



Datenbanken und SQL

(Woche 3 - Tag 1)

Agenda

Abfragen (über MEHRERE Tabelle)

- Definition + Motivation
- 2 „benachbarte“ Tabellen
 - Definition + Motivation
 - Beispielaufgabe
 - CROSS JOIN zweier benachbarter Tabellen
 - Definition + Syntax
 - Ergänzende WHERE-Klausel („technische“ und „inhaltliche“ Bedingung)
- „Beliebige“ (Anzahl und Auswahl) Tabellen
 - Beispielaufgabe
 - Technik zur Ermittlung der benötigten Tabellen
 - Ergänzende WHERE-Klausel (verknüpfte „technische“ Bedingungen)

Abfragen über **MEHRERE** Tabellen

Definition + Motivation

- Wir haben im Rahmen der Normalisierung „polythematische“ Tabellen in „monothematische“ aufgelöst. Damit haben wir dann zwar einerseits den Qualitätskriterien der **3. Normalform** entsprochen, sorgten aber andererseits auch dafür, dass (gedanklich) zusammengehörige Informationen - anschaulich gesprochen - regelrecht „auseinander gerissen“ wurden.
- Bei Abfragen über **MEHRERE Tabellen** hinweg werden wir nun lernen, diese Informationen (temporär) wieder zusammenzuführen.
- Eine solche „Temporäre Zusammenführung“ wird in der Fachsprache als **„JOIN“** bezeichnet. („Temporär“ im Sinne von: Die Struktur der implementierten Datenbank wird **nicht** geändert!)
- Es existieren unterschiedliche Formen des JOINS. Wir betrachten heute eine sehr gebräuchliche Version, für die es daher auch eine **Kurzschreibweise** gibt.

2 „benachbarte“ Tabellen

Definition + Motivation

- Unter „benachbarten“ Tabellen (anschaulich gesprochen, KEINE Fachsprache) sollen Tabellen verstanden werden, die in einer **direkten Beziehung** zueinander stehen. (Bei denen also eine der beiden Tabellen einen Fremdschlüssel besitzt, der auf die andere Tabelle referenziert.)
- Beim CROSS JOIN zweier „benachbarter“ Tabellen werden wir uns dann erstmalig die Funktionsweise eines JOINS anschauen können und dabei lernen, welche **Vorgehensweise** und **Syntax** hierfür notwendig sein wird.
- Die dort gelernten Schritte werden wir dann relativ leicht auf einen JOIN übertragen können, der sich auf **mehr als 2 Tabellen** erstreckt.

Beispielaufgabe

- Für jeden Hersteller sollen alle Attribute und die jeweils zugehörige Spedition **ausgegeben werden**. (Auch bei jeder Spedition sollen alle Attribute ausgegeben werden.)
- Wir betrachten zunächst beide Tabelle einzeln:

SELECT * FROM Hersteller;

Hersteller_ID	Spedition_ID	Hersteller_Name
1	2	Contrabit
2	1	AntiByte
3	3	UltraBug
4	5	Hatnix 1992
5	4	Ladenhut AG

SELECT * FROM Spedition;

Spedition_ID	Spedition_Name
1	Speedvan GmbH
2	RocketLogistic AG
3	Turbo Transport
4	Parktnur
5	Kriegtnix
6	Ganzal Lein

Beispielaufgabe

- Für jeden Hersteller sollen alle Attribute und die jeweils zugehörige Spedition **ausgegeben werden**. (Auch bei jeder Spedition sollen alle Attribute ausgegeben werden.)
- Wir betrachten zunächst beide Tabelle einzeln:

SELECT * FROM Hersteller;

Hersteller_ID	Spedition_ID	Hersteller_Name
1	2	Contrabit
2	1	AntiByte
3	3	UltraBug
4	5	Hatnix 1992
5	4	Ladenhut AG

SELECT * FROM Spedition;

Spedition_ID	Spedition_Name
1	Speedvan GmbH
2	RocketLogistic AG
3	Turbo Transport
4	Parktnur
5	Kriegtnix
6	Ganzal Lein

Beispielaufgabe

- Für jeden Hersteller sollen alle Attribute und die jeweils zugehörige Spedition **ausgegeben werden**. (Auch bei jeder Spedition sollen alle Attribute ausgegeben werden.)
- Wir betrachten zunächst beide Tabelle einzeln:

SELECT * FROM Hersteller;

Hersteller_ID	Spedition_ID	Hersteller_Name
1	2	Contrabit
2	1	AntiByte
3	3	UltraBug
4	5	Hatnix 1992
5	4	Ladenhut AG

SELECT * FROM Spedition;

Spedition_ID	Spedition_Name
1	Speedvan GmbH
2	RocketLogistic AG
3	Turbo Transport
4	Parktnur
5	Kriegtnix
6	Ganzal Lein

Beispielaufgabe

- Für jeden Hersteller sollen alle Attribute und die jeweils zugehörige Spedition **ausgegeben werden**. (Auch bei jeder Spedition sollen alle Attribute ausgegeben werden.)
- Wir betrachten zunächst beide Tabelle einzeln:

SELECT * FROM Hersteller;

Hersteller_ID	Spedition_ID	Hersteller_Name
1	2	Contrabit
2	1	AntiByte
3	3	UltraBug
4	5	Hatnix 1992
5	4	Ladenhut AG

SELECT * FROM Spedition;

Spedition_ID	Spedition_Name
1	Speedvan GmbH
2	RocketLogistic AG
3	Turbo Transport
4	Parktnur
5	Kriegtnix
6	Ganzal Lein

Beispielaufgabe

- Für jeden Hersteller sollen alle Attribute und die jeweils zugehörige Spedition **ausgegeben werden**. (Auch bei jeder Spedition sollen alle Attribute ausgegeben werden.)
- Wir betrachten zunächst beide Tabelle einzeln:

SELECT * FROM Hersteller;

Hersteller_ID	Spedition_ID	Hersteller_Name
1	2	Contrabit
2	1	AntiByte
3	3	UltraBug
4	5	Hatnix 1992
5	4	Ladenhut AG

SELECT * FROM Spedition;

Spedition_ID	Spedition_Name
1	Speedvan GmbH
2	RocketLogistic AG
3	Turbo Transport
4	Parktnur
5	Kriegtnix
6	Ganzal Lein

Beispielaufgabe

- Für jeden Hersteller sollen alle Attribute und die jeweils zugehörige Spedition **ausgegeben werden**. (Auch bei jeder Spedition sollen alle Attribute ausgegeben werden.)
- Wir betrachten zunächst beide Tabelle einzeln:

SELECT * FROM Hersteller;

Hersteller_ID	Spedition_ID	Hersteller_Name
1	2	Contrabit
2	1	AntiByte
3	3	UltraBug
4	5	Hatnix 1992
5	4	Ladenhut AG

SELECT * FROM Spedition;

Spedition_ID	Spedition_Name
1	Speedvan GmbH
2	RocketLogistic AG
3	Turbo Transport
4	Parktnur
5	Kriegtnix
6	Ganzal Lein

KEINEM Hersteller zugeordnet

CROSS JOIN – Definition + Syntax

Der CROSS JOIN bildet eine „**Kombinations-Tabelle**“ bei der jeder Datensatz der einen Tabelle mit jedem Datensatz der anderen Tabelle kombiniert wird:

```
SELECT * FROM Hersteller , Spedition;
```

Hersteller_ID	Spedition_ID	Hersteller_Name	Spedition_ID	Spedition_Name
1	2	Contrabit	1	Speedvan GmbH
2	1	AntiByte	1	Speedvan GmbH
3	3	UltraBug	1	Speedvan GmbH
4	5	Hatnix 1992	1	Speedvan GmbH
5	4	Ladenhut AG	1	Speedvan GmbH
1	2	Contrabit	2	RocketLogistic AG
2	1	AntiByte	2	RocketLogistic AG
3	3	UltraBug	2	RocketLogistic AG
4	5	Hatnix 1992	2	RocketLogistic AG
5	4	Ladenhut AG	2	RocketLogistic AG
1	2	Contrabit	3	Turbo Transport
2	1	AntiByte	3	Turbo Transport
3	3	UltraBug	3	Turbo Transport
4	5	Hatnix 1992	3	Turbo Transport
5	4	Ladenhut AG	3	Turbo Transport
1	2	Contrabit	4	Parktnur
2	1	AntiByte	4	Parktnur
3	3	UltraBug	4	Parktnur
4	5	Hatnix 1992	4	Parktnur
5	4	Ladenhut AG	4	Parktnur
1	2	Contrabit	5	Kriegtnix
2	1	AntiByte	5	Kriegtnix
3	3	UltraBug	5	Kriegtnix
4	5	Hatnix 1992	5	Kriegtnix
5	4	Ladenhut AG	5	Kriegtnix
1	2	Contrabit	6	Ganzal Lein
2	1	AntiByte	6	Ganzal Lein
3	3	UltraBug	6	Ganzal Lein
4	5	Hatnix 1992	6	Ganzal Lein
5	4	Ladenhut AG	6	Ganzal Lein

Alle Kombinationen von Speedvan mit allen Herstellern

CROSS JOIN – Definition + Syntax

Der CROSS JOIN bildet eine „Kombinations-Tabelle“ bei der jeder Datensatz der einen Tabelle mit jedem Datensatz der anderen Tabelle kombiniert wird:

```
SELECT * FROM Hersteller , Spedition;
```

Hersteller_ID	Spedition_ID	Hersteller_Name	Spedition_ID	Spedition_Name
1	2	Contrabit	1	Speedvan GmbH
2	1	AntiByte	1	Speedvan GmbH
3	3	UltraBug	1	Speedvan GmbH
4	5	Hatnix 1992	1	Speedvan GmbH
5	4	Ladenhut AG	1	Speedvan GmbH
1	2	Contrabit	2	RocketLogistic AG
2	1	AntiByte	2	RocketLogistic AG
3	3	UltraBug	2	RocketLogistic AG
4	5	Hatnix 1992	2	RocketLogistic AG
5	4	Ladenhut AG	2	RocketLogistic AG
1	2	Contrabit	3	Turbo Transport
2	1	AntiByte	3	Turbo Transport
3	3	UltraBug	3	Turbo Transport
4	5	Hatnix 1992	3	Turbo Transport
5	4	Ladenhut AG	3	Turbo Transport
1	2	Contrabit	4	Parktnur
2	1	AntiByte	4	Parktnur
3	3	UltraBug	4	Parktnur
4	5	Hatnix 1992	4	Parktnur
5	4	Ladenhut AG	4	Parktnur
1	2	Contrabit	5	Kriegtnix
2	1	AntiByte	5	Kriegtnix
3	3	UltraBug	5	Kriegtnix
4	5	Hatnix 1992	5	Kriegtnix
5	4	Ladenhut AG	5	Kriegtnix
1	2	Contrabit	6	Ganzal Lein
2	1	AntiByte	6	Ganzal Lein
3	3	UltraBug	6	Ganzal Lein
4	5	Hatnix 1992	6	Ganzal Lein
5	4	Ladenhut AG	6	Ganzal Lein

Alle Kombinationen von RocketLogistic mit allen Herstellern

CROSS JOIN – Definition + Syntax

Der CROSS JOIN bildet eine „**Kombinations-Tabelle**“ bei der jeder Datensatz der einen Tabelle mit jedem Datensatz der anderen Tabelle kombiniert wird:

```
SELECT * FROM Hersteller , Spedition;
```

Hersteller_ID	Spedition_ID	Hersteller_Name	Spedition_ID	Spedition_Name
1	2	Contrabit	1	Speedvan GmbH
2	1	AntiByte	1	Speedvan GmbH
3	3	UltraBug	1	Speedvan GmbH
4	5	Hatnix 1992	1	Speedvan GmbH
5	4	Ladenhut AG	1	Speedvan GmbH
1	2	Contrabit	2	RocketLogistic AG
2	1	AntiByte	2	RocketLogistic AG
3	3	UltraBug	2	RocketLogistic AG
4	5	Hatnix 1992	2	RocketLogistic AG
5	4	Ladenhut AG	2	RocketLogistic AG
1	2	Contrabit	3	Turbo Transport
2	1	AntiByte	3	Turbo Transport
3	3	UltraBug	3	Turbo Transport
4	5	Hatnix 1992	3	Turbo Transport
5	4	Ladenhut AG	3	Turbo Transport
1	2	Contrabit	4	Parktnur
2	1	AntiByte	4	Parktnur
3	3	UltraBug	4	Parktnur
4	5	Hatnix 1992	4	Parktnur
5	4	Ladenhut AG	4	Parktnur
1	2	Contrabit	5	Kriegtnix
2	1	AntiByte	5	Kriegtnix
3	3	UltraBug	5	Kriegtnix
4	5	Hatnix 1992	5	Kriegtnix
5	4	Ladenhut AG	5	Kriegtnix
1	2	Contrabit	6	Ganzal Lein
2	1	AntiByte	6	Ganzal Lein
3	3	UltraBug	6	Ganzal Lein
4	5	Hatnix 1992	6	Ganzal Lein
5	4	Ladenhut AG	6	Ganzal Lein

Alle Kombinationen von Turbo Transport mit allen Herstellern

CROSS JOIN – Definition + Syntax

Der CROSS JOIN bildet eine „**Kombinations-Tabelle**“ bei der jeder Datensatz der einen Tabelle mit jedem Datensatz der anderen Tabelle kombiniert wird:

```
SELECT * FROM Hersteller , Spedition;
```

Hersteller_ID	Spedition_ID	Hersteller_Name	Spedition_ID	Spedition_Name
1	2	Contrabit	1	Speedvan GmbH
2	1	AntiByte	1	Speedvan GmbH
3	3	UltraBug	1	Speedvan GmbH
4	5	Hatnix 1992	1	Speedvan GmbH
5	4	Ladenhut AG	1	Speedvan GmbH
1	2	Contrabit	2	RocketLogistic AG
2	1	AntiByte	2	RocketLogistic AG
3	3	UltraBug	2	RocketLogistic AG
4	5	Hatnix 1992	2	RocketLogistic AG
5	4	Ladenhut AG	2	RocketLogistic AG
1	2	Contrabit	3	Turbo Transport
2	1	AntiByte	3	Turbo Transport
3	3	UltraBug	3	Turbo Transport
4	5	Hatnix 1992	3	Turbo Transport
5	4	Ladenhut AG	3	Turbo Transport
1	2	Contrabit	4	Parktnur
2	1	AntiByte	4	Parktnur
3	3	UltraBug	4	Parktnur
4	5	Hatnix 1992	4	Parktnur
5	4	Ladenhut AG	4	Parktnur
1	2	Contrabit	5	Kriegtnix
2	1	AntiByte	5	Kriegtnix
3	3	UltraBug	5	Kriegtnix
4	5	Hatnix 1992	5	Kriegtnix
5	4	Ladenhut AG	5	Kriegtnix
1	2	Contrabit	6	Ganzal Lein
2	1	AntiByte	6	Ganzal Lein
3	3	UltraBug	6	Ganzal Lein
4	5	Hatnix 1992	6	Ganzal Lein
5	4	Ladenhut AG	6	Ganzal Lein

Alle Kombinationen von Parktnur mit allen Herstellern

CROSS JOIN – Definition + Syntax

Der CROSS JOIN bildet eine „**Kombinations-Tabelle**“ bei der jeder Datensatz der einen Tabelle mit jedem Datensatz der anderen Tabelle kombiniert wird:

```
SELECT * FROM Hersteller , Spedition;
```

Hersteller_ID	Spedition_ID	Hersteller_Name	Spedition_ID	Spedition_Name
1	2	Contrabit	1	Speedvan GmbH
2	1	AntiByte	1	Speedvan GmbH
3	3	UltraBug	1	Speedvan GmbH
4	5	Hatnix 1992	1	Speedvan GmbH
5	4	Ladenhut AG	1	Speedvan GmbH
1	2	Contrabit	2	RocketLogistic AG
2	1	AntiByte	2	RocketLogistic AG
3	3	UltraBug	2	RocketLogistic AG
4	5	Hatnix 1992	2	RocketLogistic AG
5	4	Ladenhut AG	2	RocketLogistic AG
1	2	Contrabit	3	Turbo Transport
2	1	AntiByte	3	Turbo Transport
3	3	UltraBug	3	Turbo Transport
4	5	Hatnix 1992	3	Turbo Transport
5	4	Ladenhut AG	3	Turbo Transport
1	2	Contrabit	4	Parktnur
2	1	AntiByte	4	Parktnur
3	3	UltraBug	4	Parktnur
4	5	Hatnix 1992	4	Parktnur
5	4	Ladenhut AG	4	Parktnur
1	2	Contrabit	5	Kriegtnix
2	1	AntiByte	5	Kriegtnix
3	3	UltraBug	5	Kriegtnix
4	5	Hatnix 1992	5	Kriegtnix
5	4	Ladenhut AG	5	Kriegtnix
1	2	Contrabit	6	Ganzal Lein
2	1	AntiByte	6	Ganzal Lein
3	3	UltraBug	6	Ganzal Lein
4	5	Hatnix 1992	6	Ganzal Lein
5	4	Ladenhut AG	6	Ganzal Lein

Alle Kombinationen von Kriegtnix mit allen Herstellern

CROSS JOIN – Definition + Syntax

Der CROSS JOIN bildet eine „Kombinations-Tabelle“ bei der jeder Datensatz der einen Tabelle mit jedem Datensatz der anderen Tabelle kombiniert wird:

```
SELECT * FROM Hersteller , Spedition;
```

Hersteller_ID	Spedition_ID	Hersteller_Name	Spedition_ID	Spedition_Name
1	2	Contrabit	1	Speedvan GmbH
2	1	AntiByte	1	Speedvan GmbH
3	3	UltraBug	1	Speedvan GmbH
4	5	Hatnix 1992	1	Speedvan GmbH
5	4	Ladenhut AG	1	Speedvan GmbH
1	2	Contrabit	2	RocketLogistic AG
2	1	AntiByte	2	RocketLogistic AG
3	3	UltraBug	2	RocketLogistic AG
4	5	Hatnix 1992	2	RocketLogistic AG
5	4	Ladenhut AG	2	RocketLogistic AG
1	2	Contrabit	3	Turbo Transport
2	1	AntiByte	3	Turbo Transport
3	3	UltraBug	3	Turbo Transport
4	5	Hatnix 1992	3	Turbo Transport
5	4	Ladenhut AG	3	Turbo Transport
1	2	Contrabit	4	Parktnur
2	1	AntiByte	4	Parktnur
3	3	UltraBug	4	Parktnur
4	5	Hatnix 1992	4	Parktnur
5	4	Ladenhut AG	4	Parktnur
1	2	Contrabit	5	Kriegtnix
2	1	AntiByte	5	Kriegtnix
3	3	UltraBug	5	Kriegtnix
4	5	Hatnix 1992	5	Kriegtnix
5	4	Ladenhut AG	5	Kriegtnix
1	2	Contrabit	6	Ganzal Lein
2	1	AntiByte	6	Ganzal Lein
3	3	UltraBug	6	Ganzal Lein
4	5	Hatnix 1992	6	Ganzal Lein
5	4	Ladenhut AG	6	Ganzal Lein

Alle Kombinationen von Ganzal Lein mit allen Herstellern

CROSS JOIN – Definition + Syntax

Der CROSS JOIN bildet eine „**Kombinations-Tabelle**“ bei der jeder Datensatz der einen Tabelle mit jedem Datensatz der anderen Tabelle kombiniert wird:

```
SELECT * FROM Hersteller , Spedition;
```

Hersteller_ID	Spedition_ID	Hersteller_Name	Spedition_ID	Spedition_Name
1	2	Contrabit	1	Speedvan GmbH
2	1	AntiByte	1	Speedvan GmbH
3	3	UltraBug	1	Speedvan GmbH
4	5	Hatnix 1992	1	Speedvan GmbH
5	4	Ladenhut AG	1	Speedvan GmbH
1	2	Contrabit	2	RocketLogistic AG
2	1	AntiByte	2	RocketLogistic AG
3	3	UltraBug	2	RocketLogistic AG
4	5	Hatnix 1992	2	RocketLogistic AG
5	4	Ladenhut AG	2	RocketLogistic AG
1	2	Contrabit	3	Turbo Transport
2	1	AntiByte	3	Turbo Transport
3	3	UltraBug	3	Turbo Transport
4	5	Hatnix 1992	3	Turbo Transport
5	4	Ladenhut AG	3	Turbo Transport
1	2	Contrabit	4	Parktnur
2	1	AntiByte	4	Parktnur
3	3	UltraBug	4	Parktnur
4	5	Hatnix 1992	4	Parktnur
5	4	Ladenhut AG	4	Parktnur
1	2	Contrabit	5	Kriegtnix
2	1	AntiByte	5	Kriegtnix
3	3	UltraBug	5	Kriegtnix
4	5	Hatnix 1992	5	Kriegtnix
5	4	Ladenhut AG	5	Kriegtnix
1	2	Contrabit	6	Ganzal Lein
2	1	AntiByte	6	Ganzal Lein
3	3	UltraBug	6	Ganzal Lein
4	5	Hatnix 1992	6	Ganzal Lein
5	4	Ladenhut AG	6	Ganzal Lein

5 Hersteller multipliziert mit **6 Speditionen**
=
30 Kombinationen

CROSS JOIN – Motivation

Die Kombination aller Datensätze beider Tabellen bildet zwar auch viele „**sinnlose**“ Kombinationen, Vorteil aber ist, dass auf diese Weise keine **korrekte** Kombination „übersehen“ wird, schlicht, weil eben **jede Kombination** berücksichtigt wird:

```
SELECT * FROM Hersteller , Spedition;
```

Hersteller_ID	Spedition_ID	Hersteller_Name	Spedition_ID	Spedition_Name
1	2	Contrabit	1	Speedvan GmbH
2	1	AntiByte	1	Speedvan GmbH
3	3	UltraBug	1	Speedvan GmbH
4	5	Hatnix 1992	1	Speedvan GmbH
5	4	Ladenhut AG	1	Speedvan GmbH
1	2	Contrabit	2	RocketLogistic AG
2	1	AntiByte	2	RocketLogistic AG
3	3	UltraBug	2	RocketLogistic AG
4	5	Hatnix 1992	2	RocketLogistic AG
5	4	Ladenhut AG	2	RocketLogistic AG
1	2	Contrabit	3	Turbo Transport
2	1	AntiByte	3	Turbo Transport
3	3	UltraBug	3	Turbo Transport
4	5	Hatnix 1992	3	Turbo Transport
5	4	Ladenhut AG	3	Turbo Transport
1	2	Contrabit	4	Parktnur
2	1	AntiByte	4	Parktnur
3	3	UltraBug	4	Parktnur
4	5	Hatnix 1992	4	Parktnur
5	4	Ladenhut AG	4	Parktnur
1	2	Contrabit	5	Kriegtnix
2	1	AntiByte	5	Kriegtnix
3	3	UltraBug	5	Kriegtnix
4	5	Hatnix 1992	5	Kriegtnix
5	4	Ladenhut AG	5	Kriegtnix
1	2	Contrabit	6	Ganzal Lein
2	1	AntiByte	6	Ganzal Lein
3	3	UltraBug	6	Ganzal Lein
4	5	Hatnix 1992	6	Ganzal Lein
5	4	Ladenhut AG	6	Ganzal Lein

CROSS JOIN – Motivation

Die Kombination aller Datensätze beider Tabellen bildet zwar auch viele „sinnlose“ Kombinationen, Vorteil aber ist, dass auf diese Weise keine **korrekte** Kombination „übersehen“ wird, schlicht, weil eben **jede Kombination** berücksichtigt wird:

```
SELECT * FROM Hersteller , Spedition;
```

Hersteller_ID	Spedition_ID	Hersteller_Name	Spedition_ID	Spedition_Name
1	2	Contrabit	1	Speedvan GmbH
2	1	AntiByte	1	Speedvan GmbH
3	3	UltraBug	1	Speedvan GmbH
4	5	Hatnix 1992	1	Speedvan GmbH
5	4	Ladenhut AG	1	Speedvan GmbH
1	2	Contrabit	2	RocketLogistic AG
2	1	AntiByte	2	RocketLogistic AG
3	3	UltraBug	2	RocketLogistic AG
4	5	Hatnix 1992	2	RocketLogistic AG
5	4	Ladenhut AG	2	RocketLogistic AG
1	2	Contrabit	3	Turbo Transport
2	1	AntiByte	3	Turbo Transport
3	3	UltraBug	3	Turbo Transport
4	5	Hatnix 1992	3	Turbo Transport
5	4	Ladenhut AG	3	Turbo Transport
1	2	Contrabit	4	Parktnur
2	1	AntiByte	4	Parktnur
3	3	UltraBug	4	Parktnur
4	5	Hatnix 1992	4	Parktnur
5	4	Ladenhut AG	4	Parktnur
1	2	Contrabit	5	Kriegtnix
2	1	AntiByte	5	Kriegtnix
3	3	UltraBug	5	Kriegtnix
4	5	Hatnix 1992	5	Kriegtnix
5	4	Ladenhut AG	5	Kriegtnix
1	2	Contrabit	6	Ganzal Lein
2	1	AntiByte	6	Ganzal Lein
3	3	UltraBug	6	Ganzal Lein
4	5	Hatnix 1992	6	Ganzal Lein
5	4	Ladenhut AG	6	Ganzal Lein

Ergänzende WHERE-Klausel

Um uns nun aber von den sinnlosen Kombinationen befreien zu können, bietet es sich an, mit der folgenden WHERE-Klausel buchstäblich „aufzuräumen“:

```
SELECT * FROM Hersteller , Spedition WHERE Hersteller.Spedition_ID=Spedition.Spedition_ID ;
```

Hersteller_ID	Spedition_ID	Hersteller_Name	Spedition_ID	Spedition_Name
1	②	Contrabit	②	RocketLogistic AG
2	①	AntiByte	①	Speedvan GmbH
3	③	UltraBug	③	Turbo Transport
4	⑤	Hatnix 1992	⑤	Kriegtnix
5	④	Ladenhut AG	④	Parktnur

Erläuterung:

Um den Primärschlüssel in Spedition vom gleichnamigen Fremdschlüssel in Hersteller unterscheiden zu können, verwenden wir die Schreibweise **Tabellenname.Spaltenname**, die zudem im Umgang mit dem Editor sehr komfortabel ist (siehe späteres „Live-Coding“).

„Beliebig“ viele Tabellen

Beispielaufgabe

- Gesucht werden Vor- und Nachnamen all jener Kunden, die bereits mindestens 1-mal ein Produkt kauften, das teurer als 30 Euro ist. Die Ausgabe soll sortiert nach Nachnamen aufsteigend (bzw. alphabetisch) erscheinen.
- Damit ist dann schon einmal klar, welche Attribute ausgegeben werden sollen, bzw. „was“ hinter SELECT (und vor FROM) notiert werden muss:

Beispielaufgabe - Attribute

- Gesucht werden Vor- und Nachnamen all jener Kunden, die bereits mindestens 1-mal ein Produkt kauften, das teurer als 30 Euro ist. Die Ausgabe soll sortiert nach Nachnamen aufsteigend (bzw. alphabetisch) erscheinen.
- Damit ist dann schon einmal klar, welche Attribute ausgegeben werden sollen, bzw. „was“ hinter SELECT (und vor FROM) notiert werden muss:

SELECT Vorname, Nachname **FROM** ...

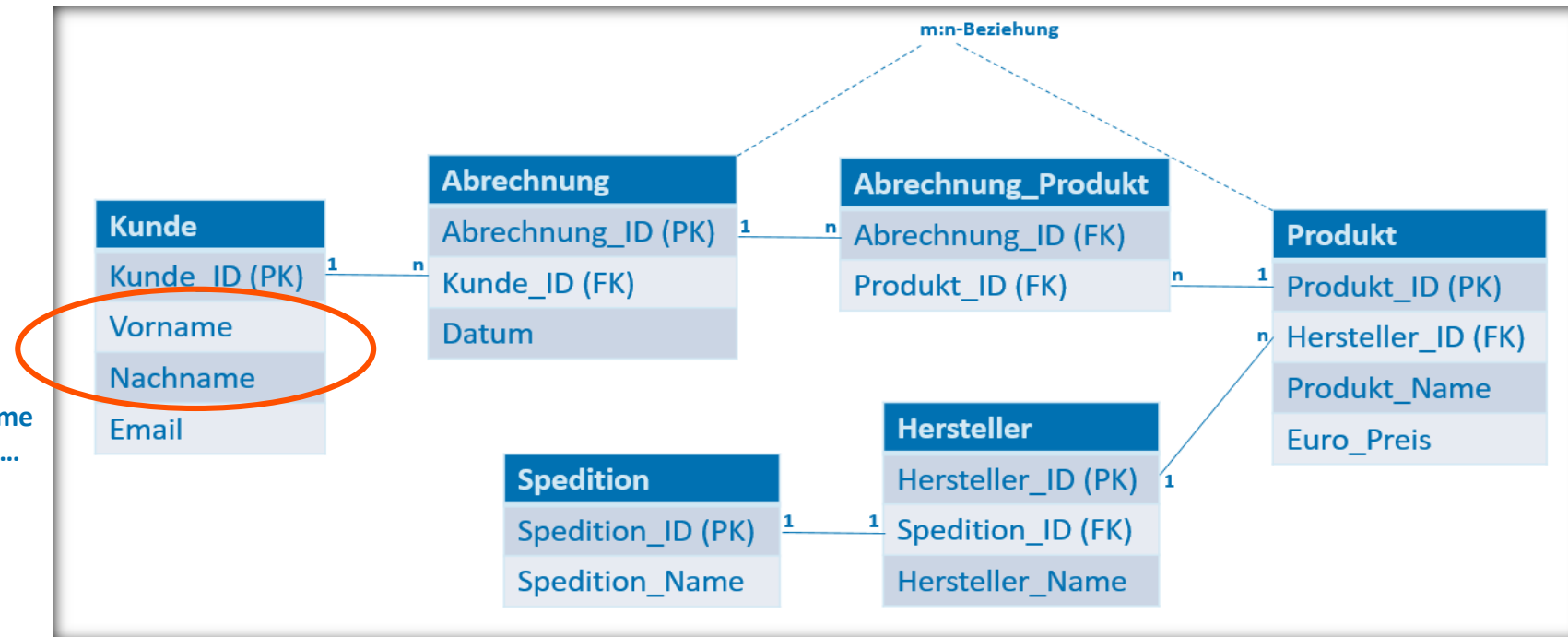
Beispielaufgabe – („inhaltliche“) **WHERE-Klausel**

- Gesucht werden Vor- und Nachnamen all jener Kunden, die bereits mindestens 1-mal ein Produkt kauften, das teurer als 30 Euro ist. Die Ausgabe soll sortiert nach Nachnamen aufsteigend (bzw. alphabetisch) erscheinen.
- Auch die Formulierung der WHERE-Klausel ergibt sich unmittelbar aus der Aufgabenstellung:

```
SELECT Vorname, Nachname FROM ...  
... WHERE Euro_Preis > 30 ...
```

Beispielaufgabe – benötigte Tabellen

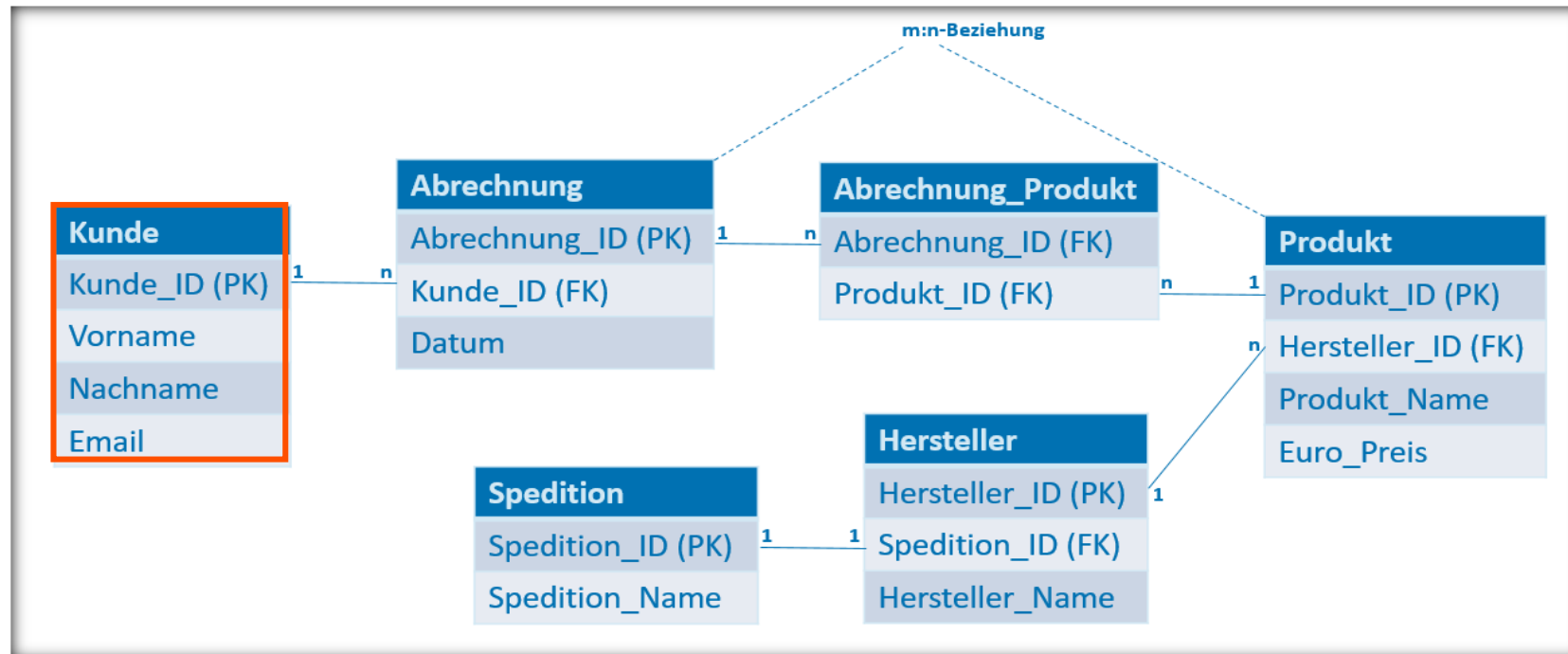
- Gesucht werden Vor- und Nachnamen all jener Kunden, die bereits mindestens 1-mal ein Produkt kauften, das teurer als 30 Euro ist. Die Ausgabe soll sortiert nach Nachnamen aufsteigend (bzw. alphabetisch) erscheinen.
- Wir betrachten hierfür das Datenbankschema von „Geld_her“:



Da wir Vor- und Nachname
ausgeben lassen wollen ...

Beispielaufgabe – benötigte Tabellen

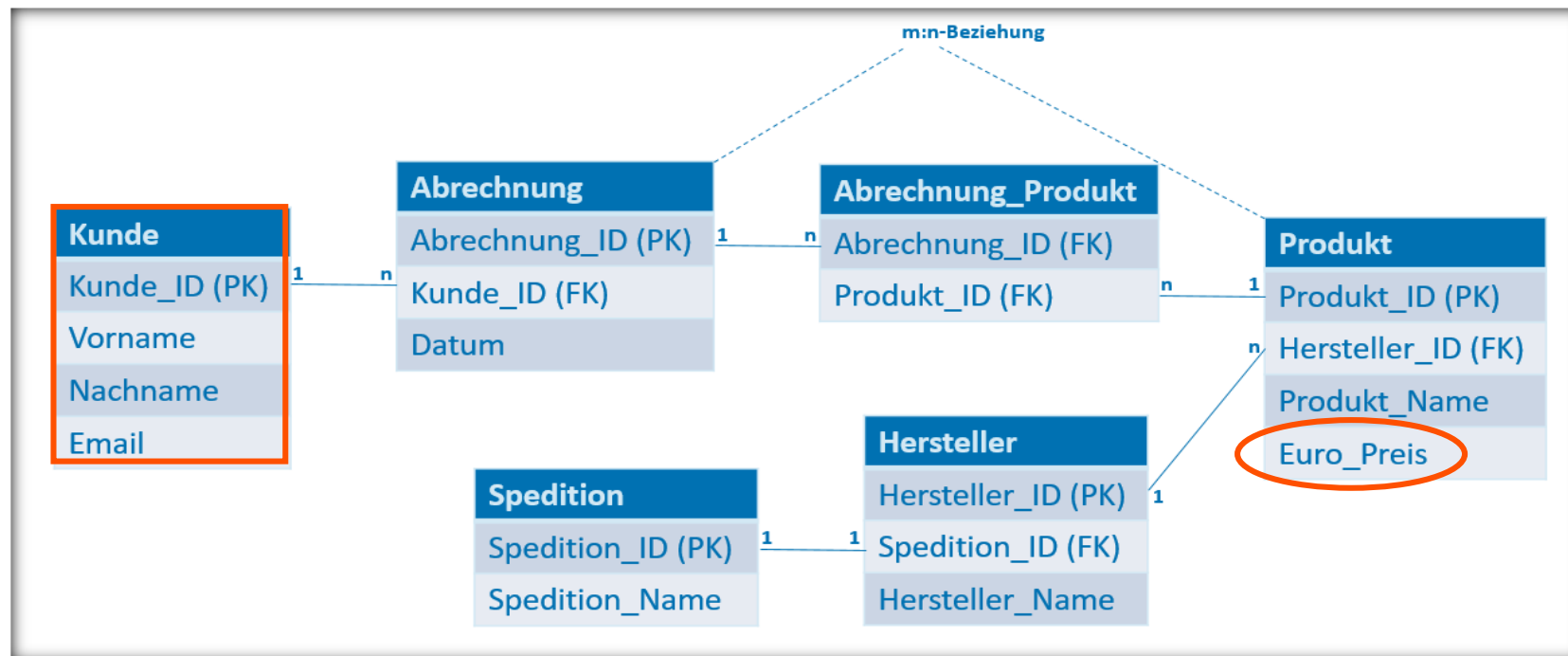
- Gesucht werden Vor- und Nachnamen all jener Kunden, die bereits mindestens 1-mal ein Produkt kauften, das teurer als 30 Euro ist. Die Ausgabe soll sortiert nach Nachnamen aufsteigend (bzw. alphabetisch) erscheinen.
- Wir betrachten hierfür das Datenbankschema von „Geld_her“:



... benötigen wir zunächst
die Tabelle **Kunde**

Beispielaufgabe – benötigte Tabellen

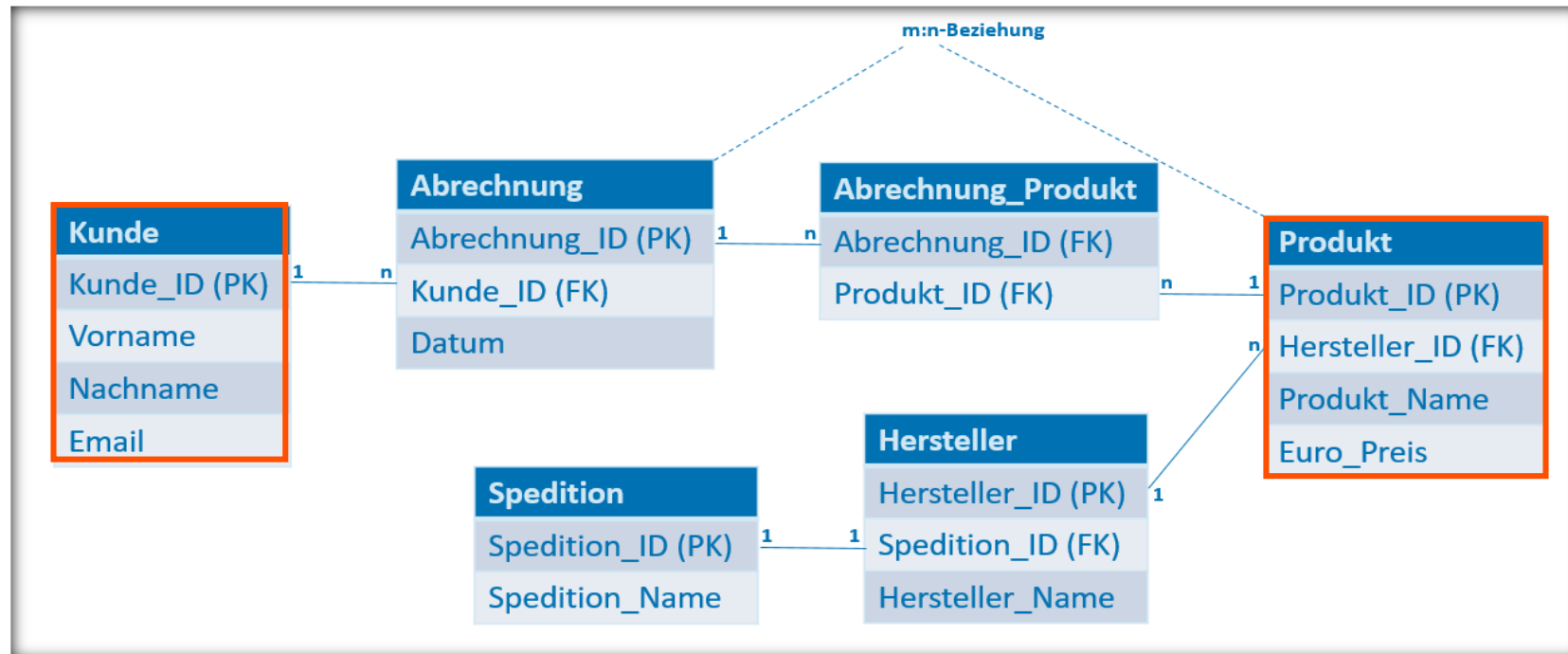
- Gesucht werden Vor- und Nachnamen all jener Kunden, die bereits mindestens 1-mal ein Produkt kauften, das teurer als 30 Euro ist. Die Ausgabe soll sortiert nach Nachnamen aufsteigend (bzw. alphabetisch) erscheinen.
- Wir betrachten hierfür das Datenbankschema von „Geld_her“:



Da die Bedingung vom Preis spricht ...

Beispielaufgabe – benötigte Tabellen

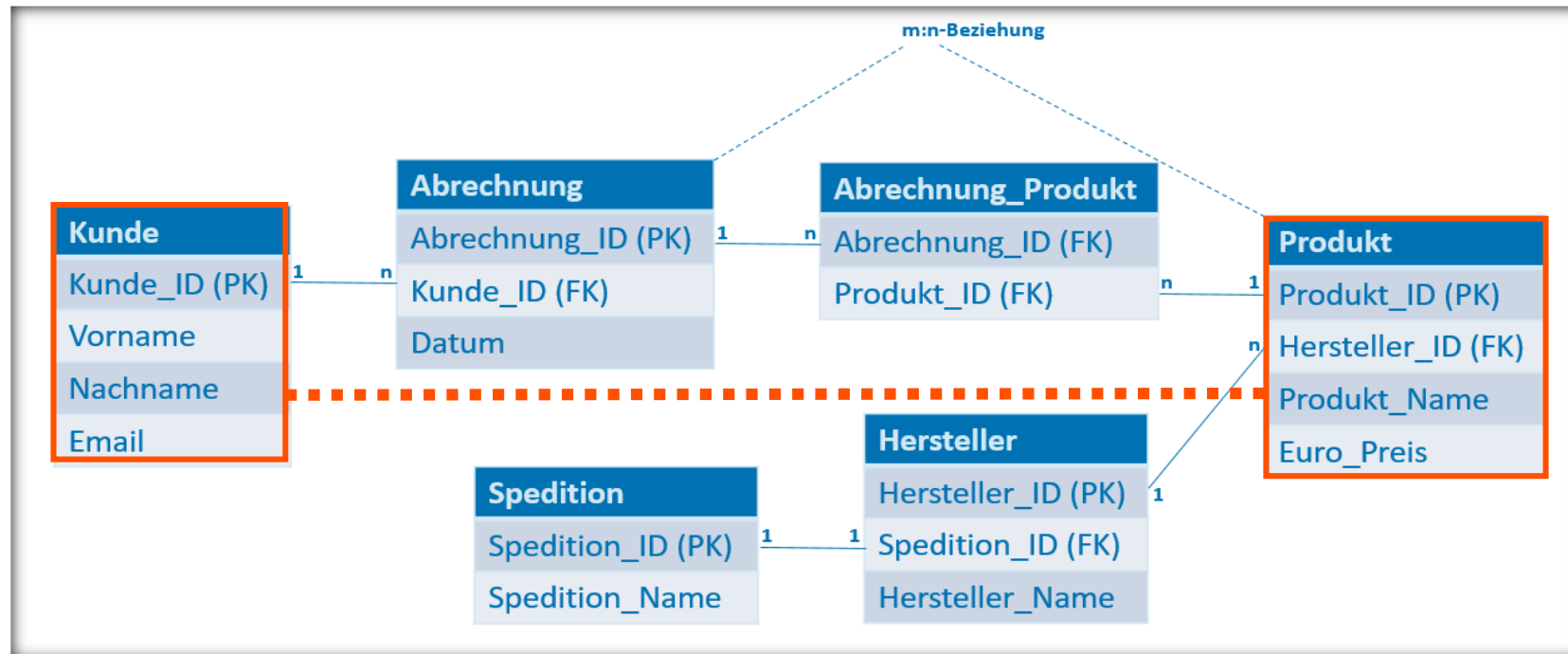
- Gesucht werden Vor- und Nachnamen all jener Kunden, die bereits mindestens 1-mal ein Produkt kauften, das teurer als 30 Euro ist. Die Ausgabe soll sortiert nach Nachnamen aufsteigend (bzw. alphabetisch) erscheinen.
- Wir betrachten hierfür das Datenbankschema von „Geld_her“:



... benötigen wir ebenfalls die Tabelle **Produkt**

Beispielaufgabe – benötigte Tabellen

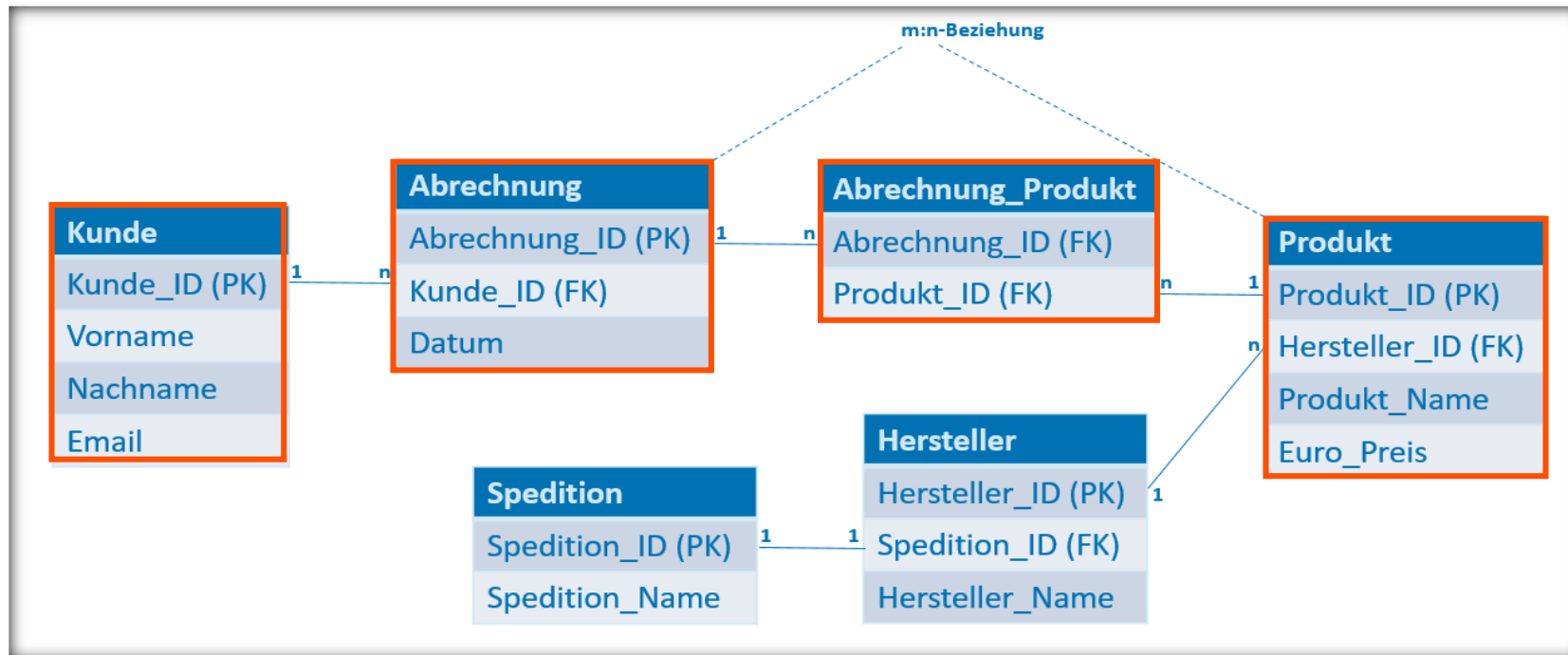
- Gesucht werden Vor- und Nachnamen all jener Kunden, die bereits mindestens 1-mal ein Produkt kauften, das teurer als 30 Euro ist. Die Ausgabe soll sortiert nach Nachnamen aufsteigend (bzw. alphabetisch) erscheinen.
- Wir betrachten hierfür das Datenbankschema von „Geld_her“:



Nicht „benachbart“ => Es gibt keine (direkte) Beziehung zwischen Kunde und Produkt:

Beispielaufgabe – benötigte Tabellen

- Gesucht werden Vor- und Nachnamen all jener Kunden, die bereits mindestens 1-mal ein Produkt kauften, das teurer als 30 Euro ist. Die Ausgabe soll sortiert nach Nachnamen aufsteigend (bzw. alphabetisch) erscheinen.
- Wir betrachten hierfür das Datenbankschema von „Geld_her“:



Nicht „benachbart“ => Es gibt keine (direkte) Beziehung zwischen Kunde und Produkt:

Wir benötigen daher auch alle „**dazwischen liegenden**“ Tabellen, denn von Kunde nach Abrechnung (Kunde_ID), von Abrechnung zur Hilfstabelle (Abrechnung_ID) und schließlich von Hilfstabelle zu Produkt (Produkt_ID) können wir die indirekte Beziehung darstellen.

Beispielaufgabe – benötigte Tabellen

- Gesucht werden Vor- und Nachnamen all jener Kunden, die bereits mindestens 1-mal ein Produkt kauften, das teurer als 30 Euro ist. Die Ausgabe soll sortiert nach Nachnamen aufsteigend (bzw. alphabetisch) erscheinen.
- Wir „join-en“ also die Tabellen **Kunde**, **Abrechnung**, **Hilfstabelle** und **Produkt**:

```
SELECT Vorname, Nachname  
FROM ...  
WHERE Euro_Preis > 30 ...
```

Beispielaufgabe – benötigte Tabellen

- Gesucht werden Vor- und Nachnamen all jener Kunden, die bereits mindestens 1-mal ein Produkt kauften, das teurer als 30 Euro ist. Die Ausgabe soll sortiert nach Nachnamen aufsteigend (bzw. alphabetisch) erscheinen.
- Wir „join-en“ also die Tabellen **Kunde**, **Abrechnung**, **Hilfstabelle** und **Produkt**:

```
SELECT Vorname, Nachname  
FROM Kunde, Abrechnung, Abrechnung_Produkt, Produkt  
WHERE Euro_Preis > 30 ...
```

Beispielaufgabe – („technische“) WHERE-Klausel

- Gesucht werden Vor- und Nachnamen all jener Kunden, die bereits mindestens 1-mal ein Produkt kauften, das teurer als 30 Euro ist. Die Ausgabe soll sortiert nach Nachnamen aufsteigend (bzw. alphabetisch) erscheinen.
- Auch bei diesem JOIN müssen wir die **Schlüsselbedingungen** eintragen:

```
SELECT Vorname, Nachname  
FROM Kunde, Abrechnung, Abrechnung_Produkt, Produkt  
WHERE Euro_Preis > 30
```

```
AND Kunde.Kunde_ID=Abrechnung.Kunde_ID
```

```
AND Abrechnung.Abrechnung_ID=Abrechnung_Produkt.Abrechnung_ID
```

```
AND Abrechnung_Produkt.Produkt_ID=Produkt.Produkt_ID ...
```

Beispielaufgabe – ORDER BY

- Gesucht werden Vor- und Nachnamen all jener Kunden, die bereits mindestens 1-mal ein Produkt kauften, das teurer als 30 Euro ist. Die Ausgabe soll sortiert nach Nachnamen aufsteigend (bzw. alphabetisch) erscheinen.
- Die Ausgabe soll nach **Nachname (alphabetisch) sortiert** werden:

```
SELECT Vorname, Nachname  
FROM Kunde, Abrechnung, Abrechnung_Produkt, Produkt  
WHERE Euro_Preis > 30
```

```
AND Kunde.Kunde_ID=Abrechnung.Kunde_ID
```

```
AND Abrechnung.Abrechnung_ID=Abrechnung_Produkt.Abrechnung_ID
```

```
AND Abrechnung_Produkt.Produkt_ID=Produkt.Produkt_ID ...
```

Beispielaufgabe – ORDER BY

- Gesucht werden Vor- und Nachnamen all jener Kunden, die bereits mindestens 1-mal ein Produkt kauften, das teurer als 30 Euro ist. Die Ausgabe soll sortiert nach Nachnamen aufsteigend (bzw. alphabetisch) erscheinen.
- Die Ausgabe soll nach **Nachname (alphabetisch) sortiert** werden:

```
SELECT Vorname, Nachname  
FROM Kunde, Abrechnung, Abrechnung_Produkt, Produkt  
WHERE Euro_Preis > 30  
  
      AND Kunde.Kunde_ID=Abrechnung.Kunde_ID  
      AND Abrechnung.Abrechnung_ID=Abrechnung_Produkt.Abrechnung_ID  
      AND Abrechnung_Produkt.Produkt_ID=Produkt.Produkt_ID  
  
ORDER BY Nachname;
```

Beispielaufgabe – DISTINCT

- Gesucht werden Vor- und Nachnamen all jener Kunden, die bereits mindestens 1-mal ein Produkt kauften, das teurer als 30 Euro ist. Die Ausgabe soll sortiert nach Nachnamen aufsteigend (bzw. alphabetisch) erscheinen.
- **Keine Dubletten** (von Kunden, die mehrfach Produkte kauften, die teurer als 30 Euro sind)

```
SELECT DISTINCT Vorname, Nachname  
FROM Kunde, Abrechnung, Abrechnung_Produkt, Produkt  
WHERE Euro_Preis > 30  
  
      AND Kunde.Kunde_ID=Abrechnung.Kunde_ID  
      AND Abrechnung.Abrechnung_ID=Abrechnung_Produkt.Abrechnung_ID  
      AND Abrechnung_Produkt.Produkt_ID=Produkt.Produkt_ID  
  
ORDER BY Nachname;
```

Gemeinsame Übung („Live-Coding“) -> A_03_01_01



Aufgabe_03_01_01

Formulieren Sie bitte entsprechende SQL-Anweisungen für folgende Aufgabestellungen:

- Für jede Abrechnung soll deren Datum, Kunden-ID und Nachnamen des Kunden, der diese Abrechnung einreichte, ausgegeben werden. Ausgabe nach Datum abfallend sortiert, auf 10 Datensätze begrenzt.
- Für alle Produkte soll der Produktname, Preis und Herstellername ausgegeben werden, allerdings nur, wenn das Produkt weniger als 1000 Euro kostet. Ausgabe alphabetisch sortiert (erstes Kriterium Herstellername, 2. Kriterium Produktname).
- Von allen Kunden sollen Vor- und Nachname ausgegeben werden, sofern der jeweilige Kunde mindestens 1-mal das Produkt „tool 2.0“ kaufte. Ausgabe alphabetisch sortiert nach (1.) Nachname und (2.) Vorname. Ausgabe-Dubletten sollen vermieden werden.
- Name aller Produkte, die am 16. Oktober 2021 gekauft wurden. Ausgabe alphabetisch sortiert, Vermeidung von Dubletten. Die ersten 2 Treffer sollen jedoch übersprungen, und nur die 3 folgenden ausgegeben werden.
- Ausgabe aller Kalenderdaten, an denen die Spedition „Speedvan GmbH“ mindestens 1 Produkt der Firma „Geld_her“ transportierte. Erneut sollen Dubletten vermieden werden. Ausgabe chronologisch sortiert. Ausgabe auf 100 Datensätze begrenzt.

WBS TRAINING AG
Lorenzweg 5
D-12099 Berlin
Amtsgericht Berlin HRB 68531
Sitz der Gesellschaft: Berlin

Vorstand:
Heinrich Kronbichler,
Joachim Giese
Aufsichtsrat (Vorsitz): Dr. Daniel Stadler
USA-Adresse: DE 209 768 248

GLS Gemeinschaftsbank eG
IBAN: DE18 4305 0967 1146 1814 00
BIC: GENODEM33GLS



GLS vertriebs GmbH
1000 Berlin
Zustimmung nach Art. 109 Abs. 1 Nr. 1 des Grundgesetzes

Vielen Dank für Ihre Aufmerksamkeit!

