Datenbanken und SQL

(Woche 2 - Tag 1)

Agenda

Physischer Datenbankentwurf

- Einführung
  - Physischer Datenbankentwurf
  - Datenbanksprache
  - Konventionen
  - Exemplarisches RDB-Schema
- Data Definition Language
  - CREATE/DROP DATABASE
  - USE DATABASE
  - CREATE/DROP TABLE
    - Allgemeine Syntax
    - Typ
    - Constraint
      - (NOT) NULL
      - AUTO_INCREMENT
    - Schlüssel
      - PRIMARY KEY
      - FOREIGN KEY
  - Fremdschlüsselüberprüfung und Referentielle Integrität

Einführung

Konzeptioneller -> Logischer -> Physischer Datenbankentwurf

- Im Prozess des Datenbankentwurfs haben wir in der **konzeptionellen** Phase ER-Modelle entworfen, um alle für die zu erstellende Datenbank relevanten Informationen in einem (groben) „Bau-Plan“ zusammenzufassen.
- Beim Übergang zum **logischen** Datenbankentwurf verfeinerten wir diesen Plan, setzten dabei die Anforderungen der 3. Normalform um und stellten die entsprechenden Ergebnisse mittels eines RDB-Schemas dar.
- Dieses Schema werden wir nun auf dem Rechner implementieren und den entsprechenden Vorgang als **„Physischen Datenbankentwurf“** bezeichnen.

Datenbanksprache SQL

- Die Implementierung wird uns mit Hilfe der Datenbanksprache **SQL** gelingen.
- SQL (Structured Query Language) ist allerdings genau genommen eine ganze „Sprachfamilie“ von der wir uns nur mit dem Dialekt **„MySQL“** befassen werden.
- Und auch dies trifft mittlerweile nur noch eingeschränkt zu, denn nach der nicht unumstrittenen Übernahme von MySQL durch Oracle, lautet die (allerdings „baugleiche“) freie Version zu MySQL nun **MariaDB**.
- „Umgangssprachlich“ wird allerdings weiterhin von „MySQL“ gesprochen.

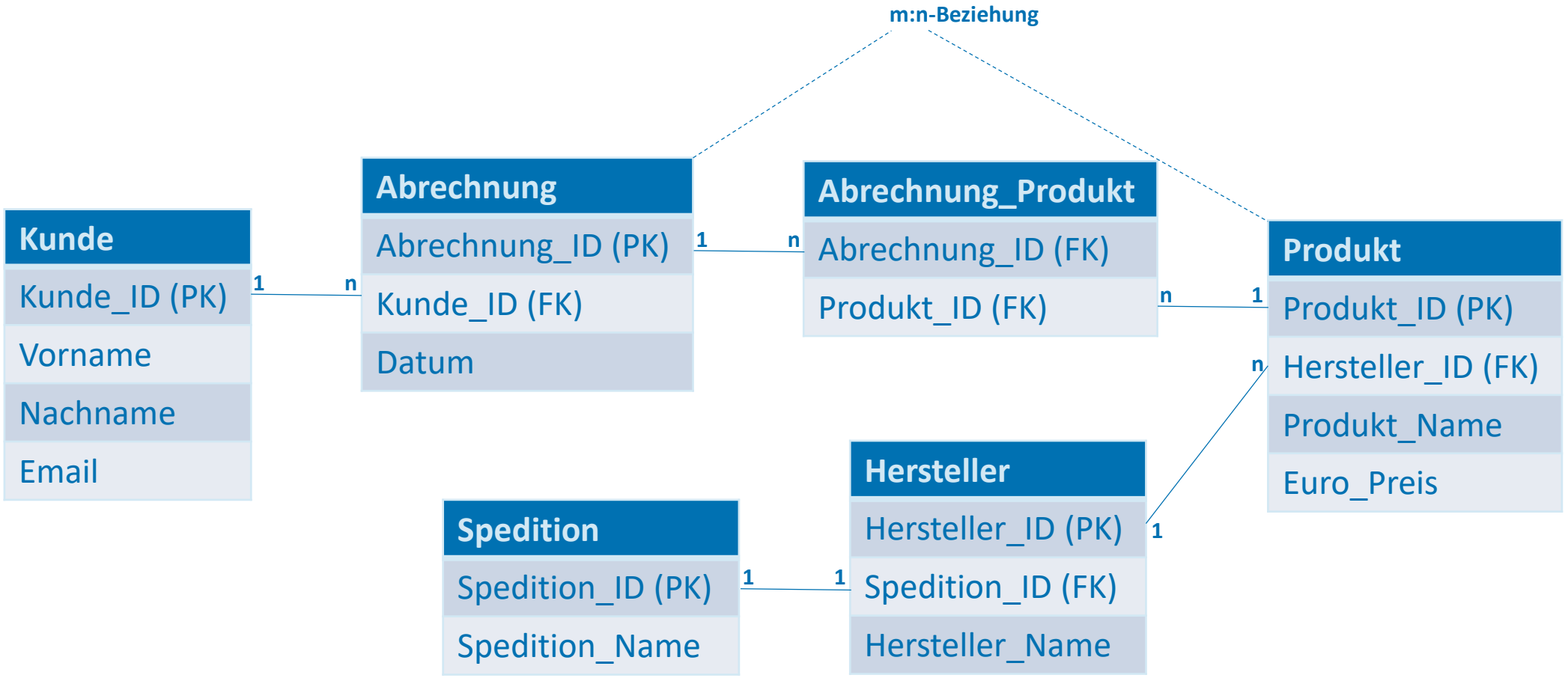
SQL -> Konventionen

- Allgemein anerkannte Konventionen wird man kaum ausmachen können, allenfalls ist nahezu „unumstritten“, dass alle SQL-Anweisungen durchgehend mit **GROSSBUCHSTABEN** notiert werden.
- Dies ist aber nicht verpflichtend, denn SQL ist **nicht case sensitive**, daher wird (intern) nicht zwischen Groß und Kleinbuchstaben unterschieden.
- Zumindest aber gilt: Das konsequente Einhalten von Konventionen innerhalb einer Datenbank erleichtert es, mit dieser zu arbeiten. Für unseren Unterricht wollen wir uns daher selbst die folgenden Regeln auferlegen:
  - Alle Tabellen erhalten einen Namen im Singular (Einzahl).
  - Alle Primärschlüssel erhalten den Tabellennamen + „_ID“.
  - Alle Fremdschlüssel erhalten (sofern möglich) den identischen Namen jenes Primärschlüssel, auf den sie sich beziehen.
  - Zwei Nichtschlüssel-Attribute aus unterschiedlichen Tabellen besitzen stets einen unterschiedlichen Namen.
  - Alle Namen (Datenbank, Tabelle, Attribut) sollten nur aus Buchstaben des englischen Alphabets (und/oder Unterstrich) bestehen.

Exemplarisches RDB-Schema

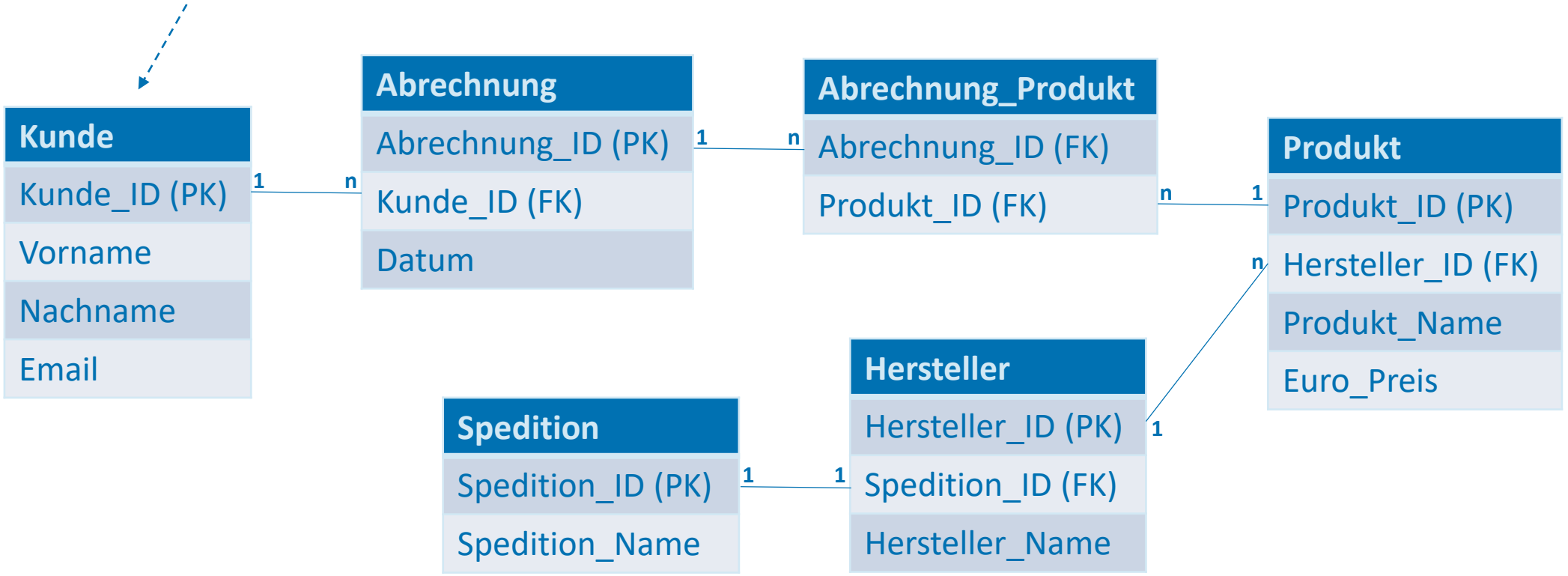
- Wir haben uns nicht umsonst die gesamte erste Woche dieses Bausteins mit dem **theoretischen Entwurf** von Datenbanken befasst, denn tatsächlich wird sich herausstellen, dass SQL-Befehle üblicherweise **nicht ohne detaillierte Kenntnis** der zu Grunde liegenden Datenbank gelingen.
- Um uns dennoch auf das Erlernen von SQL konzentrieren zu können, werden wir im Anschluss ein **exemplarisches RDB-Schema** einführen, mit dessen Hilfe wir uns bei den kommenden Inhalten nicht regelmäßig in immer wieder neue Schemata einarbeiten müssen.

RDB-Schema „Geld_her“



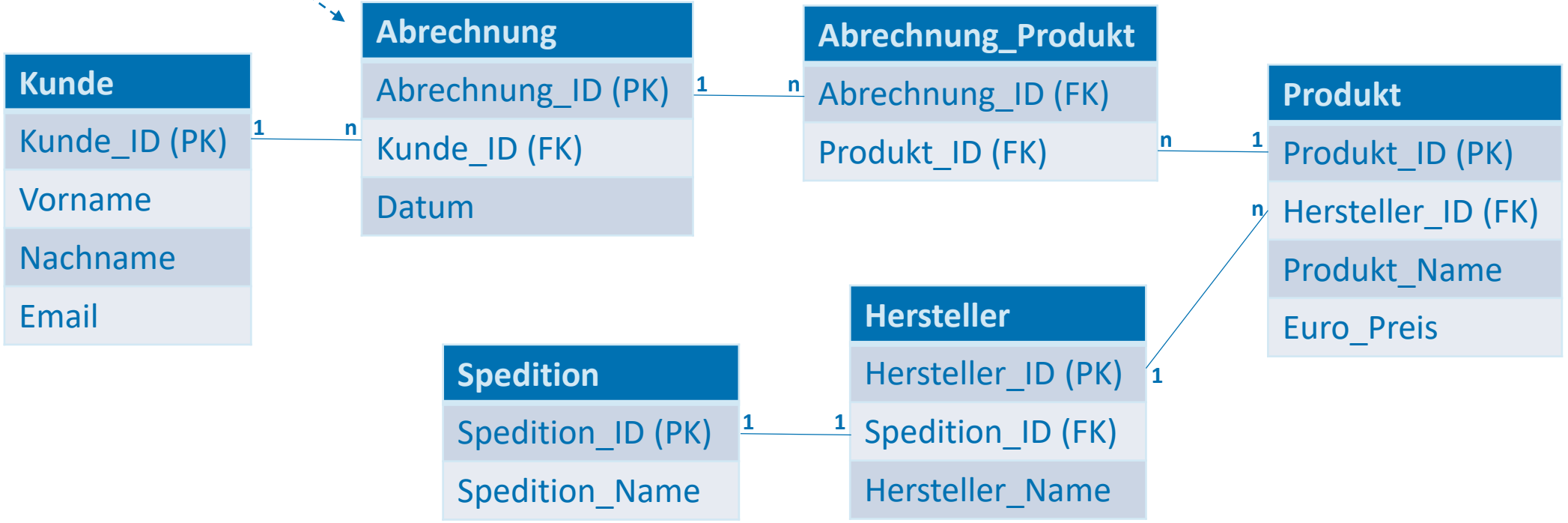
RDB-Schema „Geld_her“

Jeder, der sich beim Online-Händler „Geld_her“ registriert, wird dadurch zum **„Kunden“** und in der Datenbank eingepflegt.



RDB-Schema „Geld_her“

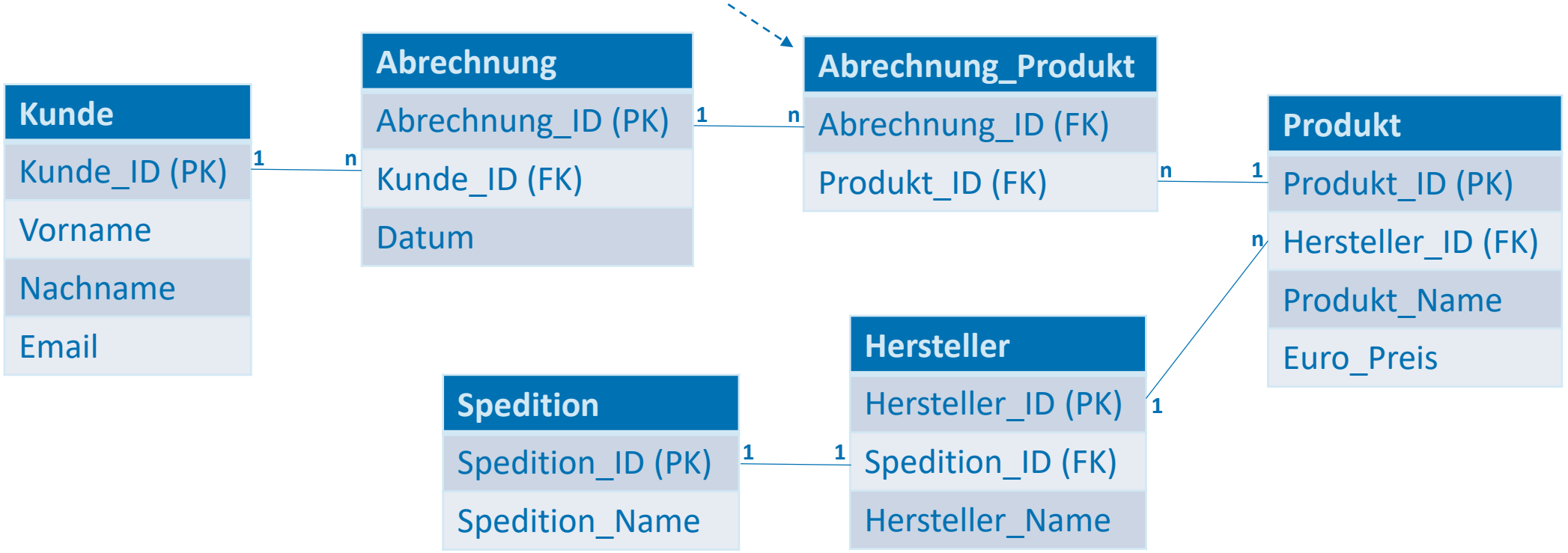
Einem Kunden, der sich auf der Website von „Geld_her“ einloggt, wird automatisch eine **Abrechnung_ID** zugewiesen. (unabhängig davon, ob er etwas kauft)



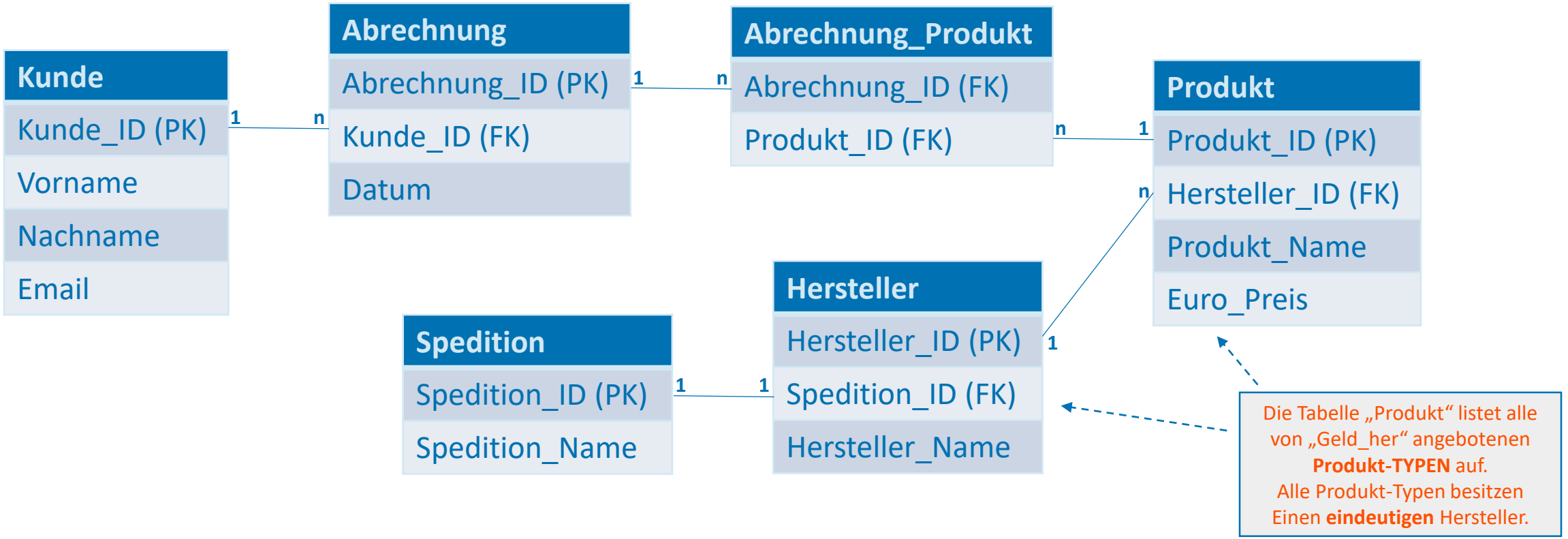
RDB-Schema „Geld_her“

Für alle angebotenen Produkte kann durch das **Anklicken eines „Kaufbuttons“** erreicht werden, dass dieses der aktuellen Abrechnung_ID des Kunden zugeordnet wird.

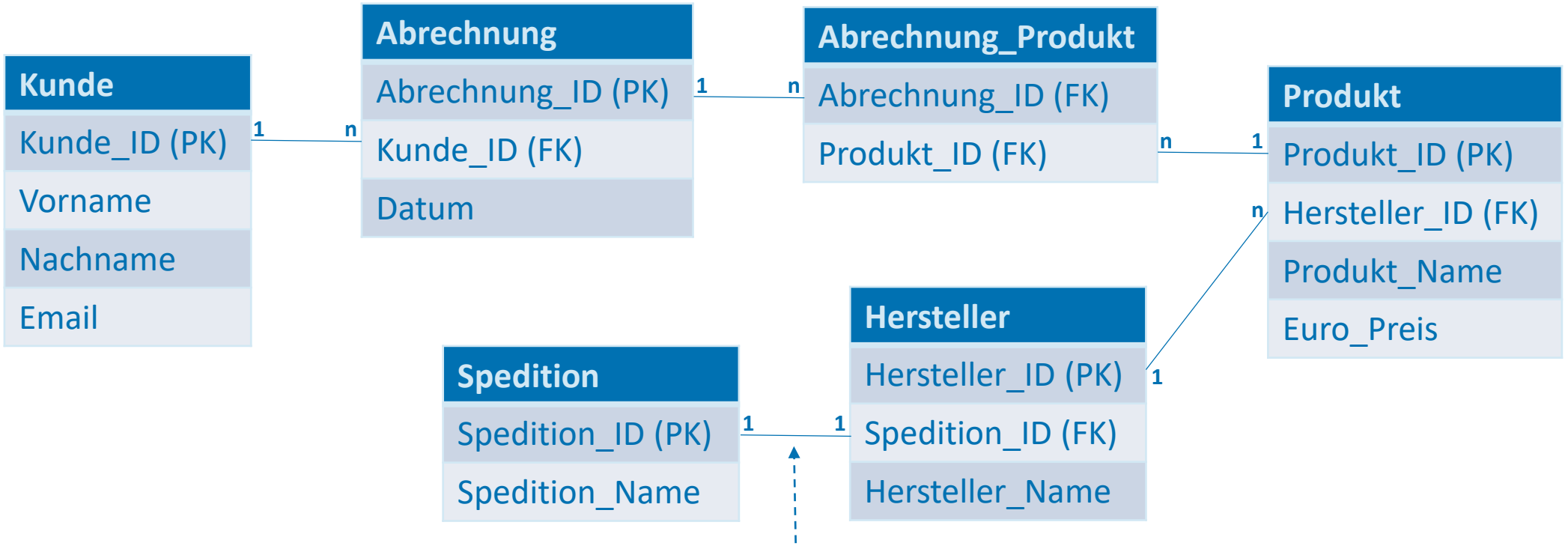
Jeder Datensatz der Hilfstabelle „Abrechnung_Produkt“ entspricht also dem **Einkauf** eines konkreten **Produkt-EXEMPLARS**.
Falls auf der **selben Abrechnung** mehrfach Exemplare des **selben Produkt-TYPS** eingekauft wurden, so werden entsprechend auch **mehrere Datensätze** in der Hilfstabelle eingepflegt.



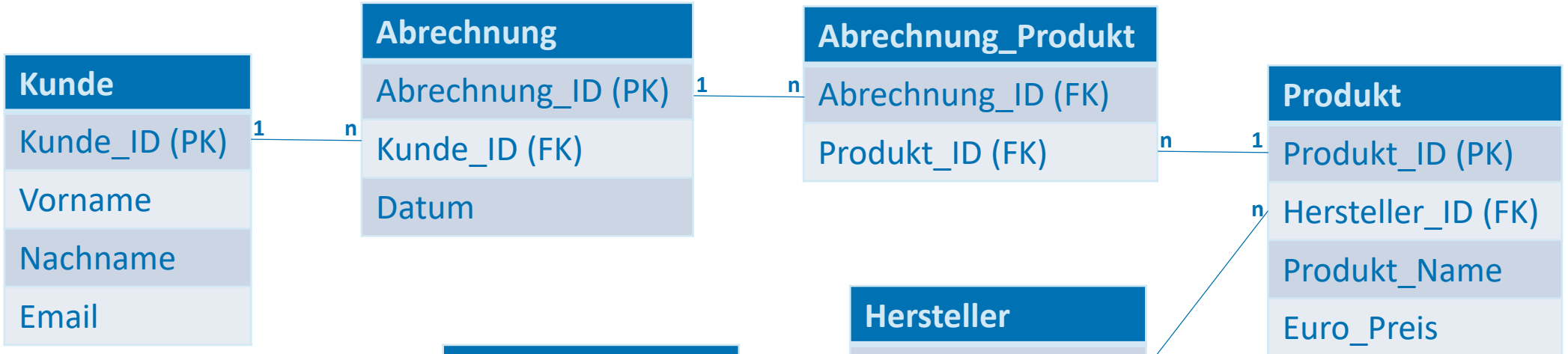
RDB-Schema „Geld_her“



RDB-Schema „Geld_her“



RDB-Schema „Geld_her“



DDL (Data Definition Language)

CREATE [DROP] DATABASE Datenbank_Name

Um die (noch leere) Datenbank „Geld_her“ zu erzeugen, verwenden wir den folgenden Befehl:

```
CREATE DATABASE Geld_her;
```

Um eine bestehende Datenbank zu löschen (in unserem Fall die Datenbank „Geld_her“), verwenden wir den Befehl:

```
DROP DATABASE Geld_her;
```

Hinweis:

Falls wir einen **einzigen** Befehl absenden, so könnten wir auf das abschließende Semikolon verzichten.
Falls hingegen (gleichzeitig) **mehrere** Befehle abgesendet werden sollen, so ist das Semikolon (nach jedem Befehl) verpflichtend.

Wir werden allerdings aus didaktischen Gründen **stets** ein Semikolon setzen (und empfehlen dies auch für die IHK-Abschlussprüfung).

USE Datenbank_Name

Wenn wir mit der Datenbank „Geld_her“ arbeiten wollen, wird es günstig sein, dies anzukündigen:

```
USE Geld_her;
```

Allerdings wird dies nicht unbedingt nötig sein – siehe folgende Folie:

CREATE TABLE Tabellen_Name

Um (z.B.) die Tabelle „Kunde“ zu erzeugen, verwenden wir den folgenden Befehl:

```
CREATE TABLE Kunde
(
        ...
        ...
);
```

hierzu gleich mehr

Der obige Befehl funktioniert nur, falls zuvor mittels **USE** Geld_her diese Datenbank ausgewählt wurde.
Würde man hingegen auf **USE** verzichten wollen, so müsste man zur Erstellung der Tabelle „Kunde“ den folgenden Code verwenden:

CREATE TABLE Geld_her.Kunde(…);

DROP TABLE Tabellen_Name

Um (z.B.) die Tabelle „Kunde“ zu löschen, verwenden wir den folgenden Befehl:

```
DROP TABLE Kunde;
```

Das Löschen einer Tabelle ist gelegentlich **nicht unproblematisch** (und wäre es auch bei der Tabelle „Kunde“).
Wir werden uns mit diesem Sachverhalt am Ende dieser Vorlesung näher befassen, und hierbei die Begriffe
„Fremdschlüsselüberprüfung“ und **„Referentielle Integrität“** kennen lernen.

CREATE TABLE -> Attribut Typ Constraint

Um (z.B.) in der Tabelle „Kunde“ auch alle Attribute einzutragen, verwenden wir die folgende Syntax:

```
CREATE TABLE Kunde
(
        Kunde_ID INT(11) AUTO_INCREMENT,
```

CREATE TABLE -> Ganze Zahl

Um (z.B.) in der Tabelle „Kunde“ auch alle Attribute einzutragen, verwenden wir die folgende Syntax:

```
CREATE TABLE Kunde
(
        Kunde_ID INT(11) AUTO_INCREMENT,
```

Die zulässigen Integerwerte liegen wie üblich zwischen -2^{31} (= -2.147.483.648) und $2^{31}-1$ (= 2.147.483.647)
Der Parameter in der Klammer bestimmt die **Breite der Spaltenausgabe**.
Wir wählen „11“, um die maximal 10-stellige Zahl zuzüglich 1 Vorzeichen darstellen zu können.

CREATE TABLE -> Automatisches Hochzählen

Um (z.B.) in der Tabelle „Kunde“ auch alle Attribute einzutragen, verwenden wir die folgende Syntax:

```
CREATE TABLE Kunde
(
        Kunde_ID INT(11) AUTO_INCREMENT,
```

Der Constraint „AUTO_INCREMENT“ sorgt dafür, dass der **Primärschlüsselwert** (beginnend bei 1) mit jedem neuen Datensatz automatisch hochgezählt wird.

Bei MySQL (bzw. MariaDB) kann der Primärschlüsselwert aber auch trotz dieses Constraints noch händisch gesetzt werden. Dies ist allerdings nicht bei jedem SQL-Dialekt der Fall.

CREATE TABLE -> Attribut Typ Constraint

Um (z.B.) in der Tabelle „Kunde“ auch alle Attribute einzutragen, verwenden wir die folgende Syntax:

```
CREATE TABLE Kunde
(
        Kunde_ID INT(11) AUTO_INCREMENT,
        Vorname VARCHAR(255) NULL,
```

CREATE TABLE -> Text

Um (z.B.) in der Tabelle „Kunde“ auch alle Attribute einzutragen, verwenden wir die folgende Syntax:

```
CREATE TABLE Kunde
(
        Kunde_ID INT(11) AUTO_INCREMENT,
        Vorname VARCHAR(255) NULL,
```

Attribute vom Typ VARCHAR können **Texte** abspeichern.

Der Parameter in der Klammer bestimmt die Anzahl der **maximal speicherbaren Zeichen**.

Früher war ein Wert größer 255 nicht zulässig. Dies änderte sich mittlerweile bei vielen Dialekten. Um allerdings möglichst kompatibel zu sein, werden wir diesen (ehemaligen) Maximalwert nicht überschreiten.

CREATE TABLE -> „leere“ Attributwerte

Um (z.B.) in der Tabelle „Kunde“ auch alle Attribute einzutragen, verwenden wir die folgende Syntax:

```
CREATE TABLE Kunde
(
        Kunde_ID INT(11) AUTO_INCREMENT,
        Vorname VARCHAR(255) NULL,
```

Falls Datensätze abgespeichert werden dürfen, bei denen bestimmte Attribut-Werte **undefiniert** bleiben, so kann dies durch den Constraint NULL explizit mitgeteilt werden.

(Dies ist allerdings üblicherweise ohnehin die Default-Einstellung)

CREATE TABLE -> Attribut Typ Constraint

Um (z.B.) in der Tabelle „Kunde“ auch alle Attribute einzutragen, verwenden wir die folgende Syntax:

```
CREATE TABLE Kunde
(
        Kunde_ID INT(11) AUTO_INCREMENT,
        Vorname VARCHAR(255) NULL,
        Nachname VARCHAR(255) NOT NULL,
        Email VARCHAR(255) NOT NULL,
```

CREATE TABLE -> „nicht-leere“ Attributwerte

Um (z.B.) in der Tabelle „Kunde“ auch alle Attribute einzutragen, verwenden wir die folgende Syntax:

```
CREATE TABLE Kunde
(
        Kunde_ID INT(11) AUTO_INCREMENT,
        Vorname VARCHAR(255) NULL,
        Nachname VARCHAR(255) NOT NULL,
        Email VARCHAR(255) NOT NULL,
```

Falls bestimmte Attribut-Werte **nicht undefiniert** bleiben dürfen,
so muss dies durch den Constraint NOT NULL explizit mitgeteilt werden.

Dieser Constraint wird allerdings nur im **„Strict Mode“** berücksichtigt.
Wie dieser in MariaDB geändert werden kann, zeigen wir am Ende der Vorlesung.

CREATE TABLE -> PRIMARY KEY

Um (z.B.) in der Tabelle „Kunde“ auch alle Attribute einzutragen, verwenden wir die folgende Syntax:

```
CREATE TABLE Kunde
(
        Kunde_ID INT(11) AUTO_INCREMENT,
        Vorname VARCHAR(255) NULL,
        Nachname VARCHAR(255) NOT NULL,
        Email VARCHAR(255) NOT NULL,
        PRIMARY KEY(Kunde_ID)
);
```

CREATE TABLE -> Kalenderdatum

```
CREATE TABLE Abrechnung
(
        Abrechnung_ID INT(11) AUTO_INCREMENT,
        Kunde_ID INT(11) NULL,
        Datum DATE,
        PRIMARY KEY(Abrechnung_ID)
);
```

CREATE TABLE -> Kommazahl

```
CREATE TABLE Produkt
(
        Produkt_ID INT(11) AUTO_INCREMENT,
        Hersteller_ID INT(11) NOT NULL,
        Produkt_Name VARCHAR(255) NOT NULL,
        Euro_Preis FLOAT(7,2) NOT NULL,
        PRIMARY KEY(Produkt_ID)
);
```

Für Kommazahlen können wir den Typ FLOAT verwenden.
Der erste Parameter spricht die **maximal zulässige Gesamtzahl aller Ziffern** an.
Der zweite Parameter gibt die **Anzahl der Nachkommastellen** an.

CREATE TABLE -> Fremdschlüssel

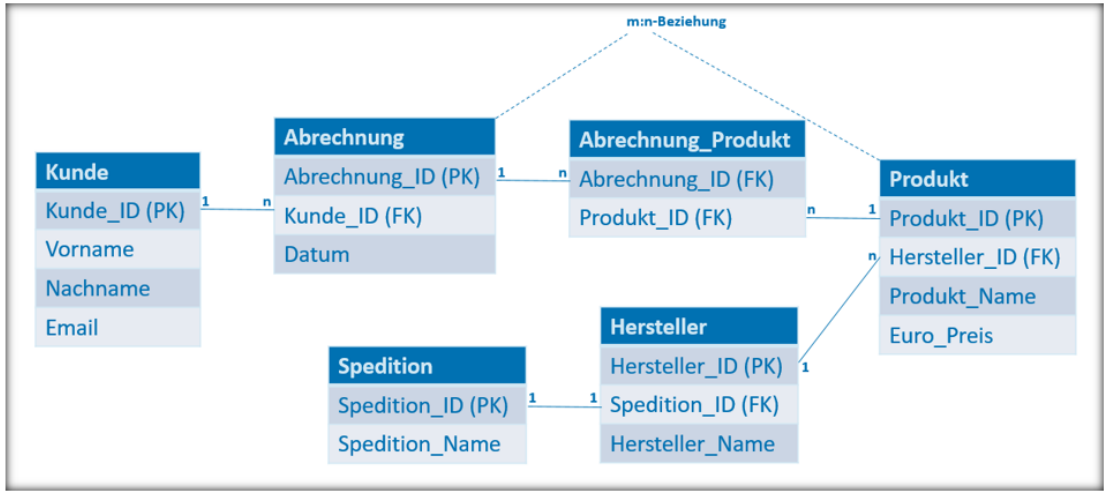
```
CREATE TABLE Abrechnung
(
        Abrechnung_ID INT(11) AUTO_INCREMENT,
        Kunde_ID INT(11) NULL,
        Datum DATE,
        PRIMARY KEY(Abrechnung_ID) ,
        FOREIGN KEY(Kunde_ID) REFERENCES Kunde(Kunde_ID)
);
```

Fremdschlüsselüberprüfung und Referentielle Integrität

- Wir haben nun erste Beispiele kennen gelernt, bei denen wir die **Beziehung zwischen Tabellen** berücksichtigen müssen. Aktuell gilt dies in folgenden Punkten:
  - Wir können **Tabellen mit Fremdschlüsseln erst dann erstellen**, wenn bereits all jene Tabellen eingeführt worden sind, auf die sich der (oder die) Fremdschlüssel beziehen (bzw.: *„auf die sie referenzieren“*).
  - Wir können eine **Tabelle „T“ in einem Schema „S“ nur dann löschen**, wenn es keine anderen Tabellen in **S** gibt, die sich (mittels Fremdschlüssel) auf die Tabelle **T** beziehen.
- Verstöße gegen diese Punkte verletzen die **„Referentielle Integrität“**. Sie werden uns vom Compiler mittels **„Fremdschlüsselüberprüfung“** aufgezeigt.

Gemeinsame Übung („Live-Coding“) -> A_02_01_01

Aufgabe_02_01_01



Aufgabenstellung:
Implementieren Sie bitte das obige Schema.