Name: Malte Bodenbach
Klasse: 2FA203
Datum:
Schuljahr: 2022/23

1. **Zentrale Datenhaltung** – Wiki Datenbank

   Was versteht man unter folgenden Begriffen?

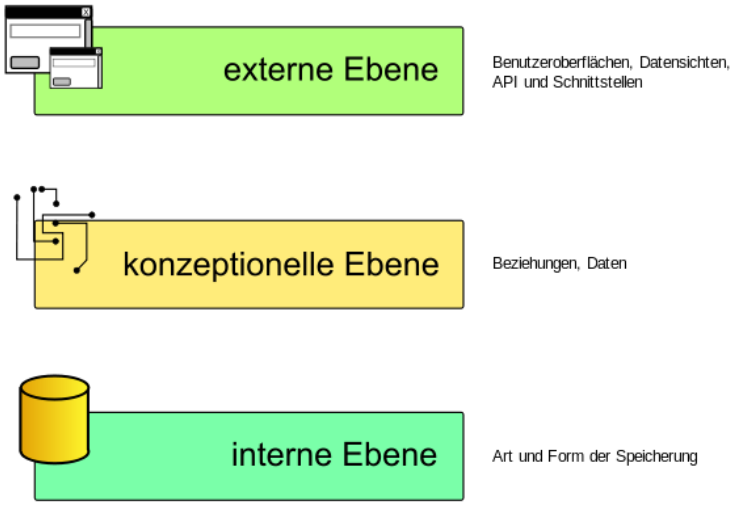
   a) Datenbank (DB)

   Eine Datenbank ist eine strukturiert gespeicherte Sammlung von Daten

   b) Datenbankmanagementsystem (DBMS)

   Ein DBMS ist die Software, um mit einer Datenbank zu interagieren (Speicherung, Abfrage, Änderung, User-Management, Funktionalitäten, Performance etc.)
   Es legt das Datenbankmodell fest.

2. **ANSI-SPARC-Architektur** – Wiki ANSI Bildquelle: Wikipedia (gemeinfrei)



   a) Beschreiben Sie die drei Ebenen:

   Die externe Ebene stellt die Schnittstelle zwischen Benutzer und der Datenbank dar.
   Beispiele: Formulare, Masken-Layouts, APIs, Listen etc.

   Die konzeptionelle Ebene ist, welche Daten in der Datenbank gespeicher sind, sowie deren Beziehung zueinander

   interne Ebene (physische Ebene) beschreibt wie und wo die Daten physisch in der Datenbank gespeichert werden

b) Nennen Sie mindestens zwei Vorteile dieser Architektur.

1) Physische Datenunabhängigkeit: Die interne Ebene ist von der konzeptionellen und externen Ebene getrennt. Änderungen des Speichermediums oder des Datenbankprodukts haben keine Auswirkung auf die konzeptionelle und externe Ebene.

2) Logische Datenunabhängigkeit: Die konzeptionelle und die externe Ebene sind getrent. Änderungen an der Datenbankstruktur (konzeptionelle Ebene) haben keine Auswirkung auf die externe Ebene.

c) Ordnen Sie die folgenden Begriffe den Ebenen im ANSI-Architekturmodell zu!
1 externe Ebene, 2 konzeptionelle Ebene, 3 interne Ebene

DBMS	2
Administrator	1, 2, 3
MariaDB	2
semantisches Modell	2
ER-Modell	2
Datei	3
GUI	1
Sachbearbeiter	1

3. **AUF-03-1** **Installieren Sie MariaDB** auf Ihrem System entweder als standalone Lösung[1] oder als Portable Installation unter Windows[2] und testen Sie diese.

Notiz:

do be datenbank

1 https://mariadb.com/kb/en/getting-installing-and-upgrading-mariadb/
2 https://it.gentiana-clusii.de/mariadb/

Phasen der Datenbankentwicklung

Das Datenmodell, das die für ein Informationssystem notwendigen Daten und Datenbeziehungen beschreibt, wird in vier Phasen entwickelt:

- **Externe Phase** **Informationsstruktur**
  - Ermittlung des Informationsbedarfs der Benutzer
  - Strukturierung dieser Informationen
  - DBMS unabhängig (auf dem Papier!)
- **Konzeptionelle Phase** **semantisches Modell – ER-Modell**
  - Formale und strukturierte Beschreibung aller relevanten Objekte und
  - deren Beziehungen untereinander.
  - DBMS unabhängig (auf dem Papier!)
- **Logische Phase** **logisches / relationales Modell – Tabellenmodell**
  - Umsetzung des semantischen Datenmodells in ein relationales Datenbankmodell.
  - DBMS unabhängig (auf dem Papier!)
- **Physische Phase** **Implementierung mit Software – SQL**
  - DBMS abhängig

Notiz:

Informationsstruktur ermitteln

In der externen Phase wird die Informationsstruktur des Datenmodells geplant. Dazu wird der Informationsbedarf der Benutzer ermittelt und strukturiert. Es muss herausgefunden werden, welche Informationen das Datenbanksystem liefern soll (Output) und welche Informationen dafür bereitzustellen sind (Input). Aus dieser Analyse ergibt sich die Informationsstruktur. Der Input bildet später die Datenbasis der Datenbank (z. B. Geschäftsobjekte wie Schüler und Klassen), der Output die zu erzielenden Ergebnisse, die Benutzersichten, z. B. in Form von Berichten und Formularen (Darstellungsobjekte wie Schülererfassungsmasken, Klassenlisten und Zeugnisse).

Ermittlung aufgrund von Realitätsbeobachtungen

Dieses Verfahren eignet sich zur Konstruktion eines globalen wie auch eines anwendungsorientierten Grobmodells, kann aber auch bei der Verfeinerung eingesetzt werden. Dabei werden die Vorgänge im wesentlichen beobachtet.

Ermittlung aufgrund von Realitätsbeobachtungen

Ermittlung aufgrund von Benutzersichtanalysen

Die Benutzersicht ist die Sicht, aus der der einzelne Benutzer die Daten sieht. Hier werden oft die Benutzer/Anwender direkt gefragt, was diese wie machen. Benutzersichten stellen zum Beispiel Formulare und Berichte dar.

Ermittlung aufgrund von Benutzersichtanalysen

Ermittlung aufgrund von Datenbestandsanalysen

Dieses Verfahren ermöglicht die Integration existierender Datenbestände in ein neues Datenmodell.

Ermittlung aufgrund von Datenbestandsanalysen

Notiz:

Fragen **AUF-03-2**

Formulieren Sie die Antworten mit Ihren eigenen Worten.

1. Nennen Sie die vier Phasen der Datenbankentwicklung.

2. Was versteht man unter „Realitätsbetrachtung“?

3. Was versteht man unter „Benutzersichtanalysen“?

4. Was versteht man unter „Datenbestandsanalysen“?

SQL - Grundlagen

SQL (das Kürzel steht für **S**tructured **Q**uery **L**anguage) ist eine Datenbanksprache zur *Definition*, *Abfrage* und *Manipulation* von Daten in relationalen Datenbanken.

Geschichte

SQL ist von ANSI und ISO standardisiert und wird von fast allen gängigen Datenbanksystemen unterstützt.

1968 Erste Versuche im Rahmen einer studentischen Arbeit in Berkeley:
„Phase Zero prototype of SEQUEL“

1974 DBMS-Sprachen von IBM wie R*, SQL/DS, und DB2

1986 erster SQL-Standard vom ANSI verabschiedet (auch **SQL1** genannt)

1987 SQL-Standard wurde von ISO ratifiziert

1992 Überarbeitung von SQL: **SQL92** (auch **SQL2** genannt)

1999 Erneute Überarbeitung: **SQL99** (auch **SQL3** genannt)

2003 Erneute Überarbeitung: **SQL:2003**

2006 Erneute Überarbeitung und Einbindung von XML: **SQL/XML**

2008 Erneute Überarbeitung: **SQL:2008**

2011 Erneute Überarbeitung: **SQL:2011**

2016 Erneute Überarbeitung: JSON wurde aufgenommen **SQL:2016**

2019 Erneute Überarbeitung: Datentyp „mehrdimensionales Feld“ wurde aufgenommen **SQL:2019**

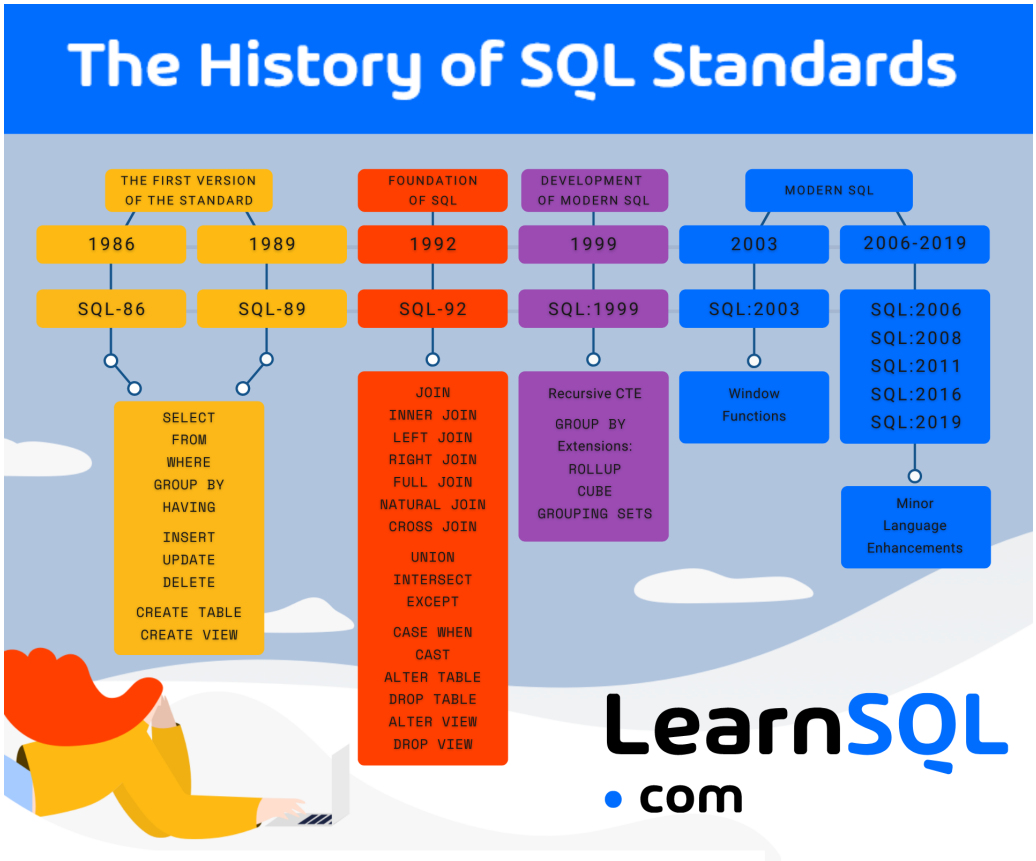
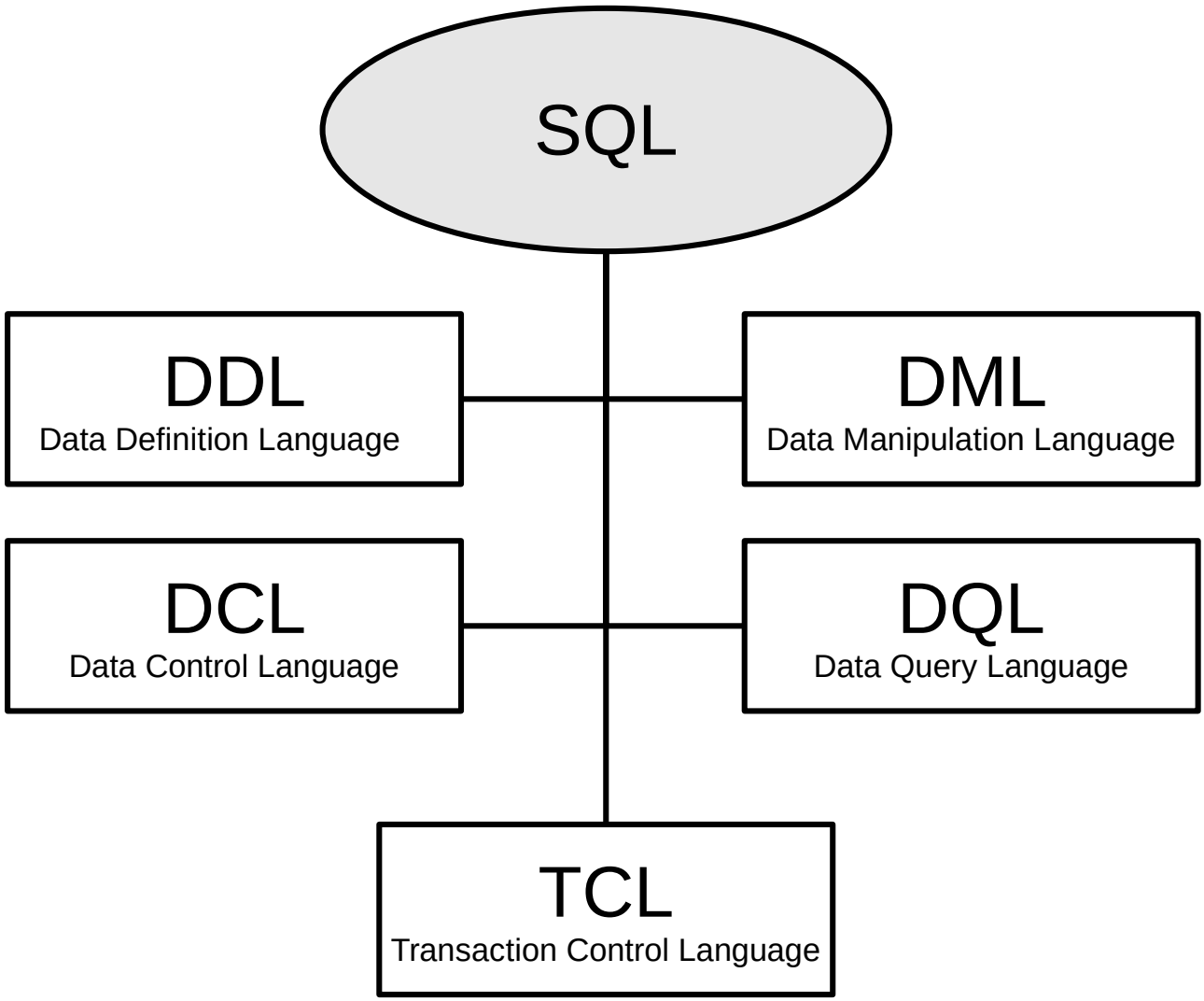


Abbildung 1: Quelle: learnsql.com Stand 06.07.2022

SQL-Übersicht



SQL gliedert sich in mehrere Bereiche auf:

DDL Die Data Definition Language definiert die Struktur der Datenbank. Dies bedeutet, damit wird beispielsweise die Tabelle angelegt, geändert, gelöscht, ...

`CREATE, ALTER, DROP, ...`

```
CREATE TABLE GRUNDBEITRAEGE (
  Beitrags_ID     INTEGER NOT NULL,
  Beitragsgruppe  VARCHAR(20),
  Grundbeitrag    NUMERIC(6,2),
  PRIMARY KEY (Beitrags_ID)
)
```

DML Die Data Manipulation Language dient dazu, die Daten in Tabellen zu ändern, einzufügen, ...

`INSERT, UPDATE, DELETE, ...`

```
INSERT INTO GRUNDBEITRAEGE VALUES (0,'Familienmitglied',0.0000);
INSERT INTO GRUNDBEITRAEGE VALUES (1,'Familienbeitrag',200.0000);
INSERT INTO GRUNDBEITRAEGE VALUES (2,'Erwachsener',100.0000);
INSERT INTO GRUNDBEITRAEGE VALUES (3,'Kind / Student',60.0000);
```

DQL Die Data Query Language dient dazu, Tabellen abzufragen.

`SELECT`

```
SELECT M_ID,Nachname,Vorname,
       date_format(Geburtsdatum,'%Y-%m-%d')
       FROM MITGLIEDER ORDER BY Geburtsdatum;
```

DCL Die Data Control Language dient dazu, Zugriffsrechte zu definieren.
GRANT, REVOKE

```
GRANT SELECT, DELETE, UPDATE ON GRUNDBEITRAEGE TO Mueller;
```

TCL Die Transaction Control Language dient dazu, sicherzustellen, dass Transaktionen (mehrere zusammenhängende SQL-Anweisungen) gesamt abgeschlossen oder der ursprüngliche Zustand im Fehlerfall wieder hergestellt wird.
COMMIT, ROLLBACK

Notiz:

SQL - create database

Nachfolgendes Skript (`create_db_gm.sql`) erzeugt die Datenbank für die Beispiele.

```
1  # create db gm[1..x], user test - kein PW
2
3  USE mysql;
4
5  DROP DATABASE IF EXISTS gm1;
6  CREATE DATABASE gm1 CHARACTER SET utf8;
7
8  DROP DATABASE IF EXISTS gm2;
9  CREATE DATABASE gm2 CHARACTER SET utf8;
10
11 DROP DATABASE IF EXISTS gm3;
12 CREATE DATABASE gm3 CHARACTER SET utf8;
13
14 DROP USER IF EXISTS 'test'@'localhost';
15 CREATE USER 'test'@'localhost' IDENTIFIED BY '';
16
17 GRANT ALL PRIVILEGES ON gm1.* TO 'test'@'localhost';
18 GRANT ALL PRIVILEGES ON gm2.* TO 'test'@'localhost';
19 GRANT ALL PRIVILEGES ON gm3.* TO 'test'@'localhost';
20
21 FLUSH PRIVILEGES;
```

Die Zeilen haben dabei folgende Bedeutung:

1 Kommentare im SQL-Skript (werden ignoriert)

`#`	Kommentar bis zum Zeilenende
`--`	Kommentar bis zum Zeilenende
`/* ... */`	Kommentar von bis
`/*! MySQL-Code */`	spezieller MySQL-Code im Kommentar verpackt (dieser Code wird nur von MySQL berücksichtigt, andere SQL-Systeme ignorieren diesen)

3 Es wird eine Verbindung zur Datenbank *mysql* aufgebaut. In dieser Datenbank werden alle systeminternen Werte und Einstellungen verwaltet. Nur der User *root* kann auf diese Datenbank zugreifen.

5 Die Datenbank „gm1“ wird gelöscht, wenn diese vorhanden ist. Auf diese Weise kann das Skript mehrfach ausgeführt werden und bricht nicht ab, wenn die Datenbank schon vorhanden ist.

6 Die Datenbank „gm1“ wird erzeugt. Dabei wird als Zeichenkodierung (Encoding) UTF-8 verwendet. Es ist immer zu empfehlen, hier das Encoding anzugeben, da sonst das Standardencoding verwendet wird, was u. a. gerade bei Umlauten zu Problemen führen kann.

8-12 ...

14 Der User „test“ wird hier gelöscht, wenn er vorhanden ist.

15 Der User „test“ wird angelegt. Dabei darf dieser nur vom lokalen Rechner (`localhost`) zugreifen. Das Passwort bleibt leer.

17 Der User „test“ bekommt alle Standardrechte für die Datenbank „gm1“.

18-19 ...

21 Die Rechte sind jetzt eingetragen, aber noch nicht aktiv. Erst nach diesem Befehl werden die Einstellungen wirksam.

Der Aufruf sieht dabei wie folgt aus:

```
mysql -u root mysql < [Verzeichnis]create_db_gm.sql
```

Wird ein Passwort verwendet, so muss zusätzlich der Parameter `-p` verwendet werden:

```
mysql -u root -p mysql < [Verzeichnis]create_db_gm.sql
```

Siehe hierzu auch Installation und Test von MariaDB.

AUF-03-3 Erstellen Sie nun die Datenbanken „gm1“, „gm2“ und „gm3“ und probieren Sie die Kennung „test“ aus.

Notiz:

ER-Modell

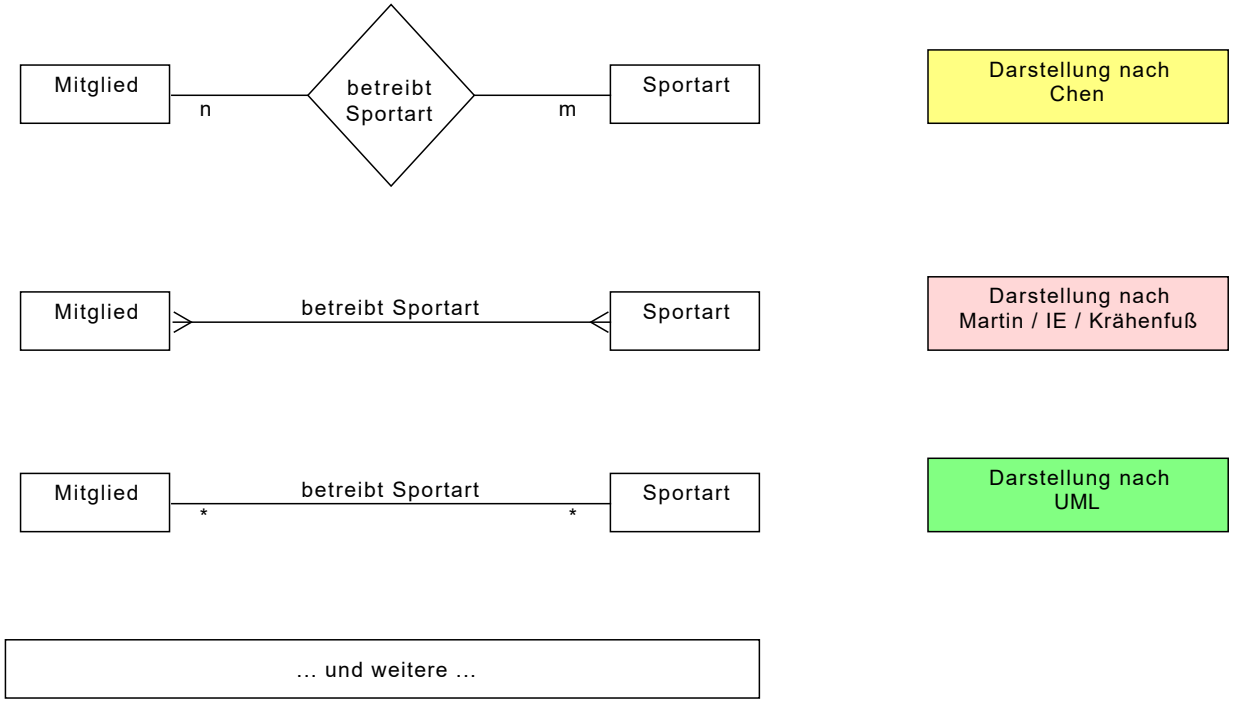
„Das **Entity-Relationship-Modell** – kurz **ER-Modell** oder ERM; deutsch so viel wie: Modell (zur Darstellung) von Dingen, Gegenständen, Objekten (= **‚entities'**) und der Beziehungen/Zusammenhänge zwischen diesen (= **‚relationship'**) – dient dazu, im Rahmen der semantischen Datenmodellierung den in einem gegebenen Kontext (z. B. einem Projekt zur Erstellung eines Informationssystems) relevanten Ausschnitt der realen Welt zu bestimmen und darzustellen. Das ER-Modell besteht im Wesentlichen aus einer Grafik (ER-Diagramm, Abk. ERD) sowie einer Beschreibung der darin verwendeten Elemente.

Ein ER-Modell dient sowohl in der **konzeptionellen Phase** der Anwendungsentwicklung der Verständigung zwischen Anwendern und Entwicklern (dabei wird nur das Was behandelt, d. h. fachlich-sachliche Gegebenheiten, nicht das Wie, z. B. die Technik) als auch in der Implementierungsphase als Grundlage für das Design der – meist relationalen – Datenbank.

Der Einsatz von ER-Modellen ist der De-facto-Standard für die Datenmodellierung, auch wenn es unterschiedliche grafische Darstellungsformen für Datenmodelle gibt.

Das ER-Modell wurde 1976 von Peter Chen in seiner Veröffentlichung The Entity-Relationship Model vorgestellt."[3]

Es gibt inzwischen viele Darstellungsformen für das ERM, jedoch haben sich drei Varianten besonders herausgestellt. Die UML-Darstellung ist dabei die modernste Version und stellt darüber hinaus mehr Möglichkeiten zur Verfügung.[4]



Informieren Sie sich über das ER-Modell u. a. mit folgenden Quellen:

- Entity-Relationship-Modell
- Chen-Notation
- Informationsmodellierung mit Entity-Relationship-Modell und UML

3 Quelle: `https://de.wikipedia.org/wiki/Entity-Relationship-Modell` Stand: 18.08.2021

4 Bei Prüfungsausgaben wird oft nur angegeben, dass ein ER-Diagramm zu erstellen ist, jedoch nicht, welche Zeichenvariante. Es ist daher zu empfehlen, dass hier beispielsweise „ERM in UML-Notation" angegeben wird.

AUF-03-4 Ermitteln Sie die Kardinalitäten (Vielfachanzahl – 1, n, m) der Beziehungen!

1. Person — ist Halter von — PKW

2. Mann — ist verheiratet mit — Frau

3. Person — spricht — Sprache

4. Person — besitzt — EC-Karte

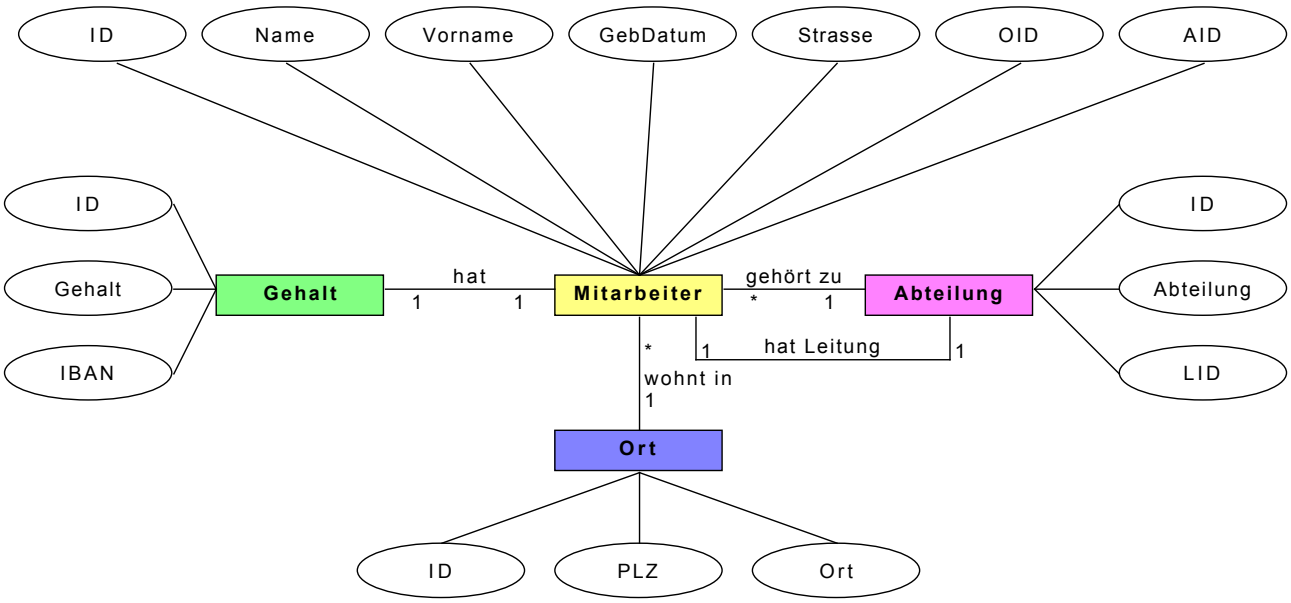
5. Person — nimmt teil — Kurs

6. Login-Name — hat — Passwort

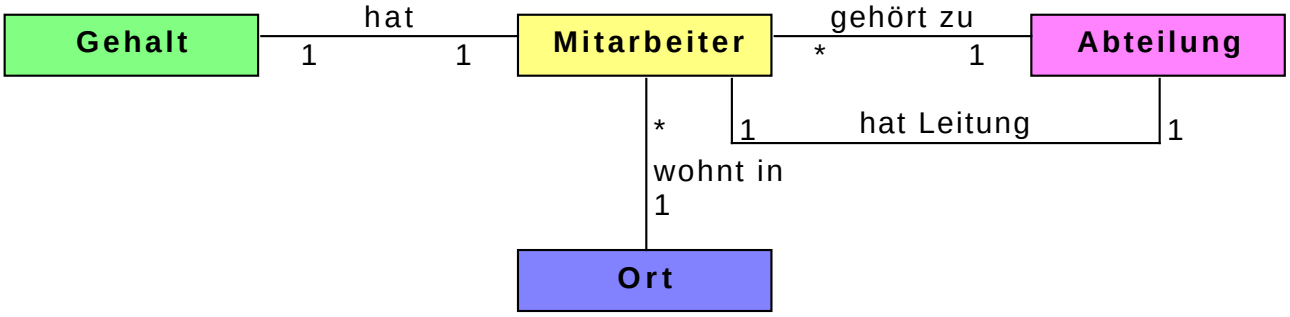
7. Person — hat — Reisepass — hat — Passfoto

Notiz:

Im Bereich „Einführung Mitarbeiter“ haben Sie die Mitarbeiterdaten „zerlegt“. Ein entsprechendes ER-Modell kann wie folgt aussehen.



Im Normalfall verzichtet man beim ER-Modell auf Attribute, da man hier im wesentlichen die Beziehungen zwischen den Entitäten darstellen möchte.[5]



AUF-03-5 Zeichnen Sie das ER-Modell einmal mit Attributen und einmal ohne Attribute. Verwenden Sie dazu das Tool UMLet.

AUF-03-6 Erstellen Sie für die Aufgaben (siehe PDF-Aufgabenblatt `ERM-ueb_01.pdf`) entsprechende ER-Diagramme.

AUF-03-7 Weitere Aufgaben finden Sie unter `ERM-ueb_02.pdf`. Erstellen Sie für die Aufgaben ein ER-Diagramm.

Notiz:

5 Achten Sie bei Prüfungsfragen darauf, ob Sie Attribute verwenden sollen oder nicht.

Logische Phase – Tabellenmodell erstellen

In der logischen Phase wird das semantische Modell (ER-Modell) zum relationalen Modell (Tabellenmodell) umgewandelt. Das relationale Modell kann die im semantischen Modell gebildeten 1-1- und 1-n-Beziehungen unmittelbar abbilden, nicht aber die n-m-Beziehungen. Diese komplex-komplexen Beziehungen müssen aufgelöst werden, indem für die Beziehungsmenge eine eigenständige Tabelle (=Relation) definiert wird. Auch wenn für die Beziehungsmenge eine eigene Relation konstruiert wurde, kann zwischen dieser Relation und einer Entitätsmenge immer noch eine n-m-Beziehung bestehen. Die Beziehungsmengen-Relation muss dann in eine weitere Beziehungsmengen-Relation (Detailtabelle) unterteilt werden.

Verwendet man den UML-Stil, so wird normalerweise anstelle von n-m immer der * verwendet.

Schlüssel

Ein Schlüssel (key) ist ein Bestandteil eines Datensatzes, der diesen eindeutig identifiziert.
Man unterscheidet:

- **Primärschlüssel** (Hauptordnungsbegriff, primary key)
  Suchbegriff, über den hauptsächlich zugegriffen wird, der primär die Datenorganisation bestimmt und der eindeutig ist.
- **Sekundärschlüssel** (Nebenordnungsbegriff, secondary key)
  Suchbegriff, der als weiterer Schlüssel vorgesehen ist und mehrfach vorkommen kann.

Arten von Schlüsseln

- **Zählschlüssel** (Künstlicher Schlüssel, artificial key)
  Es wird eine fortlaufende Nummer als Schlüssel verwendet.
- **Identifikationsschlüssel**
  Es wird ein Attribut der Tabelle als Schlüssel verwendet.
- **Verbundschlüssel** (Zusammengesetzter Schlüssel, concatenated key)
  Es werden mehrere Attribute (oder auch Teile davon) als Schlüssel verwendet.
- **Fremdschlüssel** (foreign key)
  Ein Fremdschlüssel in einer Tabelle 2 ist ein Feld oder eine Feldkombination, welche(s) in einer Tabelle 1 den Primärschlüssel bildet.
- ...

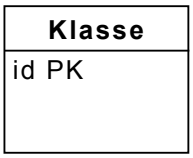
Abbildungsregeln

- **Regel 1** – Entitätsmenge
  Jede Entitätsmenge muss als eigenständige Relation definiert werden.
- **Regel 2** – 1:1-Beziehung
  Es genügt eine Relation. Aus Datenschutzgründen oder für selten benötigte Informationen kann eine eigene Relation definiert werden.
- **Regel 3** – 1:n-Beziehung
  Primärschlüssel der 1-Relation ist Fremdschlüssel in der n-Relation. Keine eigene Beziehungsmengen-Relation notwendig.
- **Regel 4** – m:n-Beziehung
  Jede komplexkomplexe Beziehungsmenge muss als eigenständige Relation definiert werden. Die Primärschlüssel der zugehörigen Entitätsmengen treten als Fremdschlüssel in der Beziehungsmengen-Relation auf.

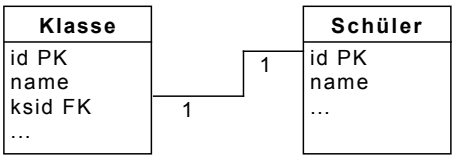
Unter einer **Entität** versteht mein ein Gegenstand oder ein Objekt, dass in Beziehung (Relation) zu anderen Entitäten steht. Sie sind die grundlegenden Modellierungsstrukturen in diesem Modell und sind unterscheidbare bestehende oder gedankliche Objekte oder Gegenstände (z. B. Mitarbeiter, Bestellungen oder Artikel in einem Lager) der bestehenden Welt.

Wie geht man dabei praktisch vor?

1. Für jede Entitätsmenge wird ein Kästchen mit dem Tabellennamen erstellt.
   Tragen Sie hier sofort den PK-Schlüssel ein: „id PK“.
   Wenn Sie für den PK immer „id“ verwenden, müssen Sie nicht lange überlegen und können bei Abfragen immer „id“ gleich verwenden, ohne den Namen erst nachsehen zu müssen.

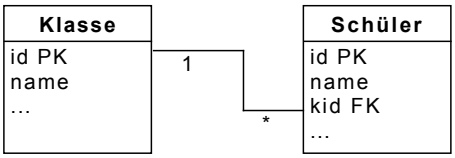


2. Für jede 1:1-Beziehung mit einer eigenen Entitätsmenge wird eine Verbindung zwischen den Kästchen erstellt. Die Linien werden praktischerweise direkt an den Spalten platziert. Welche Seite den anderen Schlüssel bekommt, ist in der Theorie erst mal egal.
   Was in der einen Tabelle PK ist, wird in der anderen Tabelle FK (Primärschlüssel der 1-Relation ist Fremdschlüssel in der „anderen“ 1-Relation.). Es empfiehlt sich hier, einen bzw. zwei Buchstaben vor „id“ zu setzen, so dass schnell erkannt wird, zu welcher Tabelle die Verbindung gehört bzw. welche Funktion dieser hat (ks - **K**lassen**s**precher).



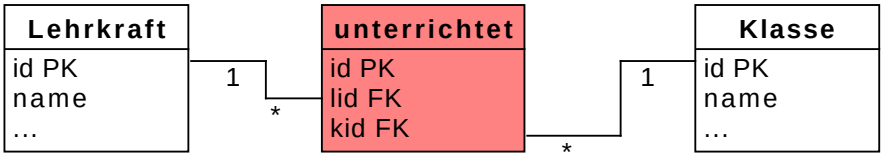
   Bei diesem Beispiel ist es jedoch sinnvoll, den FK in der Klasse zu platzieren, da jede Klasse normalerweise einen Klassensprecher hat, jedoch nicht jeder Schüler immer Klassensprecher ist.

3. Für jede 1:n-Beziehung wird eine Verbindung zwischen den Kästchen erstellt. Primärschlüssel der 1-Relation ist Fremdschlüssel in der n-Relation.



4. Für jede n:m-Beziehung wird eine Zwischentabelle angelegt. Dabei wird die n:m-Beziehung in zwei 1:n-Beziehungen aufgelöst. In der Zwischentabelle darf dabei **kein** eigenständiger PK, beispielsweise ein Zählschlüssel verwendet werden, sondern der PK wird als Verbundschlüssel aus allen FKs erstellt.

Warum darf hier kein Zählschlüssel verwendet werden?
Betrachten wir hier das „schlechte“ Beispiel mit eigenständigem Zählschlüssel!



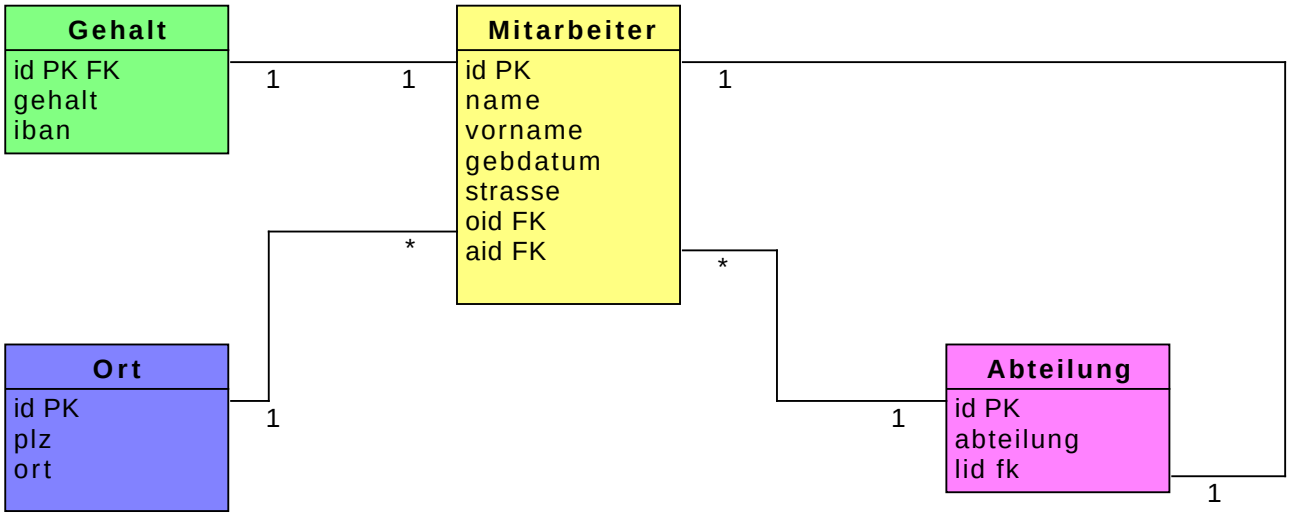
Beispieldaten:

Lehrkraft		unterrichtet			Klasse		
1	Müller	1	1	1	1	2FA101	↯
1	Müller	2	1	2	2	3FA234	
1	Müller	3	1	1	1	2FA101	↯

Wir sehen, dass es zwei Einträge für „Müller“ mit „2FA101“ gibt. Der PK ist atomar, kommt nicht doppelt vor (1, 2, 3), somit kann das DMBS den doppelten Eintrag nicht erkennen. Dies führt dann zu Anomalien, z. B. Löschanomalie bzw. Änderungsanomalie.

Notiz:

Wandelt man nun nach diesen Regeln das Mitarbeiter-ER-Modell in ein Tabellenmodell um, so ergibt sich folgende Lösung:



Dabei werden meist die Datentypen der Spalten nicht angegeben, damit das Modell übersichtlicher bleibt. Auch auf die Beschreibung der Beziehung wird hier meist verzichtet.

AUF-03-8 Zeichnen Sie das Tabellenmodell mit UMLet.

Notiz:

Daten – auf was man achten muss

Beim Beispiel der Excel-Mitarbeitertabelle haben Sie gesehen, dass viele Daten mehrmals vorkommen, also redundant gespeichert werden.

Anomalien und Inkonsistenz

Daten, die redundant gespeichert werden, verschwenden nicht nur zusätzlich Speicherplatz und machen so evtl. Abfragen langsamer, sondern bei Änderungen kann dies zu Fehlern führen, indem nur ein Teil der doppelten Einträge geändert werden, man spricht hier von **Anomalien**.

UPDATE-Anomalie Bei Änderungen werden nicht alle Datensätze geändert. Somit weiß man später nicht mehr, was der „alte Wert“ und was der „neue Wert“ ist. Somit ist die Datenbank nicht mehr stimmig, was man als **Inkonsistenz** bezeichnet. Konsistenz bezeichnet dabei die Widerspruchsfreiheit der Datenbank, die bei Inkonsistenz nicht mehr vorhanden ist.

DELETE-Anomalie Löscht man Daten, um beispielsweise Speicherplatz zu sparen, so kann es vorkommen, dass immer noch ein Verweis auf den gelöschten Datensatz vorhanden ist. Dies könnte bei einer entsprechenden Abfrage zu Fehlern führen. Bei RDBMS hilft hier die „**Referentielle Integrität**“, dies zu verhindern - später mehr dazu.

In der Excel-Mitarbeitertabelle hat das Löschen eines Mitarbeiters evtl. auch zur Folge, dass eine Abteilung verschwindet, wenn diese nur diesen Mitarbeiter hat.

INSERT-Anomalie Nimmt man die Excel-Mitarbeitertabelle, so lassen sich neue Abteilungen nur einfügen, wenn dort auch ein Mitarbeiter vorhanden ist.

Um Redundanzen zu vermeiden und somit Anomalien zu verhindern helfen die Normalformen.

Normalformen

Bei einem „sauberen“ Entwurf gelten bestimmte Regeln.
Diese werden **Normalformen** genannt.

- **Erste Normalform** (1 NF)
  Eine Relation (Tabelle) ist in der ersten Normalform, wenn alle Attribute elementar (atomar) sind.

  Negativbeispiel:

ID	Name	Wohnort
1	Müller, Peter	Ringelweg 12, 12345 Testhausen
...		

- **Zweite Normalform** (2 NF)
  Eine Relation (Tabelle) ist in der zweiten Normalform, wenn alle Nichtschlüssel-Attribute voll funktional abhängig vom Primärschlüssel sind und die erste Normalform erfüllt ist.

  Negativbeispiel:

MID	SportID	Sport
1	1	Fußball
2	1	Fußball
1	2	Handball
3	3	Schwimmen
...		

- **Dritte Normalform** (3 NF)
  Eine Relation befindet sich in der dritten Normalform, wenn alle Nichtschlüssel-Attribute voneinander funktional unabhängig sind und die zweite Normalform erfüllt ist.

  Negativbeispiel:

ID	Name	Geburtsdatum	Alter
1	Peter	1990-10-23	17
2	Uschi	1999-08-21	9
...			

Beispiel

Mit nachfolgendem Beispiel wird schrittweise gezeigt, wie die drei Normalformen umgesetzt werden.

Pers-ID	Nachname	Vorname	Ort	Abt-ID	Abt-Name	Projekt-ID	Beschreibung	Funktion
1	Maier	Hans	München	1	Entwicklung	1001	GUI-Framework	Leiter Entwicklung
						1002	Rest-Prototyp	Leiter Entwicklung
2	Huber	Uschi	Dachau	2	Buchhaltung	1001	GUI-Framework	Controlling
3	Kobold	Pumukel	München	3	Softwaretest	1001	GUI-Framework	Tester
						1002	Rest-Prototyp	Tester
4	Lutz	Marion	Erding	1	Entwicklung	1003	DB-Treiber	Programmierer

Erste Normalform (1 NF)

In obiger Tabelle sind in manchen Zellen mehrere Einträge. Damit ist die 1 NF nicht erfüllt.

Pers-ID	Nachname	Vorname	Ort	Abt-ID	Abt-Name	Projekt-ID	Beschreibung	Funktion
1	Maier	Hans	München	1	Entwicklung	1001	GUI-Framework	Leiter Entwicklung
1	Maier	Hans	München	1	Entwicklung	1002	Rest-Prototyp	Leiter Entwicklung
2	Huber	Uschi	Dachau	2	Buchhaltung	1001	GUI-Framework	Controlling
3	Kobold	Pumukel	München	3	Softwaretest	1001	GUI-Framework	Tester
3	Kobold	Pumukel	München	3	Softwaretest	1002	Rest-Prototyp	Tester
4	Lutz	Marion	Erding	1	Entwicklung	1003	DB-Treiber	Programmierer

Zweite Normalform (2 NF)

Jetzt sind die Einträge atomar, aber die `Pers-ID` ist nicht mehr eindeutig. Auch sind Spalten funktional abhängig vom Primärschlüssel (hier Pers-ID). Daher muss die Tabelle aufgeteilt werden:

- Tabelle Mitarbeiter
- Tabelle Projekt
- Tabelle VerknüpfungFunktion

Mitarbeiter

Pers-ID	Nachname	Vorname	Ort	Abt-ID	Abt-Name
1	Maier	Hans	München	1	Entwicklung
2	Huber	Uschi	Dachau	2	Buchhaltung
3	Kobold	Pumukel	München	3	Softwaretest
4	Lutz	Marion	Erding	1	Entwicklung

Projekt

Projekt-ID	Beschreibung
1001	GUI-Framework
1002	Rest-Prototyp
1003	DB-Treiber

VerknüpfungFunktion

Pers-ID	Projekt-ID	Funktion
1	1001	Leiter Entwicklung
1	1002	Leiter Entwicklung
2	1001	Controlling
3	1001	Tester
3	1002	Tester
4	1003	Programmierer

Dritte Normalform (3 NF)

Betrachtet man nun die Tabellen erneut, so stellt man fest, dass hier immer noch Redundanzen vorliegen und dass hier Nichtschlüssel-Attribute nicht funktional unabhängig sind, z. B. `Abt-ID` und `Abt-Name` in der Tabelle `Mitarbeiter`.

Daher müssen noch zwei weitere Tabellen erstellt werden:

- Tabelle Abteilung
- Tabelle Funktion

Mitarbeiter

Pers-ID	Nachname	Vorname	Ort	Abt-ID
1	Maier	Hans	München	1
2	Huber	Uschi	Dachau	2
3	Kobold	Pumukel	München	3
4	Lutz	Marion	Erding	1

Abteilung

Abt-ID	Abt-Name
1	Entwicklung
2	Buchhaltung
3	Softwaretest

Projekt

Projekt-ID	Beschreibung
1001	GUI-Framework
1002	Rest-Prototyp
1003	DB-Treiber

VerknüpfungFunktion

Pers-ID	Projekt-ID	Fkt-ID
1	1001	1
1	1002	1
2	1001	2
3	1001	3
3	1002	3
4	1003	4

Funktion

Fkt-ID	Funktion
1	Leiter Entwicklung
2	Controlling
3	Tester
4	Programmierer

Nun fällt auf, dass Orte in der Tabelle Mitarbeiter doppelt vorkommen. Dies widerspricht jedoch der 3NF nicht, führt aber zu anderen Problemen, wie z. B. der Änderungsanomalie. Daher ist es sehr sinnvoll, für die Tabelle Ort eine eigene Tabelle zu erstellen.

Aufgabe: AUF-03-9

In einer relationalen Datenbank sollen folgende Feldnamen und Feldinhalte verarbeitet werden:

Aufnr	Auftragsnummer
Kunr	Kundennummer
Kuname	Kundenname
Kuansch	Kundenanschrift
Aufdat	Auftragsdatum
Aufbearb	Bearbeiter (Kürzel des Sachbearbeiters)
Artnr1 - Artnr20	Artikelnummer (max. 20 Artikel pro Auftrag)
Artbez1 - Artbez20	Artikelbezeichnung (max. 20 Artikel pro Auftrag)
Artanz1 - Artanz20	Anzahl Artikel (max. 20 Artikel pro Auftrag)
Artpr1 - Artpr20	Artikelpreis (max. 20 Artikel pro Auftrag)

1. Erstellen Sie ein semantisches Modell. Dabei sollen nicht nur 20 Artikel verwaltet werden, sondern beliebig viele.
2. Wandeln Sie anschließend dieses semantische Modell in ein relationales Modell um.
3. Erstellen Sie die Tabellen unter Beachtung der 3NF.

Notiz:

Referentielle Integrität

Die referentielle Integrität wird bei relationalen Datenbanken dazu verwendet, um die Konsistenz und die Integrität der Daten sicherzustellen.

AUF-03-10 Beschreiben Sie in eigenen Worten nachfolgende Begriffe
z. B. mit Wikipedia: Referentielle Integrität

- Referentielle Integrität

- Änderungsweitergabe (ÄW)

- Löschweitergabe (LW)

Nachteile der Referentiellen Integrität sind:

AUF-03-11 Wandeln Sie nun die erstellten ER-Diagramme aus `ERM-ueb_01.pdf` in entsprechende Tabellenmodelle um. Verwenden Sie dazu das Tool Umlet.

AUF-03-12 Zusatz: Wandeln Sie nun die erstellten ER-Diagramme aus `ERM-ueb_02.pdf` in entsprechende Tabellenmodelle um. Verwenden Sie dazu das Tool Umlet.

Tabellen in MariaDB erstellen

Im nächsten Schritt muss man sich die Datentypen der Spalten überlegen, so dass man diese in der Datenbank anlegen kann.

Alle unterstützten Datentypen von MariaDB findet man hier.

Beschreiben Sie kurz die wichtigsten Datentypen:

INT	
DECIMAL(M,D)	
CHAR(M)	
VARCHAR(M)	
TEXT	
DATE	
DATETIME	
BLOB	
JSON	

Tabellen mit SQL erstellen

Im ersten Schritt werden wir die Tabellen mit SQL-Befehlen erstellen.[6]

Nachfolgend wird die Tabelle „ort“ erstellt.

```
CREATE TABLE ort (
  id   INT          PRIMARY KEY AUTO_INCREMENT,
  plz  VARCHAR(10),
  name VARCHAR(50)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Beschreiben Sie kurz die wichtigsten „Schlüsselwörter“:

PRIMARY KEY	
AUTO_INCREMENT	
ENGINE=InnoDB	
DEFAULT CHARSET=utf8	

```
CREATE TABLE abteilung (
  id        INT          PRIMARY KEY AUTO_INCREMENT,
  abteilung VARCHAR(50),
  lid       INT
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Bei der Tabelle „abteilung“ fällt auf, dass hier der „Foreign key“ fehlt. Das Problem hier ist, dass der FK nur gesetzt werden kann, wenn die dazugehörige Tabelle (hier „mitarbeiter“) schon vorhanden ist. Aus diesem Grund werden die FKs erst nachdem alle Tabellen erstellt worden sind, gesetzt.

```
ALTER table mitarbeiter ADD CONSTRAINT fk_mitarbeiter_abteilung
      Foreign key (aid) References abteilung (id);
```

Beschreiben Sie kurz, wie der Befehl für das Hinzufügen des FKs aufgebaut ist.

6 Bei Prüfungsaufgaben kommt es immer wieder vor, dass Tabellen mit SQL-Befehlen erstellt werden müssen.

Nachfolgend das komplette SQL-Skript (`create_tables_gm1_mitarbeiter.sql`) für das Erstellen der Tabellen. Dabei werden am Anfang (Zeile 5-6) die Überprüfungen für die Schlüssel ausgeschaltet. Somit können die Tabellen beliebig gelöscht werden, ohne dass eine Fehlermeldung erscheint. Danach (Zeile 92-93) muss dies aber wieder eingeschaltet werden. Ab Zeile 59 werden dann die Daten eingefügt, mehr dazu auf Seite 28.

```
1  use gm1;
2  SET NAMES utf8;
3
4  /* FK-/UNIQUE-Check ausschalten */
5  SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
6  SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
7
8  /* Tabellen löschen */
9  DROP TABLE IF EXISTS ort;
10 DROP TABLE IF EXISTS abteilung;
11 DROP TABLE IF EXISTS gehalt;
12 DROP TABLE IF EXISTS mitarbeiter;
13 /* Tabellen - nächste Aufgaben - auch löschen, wenn vorhanden */
14 DROP TABLE IF EXISTS landkreis;
15 DROP TABLE IF EXISTS bundesland;
16 DROP TABLE IF EXISTS funktion;
17
18 CREATE TABLE ort (
19   id   INT          PRIMARY KEY AUTO_INCREMENT,
20   plz  VARCHAR(10),
21   name VARCHAR(50)
22 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
23
24 CREATE TABLE abteilung (
25   id         INT          PRIMARY KEY AUTO_INCREMENT,
26   abteilung VARCHAR(50),
27   lid        INT
28 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
29
30 CREATE TABLE gehalt (
31   id     INT           PRIMARY KEY AUTO_INCREMENT,
32   gehalt DECIMAL(9,2),
33   iban   VARCHAR(25)
34 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
35
36 CREATE TABLE mitarbeiter (
37   id      INT          PRIMARY KEY AUTO_INCREMENT,
38   name    VARCHAR(50),
39   vorname VARCHAR(50),
40   gebdat  DATE,
41   strasse VARCHAR(50),
42   oid     INT,
43   aid     INT
44 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
45
46 ALTER table gehalt ADD CONSTRAINT fk_gehalt_mitarbeiter
47       Foreign key (id) References mitarbeiter (id);
48
49 ALTER table mitarbeiter ADD CONSTRAINT fk_mitarbeiter_ort
50       Foreign key (oid) References ort (id);
51
52 ALTER table mitarbeiter ADD CONSTRAINT fk_mitarbeiter_abteilung
53       Foreign key (aid) References abteilung (id);
54
55 ALTER table abteilung ADD CONSTRAINT fk_abteilung_mitarbeiter
56       Foreign key (lid) References mitarbeiter (id);
57
58
59 insert into ort (plz,name) values ('85609','Aschheim');
60 insert into ort (plz,name) values ('85653','Aying');
61 insert into ort (plz,name) values ('85521','Hohenbrunn');
62 insert into ort (plz,name) values ('85662','Hohenbrunn');
63 insert into ort (plz,name) values ('82061','Neuried');
64 insert into ort (plz,name) values ('82131','Neuried');
65
66 insert into abteilung (abteilung,lid) values ('Geschäftsleitung',1);
67 insert into abteilung (abteilung,lid) values ('Fahrdienst',2);
68 insert into abteilung (abteilung,lid) values ('Verkauf',8);
69 insert into abteilung (abteilung,lid) values ('Buchhaltung',5);
70
71 insert into mitarbeiter (name,vorname,gebdat,strasse,oid,aid) values ('Walker','Jonny
   ','1970-11-04','Blumenstr. 1', 1,1);
72 insert into mitarbeiter (name,vorname,gebdat,strasse,oid,aid) values ('Schlau','Susi',
   '1999-10-14','Dorfplatz 3', 2,2);
```

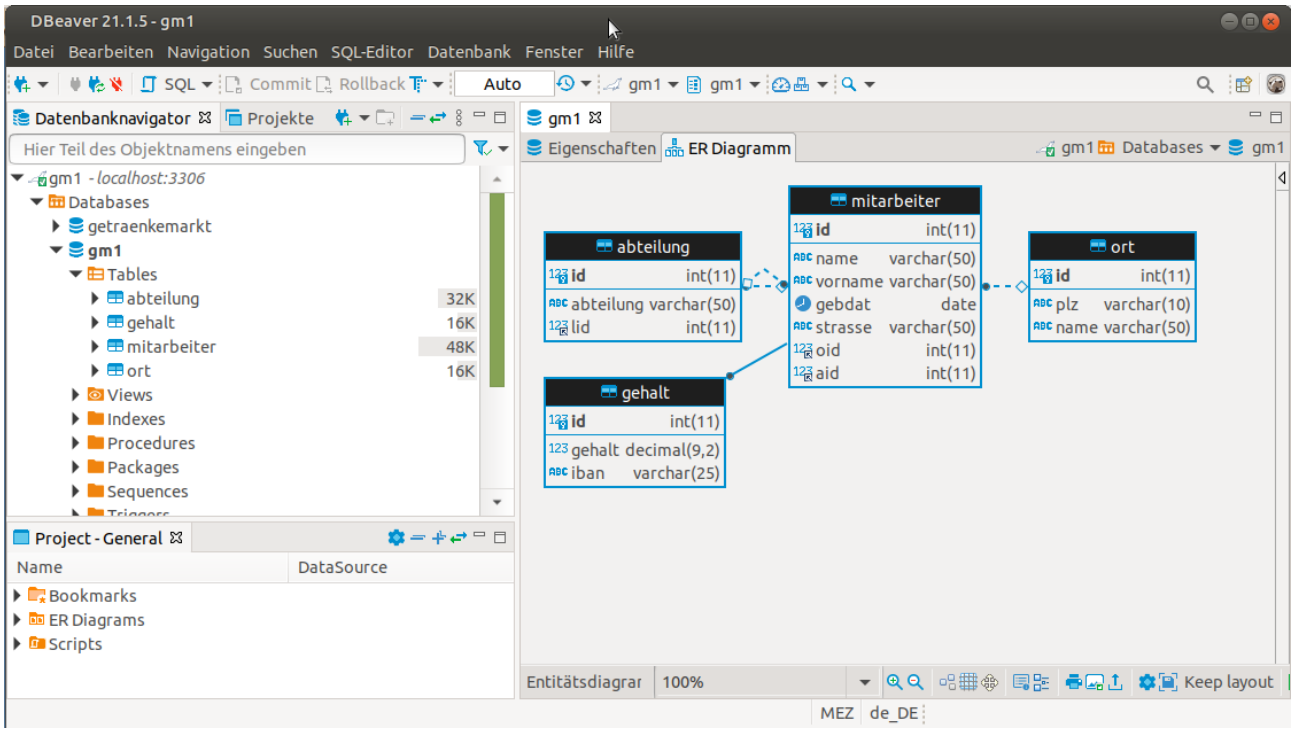

```
insert into mitarbeiter (name,vorname,gebdat,strasse,oid,aid) values ('Ratlos','Rudi', '1980-01-09','Landweg 15', 3,3);
insert into mitarbeiter (name,vorname,gebdat,strasse,oid,aid) values ('Keller','Brigitte', '1985-11-24','Zwergelstr. 12', 5,4);
insert into mitarbeiter (name,vorname,gebdat,strasse,oid,aid) values ('Keller','Josef', '1985-10-13','Zwergelstr. 12', 5,3);
insert into mitarbeiter (name,vorname,gebdat,strasse,oid,aid) values ('Huber','Sepp', '2001-04-04','Kaltenbachstr. 23', 2,1);
insert into mitarbeiter (name,vorname,gebdat,strasse,oid,aid) values ('Meier','Siglinde', '1991-05-01','Waldweg 12', 2,2);
insert into mitarbeiter (name,vorname,gebdat,strasse,oid,aid) values ('Mair','Hans', '1995-10-15','Treppenweg 3', 4,3);
insert into mitarbeiter (name,vorname,gebdat,strasse,oid,aid) values ('Maier','Peter', '1994-11-15','Bgm Huber Straße 2', 4,3);

insert into gehalt (id,gehalt,iban) values (1,11200.78,'DE23399010200021456987');
insert into gehalt (id,gehalt,iban) values (2,3200.11,'DE10245200007849635148');
insert into gehalt (id,gehalt,iban) values (3,4312.00,'DE47290500000000874596');
insert into gehalt (id,gehalt,iban) values (4,5423.56,'DE98590500000096857412');
insert into gehalt (id,gehalt,iban) values (5,4900.12,'DE98590500000096857412');
insert into gehalt (id,gehalt,iban) values (6,9234.99,'DE47660500000000811596');
insert into gehalt (id,gehalt,iban) values (7,2900.98,'DE47660500000000877786');
insert into gehalt (id,gehalt,iban) values (8,3200.48,'DE23343010200021567881');
insert into gehalt (id,gehalt,iban) values (9,3223.48,'DE23343010200021567113');

/* FK-/UNIQUE-Check einschalten */
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

AUF-03-13 Erstellen Sie nun die Tabellen in Ihrer Datenbank.

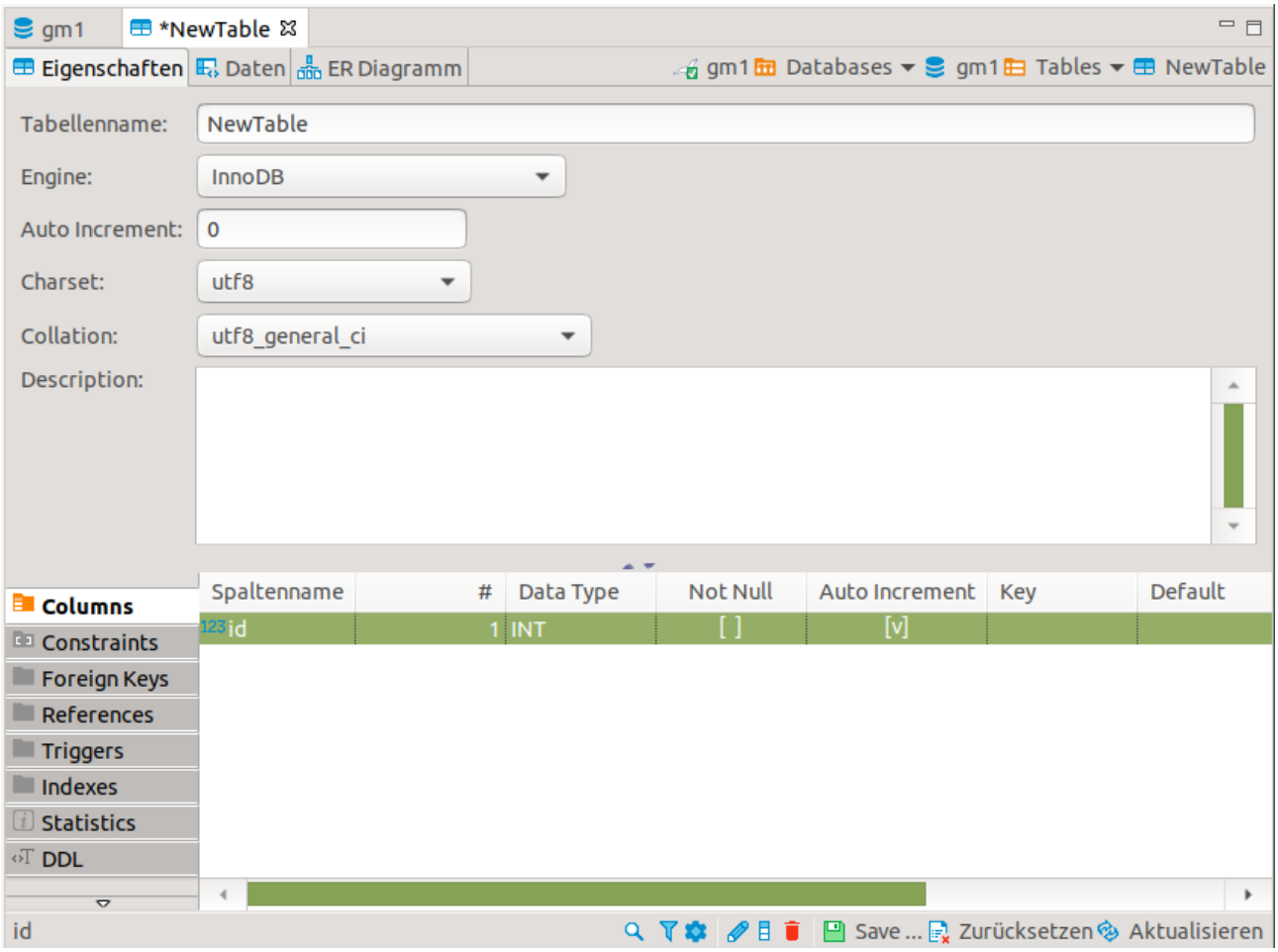
Notiz:

Tabellen grafisch erstellen

Mit dem Tool DBeaver lassen sich u. a. die Tabellen auch sehr gut grafisch erstellen und Datensätze abfragen, hinzufügen, bearbeiten und löschen. Das Tool kann noch viel mehr, beispielsweise die Datenbank als Tabellenmodell darstellen.



Über „Anlegen Tabelle“ lassen sich dann per „klick“ die Tabellen-Definitionen erstellen.



AUF-03-14 Installieren Sie sich „DBeaver“ als portable Version oder standalone Version und probieren Sie es aus.

SQL - Eingabe von Daten

Mit dem Befehl `INSERT` werden Daten in eine Tabelle eingefügt.

```
insert into ort (plz,name) values ('85609','Aschheim');
insert into ort (plz,name) values ('85653','Aying');
insert into ort (plz,name) values ('85521','Hohenbrunn');
insert into ort (plz,name) values ('85662','Hohenbrunn');
insert into ort (plz,name) values ('82061','Neuried');
insert into ort (plz,name) values ('82131','Neuried');
```

Informieren Sie sich über den insert-Befehl.

AUF-03-15 Notieren Sie die wichtigsten Parameter und die Variante mit `set`.

Notiz:

Strukturen anzeigen:

- `show databases;`

```
+--------------------+
| Database           |
+--------------------+
| gm1                |
| gm2                |
| gm3                |
+--------------------+
```

- `show tables;`

```
+---------------+
| Tables_in_gm1 |
+---------------+
| abteilung     |
| gehalt        |
| mitarbeiter   |
| ort           |
+---------------+
```

- `desc ort;`

```
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| plz   | varchar(10) | YES  |     | NULL    |                |
| name  | varchar(50) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

SQL - Abfrage von Daten

Mit dem Befehl `SELECT` werden Tabellen abgefragt. Dieser ist der mächtigste und umfangreichste Befehl, den SQL zu bieten hat.

Die vereinfachte Ausführung sieht dabei wie folgt aus:

```
SELECT select_expr, ...
   [FROM table_references]
   [WHERE where_condition]
   [GROUP BY {col_name | expr | position}]
   [HAVING where_condition]
   [ORDER BY {col_name | expr | position} [ASC | DESC], ...]
   [LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

`FROM`	Legt die Tabellen fest, die in die Abfrage einbezogen werden.
`WHERE`	Legt die Bedingung für die Abfrage fest.
`GROUP BY`	Gruppiert die Abfrage.
`HAVING`	Definiert eine Bedingung, die für `GROUP BY` gilt.
`ORDER BY`	Legt die Sortierreihenfolge fest.
`LIMIT`	Legt fest, wie viel Zeilen ausgegeben werden.

Informieren Sie sich über den select-Befehl.

Beispiel

```
select * from ort order by name;

+----+-------+------------+
| id | plz   | name       |
+----+-------+------------+
|  1 | 85609 | Aschheim   |
|  2 | 85653 | Aying      |
|  3 | 85521 | Hohenbrunn |
|  4 | 85662 | Hohenbrunn |
|  5 | 82061 | Neuried    |
|  6 | 82131 | Neuried    |
+----+-------+------------+
```

Reihenfolge

Beim `SELECT`-Befehl ist wichtig zu wissen, in welcher Reihenfolge die einzelnen Bereiche abgearbeitet werden. Im ersten Schritt wird der gesamte Ausdruck von der SQL-Engine eingelesen. Danach ist die Reihenfolge wie folgt:

1. `FROM`
   Zuerst werden die Tabellen berücksichtigt. Ein dort definierter Alias ist in allen weiteren Bereichen gültig. Die Tabelle, die die meisten Datensätze enthält, sollte an erster Stelle stehen.
2. `WHERE`
   Danach wird die Datenmenge eingeschränkt. Sind dort mehrere Bedingungen enthalten, sollte die Bedingung an erster Stelle stehen, die am meisten Daten ausfiltert. Aliase aus `FROM` können hier verwendet werden.
3. `GROUP BY`
   Nun folgt die Gruppierung (wenn vorhanden).
4. `HAVING`
   Für die Gruppierung kann zusätzlich eine Bedingung gesetzt werden.
5. `Spalten`
   Jetzt werden die Spalten abgearbeitet. Ein dort definierter Alias kann erst danach verwendet werden.
6. `ORDER BY`
   Nun kann eine Sortierung der Spalten erfolgen. Dabei kann der Spaltenname, der Alias oder die Spaltennummer verwendet werden.
7. `LIMIT`
   Damit kann zuletzt die Ausgabe eingeschränkt werden.

Der `SELECT`-Befehl wird später bei den JOINs noch sehr ausführlich behandelt.

Notiz:

Daten ändern und löschen

Fügen Sie zuerst einen weiteren Datensatz hinzu:

```
insert into ort (plz,name) values ('99947','Unstrut-Hainich');

select * from ort;
+----+-------+-----------------+
| id | plz   | name            |
+----+-------+-----------------+
|  1 | 85609 | Aschheim        |
|  2 | 85653 | Aying           |
|  3 | 85521 | Hohenbrunn      |
|  4 | 85662 | Hohenbrunn      |
|  5 | 82061 | Neuried         |
|  6 | 82131 | Neuried         |
|  7 | 99947 | Unstrut-Hainich |
+----+-------+-----------------+
```

Bei der Eingabe wurde leider die falsche PLZ verwendet, diese muss nun geändert werden.

Mit dem update-Befehl werden Datensätze geändert.

```
update ort set plz='99986' where id=7;

select * from ort;
+----+-------+-----------------+
| id | plz   | name            |
+----+-------+-----------------+
|  1 | 85609 | Aschheim        |
|  2 | 85653 | Aying           |
|  3 | 85521 | Hohenbrunn      |
|  4 | 85662 | Hohenbrunn      |
|  5 | 82061 | Neuried         |
|  6 | 82131 | Neuried         |
|  7 | 99986 | Unstrut-Hainich |
+----+-------+-----------------+
```

Informieren Sie sich über den update-Befehl.

Notiz:

Nun kann der hinzugefügte Datensatz wieder gelöscht werden. Dies wird mit `delete` umgesetzt.

```
delete from ort where id=7;

select * from ort;
+----+-------+------------+
| id | plz   | name       |
+----+-------+------------+
|  1 | 85609 | Aschheim   |
|  2 | 85653 | Aying      |
|  3 | 85521 | Hohenbrunn |
|  4 | 85662 | Hohenbrunn |
|  5 | 82061 | Neuried    |
|  6 | 82131 | Neuried    |
+----+-------+------------+
```

Informieren Sie sich über den delete-Befehl.

Notiz: