISTINCT. DISTINCT kann aber nicht Funktionen auf Gruppen ausführen.

Beispiel 3:

Wir haben in der Tabelle orders die customer_id mehrfach, da einige Kunden mehrere Bestellungen haben.

	customer_id
49	[NULL]
50	[NULL]
51	[NULL]
52	11
53	12
54	14
55	14
56	14
57	15
58	16
59	16
60	17

Wir möchten jetzt die Summe von *amount* (steht für den Bestellwert) gruppiert berechnen.

Ohne GROUP BY hätten nur einen Datensatz, da sum() für alle Zeilen berechnet wird:

```
select customer_id, sum(amount)  from orders;
```

customer_id	sum(amount)
162	52.359

Mit GROUP BY

```
select customer_id, sum(amount)  from orders
group by customer_id;
```

	customer_id	sum(amount)
	[NULL]	2.762
	11	56
	12	82
	14	245
	15	86
	16	23
	17	61
	18	37
	19	248
0	20	8
1	22	98
2	23	121
3	24	147
4	25	182

Im nächsten Schritt möchten wir auf die Gruppierung eine Funktion ausführen. Das geschieht mit dem Schlüsselwort HAVING.

Es sollen die Datensätze gruppiert werden, die einen Bestellwert ab 400 haben.

```
select customer_id, sum(amount)
from orders
group by customer_id
having sum(amount) >=400
```

Beispiel 4: GROUP BY mit JOIN

Wir ergänzen unser Beispiel einfach um einen JOIN (INNER JOIN)

```
-- join
select customers.lastname , sum(amount)
from orders
join customers on customers.id = orders.customer_id
```

```
group by customer_id
order by sum(amount)
```

9 Beziehungen zwischen Tabellen

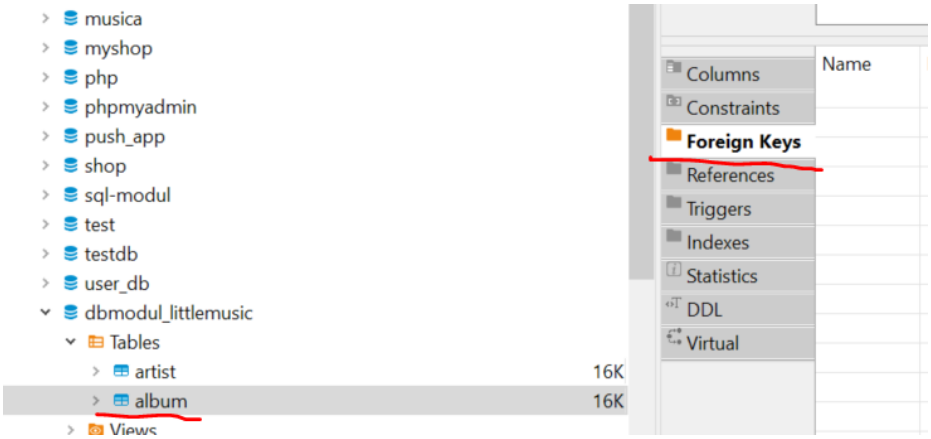
9.1 Primär - und Fremdschlüssel

9.1.1 Schlüssel einrichten mit DBEaver

Links die entsprechende Tabelle wählen. Rechts unter Foriegn Keys können mit der rechten Maustaste Fremdschlüssel hinzugefügt werden.

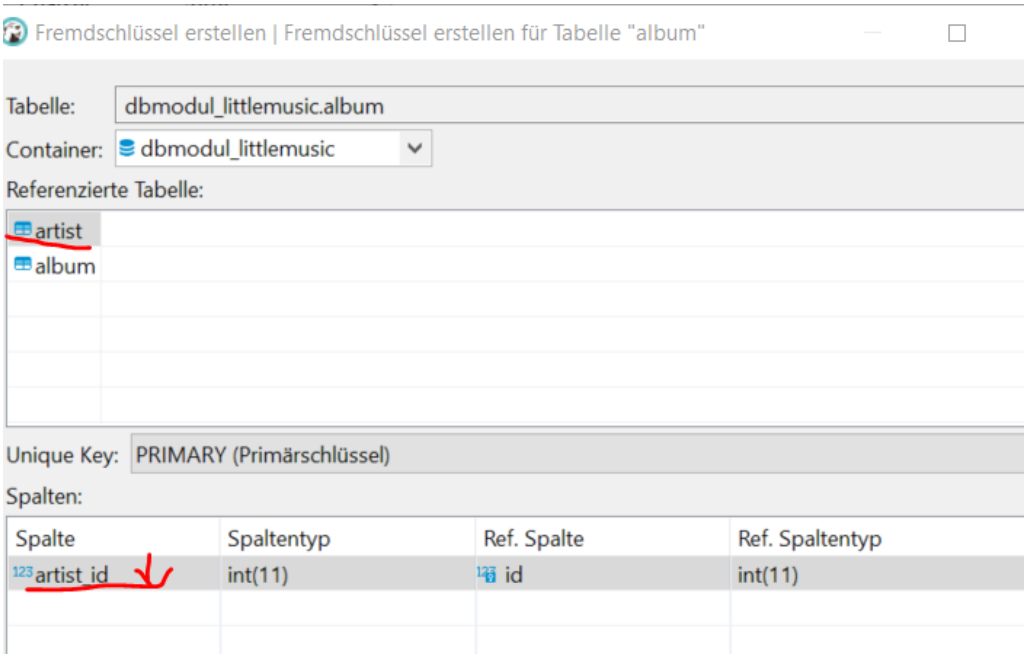
Das Ergebnis soll dieses sein:

Zunächst wählen wir links im Navigator die Tabelle aus, die den Fremdschlüssel besitzen soll. In diesem Beispiel also *album, die Basistabelle*.

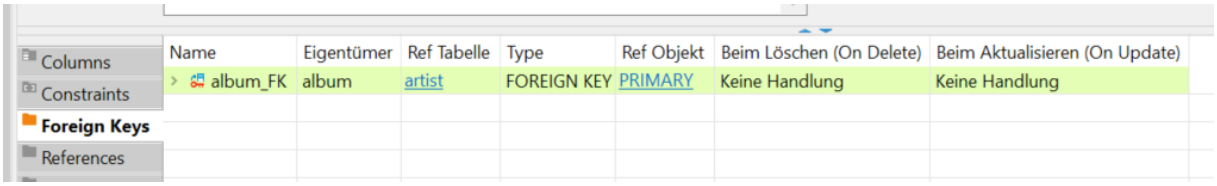


In diesem Beispiel ist artist die Referenzierte Tabelle. Diese wird jetzt oben angewählt.

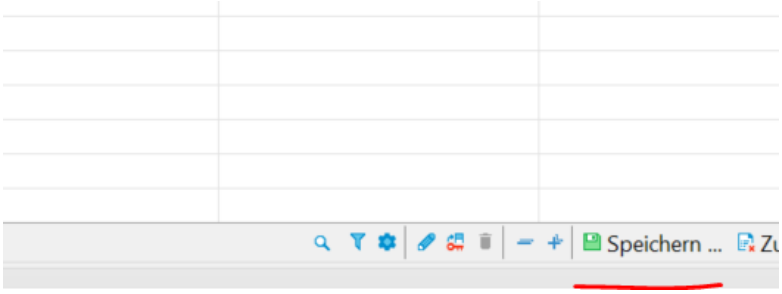
Unten wird jetzt die Fremdschlüsselspalte der Basistabelle *album* ausgewählt und bestätigt.



Das Ergebnis:



Auch diese Aktion muss wie jede andere auch, gespeichert werden:



Zur Kontrolle kann man sich das ER Diagramm anschauen, in dem man die Linie markiert.

9.2 Shared Primary Key

Bei einer 1 zu 1 (One to One) Beziehung ist es möglich auf einen expliziten Fremdschlüssel zu verzichten. Dazu verweist die zweite Tabelle mit ihrem Primärschlüssel auf den Primärschlüssel der ersten Tabelle.

Beispiel

Bei dieser Vorgehensweise muss man sicherstellen, dass jederzeit beide verknüpfte IDs übereinstimmen. Dazu können Foreign Key Constraints gesetzt werden.

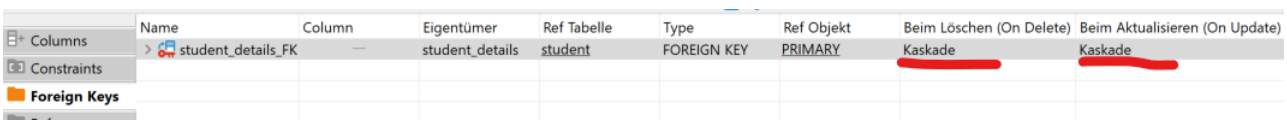
Siehe Foreign Key Constrains 9.3

9.3 Foreign Key Constraints

Wir haben in MySQL die Möglichkeit Foreign Key Constraints zu setzen. Damit sagen wir, was mit den Daten passieren soll, wenn in der „Haupttabelle“ der Datensatz gelöscht oder verändert wurde.

In diesem Beispiel sagen wir, dass der Datensatz der Detailtabelle auch gelöscht werden soll falls der Haupt-Datensatz gelöscht wurde. Das geschieht mit der Einstellung Kaskade (Cascade).

Genau wie beim OnDelete kann man beim OnUpdate Kaskade einstellen. Damit wird bewirkt, dass zum Beispiel Schlüssel-Änderungen übertragen werden.



10 Views

Views werden in der Regel aus Abfragen generiert, die wir häufiger benötigen. Eine erstellte View kann dann wie eine Tabelle verwendet und abgefragt werden.

Beispiel

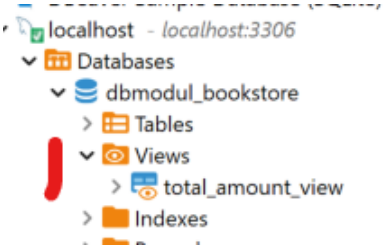
Wir gehen von diesem komplexerem Statement aus

```
select customers.lastname , sum(amount)  as `Total amount`
from orders
join customers on customers.id = orders.customer_id
group by customer_id
order by sum(amount)
```

Wir setzen jetzt ein View Statement vor dem SELECT:

```
create view total_amount_view as
select customers.lastname , sum(amount)  as `Total amount`
from orders
join customers on customers.id = orders.customer_id
group by customer_id
order by sum(amount)
```

Die View erscheint jetzt unter Views und kann genauso „normale“ Tabellen für Abfragen verwendet werden.



11 Datenbank modellieren

11.1 Normalformen

Normalisierung soll helfen, um zu möglichst redundanzfreien Tabellen zu kommen. Dafür existieren Normalformen

Die 1. Normalform

Ein Relationstyp (Tabelle) befindet sich in der ersten Normalform (1NF), wenn die Wertebereiche der Attribute des Relationstypen atomar sind und keine Wiederholungsgruppen haben.

student

Name	**Eingeschrieben**	**Kurs**
Max Meier	1.3.2020	Informatik 1, BWL, Mathe 1
Ina Schultz	1.9.2020	Mathe 1, Informatik 1

Wir verändern die Tabelle entsprechend der 1. Normalform

Vorname	Nachname	**Eingeschrieben**	**Kurs**
Max	Meier	1.3.2020	BWL
Max	Meier	1.3.2020	Mathe 1
Max	Meier	1.3.2020	Informatik 1
Ina	Schultz	1.9.2020	Mathe 1
Ina	Schultz	1.9.2020	Informatik 1

Die erste Normalform ist erfüllt. Die Tabelle zeugt aber noch schwächen.

Die 2. Normalform

Die zweite Normalform ist erfüllt, wenn alle Felder, die nicht Primärschlüssel-Kandidat sind, vom gesamten Primärschlüssel-Kandidaten abhängig sind.

Wir fügen eine S_ID (Primärschlüssel-Kandidat) für die Studenten ein.

s_id	Vorname	Nachname	**Eingeschrieben**	**Kurs**
1	Max	Meier	1.3.2020	BWL
1	Max	Meier	1.3.2020	Mathe 1
1	Max	Meier	1.3.2020	Informatik 1
2	Ina	Schultz	1.9.2020	Mathe 1
2	Ina	Schultz	1.9.2020	Informatik 1

Die Spalte Fächer ist nicht vom Schlüsselkandidaten (s_id) abhängig. Um zu 2. Normalform zu kommen, lagern wir diese Spalte in einer Tabelle aus.

kurs

id	**Fächer**
1	BWL
2	Mathe 1
3	Informatik 1

Die Originaltabelle kann jetzt reduziert werden

s_id	Vorname	Nachname	**Eingeschrieben**
1	Max	Meier	1.3.2020
2	Ina	Schultz	1.9.2020

Jetzt müssen wir noch die Verbindung der Schlüssel mit ein Zwischentabelle realisieren. Das ist immer der Fall, wenn wir eine N:M Beziehung haben

Id_student	kurs_id
1	1
1	2
1	3
2	2
2	3

11.2 Mit ER-Diagramm modellieren

https://app.diagrams.net/

12 Tools

- Testdaten generieren

https://migano.de/testdaten.php

https://www.mockaroo.com/

13 Extra

- MySQL Installation unter Linux

14 Anhang

14.1 MySQL Referenz

https://dev.mysql.com/doc/index-other.html

14.2 TOP-60 WICHTIGSTEN SQL ABFRAGEN

https://bytescout.com/blog/top-wichtigsten-sql-abfragen.html

14.3 Umgang mit Fehlern

14.3.1 INSERT mit Fremdschlüsseln

Bei mehreren Tabellen und Insert bzw. UPDATE kann folgende Fehlermeldung erscheinen:

Cannot add or update a child row: a foreign key constraint fails

bedeutet im Wesentlichen, dass wir versuchen eine Zeile hinzuzufügen, für die keine übereinstimmender Eintrag in der zweiten Tabelle vorhanden ist. Es kann zum Beispiel seien, dass man SQL Skripte in falscher Reihenfolge ausführt.
Man kann die Prüfung abstellen mit:

```
SET FOREIGN_KEY_CHECKS=0;
```

oder global

```
SET GLOBAL FOREIGN_KEY_CHECKS=0;
```