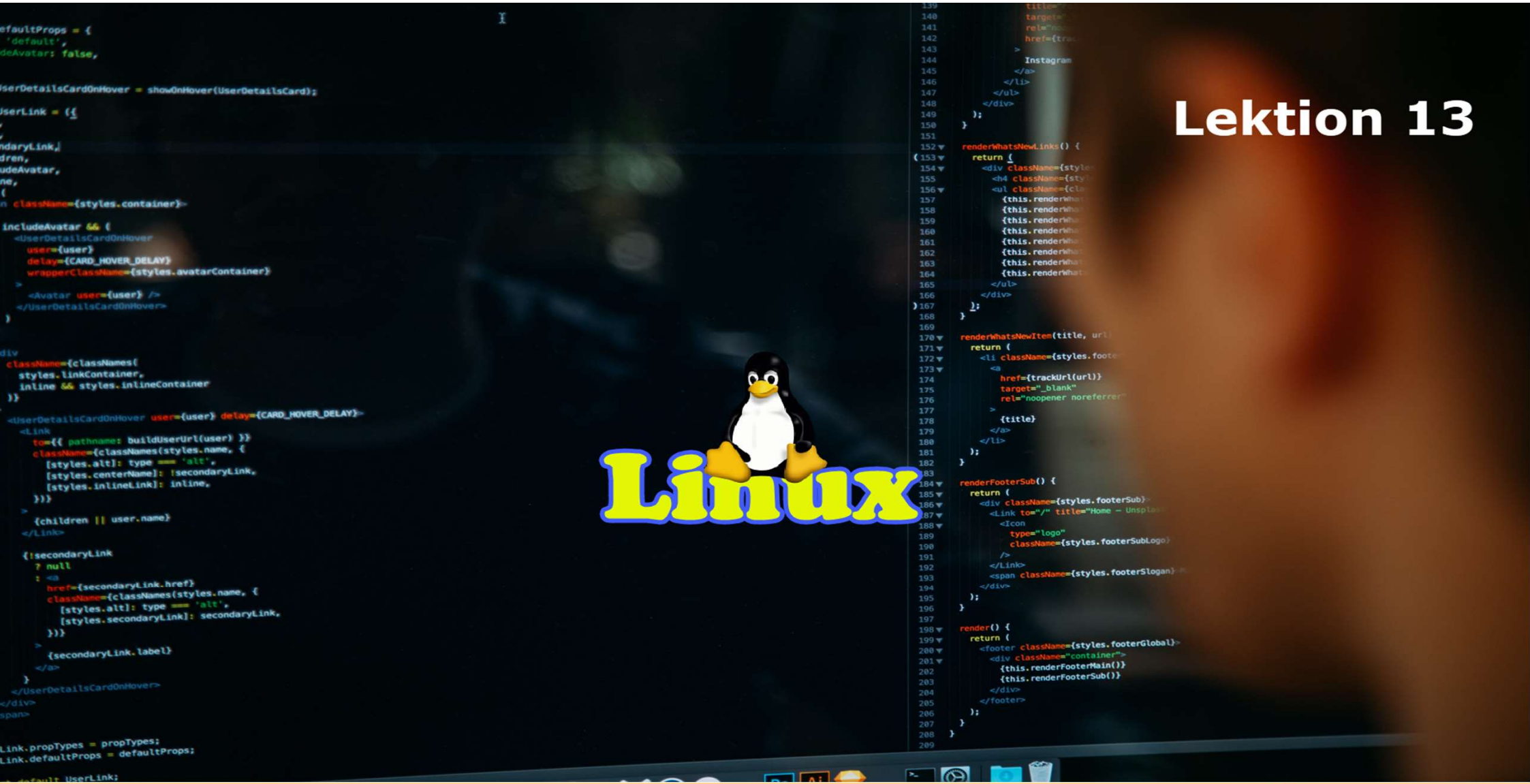

Lektion 13
Linux

Grundlegende Bemerkungen

- Datenbanken gehören mit zu den ersten Anwendungen die auf Computern liefen
- Was bedeutet der Begriff DATENBANK?
- Einfach ausgedrückt handelt es sich um eine strukturierte Ablage von Daten
- Die häufigste Ablageform ist die Nutzung mehrerer Tabellen
- Diese können in Beziehung zueinander gesetzt werden
- Natürlich könnte man auch eine einzige Tabelle aller Daten erstellen
- Doch darin zeigen sich gravierende Nachteile:
  - Die Tabelle wird sehr groß und unübersichtlich
  - Es entstehen pausenlos Redundanzen (mehrfach gleiche Daten, die jedesmal eingegeben werden müssen)

MySQL oder MariaDB

- MySQL war ursprünglich ein Open-Source Projekt
- Dieses mächtige Datenbank-System wurde 2008 von Sun Microsystems übernommen
- Sun Microsystems wiederum wurde später von Oracle aufgekauft
- Dadurch ist aus einem Open-Source-Projekt eine proprietäre Software geworden, die aus ihren Quellen heraus aber noch Open-Source Lizenzen besitzt
- So entstand aus den Quellen ein Fork (Fork = Gabel) von MySQL, mit dem Namen MariaDB der den exakt gleichen Befehlssatz nutzt und von uns bedenkenlos eingesetzt werden kann
- MariaDB wird in Wordpress und anderen CMS für dynamische Websites eingesetzt

Inhalte der Lektion

- In dieser Lektion werden Sie folgendes kennenlernen:
  - Die Grundlagen von relationalen Datenbanksystemen
  - Die Installation von MariaDB auf unserem Linux-Server deb-s1
  - Die Einrichtung von Datenbanken und deren Tabellen
  - Die Konfiguration, um auf Datenbanken zuzugreifen
  - Die Generierung von einfachen und komplexen Abfragen
- Natürlich gibt es grafische Verwaltungssysteme zur Nutzung von Datenbanken oder zumindest den komfortablen Einsatz von PHP
- Hier wollen wir aber zunächst über die grundlegenden Kommandos den internen Kontext von Datenbanken kennenlernen, was anfangs umständlich erscheint, aber das interne Verständnis fördert

Relationen

- MariaDB (auch MySQL oder MS-Access) wird als Relationales Datenbank-Management-System (RDBMS) bezeichnet
- Die Tabellen einer Datenbank stehen also in Beziehung(en) zueinander, in Relation(en)
- Eigentlich zusammengehörige Informationen werden in verschiedene Tabellen gebracht und können durch diese Beziehungen zusammengebracht werden
- Somit vermeidet man Redundanzen weitgehend (Die Tabellen bleiben schlank)
  - So werden z.B. Kunden, Mitarbeiter, Projekte und Adressen in verschiedenen Tabellen gepflegt
  - Durch eine tabellenübergreifende Abfrage mit SQL (Structured Query Language) lässt sich aber leicht ermitteln, welcher Mitarbeiter im Jahr 2023 das Projekt, eines bestimmten Kunden, betreut hat

Inhalte einer Datenbank

- Das Ziel der Datenbank bestimmt über deren Inhalte
- Es soll also eine möglichst genaue, aussagefähige Abbildung einer realen Situation geschaffen werden
- Dafür schafft man sich Objekte, die als *Entitätstypen* bezeichnet werden
- Jedes Objekt hat unterschiedliche Eigenschaften, die man *Attribute* nennt
  - Es gibt relevante und nicht-relevante Attribute für den jeweiligen Zweck des Objekts:
  - So enthält eine Mitarbeitertabelle andere Inhalte als eine Tabelle einer Dating-Plattform
  - In der ersten ist die Haarfarbe nicht relevant und der zweiten schon ☺

Praxisbeispiel

- Für das Beispiel versetzen wir uns in das IT-Unternehmen Alpha GmbH
- Die Alpha GmbH bietet u. a. für seine Kunden das Erstellen und die Wartung von Netzwerkstrukturen an
- Somit sind automatisch 3 Entitätstypen identifiziert, nämlich Mitarbeiter, Kunden und Projekte
- Jetzt müssen noch die relevanten Attribute der 3 Entitätstypen ermittelt werden

Mitarbeiter

- Die Mitarbeiter sollen nicht nur für den externen Zweck in Verbindung mit den Kunden, sondern auch intern verwaltet werden können (z.B. Jubiläen)
- Daraus ergeben sich die (minimalen) Attribute:
  - Name
  - Vorname
  - Adresse (mit Straße, PLZ und Ort)
  - Abteilung
  - Position
  - Einstellungsdatum

Kunden

- Die Kunden unterteilen sich in Geschäftskunden und private Kunden
- Daraus ergeben sich folgende Attribute:
  - Name (auch Firmenname)
  - Vorname (nur für Privatkunden)
  - Adresse (mit Straße, PLZ und Ort)
  - Ansprechpartner (nur für Firmen)

Projekte

- In Projekten findet man eine Vielzahl relevanter Attribute, wie Art des Projekts (Neuinstallation, Reparatur, …) oder Zeitraum für das Projekt, usw.
- Hier sollen nur die notwendigen eingetragen werden
  - Projektname
  - Adresse (mit Straße, PLZ und Ort)
  - Auftraggeber (Kunde)

Entitäten und Attribute

- Entitäten sind die individuellen, einzigartigen Ausprägungen eines Entitätstyps, also z.B. eines bestimmten Mitarbeiters, während
- die Attribute für alle Entitäten identisch sind z.B. Vorname, Name, …

Vorname	Name	Straße	PLZ	Ort
Klaus	Korn	Holzweg 7	80363	München
Sabine	Meier	Kurze Str. 23	80449	München

- Entitäten werden in den Zeilen dargestellt, die Attribute in den Spalten
- Der Entitätstyp Adresse wird in die Einzelteile zerlegt, also Straße, PLZ und Ort

Entity Relationship Model (ERM)

- Im ERM werden die Objekte (Entitytypen) zueinander in Beziehung gesetzt
- Hierbei hilft die wörtliche Formulierung, was die Objekte verbindet, z.B.
  - Ein bestimmter Mitarbeiter ist Mitarbeiter eines bestimmten Projektes eines Kunden
  - Ein bestimmter Kunde gibt ein bestimmtes Projekt in Auftrag
- Das wirkt sehr einfach. Andererseits lassen sich daraus aber auch bestimmte Rückschlüsse ziehen
  - Ein Mitarbeiter kann an mehreren Projekten beteiligt sein (wenn sein Zeitpotential es zulässt)
  - Ein Projekt kann von mehreren Mitarbeitern betreut werden (bei großen Projekten)
  - Ein Kunde kann mehrere Projekte in Auftrag geben, ein Projekt kann immer einem Kunden zugeordnet werden
- Daraus resultieren verschiedene Verbindungstypen

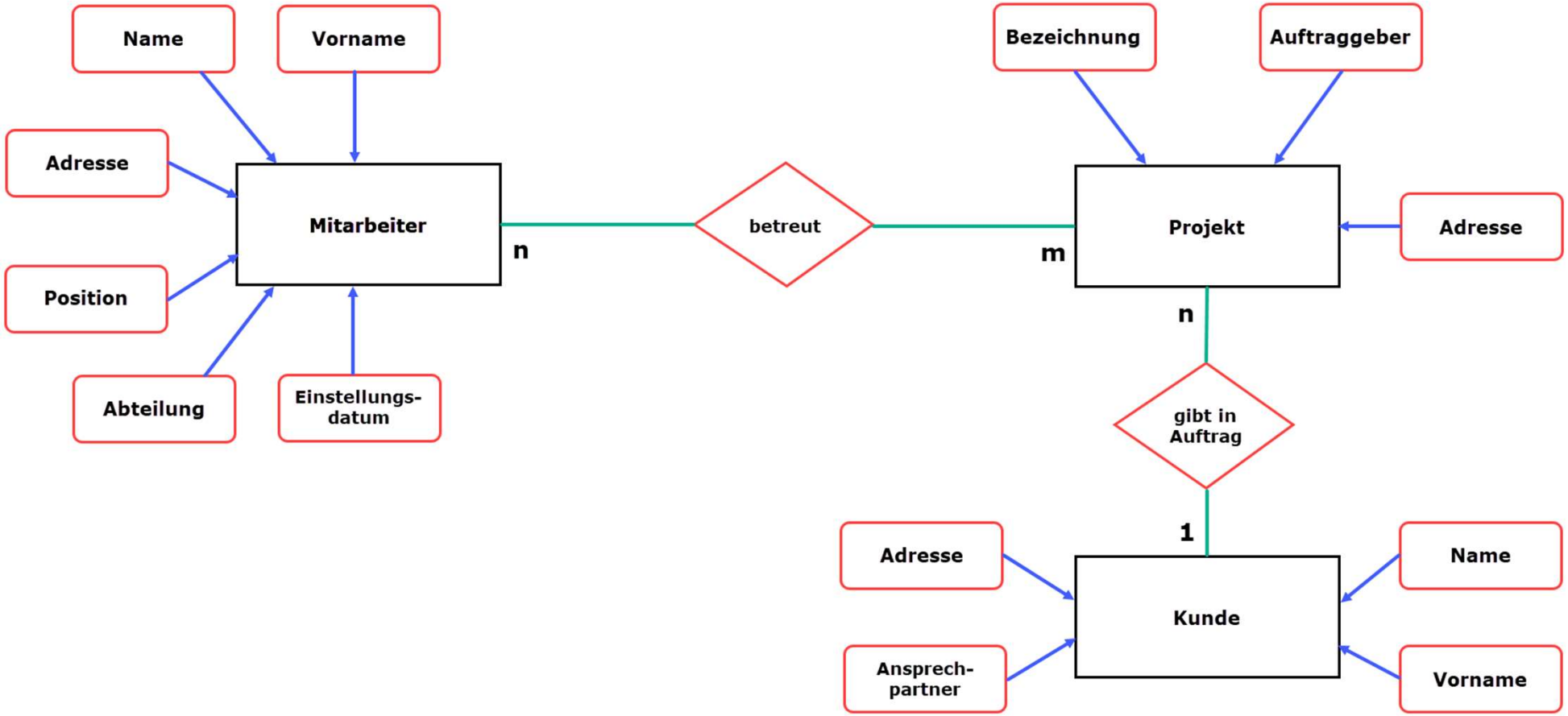
Verbindungstypen

- 1 : 1 – jeder Entität eines Entitätstyps ist genau eine Entität eines anderen Entitätstyps zugeordnet z.B.
  - Ausweis – dieser gehört immer genau einer Person
- 1 : n – jeder Entität eines Entitätstyps sind beliebig viele (auch keine!) Entitäten eines anderen Entitätstyps zugeordnet z.B.
  - Auto – eine Person kann beliebig viele Autos besitzen, aber ein Auto hat immer nur einen Eigentümer
- n : m – jeder Entität eines Entitätstyps sind beliebig viele (auch keine!) Entitäten eines anderen Entitätstyps zugeordnet. Im Gegensatz zur 1 : n-Beziehung gilt das auch in umgekehrter Richtung z.B.
  - Mitarbeiter – Projekt: Ein Mitarbeiter kann bei beliebig vielen Projekten eingebunden sein, die Projekte ihrerseits können von beleibeig vielen Mitarbeitern betreut werden

Besonderheit n : m

- Die n : m-Beziehung lässt sich nur indirekt abbilden und ist daher nicht gern gesehen. Sie kann aber auch nicht immer vermieden werden
- Gerade in größeren Projekte kann es hilfreich sein, die Beziehungen grafisch darzustellen. Wichtig ist, dass keine Entität vergessen wird
- Für die Alpha GmbH ergeben sich (grafisch gesehen) folgende Beziehungen im Entity Relationship-Model (ERM):

Unser Entity Relationship Model (ERM)



Die erste Normalform

- Wie schon gesagt, lassen sich alle Informationen in einer Tabelle unterbringen
- Dabei wären viele Informationen redundant, also mindestens doppelt vorhanden
  - Das liegt daran, dass alle Informationen, die zusammen gehören in Spalten erfasst werden müssten und
  - Alle Attribute sämtlicher Entitätstypen (Mitarbeiter, Kunden, Projekte) müssten in jeder Zeile in entsprechenden Spalten auftauchen
- Selbst in unserem einfachen Beispiel hätten wir mit den wenigen Attributen mindestens 11 Spalten
  - Arbeitet ein Mitarbeiter an zwei Projekten, würde man auch zwei Zeilen dafür benötigen, in denen bis auf die Projekte, die gleichen Informationen stehen
- Das wäre eine sehr ineffiziente Methode
  - Solche Tabellen nennt man 1. Normalform, die die Ausgangssituation gut verdeutlicht

Die dritte Normalform

- Die 2. Normalform ist eine eher theoretische Zwischenform, die man meist ignorieren kann, wenn man die Regeln der 3. Normalform kennt
- In der 3. Normalform existieren keine Redundanzen mehr
- Die Definition der 3. Normalform besagt, dass es keine Abhängigkeiten zwischen Nicht-Schlüsselattributen innerhalb einer Relation (Tabelle) gibt. Alle Attribute dürfen nur vom Primärschlüssel abhängig sein, der einen Datensatz eindeutig identifiziert.
- Harter Tobak! Es kann dauern, bis man die Logik erkennt, zumal Begriffe enthalten sind, die noch unbekannt sind

Wie erhält man die dritte Normalform?

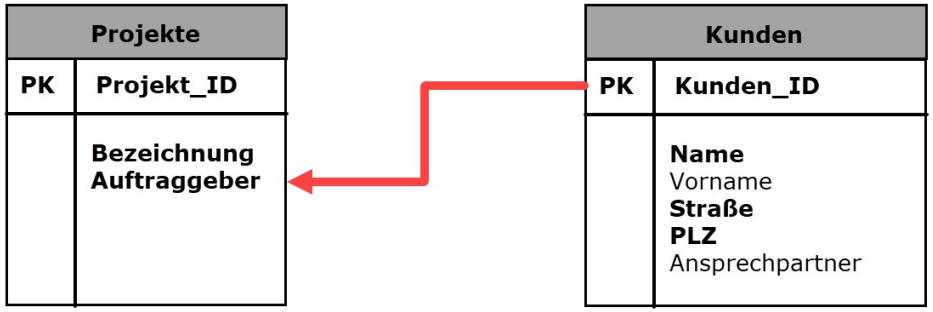
- Dazu extrahiert man alle redundanten Informationen und schreibt sie in eigene Tabellen. Hieraus entstehen die Objekttabellen (hier für die Entitätstypen Mitarbeiter, Kunden und Projekte) und ggf. Verbindungstabellen, die die Objekte zusätzlich miteinander in Beziehung setzen
  - Problem Ort:
    Der Ort ist direkt abhängig von der PLZ, das heißt, man kann ihn nicht nur vom Primärschlüssel, sondern auch direkt von der PLZ ableiten. Daraus resultieren redundante Informationen!
    Daher müssen wir, um eine korrekte 3. Normalform zu erreichen, eine weitere Objekttabelle für den Wohnort erstellen
- Bevor wir die 3. Normalform praktisch umsetzen können, müssen noch ein paar Begrifflichkeiten geklärt werden

Primärschlüssel und Fremdschlüssel

- Wie kann man einen Datensatz (entspricht einer Tabellenzeile) eindeutig identifizieren?
- Dies geschieht mit einem Primärschlüssel (Primary Key)
  - In der Praxis setzt man dazu meistens eine eindeutige Ziffer (Zahl) in die erste Tabellenspalte
  - Die Bezeichnung kann man frei wählen. Sie sollte aber ihren Zweck gut preisgeben, also ist eine Bezeichnung Mitarbeiter_ID für die Tabelle Mitarbeiter sehr sinnvoll
  - Für die Objekttabelle Wohnort ist der Primärschlüssel die PLZ, weil der Wohnort durch sie eindeutig identifiziert werden kann. Hier einen anderen Primärschlüssel einzubringen bringt keinen Nutzen
- Und wie kann man die einzelnen Tabellen in Relation bringen?
- In dem man die Primärschlüssel als Fremdschlüssel (Foreign Key) in eine andere Tabelle einfügt

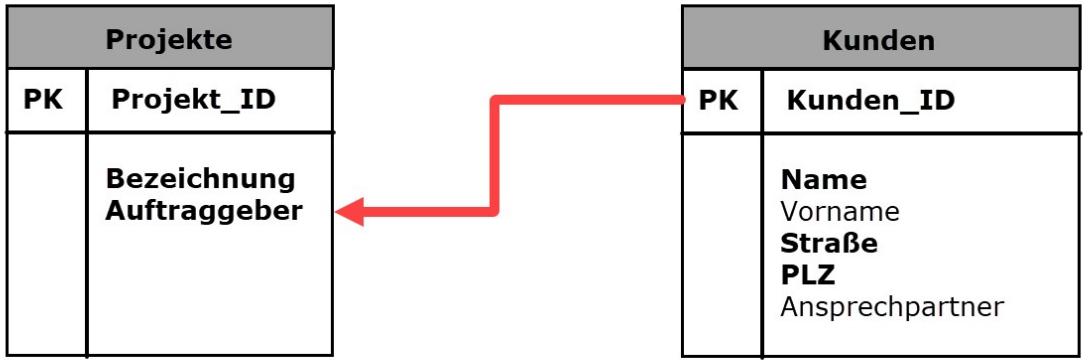
Fremdschlüssel

- Während der Primärschlüssel nur einmal in einer Tabelle vorkommen darf, kann er als Fremdschlüssel mehrfach in den Tabellen vorkommen
- So kann der Primärschlüssel Kunden_ID aus der Tabelle Kunden als Fremdschlüssel mit der Bezeichnung Auftraggeber in die Tabelle Projekte übernommen werden, wodurch sich der Auftraggeber eines Projektes eindeutig bestimmen lässt
- Das gilt auch für die PLZ als Fremd-schlüssel. Dadurch wird der Wohnort ebenfalls eindeutig identifiziert



Fremdschlüssel

- Fremdschlüssel sind bei allen 1 : 1- und 1 : n-Beziehungen sinnvoll
- Da jedes Projekt genau einem Auftraggeber zugeordnet werden kann, ist ein Fremdschlüssel in dieser Tabelle nützlich und stellt damit die Beziehung zwischen den beiden Objekten Projekt und Kunden dar

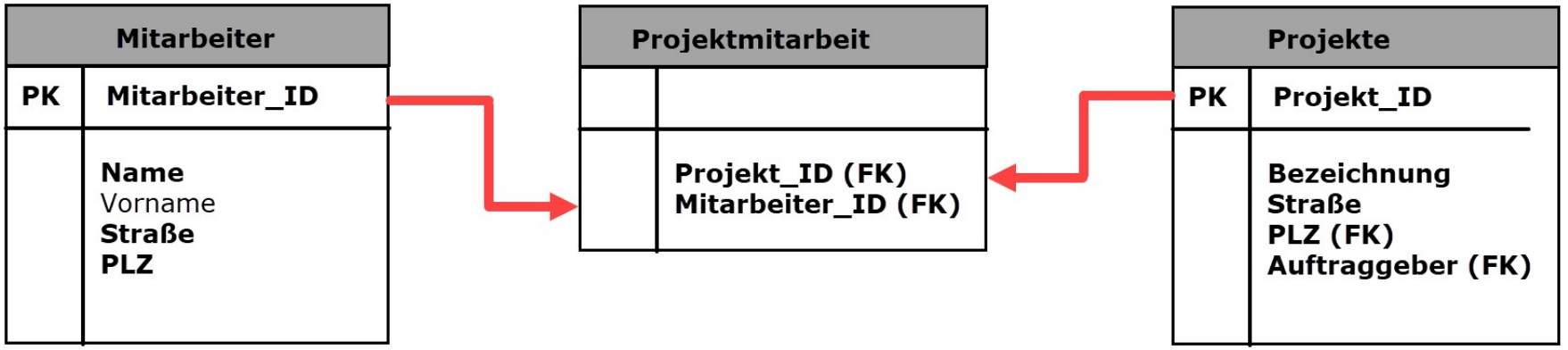


Verbindungstabellen

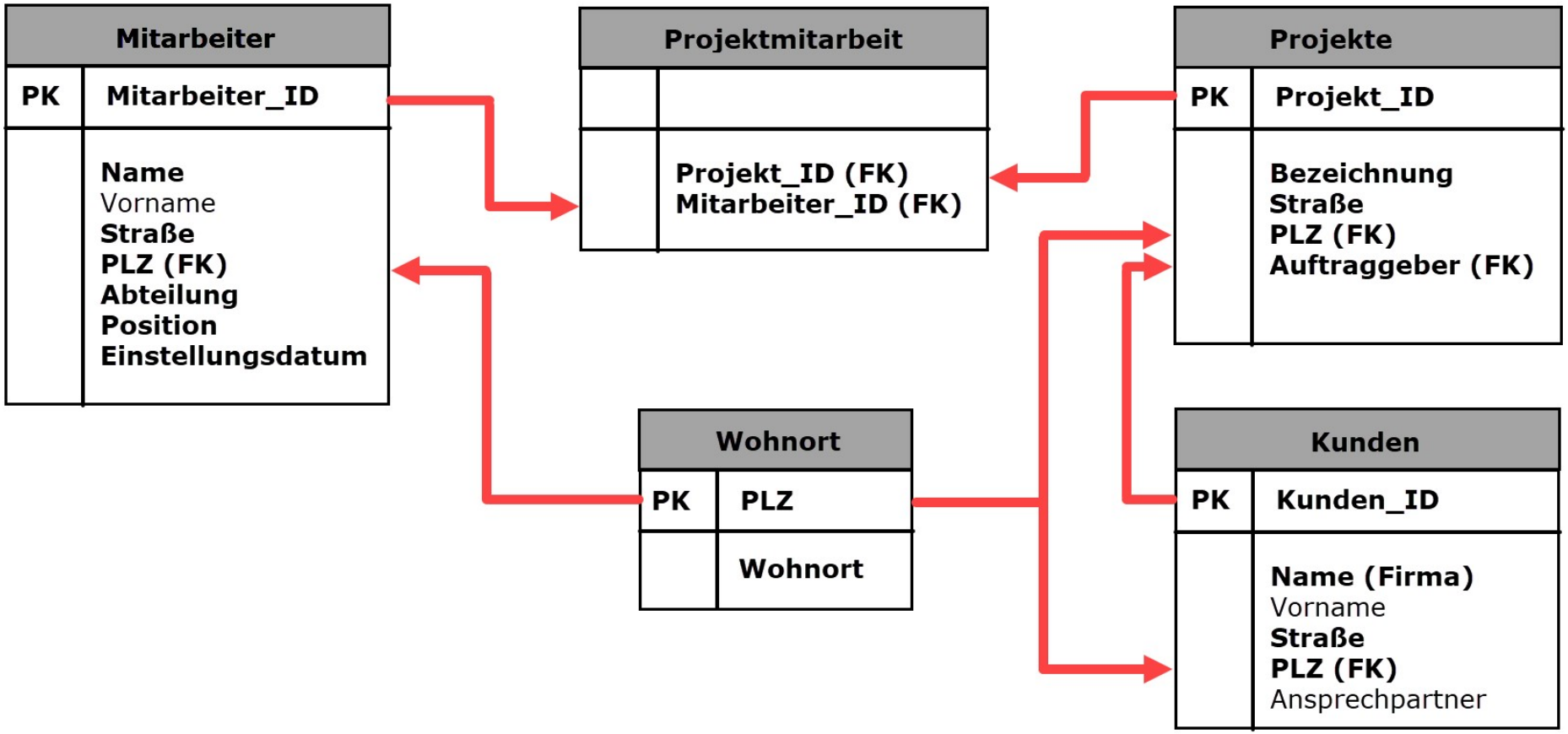
- Es gibt Fälle, in denen die Tabellen keine Aussage über die Beziehungen zwischen zwei Entitätstypen (Objekten) treffen können
- Hierzu gibt es dann die Verbindungstabellen
- Verbindungstabellen sind für n : m-Beziehungen erforderlich und bestehen typischerweise aus Fremdschlüsseln
- Sie besitzen keinen Primärschlüssel, weil dies wegen der n : m-Beziehung keinen Sinn ergibt
- In unserem Beispiel gibt es die Verbindungstabelle Projektmitarbeit
- Hier werden die Mitarbeiter erfasst, die an bestimmten Projekten beteiligt sind

Verbindungstabellen

- Unser Beispiel für die Verbindungstabelle Projektmitarbeit
- Die Primärschlüssel aus den Tabellen Mitarbeiter und Projekte werden als Fremdschlüssel in der Tabelle Projektmitarbeit genutzt
- Um die Zuordnung deutlich zu machen werden die gleichen Bezeichnungen gewählt



Aufbau der Datenbank



Bemerkungen zur Datenbank

- In der Abbildung auf der vorigen Folie sind die Tabellen vollständig definiert
- In den Tabellen sind die Primärschlüssel (PK) und die Fremdschlüssel (FK) jeweils hinter den relevanten Einträgen vermerkt, so dass sich die Struktur möglichst einfach nachvollziehen lässt.
- Die optionalen Felder sind nicht fett hinterlegt. Hiermit lassen sich Firmen- und Privatkunden gut unterscheiden (Vorname – nur für Privatkunden)
- Auch eine solche grafische Darstellung sollte in jedem Fall erstellt und ausreichend dokumentiert werden
- Die allgemeine Bezeichnung Wohnort wird auch für die Projekte-Orte angewandt

MariaDB installieren

- Wie schon erwähnt ist der Fork von MySQL die Community-Lösung MariaDB die heute in den Paketquellen aller Linux-Distributionen vorhanden ist
- Die Installation erfolgt auf unserem Server deb-s1 und wird wie gewohnt vom Client debian aus über ssh administriert
- Wie üblich wird MariaDB als Server mit dem Paketmanager apt installiert

```
root@deb-s1:~# apt install mariadb-server
```

- Nach einem Passwort wird nicht gefragt. Das root-Passwort ist zunächst nutzbar, kann aber angepasst werden, was wir später auch tun

Installation testen

- Mit mysqlshow –p kann man sich mit dem Passwort von root anmelden
- Danach werden die bereit vorhandenen Datenbanken angezeigt

```
root@deb-s1:~# mysqlshow -p
Enter password:
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
```

- Mariadb nutzt den tcp-Port 3306 (genau wie MySQL)

```
root@deb-s1:~# netstat -tulpn | grep 3306
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN      1291/mariadbd
```

SQL

- Wie treten wir mit dem MariaDB-Datenbanksystem in Kontakt?
- Hierzu gibt es die Standardsprache zur Datenbankverwaltung.
- Sie heißt SQL u. steht für Structured Query Language (oft sequel gesprochen ☺)
- SQL wird in vielen gängigen Datenbanksystemen verwendet z.B. MS SQL-Server, Oracle, PostgreSQL usw.
  - Dabei existiert für fast jedes System ein eigener SQL-Dialekt
  - SQL teilt sich auf in DDL (Data Definition Language) zur Erstellung von Datenbankstrukturen und
  - DML (Data Manipulation Language) zur Abfrage und Manipulation der Inhalte einer Datenbank
  - Für die praktische Nutzung ist diese Aufteilung rein formaler Natur

Erstellen einer Datenbank

- Es gibt viele Möglichkeiten MariaDB zu administrieren
  - Einige basieren auf grafischen Oberflächen
  - Andere sind webbasierend (werden im Browser administriert) z.B. PHPMyAdmin
- Zum jetzigen Zeitpunkt begeben wir uns aber auf die Konsole um wichtige SQL-Statements (Befehle) kennenzulernen
- Diese werden bei den grafisch basierenden Administrationstools hinter Eingabemasken und Buttons versteckt und sind damit ungeeignet die Basics zu lernen
- Wir nutzen den MariaDB-Monitor, den wir folgendermaßen aufrufen:

Mariadb-Monitor

- Mit der Option –p wird das Passwort abgefragt, also nicht vergessen!
- Die Anmeldung erfolgt in gleicher Weise, wenn man mysql –p eingibt

```
root@deb-s1:~# mariadb -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 32
Server version: 10.5.15-MariaDB-0+deb11u1 Debian 11

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

- Nehmen Sie die Informationen auf oder notieren Sie diese für den Anfang
- Durch Eingabe von exit kann man den MariaDB-Monitor jederzeit verlassen

Erste Statements

- Kommandos werden immer mit einem Semikolon ; abgeschlossen (Die Nutzung von \g ist unüblich)
- Mit help; oder \h holt man sich Hilfe
- Mit \c löscht man das derzeit eingegebene Kommando (Statement)
- MariaDB nutzt einen eigenen Prompt, der in den eckigen Klammern den Namen der jeweiligen Datenbank anzeigt (ohne Datenbank – none)
- Unsere Datenbank der Firma Alpha GmbH soll, wer hätte es gedacht, alpha heißen

Datenbank anlegen und auswählen

- Schlüsselwörter werden bei SQL aus Konventionsgründen groß geschrieben. Allerdings unterscheidet SQL nicht zwischen Groß- und Kleinschreibung
- Vergessen Sie nicht jede Anweisung mit ; abzuschließen und mit der Enter-Taste in die nächste Zeile zu wechseln

- Datenbank anlegen, auswählen und deren Tabellen anzeigen:

```
MariaDB [(none)]> CREATE DATABASE alpha;
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> use alpha
Database changed
MariaDB [alpha]> show tables;
Empty set (0.000 sec)

MariaDB [alpha]>
```

- Oh Wunder – keine Tabellen zu sehen. Das Anlegen der Tabellen ist unser Job.

Datentypen

- **Jeder Spalte (jedem Attribut) wird ein Datentyp zugewiesen, wodurch der Inhalt eines Feldes definiert wird. Hier eine Liste der wichtiger Datentypen:**

Datentyp	Bedeutung
CHAR(m)	Eine Zeichenkette fester Länge, mit max. 255 Zeichen, m gibt die zu verwendente Zeichenzahl an
TEXT	Text mit variabler Länge, aber max. 65535 Zeichen
INTEGER	Ganze Zahl mit oder ohne Vorzeichen. Wird ein Vorzeichen angegeben halbiert sich der Wertebereich von 0 bis 4294967295. Es gibt Untertypen von INTEGER z.B. SMALLINT, BIGINT, usw
FLOAT(m,n)	Fließkommazahl mit und ohne Vorzeichen, m,n gibt die Anzahl der Vor- und Nachkommastellen an. Bereich: 1.18E-38 bis 3.40E+38 bzw. -3.40E+38 bis -1.18E-38. Die Daten werden normalerweise in Dezimalform, z.B. 65,98 angegeben. Für einen erweiterten Datenbereich gibt es noch DOUBLE
DATE	Spezifisches Datumsformat: JJJJ-MM-TT
YEAR	Jahreszahl vierstellig: JJJJ
TIME	Zeitangabe 00:00:00 bis 23:59:59
BLOB	Binärdaten variabler Länge, bis max. 65535 Bytes

1. Tabelle - Mitarbeiter

- Mit CREATE TABLE <Tabellenname>; werden Tabellen erstellt

```
MariaDB [alpha]> CREATE TABLE Mitarbeiter (
    -> Mitarbeiter_ID SMALLINT NOT NULL AUTO_INCREMENT,
    -> Name CHAR(25) NOT NULL,
    -> Vorname CHAR(25) NOT NULL,
    -> Strasse CHAR(25) NOT NULL,
    -> PLZ CHAR(7) NOT NULL,
    -> Abteilung CHAR(25) NOT NULL,
    -> Position CHAR(25) NOT NULL,
    -> Einstellungsdatum DATE NOT NULL,
    -> PRIMARY KEY (Mitarbeiter_ID)
    -> );
Query OK, 0 rows affected (0.026 sec)

MariaDB [alpha]>
```

  - Die Attribute erhalten ihren Datentyp z.B. CHAR(m), mit NOT NULL wird festgelegt, dass ein Eintrag erfolgen muss, bis auf das letzte Attribut wird jede Zeile mit Komma beendet, in der letzten Zeile wird der Primärschlüssel festgelegt, hinter der schließenden runden Klammer folgt das Semikolon;

1. Tabelle - Mitarbeiter

- Mit EXPLAIN <Tabellenname>; kann man sich die Zusammenstellung anzeigen lassen

```
MariaDB [alpha]> EXPLAIN Mitarbeiter;
+--------------------+-------------+------+-----+---------+----------------+
| Field              | Type        | Null | Key | Default | Extra          |
+--------------------+-------------+------+-----+---------+----------------+
| Mitarbeiter_ID     | smallint(6) | NO   | PRI | NULL    | auto_increment |
| Name               | char(25)    | NO   |     | NULL    |                |
| Vorname            | char(25)    | NO   |     | NULL    |                |
| Strasse            | char(25)    | NO   |     | NULL    |                |
| PLZ                | char(7)     | NO   |     | NULL    |                |
| Abteilung          | char(25)    | NO   |     | NULL    |                |
| Position           | char(25)    | NO   |     | NULL    |                |
| Einstellungsdatum  | date        | NO   |     | NULL    |                |
+--------------------+-------------+------+-----+---------+----------------+
8 rows in set (0.001 sec)

MariaDB [alpha]>
```

2. Tabelle - Projekte

```
MariaDB [alpha]> CREATE TABLE Projekte (
    -> Projekt_ID SMALLINT NOT NULL AUTO_INCREMENT,
    -> Bezeichnung CHAR(25) NOT NULL,
    -> Strasse CHAR(25) NOT NULL,
    -> PLZ CHAR(7) NOT NULL,
    -> Auftraggeber CHAR(25) NOT NULL,
    -> PRIMARY KEY (Projekt_ID)
    -> );
Query OK, 0 rows affected (0.024 sec)

MariaDB [alpha]> EXPLAIN Projekte;
+--------------+-------------+------+-----+---------+----------------+
| Field        | Type        | Null | Key | Default | Extra          |
+--------------+-------------+------+-----+---------+----------------+
| Projekt_ID   | smallint(6) | NO   | PRI | NULL    | auto_increment |
| Bezeichnung  | char(25)    | NO   |     | NULL    |                |
| Strasse      | char(25)    | NO   |     | NULL    |                |
| PLZ          | char(7)     | NO   |     | NULL    |                |
| Auftraggeber | char(25)    | NO   |     | NULL    |                |
+--------------+-------------+------+-----+---------+----------------+
5 rows in set (0.001 sec)
```

3. Tabelle - Kunden

```
MariaDB [alpha]> CREATE TABLE Kunden (
    -> Kunden_ID SMALLINT NOT NULL AUTO_INCREMENT,
    -> Name CHAR(25) NOT NULL,
    -> Vorname CHAR(25),
    -> Strasse CHAR(25) NOT NULL,
    -> PLZ CHAR(7) NOT NULL,
    -> Ansprechpartner CHAR(25),
    -> PRIMARY KEY (Kunden_ID)
    -> );
Query OK, 0 rows affected (0.022 sec)

MariaDB [alpha]> EXPLAIN Kunden;
+------------------+-------------+------+-----+---------+----------------+
| Field            | Type        | Null | Key | Default | Extra          |
+------------------+-------------+------+-----+---------+----------------+
| Kunden_ID        | smallint(6) | NO   | PRI | NULL    | auto_increment |
| Name             | char(25)    | NO   |     | NULL    |                |
| Vorname          | char(25)    | YES  |     | NULL    |                |
| Strasse          | char(25)    | NO   |     | NULL    |                |
| PLZ              | char(7)     | NO   |     | NULL    |                |
| Ansprechpartner  | char(25)    | YES  |     | NULL    |                |
+------------------+-------------+------+-----+---------+----------------+
6 rows in set (0.001 sec)
```

4. Tabelle - Wohnort

```
MariaDB [alpha]> CREATE TABLE Wohnort (
    -> PLZ CHAR(7) NOT NULL,
    -> Wohnort CHAR(25) NOT NULL,
    -> PRIMARY KEY (PLZ)
    -> );
Query OK, 0 rows affected (0.026 sec)

MariaDB [alpha]> EXPLAIN Wohnort;
+---------+----------+------+-----+---------+-------+
| Field   | Type     | Null | Key | Default | Extra |
+---------+----------+------+-----+---------+-------+
| PLZ     | char(7)  | NO   | PRI | NULL    |       |
| Wohnort | char(25) | NO   |     | NULL    |       |
+---------+----------+------+-----+---------+-------+
2 rows in set (0.001 sec)
```

5. Tabelle - Projektmitarbeit

```
MariaDB [alpha]> CREATE TABLE Projektmitarbeit (
    -> Projekt_ID SMALLINT NOT NULL,
    -> Mitarbeiter_ID SMALLINT NOT NULL
    -> );
Query OK, 0 rows affected (0.027 sec)

MariaDB [alpha]> EXPLAIN Projektmitarbeit;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| Projekt_ID     | smallint(6) | NO   |     | NULL    |       |
| Mitarbeiter_ID | smallint(6) | NO   |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
2 rows in set (0.001 sec)
```

Tabellenübersicht

- Durch das Anlegen der Tabellen macht der Befehl show tables; Sinn, denn die erstellten Tabellen werden jetzt auch angezeigt

```
MariaDB [alpha]> show tables;
+--------------------+
| Tables_in_alpha    |
+--------------------+
| Kunden             |
| Mitarbeiter        |
| Projekte           |
| Projektmitarbeit   |
| Wohnort            |
+--------------------+
5 rows in set (0.000 sec)

MariaDB [alpha]>
```