



## 2. Pflichtübung

Ausarbeitung

*Von Algül, Mehmet*

*M'Sallem, Samir Faycal Tahar*

*Malik, Adeel*

*Tandogac, Ismet Enes*

*Tercüman, Mehmet Duhan*

---

*Gruppe 3.-7 Mittwoch 5. Block*

---

# Inhaltsverzeichnis

1	Einzelhandelsmarktkette .....	1
1.1	Vorwort.....	1
1.2	Überlegungen zur Modellierung .....	1
1.3	Von der Realität in ein Modell .....	2
1.4	ER-Diagramm .....	3
1.5	Umsetzung des Modells.....	4
1.5.1	Erfassung von Artikeln .....	4
1.5.2	Erfassung von Kundendaten.....	5
1.5.3	Erfassung der Filialen .....	5
1.5.4	Erfassung der Mitarbeiter .....	5
1.5.5	Erfassung der Einkäufe/Bons.....	5
1.6	Kritische Auseinandersetzung mit der Modellierung .....	7
1.7	Trigger und Stored Procedures .....	8
1.7.1	Trigger: .....	8
1.7.2	Stored Procedures: .....	8
1.8	SQL-Querys.....	9
2	Hochschul-Alumni-System.....	12
2.1	Thema: .....	12
2.2	Überlegungen:.....	12
2.3	Tabellen: .....	12
2.3.1	persoenliche_Daten.....	12
2.3.2	Private_Daten.....	13
2.3.3	Studium .....	13
2.3.4	Studiengang.....	13
2.3.5	Fachbereich .....	14
2.3.6	Mitarbeiter .....	14
<b>2.3.7</b>	Auslandsaufenthalts Daten .....	14
2.3.8	Preis .....	15
2.3.9	kompetenzprofil .....	15
2.3.10	Geschaefftliche_Daten.....	15
2.3.11	Abschluss.....	16
2.3.12	Fort_weiter_bildungsangebote .....	16
2.3.13	Inbox .....	16
2.3.14	Aufzeichnung.....	16

2.3.15	Studentenliste_1Semester .....	17
2.4	Trigger:.....	17
2.5	Stored Procedures: .....	17
2.6	ER-Diagramm .....	18
2.7	SQL-Abfragen: .....	19
2.8	Anmerkung und Kritik zur Umsetzung:.....	21
3	Paketversanddienst.....	23
3.1	Überlegung zur Modellierung .....	23
3.1.1	Skizze ERD:.....	23
3.2	phpMyAdmin Designer Ansicht.....	24
3.3	Beispiel Szenarien.....	24
3.4	Umsetzung des Modells.....	25
3.4.1	Parcel Express Shop .....	25
3.4.2	Mitarbeiter .....	26
3.4.3	Paketklassev1 .....	26
3.4.4	Paketklassen .....	26
3.4.5	Leistungen .....	26
3.4.6	Auftragv1.....	27
3.4.7	Kunde .....	27
3.4.8	Bezirk.....	27
3.4.9	Vergütungsgruppe .....	28
3.4.10	Sendungsstatus.....	28
3.4.11	Zusteller.....	28
3.4.12	Abteilung.....	28
3.4.13	Log_Pakete_zugestellt .....	29
3.4.14	Log_retoure .....	29
3.5	SQL – Abfragen .....	29
3.6	Trigger.....	32
3.7	Stored Procedure .....	32
3.8	Bemerkungen und Kritik an der Umsetzung .....	33
4	Pflegedienst .....	34
4.1	Grundlagen zur Modellierung: .....	34
4.1.1	Umsetzung des Modells:.....	34
4.2	PhpMyAdmin Designer Ansicht:.....	35
4.3	Grundstruktur: .....	35
4.3.1	Erfassung, Planung und Abrechnung: .....	36

4.4	SQL-Querys.....	37
4.4.1	<b>Gesucht wird welcher Patient die Grundpflege bekommt.....</b>	<b>37</b>
4.4.2	<b>Welcher Patient ist bei der Techniker Krankenversicherung .....</b>	<b>37</b>
4.4.3	<b>Welcher Mitarbeiter hat 5 Punkte oder weniger in der Evaluation .....</b>	<b>37</b>
4.4.4	<b>Alle Mitarbeiter und wie oft sie Frühschicht haben .....</b>	<b>37</b>
4.4.5	<b>Welches Fahrzeug wird nicht genutzt? .....</b>	<b>37</b>
4.4.6	<b>Welcher Patient wohnt in Bornheim .....</b>	<b>37</b>
4.4.7	<b>Welcher Mitarbeiter fährt einen Polo beim Einsatz .....</b>	<b>38</b>
4.4.8	<b>Gesamtbetrag der Abrechnung mit den Krankenkassen .....</b>	<b>38</b>
4.4.9	<b>Gesamtanzahl der Behandlungen für einen Patienten und deren Gesamtkosten .....</b>	<b>38</b>
4.4.10	<b>Welche Mitarbeiter in den ersten zwei Februarwochen 2021 Urlaub haben .....</b>	<b>38</b>
4.5	Trigger:.....	38
4.6	Stored Procedures: .....	39
4.7	Kritik und Ergänzung zur Umsetzung.....	39
4.8	ERD-Skizze: .....	40

# 1 Einzelhandelsmarktkette

## 1.1 Vorwort

Im Folgenden wird die Modellierung der in Aufgabe 1 zugrunde liegenden Einzelhandelsmarktkette beschrieben und durch eine Datenbank erarbeitet.

Diese soll Daten über ein Supermarktkette speichern, welches mehrere Filialen besitzt.

Die in der Aufgabenstellung aufgezeigte Modellierung ist nur grob, weshalb es notwendig ist, die Datenbank in ihrer Gesamtheit so zu unterteilen, dass Redundanzen und mögliche Komplikationen beseitigt werden können.

## 1.2 Überlegungen zur Modellierung

Grundlage dafür sind folgende Überlegungen:

- Es existieren n-viele Artikel, Mitarbeiter, Lieferanten, Kunden, Bons, Bon-Positionen und Filialen
- Jedem dieser Tupel wird eine ID zugeordnet
- Jeder Entitätstyp enthält neben diesem Primärschlüssel auch Stammdaten (Attribute)
- Es existieren Verknüpfungen und Abhängigkeiten zwischen diesen (identifizierend und nicht identifizierend)
- Informationen sollen nicht doppelt vorkommen und es soll auf Redundanzen verzichtet werden

### 1.3 Von der Realität in ein Modell

Was passiert, wenn ein Mitarbeiter austritt?

Was wenn ein Vertrag mit einem Lieferanten ausläuft?

Was wenn ein Preis von einem Artikel geändert wird und die Daten werden nachträglich in der Finanzabteilung benötigt?

Das hier zugrunde liegende Modell versucht auf diese Fragen eine Antwort zu finden und die damit verbundenen Komplikationen zu beseitigen. Die Constraints wurden daher so angesetzt, dass sie logisch schlüssige Operationen erlauben.

Wenn ein Mitarbeiter austritt, so wird in einem Bon der Mitarbeiter NULL sein. Dem Unternehmen ist es in unserer Modellierung nur wichtig, dass die Daten beispielsweise zur Meldung an das Finanzamt weiter existieren und eine Umsatzbildung ermöglicht bleibt.

Artikel, deren Lieferanten fortan nicht mehr existieren, weil ein Vertrag ausläuft, existieren weiterhin. Sie haben nach unserem Modell nur keinen Lieferanten mehr und können demnach temporär nicht mehr in die Filialen ausgeliefert werden. Letzteres wird aber nicht in der Datenbank behandelt, da zusätzliche Inventarlisten und Lieferanweisungen den Umfang der Datenbank sprengen würden.

Für die dritte Frage gibt es eine in der Datenbank implementierte Lösung. Das Abrechnungsdatum eines Bons ist bekannt, ebenso existiert eine Historie über Preisänderungen für jeden Artikel. Sollte also ein Endpreis in einem Bon berechnet werden, so könnte man über die Stichtage in der Preishistorie eines Artikels nachvollziehen, welchen tatsächlichen Preis der Artikel zu dem gegebenen Datum hatte.

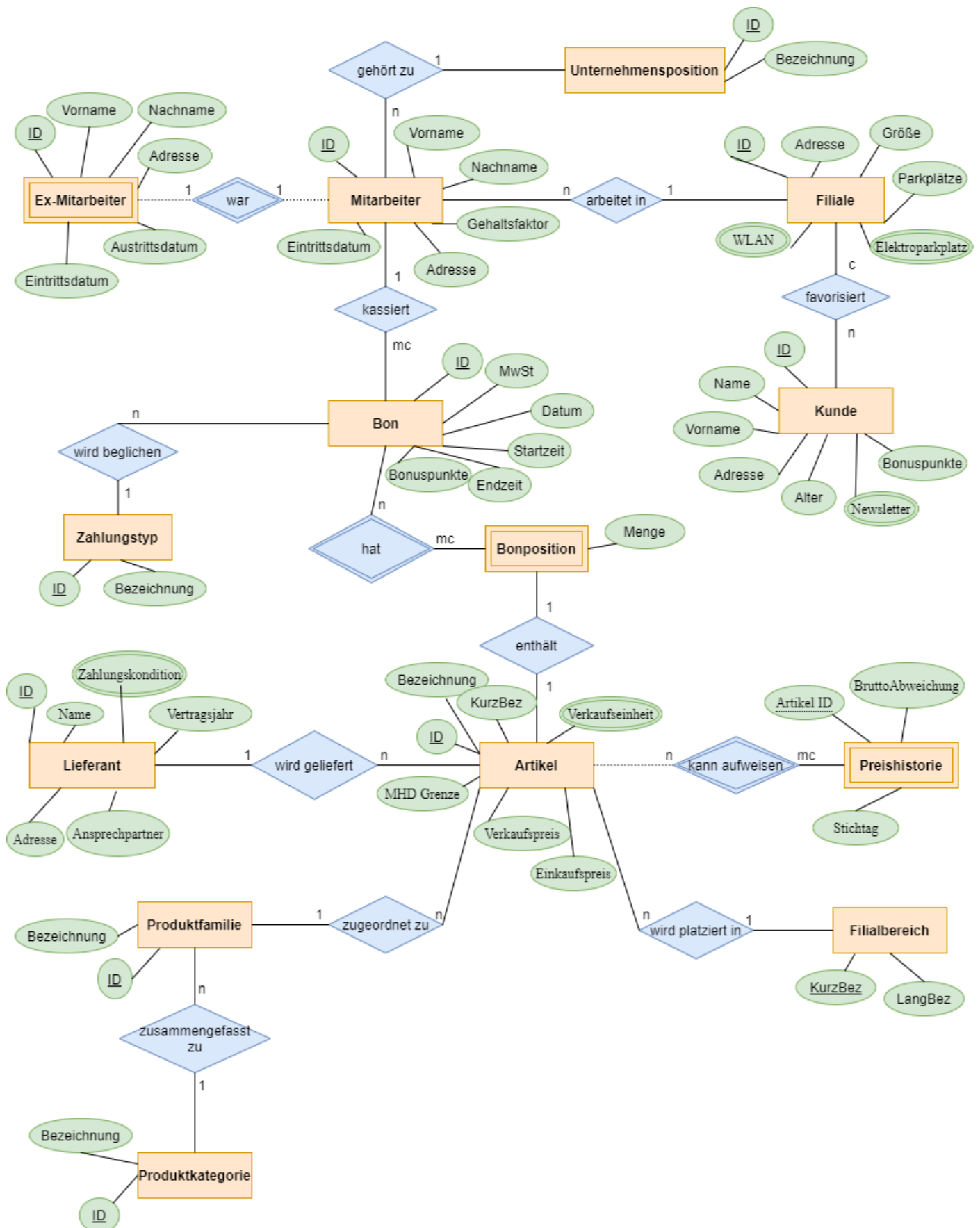
Bonuspunkte existieren im Modell zwar, es existiert aber keine Verknüpfung, an der man ausmachen könnte wie Kunden ihre Bonuspunkte einsetzen können bzw. wie sie sie erhalten. Eine Möglichkeit wäre folgende: Bonuspunkte können als Bezahlungsmethode verwendet werden, sofern die Zahlungsmethode „Bonuspunkte“ ausgewählt wird. Dann könnte man über einen Trigger eine Transaction vornehmen und den Punktestand ändern. Selbiges gilt für den Erhalt von Bonuspunkten nach einem Einkauf.

Für einen kommerziellen Einsatz wäre weiterhin die Definition diverser Views sinnvoll. Filialen können Stammdaten ihrer Mitarbeiter abrufen, allerdings nicht den Gehaltsfaktor, dieser bleibt nur der Personalabteilung abrufbar.

Auch könnte man damit auf jede Filiale die einzelnen Umsätze projizieren, welche dann an die Gesamt-Unternehmensstruktur wie in diesem Fall hier, weitergeleitet wird.

Generell wäre es sinnvoller eine Datenbank für jede Filiale anzulegen, deren grundlegende Daten dann an eine übergeordnete Unternehmensdatenbank weitergeleitet wird. Eine Filiale interessiert es sehr wohl zu wissen, welcher Artikel in welchem Bon verkauft wurde. Ein Unternehmen, welches mehrere Filialen überwacht ist eher daran interessiert, die Tagesumsätze zu erfahren und Statistiken über die gesamten Abverkaufszahlen zu erhalten.

## 1.4 ER-Diagramm



## 1.5 Umsetzung des Modells

Aus diesen Ausführungen kristallisiert sich eine Struktur heraus, welche via phpMyAdmin in die Datenbank eingearbeitet wurde. Die dort entstandenen Entitätstypen werden nun in ihrer Einzelheit beschrieben. **Entitätstypen welche dick gedruckt sind, enthalten für jede Entität einen eindeutigen Identifikator (ID).**

### 1.5.1 Erfassung von Artikeln

- **Artikel**
  - hat eine (Kurz- und Lang-) Bezeichnung
  - hat einen **Lieferanten**
  - hat einen **Filialbereich** in dem er in der Filiale platziert wird
  - wird einer **Produktfamilie** zugeordnet
  - besitzt eine Verkaufseinheit (in was wird er verkauft?) → Enums
  - hat ein MHD und eine daraus folgende Grenze wie lang der Artikel gelagert und verkauft werden darf
  - hat einen Einkaufs- und Verkaufspreis
- Preishistorie
  - Wenn ein Artikel eine Preisveränderung erfährt wird über einen Trigger diese Veränderung dokumentiert
  - Enthält die ID des Artikels, sowie die Veränderung in Euro und den Stichtag zu dem es geändert wurde
- **Produktfamilie**
  - Zugehörigkeit zu einer spezifischen Produktfamilie, etwa Milchprodukte, Weizenerzeugnisse, etc.
  - Eine Produktfamilie lässt sich darüber hinaus einer **Produktkategorie** zuordnen
  - Demnach sind Weizenerzeugnisse (etwa Brot) und Tierische Produkte (ein Stück Fleisch) zwar unterschiedliche Produktfamilien, sie eint aber die Tatsache, dass sie zur **Produktkategorie** Lebensmittel gehören
- **Produktkategorie**
  - Überbegriff der Zugehörigkeiten
  - Unterteilt nur grob in etwa Lebensmittel, Verbrauchsmaterial, Dienstleistung usw.
  - Nähere Unterteilung erfolgt dann in **Produktfamilie**
- **Filialbereich**
  - Hat eine Lang- und Kurzbezeichnung
  - Beispiele hierfür sind Non Food Bereich, MoPro (Molkereiprodukte), Troso (Trockensortiment)
  - diese Information wurde als unentbehrlich empfunden, da es für die Planung der Filiale notwendig ist zu wissen, wo ein Artikel zu platzieren ist und ob bei diesem etwa eine Kühlkette eingehalten werden muss
- **Lieferant**
  - Besitzt Stammdaten sowie Vertragsbedingungen und ein Vertragsjahr



#### 1.5.2 Erfassung von Kundendaten

- **Kunde**
  - Besitzt Stammdaten sowie eine **Stammfiliale** und ein Bonuspunktekonto, welches durch jeden Einkauf gefüllt werden soll

#### 1.5.3 Erfassung der Filialen

- **Filiale**
  - Besitzt Stammdaten über deren Ort, Größe und Beschaffenheit

#### 1.5.4 Erfassung der Mitarbeiter

- **Mitarbeiter**
  - Besitzt Stammdaten
  - Hat eine **Position im Unternehmen**
  - Kann einer Filiale zugeordnet werden (in der er arbeitet)
- **Unternehmensposition**
  - Bezeichnung der Stelle die ausgeführt wird
- **Ex-Mitarbeiter**
  - Sobald ein Mitarbeiter das Unternehmen verlässt wird durch einen weiteren Trigger seine Informationen, sowie sein Austrittsdatum in dieser Tabelle vermerkt.
  - So kann gewährleistet werden, dass überflüssige Daten in der Mitarbeiter Tabelle vermieden werden und gleichzeitig behält man Kontrolle über die Daten

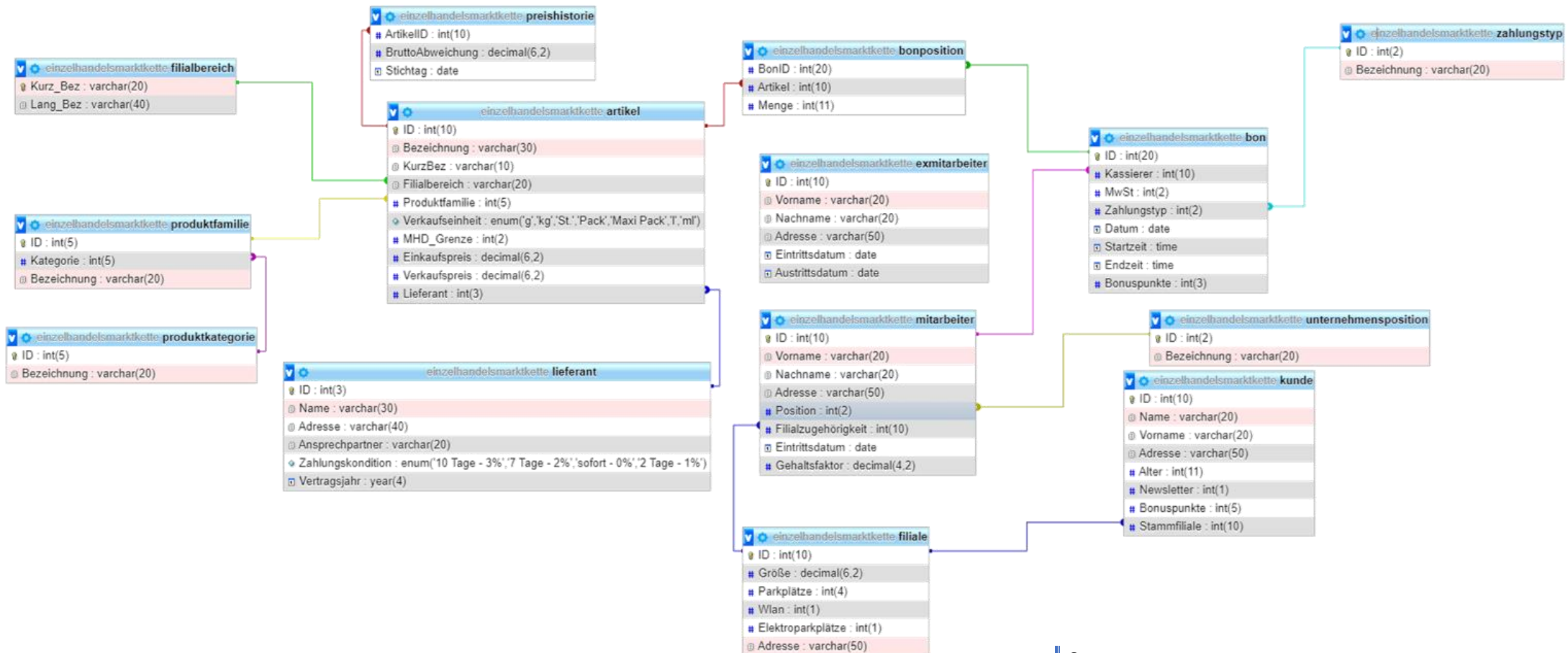
#### 1.5.5 Erfassung der Einkäufe/Bons

- **Bon**
  - Wird von einem **Mitarbeiter** in einer Filiale kassiert und erfasst
  - Über Mitarbeiter wird Filiale ermittelt: Bon -> kassiert von Mitarbeiter -> arbeitet in Filiale X
  - Er hat eine MwSt., welche auf den Gesamtpreis draufgerechnet werden muss
  - Ein Bon wird über eine **Zahlungsmethode** beglichen
  - Für jeden Bon werden Bonuspunkte verteilt
  - Ein Bon hat einen Zeitstempel
- **Bon Position**
  - Enthält eine Position (**Artikel**, Menge) welche ein Teil eines **Bons** bildet
  - Zeigt auf einen **Bon**
- **Zahlungstyp**
  - Zahlungstypen die während der Transaktion mittels eines **Bons** verwendet werden können

Ein Verständnis der Relationen liefert ebenfalls das Beziehungsschema aus phpMyAdmin. Dabei seien die Verknüpfungspunkte welche eine Pfeilspitze enthalten, die Ausgangspunkte, der Fremdschlüssel, welche auf einen Primärschlüssel der jeweiligen Tabelle zeigen.

Im obigen Beispiel bedeutet dies, dass das Attribut BonID in der Tabelle bonposition, eine ID aus der Tabelle bon referenziert. Somit geht eine Fremdschlüsselbeziehung von bonposition.BonID auf bon.ID aus. Jede Linie repräsentiert folglich eine Fremdschlüsselbeziehung.

Daraus ergibt sich dann folgendes Relationenmodell:



## 1.6 Kritische Auseinandersetzung mit der Modellierung

Ziel unseres Vorhabens war, eine sinnvolle und schlüssige Modellierung zu erstellen. Diese sollte sowohl die Zwecke der Normalisierung einhalten, als auch eine referenzielle Integrität gewährleisten, sodass Abläufe innerhalb einer solchen imaginären Einzelhandelsmarktkette auch noch dann funktionieren, wenn beispielsweise ein Mitarbeiter das Unternehmen verlässt.

Die erste Vorgehensweise war, Informationen die mehrfach vorkommen in eine eigene Tabelle auszulagern. So wurden etwa Produktfamilien und Produktkategorien in eigenen Tabellen angelegt und mit einem Primärschlüssel versehen. Des Weiteren wurde einem Bon eine ID zugewiesen. Anschließend wurde in einer gesonderten Tabelle Bonposition, lediglich die durch einen Fremdschlüssel referenzierte ID angegeben, sowie der Artikel und die dazugehörige Menge.

Da das vorliegende Modell bereits sehr umfangreich ist und eine sehr hohe Skalierbarkeit aufweist, wurde auf Punkte wie etwa ein Lagerbestand für jede Filiale verzichtet. Nichtsdestotrotz erfüllt das Modell alle Anforderungen und speichert Daten effizient und ohne Redundanzen.

Die Trigger und Stored Procedures haben eine deutliche Sinnhaftigkeit und sind realitätsnah. Diese würden auch in der Realität in einem solchen Modell benötigt werden.

Eine Stärke des Modells ist definitiv, dass sehr viele Informationen gespeichert werden und sich daraus ein enormer Mehrgewinn generieren lässt.

Wie aus den SQL-Querys zu entnehmen ist, ist diese Datenbank bei richtiger Einführung in ein geeignetes Unternehmen eine sehr große Datenmacht, da sie so gut wie alles speichert.

Müsste man eine Schwäche von diesem Modell ausmachen, so würden wir festmachen, dass in diesem Modell keine Kontrolle über den Fluss von Waren herrscht. Zwar existieren zu jedem Bon detaillierte Informationen, welcher Artikel wie oft, zu welchem Preis, in welcher Filiale verkauft wurde, allerdings fehlt der logistische Überblick dahinter, inwieweit die Waren nach einem Einkauf durch einen Kunden wieder neu geliefert werden müssten oder welcher Artikel überhaupt wie oft existiert.

Zuletzt soll aber betont werden, dass diese Kritikpunkte lediglich eine Erweiterung des Modells darstellen. Das Modell in seiner jetzigen Form ist performant und schlüssig, alles Weitere darüber hinaus wäre nur eine gelungene Erweiterung zu einer an sich selbst gelungenen Schnittstelle.

## 1.7 Trigger und Stored Procedures

### 1.7.1 Trigger:

- Mitarbeiter werden bei Löschung in Ex-Mitarbeiter Liste verschoben und Austrittsdatum wird dokumentiert
  - Erstellt dafür wurde die Tabelle Ex-Mitarbeiter
  - ```
CREATE TRIGGER `VerschiebeinExMitarbeiter` BEFORE DELETE ON `mitarbeiter` FOR EACH ROW INSERT INTO exmitarbeiter VALUES (OLD.ID, OLD.Vorname, OLD.Nachname, OLD.Adresse, OLD.Eintrittsdatum, CURRENT_DATE)
```
- Preisveränderung der Artikel wird dokumentiert mit Preisveränderung und Stichtag
  - Hierfür wurde die Tabelle Preishistorie angelegt
  - ```
CREATE TRIGGER `NotierePreisveraenderung` AFTER UPDATE ON `artikel` FOR EACH ROW INSERT INTO preishistorie VALUES(NEW.id, NEW.Einkaufspreis-OLD.Einkaufspreis, CURRENT_DATE)
```

### 1.7.2 Stored Procedures:

- MwSt. Anpassung durch Corona, alle Bons vom 01.07 bis 31.12 erhalten statt 19% nur 16% MwSt.
  - ```
CREATE PROCEDURE `MwstAnpassungCorona`() COMMENT 'Setzt die MwSt auf 16% aufgrund von Corona in der Zeit 1.7-31.12' NOT DETERMINISTIC MODIFIES SQL DATA SQL SECURITY DEFINER UPDATE bon SET MwSt = 16 WHERE Datum > '2020-06-30' AND Datum < '2021-01-01'
```
- Verkaufspreis Berechnung: Kalkulation durch Einkaufspreis \* Faktor (abhängig von Produktfamilie)
  - ```
CREATE PROCEDURE `VerkaufspreisausEinkaufspreis`() COMMENT 'Setze Verkaufspreis durch Einkaufspreis mit imaginärem Faktor' DETERMINISTIC MODIFIES SQL DATA SQL SECURITY DEFINER UPDATE artikel SET Verkaufspreis = CASE produktfamilie  
WHEN 1 THEN Einkaufspreis * 1.4  
WHEN 2 THEN Einkaufspreis * 1.4  
WHEN 3 THEN Einkaufspreis * 4.0  
WHEN 4 THEN Einkaufspreis * 2.0  
WHEN 6 THEN Einkaufspreis * 1.2  
WHEN 8 THEN Einkaufspreis * 2.0  
WHEN 9 THEN Einkaufspreis * 1.3  
WHEN 17 THEN Einkaufspreis * 2.6  
WHEN 18 THEN Einkaufspreis * 1.4  
ELSE Einkaufspreis END
```

## 1.8 SQL-Querys

- Um neue Mitarbeiter besser einer Filiale zuzuordnen, soll überprüft werden, ob die Filiale einen Bedarf an neuen Mitarbeitern hat. Gefragt ist nun, ob die Filiale von Herrn Krüger Unterstützung braucht, indem die Anzahl der Mitarbeiter ermittelt wird

- SELECT COUNT(mitarbeiter.ID) as Anzahl\_Mitarbeiter from mitarbeiter WHERE mitarbeiter.Filialzugehörigkeit = (SELECT filiale.ID FROM mitarbeiter, filiale, unternehmensposition WHERE mitarbeiter.Nachname='Krüger' AND filiale.ID = mitarbeiter.Filialzugehörigkeit AND mitarbeiter.Position=unternehmensposition.ID AND unternehmensposition.Bezeichnung = 'Filialleiter')

Anzahl_Mitarbeiter
2

- Liste aller **Lebensmittel**, welche im Laden verkauft werden

- SELECT artikel.Bezeichnung, artikel.Verkaufspreis, produktfamilie.Bezeichnung from artikel, produktfamilie, produktkategorie WHERE artikel.Produktfamilie = produktfamilie.ID AND produktfamilie.Kategorie = produktkategorie.ID AND produktkategorie.Bezeichnung='Lebensmittel'

Bezeichnung	Verkaufspreis <small>Einkaufspreis * Faktor (Procedure regelt dies)</small>	Bezeichnung
Joghurt Müller 300g	0.70	Milchprodukt
Käse 150g Aufschnitt	0.92	Milchprodukt
Milch Laktosefrei	0.17	Milchprodukt
Kaiserbrötchen	0.04	Weizenerzeugnis
TBone Steak	62.68	Tierisches Produkt
Kabeljau Stück	19.52	Tierisches Produkt
Bodenhaltungs Eier 12 Stück	1.36	Tierisches Produkt
Radieschen	0.22	Obst/Gemüse
Esskastanien	5.49	Obst/Gemüse
Veganer Burgerpatty 200g	4.85	Obst/Gemüse
Mineralwasser still 1,5l	0.03	Getränk
Orangensaft Valensina	1.22	Getränk

- Um eine Rundmail an alle Filialleiter zu schicken gebe die Liste aller Filialen mit deren Filialleiter (Vorname und Nachname in einer Spalte) aus.

- SELECT filiale.Adresse, CONCAT(mitarbeiter.Vorname, " ", mitarbeiter.Nachname) AS Filialleiter FROM filiale, mitarbeiter, unternehmensposition WHERE filiale.ID = mitarbeiter.Filialzugehörigkeit AND mitarbeiter.Position=unternehmensposition.ID AND unternehmensposition.Bezeichnung='Filialleiter' GROUP BY filiale.ID

Adresse	Filialleiter
Leipziger Straße 94 34466 Wolfhagen	Jens Ostermann
Heiligengeistbrücke 23 91710 Gunzenhausen	Alfred Mayer
Jenaer Strasse 27 45470 Mülheim an der Ruhr	Frank Schröder
Ruschestrasse 1 19222 Hagenow	Leon Klug
Reeperbahn 97 18203 Bad Doberan	Misbah Munib Kouri
Rathausstrasse 43 90015 Nürnberg	Tim Krüger

4. Ein Kunde hat sich über den Preis des Artikels „Joghurt Müller 300g“ beschwert. Gebe die Preisveränderung aus, um zu prüfen ob dies gerechtfertigt ist.

- SELECT preishistorie.BruttoAbweichung+artikel.Verkaufspreis AS Preis, preishistorie.Stichtag FROM preishistorie, artikel WHERE artikel.ID=preishistorie.ArtikelID AND artikel.Bezeichnung = 'Joghurt Müller 300g'

Preis	Stichtag
0.75	2020-12-26
0.60	2020-12-26
0.60	2020-12-26
0.70	2020-12-27

5. Günstigster und teuerster Artikel, welcher in Gramm verkauft wird (also Verkaufseinheit ,g‘)

- SELECT artikel.Bezeichnung, artikel.Verkaufspreis, 'Teuerster Artikel' AS Eigenschaft from artikel WHERE artikel.Verkaufseinheit='g' AND artikel.Verkaufspreis = (SELECT MAX(artikel.Verkaufspreis) from artikel WHERE artikel.Verkaufseinheit='g') UNION SELECT artikel.Bezeichnung, artikel.Verkaufspreis, 'Günstigster Artikel' AS Eigenschaft from artikel WHERE artikel.Verkaufseinheit='g' AND artikel.Verkaufspreis = (SELECT MIN(artikel.Verkaufspreis) from artikel WHERE artikel.Verkaufseinheit='g')

Bezeichnung	Verkaufspreis	Eigenschaft
TBone Steak	62.68	Teuerster Artikel
Kabeljau Stück	19.52	Günstigster Artikel

6. Filiale, welche sowohl WLAN, als auch Elektroparkplätze bietet und mindestens 1000qm Verkaufsfläche hat

- SELECT id, Adresse FROM filiale WHERE Wlan=1 AND Elektroparkplätze=1 AND Größe>=1000

id	Adresse
1	Leipziger Straße 94 34466 Wolfhagen

7. Bons, welche in der Filiale in Gunzenhausen abgewickelt wurden mit Bonnr., vollständigem Mitarbeiter Name, Adresse der Filiale und Details zum Bon

- SELECT bon.ID AS Bon, CONCAT(mitarbeiter.Vorname, " ", mitarbeiter.Nachname) AS Mitarbeiter , filiale.ID AS Filiale, filiale.Adresse, bon.Datum, bon.Startzeit, bon.Zahlungstyp from mitarbeiter, bon, filiale WHERE bon.Kassierer=mitarbeiter.ID AND filiale.ID=mitarbeiter.Filialzugehörigkeit AND filiale.Adresse LIKE '%Gunzenhausen%'

Bon	Mitarbeiter	Filiale	Adresse	Datum	Startzeit	Zahlungstyp
6	Alfred Mayer	2	Heiligengeistbrücke 23 91710 Gunzenhausen	2020-08-18	19:14:16	2
7	Mehmet Tercüman	2	Heiligengeistbrücke 23 91710 Gunzenhausen	2010-09-19	11:16:36	4

8. Um in Zukunft mehr Klopapier in Filialen zu liefern, in denen Hamsterkäufer unterwegs sind, sollen die Bons mit Filiale, Mitarbeiter und Datum ausgegeben werden, wenn über einen Bon mehr als 3 Packungen Toilettenpapier gekauft wurden und der Bon 2020 erfasst wurde.

- SELECT bon.ID, mitarbeiter.Nachname, filiale.Adresse, bon.Datum from bon, artikel, bonposition, filiale, mitarbeiter WHERE bonposition.BonID = bon.ID AND bonposition.Artikel = artikel.ID AND bon.Kassierer = mitarbeiter.ID AND mitarbeiter.Filialzugehörigkeit=filiale.ID AND artikel.Bezeichnung LIKE '%Toilettenpapier%' AND bonposition.Menge >= 3 AND bon.Datum > '2019-12-31'

ID	Nachname	Adresse	Datum
4	Klug	Ruschestrasse 1 19222 Hagenow	2020-12-09

9. Um zu überprüfen ob Bargeldtransporte aus Filialen öfter benötigt werden als bisher, soll eine Statistik angezeigt werden, die Aufschluss über jede verwendete Zahlungsart und deren Verwendungsanzahl liefert.

- SELECT zahlungstyp.Bezeichnung, COUNT(bon.Zahlungstyp) as 'Anzahl Nutzung' from bon, Zahlungstyp WHERE Zahlungstyp.ID=bon.Zahlungstyp GROUP BY Zahlungstyp.ID

Bezeichnung	Anzahl Nutzung
Barzahlung	2
Guthabekarte	1
Girokonto	1
Kreditkarte	3
Gutschrift	2

10. Ein Kunde möchte eine Packung Milch reklamieren, da diese offenbar schlecht ist. Leider hat er keinen Kassenbon mehr. Er kann sich nur daran erinnern, dass er in der Filiale mit der PLZ 18203 am 18.12.2020 einkaufen war und noch ein Bund Radieschen eingekauft hat. Sollte ein Bon mit diesen Eckdaten existieren, so wird die Milch kulanterweise gegen eine Neue eingetauscht.

- SELECT bon.ID, bon.Datum from bon, filiale, mitarbeiter, bonposition, artikel WHERE bon.ID=bonposition.BonID AND bon.Kassierer=mitarbeiter.ID AND mitarbeiter.Filialzugehörigkeit = filiale.ID AND bonposition.Artikel = artikel.ID AND filiale.Adresse LIKE '%18203%' AND bon.Datum='2020-12-18' AND artikel.Bezeichnung LIKE '%Radieschen%' GROUP BY bon.ID

ID	Datum
10	2020-12-18



## 2 Hochschul-Alumni-System

### 2.1 Thema:

Für immer mehr Hochschulen nimmt der Kontakt zu ihren „Ehemaligen“ (Absolventen, ehemalige Mitarbeiter) einen hohen Stellenwert ein, u.a. als potenzielle Förderer. Nachfolgend wird eine Datenbank für den Aufbau eines Ehemaligen-Netzwerkes an ihrer Hochschule erstellt, wobei das sog. Alumni-System u.a. folgende Daten erfassen soll:

- Persönliche Daten
- Private Daten
- Hochschulaufenthaltsdaten
- Auslandsaufenthaltsdaten
- Geschäftliche Daten
- Kompetenzprofil

### 2.2 Überlegungen:

- Jede Person hat immer persönliche Daten (mit einer ID)
- Alle anderen Informationen referenzieren zu der ID der jeweiligen Person
- Hochschulaufenthaltsdaten haben wir in 2 Tabellen geteilt, da es mehr Sinn machen würde und der Übersichtlichkeit dient
- Umsortieren der Inhalte aus persönlichen Daten zu privaten Daten (Passwort)
- Studium näher definiert (Studiengang->Fachbereich->Abschlussart)
- Um einem Alumnus über ein Weiterbildungsangebot zu informieren braucht man eine separate Tabelle (Inbox)
- Orientierung der Daten an der THM (Fachbereich, Abschlussarten, Studiengänge, etc.)

### 2.3 Tabellen:

#### 2.3.1 persoenliche Daten

- ID (PK)
- Name
- Titel
- Vorname
- Ggf. Geburtsname
- Nationalität
- EhemaligenStatus
- E-Mail
- Benutzername

ID	Name	Titel	Vorname	Geburtsname	Nationalität	Ehemaligen_Status	E_Mail	Benutzername
1	Tandogac	NULL	Enes	Ismet	Deutsch	Student	sonmez2748@gmail.com	ietn20
2	Salem	Dr	Samir	NULL	Deutsch	Student	samirbaba@gmail.com	smsl30

Die ID hat hierbei einen PRIMARY KEY da jeder nur einmal persönliche Daten haben kann



### 2.3.2 Private Daten

- ID (PK, FK)
- Adresse
- Telefon
- Mobile
- Website
- Geburtstag
- passwort

Die ID hat auch hier einen PRIMARY KEY da jeder nur einmal private Daten haben kann. Auch referenziert die private\_Daten.ID auf die ID von persoenliche\_Daten da diese Tabelle nur als eine Ergänzung von Daten einer Person gilt.

id	Adresse	Telefon	Mobile	Web_Site	Geburtstag	Passwort
1	Schottstraße 4	069 48448800	0176 45683755		2001-01-03	48448800
2	Ringallee 27	069 4562273	0175 52289651	saalemshop.com	2000-06-23	SagIchNicht

### 2.3.3 Studium

- ID (PK)
- PersonID (FK)
- Studiengang (FK)
- Studiumbeginn
- Studiumende
- benoetigteSemester

Die ID dient dazu, um einen bestmöglichen und eindeutigen Zugriff auf die einzelnen Zeilen zu haben, dies erleichtert somit die Bearbeitung der Daten.

PersonID referenziert zur persoenliche\_Daten.ID.

Studiengang referenziert zur Tabelle Studiengang.ID.

benoetigteSemster sagt aus wie viele Semester der Student für diesen Studiengang gebraucht hat.

ID	PersonID	Studiengang	Studiumbeginn	Studiumende	benoetigteSemester
1	1	1	2019-10-01	2023-04-22	7
2	2	1	2019-10-01	2022-09-30	6

### 2.3.4 Studiengang

- ID (PK)
- Studiengang
- Fachbereich (FK)
- Regelstudienzeit
- AbschlussNr (FK)

ID	Studiengang	Fachbereich	Regelsudienzeit	AbschlussNr
1	Informatik	1	6	1
2	Ingenieur-Insformatik	1	6	1

Die ID dient dazu, um einen bestmöglichen und eindeutigen Zugriff auf die einzelnen Zeilen zu haben, dies erleichtert somit die Bearbeitung der Daten.

Fachbereich referenziert zur Spalte Fachbereich.ID der Tabelle Fachbereich.

Regelstudienzeit sagt aus wie lange man für das jeweilige Studiengang „in der Regel“ brauchen würde.

### 2.3.5 Fachbereich

- ID (PK)
- Kürzel
- Name

Die ID dient dazu, um einen bestmöglichen und eindeutigen Zugriff auf die einzelnen Zeilen zu haben, dies erleichtert somit die Bearbeitung der Daten.

ID	Kürzel	Name
1	MNI	Mathematik, Naturwissenschaften und Informatik
2	BAU	Bauwesen

### 2.3.6 Mitarbeiter

- ID (PK)
- PID (FK)
- Mitarbeiter\_als
- Mitarbeiter\_bei
- Mitarbeiterbeginn
- Mitarbeiterende
- Weitere\_Taetigkeiten

ID	PID	Mitarbeiter_als	Mitarbeiter_bei	Mitarbeiterbeginn	Mitarbeiterende	weitere_Taetigkeiten
1	3	Auslandsbeauftragte	International Office	1999-04-03	2017-10-21	Unterstützt die ins Ausland gehenden Studenten (Ou...

Die ID dient dazu, um einen bestmöglichen und eindeutigen Zugriff auf die einzelnen Zeilen zu haben, dies erleichtert somit die Bearbeitung der Daten.

Die PID referenziert zur persoenliche\_Daten.ID.

### 2.3.7 Auslandsaufenthalts Daten

- ID (PK)
- PID (FK)
- Hochschule
- Firma
- Land
- Taetigkeit
- Beginn
- Ende

ID	PID	Hochschule	Firma	Land	Taetigkeit	Beginn	Ende
1	1	Istanbul Ege Üniversitesi	NULL	Türkei	Auslandssemester	2021-10-01	2021-01-03
2	7	University of Massachusetts System	NULL	USA	Auslandssemester	2022-08-12	2023-01-05

Die ID dient dazu, um einen bestmöglichen und eindeutigen Zugriff auf die einzelnen Zeilen zu haben, dies erleichtert somit die Bearbeitung der Daten.

Die PID referenziert zur personenliche\_Daten.ID.

Falls ein Student ein Auslandsaufenthalt an einer Hochschule verbracht hat dann wird nur die spalte hochschule ausgefüllt, wenn er aber an einer Firma war dann wird die Spalte hochschule frei gelassen und dafür die Spalte Firma ausgefüllt.

### 2.3.8 Preis

- ID (PK)
- PID (FK)
- Preis
- Datum\_der\_Verleihung

ID	PID	Preis	Datum_der_Verleihung
1	1	500.00	2021-06-03
2	8	500.00	2021-06-03

Die ID dient dazu, um einen bestmöglichen und eindeutigen Zugriff auf die einzelnen Zeilen zu haben, dies erleichtert somit die Bearbeitung der Daten.

Die PID referenziert zur personenliche\_Daten.ID.

### 2.3.9 kompetenzprofil

- ID (PK)
- PID (FK)
- Ich
- Sprachen
- Faehigkeiten
- Interessen

ID	PID	Ich	Sprachen	Faehigkeiten	Interessen
1	4	Suche	Deutsch	Architekt	

Die ID dient dazu, um einen bestmöglichen und eindeutigen Zugriff auf die einzelnen Zeilen zu haben, dies erleichtert somit die Bearbeitung der Daten.

Die PID referenziert zur personenliche\_Daten.ID.

Bei der spalte kann der User erläutern was er will (ob er z.B. etwas ‚Sucht‘ oder etwas ‚Bietet‘) wodurch die Zeile einen entscheidenden Sinn erhält

### 2.3.10 Geschaeftliche Daten

- ID (PK)
- PID (FK)
- Arbeitgeber
- Abteilung
- Branche
- Taetigkeit
- Adresse
- E\_Mail
- Web\_Site
- Arbeitet\_seit
- Gearbeitet\_bis

ID	PID	Arbeitgeber	Abteilung	Branche	Taetigkeit	Adresse	E-Mail	Web_Site	Arbeits_seit	gerArbeits_bis
1	2	Google	Softwareentwicklung	IT	Softwareentwicklung	Barrow Street Dublin 4	support-deutschland@google.com	https://careers.google.com/jobs/results/	2023-08-15	2028-08-24
2	7	Google	Softwareentwicklung	IT	Softwareentwicklung	Barrow Street Dublin 4	support-deutschland@google.com	https://careers.google.com/jobs/results/	2023-12-27	2032-11-04

Die ID dient dazu, um einen bestmöglichen und eindeutigen Zugriff auf die einzelnen Zeilen zu haben, dies erleichtert somit die Bearbeitung der Daten.

Die PID referenziert zur personenliche\_Daten.ID.

### 2.3.11 Abschluss

- ID (PK)
- Akademischer\_Grad

ID	Akademischer_Grad
1	Bachelor of Science
2	Bachelor of Arts

Die ID dient dazu, um einen bestmöglichen und eindeutigen Zugriff auf die einzelnen Zeilen zu haben, dies erleichtert somit die Bearbeitung der Daten.

Die Tabelle dient als eine Legende, womit später bei Veränderungen nicht jede Zeile einzeln verändert werden muss bei den Tabellen, die auf diese Tabelle referenzieren.

### 2.3.12 Fort weiter bildungsangebote

- ID (PK)
- Thema
- Empfohlen\_fuer (FK)

ID	Thema	Empfohlen_fuer
1	Top 10 Programmiersprachen 2021	1
2	Weltarchitektur	2
3	Den Menschen Arbeit abnehmen?	3

Die ID dient dazu, um einen bestmöglichen und eindeutigen Zugriff auf die einzelnen Zeilen zu haben, dies erleichtert somit die Bearbeitung der Daten.

Empfohlen\_fuer referenziert zu Fachbereich.ID

In diese Tabelle kommen Weiterbildungsangebote, die jeweils zu einem Fachbereich passen.

### 2.3.13 Inbox

- ID (PK)
- PID (FK)
- Empfohlene Weiterbildungsangebote (FK)

ID	PID	Empfohlene Weiterbildungsangebote
1	1	1

Die ID dient dazu, um einen bestmöglichen und eindeutigen Zugriff auf die einzelnen Zeilen zu haben, dies erleichtert somit die Bearbeitung der Daten.

Die PID referenziert zur personenliche\_Daten.ID.

Empfohlene Weiterbildungsangebote referenziert zu Fort\_weiter\_bildungsangebote.ID.

In dieser Tabelle werden alle Weiterbildungsangebote den passenden Absolventen zugeteilt.

### 2.3.14 Aufzeichnung

- ID (PK)
- Tabelle
- PID (FK)
- WURDE
- Datum

ID	Tabelle	PID	WURDE	Datum
3		9	AKTUALISIERT	2021-01-01 23:18:57
4		9	AKTUALISIERT	2021-01-01 23:19:00
5	persoenliche_daten	9	AKTUALISIERT	2021-01-02 19:21:20

Die ID dient dazu, um einen bestmöglichen und eindeutigen Zugriff auf die einzelnen Zeilen zu haben, dies erleichtert somit die Bearbeitung der Daten.

Die PID referenziert zur `persoenliche_Daten.ID`.

In diese Tabelle werden die Einträge durch Trigger automatisch eingetragen. Sie dient der Überwachung der Datenveränderung und gibt auch die Aktivität der User wieder. Sie sagt aus wann bei wem in welcher Tabelle was gemacht wurde.

### 2.3.15 Studentenliste\_1Semester

- Matrikelnummer (PK)
- Name
- Nachname
- Fachbereich (FK)

Matrikelnummer	Name	Nachname	Fachbereich
1274321	Marlene	Lambrecht	6
2345521	Alina	Schepers	2
3388291	Ahmet	Gursoy	4
3452143	Lara	Nehls	6
3454321	Liam	Fischer	1

Die ID dient dazu, um einen bestmöglichen und eindeutigen Zugriff auf die einzelnen Zeilen zu haben, dies erleichtert somit die Bearbeitung der Daten.

Die Fachbereich referenziert zur `Fachbereich.ID`.

Diese Tabelle präsentiert alle jetzigen Studenten welche nur noch 1 Semester vor ihrem Abschluss haben mit der Information in welchem Fachbereich ihr Studiengang liegt.

## 2.4 Trigger:

Wie oben erwähnt wird die Tabelle aufzeichnung durch Trigger automatisch mit Daten befüllt.

In den Tabellen (auslandsaufenthaltstitel, geschaeftliche\_daten, inbox, kompetenzprofil, mitarbeiter, persoenliche\_daten, preis, private\_daten, studium) befinden sich je diese 3 Trigger:

Ein Insert Trigger: AFTER INSERT -> `INSERT INTO aufzeichnung VALUES(null,'tabellenname', new.pid, 'HINZUGEFÜGT', now())`

Ein Update Trigger: AFTER UPDATE -> `INSERT INTO aufzeichnung VALUES(null,'tabellenname', new.pid, 'AKTUALISIERT', now())`

Ein Delete Trigger: AFTER DELETE -> `INSERT INTO aufzeichnung VALUES(null,'tabellenname', old.pid, 'GELÖSCHT', now())`

## 2.5 Stored Procedures:

### Anwendung:

Der User kann, wenn er als Mentor für einen Stunden fungieren will diesen Stored Procedures anwenden. Er gibt dann in das Input Feld die ID seines Fachbereichs an, wodurch dann alle Studenten die nur noch 1 Semester vor ihrem Abschluss haben angezeigt welche auch aus demselben Fachbereich sind wie der User. Dabei muss in Beachtung genommen werden das es nur Fachbereiche gibt, welche die ID's von 1 bis 7 haben.

```
CREATE PROCEDURE Mentoring(
    IN FID int
)
BEGIN
    SELECT *
    FROM studentenliste_1Semester
    WHERE fachbereich = FID;
EN
```

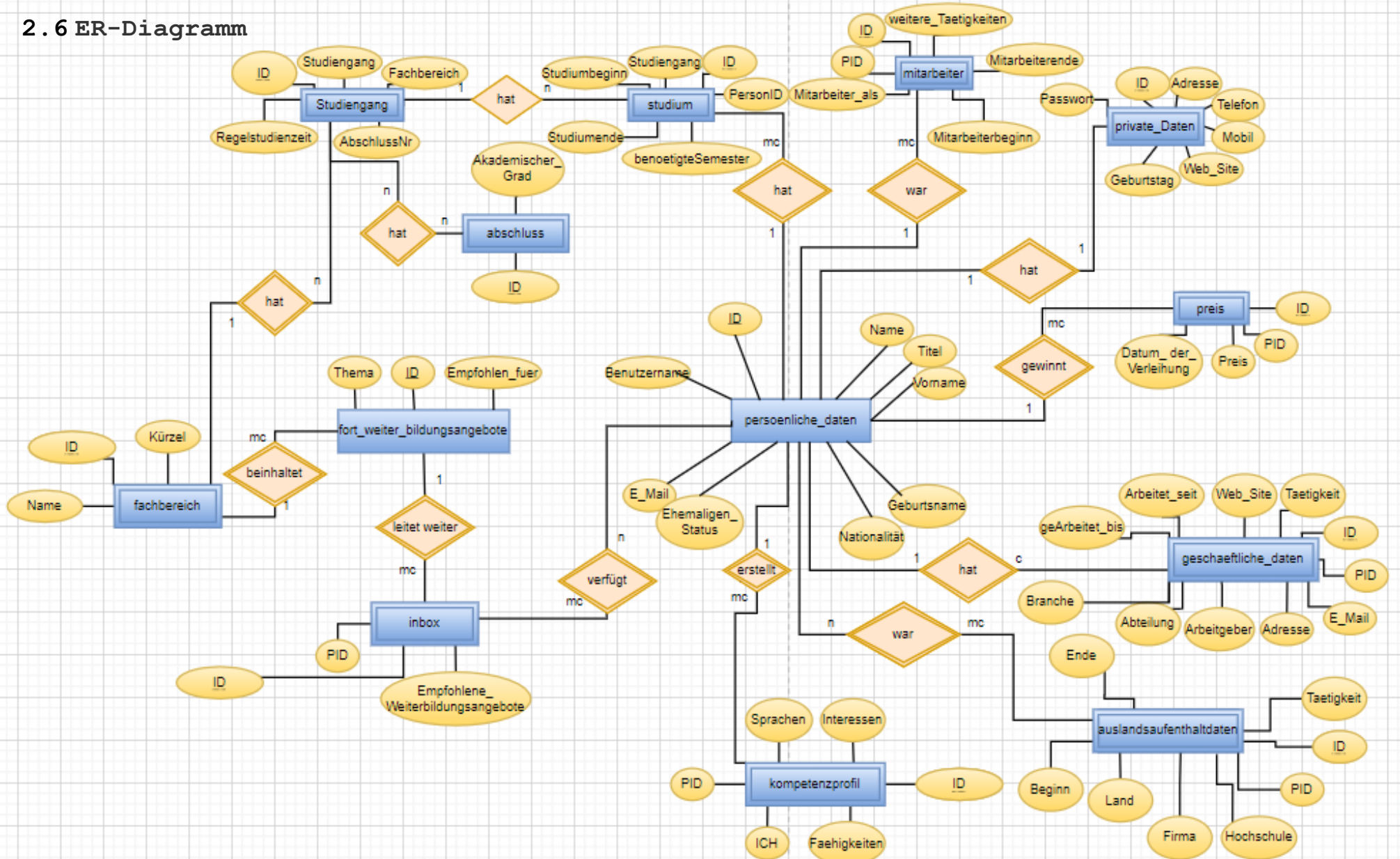
Bsp.:

The screenshot shows a window titled "Führe Prozedur aus 'Mentoring'". Inside, there is a table-like structure for "Prozeduren-Parameter". It has four columns: "Name", "Typ", "Funktion", and "Wert". The first row shows "FID" as the name, "INT" as the type, and "1" as the value. There are "OK" and "Schließen" buttons at the bottom right.

Beispielergbnis:

Matrikelnummer	Name	Nachname	Fachbereich
3454321	Liam	Fischer	1
5382001	Oliver	Qian-Li	1
6434321	Mila	Heilrich	1

## 2.6 ER-Diagramm



## 2.7 SQL-Abfragen:

### 1. *Alle Absolventen nach Jahrgang, Studiengang und Akademischen Grad:*

```
SELECT vorname, Name, year(studiumende) as Jahrgang,  
studiengang.Studiengang, abschluss.Akademischer_Grad from  
persoenliche_daten join studium on persoenliche_daten.ID = studium.PersonID  
join studiengang on studium.Studiengang = studiengang.ID JOIN abschluss on  
studiengang.AbschlussNr = abschluss.ID order by Studium.Studiumende desc,  
name asc
```

vorname	Name	Jahrgang	Studiengang	Akademischer_Grad
Mehmet	Algül	2024	Ingenieur-Informatik	Master of Science
Samir	Salem	2024	Informatik	Master of Science

### 2. *Anzahl aller Studenten die einen Auslandsaufenthalt hatten:*

```
SELECT count(Distinct Name) as Anzahl_Studierender_mit_Auslandsaufenthalt  
from persoenliche_daten join auslandsaufenthaltsdaten on  
persoenliche_daten.ID = auslandsaufenthaltsdaten.PID ORDER by  
auslandsaufenthaltsdaten.ID
```

Anzahl\_Studierender\_mit\_Auslandsaufenthalt

5

### 3. *Namen aller Hochschulen und Firmen, die besucht wurden:*

```
SELECT Hochschule from auslandsaufenthaltsdaten where Hochschule is NOT null  
UNION SELECT Firma from auslandsaufenthaltsdaten where Firma is NOT null
```

#### Hochschule

Istanbul Ege Üniversitesi

University of Massachusetts System

#### 3.1 *Mit Namen der Studenten:*

```
(SELECT persoenliche_daten.Name ,  
auslandsaufenthaltsdaten.Hochschule FROM auslandsaufenthaltsdaten  
JOIN persoenliche_daten ON auslandsaufenthaltsdaten.PID =  
persoenliche_daten.ID WHERE Hochschule IS NOT NULL UNION SELECT  
NAME, Firma FROM auslandsaufenthaltsdaten JOIN persoenliche_daten  
ON auslandsaufenthaltsdaten.PID = persoenliche_daten.ID WHERE  
Firma IS NOT NULL)
```

Name	Hochschule
Tandogac	Istanbul Ege Üniversitesi
Algül	University of Massachusetts System

#### 4. Studenten die einen Master nach Bachelor gemacht haben in Prozent:

```
SELECT ROUND( AbschlussNr / persoenliche_daten.ID, 2 ) AS Prozentsatz FROM
persoenliche_daten JOIN studium ON persoenliche_daten.ID = studium.PersonID
JOIN studiengang ON studium.studiengang = studiengang.ID JOIN abschluss ON
studiengang.AbschlussNr = abschluss.ID WHERE persoenliche_daten.ID IN(
SELECT COUNT(persoenliche_daten.ID) FROM persoenliche_daten WHERE
persoenliche_daten.Ehemaligen_Status = 'Student' ) AND AbschlussNr IN( SELECT
COUNT(AbschlussNr) FROM studiengang WHERE AbschlussNr > 3 )
```

Prozentsatz

0.57

#### 5. Suche nach Kompetenzprofil mit Leuten, die eine Fähigkeit anbieten:

```
select Name, Faehigkeiten FROM kompetenzprofil join persoenliche_daten on
persoenliche_daten.ID = kompetenzprofil.PID where ich LIKE '%iete%' or ich LIKE
'%IETE%' ORDER by PID
```

Name	Faehigkeiten
Algül	Fachkompetenzen: Controlling

##### 5.1 Suche nach Kompetenzprofil (Bsp: jmd bietet die fähigkeit Controlling an):

```
Select Name, Faehigkeiten FROM kompetenzprofil join persoenliche_daten on
persoenliche_daten.ID = kompetenzprofil.PID where Faehigkeiten LIKE
'%Controlling%' and ich LIKE '%iete%' ORDER by PID
```

Name	Faehigkeiten
Algül	Fachkompetenzen: Controlling

#### 6. Durchschnittsalter aller User:

```
SELECT round(AVG(YEAR(NOW()) - YEAR(private_daten.Geburtstag))) as
durchschnittsalter from private_daten
```

durchschnittsalter

32

#### 7. Studierende die das Studium in der Regelzeit beendet haben:

```
SELECT studium.PersonID, benoetigteSemester, Studiengang.Regelsudienzeit,
abschluss.Akademischer_Grad FROM studium JOIN studiengang on studium.Studiengang =
Studiengang.ID join abschluss on studiengang.AbschlussNr = abschluss.ID where
studium.benoetigteSemester <= studiengang.Regelsudienzeit
```

PersonID	benoetigteSemester	Regelsudienzeit	Akademischer_Grad
2	6	6	Bachelor of Science
2	4	4	Master of Science
7	6	6	Bachelor of Science

#### 8. Student aus Fachbereich MNI mit niedrigstem Preis:

```
SELECT PID, preis FROM `preis` JOIN studium on Preis.PID = studium.PersonID JOIN
```

PID	preis
9	100.00



studiengang on studiengang.ID = studium.Studiengang where preis in (select MIN(Preis)  
FROM Preis) and studiengang.Fachbereich = 1

#### 9. **Alle Namen der Hochschulen die besucht wurden im Ausland:**

```
SELECT DISTINCT hochschule from persoenliche_daten join  
auslandsaufenthaltsdaten on persoenliche_daten.ID =  
auslandsaufenthaltsdaten.PID where hochschule is NOT null  
ORDER by auslandsaufenthaltsdaten.ID
```

##### hochschule

Istanbul Ege Üniversitesi

University of Massachusetts System

Dänemark-Via University College

## 2.8 Anmerkung und Kritik zur Umsetzung:

Berücksichtigt man die verschiedenen Tabellen und ihre Attribute, so sieht man, dass das Modell als eine reale Implementierung für das Hochschul-Alumni-System in Betracht gezogen werden kann und in der Praxis sehr gut auf diese Weise funktioniert.

Der Zweck unseres Projekts ist es, eine aussagekräftige und konsistente Modellierung zu erstellen. Dies sollte nicht nur dem Zweck der Standardisierung entsprechen, sondern auch die referenzielle Integrität gewährleisten, damit der hypothetische Prozess eines solchen Alumni-Systems der Hochschule weiterhin gültig ist, wenn Studenten als Absolventen die Hochschule verlassen.

All dies wird unter anderem, durch Fremdschlüsselbeziehungen garantiert. Trigger und gespeicherte Prozeduren werden ordnungsgemäß implementiert, nicht nur basierend auf Spezifikationen. Ein Vorteil wäre das man durch die Tabelle Inbox Statistiken rauslesen kann, wie die Aktivität der gesamten User.

Weiterhin lag es in unserem Ermessen eine gute Informationsübersicht über, sowohl die Absolventen als auch über die ehemaligen Mitarbeiter zu bieten und diese durch einfache SQL-Abfragen auszulesen. Dadurch ist es der Hochschule möglich auch weiterhin Kontakt zu den Absolventen und den ehemaligen Arbeiter aufzunehmen. Der Grund für die Kontaktaufnahme könnte sein, um einen Nutzen daraus zu ziehen, sei es um die gebildeten Absolventen, vielleicht um Rat oder Hilfe zu bitten, oder die ehemaligen Mitarbeiter vielleicht um eine erneute Einstellung zu bitten bei mangelndem Personal.

Weiterhin ist es durch unsere Datenbank möglich auf persönliche Daten eines EX-Studenten zuzugreifen, falls das Interesse eines anderen ehemaligen Kommilitonen besteht, diesen zu kontaktieren. Aus Datenschutzgründen ist es dem Kommilitonen nicht erlaubt auf private Daten zu zuzugreifen, weswegen wir noch eine extra Tabelle angefertigt haben, in der wir diese aufteilen.

Den Kompetenzprofil in unserer Datenbank haben wir so aufgefasst, dass sich die Absolventen sich selbst dort ausstellen können mit Fähigkeiten, die Sie haben bzw. anbieten, oder aber eine Anzeige aufgeben wird mit Fähigkeiten, die gesucht werden wie eine Art Jobangebot.

Darüber hinaus hat das Alumni-Hochschul-System eine Annäherung zu unserer Vorstellung, wie sich eine mögliche Datenbank der Technischen Hochschule Mittelhessen modellieren lässt. Diese machen eine Besonderheit dieses Modells aus.

Ein Kritikpunkt am Modell wäre etwa, dass zum Beispiel das Ziel eines unseres Triggers unter den Namen Empfehlung dafür sorgt, dass ein Alumnus über die Tabelle Inbox über ein Fort weiter Bildungsangebot benachrichtigt werden soll, aber dieser sich nicht direkt darüber eintragen kann.

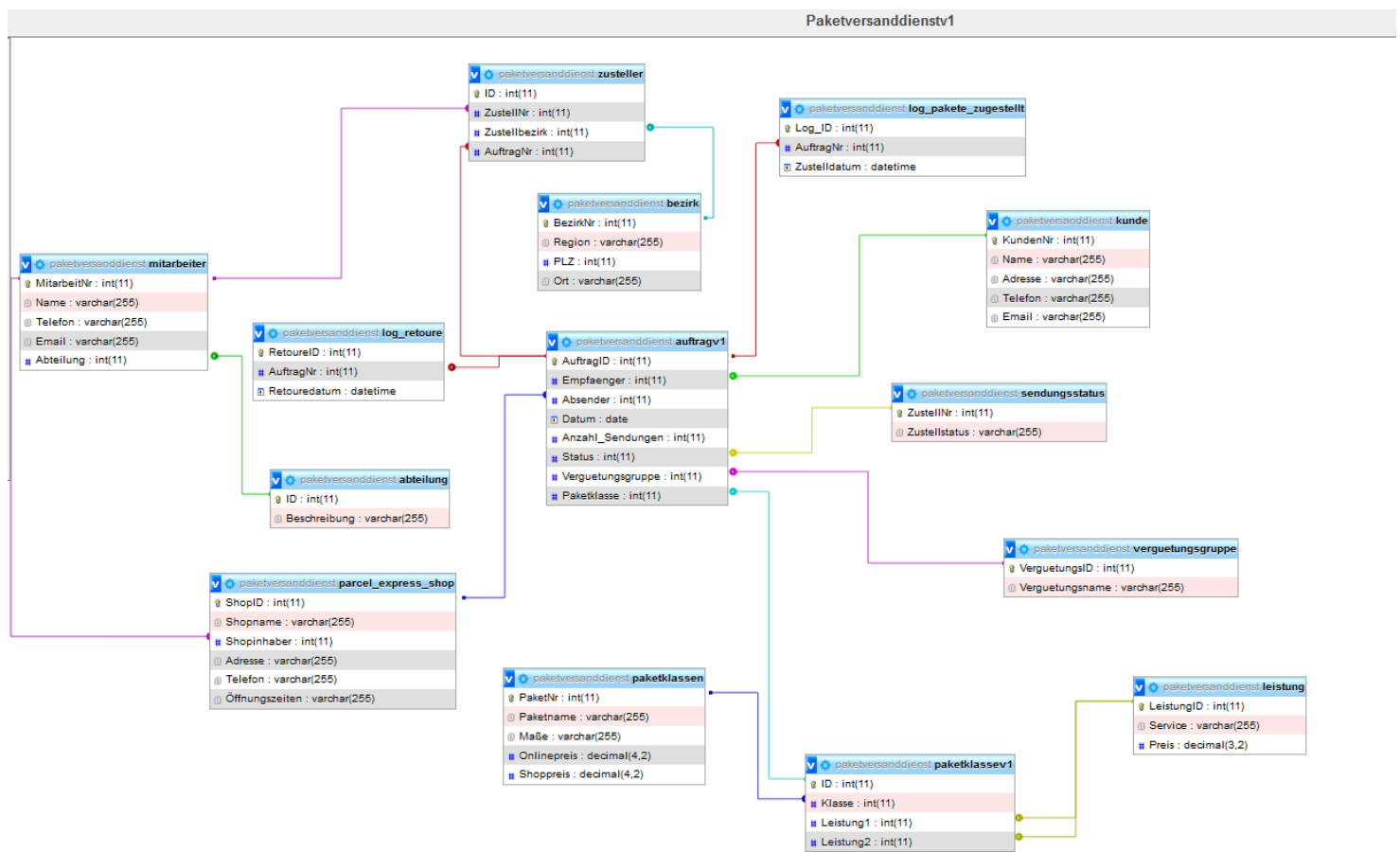
Ein Szenario wäre zum Beispiel, dass ein Absolvent namens Enes über die Inbox informiert werden soll, über ein Fort weiter Bildungsangebot namens Top 10 Programmiersprachen 2021 über die ID von Fachbereich, für Leute des Fachbereichs MNI, da dieser ein ehemaliger Absolvent dort war.

Ein weiterer Fall, welcher nicht abgedeckt wird ist das fast alle unsere Tabellen zu der Tabelle persönliche Daten referenziert und eine starke Abhängigkeit entsteht. Will man also eine Person von persönlichen Daten entfernen so wird das aufgrund der vielen Fremdschlüssel nicht zugelassen, da diese NOT NULL werden können.

Zum Schluss kann man sagen, dass das Datenbanksystem als ein funktionierendes Grundbaustein dienen würde und man diese in der Realität benutzen könnte, jedoch noch Anpassungsbedarf für spezifisch alltägliche Szenarien, wie zum Beispiel das Jemand mehrere Adressen besitzen könnte (o.ä.).



## 3.2 phpMyAdmin Designer Ansicht



Der Designer aus phpMyAdmin verdeutlicht das Beziehungsschema der jeweiligen Relationen. Hierbei ist zu beachten, dass die Beziehungsrichtung folglich vom großen „halbkreis“ ausgeht. Diese stellen die Verknüpfung vom Fremdschlüssel dar. Ein Beispiel zudem.

Die Tabelle mitarbeiter besitzt einen Primary Key „MitarbeiterNr“ und die Tabelle zusteller besitzt unter anderem den Foreign Key „ZustellNr“. Von ZustellNr wird auf MitarbeiterNr referenziert, somit ist die Beziehung deren kenntlich gemacht worden.

## 3.3 Beispiel Szenarien

Was passiert, wenn ein Mitarbeiter, der Zusteller oder Shopinhaber ist, gelöscht wird?

Nun, in diesem Fall steht in der parcel\_express\_shop-Tabelle in der Spalte Shopinhaber ein NULL solange kein neuer für diese Position ernannt wurde. Genauso geschieht dasselbe in der Zusteller-Tabelle. Wobei das Problem ist, dass jeder Zusteller einem Auftrag zugeordnet wird, d.h., dass wenn der eine Zusteller entfernt wird für beispielsweise den Auftrag mit der ID 2, so bleibt der Auftrag noch bestehen, jedoch ohne Zusteller. Daher muss manuell schnellstmöglich für den Auftrag ohne einen Zusteller ein Neuer hinzugefügt werden. Ansonsten würde der Auftrag nie ausführbar werden und den Status „verloren gegangen?“ erhalten, da die Spanne einer Lieferung innerhalb von zwei Wochen erfolgen muss.

Jede Paketklasse besitzt genau zwei Leistungen, die mit den Spalten Leistung1 und Leistung2 realisiert wurde. Was passiert, aber wenn eine Paketklasse nur noch eine Leistung besitzen soll oder sogar mehr als zwei?

In diesem Fall wird die Leistung einer der beiden Spalten gelöscht und erhält den Wert NULL. Falls eine dritte oder vierte Leistung derselben Paketklasse, zum Beispiel bei einem speziellen Angebot, hinzugefügt werden soll, muss eine neue Spalte hinzugefügt werden, die auf die Leistung-Tabelle referenziert.

Ein letztes Szenario könnte sein, dass ein Auftrag als Absender oder Empfänger den Wert NULL hat. Doch was bedeutet das?

Das bedeutet, dass für den Auftrag noch nicht entschieden wurde welchem Shop dieser Auftrag gehören soll. Jedoch sollte man sich schnell einig werden, da das Bestelldatum existent bleibt und nach 14 Tagen der Auftrag den Status verloren gegangen? erhält. Doch sofern ein Auftrag einen Absender, sprich Shop hat, kann kein Shop einfach gelöscht werden. Das heißt, falls ein Shop gelöscht werden müsste, weil die Umsätze nicht stimmen, dann muss jeder Auftrag, der zu dem Shop gehört erledigt worden sein. Ansonsten geht es nicht. Das Problem jedoch ist, dass die Aufträge nach der Zustellung oder Retoure nicht verschwinden, sondern einen extra Eintrag in den Tabellen „log\_pakete\_zugestellt“ und „log\_retoure“ erhalten mit dem entsprechenden Datum. Dies könnte man anhand eines Stored Procedure lösen, indem man sagt, der Shop mit keinen Aufträgen und weniger als XXX Umsätzen pro Jahr muss geschlossen werden. Doch diese Lösung habe ich in meiner Implementierung nicht berücksichtigt, da ich davon ausgehe, dass kein Shop gelöscht werden müsste, sondern höchstens die Mitarbeiter entlassen werden.

Nun zum Empfänger, der Empfänger ist der jeweilige Kunde, der die Bestellung abgegeben hat und sofern es einen Kunden gibt, der einen Auftrag abgegeben hat, kann dieser in der Kunden-Tabelle nicht gelöscht werden. Die Straße, der Name, etc. vom Kunden kann geändert werden, jedoch manuell.

Das sind ein paar Beispiel Szenarien, die in der Datenbank auftreten können mitsamt ihrem Lösungsweg. Jedoch ist die Datenbank nicht perfekt umgesetzt worden, sodass solche Szenarien einen hohen Aufwand und viel Konzentration mit sich bringen.

### 3.4 Umsetzung des Modells

Mithilfe der erarbeiteten Grundlagen, die von Nöten für die Realisierung sind, ist eine Struktur erkennbar, dass durch phpMyAdmin in die Datenbank eingearbeitet wurde.

Letztlich folgen die Tabellen und deren Einzelheiten, wobei die dick gedruckten, die Tabellen sind, dessen Entitäten einen eindeutigen Identifikator erhalten -> z.B. eine ID

#### 3.4.1 Parcel Express Shop die Tabelle enthält,

- eine Shop-ID (Primary Key)
- einen Shopnamen
- einen Shopinhaber (Foreign Key)
- eine Adresse
- eine Telefonnummer
- eine Öffnungszeit

- ➔ Shop-ID, falls mehrere Shops denselben Namen haben, um sie eindeutig identifizieren zu können
- ➔ Shop-Inhaber, referenziert zu der Tabelle „Mitarbeiter“, denn ein Mitarbeiter kann Zusteller, Shop-Inhaber oder vor Ort als Kassierer tätig sein

#### 3.4.2 Mitarbeiter

die Tabelle enthält,

- eine MitarbeitNr (Primary Key)
  - einen Namen
  - eine Telefonnummer
  - eine E-Mail
  - eine Abteilung
- ➔ MitarbeitNr ist quasi die Nummer um Mitarbeiter z.B. mit denselben Namen, Abteilungen,etc zu unterscheiden

#### 3.4.3 Paketklassev1

die Tabelle enthält,

- ID (Primary Key) -> zur Identifizierung
  - eine Klasse (Foreign Key)
  - eine Leistung1 (Foreign Key)
  - eine Leistung2 (Foreign Key)
- ➔ Um die Leistungen zu bestimmen existiert eine weitere Tabelle „**Leistung**“ mit den jeweiligen Service-Angeboten und deren Preise
- ➔ Paketklassev1 referenziert auf „**Paketklassen**“. Diese sind die möglichen Varianten

#### 3.4.4 Paketklassen

Die Tabelle enthält,

- eine PaketNr (Primary Key)
  - einen Paketname
  - eine Maße
  - einen Onlinepreis
  - einen Shoppreis
- ➔ Diese Tabelle gibt Auskunft, welche Paket Varianten möglich ist.

#### 3.4.5 Leistungen

die Tabelle enthält,

- eine LeistungID (Primary Key)
- ein Service
- einen Preis -> Was kosten die jeweiligen Service

- ➔ Die Tabelle „**Leistungen**“ wie oben erwähnt, ist für die Service-Angebote mit deren Preisen zuständig.

#### 3.4.6 Auftragv1

die Tabelle enthält,

- eine AuftragID (Primary Key)
  - einen Empfänger (Foreign Key) -> an wen wird der Auftrag gesendet
  - einen Absender (Foreign Key) -> von welchem Shop geht der Auftrag aus
  - ein Datum -> ist das Bestelldatum
  - eine Anzahl der Sendungen -> wie viele Sendungen hat der Auftrag
  - ein Status (Foreign Key) -> der momentane Sendungsstatus
  - eine Vergütungsgruppe (Foreign Key) -> welche Kategorie wird im Auftrag versendet
  - eine Paketklasse (Foreign Key) -> Klasse mit Preisen inklusive Leistungspreise
- ➔ AuftragID ist der PrimaryKey und sorgt dafür, dass wenn ein Absender mehrere Pakete absendet diese auch anhand der Nummer zu identifizieren sind
  - ➔ Absender referenziert auf die ShopID der „**Parcek Express Shop**“ Tabelle um festzustellen von welchem Shop die Pakete versendet werden
  - ➔ Empfaenger referenziert auf die KundenNr in der Tabelle „**Kunde**“ um den Kunden, der das Paket bekommt zu identifizieren
  - ➔ Status referenziert auf die ZustellNr in der „**Sendungsstatus**“ Tabelle um den momentanigen Status des Auftrags zu bekommen
  - ➔ Verguetungsgruppe referenziert auf die Tabelle „**Verguetungsgruppen**“, um festzustellen welcher Auftrag welche Kategorie bzw. Gruppe bedient
  - ➔ Paketklasse referenziert auf die Tabelle „**preisklassev1**“ um die Preise mitsamt den Leistungen zu ermitteln. (Jeder Auftrag kann nur eine Paketklasse haben!-> vereinfachte Idee)

#### 3.4.7 Kunde

die Tabelle enthält,

- eine KundenNr (Primary Key) -> Identifikator der Kunden
  - einen Namen
  - eine Adresse
  - eine Telefonnummer
  - eine E-Mail
- ➔ Diese Tabelle ist eine zufällig gewählte Kundenliste mit der zusätzlichen KundenNr, der auf die **Auftragstabelle** referenziert, um den Empfänger festzustellen

#### 3.4.8 Bezirk

Die Tabelle enthält,

- eine BezirkNr (Primary Key) -> Identifikator der Bezirke
- eine Region
- eine Postleitzahl
- einen Ort

- ➔ Die Tabelle beinhaltet alle Bezirke, die die Shops liefern. In dem Falle sind es einige aus Hessen und zwei aus Nordrhein-Westfalen (es sollte eine kleine Paketversanddienst Netzwerk darstellen, also nicht vergleichbar mit der DHL, etc!)
- ➔ Jeder zusteller ist einem speziellen Gebiet zugeordnet, jedoch kann es bei unengen an Lieferungen dazu kommen, dass zwei Zusteller einen Bezirk abdecken müssen, daher das Referenzieren von ZustellBezirk aus der **Zustellertabelle** mit dieser hier.

#### 3.4.9 Vergütungsgruppe

Die Tabelle enthält,

- eine VergütungsID (Primary Key) -> Identifikator der Vergütungen
- eine Vergütungsname
- ➔ Gibt mithilfe einer ID an welche Gruppen existieren, die die Kunden erwerben können.  
Grundsätzlich haben alle Shops alle Vergütungsgruppen auf Lager.

#### 3.4.10 Sendungsstatus

Die Tabelle enthält,

- Eine ZustellNr (Primary Key) -> Identifikator vom Status der Lieferung
- Eine Zustellstatus
- ➔ Gibt die Auswahl der Statusliste insgesamt an.

#### 3.4.11 Zusteller

die Tabelle enthält,

- ID (Primary Key) -> zur Identifizierung
- eine ZustellNr (Foreign Key)
- einen Zustellbezirk (Foreign Key)
- eine AuftragsNr (Foreign Key)
- ➔ Ist der Lieferant des jeweiligen Shops, das auf die **Mitarbeiter,- Bezirk- und Auftragsv1tabelle** referenziert.

#### 3.4.12 Abteilung

Die Tabelle enthält,

- ID (Primary Key)
- Beschreibung
- ➔ Die Tabelle dient dazu die Abteilungen der Mitarbeiter zugeben.



### 3.4.13 Log Pakete zugestellt

Die Tabelle enthält,

- eine LogID (Primary Key)
  - eine AuftragNr (Foreign Key)
  - einen Datum -> Zustelldatum
- ➔ Wenn ein Paket den Status in „zugestellt“ ändert, wird dieser in der Tabelle mit seiner ID, seiner Auftragsnummer und Zustelldatum dokumentiert.

### 3.4.14 Log retourne

Die Tabelle enthält,

- eine Retourne-ID (Primary Key)
  - eine AuftragsNr (Foreign Key)
  - ein Retouredatum
- ➔ Wenn ein Paket den Status „Retourne“ erhält wird dieser in der Tabelle mitsamt einer ID, dem Auftragsnummer und dem Retouredatum dokumentiert

## 3.5 SQL – Abfragen

1. **SELECT** mitarbeiter.Name FROM auftragv1, mitarbeiter JOIN zusteller ON  
mitarbeiter.MitarbeitNr = zusteller.ZustellNr WHERE auftragv1.AuftragID = 1 **AND**  
auftragv1.AuftragID = zusteller.AuftragNr GROUP BY mitarbeiter.Name

**Name**  
Jürgen Klaus

- Filtert den Namen des Mitarbeiters, der für die Auftrag-Nummer XY zuständig ist.

2. **SELECT** auftragv1.AuftragID, sendungsstatus.Zustellstatus  
**FROM** auftragv1, sendungsstatus  
**WHERE** auftragv1.AuftragID = 1 **AND** sendungsstatus.ZustellNr = auftragv1.Status  
**GROUP BY** auftragv1.AuftragID;

AuftragID	Zustellstatus
1	zugestellt

- Sendungsstatus eines bestimmten Auftrags.

3. **SELECT** paketklassen.Paketname, paketklassen.Onlinepreis,  
paketklassen.Shoppreis  
**FROM** paketklassev1, paketklassen  
**WHERE** paketklassev1.Klasse = paketklassen.PaketNr  
**GROUP BY** paketklassev1.Klasse;

Paketname	Onlinepreis	Shoppreis
2 Kg-Paket S	2.50	1.75
2 Kg-Paket M	3.80	2.75
2 Kg-Paket XL	4.75	5.50
5 Kg-Paket	5.99	6.99
10 Kg-Paket	8.85	9.75
31,5 Kg-Paket	14.50	12.99

- Angabe der Paketklassen mit den Preisen.

4. SELECT mitarbeiter.Name

FROM zusteller, bezirk, mitarbeiter

WHERE zusteller.ZustellNr = mitarbeiter.MitarbeitNr AND zusteller.Zustellbezirk =  
bezirk.BezirkNr AND bezirk.PLZ = 34117  
GROUP BY mitarbeiter.Name;

**Name**

Ken Kaneki

- Filtert den Zusteller bei Namen für den Bezirk bzw PLZ XY.

5. SELECT mitarbeiter.Shopinhaber

FROM parcel\_express\_shop, mitarbeiter

WHERE parcel\_express\_shop.Shopinhaber = mitarbeiter.MitarbeitNr;

**Name**

Bernd Welk

Violete Helli

Ahmet Özbek

Felix Klein

Lee Chong Chu

- Alle Shopinhaber der Mitarbeitertabelle.

6. SELECT parcel\_express\_shop.Shopinhaber AS ID, mitarbeiter.Name,  
parcel\_express\_shop.Shopname

FROM parcel\_express\_shop, mitarbeiter

WHERE parcel\_express\_shop.ShopID = 2 AND parcel\_express\_shop.Shopinhaber =  
mitarbeiter.MitarbeitNr;

- Shopinhaber einer bestimmten Filiale XY mit der ID in der Mitarbeitertabelle.

ID	Name	Shopname
2	Bernd Welk	Sell Paradise

7. Select PaketNr, Paketname, (COUNT(LT1.Service)+COUNT(LT2.Service)) AS  
Anzahl\_Leistungen, (paketklassen.Onlinepreis+(LT1.Preis+LT2.Preis)) AS  
Gesamtpreis\_Online, (paketklassen.Shoppreis+(LT1.Preis+LT2.Preis)) AS  
Gesamtpreis\_Shop  
FROM paketklassen  
JOIN paketklassev1 ON paketklassen.PaketNr=paketklassev1.Klasse  
JOIN leistung LT1 ON LT1.LeistungID=paketklassev1.Leistung1  
JOIN leistung LT2 ON LT2.LeistungID=paketklassev1.Leistung2 AND  
LT1.LeistungID<>LT2.LeistungID  
WHERE paketklassen.PaketNr= '1'

- Ausgabe der Gesamtpreise einer Paketklasse inklusive den Serviceanzahl für Online -und Shoppreis.

PaketNr	Paketname	Anzahl_Leistungen	Gesamtpreis_Online	Gesamtpreis_Shop
1	2 Kg-Paket S	2	7.45	6.70

8. SELECT kunde.Name, kunde.Adresse  
FROM kunde, auftragv1  
WHERE kunde.KundenNr = auftragv1.Empfaenger AND auftragv1.AuftragID = 1;

- Gibt den Kundennamen und seine Adresse an für seinen Auftrag.

Name	Adresse
Tom Cruise	60313 Frankfurt, Bahnhofsstraße 30

9. SELECT kunde.Name, kunde.Adresse, SUM(auftragv1.Anzahl\_Sendungen)AS Anzahl\_Pakete  
FROM kunde, auftragv1  
WHERE kunde.KundenNr = auftragv1.Empfaenger AND auftragv1.AuftragID = 3;

- Anzahl der Pakete bzw Sendungen eines Kunden für eine bestimmten Auftrag.

Name	Adresse	Anzahl_Pakete
Joshua Stahl	33602 Bielefeld, Schöngasse 17	1

10. SELECT COUNT(auftragv1.Status) AS Auftrag\_der\_Retoure  
FROM auftragv1, sendungsstatus  
WHERE sendungsstatus.Zustellstatus = 'Retoure' AND auftragv1.Status =  
sendungsstatus.ZustellNr;

**Auftrag\_der\_Retoure**

- Gesamtzahl aller Retouren aller Shops.

1

11. SELECT log\_retoure.Retouredatum AS Datum FROM log\_retoure LEFT JOIN auftragv1  
ON log\_retoure.AuftragNr = auftragv1.AuftragID AND auftragv1.Status = 'Retoure' AND  
log\_retoure.RetoureID = 1

**Datum**  
2021-01-10 01:35:16

- Ausgabe vom Datum der Retoure.

## 3.6 Trigger

```
1 CREATE TRIGGER `Paket_zugestellt` AFTER UPDATE ON
  `auftragv1`
2   FOR EACH ROW BEGIN
3   IF NEW.Status = 6
4   THEN INSERT INTO
      log_pakete_zugestellt(log_pakete_zugestellt.AuftragN
r,log_pakete_zugestellt.Zustelldatum) VALUES
5   (OLD.AuftragID, NOW());
6   END IF;
7   END
```

➔ Der Trigger dokumentiert alle Änderungen der Aufträge in den Status zugestellt (=6) mitsamt ihrer AuftragsID sowie das Zustelldatum, das in der Tabelle log\_pakete\_zugestellt gespeichert wird.

➔

```
1 CREATE TRIGGER `Paket_Retoure` AFTER UPDATE ON
  `auftragv1`
2   FOR EACH ROW BEGIN
3   IF NEW.Status = 7
4   THEN INSERT INTO
      log_retoure(log_retoure.AuftragNr,log_retoure.Retour
edatum) VALUES
5   (OLD.AuftragID, NOW());
6   END IF;
7   END
```

➔ Analog die Änderungen der Aufträge in den Status retour (=7) mitsamt ihrer AuftragsID sowie Retoredatum, das in der Tabelle log\_retoure gespeichert wird.

## 3.7 Stored Procedure

```
1 DELIMITER $$
2 CREATE DEFINER=`root`@`localhost` PROCEDURE
  `Sendungsverifizierung`()
3   MODIFIES SQL DATA
4   COMMENT 'Wenn Sendung länger als 14 Tage,
  Frage-> Was passiert?'
5   UPDATE auftragv1 SET auftragv1.Status = 8
6   WHERE auftragv1.Status < 6 AND CURRENT_DATE -
  auftragv1.Datum > 14$$
7 DELIMITER ;
```

Schaut in der Tabelle Auftrag, ob ein Auftrag länger als 2 Wochen dauert und nicht zugestellt oder als Retoure betitelt wurde, wird sich gefragt ob sie verloren gegangen ist.

### 3.8 Bemerkungen und Kritik an der Umsetzung

Primär galt unser Vorhaben der sinnvollen und schlüssigen Modellierung der Aufgabenstelle. Die Kriterien für die Umsetzung der Modellierung waren zumal die Zwecke der Normalisierung einzuhalten sowie die referenzielle Integrität zu beachten. Letzteres sorgt für die Funktionalität der Abläufe einer solchen imaginären Paketversanddienst, wenn beispielsweise ein Auftrag erfüllt und somit das Paket zugestellt wurde oder ein Mitarbeiter gekündigt wurde. All dies wurde mithilfe von Fremdschlüsselbeziehungen abgesichert.

Die Funktionen von Trigger und Stored Procedure wurden sinnvoll der Realität entsprechend in die Datenbank integriert. Ohne diese beiden Integrationen würde die Datenbank im Wesentlichen funktionieren, jedoch würde die realitätsnahe Umsetzung fehlen. Sie unterschreiben eine Besonderheit des Modells.

Ein grundsätzlicher Kritikpunkt des Modells besteht darin, dass die Kunden keine Rechnungsposition über ihre Bestellungen erfahren, sowie keine Benachrichtigung per E-Mail über ihre Sendung erhalten, das heutzutage in der Realität gang und gäbe geworden ist. Ebenso ist ein wesentlicher Kritikpunkt, dass die Bezirke, die geliefert werden, enorm beschränkt ist. Ein realitätskonformer Paketversanddienst würde im ganzen Land liefern.

Ein weiterer Kritikpunkt liefert die Tabelle „Verguetungsgruppe“. Dieser ist begrenzt an den möglichen Gruppen, die von den Kunden erworben werden können. Jedoch will ich an der Stelle anmerken, dass die Firma bzw. die Dienstleistung eine neugegründete und kleine darstellen soll. Sie ist keineswegs vergleichbar mit der DHL oder ähnlichem.

Genug mit der Kritik an der Modellierung. Die positiven Aspekte des Modells sind die realitätsnahe Modellierung, die Vielfältigkeit sowie die Funktionsfähigkeit im realen Gebrauch. Die Tabellen „log\_pakete\_zugestellt“ und „log\_retoure“ bieten eine super Übersicht über die bereits fertigen Aufträge, sowie die Aufträge die als Retoure gelten. Ohne eine solche Dokumentation würde ein Chaos eintreffen.

Genauso bietet der Procedure eine gute Rückmeldung über Aufträge, die nach zwei Wochen noch immer nicht den Status „zugestellt“ oder „retoure“ erhalten haben, denn so wird sich gefragt was mit dieser Lieferung passiert ist.

Durch die ausgedachten SQL-Befehle lässt sich schließen, dass diese Modellierung in der Realität eine Anwendung finden würde.

Abschließend möchte ich erwähnen, dass für die Datenbank einige weitere Verbesserungen gemacht werden können. Anhand der Kritikpunkte wird verdeutlicht, dass nicht alles zu 100% funktioniert, jedoch als Grundgerüst eine sehr gute Schablone dient.

## 4 Pflegedienst

Im Folgenden wird die Modellierung der Aufgabe 10 „Pflegedienste“ näher erläutert und durch eine Datenbank beschrieben. Diese soll die Verwaltung/Abwicklung eines Pflegedienstes erleichtern.

### 4.1 Grundlagen zur Modellierung:

Ich habe mir folgendes überlegt:

- Es existieren n-viele Mitarbeiter, Patienten, Fahrzeuge, Krankenkassen und Pflegeleistungen
- Jedem dieser Tupel wird ein Primärschlüssel zugeordnet
- Jede Entität enthält neben dem PS auch Attribute
- Es soll auf Redundanz verzichtet werden

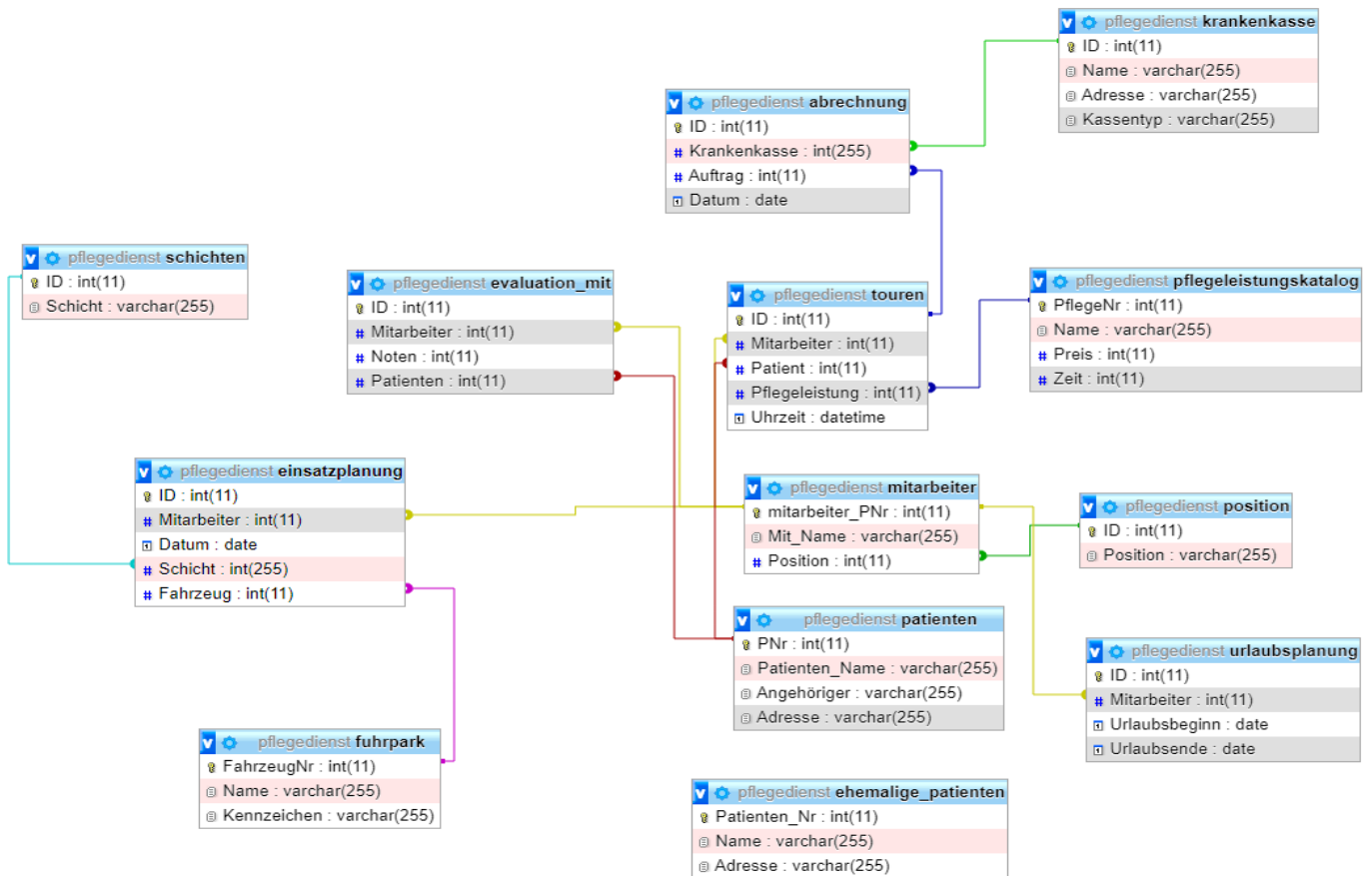
#### 4.1.1 Umsetzung des Modells:

Ich hatte mir zu dieser Datenbank überlegt, dass ich erstmal Tabellen die Grundstruktur eines Pflegedienstes erstelle, um Redundanz zu vermeiden. Im Zweiten Abschnitt habe ich dann anhand der Entitäten aus dem ersten Abschnitt verschiedene Tabellen gebildet, die die Planung, Erfassung und Abrechnung realisieren. Ganz am Ende habe ich noch eine ER-Diagramm zu diesem Modell.

Im Folgenden werde ich meine Entitätstypen erläutern, die via phpMyAdmin in die Datenbank eingearbeitet wurde.

## 4.2 PhpMyAdmin Designer Ansicht:

Hier werden die jeweiligen Beziehungen der Relationen deutlich, manche Tabellen, wie z.B



Mitarbeiter, dienen häufiger als Fremdschlüssel in anderen Tabellen.

## 4.3 Grundstruktur:

- **Mitarbeiter:**
  - o Personalnummer als eindeutiger Identifikation
  - o Hat einen Namen
  - o Hat eine Position (Aus **Position**)
- **Patienten:**
  - o Hat eine Patientennummer als ID
  - o Hat einen Namen
  - o Hat den Namen eines Angehörigen und deren Adresse
  - o Hat seine eigene Adresse
- **Ehemalige Patienten:**
  - o Hier werden die ehemaligen Patienten gespeichert
- **Fuhrpark:**
  - o Hat eine Fahrzeugnummer als ID
  - o Hat das Modell als Namen
  - o Hat ein Kennzeichen
- **Krankenkasse:**
  - o Hat eine ID
  - o Hat einen Namen

- Hat eine Adresse
- Hat einen Kassentyp
- **Pflegeleistungskatalog:**
  - PflegeNr als ID
  - Hat eine Bezeichnung der Leistung
  - Hat einen Preis
  - Hat die durchschnittliche Dauer der Leistung
- **Schichten:**
  - Die verschiedenen Schichten als Hilfstabelle mit einer einzigartigen ID
- **Position:**
  - ID Als Primärschlüssel
  - Verschiedene Positionen in einem Unternehmen

#### 4.3.1 Erfassung, Planung und Abrechnung:

- **Einsatzplanung**
  - In der Einsatzplanung wird der Arbeitstag eines Mitarbeiters bestimmt, d.h welche Schicht und welches Fahrzeug er während dieser fährt
  - ID als Primärschlüssel
  - Welcher Mitarbeiter arbeitet (aus der Tabelle **Mitarbeiter**)
  - Welches Datum es ist
  - Welche Schicht es ist (aus der Tabelle **Schichten**)
  - Welches Fahrzeug benutzt wird (aus der Tabelle **Fuhrpark**)
- **Urlaubsplanung**
  - ID als Primärschlüssel
  - Welcher Mitarbeiter im Urlaub ist (aus der Tabelle **Mitarbeiter**)
  - Der Urlaubsbeginn und Urlaubsende
- **Evaluation\_Mit**
  - ID als Primärschlüssel
  - Welcher Mitarbeiter bewertet wird (Aus **Mitarbeiter**)
  - Welche Note
  - Welcher Patient bewertet (aus **Patienten**)
- **Touren**
  - In Touren wird ein spezifischer Auftrag während einer Schicht beschrieben
  - ID als Primärschlüssel
  - Welcher Mitarbeiter die Tour fährt (aus **Mitarbeiter**)
  - Welcher Patient behandelt wird (aus **Patienten**)
  - Welche Pflegeleistung durchgeführt wird (aus **Pflegeleistungen**)
  - Welches Datum und welche genaue Uhrzeit
- **Abrechnung**
  - ID als Primärschlüssel
  - In welcher Krankenkasse der Patient ist (Aus **Krankenkasse**)
  - Um welchen Auftrag es sich handelt (aus **Touren**)
  - Um welches Datum es sich handelt



## 4.4 SQL-Querys

### 4.4.1 Gesucht wird welcher Patient die Grundpflege bekommt

```
SELECT patienten.Patienten_Name AS Grundpflege FROM patienten
JOIN touren
ON patienten.PNr = touren.Patient
WHERE touren.Pflegeleistung = 5
```

#### Grundpflege

Bettina Buchse

Flamur Shapiri

Max Tröter

Hans Peter

### 4.4.2 Welcher Patient ist bei der Techniker Krankenversicherung

```
SELECT patienten.Patienten_Name AS Techniker_Versicherte FROM
patienten
JOIN touren ON touren.Patient = patienten.PNr JOIN abrechnung ON
touren.ID = abrechnung.Auftrag
WHERE abrechnung.Krankenkasse = 2
```

#### Techniker\_Versicherte

Max Tröter

Flamur Shapiri

### 4.4.3 Welcher Mitarbeiter hat 5 Punkte oder weniger in der Evaluation

```
SELECT mitarbeiter.Mit_Name AS Mitarbeiter FROM mitarbeiter
JOIN evaluation_mit on mitarbeiter.mitarbeiter_PNr = evaluation_mit.Mitarbeiter
WHERE evaluation_mit.Noten < 5
```

#### Mitarbeiter

Burak Yildirim

Dimitri Pavlov

Maximillian Schmidt

Mehmet Yilmaz

### 4.4.4 Alle Mitarbeiter und wie oft sie Frühschicht haben

```
SELECT mitarbeiter.Mit_Name AS Mitarbeiter, COUNT(einsatzplanung.Schicht)
AS Anzahl_FS FROM einsatzplanung JOIN mitarbeiter
ON mitarbeiter.mitarbeiter_PNr = einsatzplanung.Mitarbeiter
WHERE einsatzplanung.Schicht = 1
GROUP by Mit_Name
ORDER BY COUNT (einsatzplanung.Schicht) DESC
```

Mitarbeiter	Anzahl_FS
Chico Ambau	2
Boef man	1
Büsra Denizli	1
WasSolllch SagenBruder	1
Hans Wurst	1
Dimitri Pavlov	1

### 4.4.5 Welches Fahrzeug wird nicht genutzt?

```
SELECT fuhrpark.Name AS Name, fuhrpark.FahrzeugNr
FROM fuhrpark LEFT JOIN
einsatzplanung on einsatzplanung.Fahrzeug = fuhrpark.FahrzeugNr
WHERE einsatzplanung.Fahrzeug is null
```

Name	FahrzeugNr	Kennzeichen
VW Polo	3	F AM 77

### 4.4.6 Welcher Patient wohnt in Bornheim

```
SELECT patienten.Patienten_Name from patienten
WHERE patienten.Adresse like '%Bornheim%'
```

		Patienten_Name
<input type="checkbox"/>	Bearbeiten	Kopieren  Löschen David Schlosser
<input type="checkbox"/>	Bearbeiten	Kopieren  Löschen Celo Pelo

#### 4.4.7 Welcher Mitarbeiter fährt einen Polo beim Einsatz

```
SELECT mitarbeiter.Mit_Name AS Polonutzer
FROM mitarbeiter join einsatzplanung
ON mitarbeiter.mitarbeiter_PNr=einsatzplanung.Mitarbeiter
JOIN fuhrpark on einsatzplanung.Fahrzeug = fuhrpark.FahrzeugNr
WHERE fuhrpark.Name = 'VW Polo'
GROUP BY mitarbeiter.Mit_Name
```

Polonutzer
Chico Ambau
Dimitri Pavlov
Maximilian Schmidt
Mehmet Yilmaz
WasSollich SagenBruder

#### 4.4.8 Gesamtbetrag der Abrechnung mit den Krankenkassen

```
SELECT SUM (pflegeleistungskatalog.Preis) AS Gesamtkosten from pflegeleistungskatalog
join touren on touren.Pflegeleistung = pflegeleistungskatalog.PflegeNr
join abrechnung on abrechnung.Auftrag= pflegeleistungskatalog.PflegeNr
```

Gesamtkosten
555

#### 4.4.9 Gesamtanzahl der Behandlungen für einen Patienten und deren Gesamtkosten

```
SELECT patienten.Patienten_Name AS Patient, COUNT(patienten.Patienten_Name) as Anzahl,
SUM(pflegeleistungskatalog.Preis) as Gesamtkosten FROM patienten
JOIN touren on touren.Patient = patienten.PNr
JOIN pflegeleistungskatalog ON
pflegeleistungskatalog.PflegeNr = touren.Pflegeleistung
GROUP by patienten.Patienten_Name
ORDER BY COUNT(patienten.Patienten_Name) DESC
```

Patient	Anzahl	Gesamtkosten
Ließchen Fuchs	3	105
Sibel Salo	2	80
Flamur Shapiri	2	80
Max Tröter	2	80
Hans Peter	2	120
Celo Pelo	1	30
Bettina Buchse	1	60

#### 4.4.10 Welche Mitarbeiter in den ersten zwei Februarwochen 2021 Urlaub haben

```
SELECT Mitarbeiter.Mit_Name AS Urlaub FROM Mitarbeiter JOIN urlaubsplanung
ON Mitarbeiter.mitarbeiter_PNr = urlaubsplanung.Mitarbeiter
WHERE urlaubsplanung.Urlaubsbeginn BETWEEN '20210201' AND '20210214'
```

Urlaub
Burak Yildirim
Robert Geiger

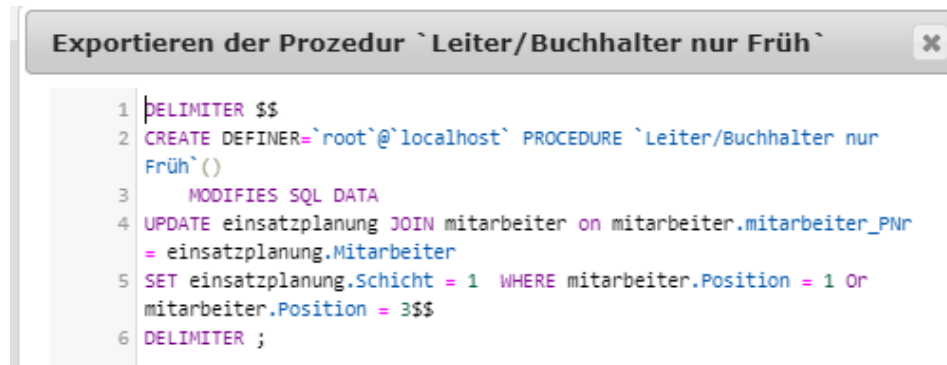
#### 4.5 Trigger:

Bei Löschung eines Patienten wird dieser in die Tabelle **ehemalige Patienten** hinzugefügt:

```
1 CREATE TRIGGER `Löschung` BEFORE DELETE ON `patienten`
2 FOR EACH ROW INSERT INTO ehemalige_patienten VALUES
(old.PNr,old.Patienten_Name,old.Adresse)
```

## 4.6 Stored Procedures:

Falls ein Mitarbeiter, der kein Pfleger ist eine andere Schicht, als Frühschicht hat, wird dies auf Frühschicht geändert.



```
1 DELIMITER $$
2 CREATE DEFINER=`root`@`localhost` PROCEDURE `Leiter/Buchhalter nur
  Früh`()
3     MODIFIES SQL DATA
4     UPDATE einsatzplanung JOIN mitarbeiter on mitarbeiter.mitarbeiter_PNr
    = einsatzplanung.Mitarbeiter
5     SET einsatzplanung.Schicht = 1 WHERE mitarbeiter.Position = 1 Or
    mitarbeiter.Position = 3$$
6 DELIMITER ;
```

## 4.7 Kritik und Ergänzung zur Umsetzung

Das Ziel der Aufgabe war es, eine sinnvolle und funktionierende Datenbank zu erstellen, die sowohl die Aspekte der Normalisierung beachtet, als auch eine referenzielle Integrität gewährleistet.

Zuerst wurden die repetitiv vorkommenden Informationen ausgelagert und im Abschnitt Grundstruktur zusammengefasst, diese wurden dann natürlich mit Primärschlüsseln versehen, da sie im zweiten Abschnitt als Fremdschlüssel verwendet worden sind.

Mit dieser erstellten Datenbank könnte man die Organisation/Abrechnung eines Pflegedienstes gut umsetzen, da sie alle erforderlichen Daten effizient und ohne Redundanzen abspeichert, eine Skalierung auf eine höhere Anzahl an Patienten und Mitarbeitern wäre auch ohne weiteres möglich.

Nichtsdestotrotz ist der Umgang mit kranken Patienten und Abrechnung mit Krankenkassen ein sehr komplexes Gebiet, da es viele Gesetze und Normen gibt, die zu beachten sind, die aber den Rahmen dieser Aufgabe sprengen würden, dieses Modell ist voll funktionsfähig und gilt eher als Grundstruktur, die ohne weiteres erweitert werden kann.

Im Folgenden werde ich näher auf einige Punkte eingehen, die noch ergänzt werden könnten.

In meinem Modell wird nicht auf die Krankenhistorie, Alter und Pflegegrad des Patienten eingegangen, was eigentlich wesentlich für die Abrechnung mit den Krankenkassen ist, des Weiteren wird auch bei Tabelle Mitarbeiter nicht darauf eingegangen, ob jeder Mitarbeiter überhaupt die richtige Schulung, Zertifikat oder Ähnliches hat, um die passende Pflegeleistung durchzuführen, es wird davon ausgegangen, dass alle Mitarbeiter alle Pflegeleistungen machen können.

Als letzten Kritikpunkt möchte ich auf den Fuhrpark eingehen, dieser ist in meiner Datenbank sehr einfach gehalten und könnte noch um viele Punkte ergänzt werden, wie z.B die nächste HU, Leasing oder nicht und Kilometerstand. Natürlich gibt es noch viele andere Dinge, die ergänzt werden können, je nach Einsatzfeld, das vorliegende Modell ist nur, wie schon oben erwähnt, ein voll funktioniertes Grundgerüst, das erweitert werden kann.

## 4.8 ERD-Skizze:

