

# EINFÜHRUNG IN DATENBANKEN

PROF. DR. RER. NAT. ALEXANDER VÖß  
INFORM-PROFESSUR

FH AACHEN  
UNIVERSITY OF APPLIED SCIENCES

15.01.2022

# Inhaltsübersicht

---

- Unit 0x00: Formales
- Unit 0x01: Motivation, Fallbeispiel
- Unit 0x02: Grundlagen Datenbanksysteme
- Unit 0x03: Modellierung und Entity-Relationship-Modell I
- Unit 0x04: Entity-Relationship-Modell II
- Unit 0x05: Relationales Modell und Relationale Algebra I
- Unit 0x06: Relationale Algebra II
- Unit 0x07: Normalformen I
- Unit 0x08: Normalformen II
- Unit 0x09: Normalformen III

Fortsetzung nächste Seite

## Inhaltsübersicht Fortsetzung

---

- Unit 0x0a: SQL I
- Unit 0x0b: SQL II
- Unit 0x0c: Transaktionen, Anfrageoptimierung
- Unit 0x0d: ORM und Hibernate
- Unit 0x0e: ODBC/JDBC
- Unit 0x0f: NoSQL
- Unit 0x10: XML



## Danksagung

Manche Grafik, aber auch Aufarbeitung und Zusammenstellung, entstammt den Unterlagen von Herrn Prof. Dr. Jörg Striegnitz.

Für die Bereitstellung und Diskussion meinen ausdrücklichen Dank an dieser Stelle!

# UNIT 0x00

# FORMALES

## Vorlesung und Übung

---

### **Vorlesung (2 SWS)**

- Vogelperspektive
- Definitionen, Grundprinzipien und Zusammenhänge
- Kleinere Beispiele

### **Übung (2 SWS)**

- Umfangreichere und detailliertere Beispiele zu Vorlesungsthemen
- SQL Praktikum

## Modulprüfung und Leistungsnachweis

---

### Reguläre Modulprüfung Jan./Feb. (120 Min.)

- Aufgaben zu Themen der Vorlesung und zu SQL
- Thematisch aufgeteilt in Teil A und Teil B - normalerweise.



#### Ziel

Besseres Zeit- und Lernmanagement.

### Leistungsnachweis Ende Nov./Dez. optional (ca. 50 Min.)

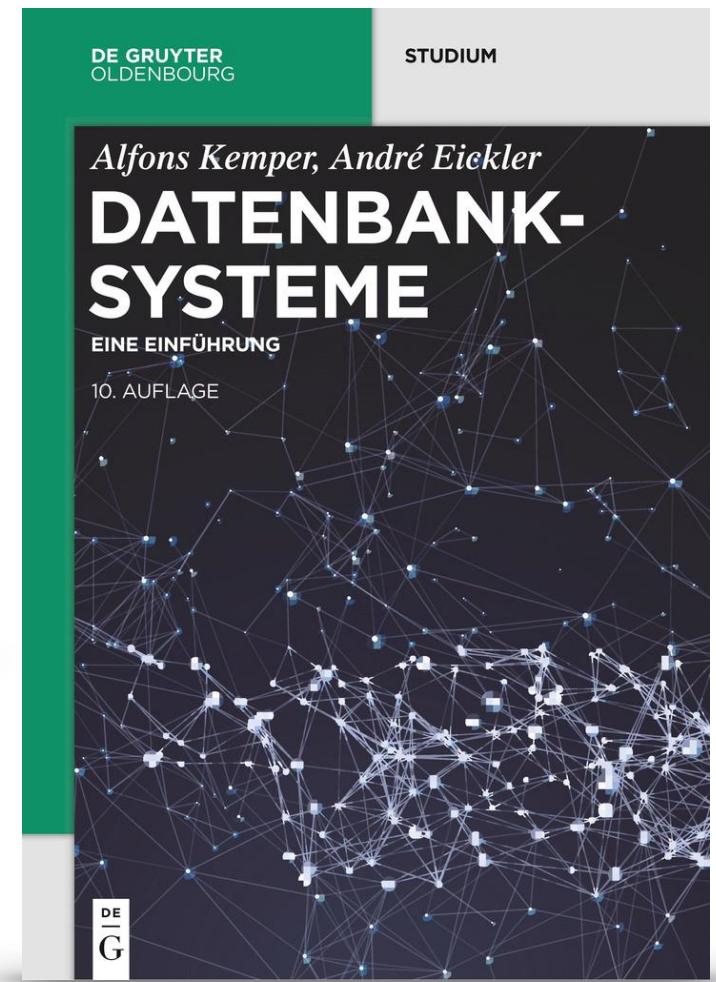
- Kann in Modulprüfung Teil A ersetzen - normalerweise.

Aber ... aufgrund der Corona-Pandemie wollen wir dieses Jahr den Leistungsnachweis nicht anbieten, um die Präsenzphasen insgesamt so gering wie möglich zu halten. Wir bitten hier im Sinne der Gesundheit aller um Verständnis.

## Materialien

### Literatur

- Datenbanksysteme. Eine Einführung.  
Kemper, Eickler (Standardwerk)
- Datenbanken: Grundlagen und Design. Geisler
- Grundlagen von Datenbanksystemen.  
Elmasri, Navathe



A screenshot of a course management system interface. It shows three main sections: 'Übungen' (Exercises) containing 12 entries from DB\_Uebung01 to DB\_Uebung12; 'Material' (Materials) containing 12 entries from DB\_Einführung\_20171009 to DB\_Uebung05\_20171114; and 'Daten' (Data) containing 12 entries from DB\_Kemper\_Data\_20171023 to DB\_Uebung05\_20171114. Each entry includes a thumbnail, file name, size, and date.

### Ilias, FH Aachen

- Präsentationen, Übungen, Daten
- SQL Praktikum
- Weitere Links, z.B. zu Videos

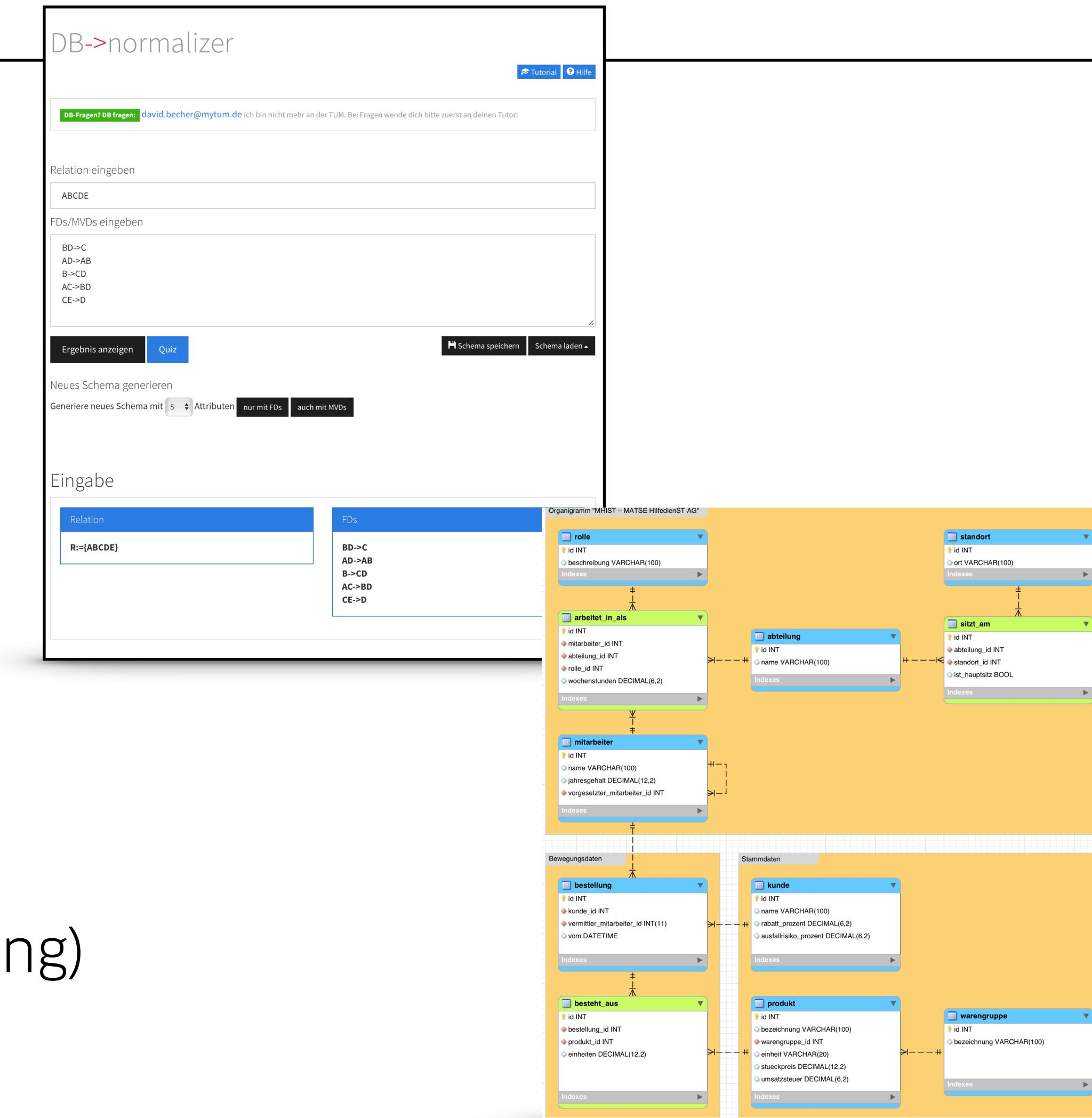
## Materialien

### Tools

- Online-Tools, z.B. Normalizer
- DataGrip, MySQL Workbench

## Datenbanksysteme (DBS)

- MariaDB
- MySQL Community
- Connectors (programmatische Anbindung)



# UNIT 0x01

## MOTIVATION, FALLBEISPIEL

## Datenbanken im Alltag

---

### Q&A

- Was verstehen Sie unter Datenbanken?
- Wo werden Datenbanken eingesetzt?
- Was charakterisiert diese Daten?

## Beispiel Google

---

### Larry Page (Mitgründer Google) Über die perfekte Suchmaschine

"Sie versteht genau das,  
was man meint, und  
liefert genau das, was  
man sucht."



#### Q&A

- Art der Daten?
- Use Cases?
- Zielgruppe?

## Beispiel Google

### Art und Eigenschaften der Daten

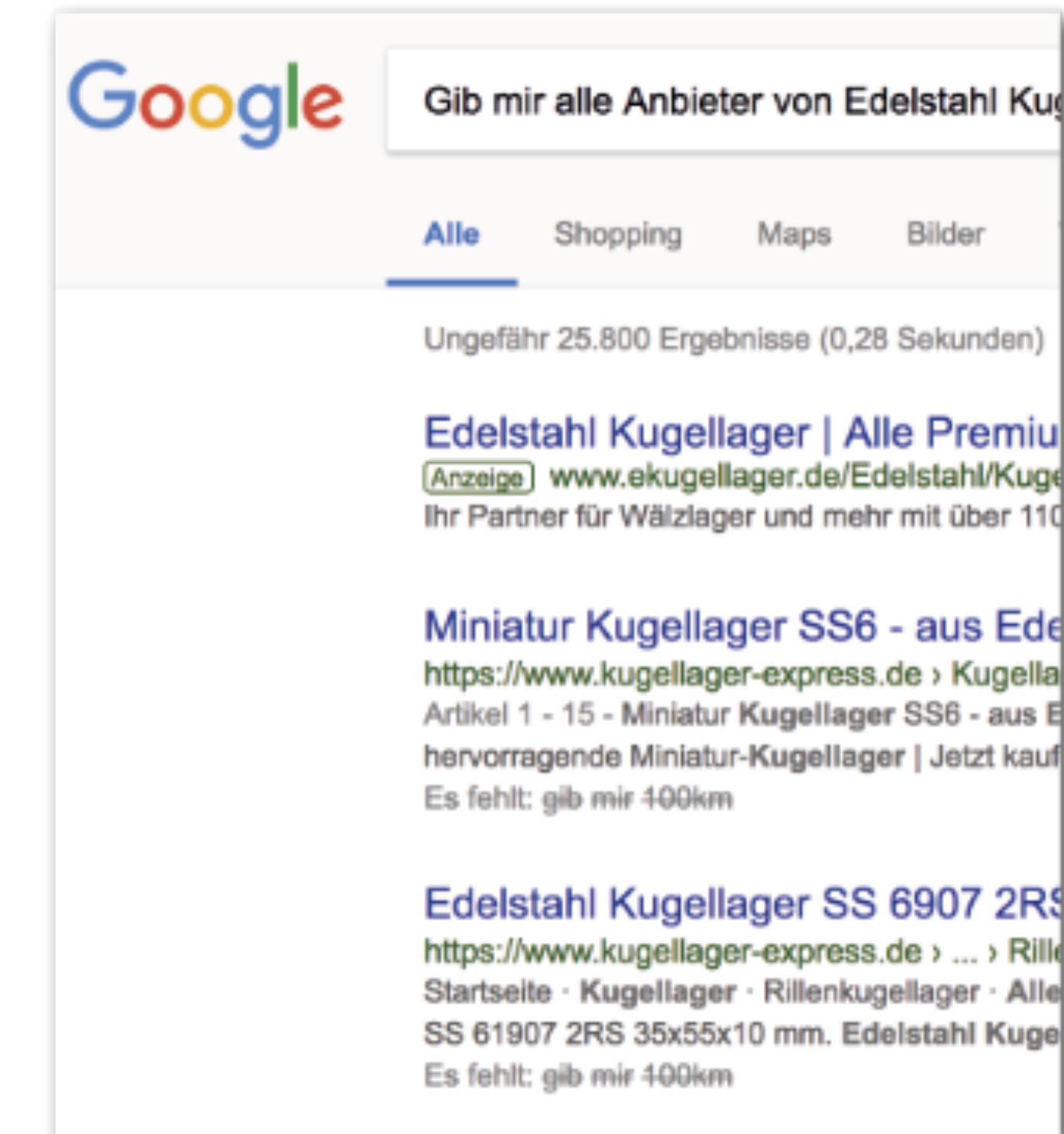
- Suchergebnisse, Websiteanalysen, Marketingdaten
- Dynamisch aufgebaut und flüchtig

### Use Case

- "Suche Anbieter von Edelstahlkugellagern"

### Zielgruppe

- Business to Consumer (B2C) / Business (B2B)



## Beispiel Amazon

---

### Jeff Bezos (Gründer) zu seiner Philosophie

"If you don't understand the details of your business you are going to fail."



#### Q&A

- Art der Daten?
- Use Cases?
- Zielgruppe?

## Beispiel Amazon

---

### Art und Eigenschaften der Daten

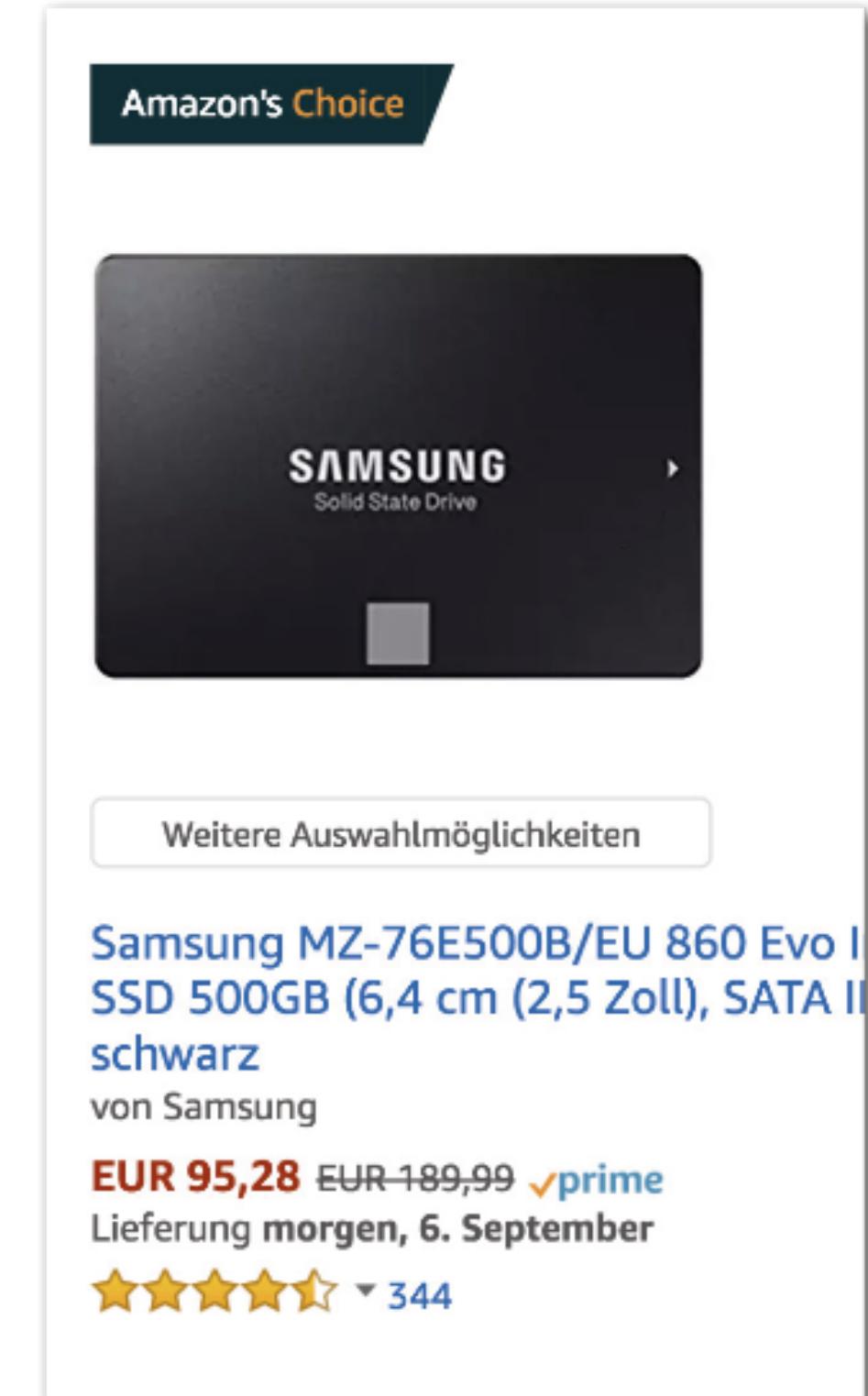
- Daten sind z.B. Produkte und Einkäufe, aber auch Medien
- Dynamisch und statisch

### Use Case

- "Suche Anbieter von Edelstahlkugellagern"

### Zielgruppe

- B2C und B2B



## Beispiel Sparkasse

---

**Wir kennen alle den Slogan  
und die Melodie**

"Wenn's um Geld geht - Sparkasse."



### Q&A

- Art der Daten?
- Use Cases?
- Zielgruppe?

## Beispiel Sparkasse

---

### Art und Eigenschaften der Daten

- Kunden- und Kontodaten
- Sensibel und konsistent

### Use Case

- "Überweise 100€ von Konto 1 auf Konto 2"

### Zielgruppe

- B2C und B2B



## Beispiel SAP

---

### Hasso Plattner (Mitgründer) zum Cloud-Geschäft

"Es gibt durchaus auch interessante Technologie innerhalb der SAP [...] dazu gehört vermutlich auch die Datenbank [...]"



#### Q&A

- Art der Daten?
- Use Cases?
- Zielgruppe?

## Beispiel SAP

### Art und Eigenschaften der Daten

- Unternehmens- und Prozessdaten
- Daten mit "Beziehungen" untereinander

### Use Case

- "Gib mir alle heutigen Rechnungen"

### Zielgruppe

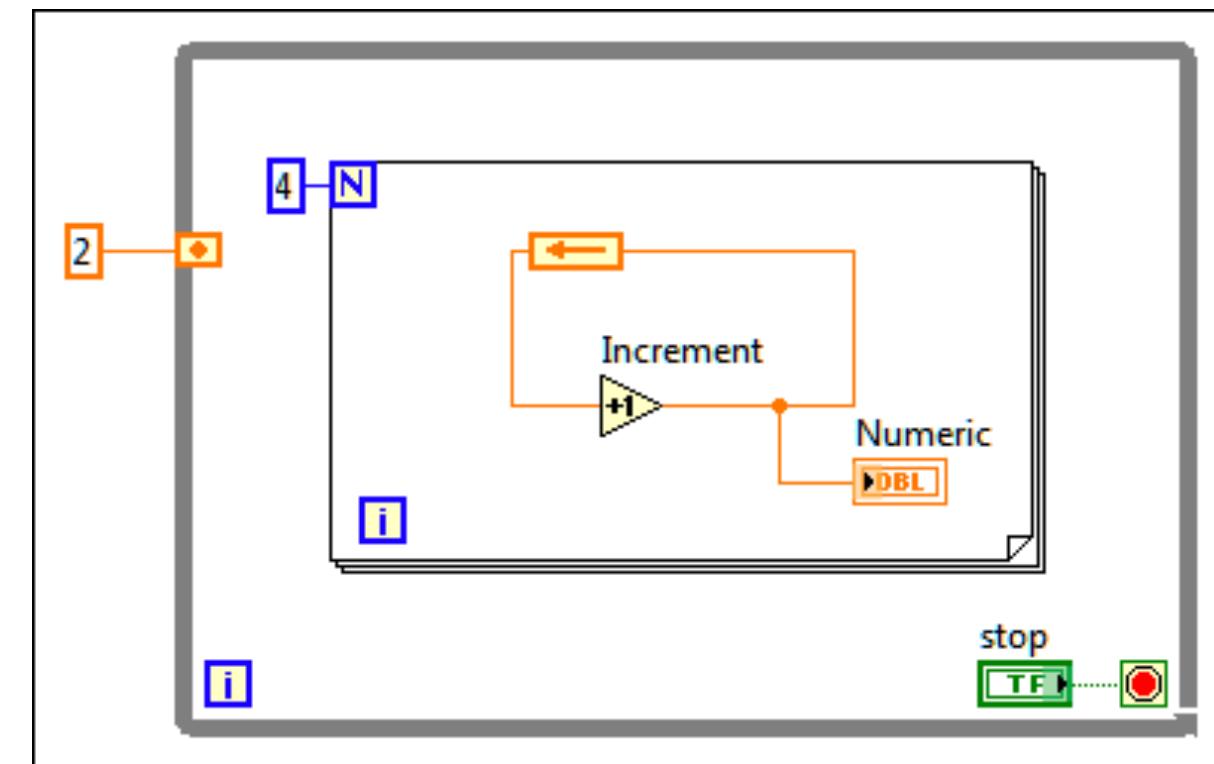
- B2B

The screenshot shows a travel booking application interface. At the top, there are tabs for 'Travel', 'Trip Library', 'Templates', and 'Tools'. Below the tabs, a 'Trip Summary' section displays a trip from 'NEW YORK, NY TO LON' for 'TUE, AUG 15 - FRI, AUG 18'. It includes a 'Select Flights or Trains' button, a 'Round Trip' indicator, and departure/return dates. A 'Finalize Trip' button is also present. To the right of the summary, a 'Change Search' panel is open, showing search fields for 'From' (NYC - New York Area Airports - New York, NY) and 'To' (LON - London Area Airports - London). It also shows 'Depart' and 'Return' dates/times. To the right of the search panel, a 'Shop by Fares' section lists flight options: 'All' (172 results), 'Nonstop' (62 results), and '1 stop' (110 results). The 'All' option is selected. Below these, a 'Flight Number Search' section is partially visible. The overall interface is clean and modern, typical of business travel management software.

## Beispiel National Instruments

**Jeff Kodosky  
(Father of LabVIEW) zur Historie:**

"It would have the simplicity  
of dataflow and the composability  
of structured programming [...]  
and the rest is history."



Loops in  
LabVIEW (NI)

### Q&A

- Art der Daten?
- Use Cases?
- Zielgruppe?

## Beispiel National Instruments

### Art und Eigenschaften der Daten

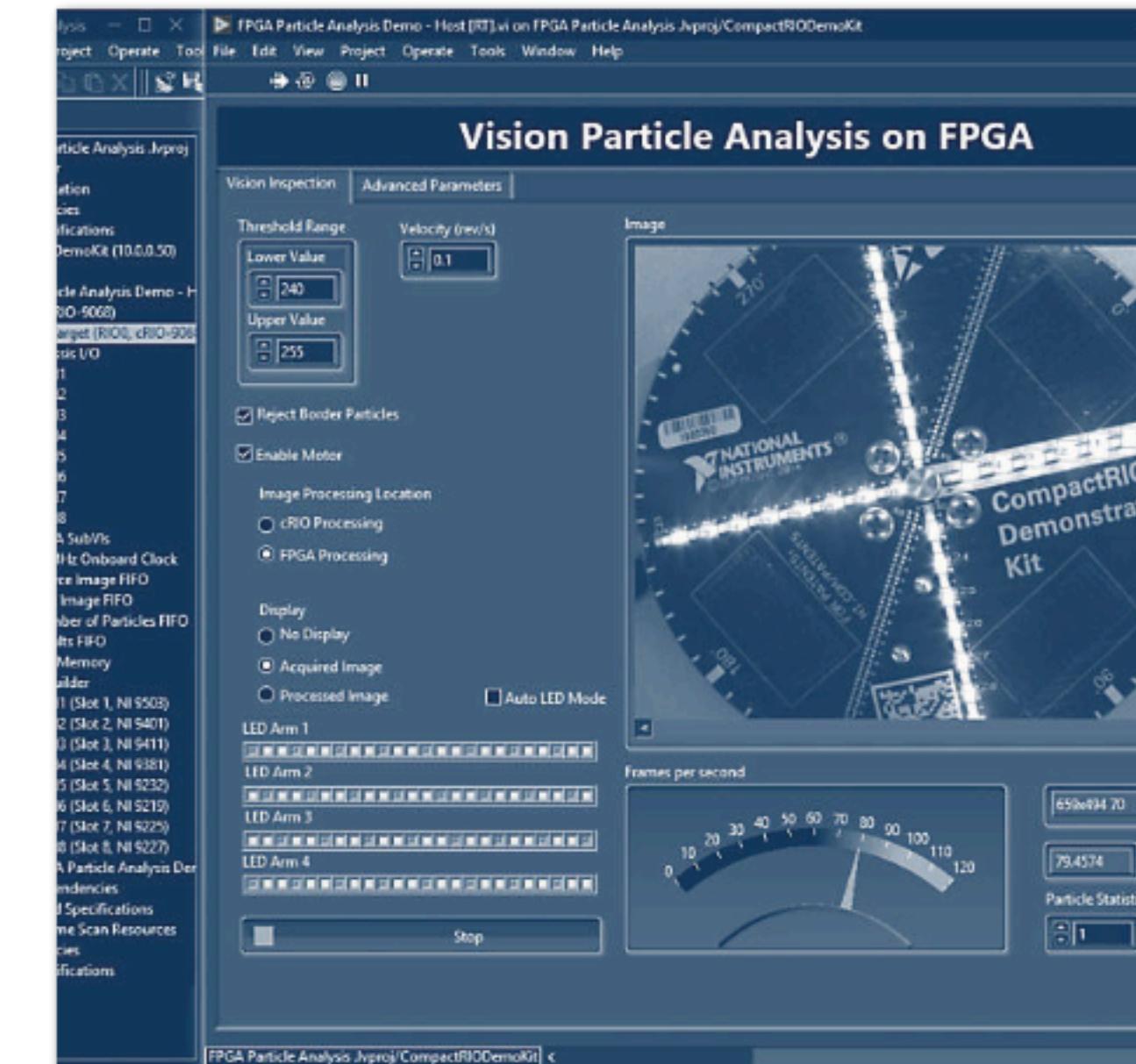
- Mess- und Prozessdaten
- große Datenmengen in kurzer Zeit

### Use Case

- "Löse bei 120°C Motortemperatur ein Signal aus"

### Zielgruppe

- B2B



## Abgeleitete Anforderungen

---

### Q&A

**Welche Anforderungen können Sie hinsichtlich**

- Menge
- Geschwindigkeit
- Sensibilität
- Konsistenz bzw. Integrität

**aus den Beispielen ableiten?**

## Abgeleitete Anforderungen

---

### Anforderungen an "Datenbanken"

- Widerspruchsfreie, dauerhafte, effiziente und schnelle Speicherung von Daten jeder Art
- Bedarfsgerechte und optimierte Bereitstellung von Daten
- Datensicherheit und Datenschutz
- Mehrbenutzerbetrieb



Konkrete Anforderungen und Daten sind offenbar sehr Anwendungsspezifisch.

## Abgeleitete Anforderungen

---

### Q&A

Was bedeutet "gut" oder "geeignet"  
hinsichtlich

- einer Modellierung
- einer Implementierung

ganz genau?



Benötigt werden Werkzeuge und  
Qualitätskriterien für Datenbanken.

## Beispiel einer "Datenbank" – Erster Entwurf

### Entwurf als Excel-Tabelle

- Eine Liste mit Verweisen auf interessante Artikel in Fachzeitschriften, um einzelne Themen schnell wiederzufinden.

Nr (EAN,ISBN)	Zeitschrift	Verlags- gründung	Themen
...aa11	Heise - c't - 11/18	1949	S.15 C++, S.24 C#
...bb22	Heise - c't - 12/18	1949	S.12 Python, S.29 C#
...cc33	Heise - ix - 08/18	1949	S.9 RAID, S.23 gpio
...dd44	Computec - buffed - 08/18	1989	S.11 WoW
...ee55	Webedia - GameStar - 08/18	2007	S.13 LoL, S.20 WoW

### Q&A

Ist der Ansatz gut?

- Pro/Conta und Warum?

## Erster Entwurf - Analyse

Nr (EAN,ISBN)	Zeitschrift	Verlags- gründung	Themen
...aa11	Heise - c't - 11/18	1949	S.15 C++, S.24 C#
...bb22	Heise - c't - 12/18	1949	S.12 Python, S.29 C#
...cc33	Heise - ix - 08/18	1949	S.9 RAID, S.23 gpio
...dd44	Computec - buffed - 08/18	1989	S.11 WoW
...ee55	Webedia - GameStar - 08/18	2007	S.13 LoL, S.20 WoW

### Pro

- Kompakte Darstellung
- Auf den ersten Blick ausreichend

### Contra

- Redundante Daten
- Spalten enthalten mehrere Informationen

## Beispiel einer "Datenbank" – Zweiter Entwurf

### Modifikation

- Einzelne Informationen (Zeitschrift und Themen) sind separiert, d.h. in eigenen Spalten aufgeführt.

Nr (N)	Verlag (V)	Magazin (M)	Ausgabe (A)	Gründung (G)	Seite (S)	Thema (T)
aa11	Heise	c't	11/18	1949	15	C++
aa11	Heise	c't	11/18	1949	24	C#
bb22	Heise	c't	12/18	1949	12	Python
bb22	Heise	c't	12/18	1949	29	C#
cc33	Heise	iX	08/18	1949	9	RAID
cc33	Heise	iX	08/18	1949	23	gpio
dd44	Computec	buffed	08/18	1989	11	WoW
ee55	Webedia	GameStar	08/18	2007	13	LoL
ee55	Webedia	GameStar	08/18	2007	20	WoW

### Q&A

Ist der Ansatz besser?

- Pro/Conta und Warum?

## Zweiter Entwurf – Analyse

---

Nr (N)	Verlag (V)	Magazin (M)	Ausgabe (A)	Gründung (G)	Seite (S)	Thema (T)
aa11	Heise	c't	11/18	1949	15	C++
aa11	Heise	c't	11/18	1949	24	C#
bb22	Heise	c't	12/18	1949	12	Python
bb22	Heise	c't	12/18	1949	29	C#
cc33	Heise	iX	08/18	1949	9	RAID
cc33	Heise	iX	08/18	1949	23	gpio
dd44	Computec	buffed	08/18	1989	11	WoW
ee55	Webedia	GameStar	08/18	2007	13	LoL
ee55	Webedia	GameStar	08/18	2007	20	WoW

### Q&A

- Wozu eine eindeutige Nr?

### Contra

- Mehr Daten redundant.
- Nr nicht mehr eindeutig.
- Sachverhalte nach wie vor gemischt.

## Beispiel einer "Datenbank" – Dritter Entwurf

### Modifikation

- Sachverhalte sind separiert.
- Eindeutige "Schlüssel", d.h. Nr bzw. Nr+Seite ergeben eindeutigen Datensatz.

#### Q&A

Ist der Ansatz besser?

- Pro/Conta und Warum?

Nr (N)	Verlag (V)	Magazin (M)	Ausgabe (A)	Gründung (G)
aa11	Heise	c't	11/18	1949
bb22	Heise	c't	12/18	1949
cc33	Heise	iX	08/18	1949
dd44	Computec	buffed	08/18	1989
ee55	Webedia	GameStar	08/18	2007

Nr (N)	Seite (S)	Thema (T)
aa11	15	C++
aa11	24	C#
bb22	12	Python
bb22	29	C#
cc33	9	RAID
cc33	23	gpio
dd44	11	WoW
ee55	13	LoL
ee55	20	WoW

## Dritter Entwurf - Analyse

---

Nr (N)	Verlag (V)	Magazin (M)	Ausgabe (A)	Gründung (G)
aa11	Heise	c't	11/18	1949
bb22	Heise	c't	12/18	1949
cc33	Heise	iX	08/18	1949
dd44	Computec	buffed	08/18	1989
ee55	Webedia	GameStar	08/18	2007

Nr (N)	Seite (S)	Thema (T)
aa11	15	C++
aa11	24	C#
bb22	12	Python
bb22	29	C#
cc33	9	RAID
cc33	23	gpio
dd44	11	WoW
ee55	13	LoL
ee55	20	WoW

### Contra

- Mehrere Tabellen, verteilte Informationen
- Indirekte Abhängigkeiten, d.h. z.B. Gründung gehört nur zum Verlag, nicht speziell zur Ausgabe.

### Q&A

- Wie verbessern?

## Beispiel einer "Datenbank" – Vierter Entwurf

### Modifikation

- Indirekte Abhängigkeiten in eigener Tabelle aufgelöst.

Verlag (V)	Gründung (G)
Heise	1949
Computec	1989
Webedia	2007

Nr (N)	Verlag (V)	Magazin (M)	Ausgabe (A)
aa11	Heise	c't	11/18
bb22	Heise	c't	12/18
cc33	Heise	iX	08/18
dd44	Computec	buffed	08/18
ee55	Webedia	Comstar	08/18

### Q&A

#### Ist der Ansatz besser?

- Pro/Conta und Warum?

Nr (N)	Seite (S)	Thema (T)
aa11	15	C++
aa11	24	C#
bb22	12	Python
bb22	29	C#
cc33	9	RAID
cc33	23	gpio
dd44	11	WoW
ee55	13	LoL
ee55	20	WoW

## Vierter Entwurf - Analyse

Verlag (V)	Gründung (G)
Heise	1949
Computec	1989
...bb22	2007

Nr (N)	Verlag (V)	Maga
aa11	Heise	c't
bb22	Heise	c't
cc33	Heise	iX
dd44	Computec	buffer
ee55	Webedia	Gamedev

Nr (N)	Seite (S)	Thema
aa11	15	C++
aa11	24	C#
bb22	12	Python

### Contra

- Informationen, z.B. alle Artikel eines Magazins, wieder zusammenzustellen

...bb22 Heise - c't - 12/18 1949 S.12 Python, S.29 C#

ist aufwändig.

### Q&A

- Wie sähe eine Modellierung praktisch aus?

## Modellierung Mini-Welt - Beschreibung

---

### **Mini-Welt / Ausschnitt**

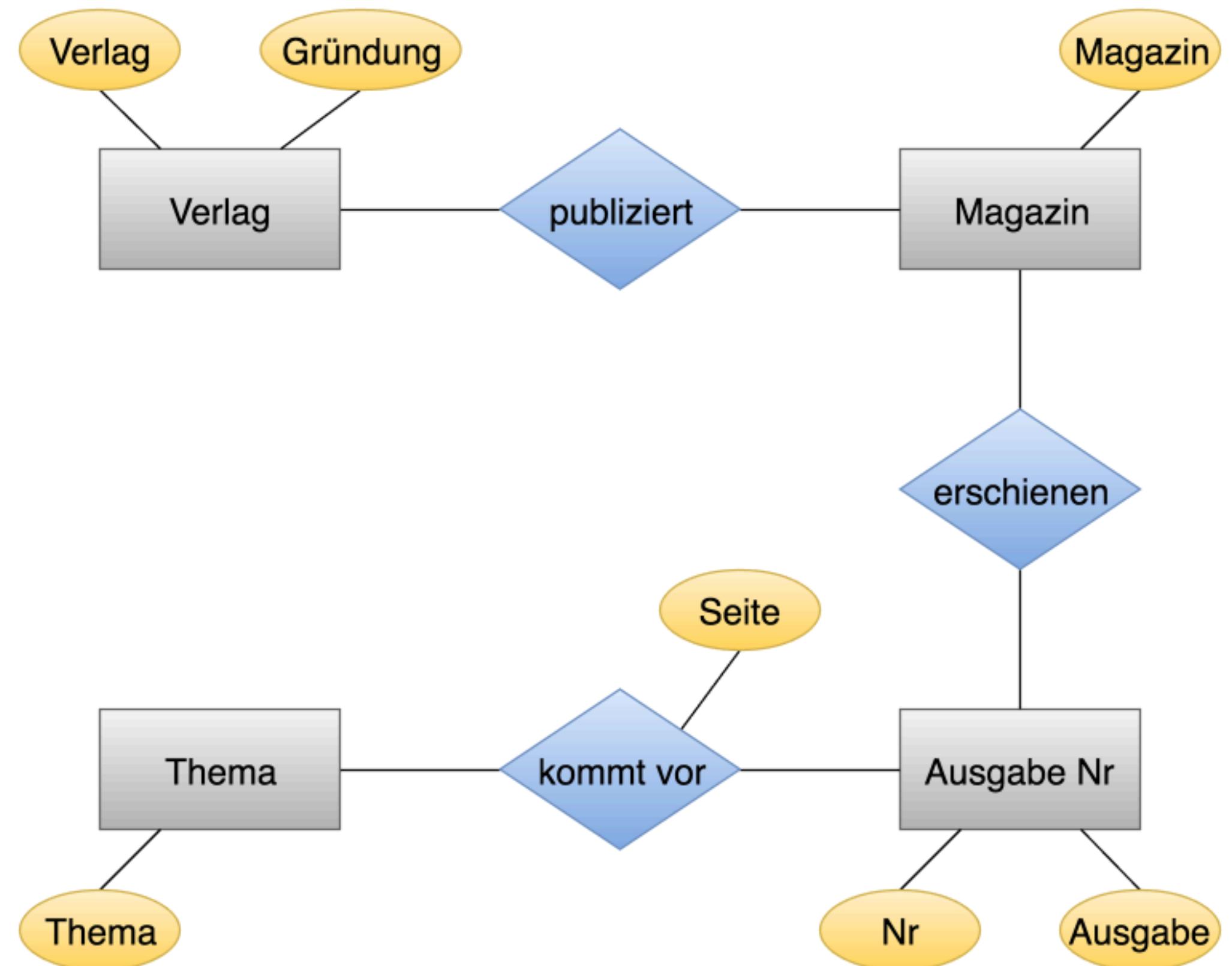
- Realisiert wird eine Liste mit Verweisen auf interessante Artikel zu Themen in bestimmten Ausgaben verschiedener Magazine eines Verlages. [...].
- Verlage haben einen Namen und ein Gründungsjahr, Magazine einen Namen. [...].
- Weitere Eigenschaften bzw. Anforderungen.

### **Modellierung**

- Entitätstypen
- Attribute/Eigenschaften
- Schlüssel
- Beziehungen
- Kardinalitäten

Genaue Definitionen folgen.

## Modellierung Mini-Welt - ER-Diagramm



### Modellierung

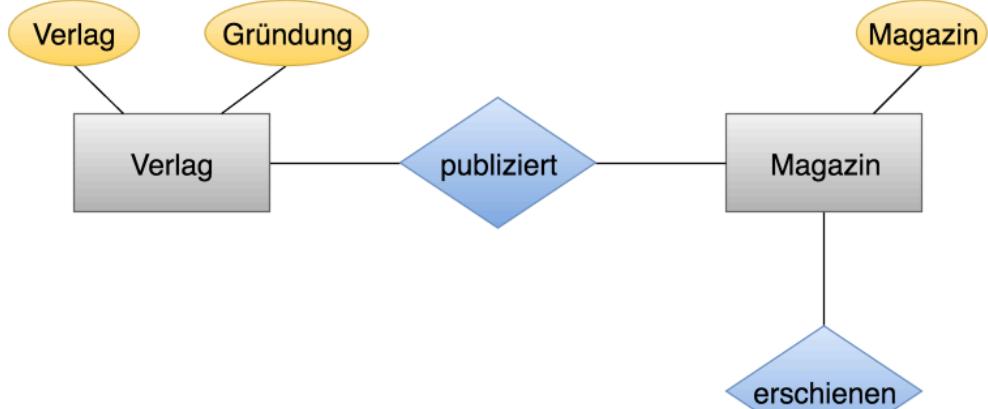
- Entitätstypen
- Attribute
- Schlüssel
- Beziehungen
- Kardinalitäten



Genormte Symbolik  
bedeutet gemeinsame  
Sprache

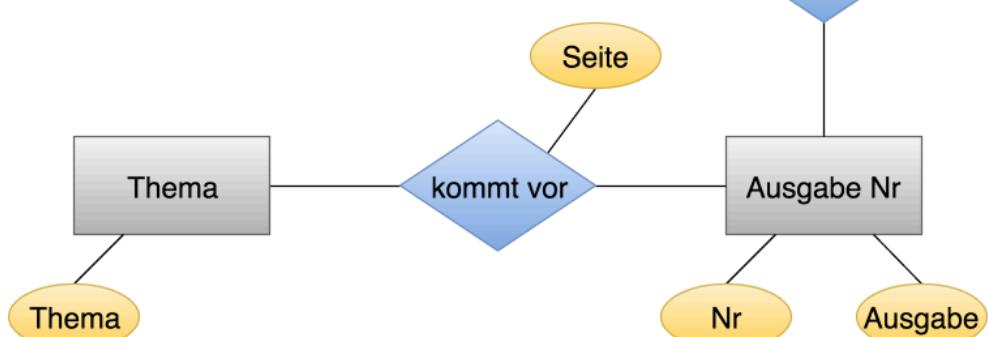
# Modellierung Mini-Welt - Daten und Tabellen

V-ID	Verlag (V)	Gründung (G)
101	Heise	1949
102	Computec	1989
103	Webedia	2007



M-ID	V-ID	Magazin (M)
301	101	c't
302	101	iX
303	102	buffed
304	103	GameStar

T-ID	Thema (T)
201	C++
202	C#
203	Python
204	RAID
205	gpio
206	WoW
207	LoL



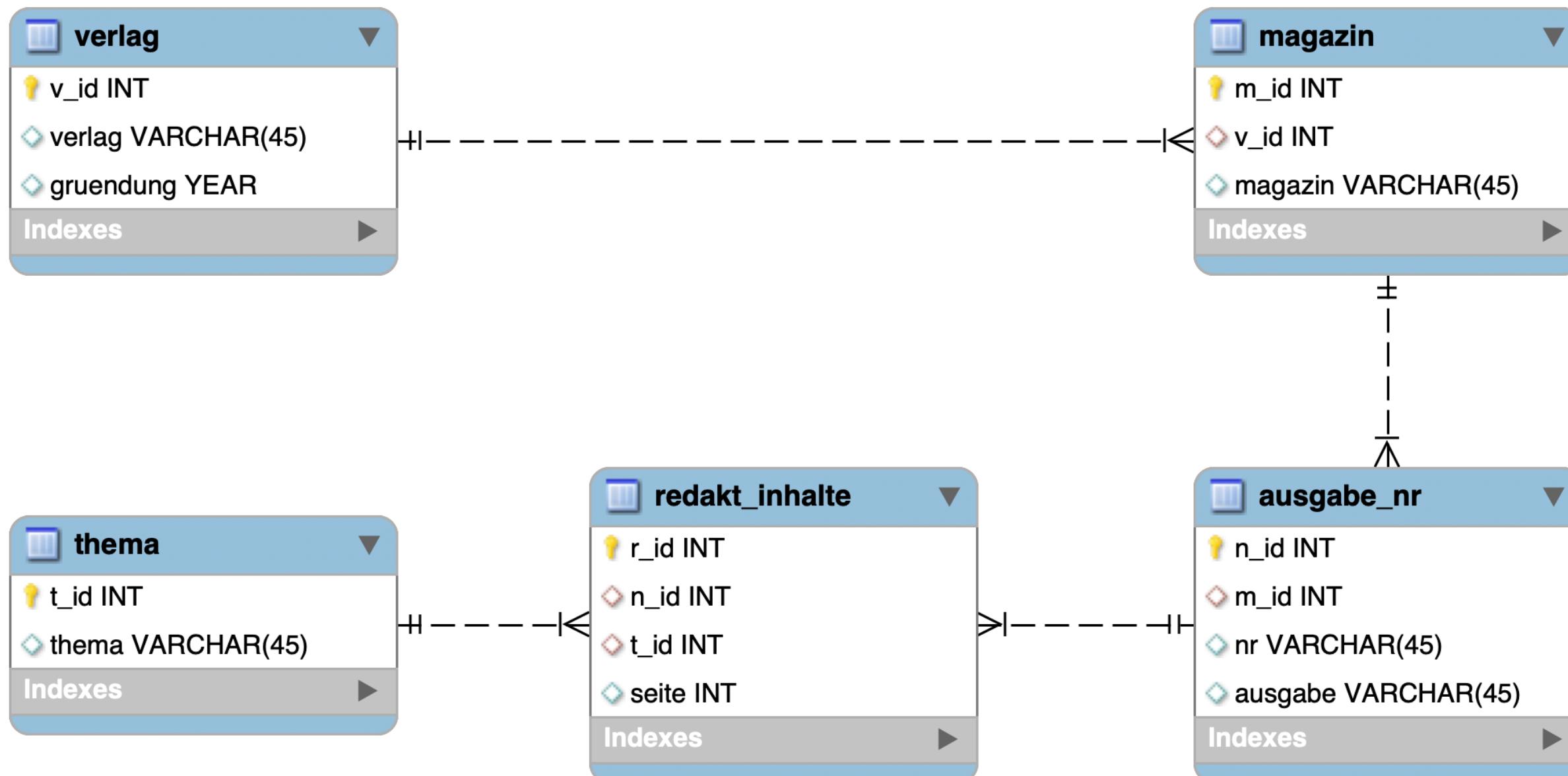
N-ID	M-ID	Nr (N)	Ausgabe (A)
401	301	aa11	11/18
402	301	bb22	12/18
403	302	cc33	08/18
404	303	dd44	08/18
405	304	ee55	08/18

N-ID	T-ID	Seite (S)
401	201	15
401	202	24
402	203	12
402	202	29
403	204	9
403	205	23
404	206	11
405	207	13
405	206	20

## Modellierung

- Schlüssel (IDs) ohne inhaltliche Bedeutung.
- Beziehungen unterschiedlich realisiert.

## Modellierung Mini-Welt - Datenbankmodell



### Modellierung

- Entitätstypen
- Attribute
- Schlüssel
- Beziehungen
- Kardinalitäten



ER-Modell und Datenbankmodell sind nicht identisch - aber stark verwandt.

## Modellierung Mini-Welt – Structured Query Language (SQL)

```

1
2 • SELECT v_id,verlag,gruendung FROM verlag;

```

100% 35:2

**Result Grid** Filter Rows: Search Edit:

v_id	verlag	gruendung
101	Heise	1949
102	Computec	1989
103	Wedia	2007

```

1
2 • SELECT t_id,thema FROM thema;

```

100% 24:2

**Result Grid** Filter Rows: Search Edit:

t_id	thema
201	C++
202	C#
203	Python
204	RAID

```

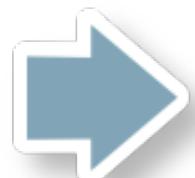
9 • select a.nr,v.verlag,m.magazin,a.ausgabe,v.gruendung,t.thema
10 from ausgabe_nr a join magazin m using (m_id) join verlag v
11 join redakt_inhalte r using (n_id) join thema t using (t_id)
12 where t.thema="C#";
13

```

100% 6:9

**Result Grid** Filter Rows: Search Export:

nr	verlag	magazin	ausgabe	gruendung	thema	seite
aa11	Heise	c't	11/18	1949	C#	24
bb22	Heise	c't	12/18	1949	C#	29



...bb22 Heise - c't - 12/18 1949 S.12 Python, S.29 C#



Praktische Aufgaben  
sind Teil der Übung  
und Teil der Prüfung.

## Tour d'Horizont

### Angesprochene Themen

- Definitionen und Werkzeuge
- Anforderungen und Analyse der Problemstellung
- Modellierung nach Qualitätskriterien
- Implementierung der Datenbank
- Datenbankmanagement und Datenabfrage
- Best Practices



- Datenbanken können wir gut und schlecht modellieren, 'Gut' bedeutet: 'Erfüllt die Anforderungen und Kriterien'.
- Wir benötigen eine gemeinsame Sprache und Werkzeuge, sowohl mathematische als auch in Form von Software.
- Best Practice führt mit Hintergrundwissen schnell zu gutem Design.
- Andere Typen von Datenbanken sind noch zu behandeln.

Und es gibt natürlich noch mehr Themen...

# UNIT 0x02

# GRUNDLAGEN DATENBANKSYSTEME

## Datenbanksysteme

---

### Q&A

**Was ist eine Datenbank, ein Datenbanksystem oder ein Datenbankmanagementsystem genau?**

- Definition?
- Typen?
- Content?
- Anforderungen?
- Aufbau/Komponenten?

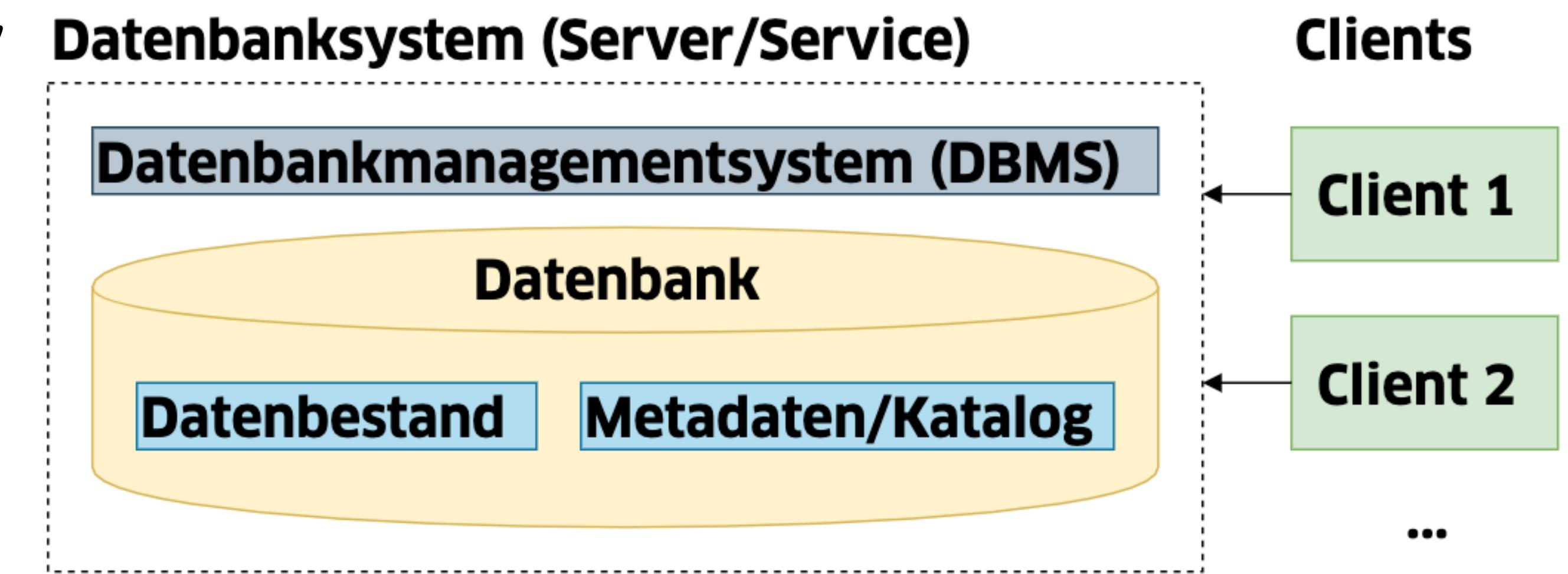
## Datenbanksysteme – Definitionen

### Begriffsbestimmung DBS, DBMS, Datenbank

- Ein Datenbanksystem (DBS) besteht aus dem Datenbankmanagementsystem (DBMS), d.h. der Software, und den Daten, also der Datenbank.
- Die Clients verbinden sich mit dem DBMS und kommunizieren über eine spezielle Datenbanksprache, z.B. SQL, um Daten der Datenbank abzufragen, zu speichern oder zu administrieren.



- Im Sprachgebrauch wird zwischen DBS, DBMS, 'Datenbankverwaltungssystem' oder auch nur 'Datenbank' oft nicht unterschieden.
- 'Datenbank' meint in unserem Kontext die Menge der Daten.



## Datenbanksysteme – Definitionen

---

### Charakterisierung DBMS

- Ein Datenbankmanagementsystem (DBMS) ist eine Software zur *sicheren, konsistenten und persistenten Speicherung großer Datenmengen*, mit dem Ziel *mehreren Benutzern (gleichzeitig) effizienten, zuverlässigen, sicheren und bequemen Zugriff auf diese Daten zu ermöglichen*. → **Anforderungen an ein DBS bzw. DBMS**
- Es besteht aus Komponenten zur Abfrage, Verwaltung und Administration der Daten bzw. Datenbank. → **Aufbau DBMS**

## Datenbanksysteme – Typen

---

### **DBS bzw. DBMS Typen**

Je nach Art der Daten und des Anwendungsfalls eignen sich unterschiedliche Typen von DBS, genauer DBMS, unterschiedlich gut. Da sind beispielsweise

- **relationale DBMS**, die Objekte gleicher Struktur (Attribute) in Tabellen verwalten.

Oder aber

- **hierarchische oder objektorientierte DBMS**, die Objekte mit Eltern-Kind- oder Vererbungsbeziehungen besitzen und einzelne Strukturen gemeinsam haben.

Und auch

- **dokumentenorientiert DBMS**, die Objekte ohne vergleichbare Strukturen verwalten.



Es gibt noch mehr Typen und in der Praxis sind es häufig Mischformen...

## Datenbanksysteme – Typen

---

### RDBMS

- Am verbreitetsten sind **relationale DBMS (RDBMS)**, weswegen das auch ein großer Schwerpunkt der Vorlesung ist. Damit einher geht SQL als Abfragesprache.
- Der Erfolg und die lange Dominanz der RDBMS begründet sich sicherlich in einer "normierten" Abfragesprache und der vergleichsweise einfachen und effizienten Behandlung von Objekten gleichen Aufbaus in Tabellen. Und in der Praxis sind viele Daten strukturiert und typisiert.
- Hinzu kommt, dass andere Anwendungsfälle, etwa hierarchische Daten, mit entsprechendem Overhead, dennoch in RDBMS abgebildet werden können. Dies und weitere nicht-relationale DBMS behandeln wir am Ende der Vorlesung.



Fragestellungen rund um Datenbanken, etwa Definitionen, Anforderungen oder Aufbau sind im Kern unabhängig vom konkreten DBS-Typ.

## Datenbanksysteme - Content

---

### **'Daten' bzw. 'Datum'**

- Daten sind zunächst Folgen von Zeichen (Zahlen, Buchstaben, Symbole) oder auch strukturierte Objekte.
- Sie folgen einer bestimmten Syntax.
- Beispiel: 112

### **'Informationen'**

- Aus Daten entstehen Informationen, wenn die Bedeutung bekannt ist.
- Durch die Hinzunahme von Informationen, ggf. weiteren Daten, kann das ursprüngliche Datum interpretiert bzw. verstanden werden.
- Beispiel: '112' öffnet den Haussafe

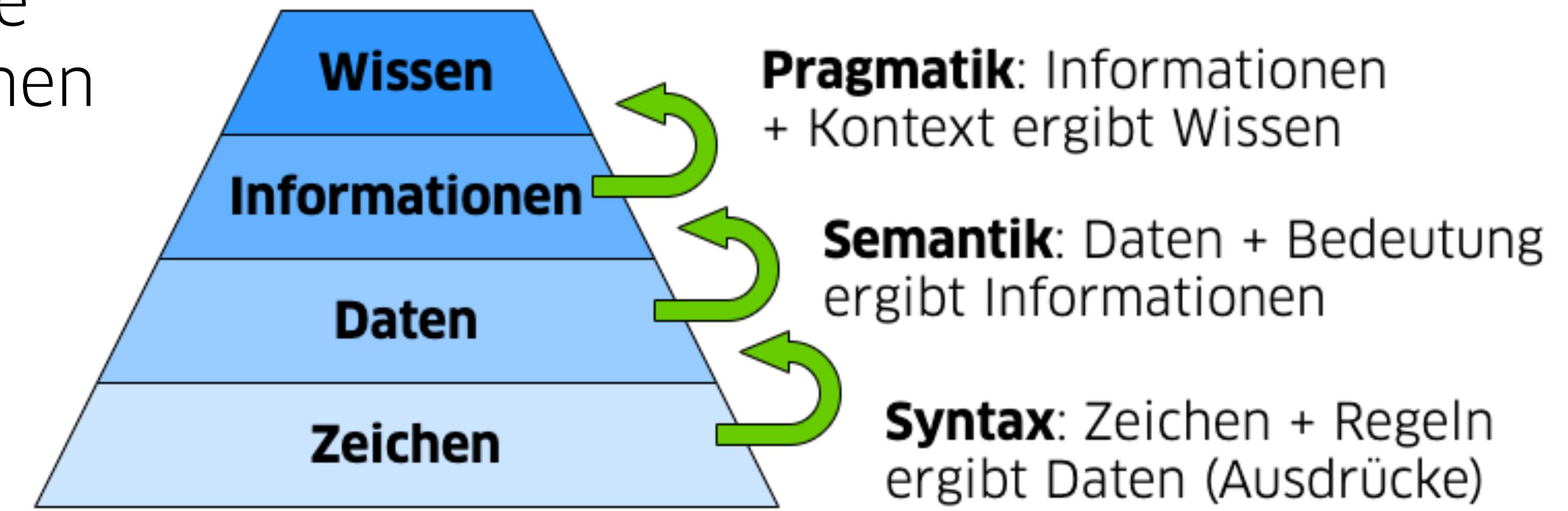
## Datenbanksysteme – Content

### 'Wissen'

- Wissen entsteht durch systematische Verknüpfung von Daten, Informationen und (subjektiver) Erfahrung.
- Beispiel: '112' öffnet den Haussafe, aber wir wissen, dass das kein sicheres Passwort ist, da der Safe bei der Feuerwehr steht...



Achtung: Hier gibt es im Detail durchaus unterschiedliche Definitionen.



### Q&A

#### Was enthalten Datenbanken nun?

- Daten, Informationen oder Wissen?

## Datenbanksysteme – Anforderungen an ein DBMS

---

### Redundanz und Konsistenz

- Beispiel redundanter Daten: n-mal 'Heise' in der Datenbank.
- Beispiel inkonsistenter Daten: Thema 'C#' und (zweites?) Thema 'CSharp'.
- Redundante Daten ermöglichen inkonsistente Daten und belegen Speicher.

### Integrität und Transaktionen

- Beispiel korrumpter Daten: 'Bild' im Heise-Verlag, Gründung '2089'.
- Transaktionen sichern mehrere Datenbewegungen als eine Aktion ab, die ganz oder gar nicht ausgeführt wird, Beispiel: 100€ von einem Konto auf ein anderes buchen.
- Fehlende Transaktionen ermöglichen korrupte Daten.

## Datenbanksysteme – Anforderungen an ein DBMS

---

### **Relationen/Verknüpfungsmöglichkeiten**

- Beispiel von Daten mit Beziehung: Magazin *erscheint im Verlag*.
- In jeder ernsthaften Datenmenge gibt es abzubildende Beziehungen.

### **Zugriffsberechtigungen**

- Beispiel fehlender Rechte: Student trägt Note selber ein (wie praktisch...).

### **Mehrbenutzerbetrieb**

- Beispiel Single-User-Betrieb: Bitte noch warten bis zum Aufgeben Ihrer Bestellung, ein Benutzer aus Brasilien ist eingeloggt.
- Gerade mit Blick auf Transaktionen gibt es hier ein großes Fehlerpotenzial.

## Datenbanksysteme – Anforderungen an ein DBMS

### Persistenz und Datensicherung

- Beispiel nicht-persistenter Datensicherung (aus MySQL Dokumentation):  
*"The BLACKHOLE storage engine acts as a “black hole” that accepts data but throws it away and does not store it. Retrievals always return an empty result."*
- Beispiel fragwürdiger Datensicherung:  
Bei Microsofts Tochterfirma 'Danger' ist der Name offenbar Programm [...] (Spiegel 10/2009)

"Bedauerlicherweise müssen wir Sie nun darüber informieren, dass, basierend auf der letzten Einschätzung zur Datenwiederherstellung von Microsoft/Danger, persönliche Daten, die Sie auf Ihrem Sidekick gespeichert haben, mit höchster Wahrscheinlichkeit auf Grund eines Server-Fehlers bei Microsoft/Danger verlorengegangen sind."



## Datenbanksysteme – Anforderungen an ein DBMS

---

### Kennzahlen DBMS, Beispiele

- Status
- User
- Performanceanalyse
- Abfrageanalyse
- ..

## Datenbanksysteme – Anforderungen an ein DBMS

---

### Aus den Anforderungen abgeleitete Themen der Vorlesung

- Modellierung eines realen Aspekts (Mini-Welt) in einem DBMS führt zu Abstraktionsebenen und in der Folge zu ER-Diagrammen, Entitäten, Relationen und DBMS-Strukturen.
- Redundanzen, bzw. mögliche Anomalien, führen zum Konzept der Normalisierung, also einem Qualitätsmaß für eine Modellierung bzw. für DBMS-Strukturen.
- Zugriffsberechtigungen und Mehrbenutzerbetrieb erfordern Authentisierung, Transaktionen und Abstraktionen wie Sichten.
- Persistenz führt zu unterschiedlichen Storage-Engines und Indizes, also physische Abbildungen der Daten mit Blick auf Speicher und Performance, d.h. der Diskussion um den internen Aufbau eines DBMS.

## Datenbanksysteme – Aufbau

### Komponenten DBMS

- **Data Manipulation Language (DML)**

**Compiler:** Überführt Anfrage in ausführbare Form, primär zur Manipulation von Daten.

- **Data Definition Language (DDL)**

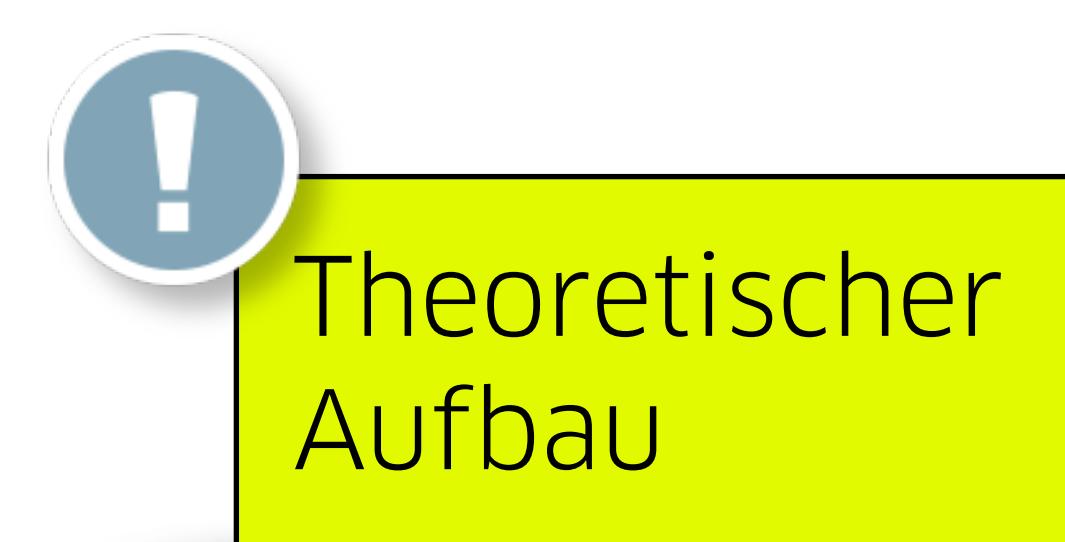
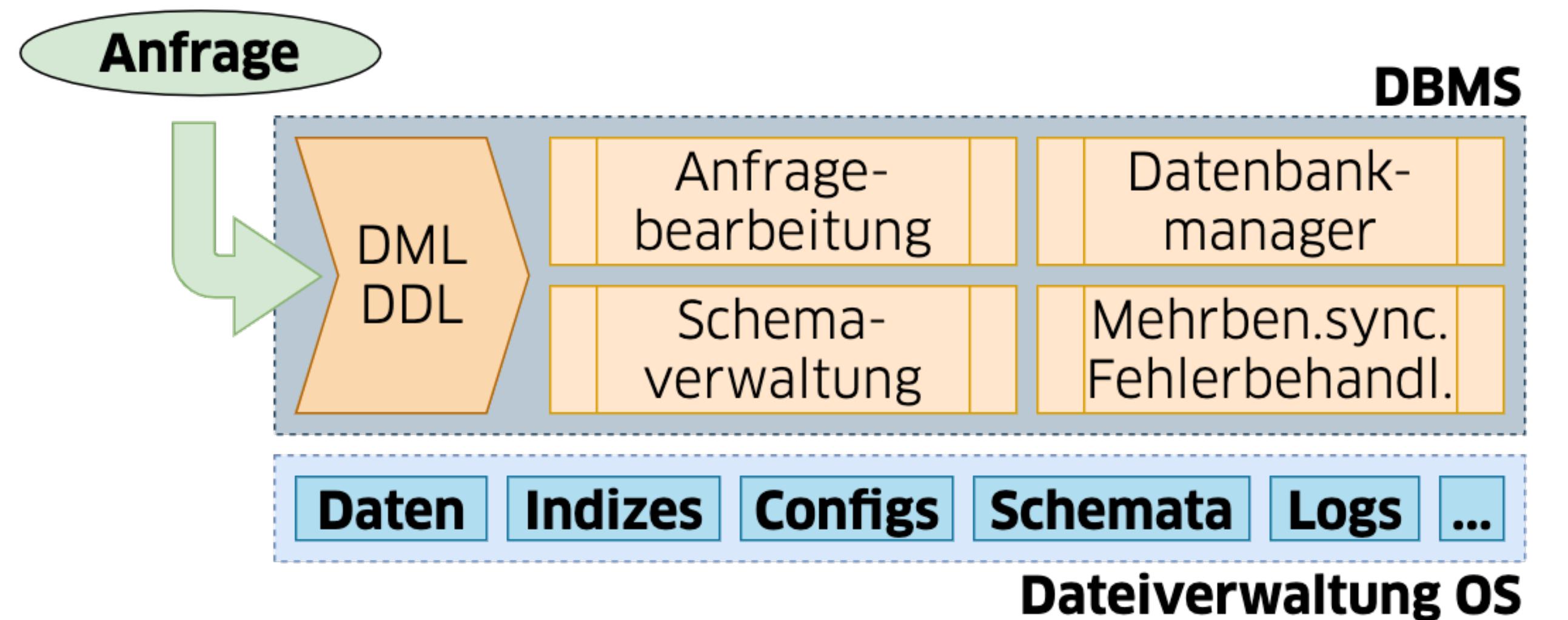
**Compiler:** Wie DML, nur für Datenstrukturen.

- **Anfragebearbeitung:** Erstellen eines Ablaufplans für eine Abfrage inkl. logischer und physischer Optimierung.

- **Datenbankmanager:** Kern des DBMS, Ausführung der Anfragen.

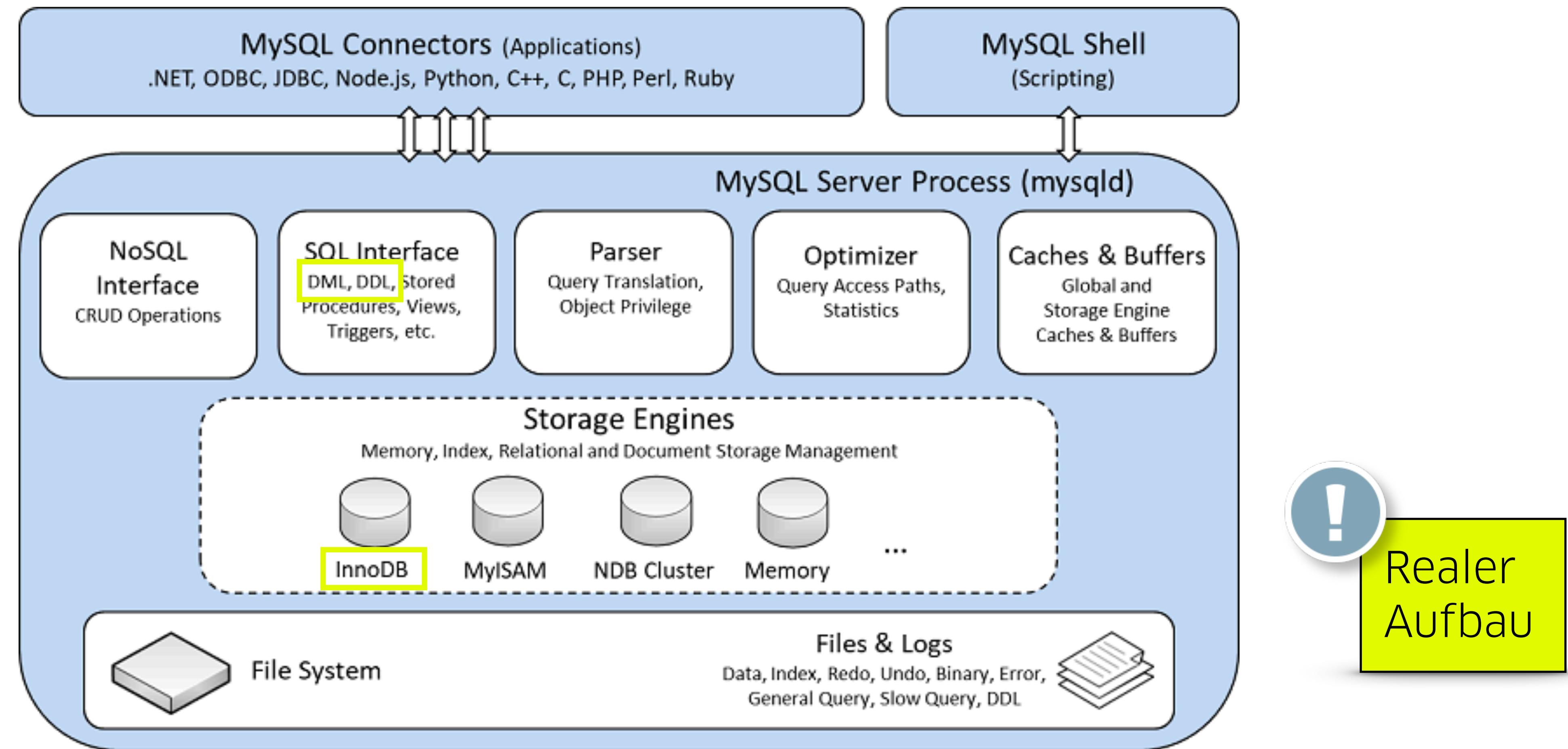
- **Schemaverwaltung:** Verwaltung der Metadaten, 'Typsystem' des DBMS.

- **Mehrbenutzersynchronisation, Fehlerbehandlung:** Transaktionsverwaltung.



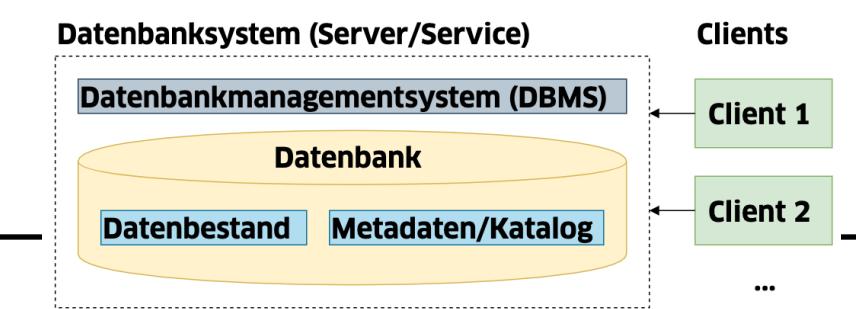
## Datenbanksysteme – Aufbau

### Beispiel MySQL Architektur



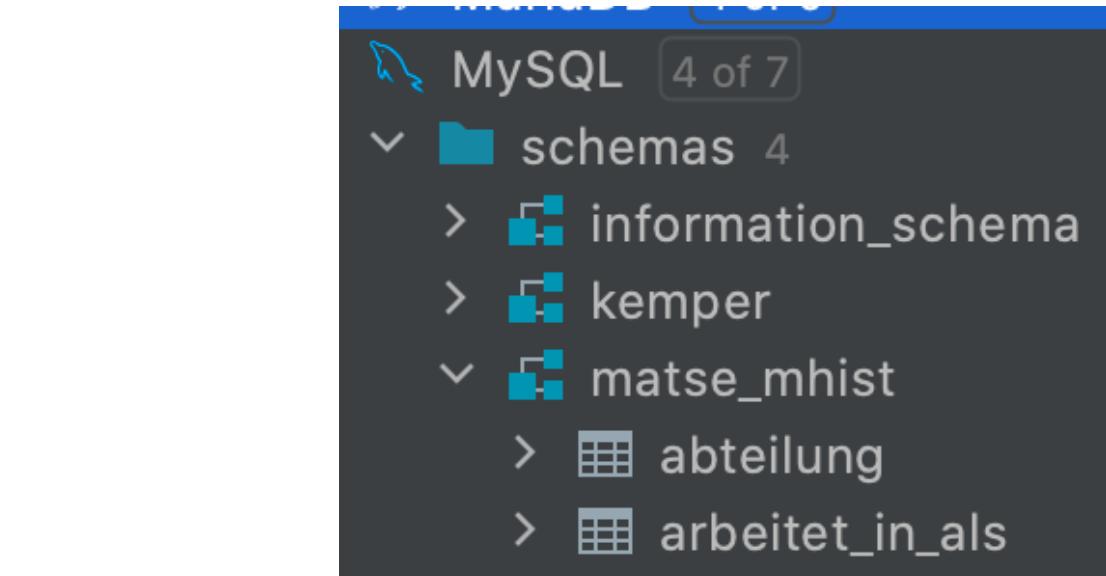
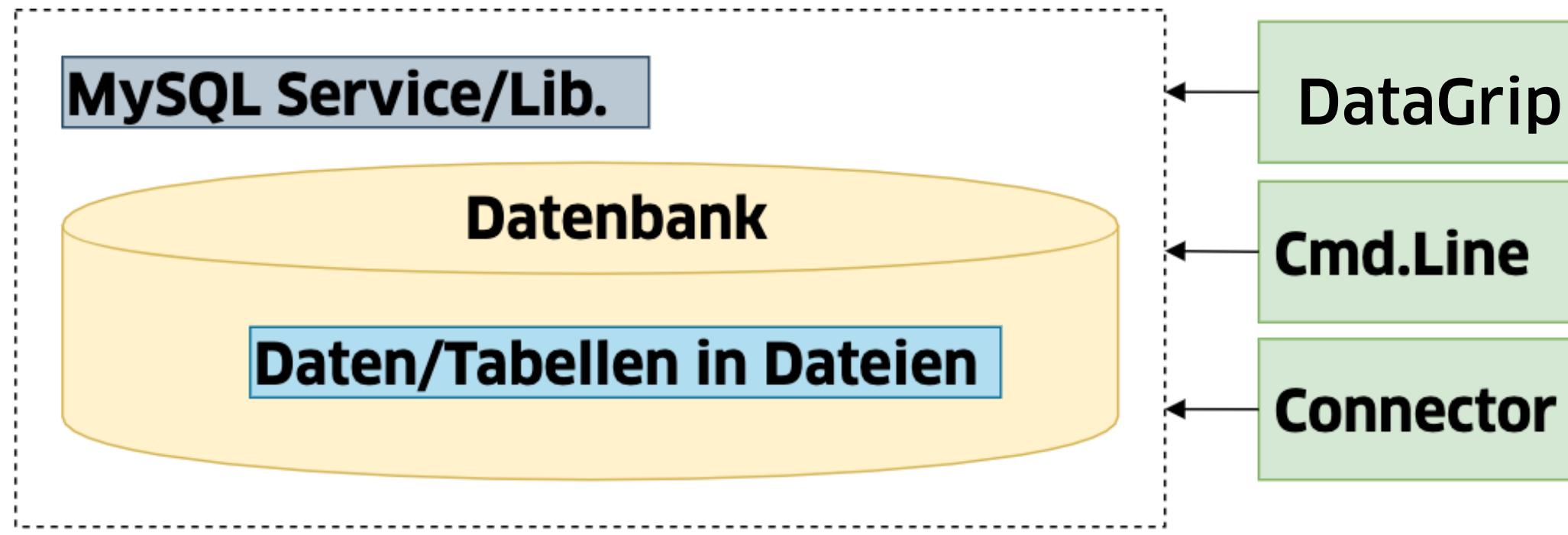
MySQL Dokumentation 'MySQL Architecture with Pluggable Storage Engines'

## Datenbanksysteme – Aufbau



## Beispiel Storage Engine MySQL-InnoDB

### Datenbanksystem MySQL (Server)



Persistenz Tabellen  
DataGrip vs. Dateisystem

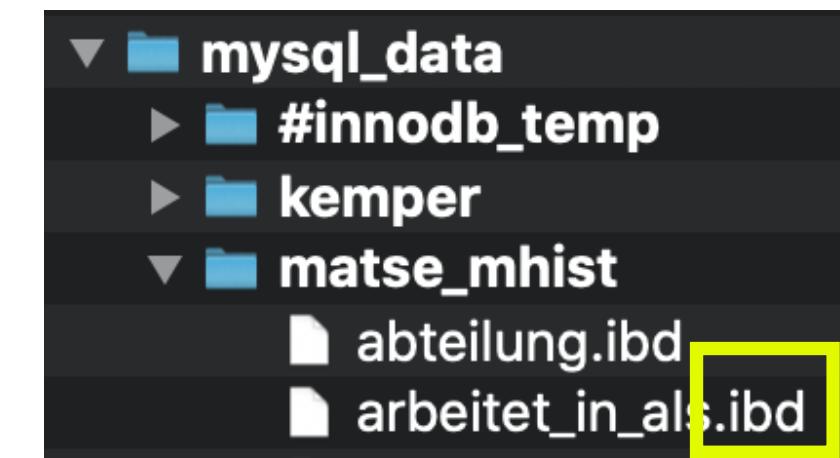
Metadaten

Ein Screenshot der MySQL Workbench Abfrage- und Ergebnisansicht. Die Abfrage ist:

```
SELECT engine, TABLE_COLLATION, CREATE_TIME
FROM information_schema.tables
WHERE table_schema = 'matse_mhist'
AND table_name = 'produkt';
```

Das Ergebnis zeigt eine Tabelle mit einer einzigen Zeile:

ENGINE	TABLE_COLLATION	CREATE_TIME
InnoDB	utf8_general_ci	2020-10-12 21:36:21



- ▼ The InnoDB Storage Engine
  - Introduction to InnoDB
  - InnoDB and the ACID Model
  - InnoDB Multi-Versioning
  - InnoDB Architecture
  - InnoDB In-Memory Structures
  - InnoDB On-Disk Structures
  - InnoDB Locking and Transaction

MySQL Dok. InnoDB  
Formate und \*.ibd

- ▼ Alternative Storage Engines
  - Setting the Storage Engine
  - The MyISAM Storage Engine
  - The MEMORY Storage Engine
  - The CSV Storage Engine
  - The ARCHIVE Storage Engine
  - The BLACKHOLE Storage Engine
  - The MERGE Storage Engine

Alternative  
Engines

# Datenbanksysteme – Aufbau

## Beispiel MySQL-InnoDB Type Storage Requirements

### Numeric Type Storage Requirements

Data Type	Storage Required
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT, INTEGER	4 bytes
BIGINT	8 bytes
FLOAT ( <i>p</i> )	4 bytes if $0 \leq p \leq 24$ , 8 bytes if $25 \leq p \leq 53$
FLOAT	4 bytes
DOUBLE [ PRECISION ], REAL	8 bytes
DECIMAL ( <i>M</i> , <i>D</i> ), NUMERIC ( <i>M</i> , <i>D</i> )	Varies; see following discussion
BIT ( <i>M</i> )	approximately $(M+7)/8$ bytes

### String Type Storage Requirements

In the following table, *M* represents the declared column length in characters for nonbinary string types and bytes for binary string types. *L* represents the actual length in bytes of a given string value.

Data Type	Storage Required
CHAR ( <i>M</i> )	The compact family of InnoDB row formats optimize storage for variable-length character sets. See <a href="#">COMPACT Row Format Storage Characteristics</a> . Otherwise, $M \times w$ bytes, $\leq M \leq 255$ , where <i>w</i> is the number of bytes required for the maximum-length character in the character set.
BINARY ( <i>M</i> )	<i>M</i> bytes, $0 \leq M \leq 255$
VARCHAR ( <i>M</i> ), VARBINARY ( <i>M</i> )	<i>L</i> + 1 bytes if column values require 0 – 255 bytes, <i>L</i> + 2 bytes if values may require more than 255 bytes
TINYBLOB, TINYTEXT	<i>L</i> + 1 bytes, where $L < 2^8$
BLOB, TEXT	<i>L</i> + 2 bytes, where $L < 2^{16}$
MEDIUMBLOB, MEDIUMTEXT	<i>L</i> + 3 bytes, where $L < 2^{24}$

- ▼ Data Types
  - Numeric Data Types
  - Date and Time Data Types
  - String Data Types
  - Spatial Data Types

# UNIT 0x03

## MODELLIERUNG UND

## ENTITY-RELATIONSHIP-MODELL I

## Motivation

---

### Q&A

- Wozu Modellierung?
- Welche Rollen sind beteiligt? Wer sieht was?
- Wie läuft der Prozess der Modellierung ab?

## Motivation

---

### Wozu Modellierung?

- Übersetzt ein reales Problem (Szenario/Mini-Welt, z.B. die lesenswerten Artikel) in eine 'bearbeitbare' Version, das Modell, um final ein Ergebnis (Datenbankentwurf) systematisch zu entwickeln.



Ein Zentrales Ziel der Datenbankvorlesung.

### Was wird benötigt?

- Grundverständnis der Vorgehensweise und Definitionen.
- Abstraktionskonzepte und Tools, z.B. Entity-Relationship-Diagramme (ER-Diagramme) und Implementationskonzepte.

### Nebenbedingungen an das Modell

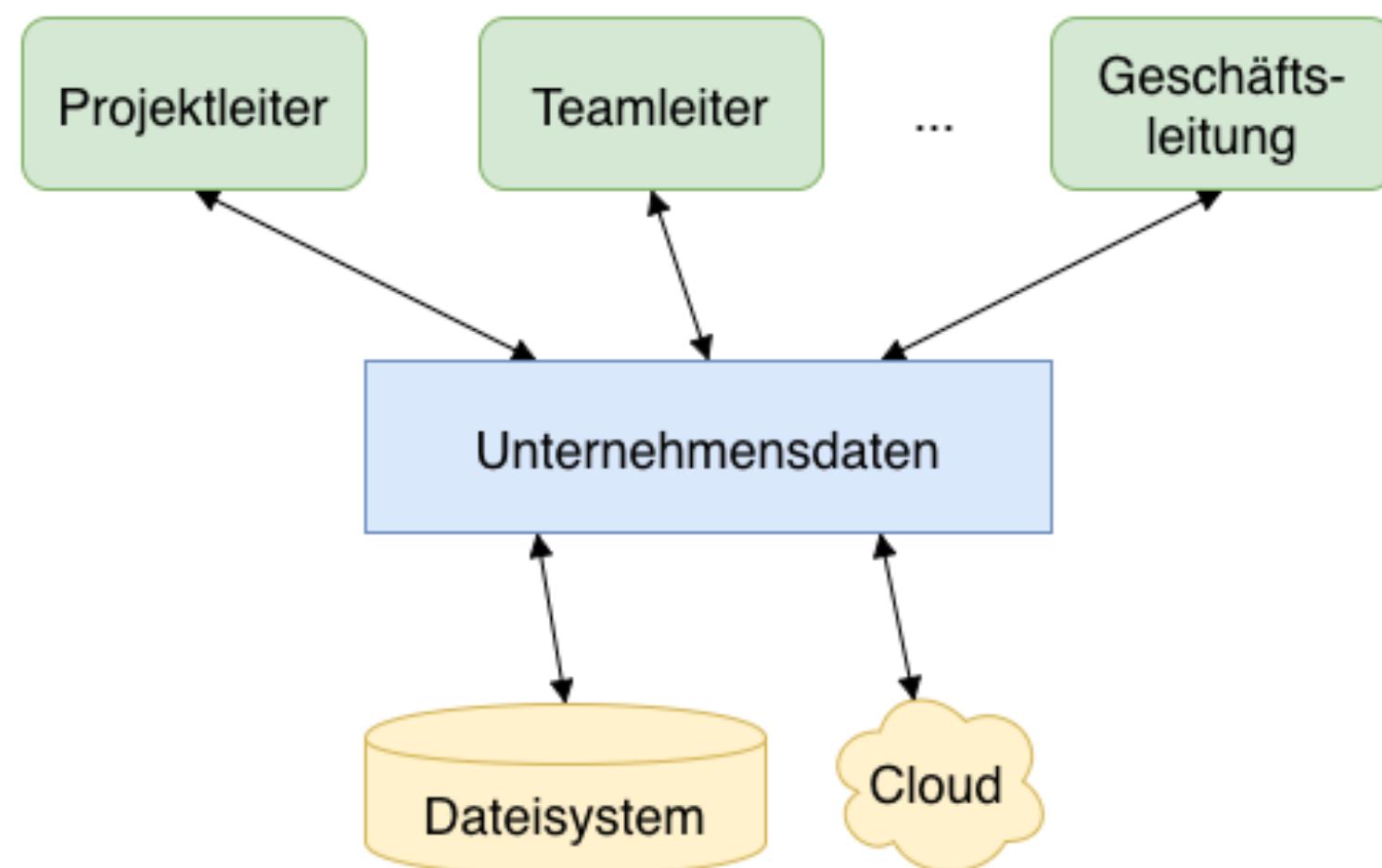
- Vollständig, korrekt, minimal, verständlich, erweiterbar.

**Q&A**

Woran erinnert Sie das?

## Abstraktionsebenen

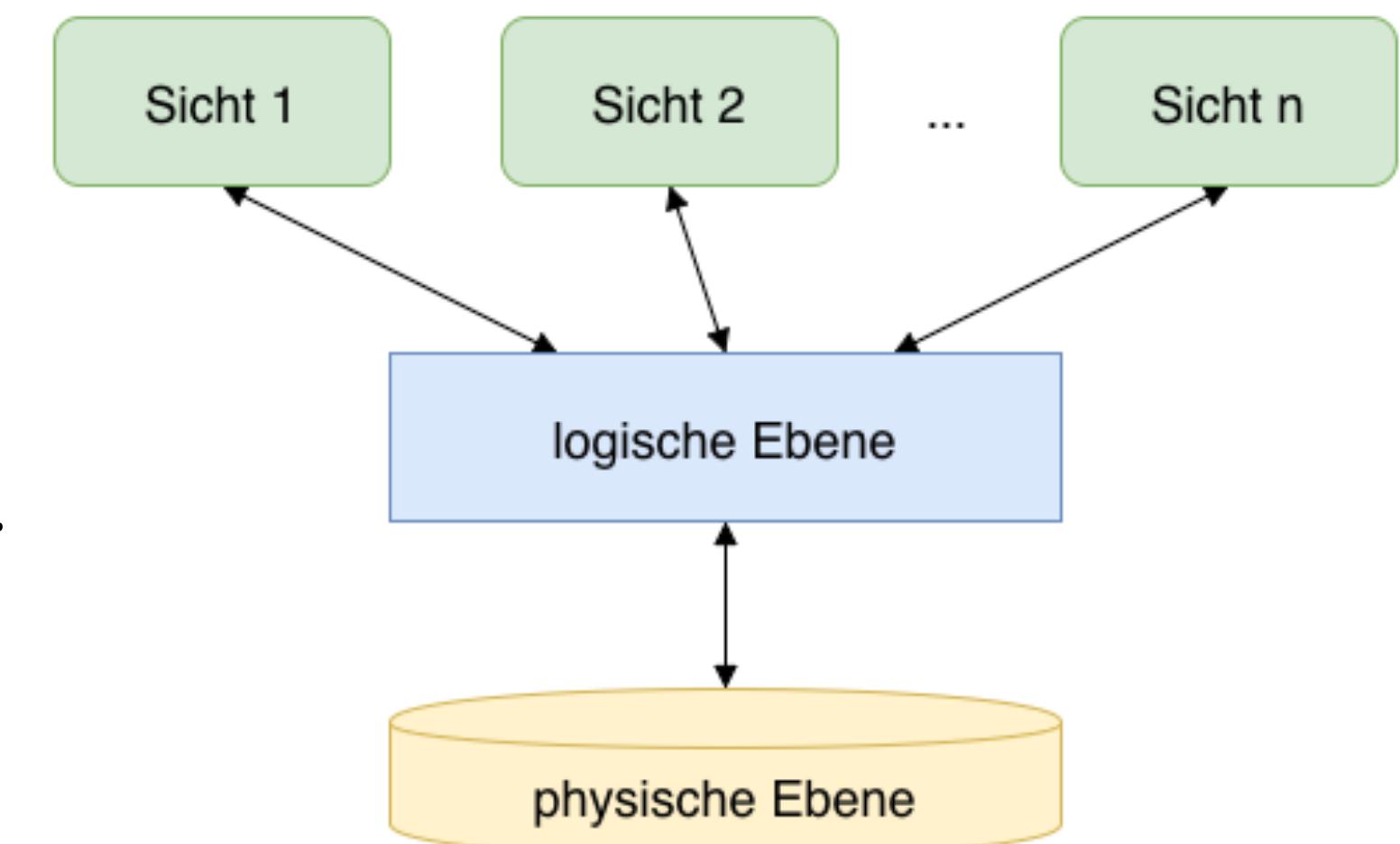
### Ein erster Ansatz...



Je Rolle unterschiedliche Sicht auf Datenteilmenge.

Logische Ebene der Daten,  
d.h. Strukturierung, Schema.

Physische Ebene der Daten,  
d.h. Storage Engine, Cloud.



**Idee:** Aufbau in Schichten, so dass Austausch oder Modifikation darunter liegender Schichten problemlos möglich ist.

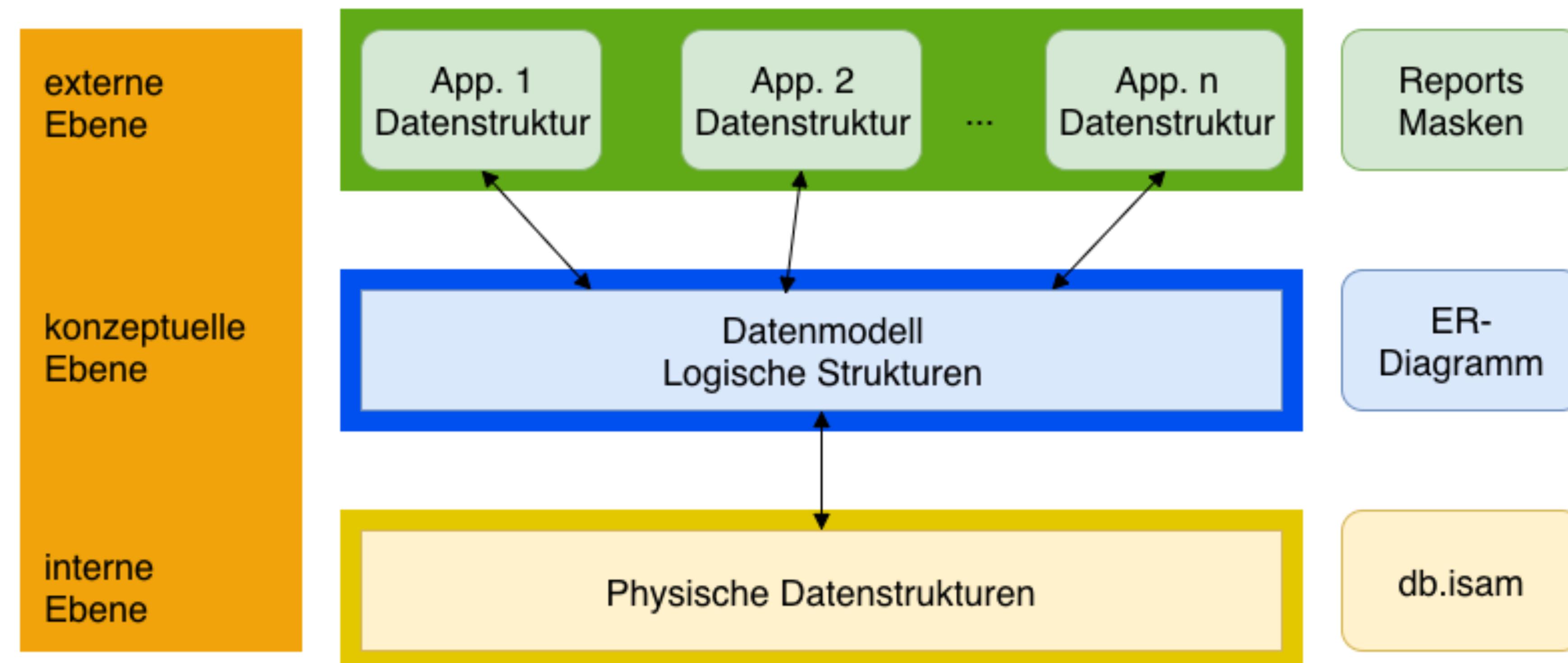


DBMS erfüllen zumeist nur physische Unabhängigkeit.

## Abstraktionsebenen

---

### ANSI/SPARC-Architektur



## Abstraktionsebenen

### ANSI/SPARC-Architektur

#### Drei getrennte Ebenen

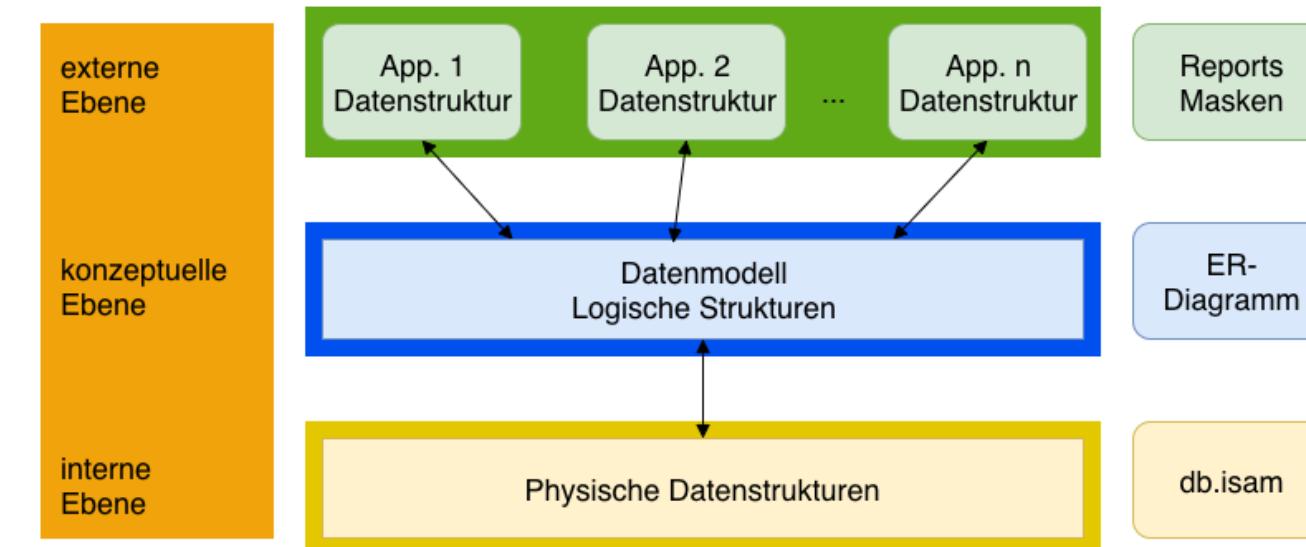
- *Externe Ebene*: Individuelle Benutzersichten.
- *Konzeptuelle Ebene*: Vollständige und redundanzfreie Darstellung aller Informationen, Konzeptuelles (ER-Modell) und Datenbankschema mit Daten und Relationen.
- *Interne/physische Ebene*: Physische Sicht der Datenbank im Computer.

#### Vorteile

- *Logische Datenunabhängigkeit*: Änderungen auf der konzeptuellen Ebene haben keine Auswirkungen auf die externe Ebene.
- *Physische Datenunabhängigkeit*: Änderungen auf der internen Ebene, z.B. Wechsel des DBS, wirken sich nicht auf die konzeptuelle oder externe Ebene aus.



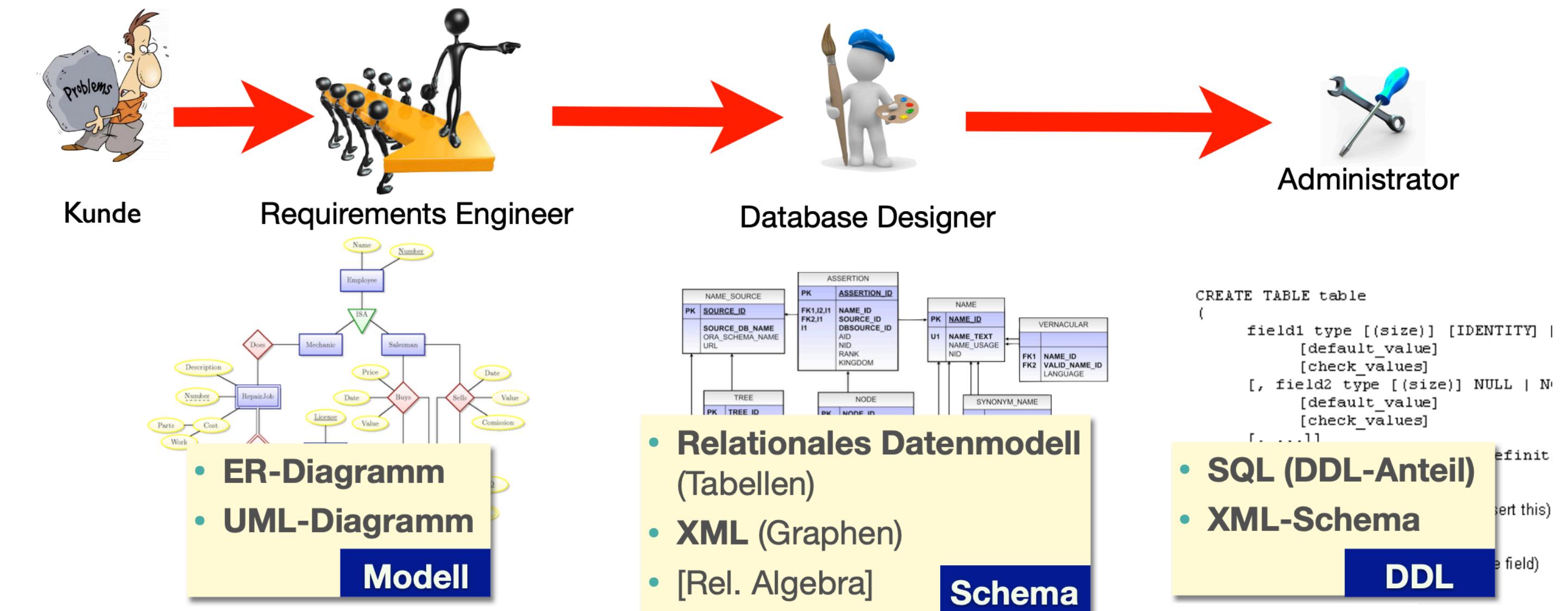
Soweit die Theorie...



## Modellierungsprozess

### Wie läuft die Entstehung einer Datenbank ab?

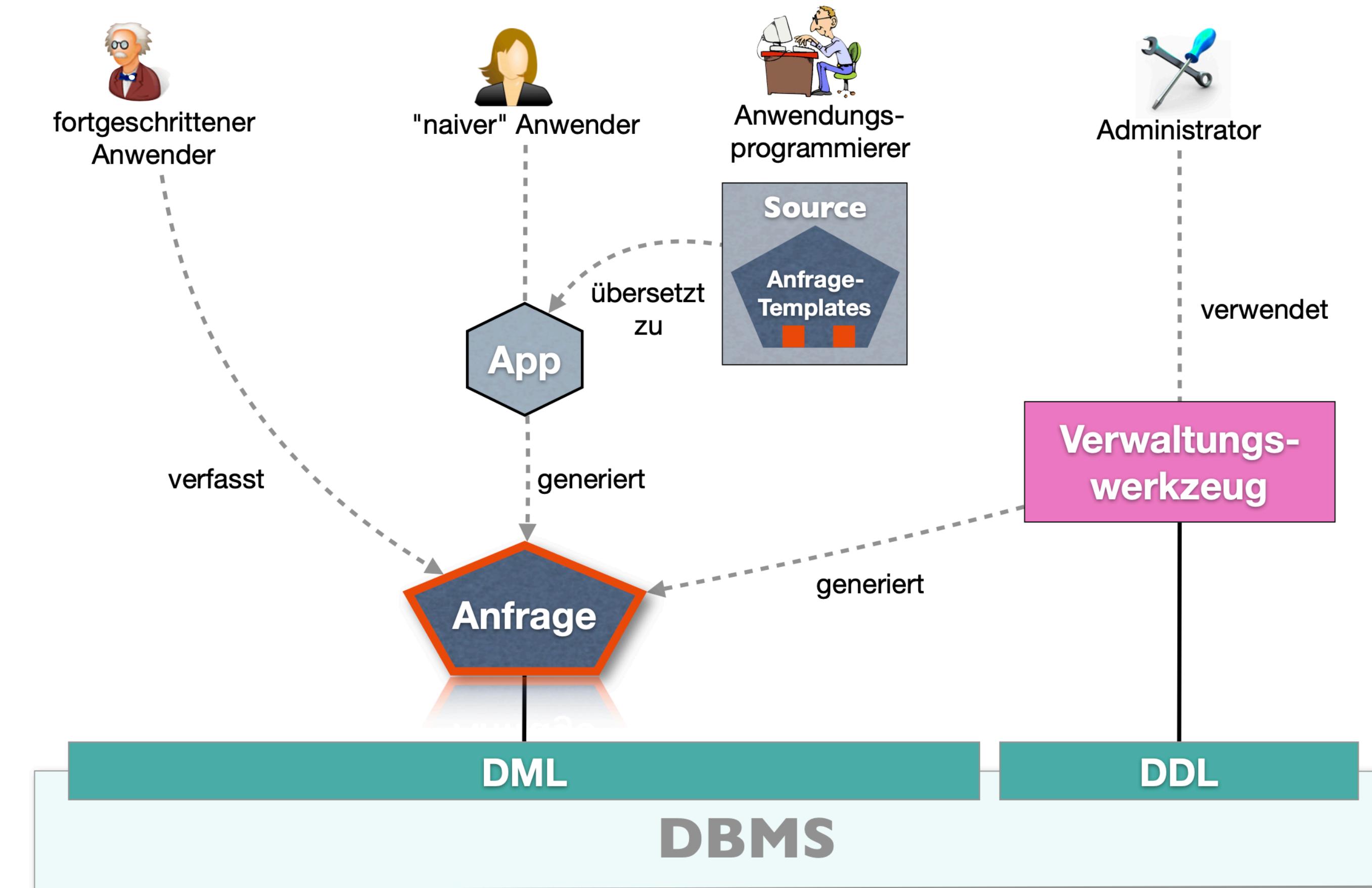
- **Rollen:** Wer übernimmt welche Aufgaben?
- **Anforderungsanalyse:** Was soll erreicht werden?
- **Entwürfe:** Konzeptuelle, Implementations-, Physische



Unterlagen Prof. Striegnitz

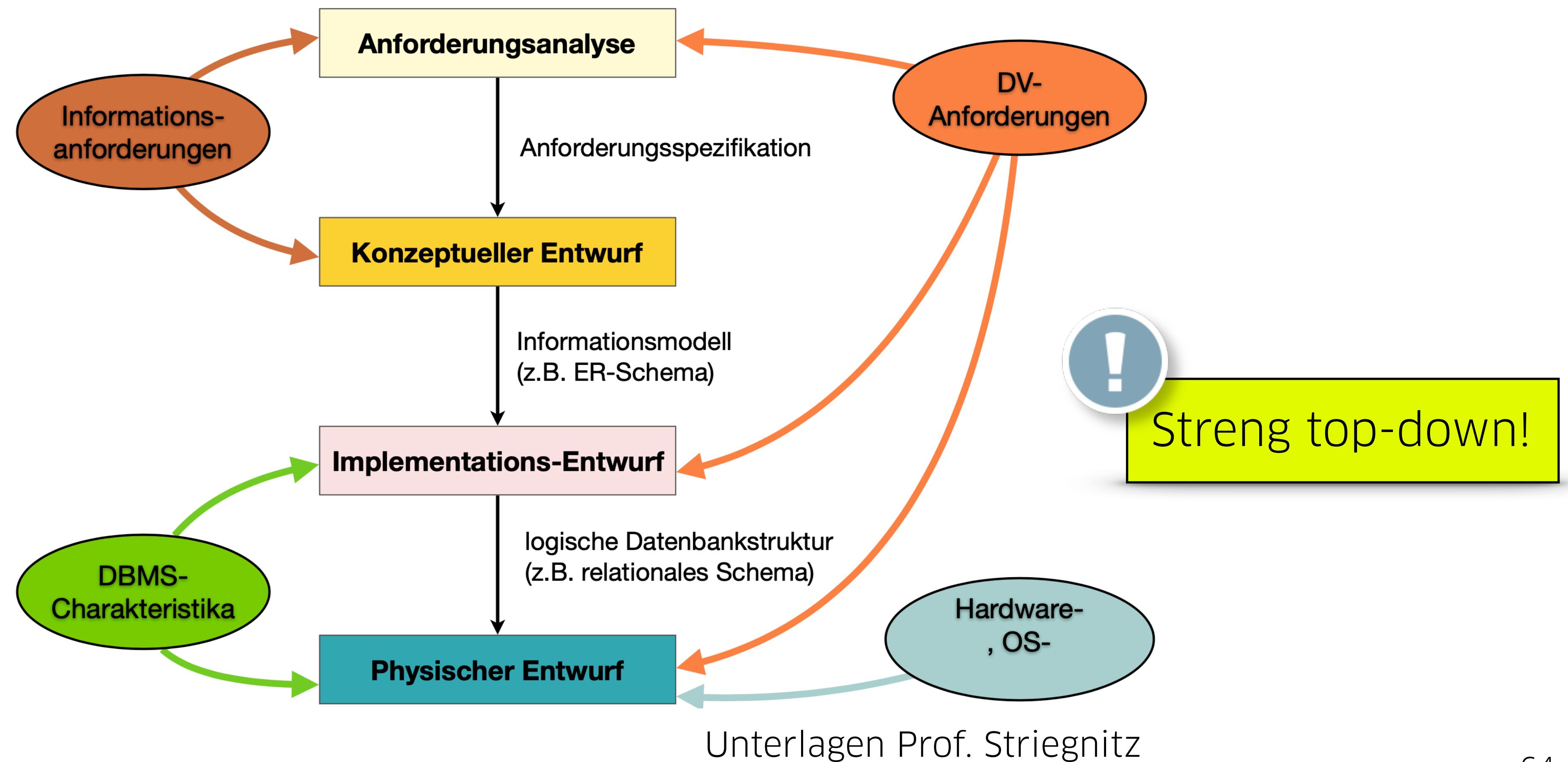
## Modellierungsprozess

### Architektur und Anwender



## Modellierungsprozess

### Phasen des Datenbankentwurfs

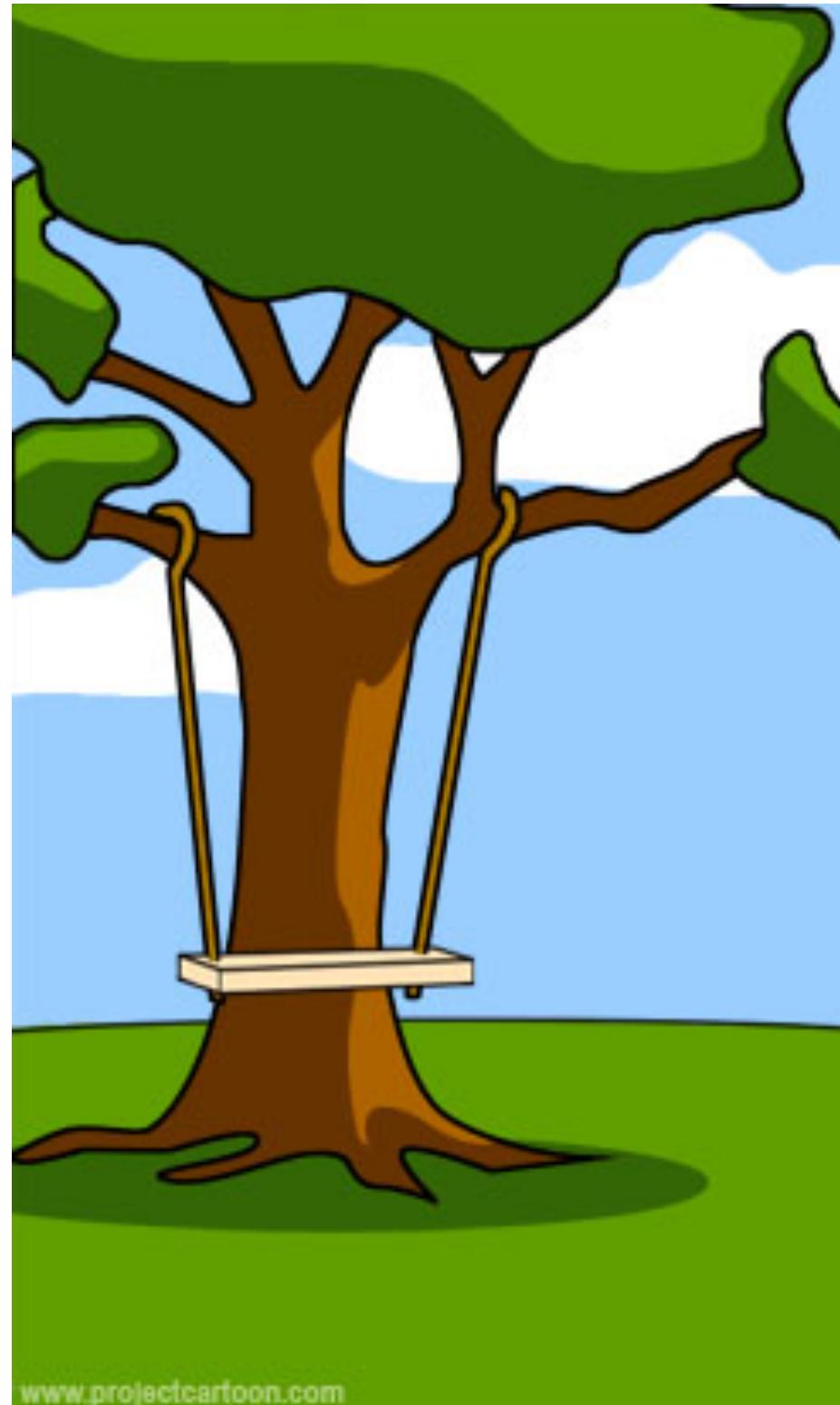


## Anforderungsanalyse

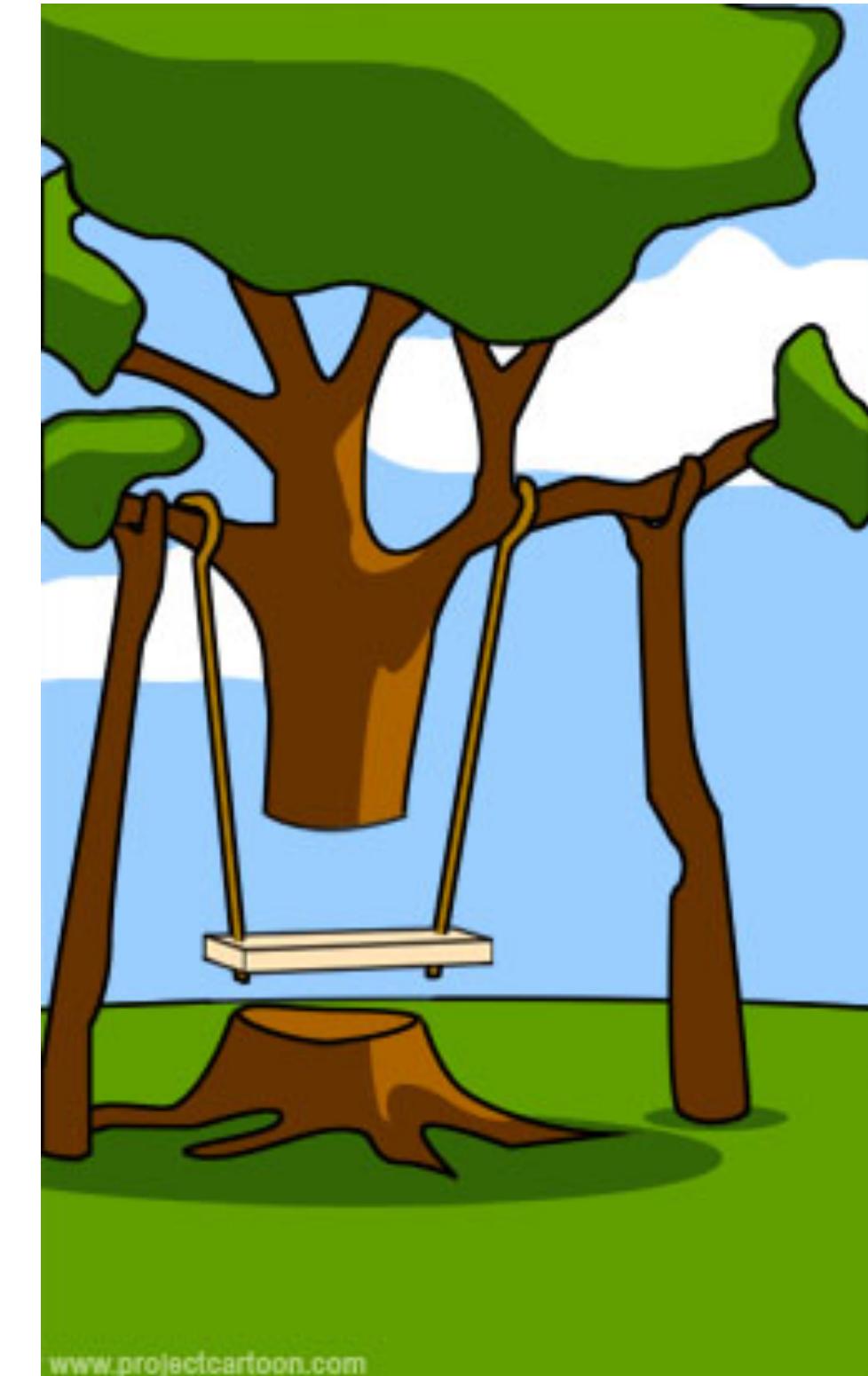
---



Wie der Kunde  
es erklärt hat



Was der Projekt-  
leiter versteht



Wie der Analyst  
es auffasst

Projectcartoon

## Anforderungsanalyse

---



Was der Programmierer  
geschrieben hat



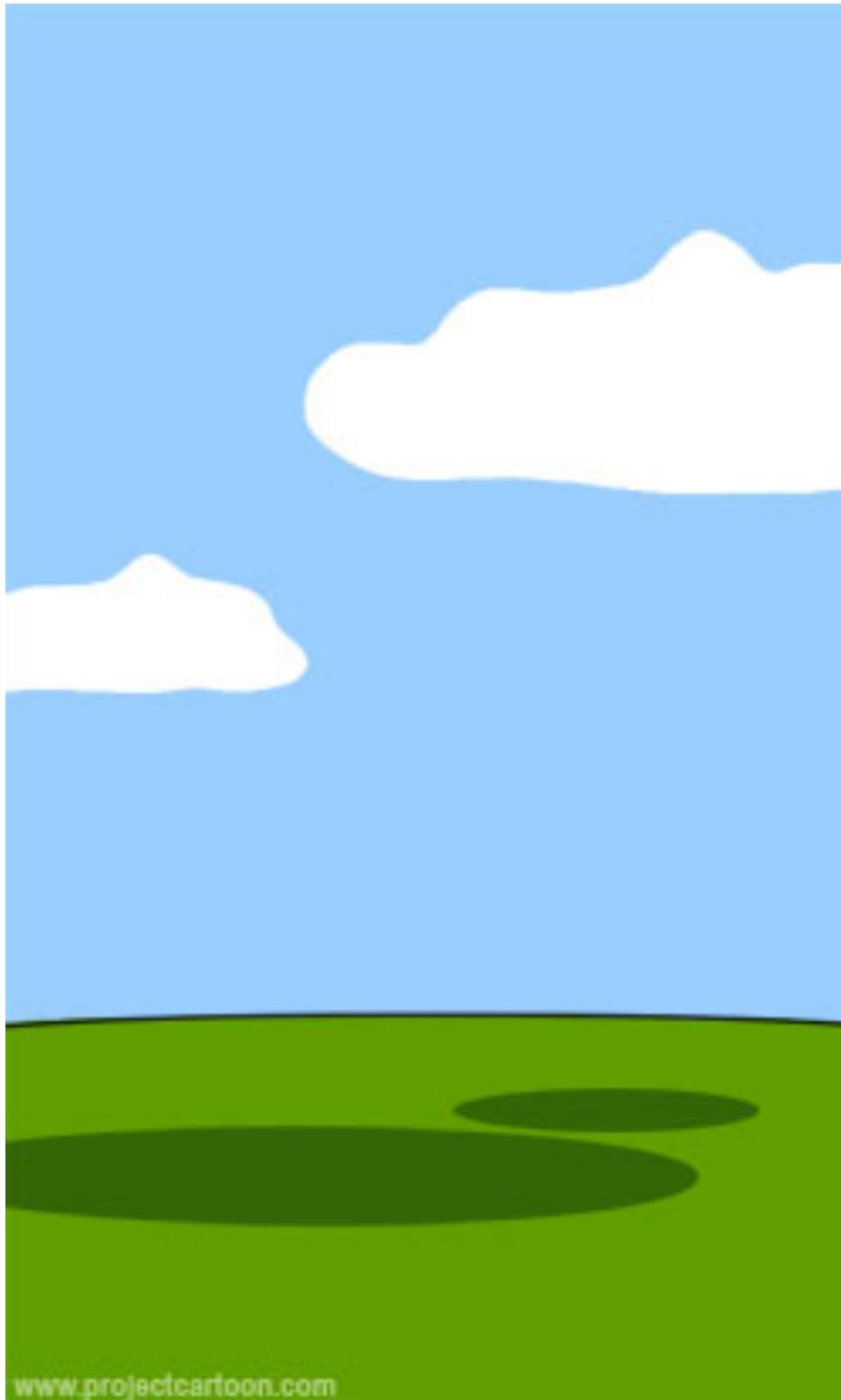
Was die Beta-  
Tester erhalten



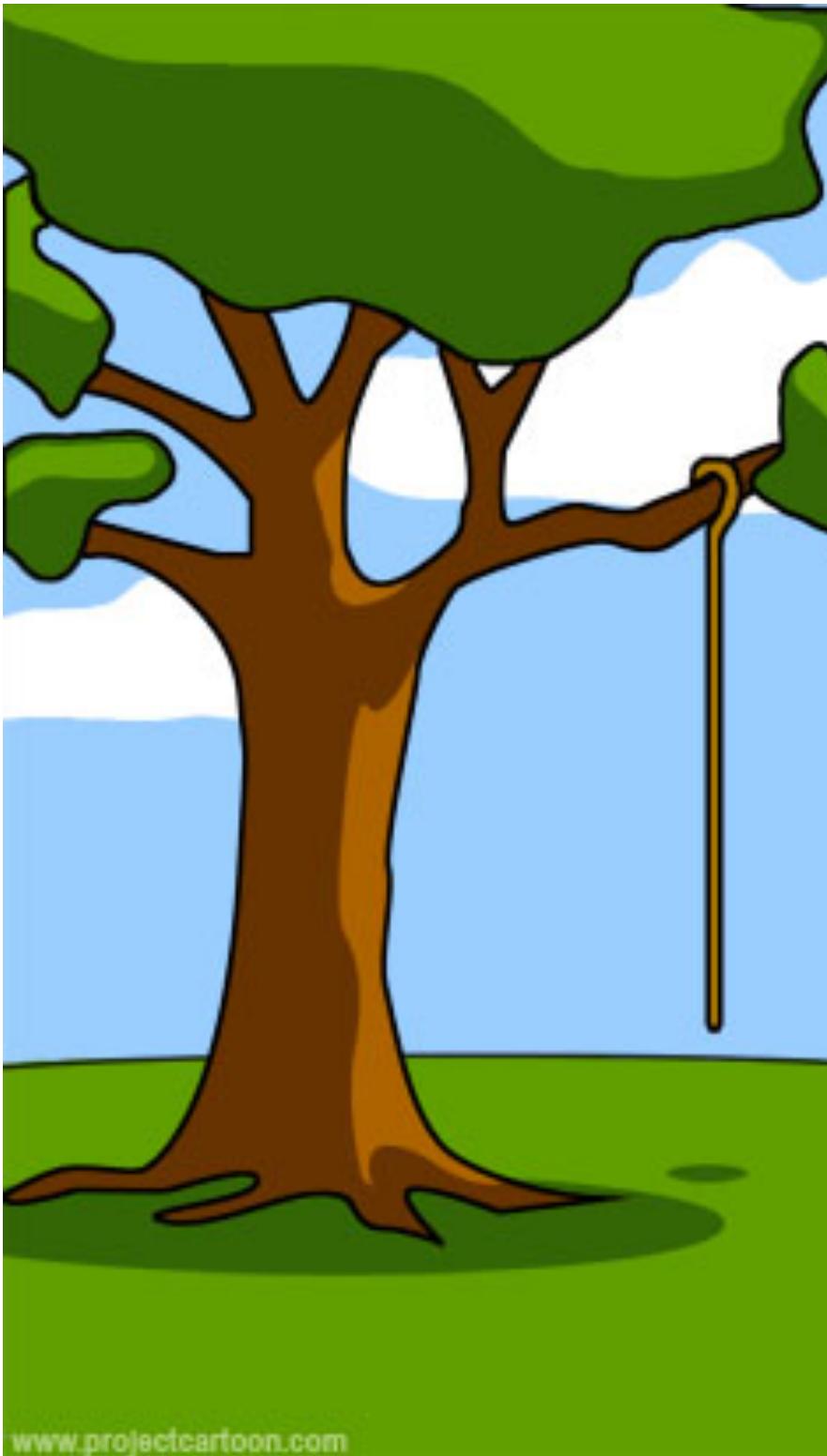
Wie der Vertrieb  
es verkauft

## Anforderungsanalyse

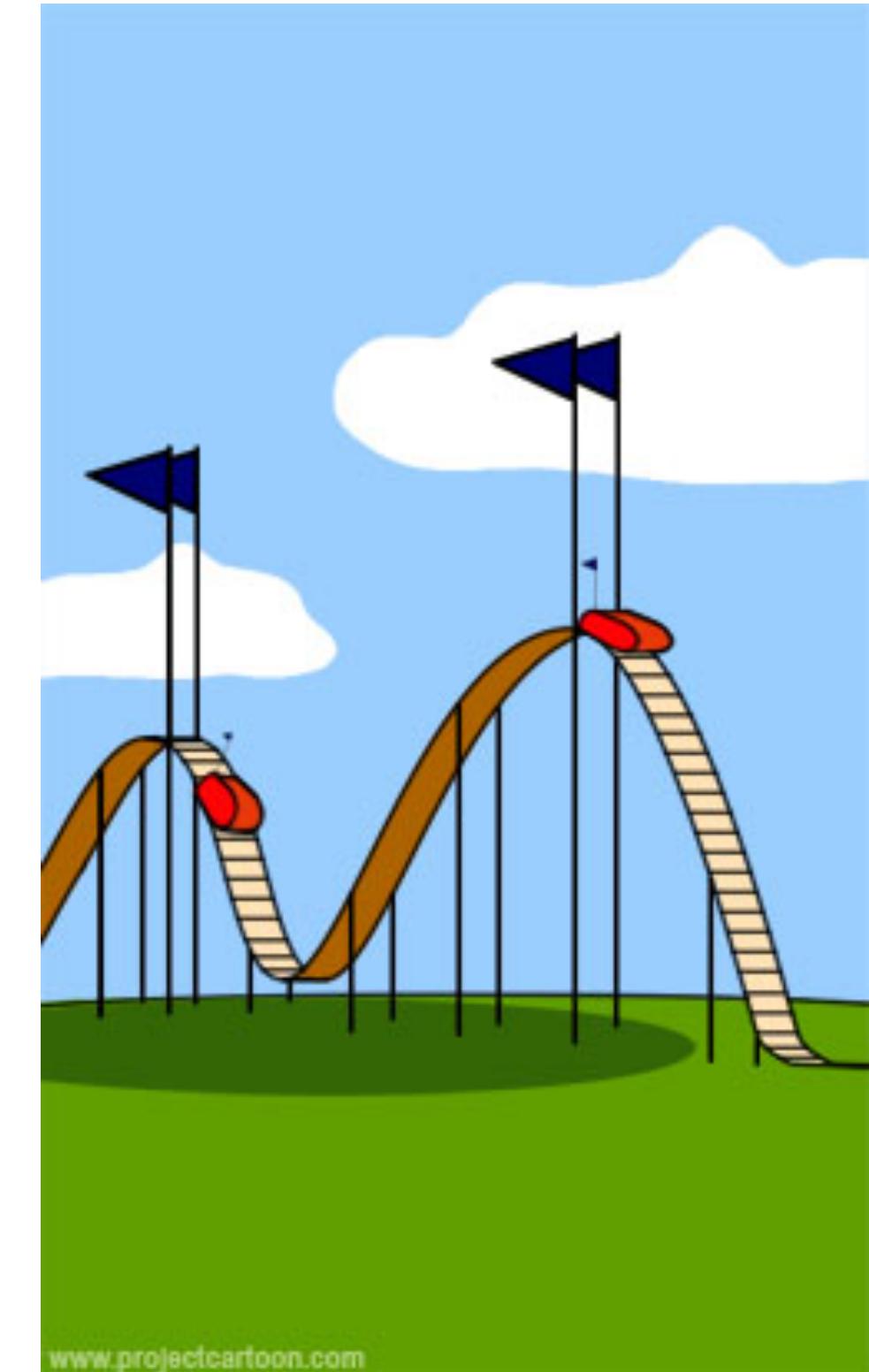
---



Wie die Doku.  
aussieht



Was installiert  
wurde

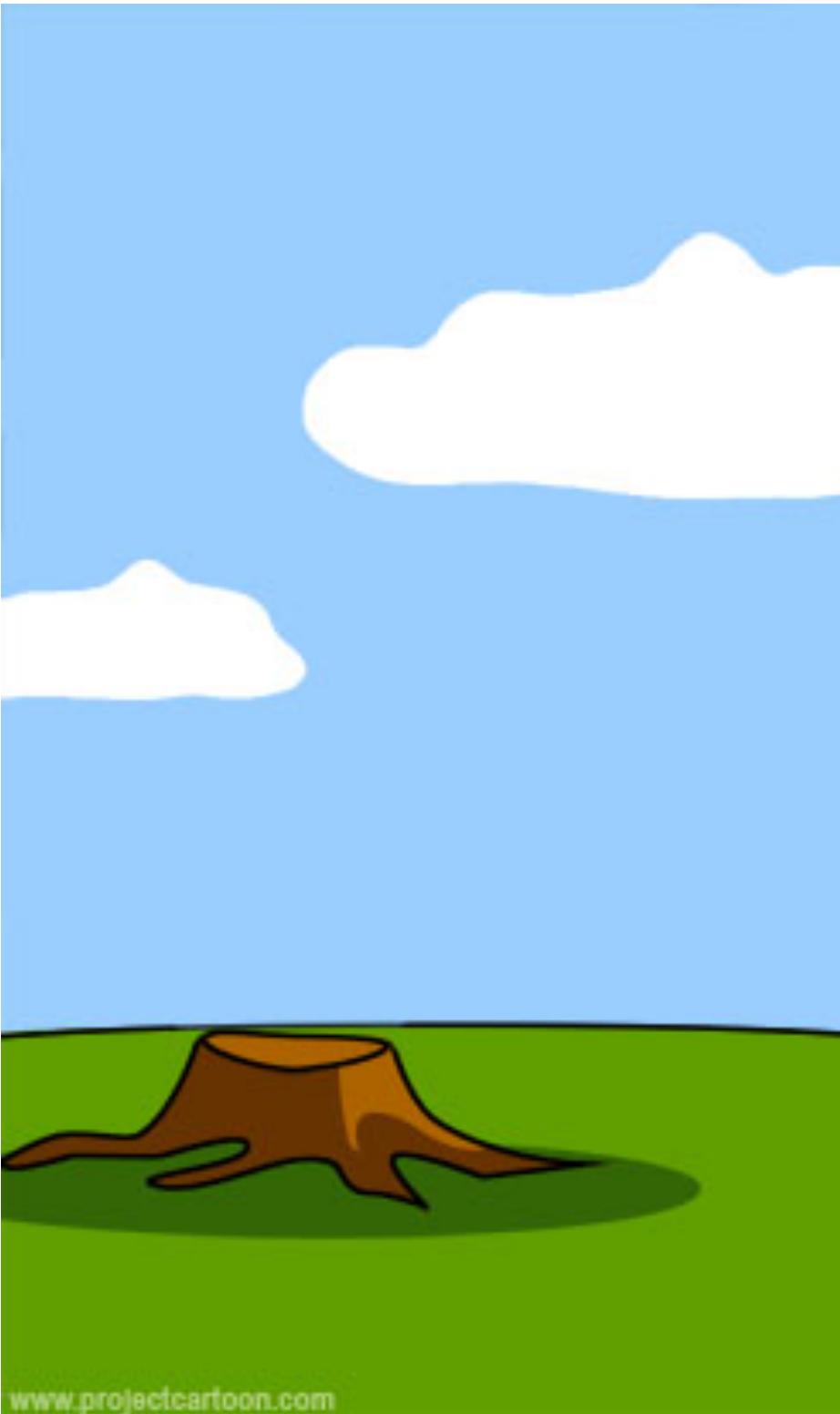


Wie es dem Kunden  
berechnet wurde

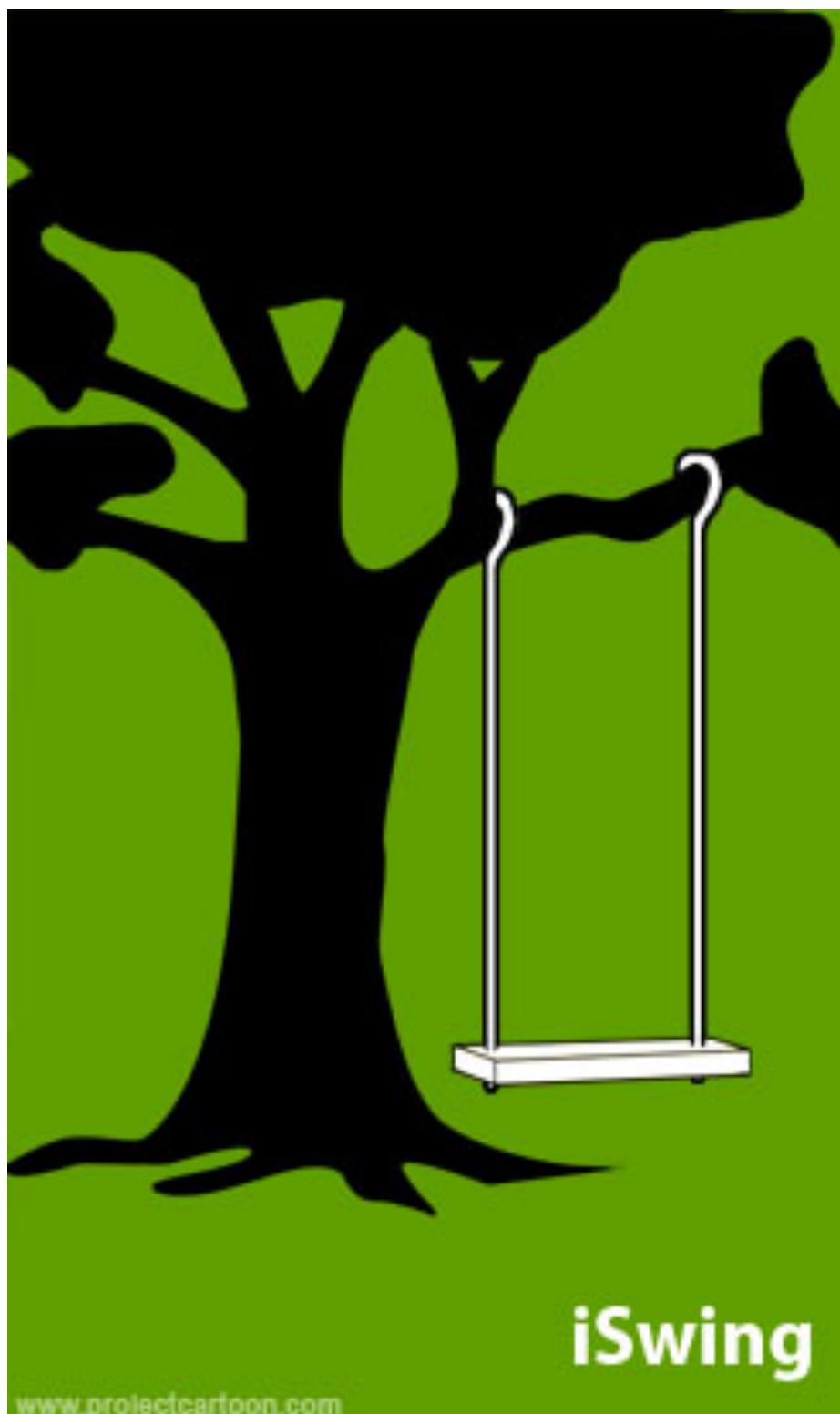
Projectcartoon

## Anforderungsanalyse

---



Wie der Support aussieht



Wie das Marketing damit wirbt



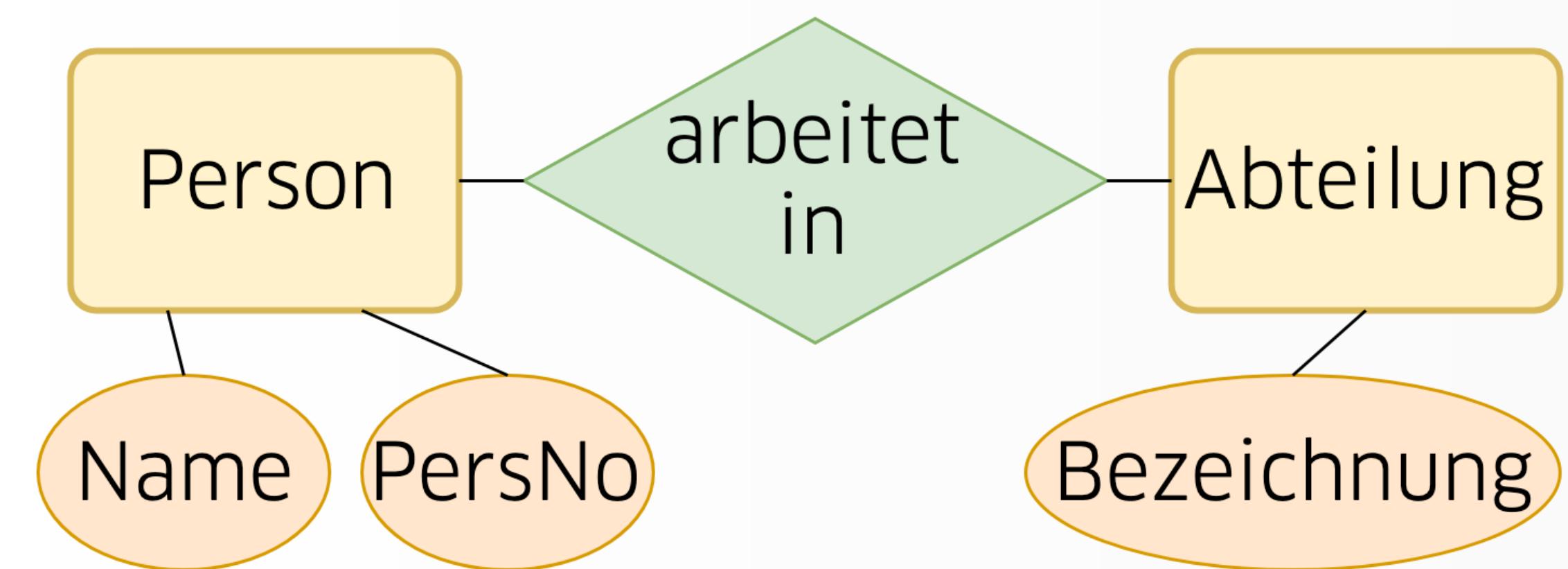
Was der Kunde gebraucht hätte

## Konzeptueller Entwurf: Entity-Relationship-Modell / -Diagramm

**Mini-Welt:** Person mit Name und Personalnummer arbeitet in Abteilung.

**Anforderungen:** ...

**Konzeptueller Entwurf:**



**ER-Diagramm**

### Q&A

- Wie sieht die Datenbank, bzw. das Schema, dazu aus?

- **Entity-Mengen**
- **Attribute**
- ◆ **Relation**

## Konzeptueller Entwurf: Datenbankentwurf

### Welche Möglichkeiten des Datenbankentwurfs gibt es hier?

- Klar ist eine Tabelle zu Person und eine zu Abteilung mit den jeweiligen Attributen.
- Aber wie die Relation `arbeitet_in` genau abzubilden ist, hängt an den zuvor ermittelten Anforderungen!
  - Wenn dort festgelegt ist, dass jede Person *in genau einer* Abteilung arbeitet, dann könnte man die Information über die Abteilung bei der Person ablegen.
  - Wenn aber eine Person *in mehreren* Abteilungen arbeitet, dann kann man entweder mehrere Abteilungen bei der Person ablegen - führt zu anderen Schwierigkeiten - oder aber man muss diese Zuordnung, also die Relation, gesondert speichern.

Wie beim  
Softwareentwurf!

#### Q&A

- Wie genau?
- Achtung:  
Redundanzen!

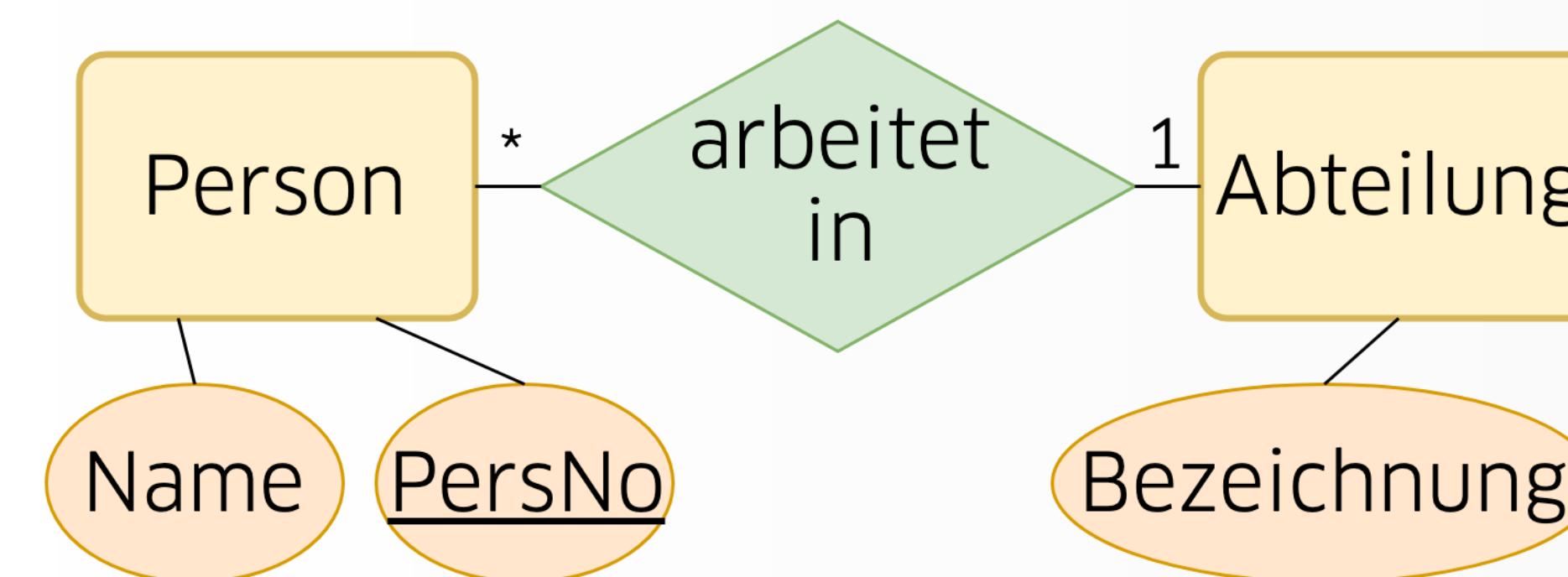
Der Datenbankentwurf ist nicht gleich dem ER-Diagramm!

## Entity-Relationship-Modell / -Diagramm

**Entwickelt 1976 von  
Peter Pin-Shan Chen**



- Entity-Mengen, Attribute, Schlüssel, Relationen, Kardinalitäten, Wertebereiche;
- unterstützt Klassifikation und Aggregation;
- grafische Darstellung durch (einheitliche) Diagramme.



**ER-Diagramm**

- **Entity-Mengen**
- **Attribute**
- **Schlüssel**
- ◆ **Relation**
- 1, \* **Kardinalität**



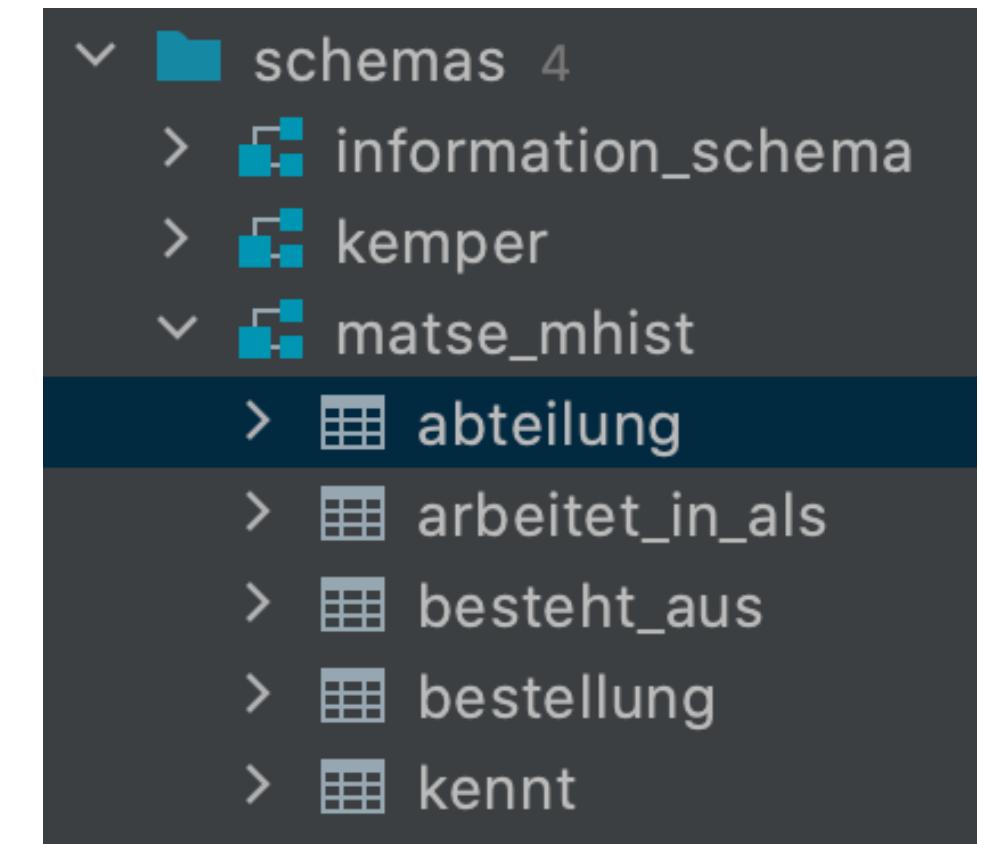
Gemeinsames Verständnis!

## Definitionen

---

### Schemata (Schemas)

- Allgemein formale Beschreibung der Struktur von Daten, z.B. XML-Schemata oder Datenbankschemata.
- Vergleichbar einem Namensraum aus der Programmierung.
- In einer relationalen Datenbank sind die Tabellen oft einem Schema zugeordnet. Dem gegenüber werden manchmal z.B. sog. NoSQL-DBS als 'schema-less' bezeichnet, aber es gibt dort ebenfalls 'collections' oder 'namespaces' und Strukturen in den Daten – Stichwort 'Schema validation'.



Schema in MariaDB

### Datenbankenschemata

- Konkrete Festlegung, welche Daten in welcher Form gespeichert werden und welche Beziehungen dazwischen bestehen.

## Definitionen

---

### Entity/Entität

- Repräsentiert abstraktes oder physisches, aber konkretes Objekt der realen Welt, z.B. Person 'Max', Stadt 'Aachen', Lied 'Hello' etc..
- Besitzt Werte zu Attributen bzw. Eigenschaften jeweils mit Typ, z.B. 'Name' 'Max' vom Typ 'Text'.
- Vergleichbar einer Objektinstanz.
- Entspricht einem einzelnen Datensatz in einer Datenbank, z.B. der Zeile in einer Tabelle oder einem 'Dokument'.



id	bezeichnung	stueckpreis	einheit
1	Spinat	1.99	PK
2	Vier Käse Pizza	2.39	ST
3	Spinatpizza	2.29	ST
4	Fischstäbchen	1.99	PK
5	Nudelpfanne	3.29	PK

Entitäten der Tabelle 'produkt'

## Definitionen

### Entity-Set/Entity-Menge oder Entitätstyp

- Menge E von Entitäten  $e \in E$  mit gleichen Eigenschaften (Klassifikation), z.B. 'Person'. Hier liegt eine Betrachtung als *Menge von Objekten* zugrunde.
- Oder, mit Schwerpunkt auf der Beschreibung der Eigenschaften, ist E ein Entitätstyp mit in der Regel typisierten Attributen, z.B. 'Name' vom Typ 'Text' (varchar).
- Entitätstyp ist vergleichbar einer Klassenbeschreibung.
- Entspricht häufig einer Tabelle in einer relationalen Datenbank, hierbei sind die Spalten die Attribute.

Definition etwas unscharf - ob Menge oder Typ gemeint ist, hängt am Kontext.

E

Symbol eines Entitätstypen im ER-Diagramm

Table:	
produkt	
	Columns (6)
id	int(11) -- part of primary key
bezeichnung	varchar(100)
warengruppe_id	int(11)
einheit	varchar(20)
stueckpreis	decimal(12,2)
umsatzsteuer	decimal(6,2)

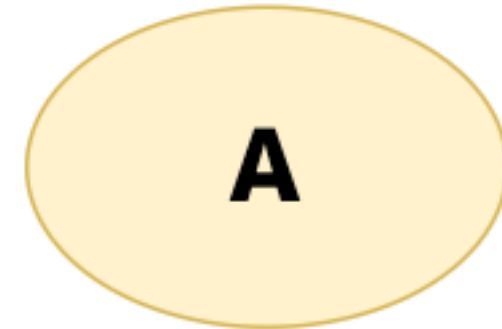
Tabelle 'produkt' mit Attributen und Datentypen

## Definitionen

---

### Attribute und Wertebereiche

- Eine Entität e ist durch die Werte ihrer Attribute charakterisiert und ein Entitätstyp E durch die Attribute selber. Jedem Attribut A ist ein Datentyp T und damit implizit oder auch explizit ein Wertebereich D (Domain) zugeordnet, der festlegt, welche Attributwerte zulässig sind:  $A \in D$  oder  $A:D$  bzw.  $A:T$ .
- Achtung *Besonderheit*: Nullwert (NULL). Spezieller Attributwert, dessen Bedeutung variiert, d.h. z.B. ist der Wert unbekannt oder noch nicht festgelegt oder nicht möglich. NULL kann, muss aber nicht, im Wertebereich liegen.
- Wir notieren  $E [A_1:D_1, A_2:D_2, \dots A_n:D_n]$  für einen Entitätstyp E mit den Eigenschaften  $A_1, \dots, A_n$ . Wertebereiche bzw. Typen werden, wenn nicht relevant, auch ausgelassen.
- $W(D)$  bezeichnet alle real existierenden Werte der Domain D in einer Datenbank.



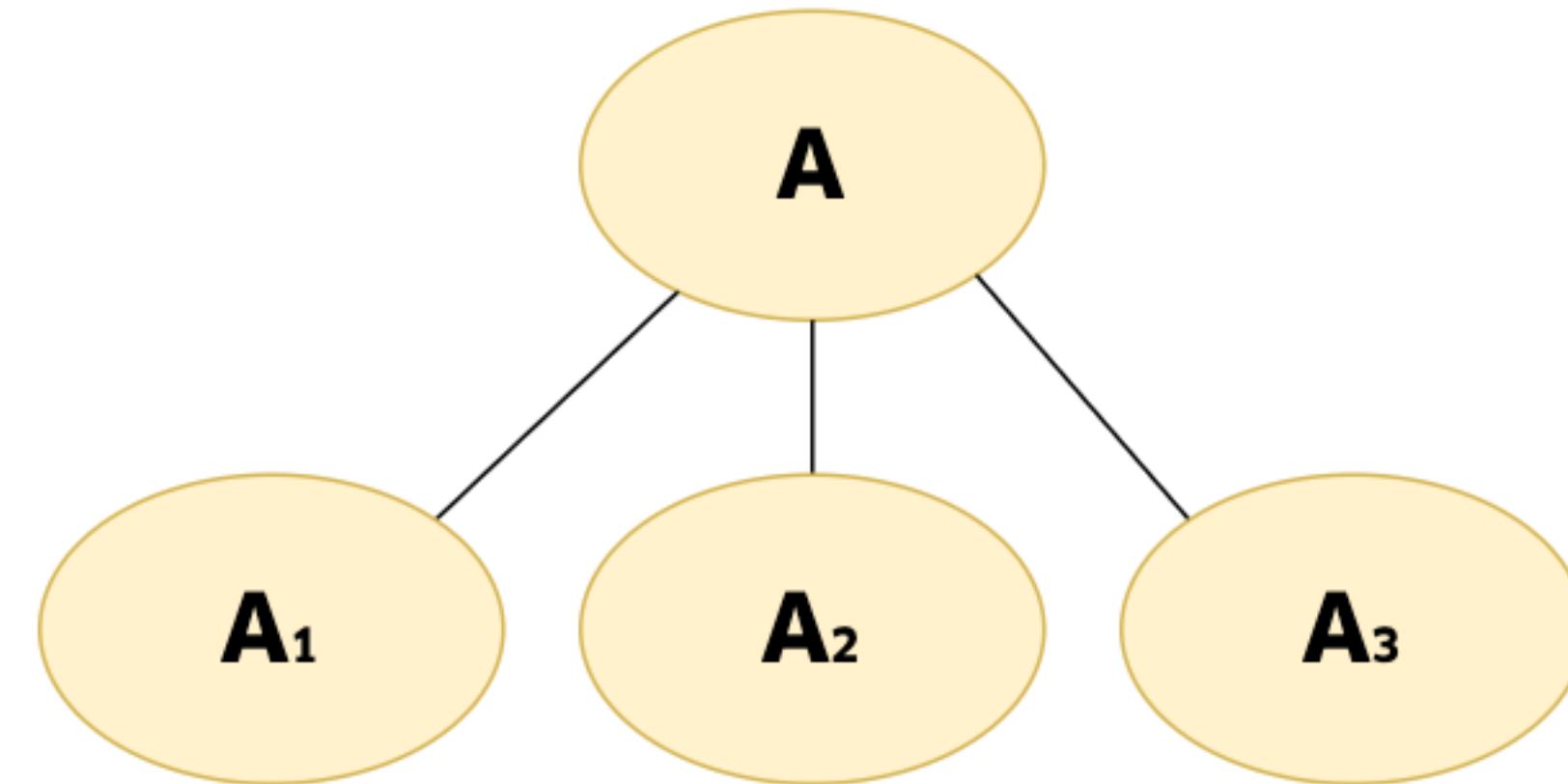
Symbol eines Attributs  
im ER-Diagramm

## Definitionen

---

### Zusammengesetzte Attribute

- Zusammengesetzte Attribute haben selbst Attribute, z.B.
  - NAME: [Vorname: char(30), Nachname: char(30) ]
  - ANSCHRIFT: [Strasse: char(30), Ort: char(30), PLZ: char(5) ]
- Domain für ein zusammengesetztes Attribut A mit Unterattributen  $A_1, \dots, A_n$ :  
 $A: [A_1:D_1, \dots, A_n:D_n]: D_1 \times \dots \times D_n$ .



Symbol eines zusammengesetzten Attributs im ER-Diagramm

### Q&A

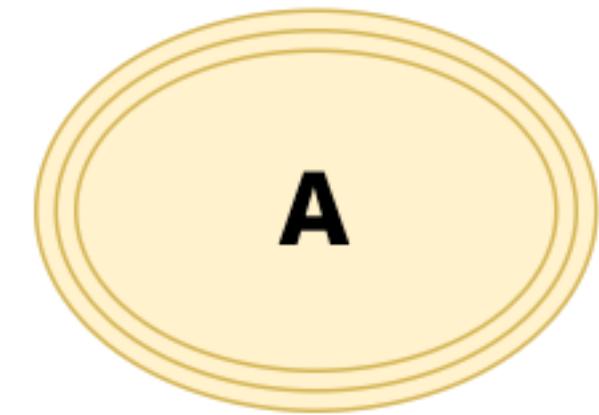
- Was für ein Problem ist mit Blick auf eine Datenbank bzw. Tabelle zu erwarten?

## Definitionen

---

### Mehrwertige Attribute

- Ein mehrwertiges Attribut kann mehrere Ausprägungen (Werte) haben, z.B.
  - AUTOFARBE: {char (20)} (ein Auto hat mehrere Farben)
  - TELEFONNR: {char (30)} (eine Person hat mehrere Tel.nr.)
- Domain für ein mehrwertiges Attribut E: { A }



Symbol eines mehrwertigen Attributs im ER-Diagramm

### Q&A

- Was für ein Problem ist mit Blick auf eine Datenbank bzw. Tabelle zu erwarten?

## Definitionen

---

### Schlüsselkandidaten

- Ein Schlüsselkandidat, oder kurz Schlüssel (key), ist ein einwertiges Attribut oder eine Attributkombination, die jede Entität eindeutig identifiziert.
- Es ist möglich, dass mehrere Schlüsselkandidaten existieren. Der sog. Primärschlüssel ist ein ausgewählter Schlüsselkandidat. Seine Primärschlüsselattribute werden im ER-Diagramm durch Unterstreichung gekennzeichnet.
- Häufig wird eine künstliche Identifikationsnummer (ID) als Primärschlüssel gewählt, aber eine Postleitzahl in einem Städteverzeichnis wäre auch möglich.

#### Q&A

- Vor- oder Nachteile künstlicher Primärschlüsseln?

A

Symbol eines Primärschlüssel-attributs im ER-Diagramm

Table:			
Columns (6)		Keys (1)	Indices (1)
		<b>id</b> int(11) -- part of primary key	
		bezeichnung varchar(100)	
		warengruppe_id int(11)	

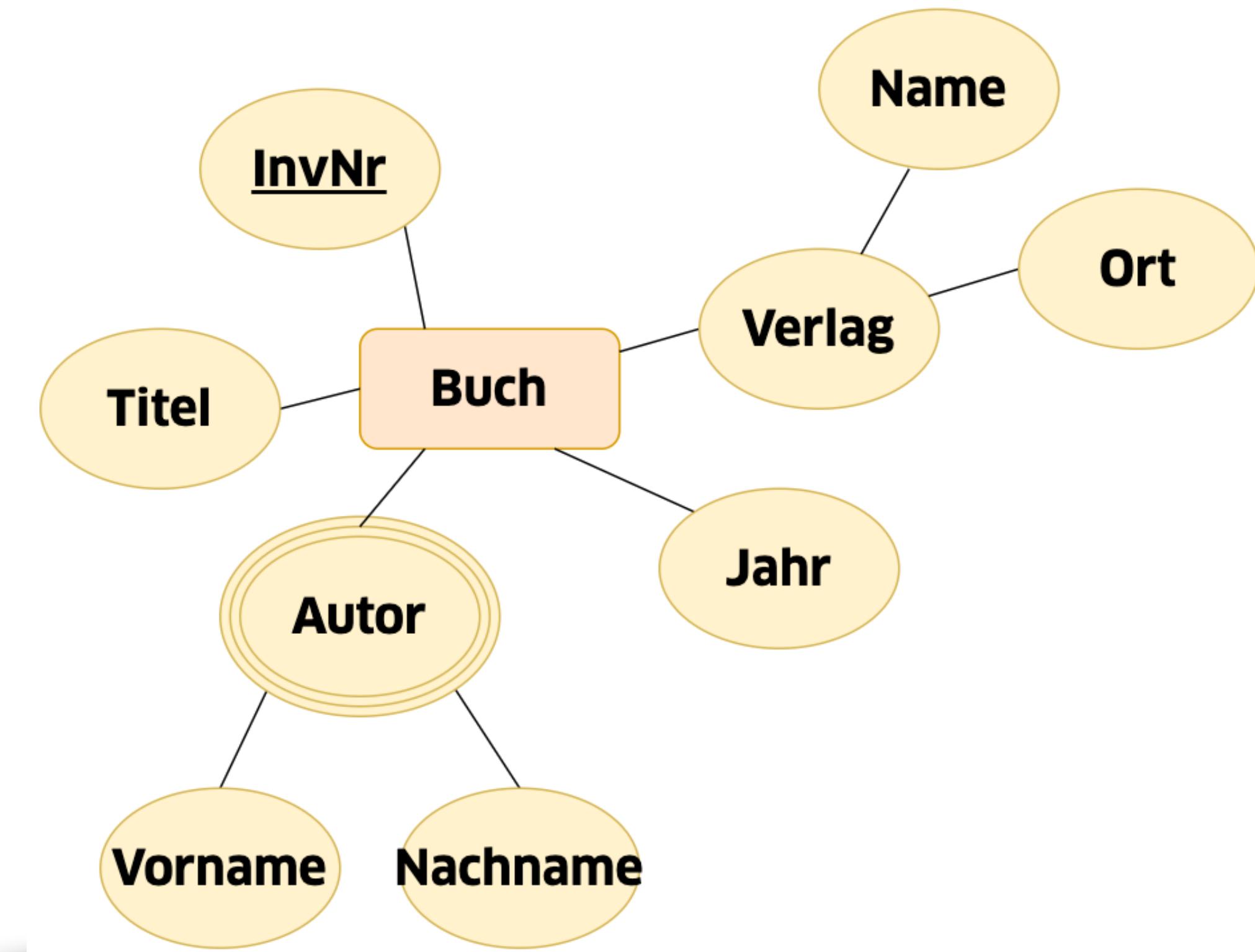
Primärschlüssel 'id'  
in der Tabelle 'produkt'

## Beispiel ER-Diagramm

---

### Beispiel Buch

- Angabe des Entitättyps in der Form E: <[Attribute],{Primärschlüssel}>, d.h.



Buch: < [ InvNr, Titel, Jahr, Verlag:[Name,Ort], {Autor:[Vorname,Nachname]} ], {InvNr} >

# ER-Diagramm Übersicht

## Grafische vs. formale Darstellung

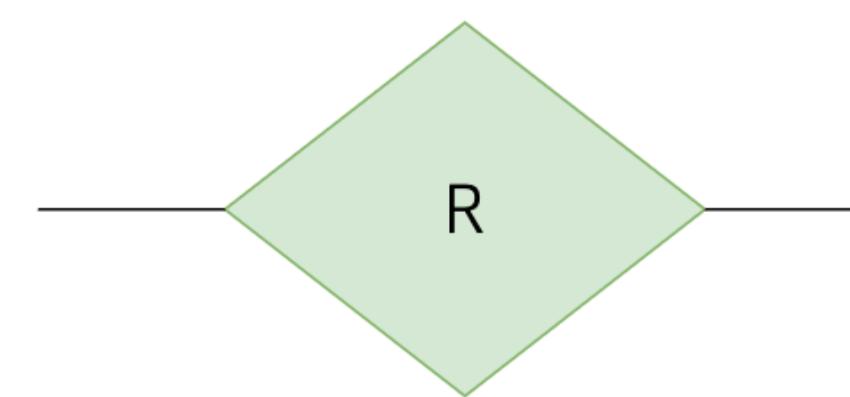
	ER-Diagramm	Formal
Entity-Set		$E: <[S], \{K\}>$ $S = [A_1, \dots, A_n]$ $K = \{A_{i1}, \dots, A_{im}\}$
Attribut		$A:D$
Mehrwertiges Attribut		$A:\{D\}$
Zusammengesetztes Attribut		$A:[A_1:D_1, \dots, A_n:D_n]$

## Definitionen

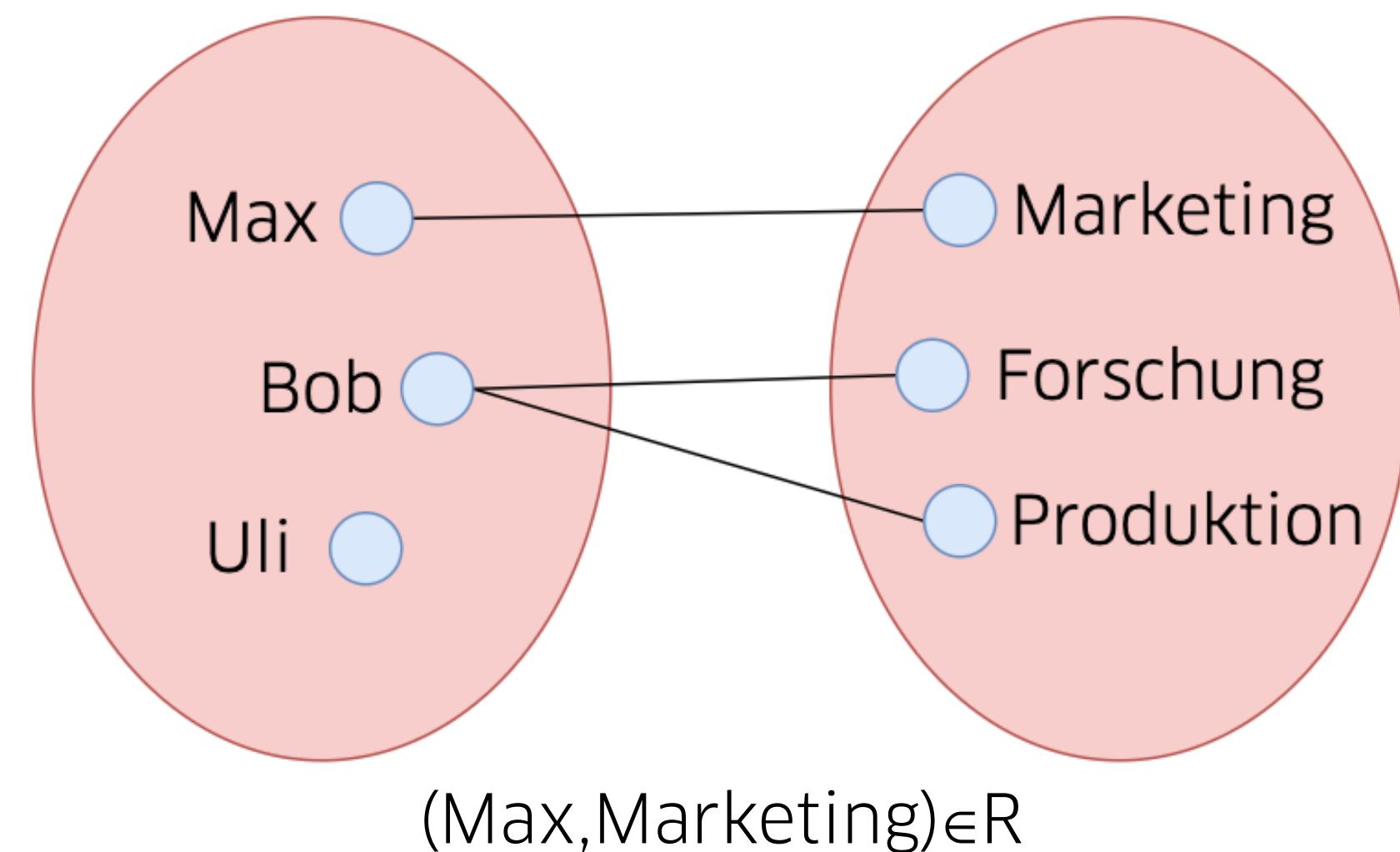
---

# Relations/Relationships/Relationen/Beziehungen

- Eine Relationship-Menge  $R$  entspricht einer mathematischer Relation zwischen  $n$  Entity-Mengen  $E_i$ :  
 $R \subseteq E_1 \times \dots \times E_n$ , häufig  $n=2$  oder  $n=3$ .
- Anders ausgedrückt: Eine Relation gibt an, ob eine Entität  $e_1 \in E_1$ , also  $e_1 \in E_1$ , zu einer anderen Entität einer zweiten Menge,  $e_2 \in E_2$ , in Beziehung steht oder nicht, d.h.  $(e_1, e_2) \in R$  oder  $(e_1, e_2) \notin R$ .
- Die Relation ist beschreibend: arbeitet in



Symbol einer  
Relation im  
ER-Diagramm

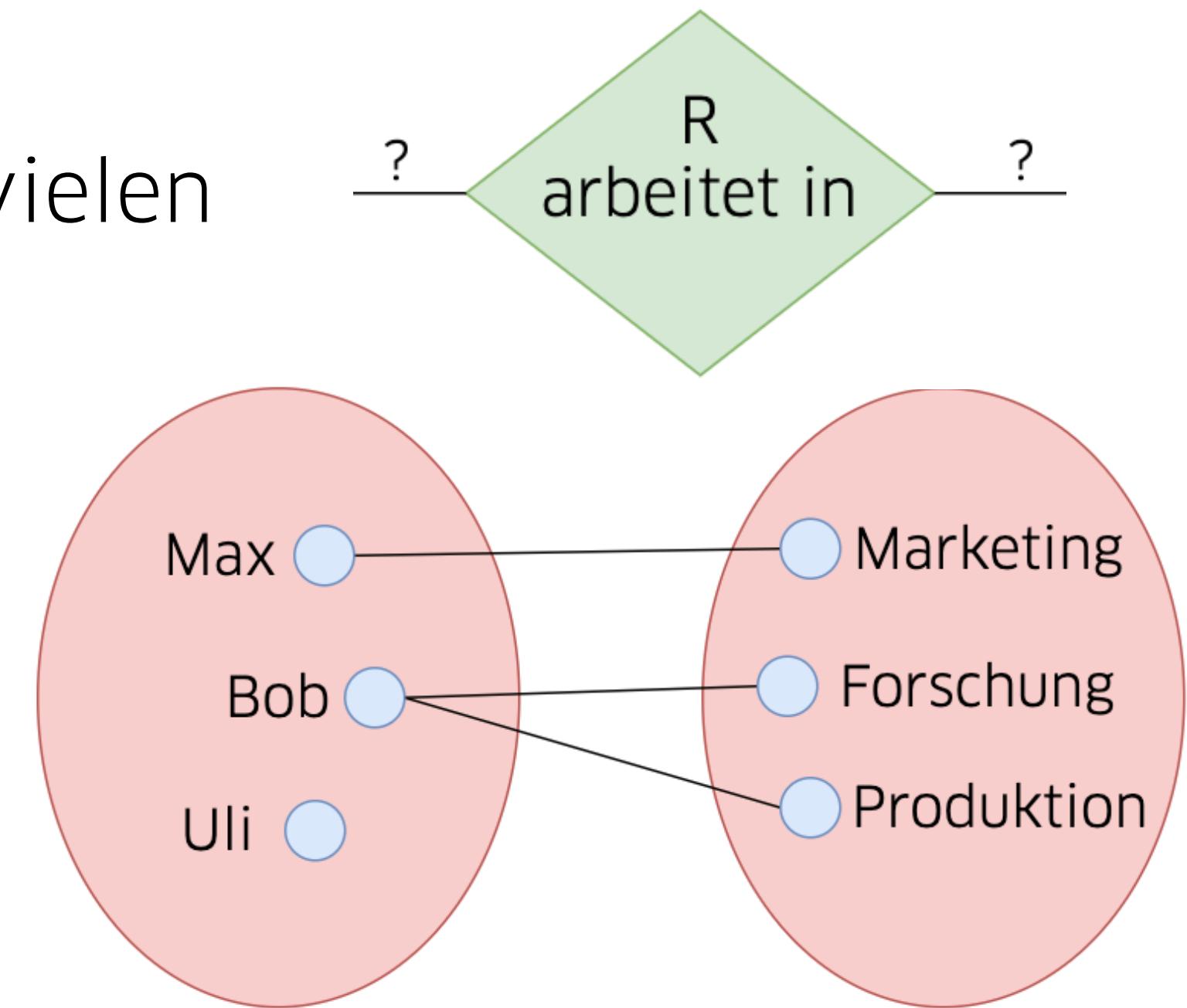


## Definitionen

---

### Kardinalität

- Die Kardinalität (die '?' an der Relation) gibt an, zu wie vielen Elementen ein Element in Beziehung steht.
- Es gibt grundsätzlich folgende Abbildungsbeziehungen zwischen Entity-Mengen in der Chen-Notation: **1:1, n:1, 1:n, n:m**. (Details und Beispiele folgen).
- Die Kardinalität ergibt sich im Wesentlichen aus den Anforderungen und bestimmt ganz massgeblich den Datenbankentwurf.
- Die Abbildung des konzeptuellen Entwurfs, also des ER-Diagramms, auf den relationalen Implementationsentwurf, d.h. z.B. die Abbildung in die Datenbankstrukturen/Tabellen, folgt im Kapitel 'Relationales Modell'. Mögliche Abbildungen in Nicht-relationalen Datenbanken betrachten wir gegen Ende.



## Relationen und Kardinalität

---

### Mini-Welt Fotoshooting I

**Hintergrund:** Auf einer Familienfeier werden Fotos aufgenommen. Diese Fotos sollen in einer Datenbank abgelegt werden. Dabei besitzen Fotos u.a. Datum und Zeit und eine fortlaufende Nummer und Personen mind. einen Namen. Es gilt, dass auf einem Foto max. eine Person zu sehen sein soll und dass es max. ein Bild von einer Person geben darf.

- **Ziel Konzeptueller Entwurf:**

- zunächst ER-Model
- später: Implementationsentwurf  
(DB-Schema)

#### Q&A

#### Was können Sie identifizieren?

- Entitätstypen?
- Attribute?
- Relationen?
- Kardinalitäten?

## Relationen und Kardinalität

---

### Mini-Welt Fotoshooting I

**Hintergrund:** Auf einer Familienfeier werden **Fotos** aufgenommen. Diese Fotos sollen in einer Datenbank abgelegt werden. Dabei besitzen Fotos u.a. *Datum* und *Zeit* und eine fortlaufende *Nummer* und **Personen** mind. einen *Namen*. Es gilt, dass auf einem Foto max. eine Person zu sehen sein soll und dass es max. ein Bild von einer Person geben darf.

### Mögliche Vorgehen

- Sie identifizieren zunächst die massgeblichen **Entitätstypen**,
- zusammen mit deren *Eigenschaft/Attributen*, und
- den Beziehungen und Kardinalitäten untereinander.  
Kardinalitäten sind ggf. implizit gegeben. Die Aussage max. eine Person zu max. einem Bild beschreibt eine 1:1-Beziehung zwischen Foto und Person.



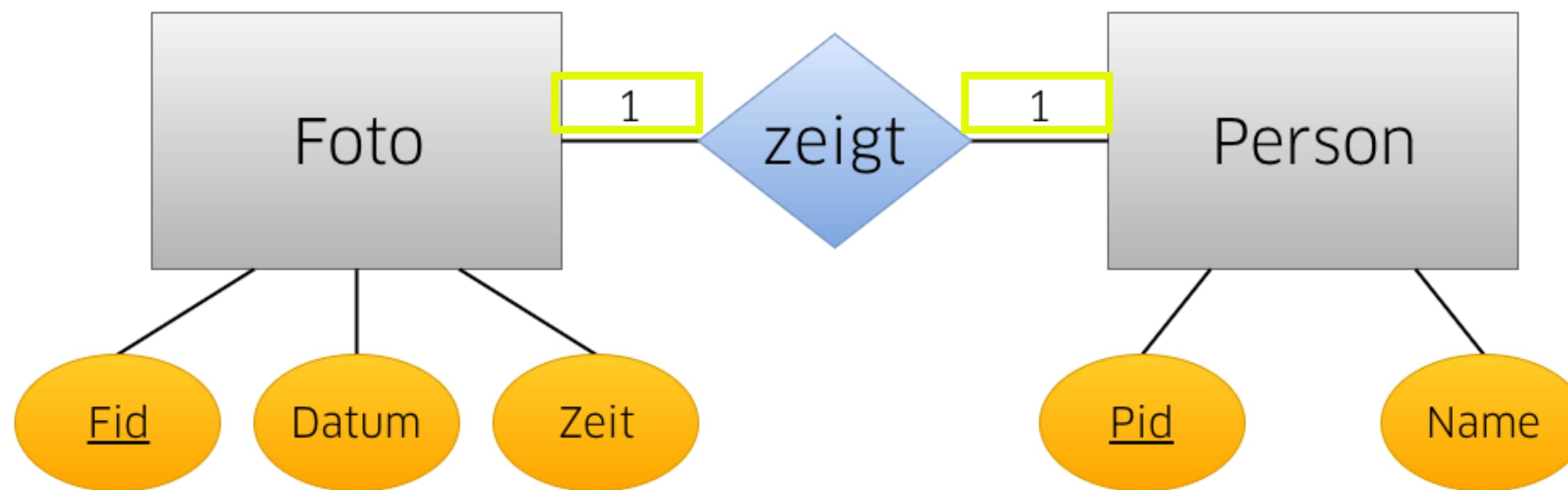
Markieren Sie  
die Begriffe

## Relationen und Kardinalität

---

### ER-Diagramm mit Chen-Notation

- Die sog. **Chen-Notation** der Kardinalität lässt eine '1', ein 'n' oder alternativ '\*' zu.
- Achtung: '1' meint allerdings 0 oder 1, d.h. hier dürfen Elemente auch keine Partner haben, d.h. es darf ein Foto auch keine Person zeigen oder keine Person auf einem Foto abgebildet sein.



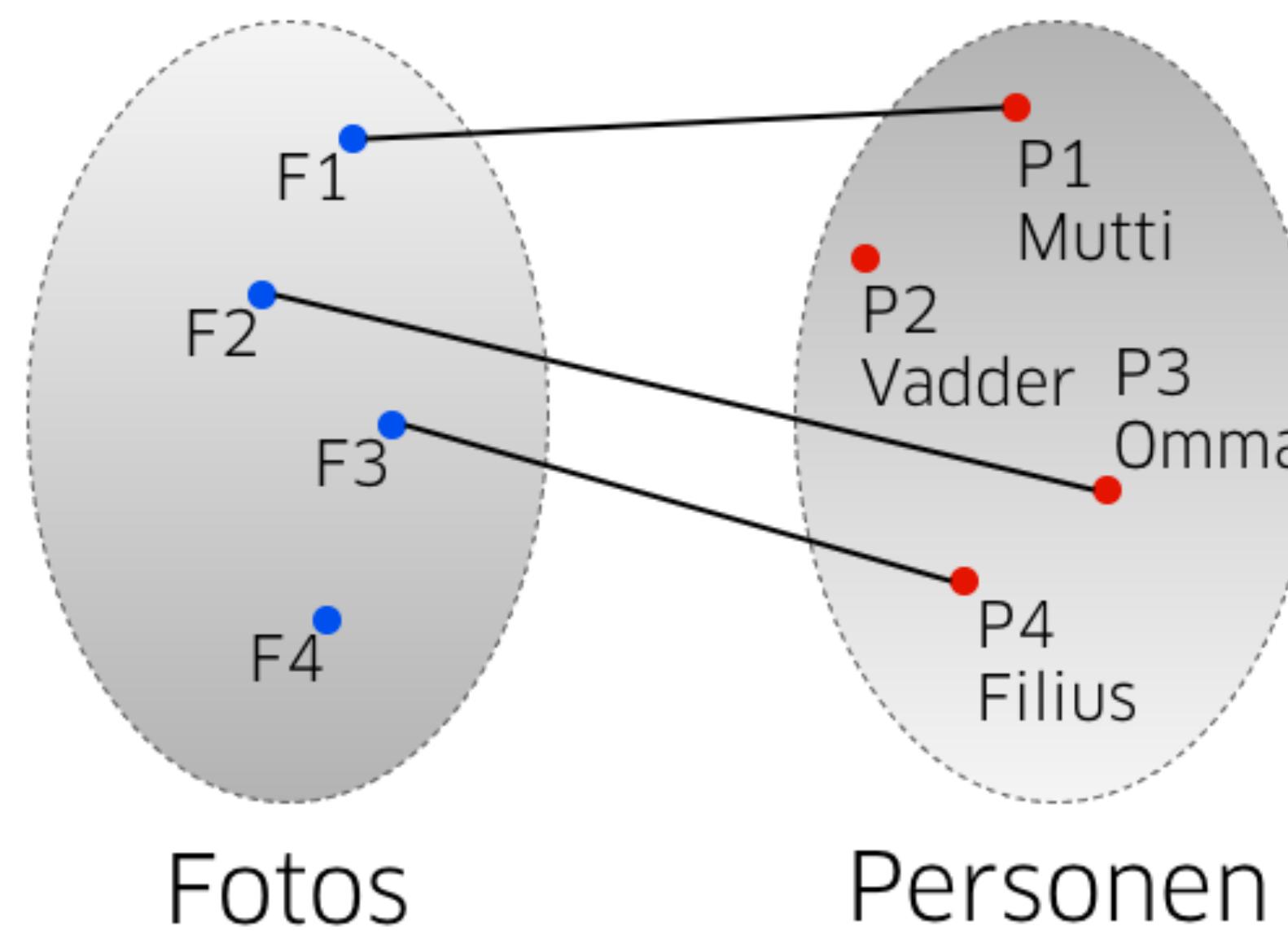
## Relationen und Kardinalität

### 1:1 Relation

Die Mengendiagramme zeigen eine mögliche Zuordnung bei einer 1:1 Relation. Insbesondere gibt es Personen (P2) und Fotos (F4) ohne Partner.

Die Tabellen zeigen eine mögliche Variante (es gibt weitere), die Relation (letzte Tabelle Fid zu Pid) über sog. Fremdschlüssel abzubilden.

Man achte insbes. auf die Anzahl der Paare je Element. Bei einer 1:1 Relation kommt ein Element in max. einer Kombination vor.



Ein Foto zeigt max. eine Person und eine Person ist auf max. einem Foto.

Fid	Datum	Zeit	Pid	Name	Fid	Pid
F1	24.12.18	09:00	P1	Mutti	F1	P1
F2	24.12.18	09:45	P2	Vadder	F2	P3
F3	25.12.18	00:05	P3	Omma	F3	P4
F4	25.12.18	04:15	P4	Filius		

## Relationen und Kardinalität

### Kardinalität

Die Bestimmung der Chen-Kardinalität kann man sich wie eine Abbildung der einen Seite (Foto) auf die andere Seite (Person) vorstellen:

- Wählen Sie ein Foto aus (wie in der Analysis, wählen Sie ein beliebiges aber festes x) und bestimmen Sie, wieviele Personen diesem Foto max. zugeordnet werden können (hier 1)  
→ Kardinalität für Personen.
- Achtung: Foto festhalten links der Relation ergibt Kardinalität auf der rechten Seite.
- Die Kardinalität auf der linken Seite ergibt sich analog.

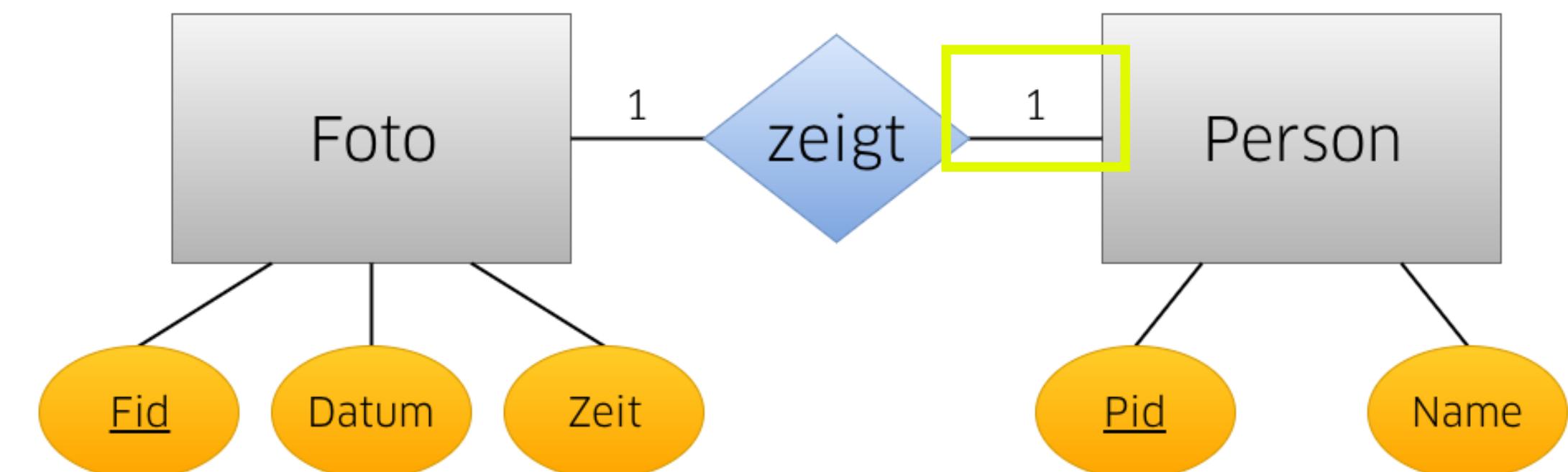


Foto festhalten

ergibt Kardinalität gegenüber

## Relationen und Kardinalität

---

### Mini-Welt Fotoshooting II

**Hintergrund:** Auf einer Familienfeier [...]

Am zweiten Weihnachtstag gilt jedoch: Ein Foto zeigt beliebig viele Personen, aber eine Person ist weiter nur auf max. einem Foto zu sehen.

#### Q&A

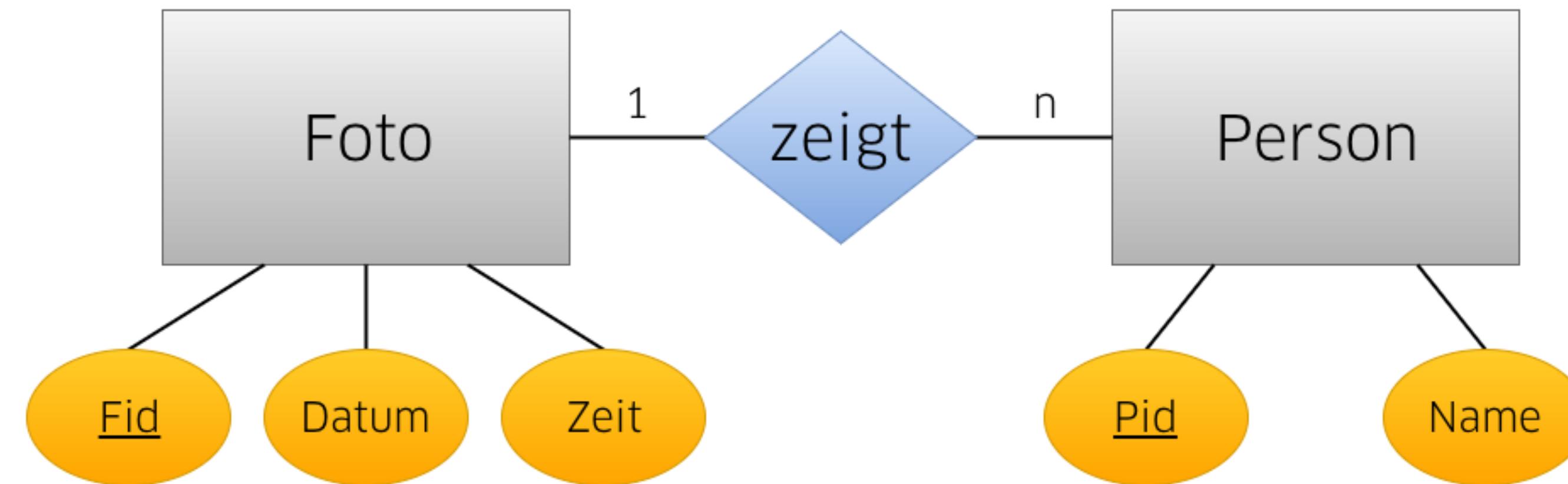
- Kardinalitäten?
- Mengensituation?
- Tabellen?

## Relationen und Kardinalität

---

### 1:n Relation

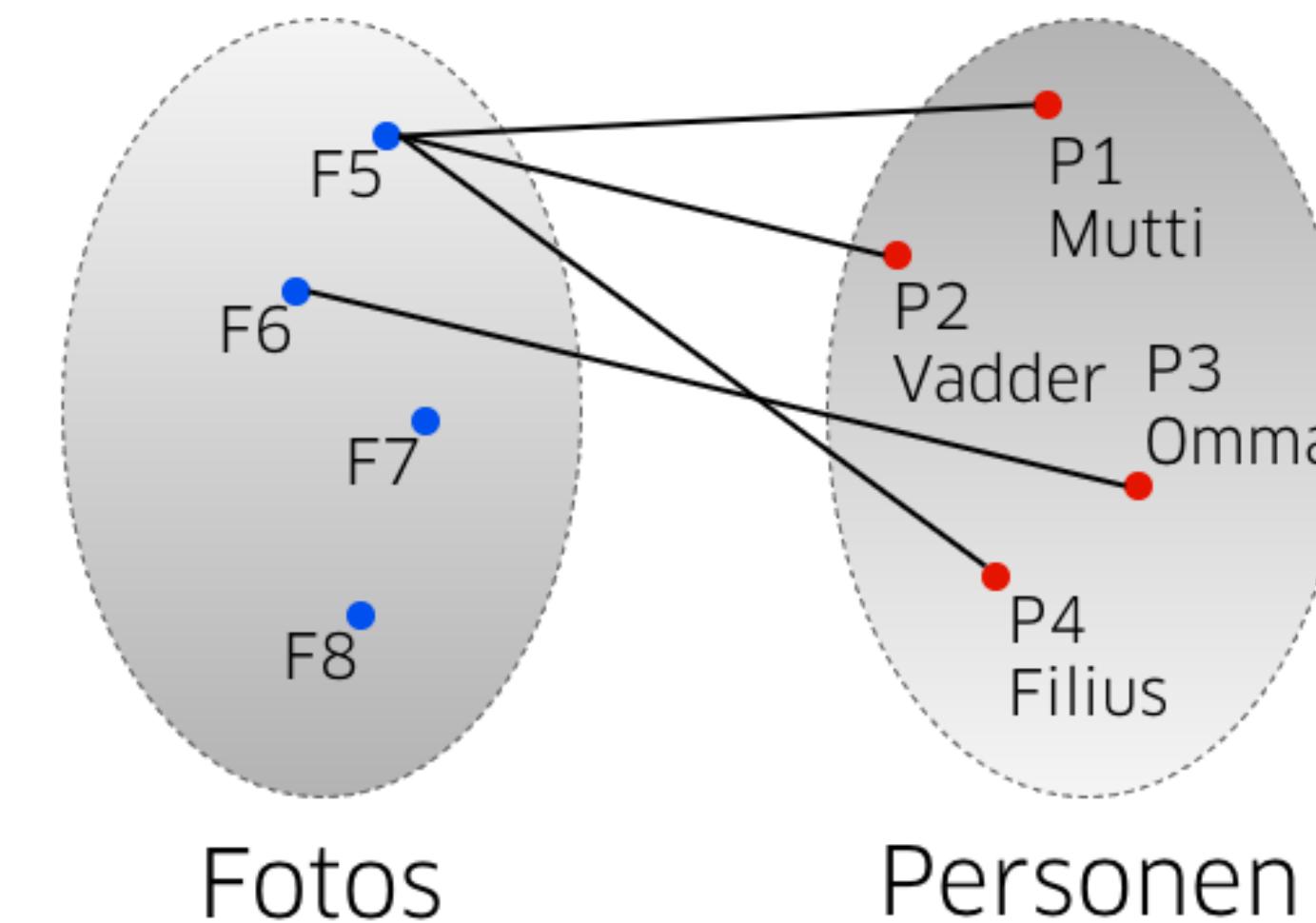
- Ein (fest gewähltes) Foto zeigt mehrere ('n' oder '\*') Personen.
- ← Eine (fest gewählte) Person ist nur auf einem ('1') Foto zu sehen.



## Relationen und Kardinalität

### 1:n Relation

- **Achtung:** Ein Foto, was beliebig viele Personen zeigt kommt natürlich beliebig häufig in der Relationen-Tabelle vor (z.B. F5). Umgekehrt kommt eine Person hier nur max. einmal vor. Diese Sichtweise ist genau umgekehrt zur Notation.



Ein Foto zeigt beliebig viele Personen aber eine Person ist auf max. einem Foto zu sehen.

Fid	Datum	Zeit	Pid	Name	Fid	Pid
F5	26.12.18	15:00	P1	Mutti	F5	P1
F6	26.12.18	15:05	P2	Vadder	F5	P2
F7	26.12.18	15:10	P3	Omma	F5	P4
F8	26.12.18	15:15	P4	Filius	F6	P3

## Relationen und Kardinalität

---

### Mini-Welt Fotoshooting III

**Hintergrund:** Auf einer Familienfeier [...]

Am Sylvester wird die Regel umgedreht. Das bedeutet, ein Foto zeigt max. eine Person, aber diese kann auf beliebig vielen Fotos zu sehen sein.

#### Q&A

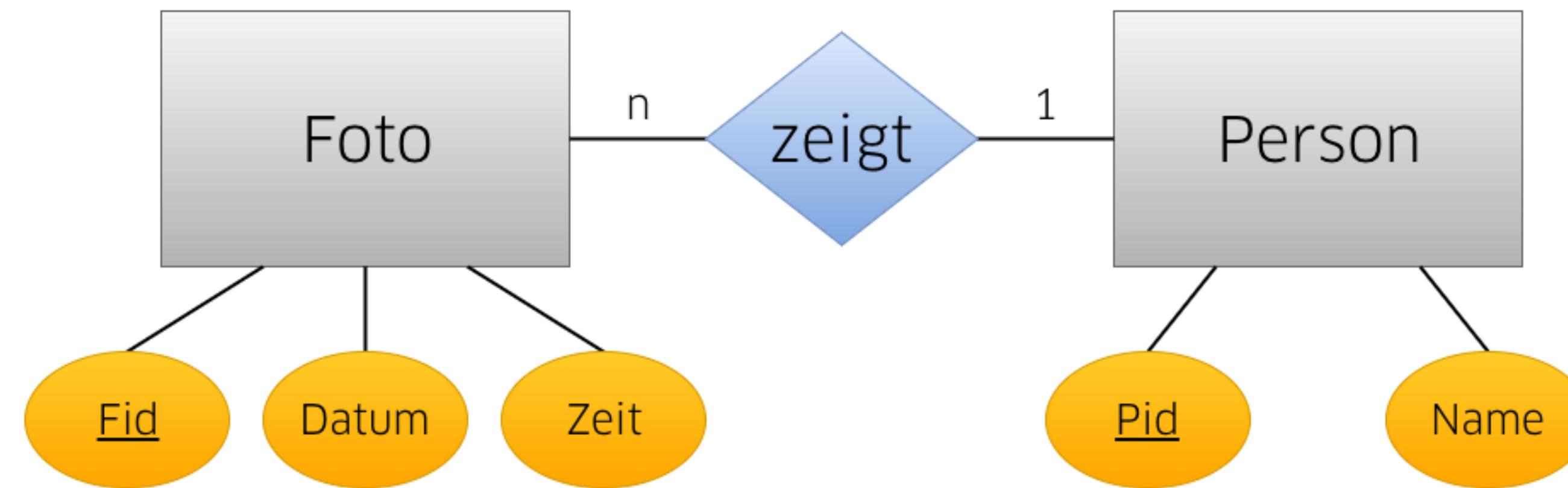
- Kardinalitäten?
- Mengensituation?
- Tabellen?

## Relationen und Kardinalität

---

### n:1 Relation

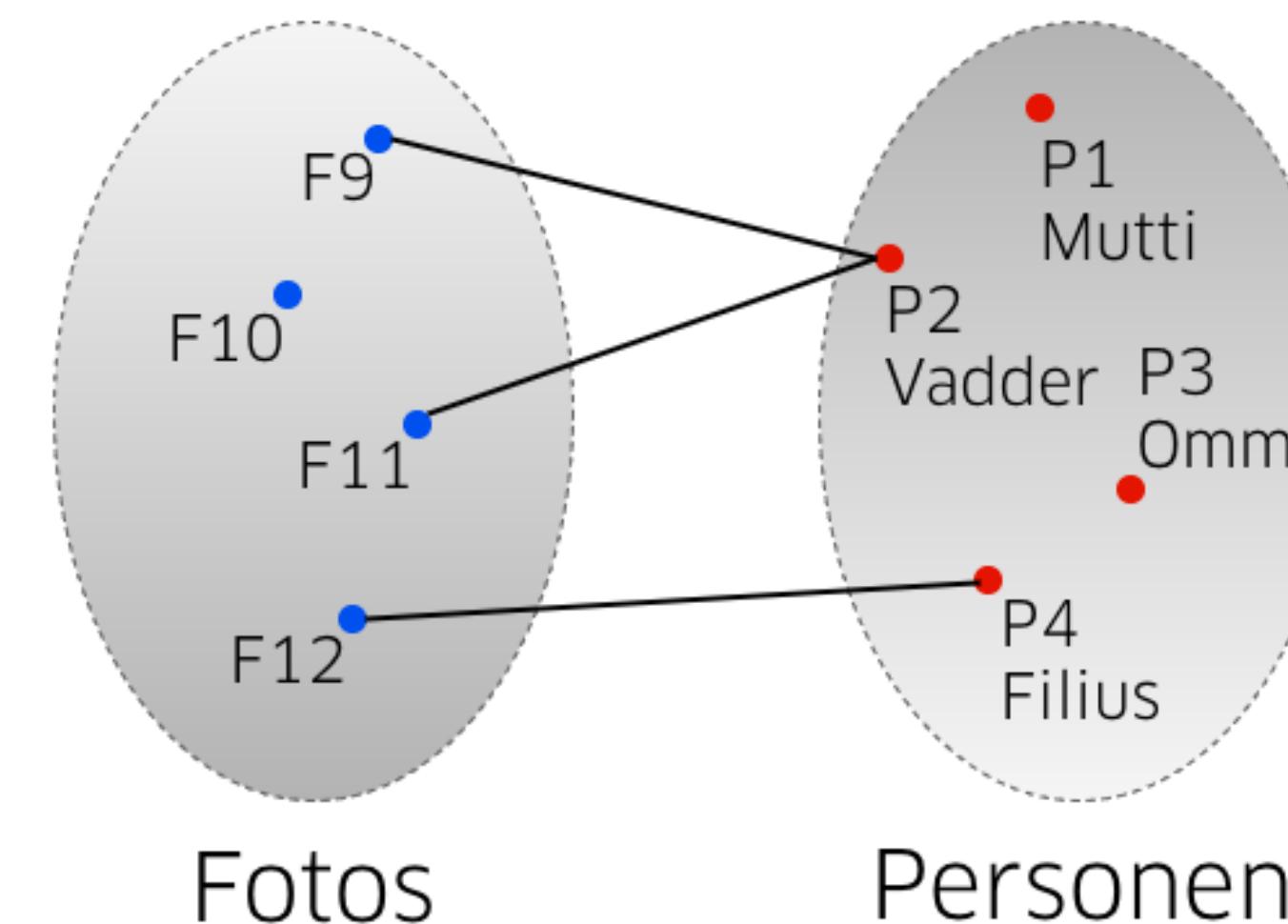
- Ein (fest gewähltes) Foto zeigt genau eine ('1') Person.
- ← Eine (fest gewählte) Person ist auf mehreren ('n' oder '\*') Fotos zu sehen.



## Relationen und Kardinalität

### n:1 Relation

- **Achtung:** Ein Foto kommt in der Relationen-Tabelle max. einmal vor, wogegen die Personen beliebig oft vorkommen (z.B. F2).



Ein Foto zeigt max. eine Person aber diese kann auf beliebig vielen Fotos zu sehen sein.

Fid	Datum	Zeit	Pid	Name	Fid	Pid
F9	31.12.18	23:45	P1	Mutti	F9	P2
F10	31.12.18	23:50	P2	Vadder	F11	P2
F11	31.12.18	23:55	P3	Omma	F12	P4
F12	31.12.18	23:59	P4	Filius		

## Relationen und Kardinalität

---

### Mini-Welt Fotoshooting IV

**Hintergrund:** Auf einer Familienfeier [...]

An Neujahr ist alles erlaubt, d.h. ein Foto zeigt beliebig viele Personen und eine Person kann auf beliebig vielen Fotos zu sehen sein.

#### Q&A

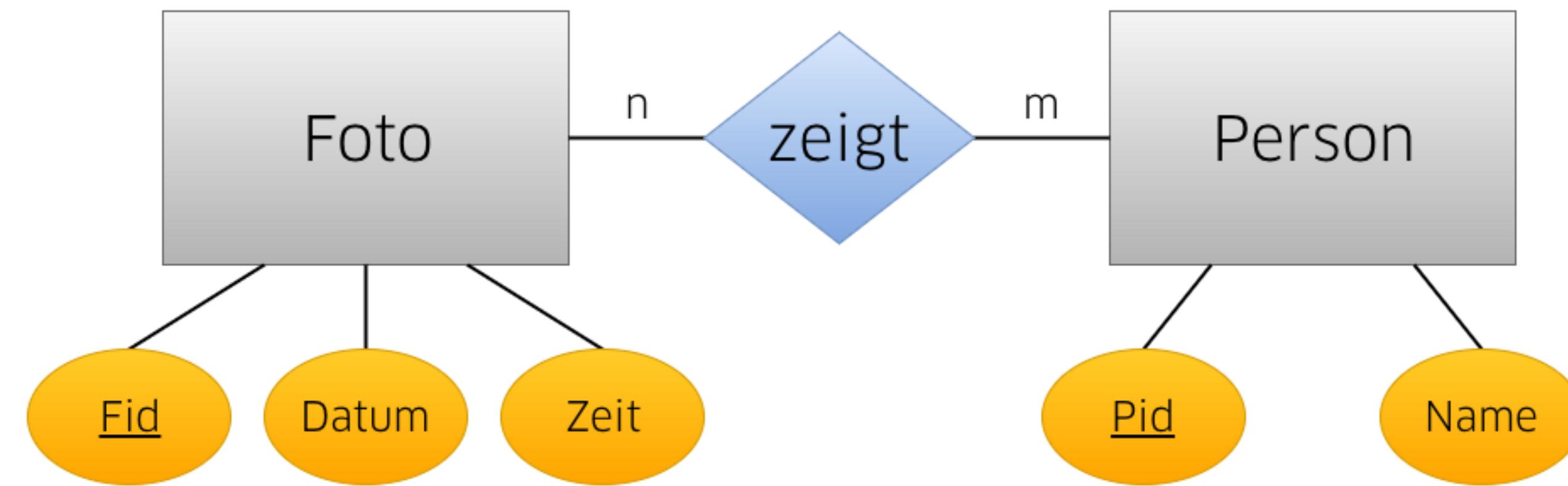
- Kardinalitäten?
- Mengensituation?
- Tabellen?

## Relationen und Kardinalität

---

### n:m Relation

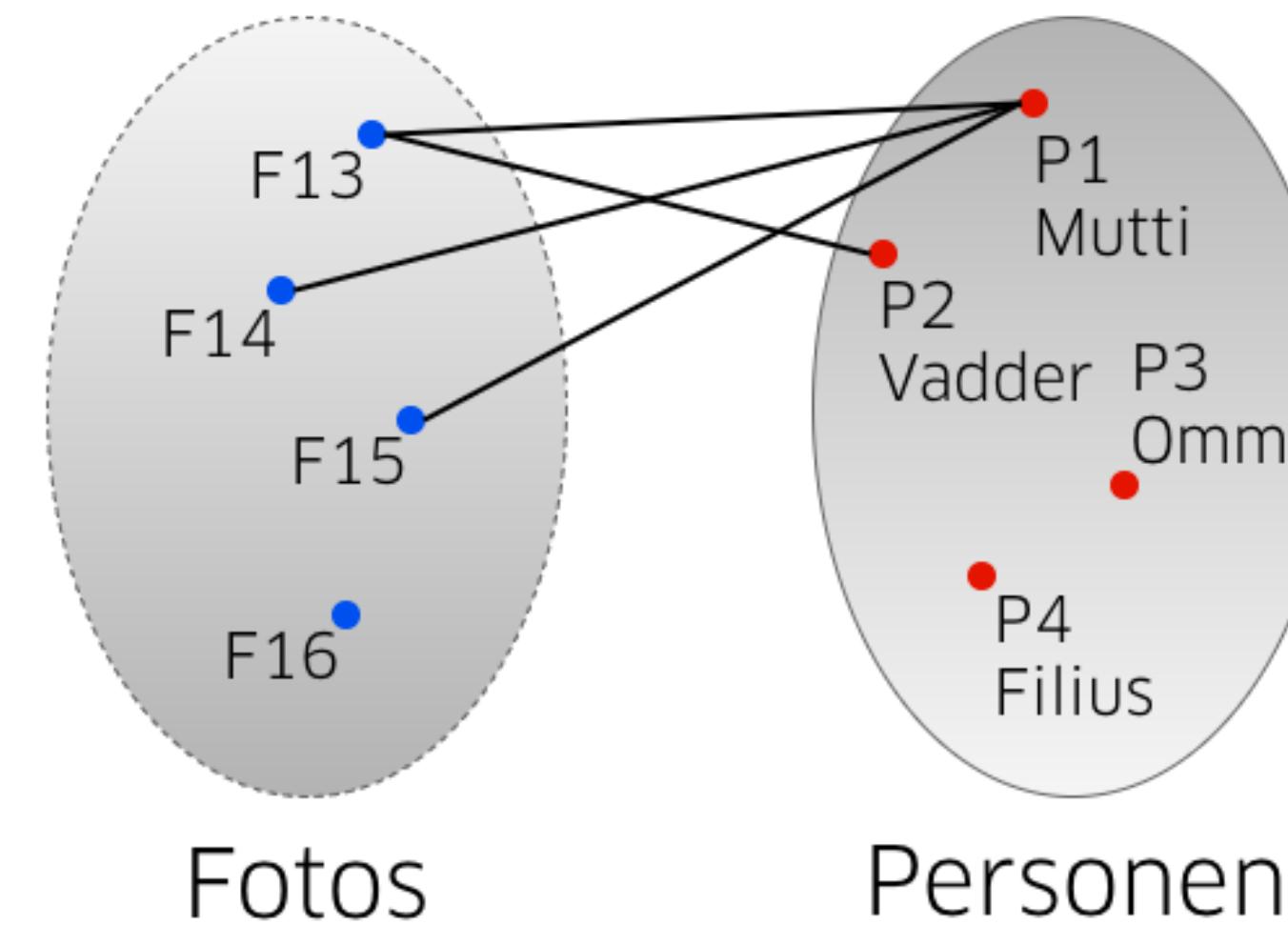
- Ein (fest gewähltes) Foto zeigt beliebig viele ('m' oder '\*') Personen.
- ← Eine (fest gewählte) Person ist auf mehreren ('n' oder '\*') Fotos zu sehen.



## Relationen und Kardinalität

### n:m Relation

- **Achtung:** Fotos und Personen kommen in der Relationen-Tabelle beliebig oft vor (z.B. F13, P1).



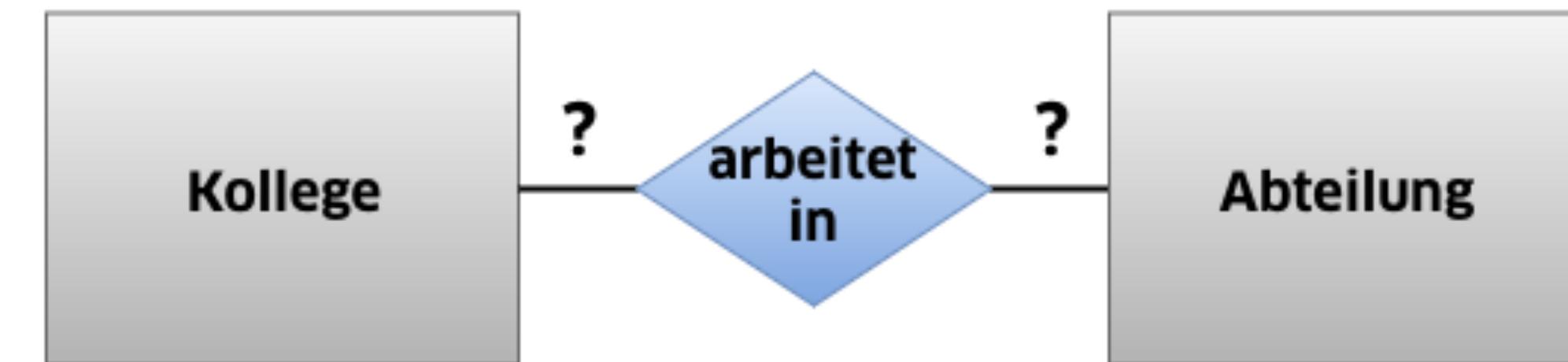
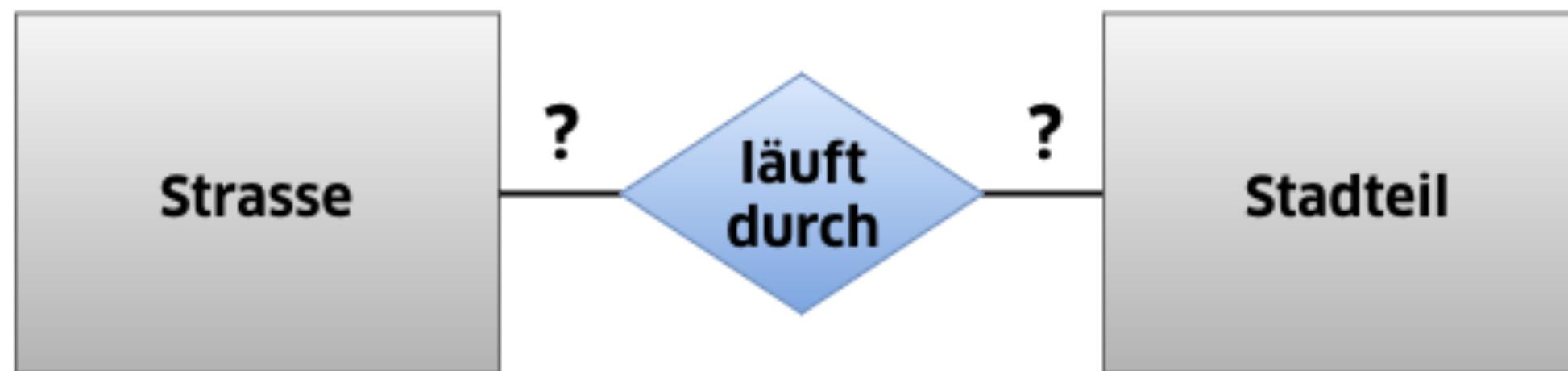
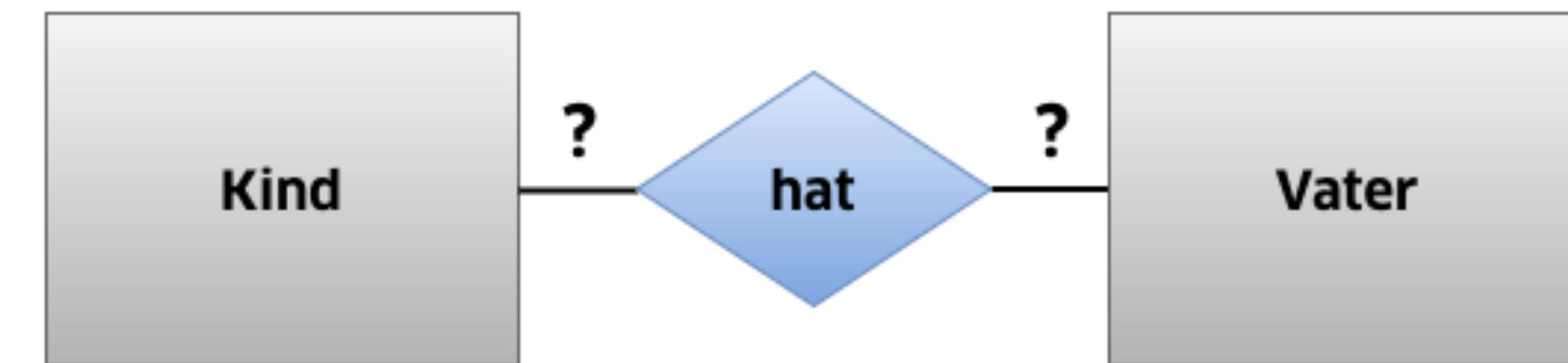
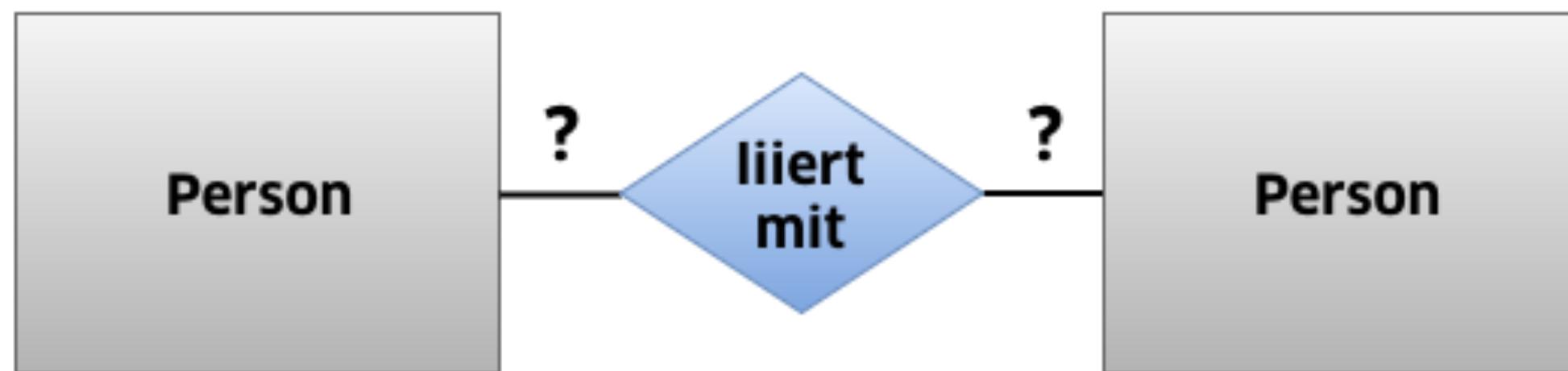
Ein Foto zeigt beliebig viele Personen und eine Person kann auf beliebig vielen Fotos zu sehen sein.

Fid	Datum	Zeit	Pid	Name	Fid	Pid
F13	01.01.18	13:01	P1	Mutti	F13	P1
F14	01.01.18	13:02	P2	Vadder	F13	P2
F15	01.01.18	13:03	P3	Omma	F14	P1
F16	01.01.18	13:04	P4	Filius	F15	P1

## Relationen und Kardinalität

---

### Check Chen-Notation



## Relationen und Kardinalität

---

### Pro/Contra Chen-Notation

#### Pro

- Einfache Darstellung.
- 1:1, 1:n, n:1, n:m reicht oft für die Modellierung aus.

#### Contra

- Ungenau, insbesonders ist oftmals interessant, ob 0 erlaubt ist.

#### Q&A

- Wie ginge es besser?

## Zusammenfassung

---

### Was ist wichtig?

- Ablauf einer Modellierung, insbes. Trennung der Ebenen (ANSI/SPARC-Architektur).
- Rolle der ER-Diagramme und Abgrenzung zur Abbildung in die Datenbankstrukturen.
- ER-Diagramme, Symbolik, Chen-Notation.

### Was folgt?

- Fortsetzung ER-Diagramme mit Angabe der Kardinalität in der Min-Max-Notation und der UML-Notation.
- Implementationsentwurf und 'Relationales Modell'.

UNIT 0x04

ENTITY-RELATIONSHIP-MODELL II

# Übersicht

---

## Themen

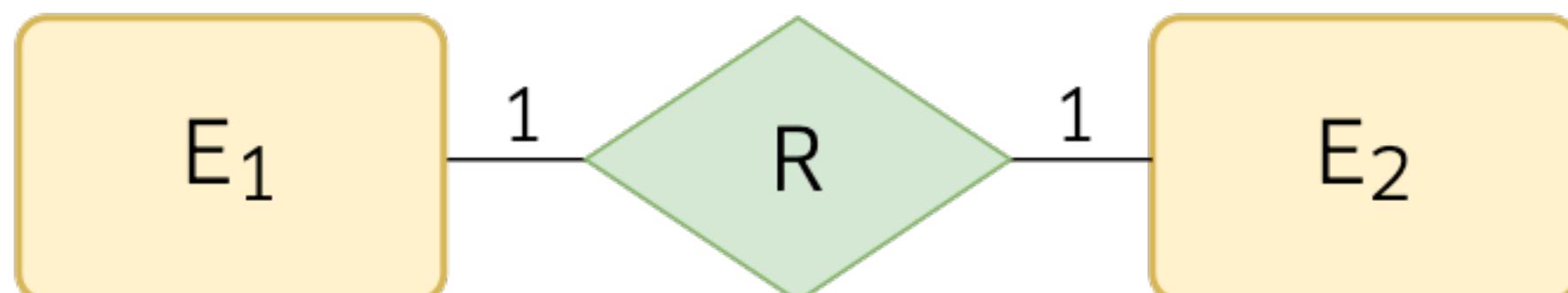
- Wiederholung der Elemente im ER-Diagramm
- Relationen
  - n-stellige Relationen
  - Existenzabhängige Entitätstypen
- Kardinalitäten
  - Min-Max-Notation
  - UML-Notation
- Generalisierung, Spezialisierung
  - Objekthierarchien

## Wiederholung

---

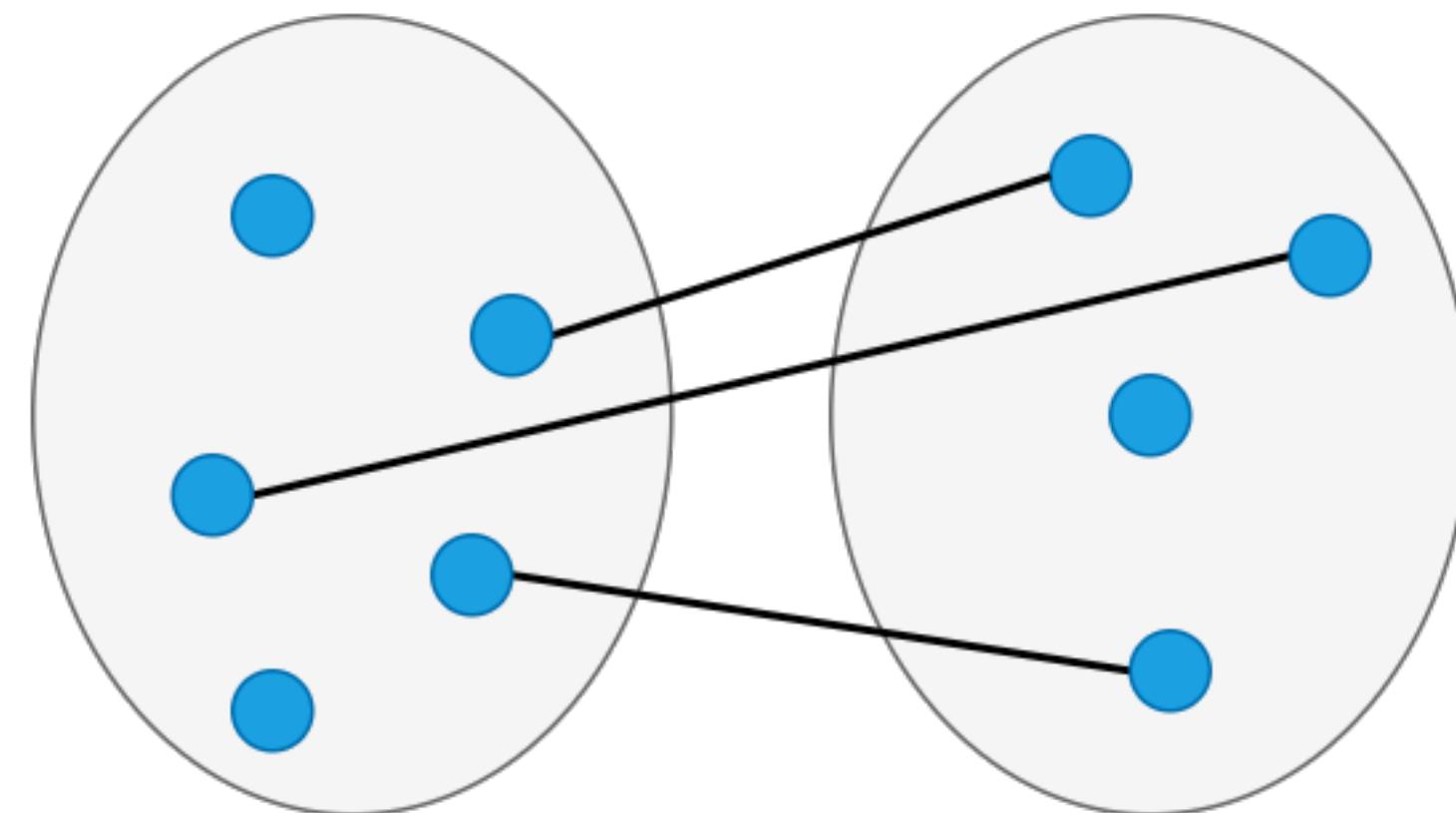
### 1:1

- Achtung: 0 oder 1, d.h. es existieren Entitäten ohne Partner.



### Beispiel

- Mini-Welt: Person hält genau ein Haustier und Haustier hat genau eine/n Besitzer/in.
- Person ( $E_1$ ) verantwortlich\_für Haustier ( $E_2$ ).
- (Max,Wuff)  $\in R$

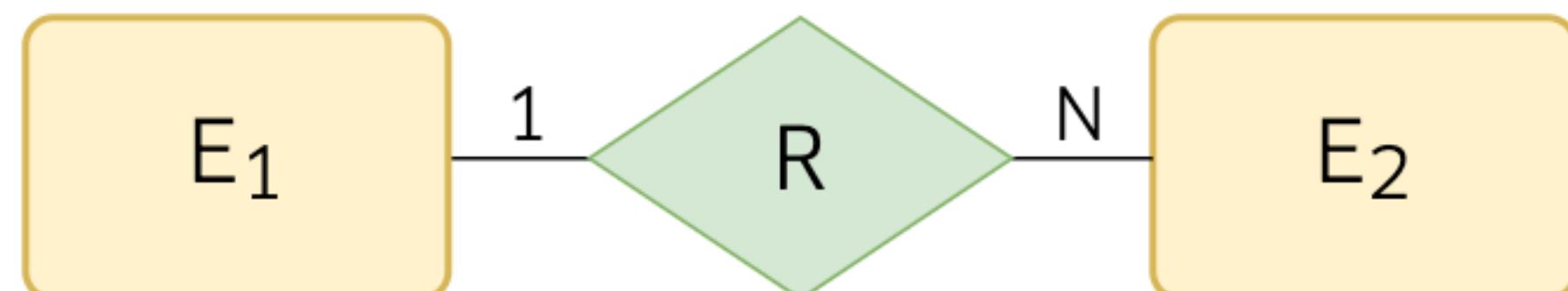


## Wiederholung

---

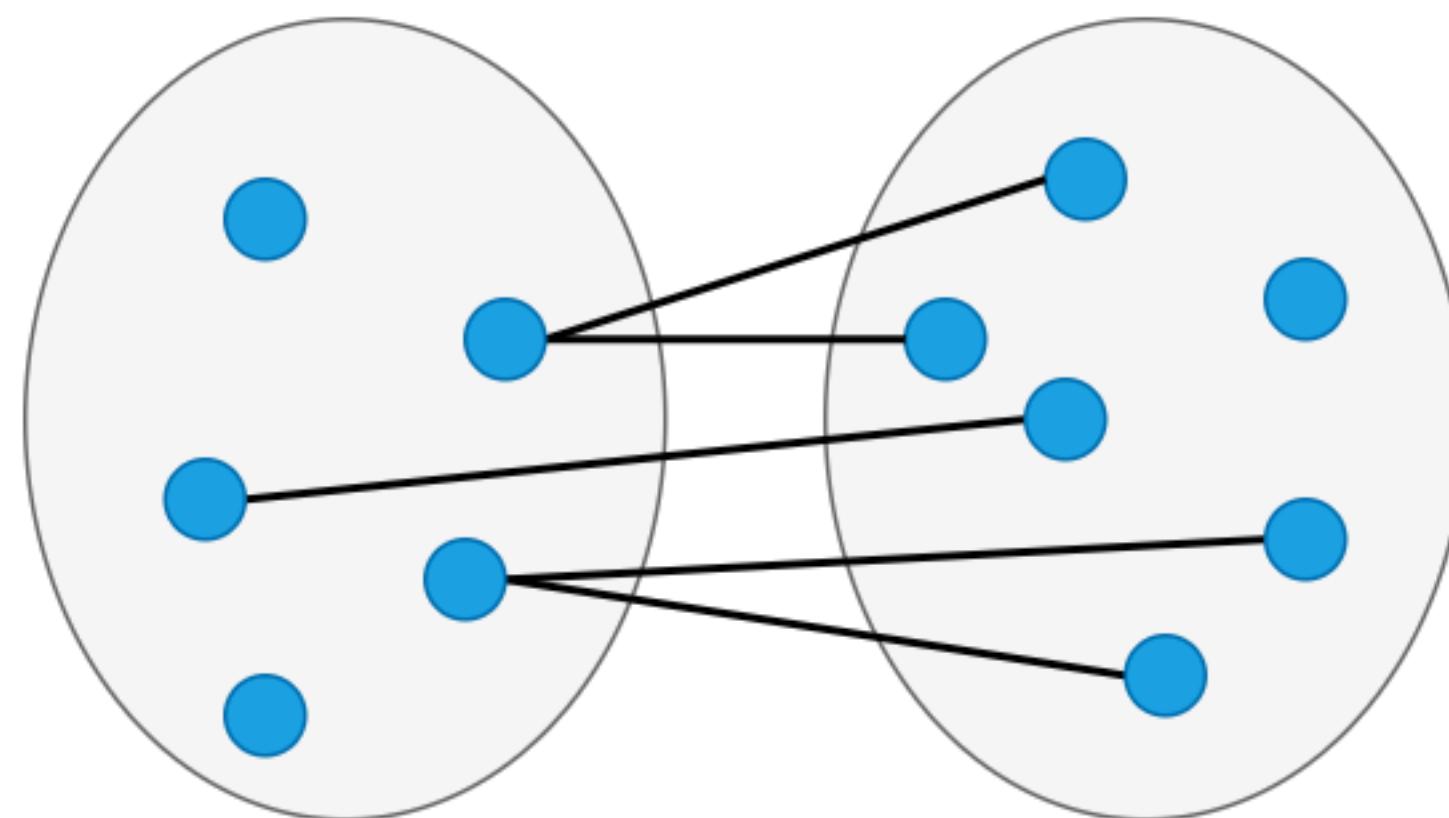
### 1:N und N:1

- Achtung: 0 oder 1, d.h. es existieren Entitäten ohne Partner.
- Für N:1 einfach die Rollen tauschen.
- Statt 'N' bzw. 'n' auch '\*'.



### Beispiel

- Mini-Welt: Abteilung sind Mitarbeiter zugeordnet und ein/e Mitarbeiter/in arbeitet in genau einer Abteilung.
- Abteilung ( $E_1$ ) beschäftigt Mitarbeiter ( $E_2$ ).
- (Marketing,Max)  $\in R$

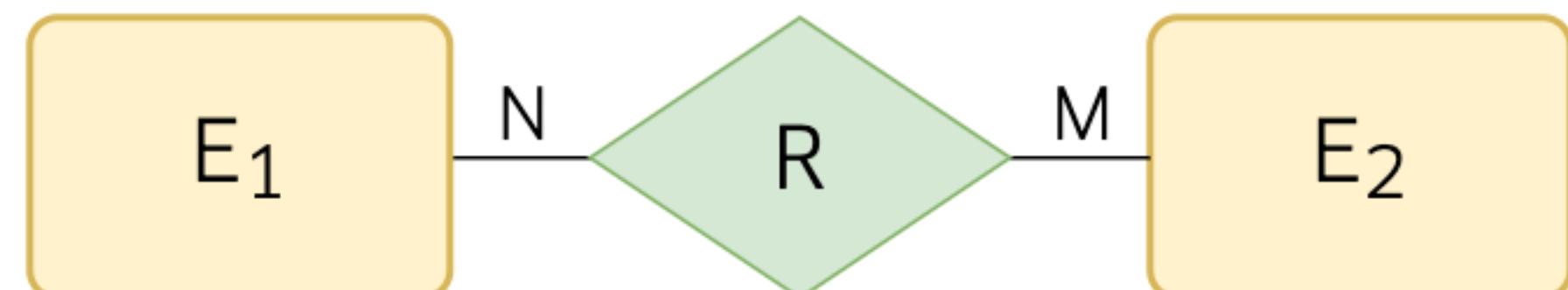


## Wiederholung

---

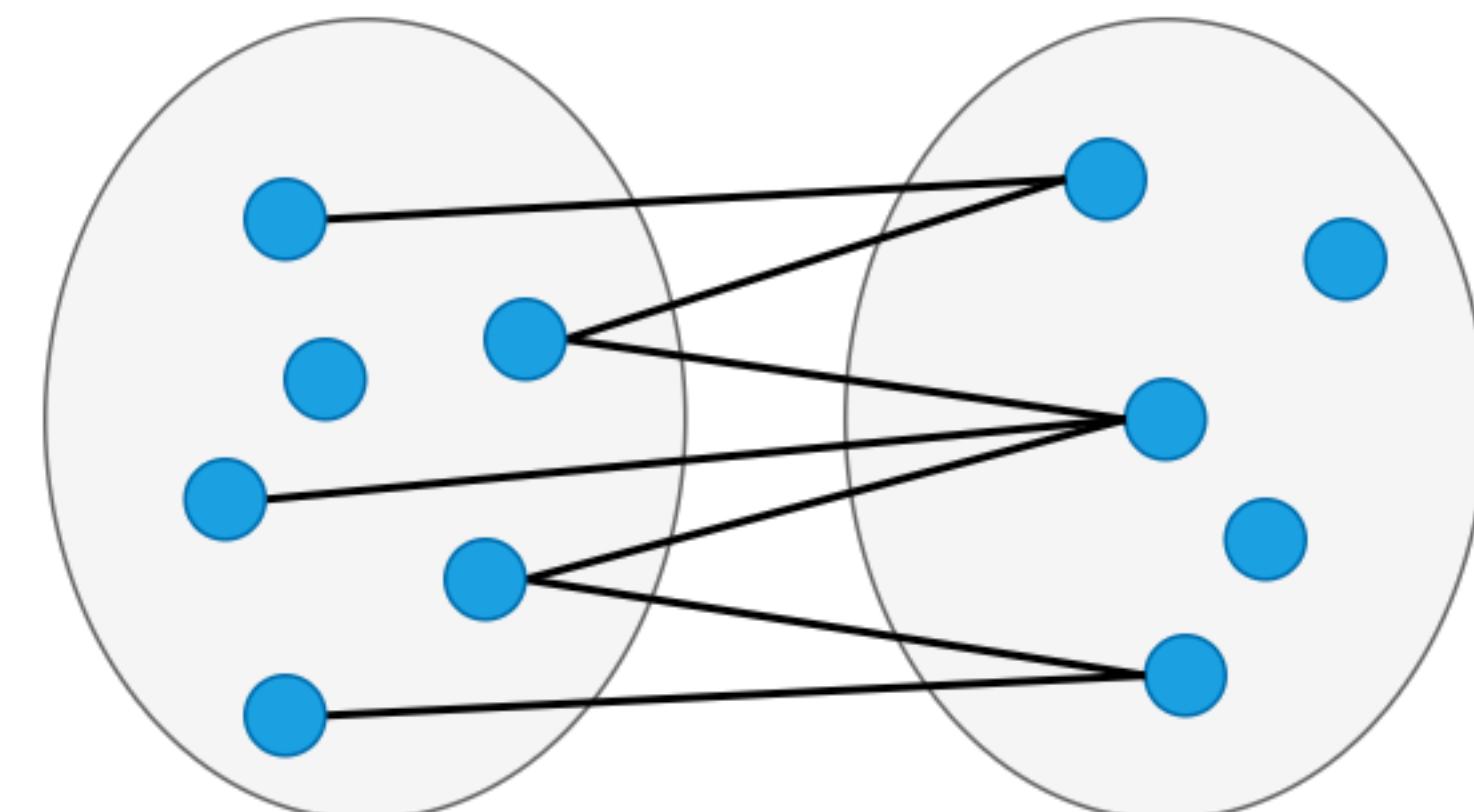
### N:M

- Achtung: 0 oder 1, d.h. es existieren Entitäten ohne Partner.
- Statt 'N' bzw. 'n' auch '\*'.



### Beispiel

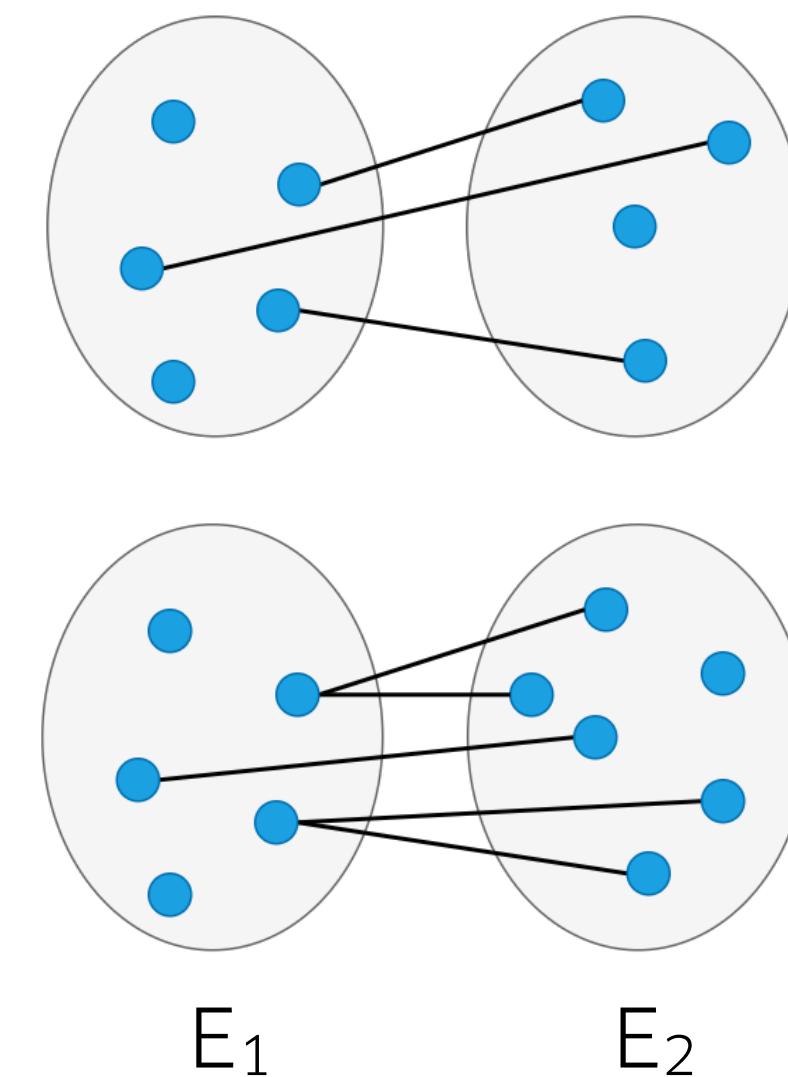
- Mini-Welt: Vorlesungen werden von Studierenden gehört.
- Studierende ( $E_1$ ) besuchen Vorlesungen ( $E_2$ ).
- $(\text{Max}, \text{Analysis 1}) \in R$



## Relationen

### Partielle und totale Funktion

- Analog zur Funktion  $g: \mathbb{R} - \{0\} \rightarrow \mathbb{R}, x \mapsto 1/x$  mit Definitionslücke  $\{0\}$  ist man an einem Funktionsbegriff interessiert, der Ausnahmen zulässt. Man nennt  $f: X - L \rightarrow Y$  und  $L \subseteq X$ 
  - **eine partielle Funktion**,  $f: X \rightarrow Y$ , falls  $L \neq \emptyset$ , oder klassisch
  - **eine totale Funktion**,  $f: X \rightarrow Y$ , falls  $L = \emptyset$ .
- So definiert eine 1:1-Relation implizit zwei partielle Funktionen  $f_1: E_1 \rightarrow E_2$  und  $f_2: E_2 \rightarrow E_1$ .
- Ebenso impliziert eine 1:N-Relation zumindest noch eine, nicht notwendigerweise injektive, partielle Funktion  $f_2: E_2 \rightarrow E_1$ . Die Abbildung von  $E_1$  nach  $E_2$  ist keine klassische Funktion mehr, da wir keine eindeutige Zuordnung haben.
- Letzteres gilt analog für N:M-Relation in beiden Richtungen.

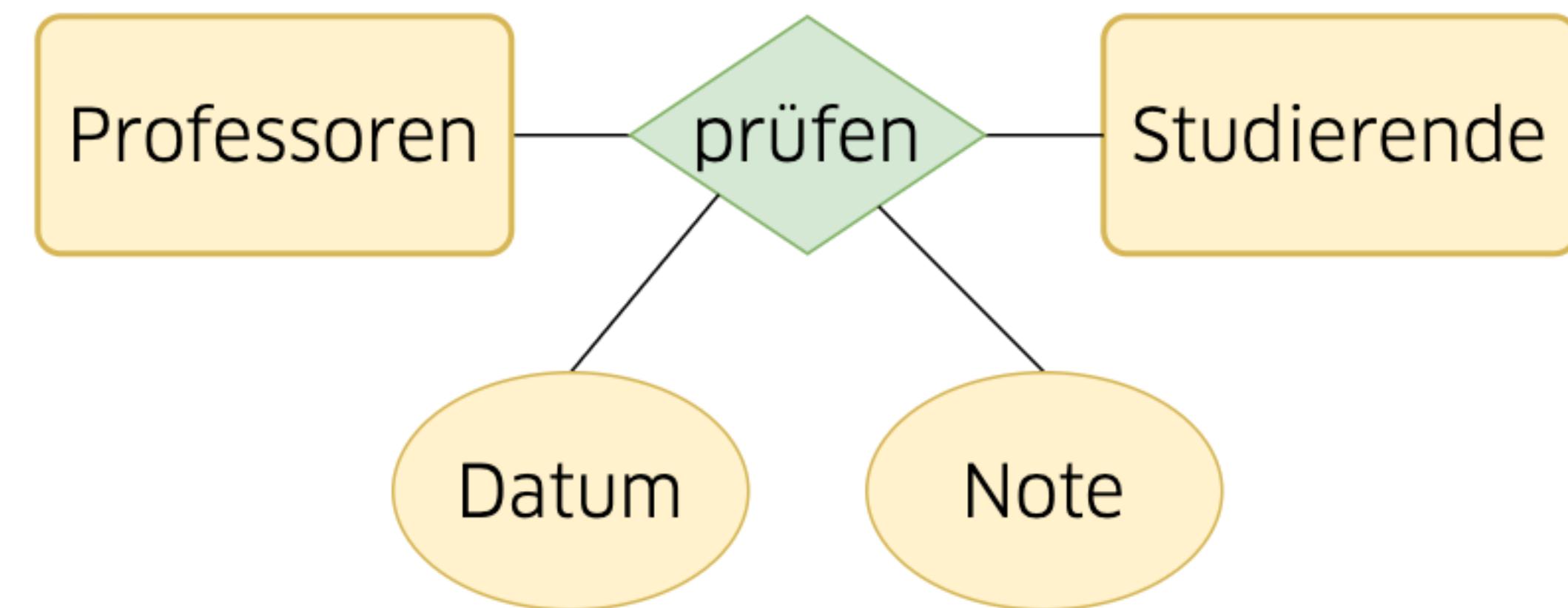


## Relationen

### Relationen mit Attributen

- Sei  $A=[A_1:D_1, \dots A_m:D_m]$  ein Tupel mit  $m$  Attributen  $A_1, \dots, A_m$  und  $E_1, \dots, E_n$  eine Menge von Entitätstypen. Dann ist  $R \subseteq E_1 \times \dots \times E_n \times D_1 \times \dots \times D_m$  eine mit  $A$  attributierte Relation.

### Beispiel $n=2, m=2$



- $\text{prüfen } (R) \subseteq \text{Professoren } (E_1) \times \text{Studierende } (E_2) \times \text{Datum } (D_1) \times \text{Note } (D_2)$

### Q&A

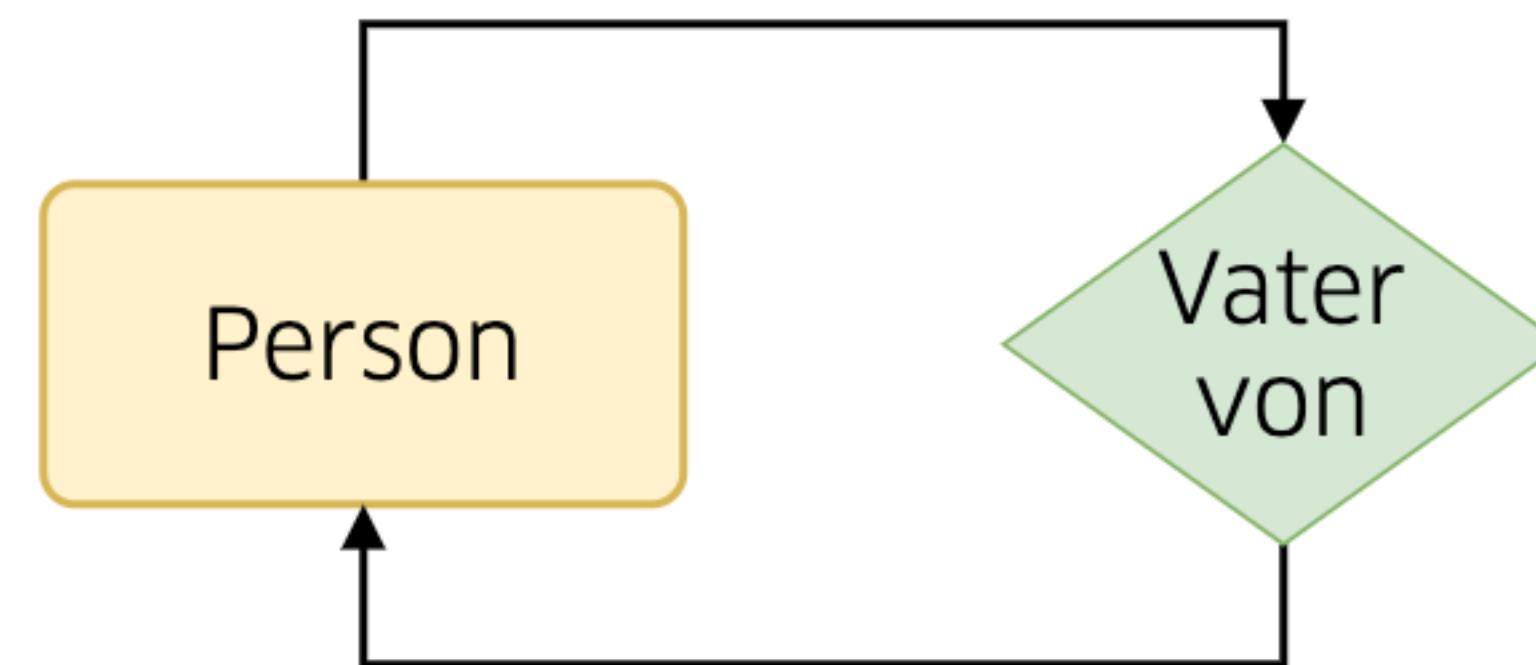
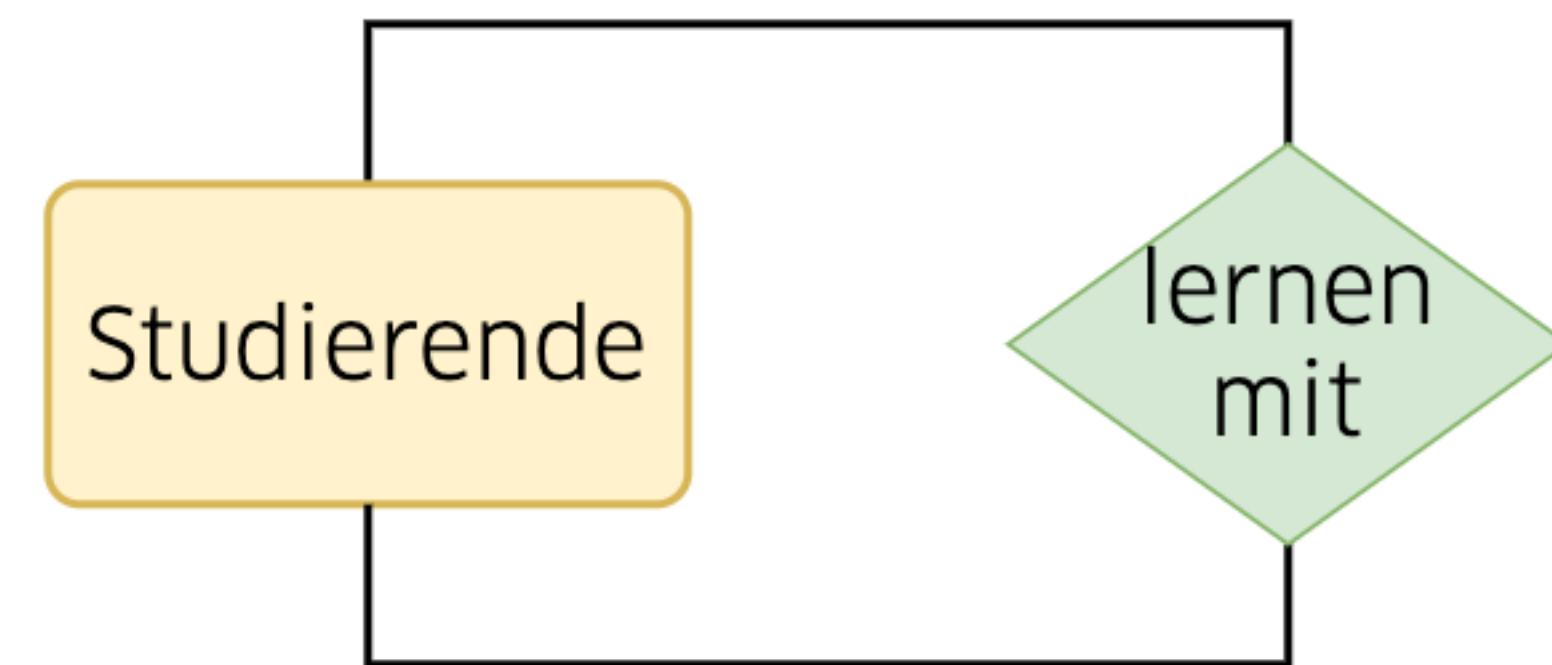
- Gehören Attribute an die Entitätstypen oder an die Relation?

## Relationen

---

### 1:1-, 1:N-, N:1-, N:M-Relationen zu sich selbst

- Auch wenn wir bislang allgemein von Mengen  $E_k$  in den Definitionen und Beispielen gesprochen haben, können diese natürlich identisch sein. Das bedeutet, dass eine Entitätsmenge zu sich selber in Relation stehen kann.
- Wenn es notwendig ist, können Pfeile die Beziehung verdeutlichen.

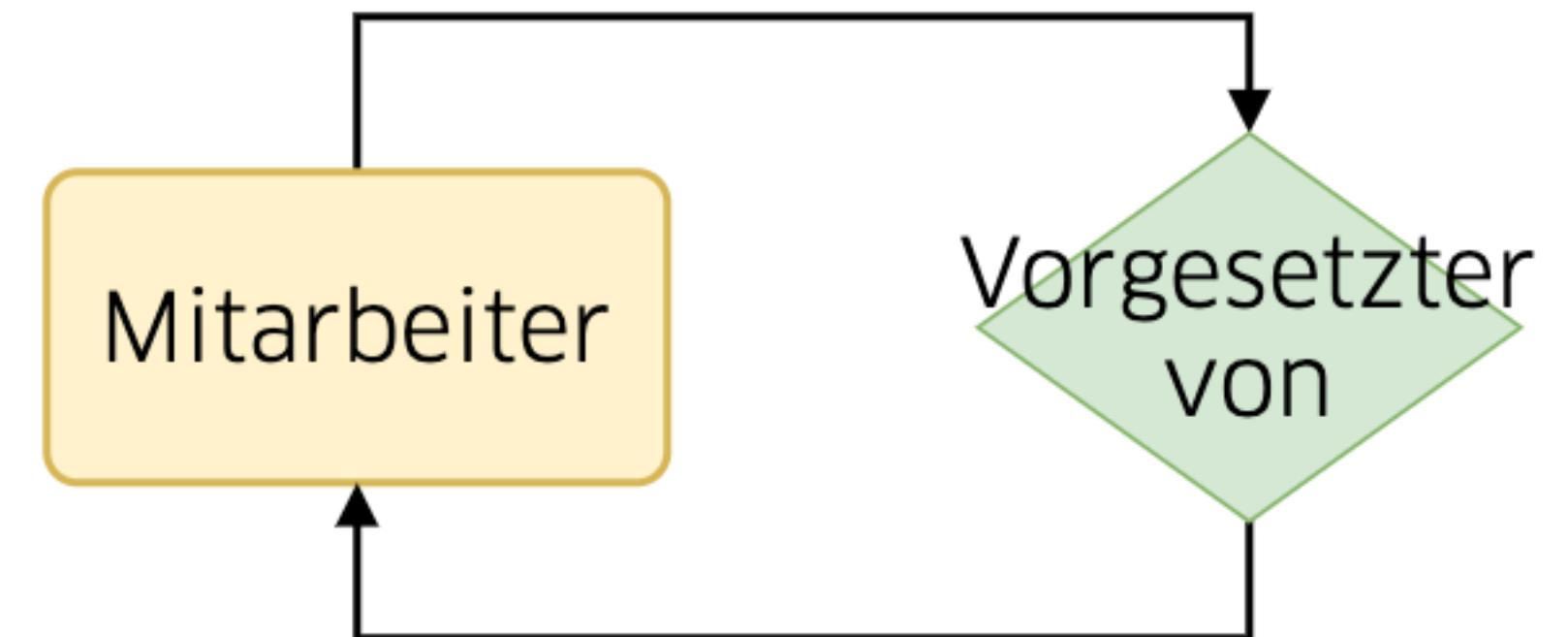


## Relationen

---

### 1:1-, 1:N-, N:1-, N:M-Relationen zu sich selbst

- Beispiel matse\_mhist
  - Relation vorgesetzter\_von zwischen mitarbeiter und mitarbeiter
  - aber in der Tabelle umgekehrt abgebildet über vorgesetzter\_mitarbeiter\_id



	id	name	jahresgehalt	vorgesetzter_mitarbeiter_id
1	1	Mia	110000.00	26
2	2	Ben	90000.00	26
3	3	Emma	70000.00	12
4	4	Paul	5000.00	27
5	5	Hannah	5000.00	27
6	6	Luka	5000.00	27
7	7	Sofia	50000.00	1
8	8	Jonas	50000.00	1
9	9	Anna	50000.00	2
10	10	Finn	50000.00	2

Beispiel: Mia (id 1) ist Vorgesetzte von Jonas (id 7)

**Q&A**

- Warum so?

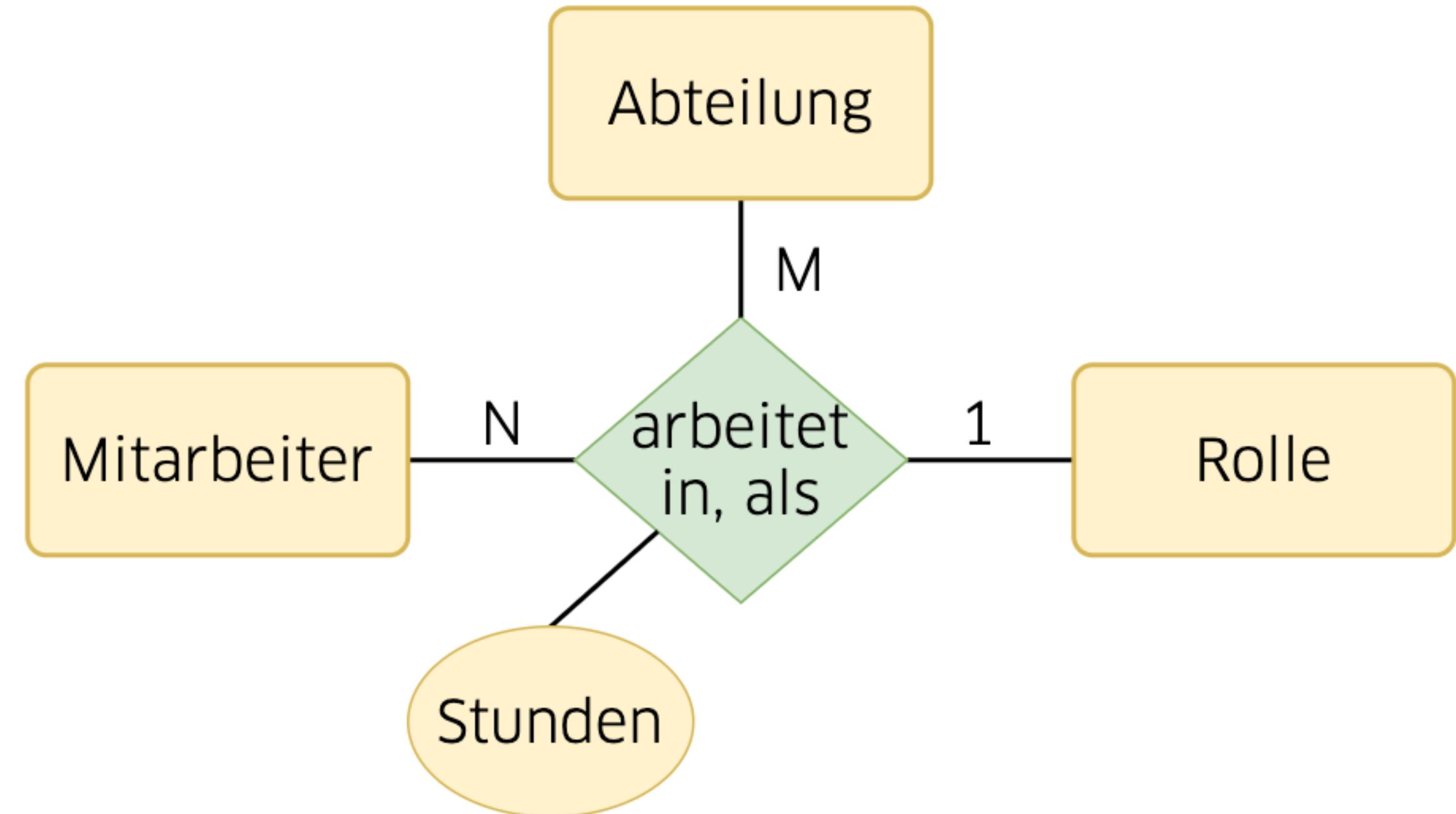
## Relationen

### n-stellige Relationen

- Es ist bekanntlich möglich, auch mehr als zwei Entitätstypen in Beziehung zu setzen, denn  $R \subseteq E_1 \times \dots \times E_n$ .

#### Q&A

- Wie kommt man hier auf die Chen-Kardinalitäten?
- Wie sehen die Tabellen aus?
- Ist die Situation äquivalent zu drei Relationen?

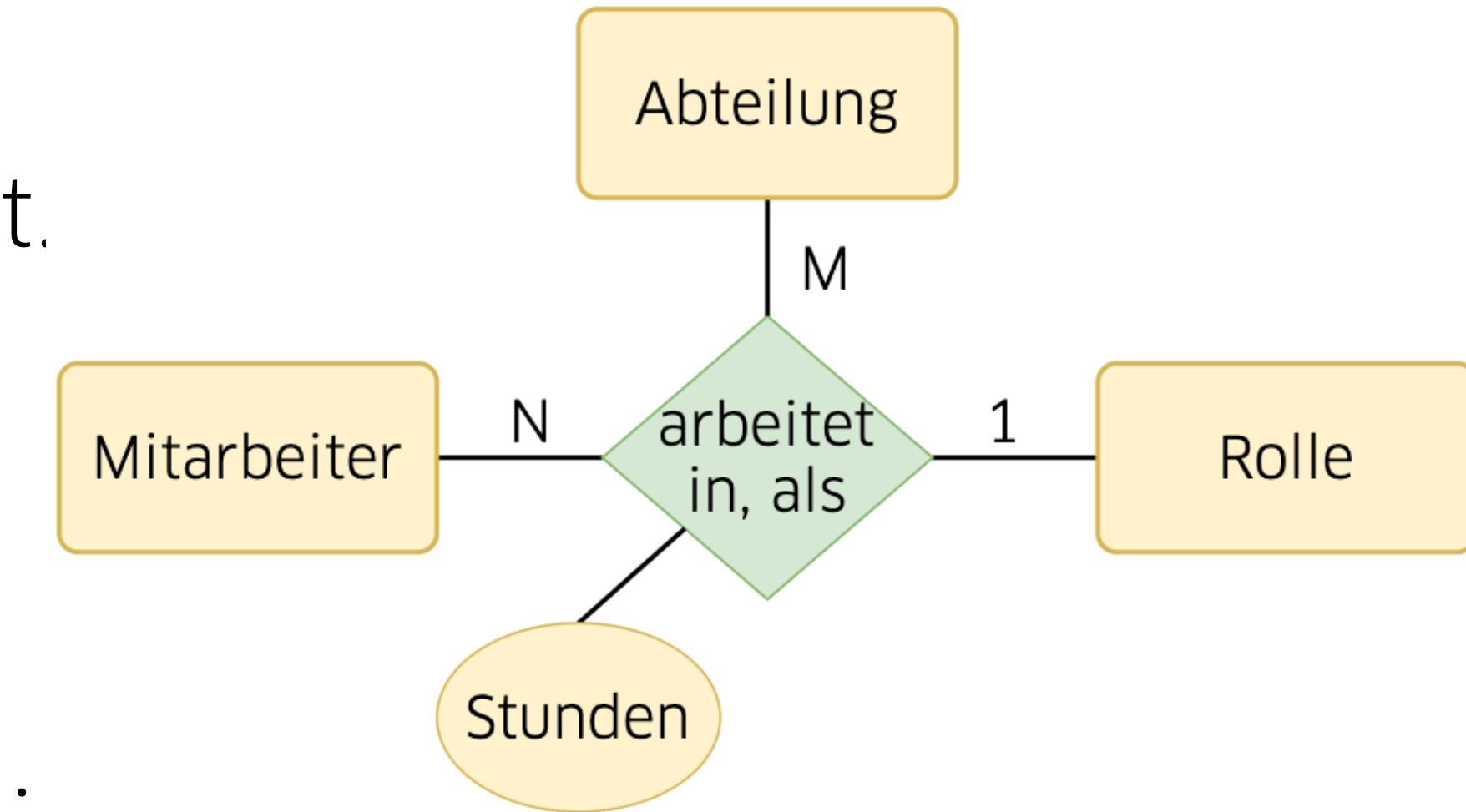


Beispiel: arbeitet\_in\_also aus matse\_mhist

## Relationen

### n-stellige Relationen - Chen-Notationen

- Zunächst ein Blick auf die Anforderungen/Mini-Welt.  
Hier könnte stehen, dass ein Mitarbeiter in einer Abteilung immer nur in einer Rolle arbeitet.
- Wie zuvor (siehe Chen-Notation) halten wir n-1 von n Entitäten fest (beliebig, aber fest) und notieren die Kardinalität 'gegenüber', d.h. hier bspw.:
  - Max (Mitarbeiter) arbeitet in der Forschung (Abteilung) nur als Berater (Rolle), d.h. Mitarbeiter fest, Abteilung fest (s.o.) → 1 Rolle.
  - In der Forschung (Abteilung) arbeiten als Berater (Rolle) Max und Mia, d.h. Abteilung fest, Rolle fest → N Mitarbeiter.
  - Mia (Mitarbeiter) kann als Berater (Rolle) in Forschung und Marketing helfen, d.h. Mitarbeiter fest, Rolle fest → M Abteilungen.

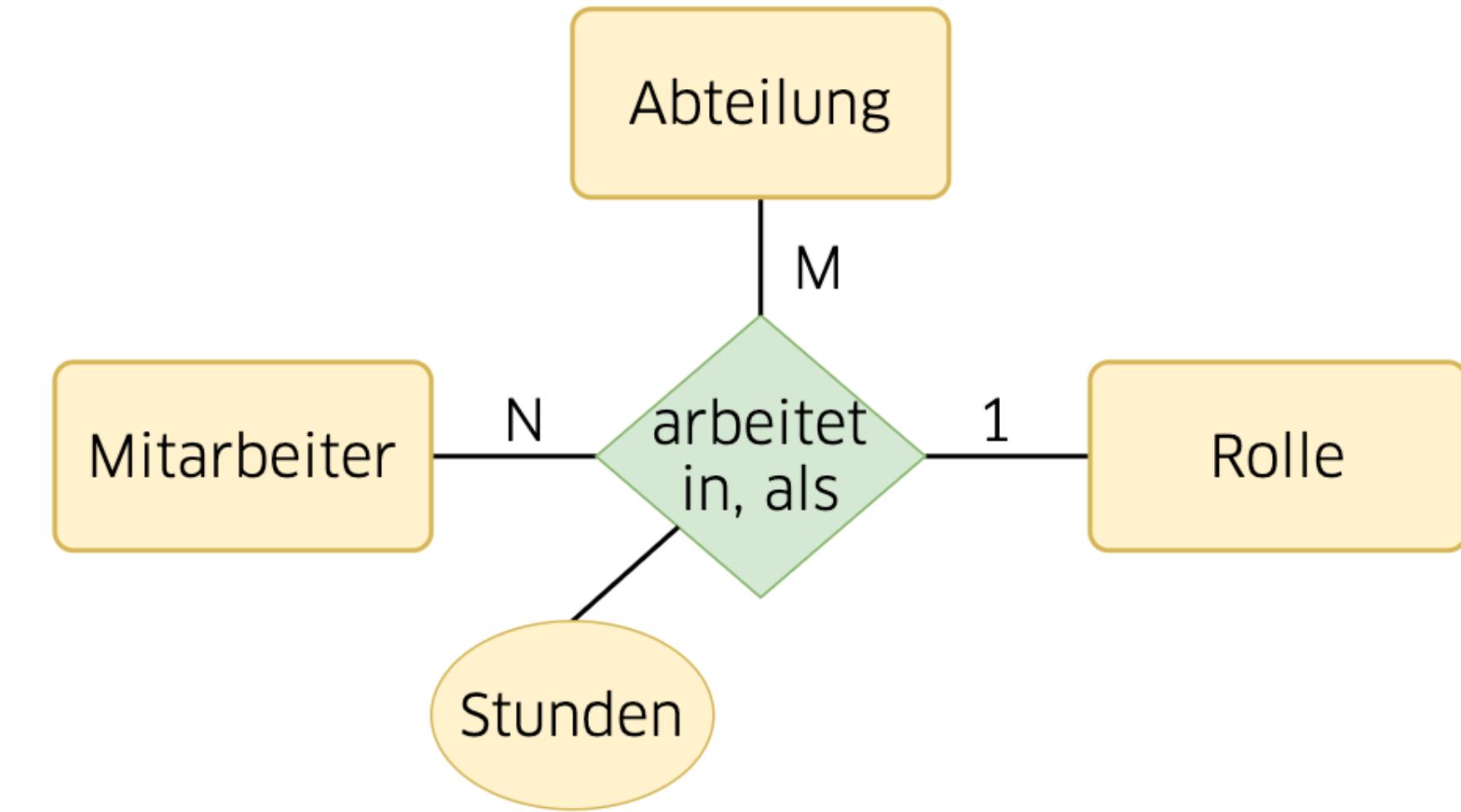


# Relationen

---

## n-stellige Relationen - Tabellen

- Ausser, dass in der Relation arbeitet\_in\_als hier mehrere Fremdschlüssel vorkommen, ändert sich nichts.
- Die Stunden (wochenstunden) sind ein Attribut der Relation – wie zuvor besprochen.



	<b>id</b>	<b>mitarbeiter_id</b>	<b>abteilung_id</b>	<b>rolle_id</b>	<b>wochenstunden</b>
1	1		1	3	10.00
2	2		1	4	5.00
3	3		1	3	10.00
4	4		2	4	15.00
5					

Beispiel: arbeitet\_in\_als aus matse\_mhist

	<b>id</b>	<b>name</b>
1	1	Mia
2	2	Ben
3	3	Emma
4	4	Paul
5		Hannah

mitarbeiter

	<b>id</b>	<b>name</b>
1	1	Vorstand
2	2	HR/Buchhalt
3	3	Vertrieb
4	4	Marketing
5		Einkauf

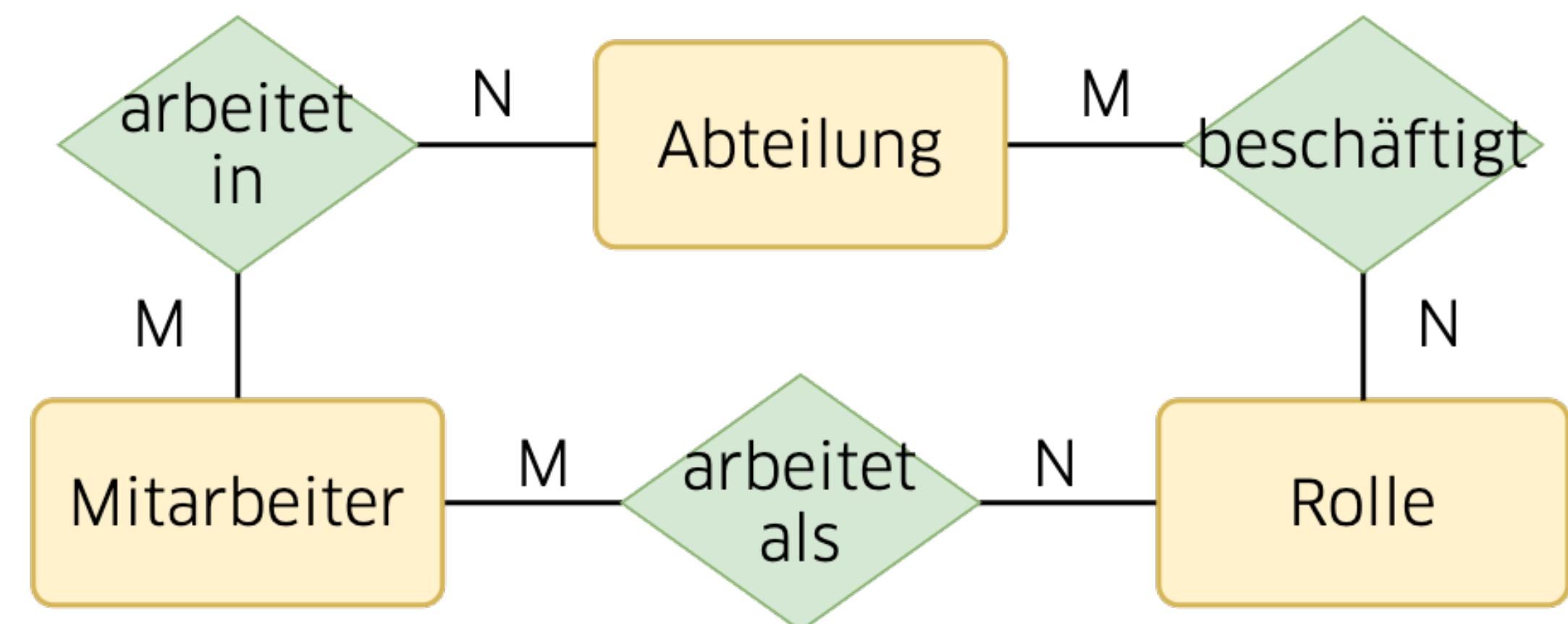
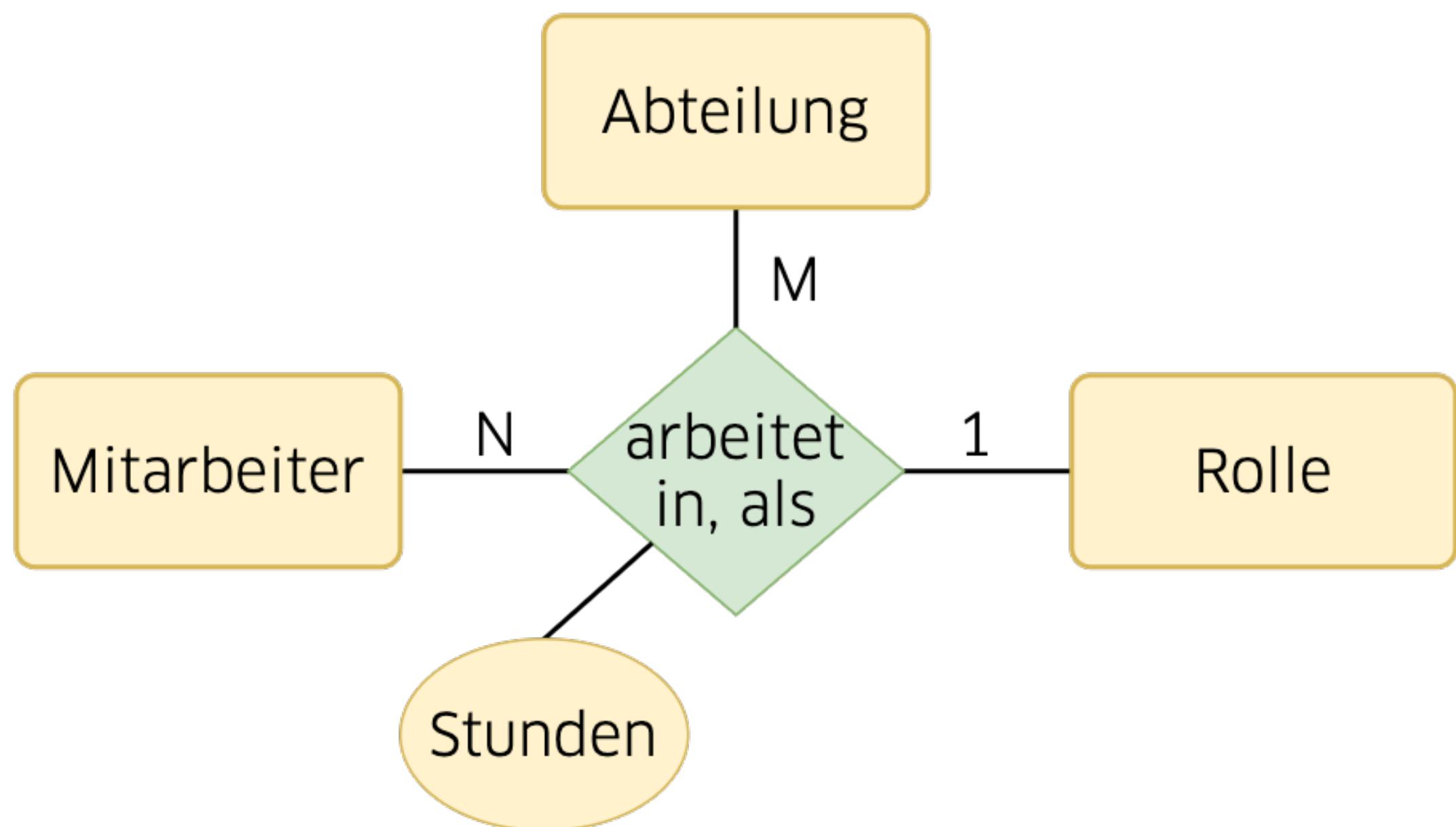
abteilung

	<b>id</b>	<b>beschreib</b>
1	3	Leitung
2	4	Mitarbeiter
3	7	Student
4	8	Auszubilden
5		Schüler

rolle

## Relationen

### n-stellige Relationen - keine Äquivalenzen



- In der rechten Version gehen Informationen verloren. Mitarbeiter arbeiten in Abteilungen, aber es ist unklar 'als was'.
- Die Anforderung aus der Mini-Welt (ein Mitarbeiter in einer Abteilung nur in einer Rolle) ist so nicht abzubilden.

#### Q&A

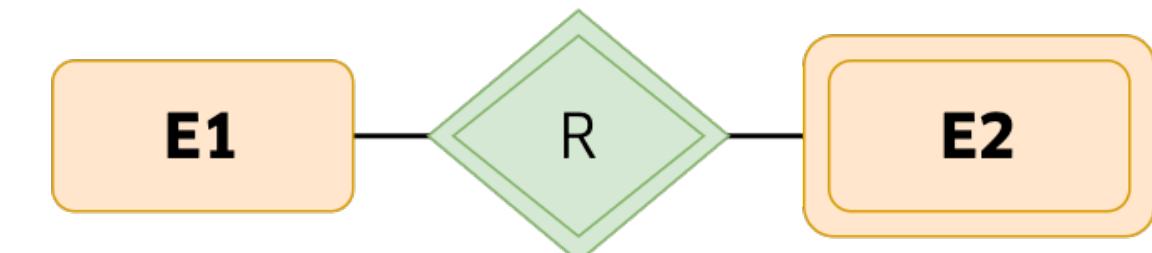
- Wo passen die Stunden hin?

## Relationen

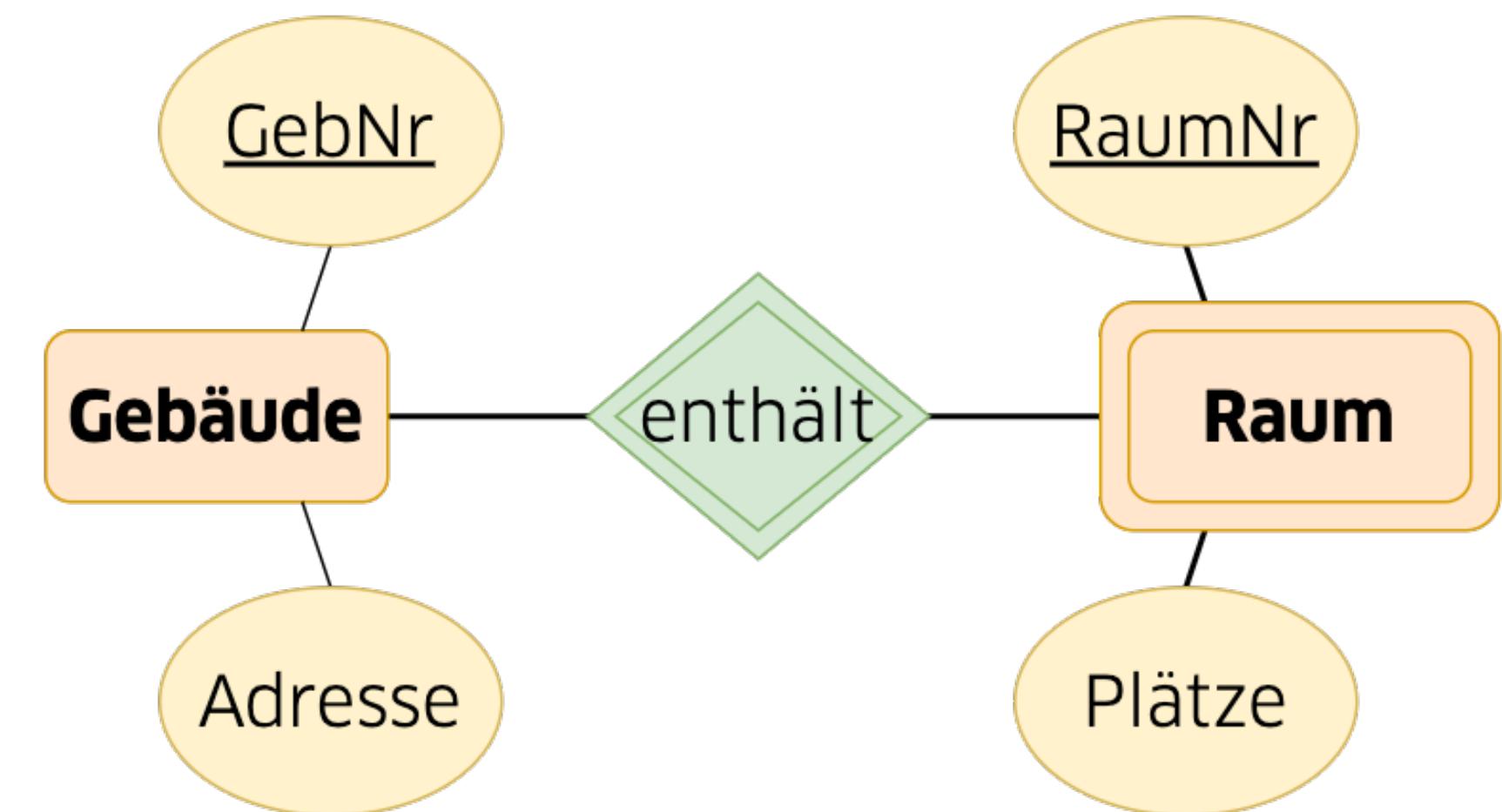
---

### Existenzabhängige Entitätstypen

- *Grundidee:* Entitäten existieren nur in Verbindung mit anderen Entitäten (wie UML Komposition).
- *Beispiel:* Ein Raum in einem Gebäude.
- Kein eigener Schlüsselkandidat (RaumNr alleine nicht geeignet), sondern Identifikation über Beziehung zur übergeordneten Entität, z.B. über Kombination (GebNr,RaumNr).
- Max. 1:1- oder 1:N-Relationen zwischen abhängigem (schwachem) Entitätstyp (Raum) und Vater-Entitätstyp (Gebäude), wobei die schwache Entität in Relation stehen muss!



Symbol eines existenzabhängigen Entitätstypen im ER-Diagramm



## Relationen und Kardinalität

---

### Chen-Notation

- Contra: Angabe ungenau, nur '1' (0 oder 1) und 'viele' (N,\*) möglich.

### Min-Max-Notation

- *Idee:* In der Min-Max-Notation wird angegeben, wieviele Relationen minimal bzw. maximal möglich sein sollen.
- Achtung: Hier werden die Elemente der Relation gezählt!

## Relationen und Kardinalität

---

### Mini-Welt Fotoshooting V

**Hintergrund:** Auf einer Familienfeier [...]

Auf den Fotos am Geburtstag sollen der Geselligkeit wegen auf jedem Foto mind. zwei aber max. vier Personen zu sehen sein. Umgekehrt soll eine Person mind. einmal aber auf nicht mehr als drei Fotos zu sehen sein.



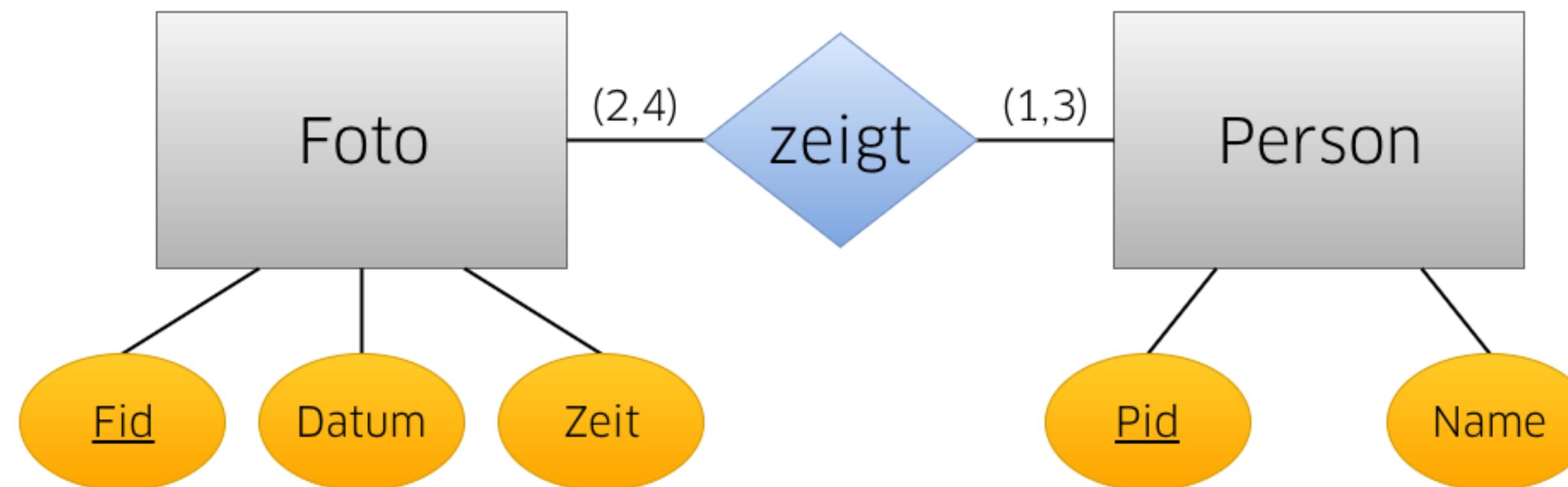
Ist so in Chen nicht abzubilden.

## Relationen und Kardinalität

---

### ER-Diagramm mit Min-Max-Kardinalität

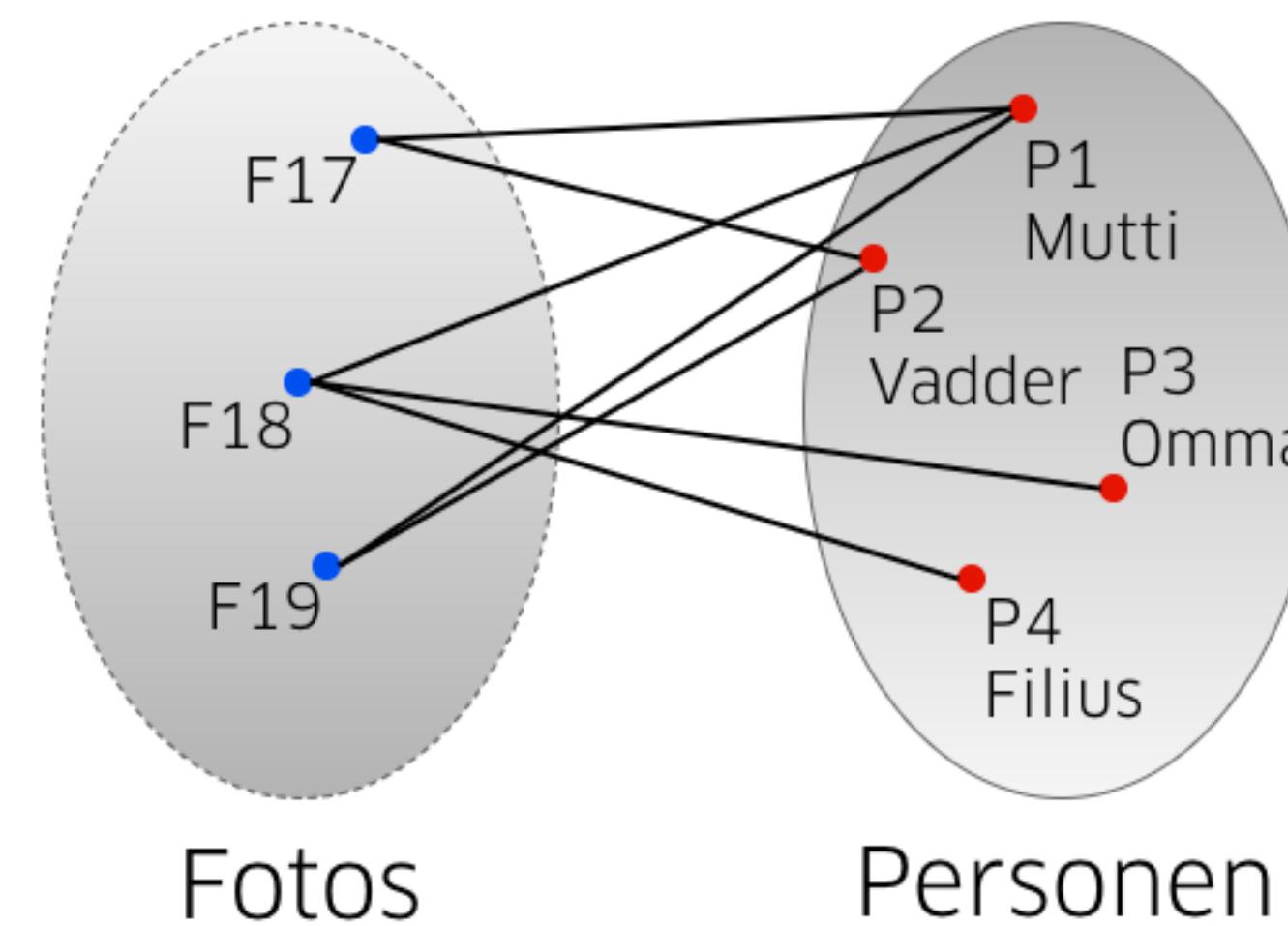
- Auf jedem Foto sind mind. zwei aber max. vier Personen zu sehen.
- ← Eine Person ist mind. einmal aber auf nicht mehr als drei Fotos.



## Relationen und Kardinalität

### ER-Diagramm mit Min-Max-Kardinalität

- Hier werden die Elemente der Relation gezählt!



Ein Foto zeigt mind. 2, max. 4 Personen und eine Person ist min. einmal, max. 3 mal auf Fotos.

Fid	Datum	Zeit
F17	29.02.16	21:01
F18	29.02.16	21:02
F19	29.02.16	21:03

Pid	Name
P1	Mutti
P18	P1
P18	P3
F18	P4
F19	P1
P4	Filius
F19	P2

Fid	Pid
F17	P1
F17	P2



$$2 \leq |F_i| \leq 4, 1 \leq |P_j| \leq 3$$

## Relationen und Kardinalität

---

### Min-Max-Notation

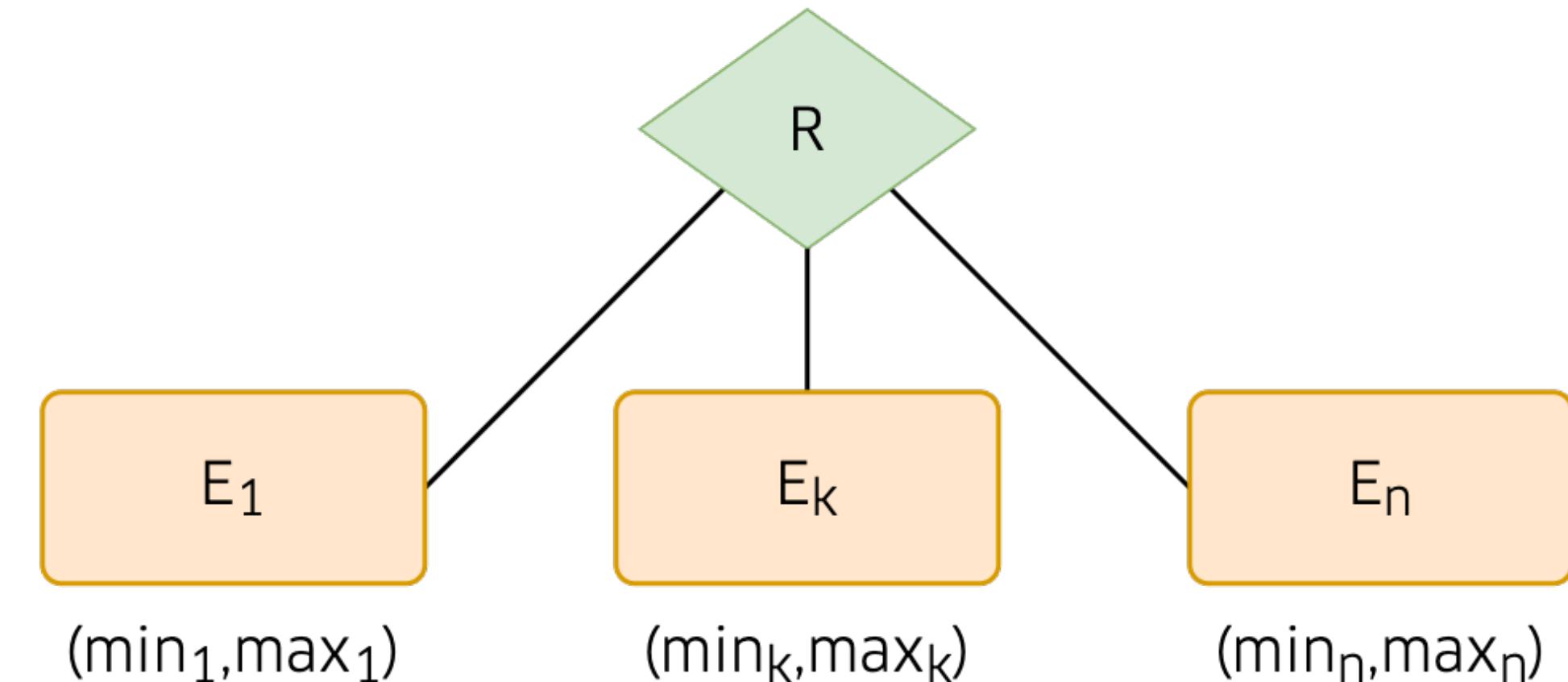
- Für jeden an einer Relation beteiligten Entitätstyp  $E_k$  wird angegeben, wie oft eine Entität minimal und maximal in der Relation enthalten sein darf (vgl. Beispiel).
- Sei  $R \subseteq E_1 \times \dots \times E_n$  und die Projektion auf das  $k$ -te Element bzw. die Anzahl eines Wertes an  $k$ -ter Stelle definiert durch

$$\text{proj}_k(e_1, \dots, e_k, \dots, e_n) = e_k,$$

$$\text{count}_k(e) = |\{r \in R \text{ mit } \text{proj}_k(r) = e\}|,$$

dann muss für alle  $k$  gelten:

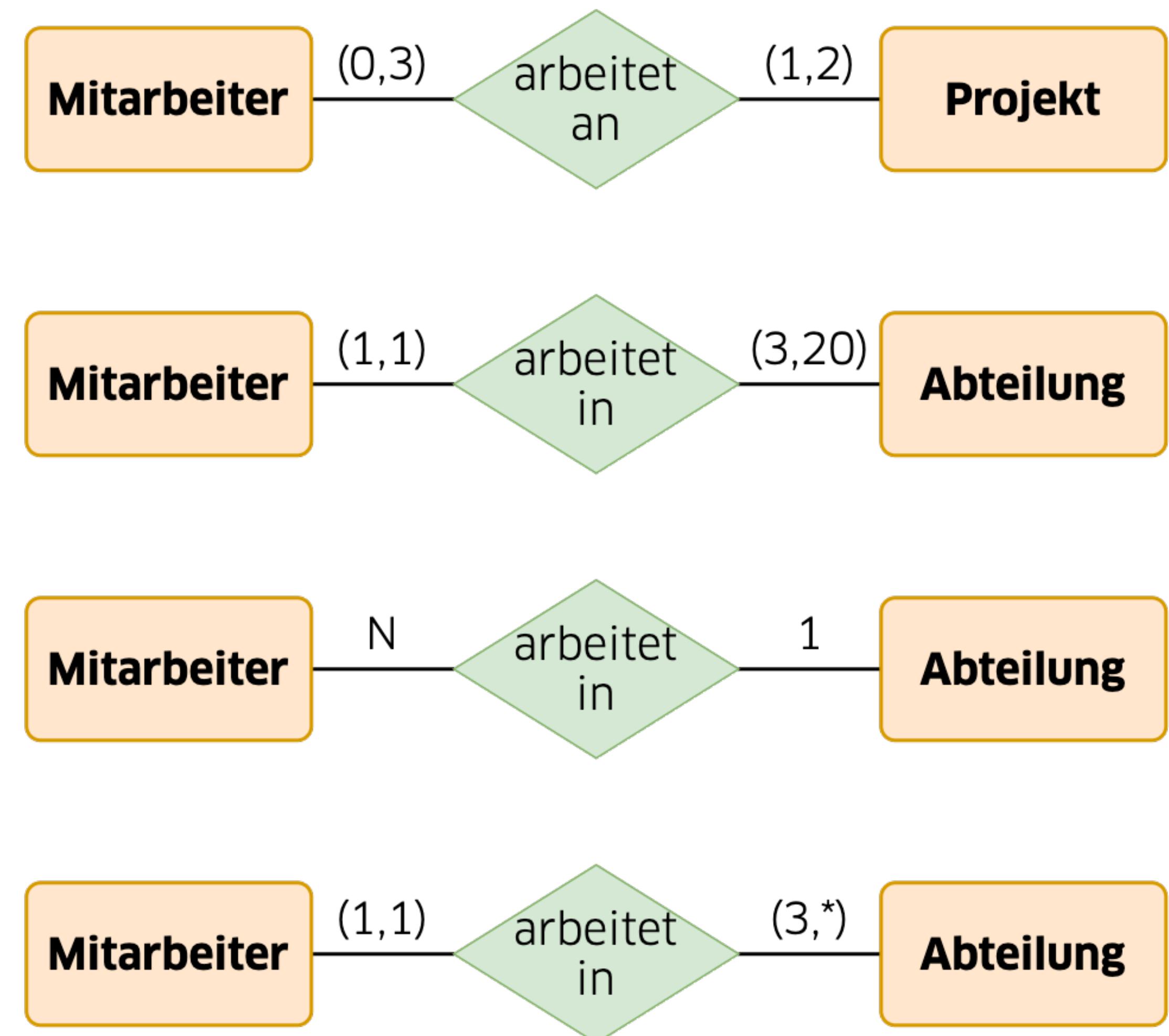
$$\forall e_k \in E_k : \min_k \leq \text{count}_k(e_k) \leq \max_k.$$



# Relationen und Kardinalität

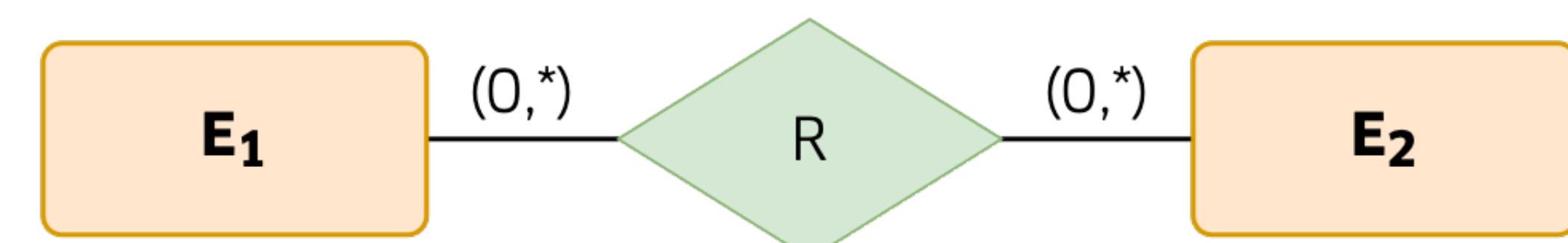
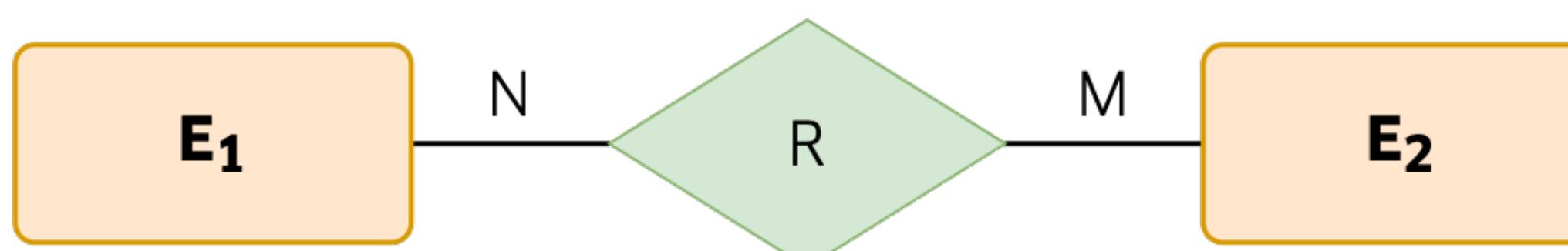
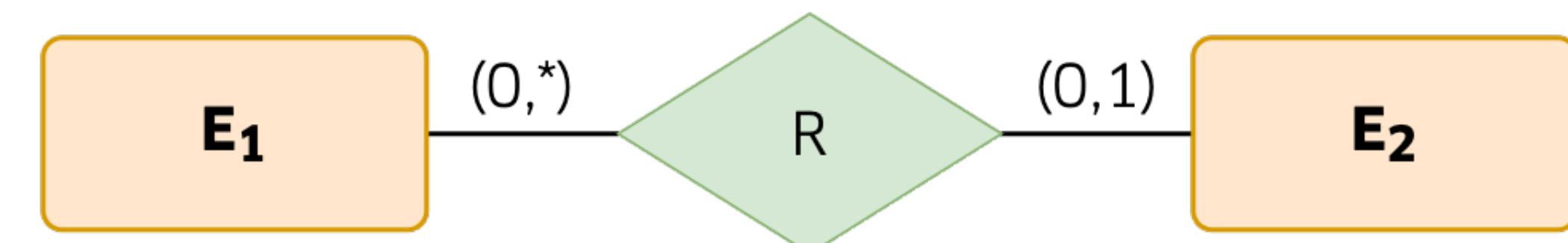
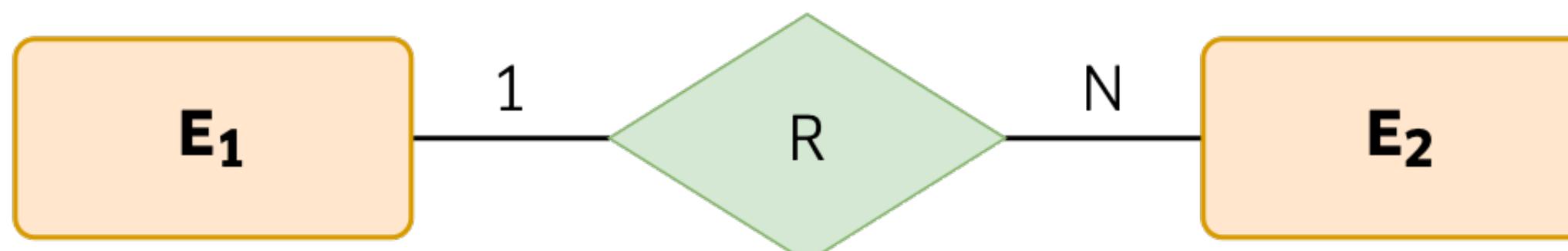
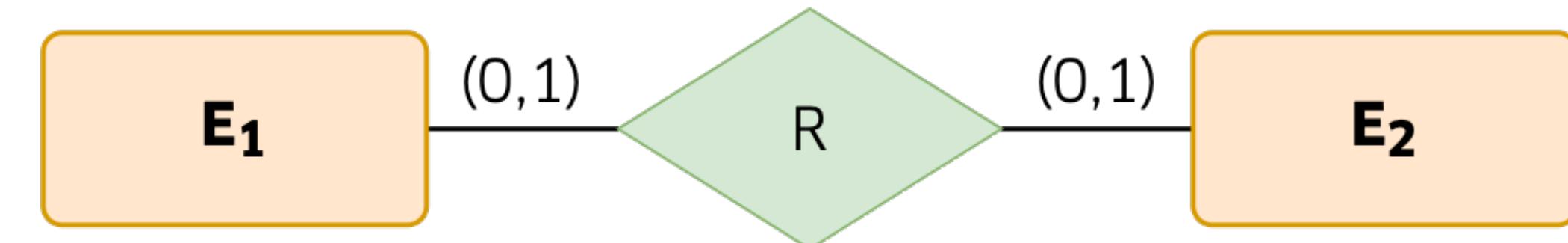
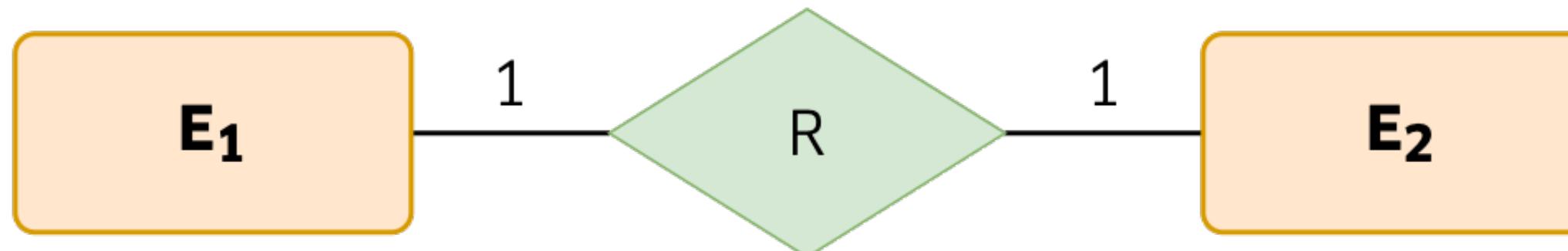
## Min-Max-Notation

- Jeder Mitarbeiter arbeitet an 0 bis 3 Projekten und ein Projekt wird von einem oder zwei Mitarbeitern bearbeitet.
- Jeder Mitarbeiter arbeitet in genau einer Abteilung und eine Abteilung besteht aus mind. 3, max. aber 20 Mitarbeitern.
- Gegenüber gestellt die Chen-Notation. Achtung: Umgekehrt!
- Wie bei Chen auch, darf ein '\*' oder 'N' verwendet werden, wenn eine Entität beliebig oft in einer Relation enthalten sein darf.



## Relationen und Kardinalität

### Chen-Notation vs. Min-Max-Notation



Hier unbedingt die Unterschiede und unterschiedlichen Ausprägungen in den Relationen-Tabellen verstehen.



## Relationen und Kardinalität

---

### Kardinalität in UML-Notation

- Idee: Kardinalität wie in der Chen-Notation, aber mit Min-Max Angaben.



... nur nicht verwirren lassen ...

### Hintergrund UML

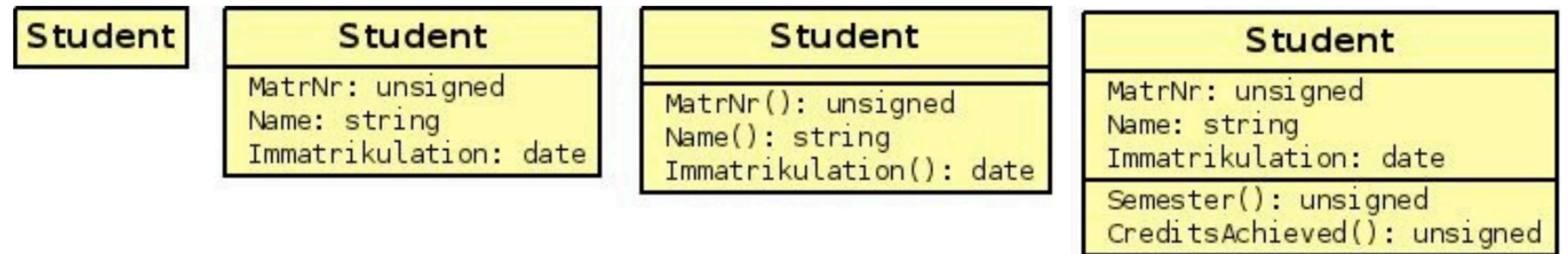
- Notation zur objektorientierten Modellierung (siehe z.B. Booch, Rumbaugh, Jacobson).
- Standardisierung 1997 Object Management Group (OMG).
- Verschiedene Diagrammtypen → Thema in SWE.

## UML Klassen

---

### Klassensymbol

- Rechteck mit drei Sektionen für
  - Klassenname,
  - Attribute (optional) und
  - Methoden (optional).
  - Hier werden häufig nur die relevanten Details gezeigt.



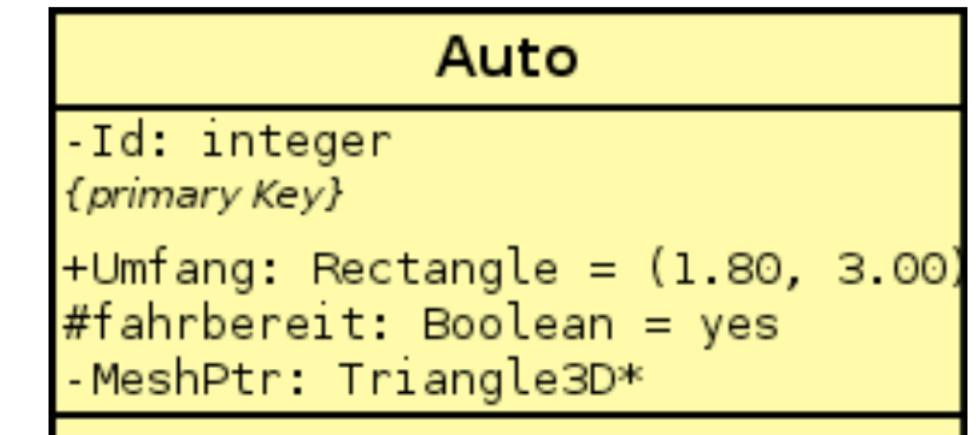
UML-Diagramme  
Prof. Striegnitz

## UML Klassen

---

### Attributspezifikation

- Sichtbarkeit: Name:Typ = Vorgabewert { Eigenschaften }



### Operationen/Methoden

- Sichtbarkeit: Name(Parameterliste): Rückgabetyp { Eigenschaften }

UML-Diagramme  
Prof. Striegnitz

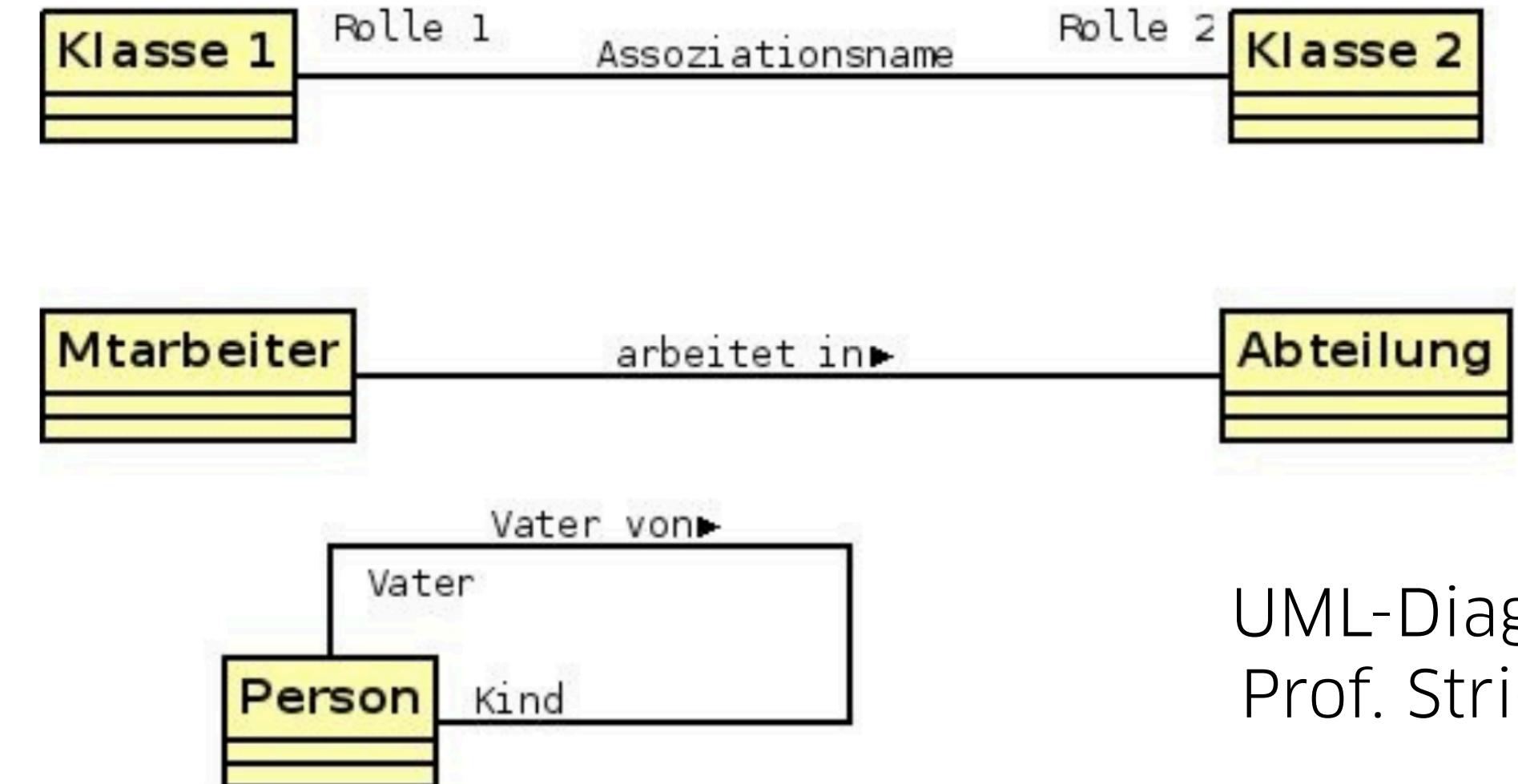
### Notationen

- Sichtbarkeit: +:public -:private #:protected
- Unterstrichene Attribute/Operationen stehen für Klassenattribute/Operationen.
- Eigenschaften können Bedingungen (constraints) sein.
- Schlüsselattribute z.B. als Kommentar / Eigenschaft (vgl. Attributierung).

## UML Assoziationen

### Repräsentation von Relationen

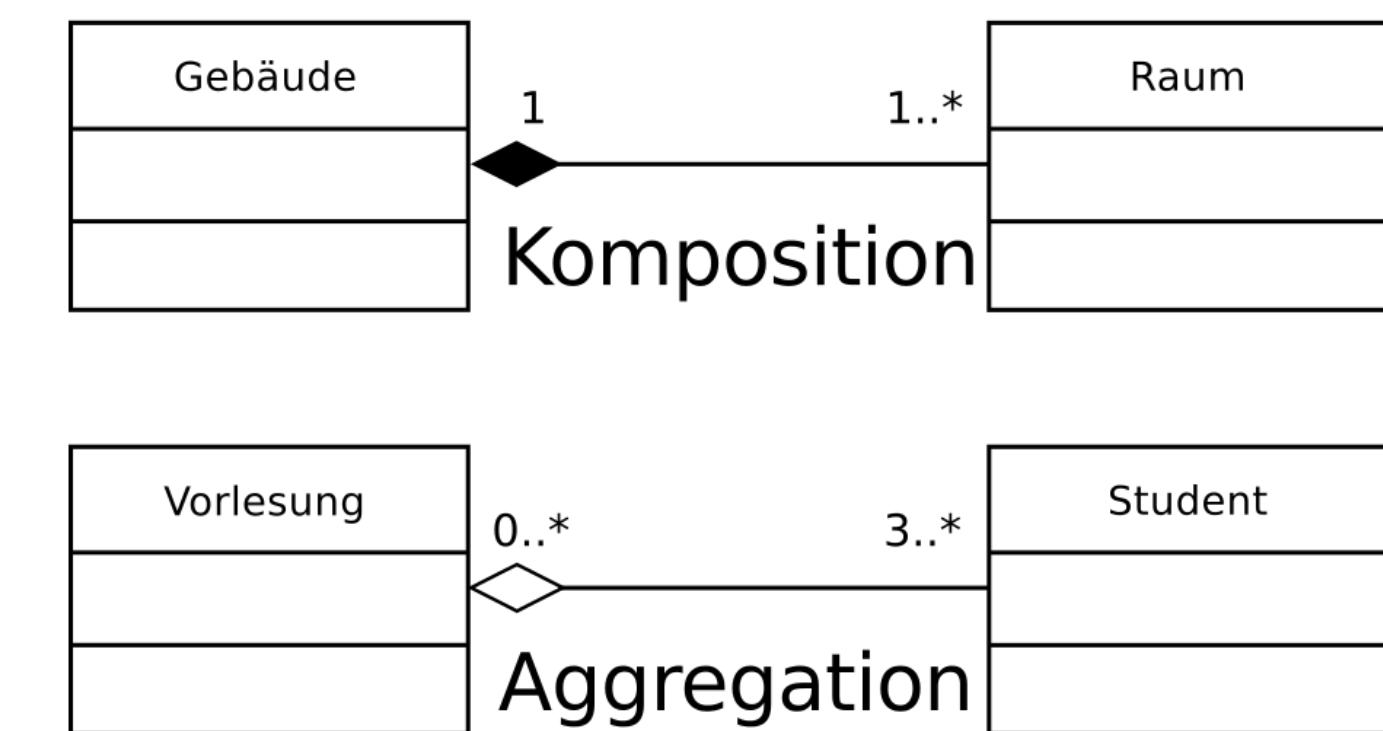
- Optional
  - Assoziationsname
  - Leserichtung ( $\blacktriangleright$  oder  $\blacktriangleleft$ )
  - Rollenname, Sichtbarkeit von Rollen
  - Kardinalitäten



UML-Diagramm  
Prof. Striegnitz

### Spezialfälle der Assoziation

- Komposition: von der Existenz des Ganzen abhängig
- Aggregation: eine "Teil von"-Beziehung

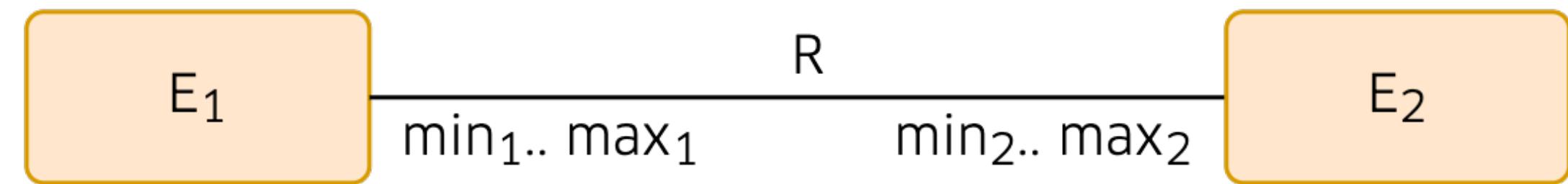


## Relationen und Kardinalität

---

### UML-Notation

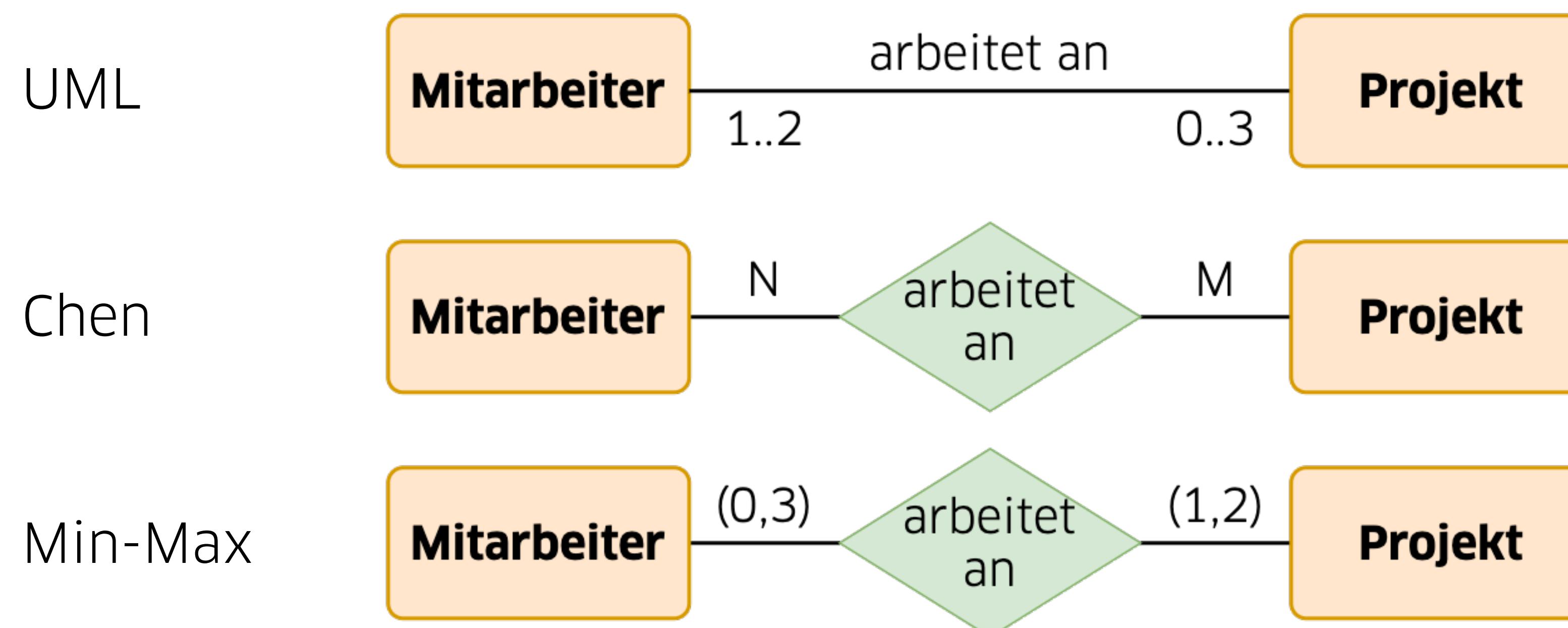
- Für jeden an einer Relation beteiligten Entitätstyp  $E_k$  wird angegeben, wie oft eine Entität minimal und maximal in  $E_k$  enthalten sein darf, wenn der andere Eintrag ( $n=2$ ) bzw. die anderen Einträge ( $n>2$ ) fest gewählt ist bzw. sind.
- Gleiche Sichtweise wie bei der Chen-Notation, nur mit Angabe der minimalen und maximalen Elemente in der Entitätenmenge.
- Dadurch ist eine Angabe der Fälle
  - *optional*, also Kardinalität=0, und
  - *obligatorisch*, also Kardinalität>0, möglich.



## Relationen und Kardinalität

### UML-Notation im Vergleich

- Jeder Mitarbeiter arbeiten an keinem, höchstens aber an 3 Projekten mit.
- Ein Projekt wird von einem oder zwei Mitarbeitern bearbeitet.

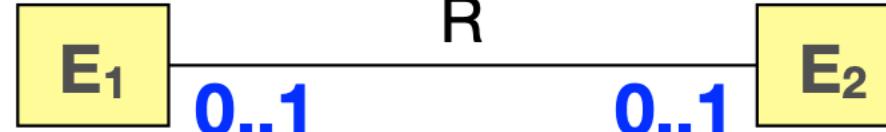
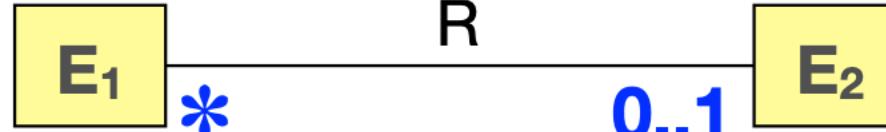
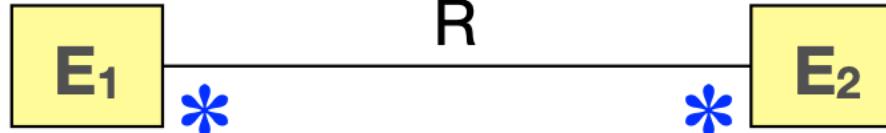


## Relationen und Kardinalität

---

### Vollständige vs. unvollständige UML-Relationen

- Vollständig, d.h. alle Entitäten nehmen teil.
- Unvollständig, d.h. es gibt Entitäten, nicht teilnehmen.
- Angabe in Kurzform, d.h. statt '0..\*' kurz '\*' und statt '1..1' kurz '1', Default ist '1'.

Typ	unvollständig	vollständig
<b>One-to-One</b>		
<b>Many-to-One</b>		
<b>Many-to-Many</b>		

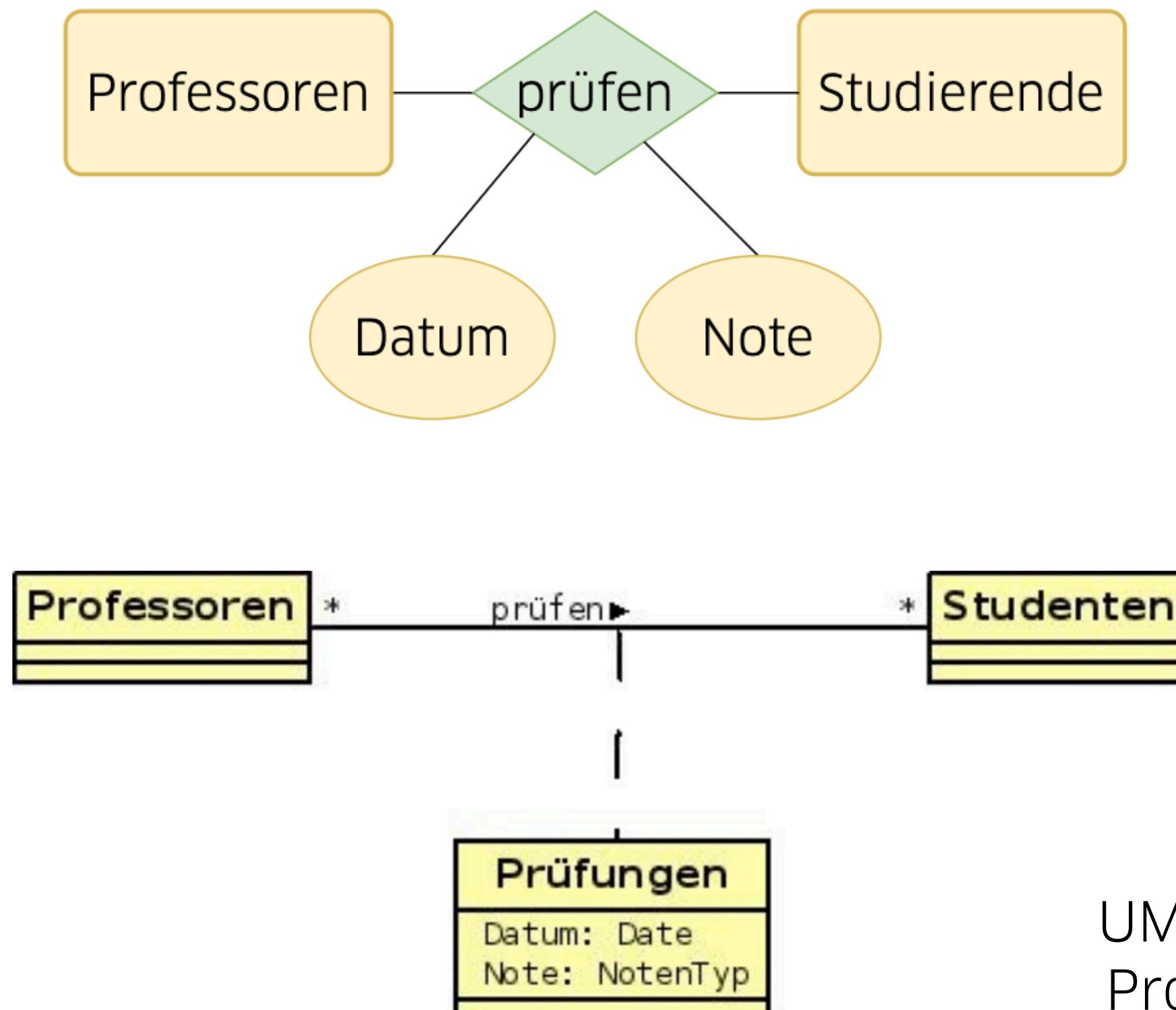
UML-Diagramm  
Prof. Striegnitz

## Relationen und Kardinalität

---

### UML-Relationen mit Attributen bzw. Relationen mit assoziierten Klassen

- Annotation von Relationen mit Attributen, in UML zusammengefasst in einer assoziierten Klasse.

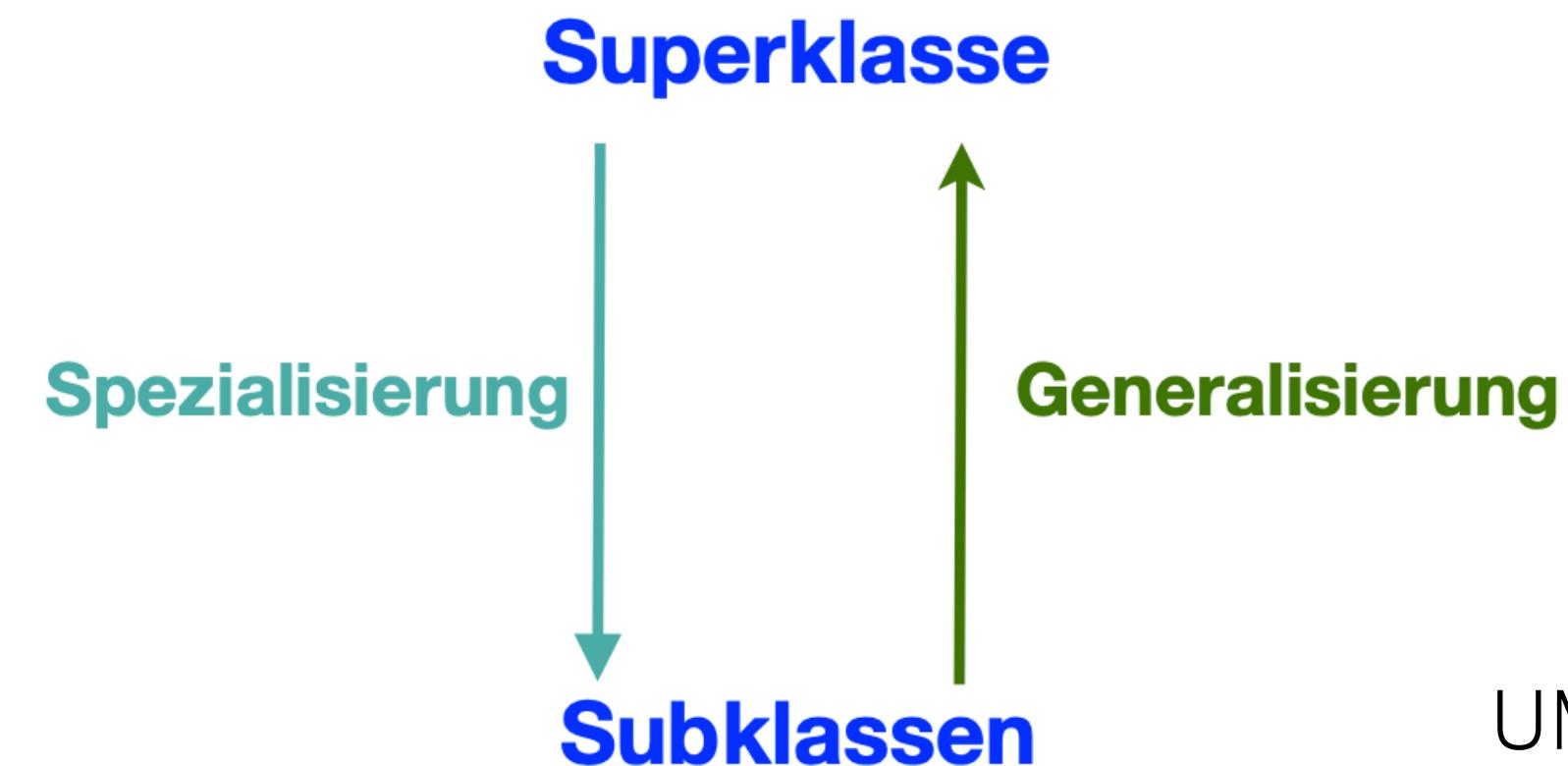
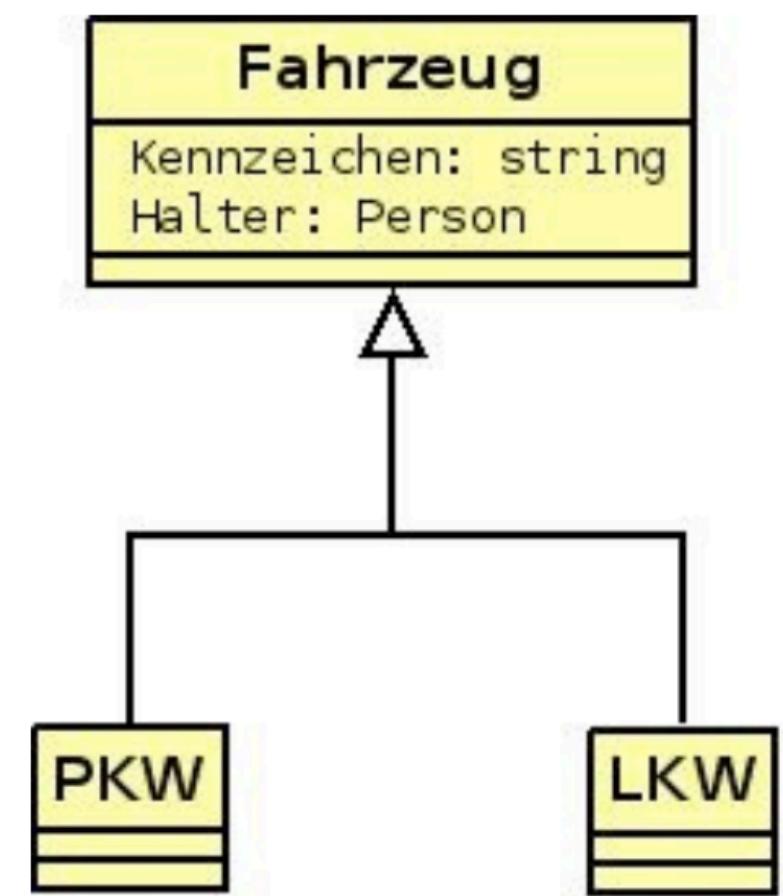


UML-Diagramm  
Prof. Striegnitz

## Generalisierung, Spezialisierung

### Ist-Ein-Beziehung

- Inklusionspolymorphie, d.h. jedes Objekt aus  $E_1$  ist auch ein Objekt aus  $E_2$ , kurz  $E_1$  'ist-ein'  $E_2$
- Vererbung von Eigenschaften bzw. Attributen der Superklasse an alle Subklassen



UML-Diagramm  
Prof. Striegnitz

## Generalisierung, Spezialisierung

### Inklusionspolymorphie

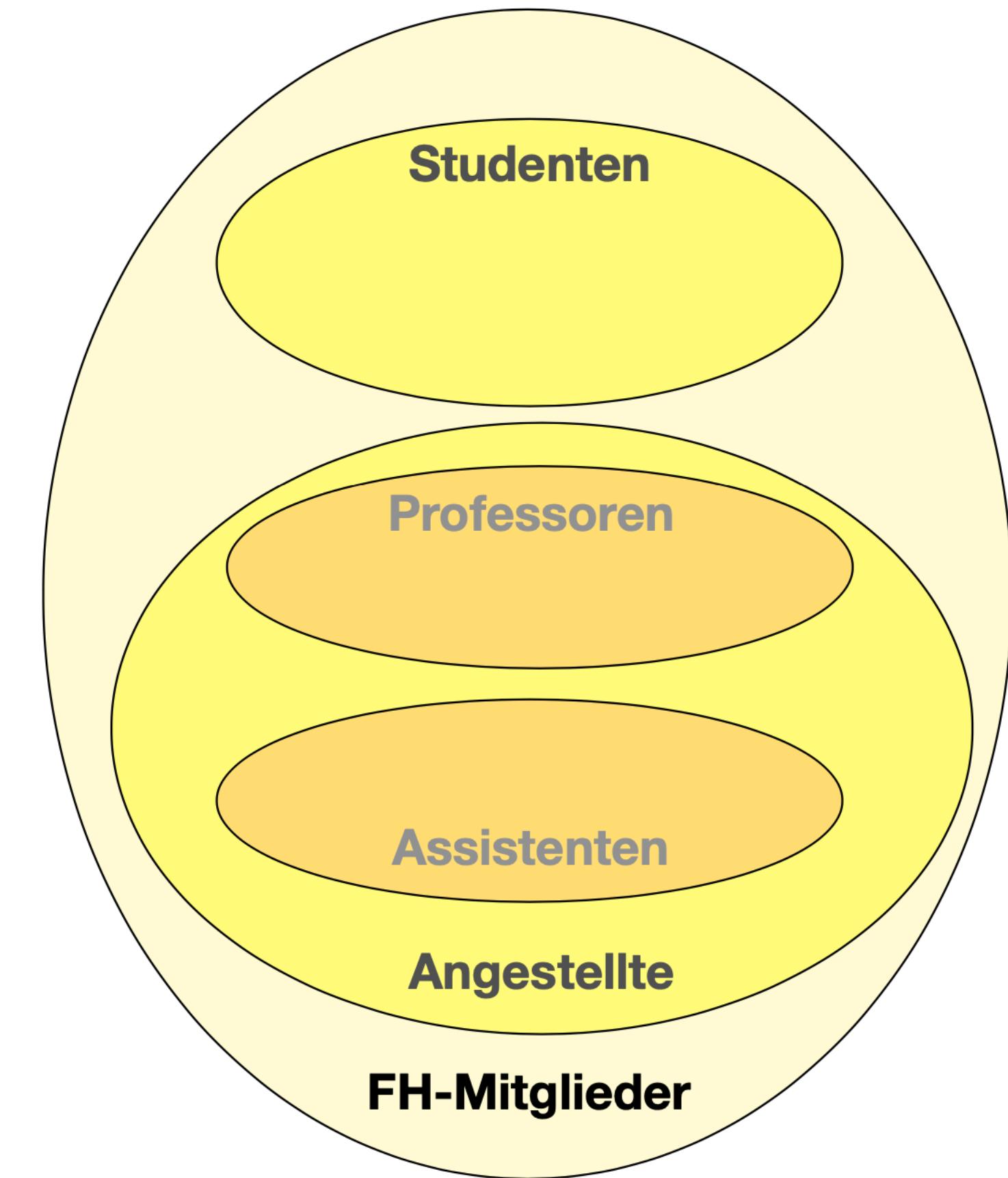
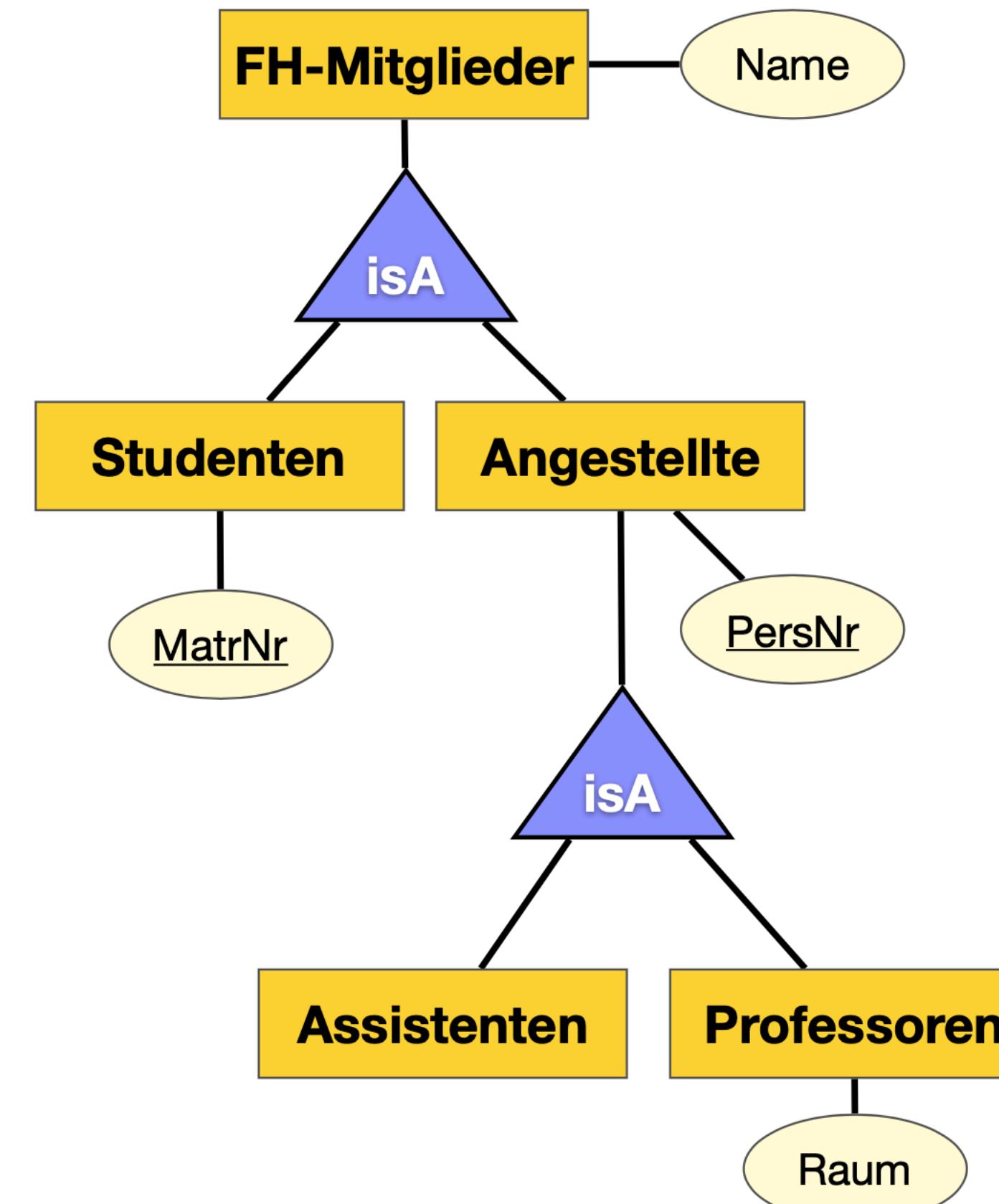


Diagramm  
Prof. Striegnitz

## Generalisierung, Spezialisierung

### Disjunkte / nicht disjunkte Vererbung

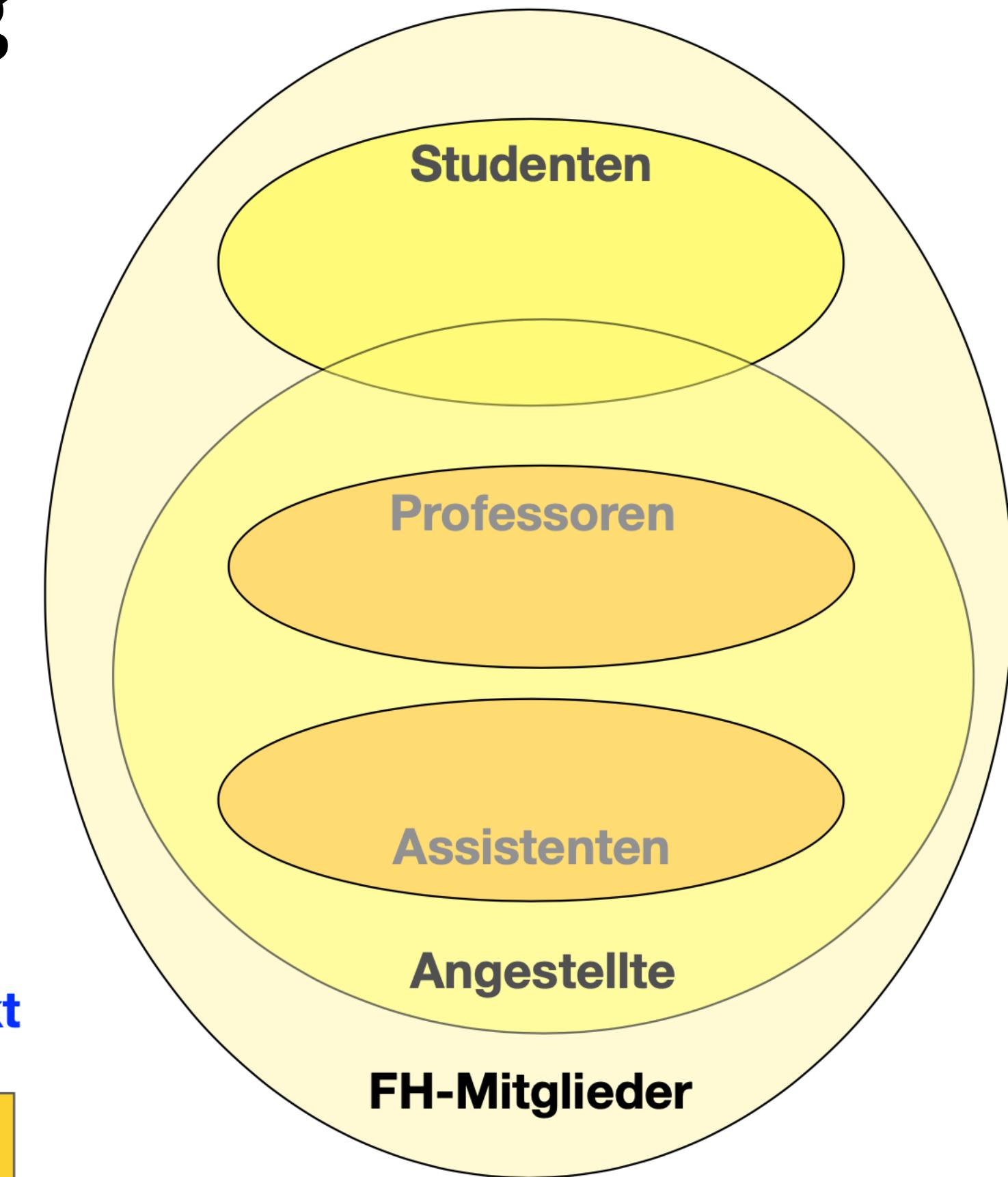
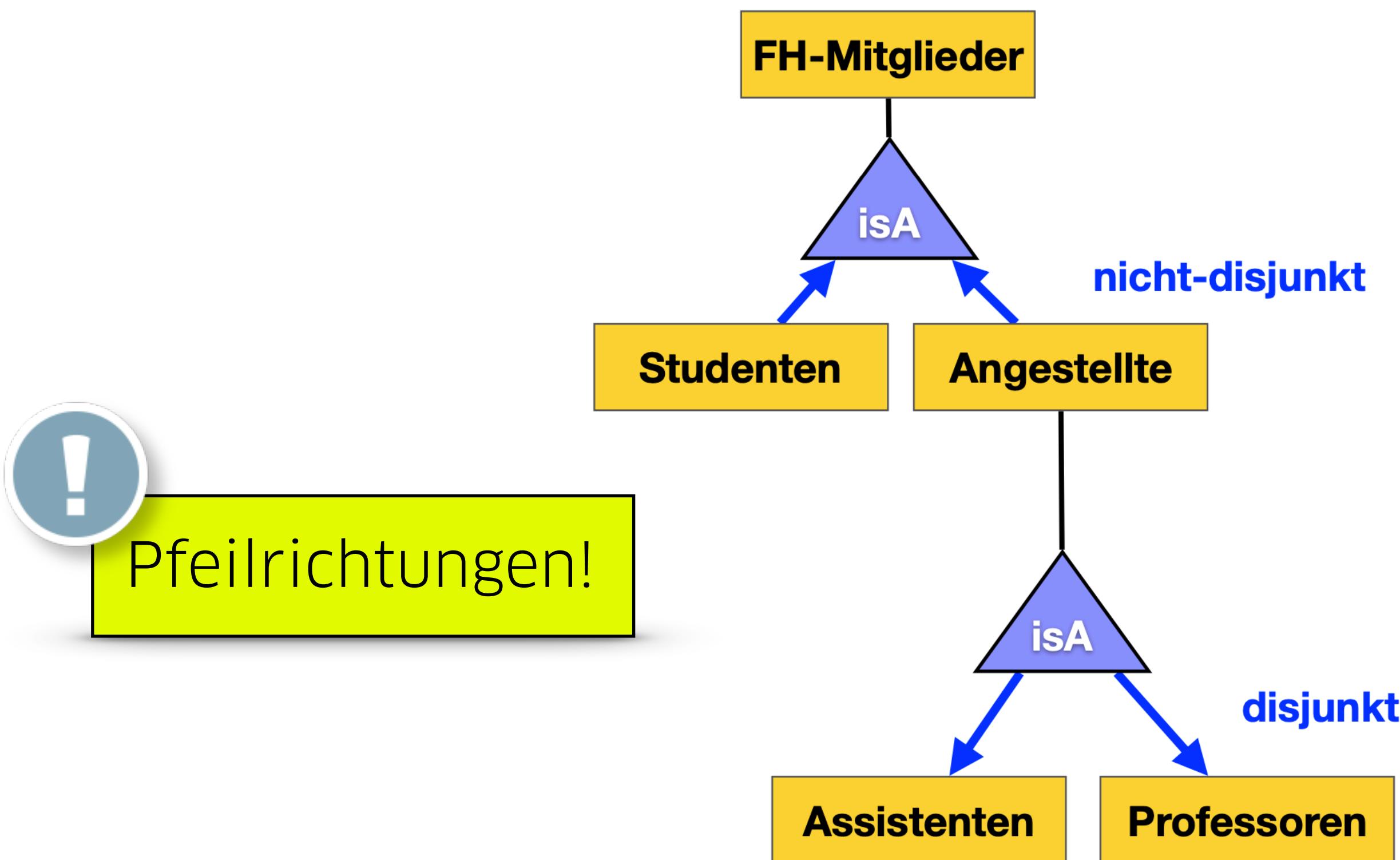


Diagramm  
Prof. Striegnitz

## Generalisierung, Spezialisierung

### Beispiel

- Randbedingungen
  - Nur Vierecke mit Innenwinkel von  $90^\circ$ .
  - Ein Quadrat sei kein Sonderfall eines Rechtecks.
- 't': *Totale Spezialisierung*, d.h. es gibt nur Rechtecke und Quadrate und sonst nichts, bzw.  
 $\text{Rechtecke} \cup \text{Quadrate} = \text{Vierecke}$
- *Disjunkte Spezialisierung*, d.h. keine Überlappung, bzw.  
 $\text{Rechtecke} \cap \text{Quadrate} = \emptyset$

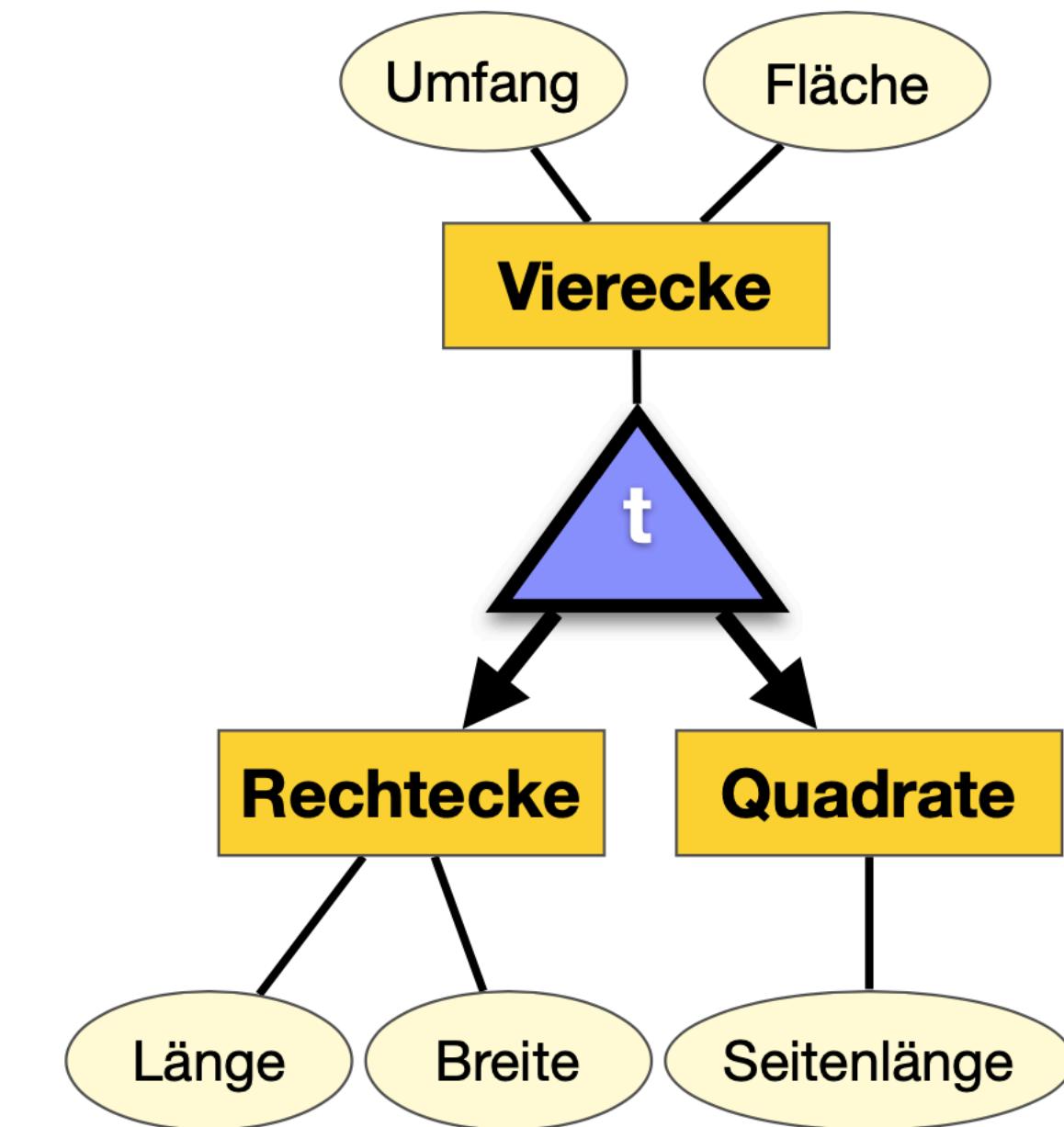


Diagramm  
Prof. Striegnitz

## Generalisierung, Spezialisierung

---

### Beispiel

- Randbedingungen
  - Nur Vierecke mit Innenwinkel von  $90^\circ$ .
  - Quadrat ist als Sonderfall eines Rechtecks erlaubt.
- 't': *Totale Spezialisierung*, d.h.  
 $\text{Rechtecke} \cup \text{Quadrate} = \text{Vierecke}$
- *Nicht-Disjunkte Spezialisierung*, d.h.  
 $\text{Rechtecke} \cap \text{Quadrate} \neq \emptyset$

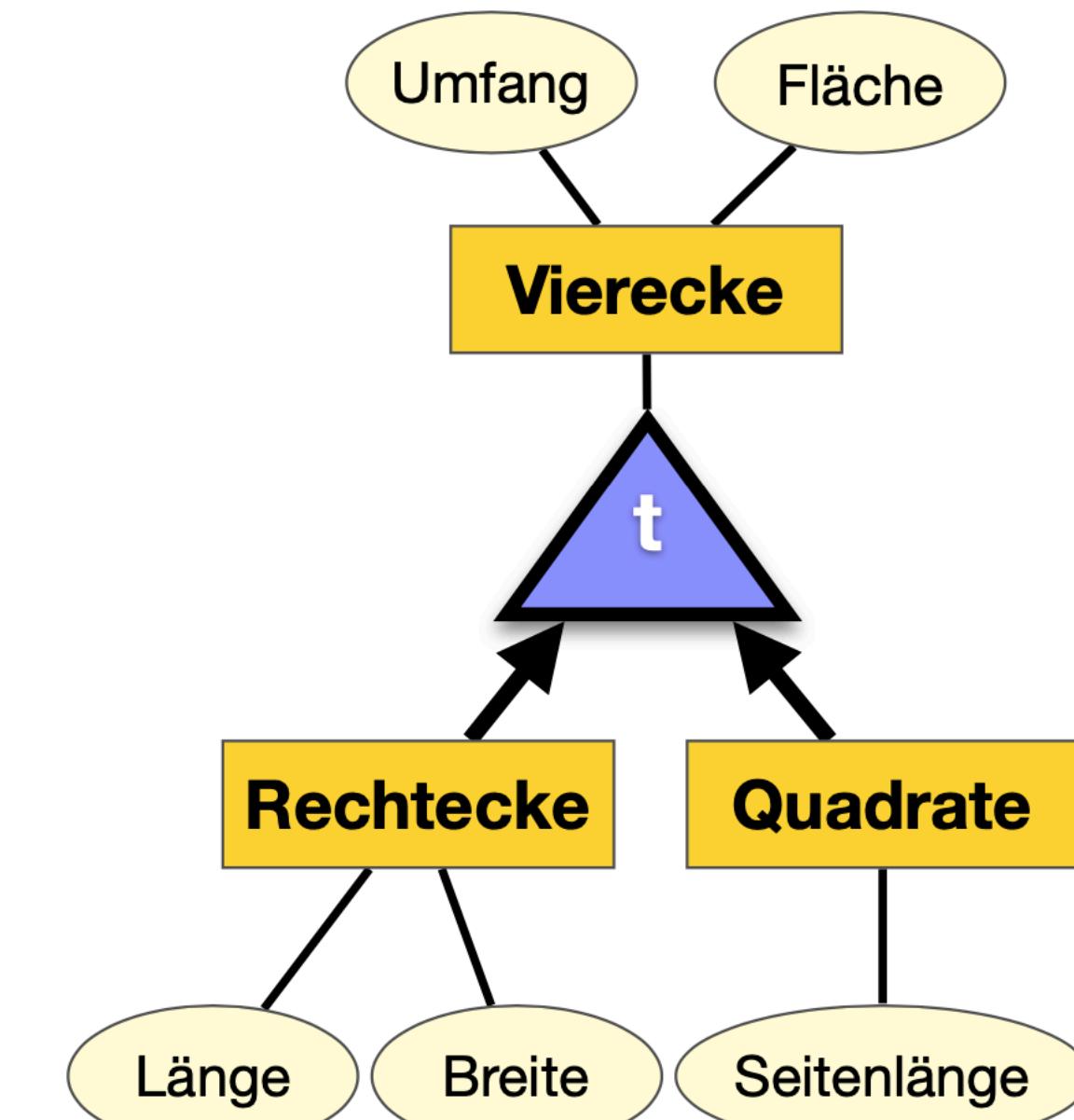


Diagramm  
Prof. Striegnitz

## Generalisierung, Spezialisierung

### Beispiel

- Randbedingungen
  - Beliebige Vierecke erlaubt, also auch Trapez, Parallelogram etc.
  - Ein Quadrat sei kein Sonderfall eines Rechtecks.
- 'p': *Partielle Spezialisierung*, d.h. es gibt noch mehr, was ggf. aber hier nicht aufgeführt ist, bzw.  
 $\text{Rechtecke} \cup \text{Quadrate} \subset \text{Vierecke}$
- *Disjunkte Spezialisierung*, d.h.  
 $\text{Rechtecke} \cap \text{Quadrate} = \emptyset$

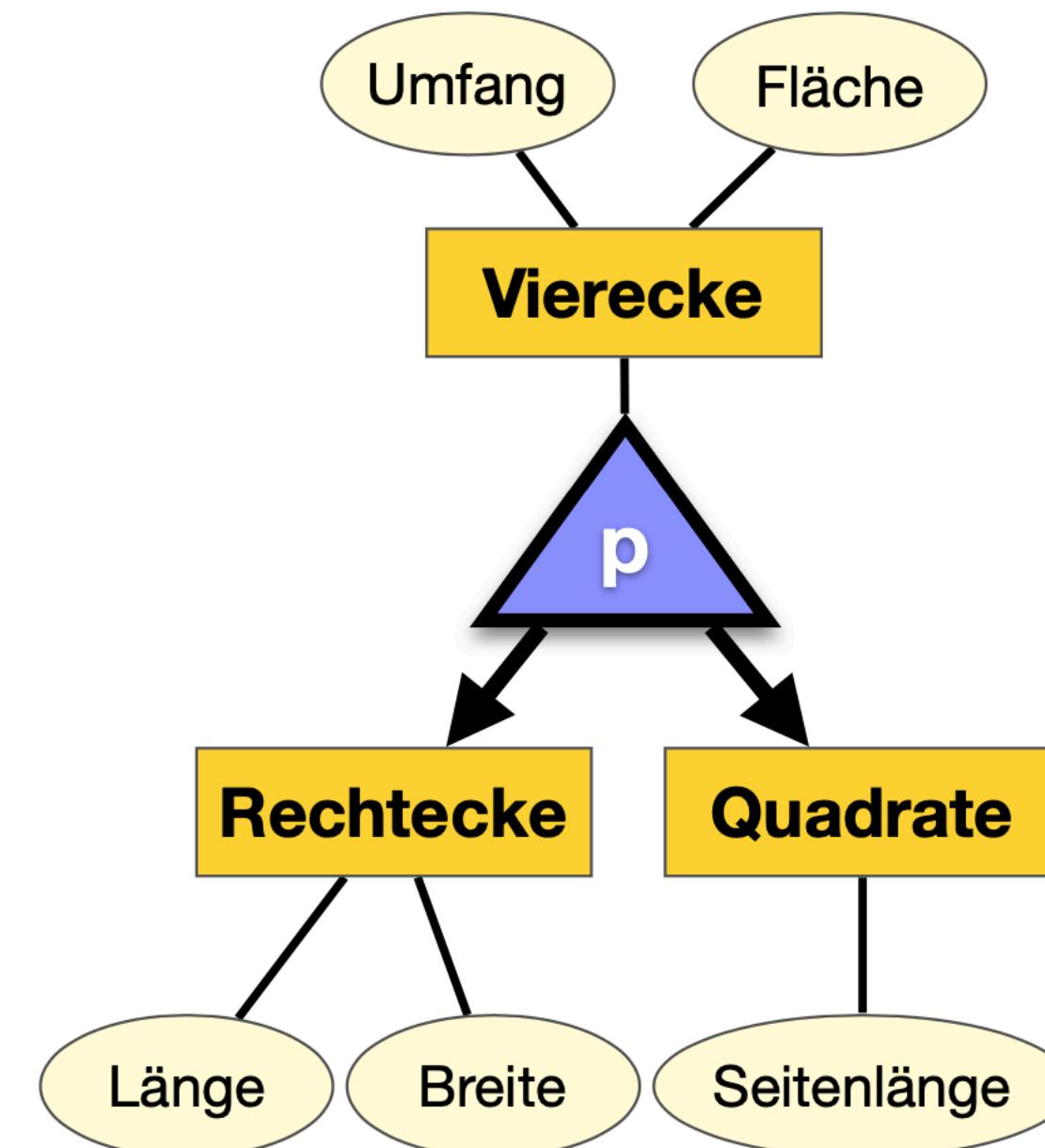


Diagramm  
Prof. Striegnitz

## Generalisierung, Spezialisierung

### Beispiel

- Randbedingungen
  - Beliebige Vierecke erlaubt, also auch Trapez, Parallelogram etc.
  - Quadrat ist als Sonderfall eines Rechtecks erlaubt.
- 'p': *Partielle Spezialisierung*, d.h.  
Rechtecke  $\cup$  Quadrate  $\subset$  Vierrecke
- *Nicht-Disjunkte Spezialisierung*, d.h.  
Rechtecke  $\cap$  Quadrate  $\neq \emptyset$

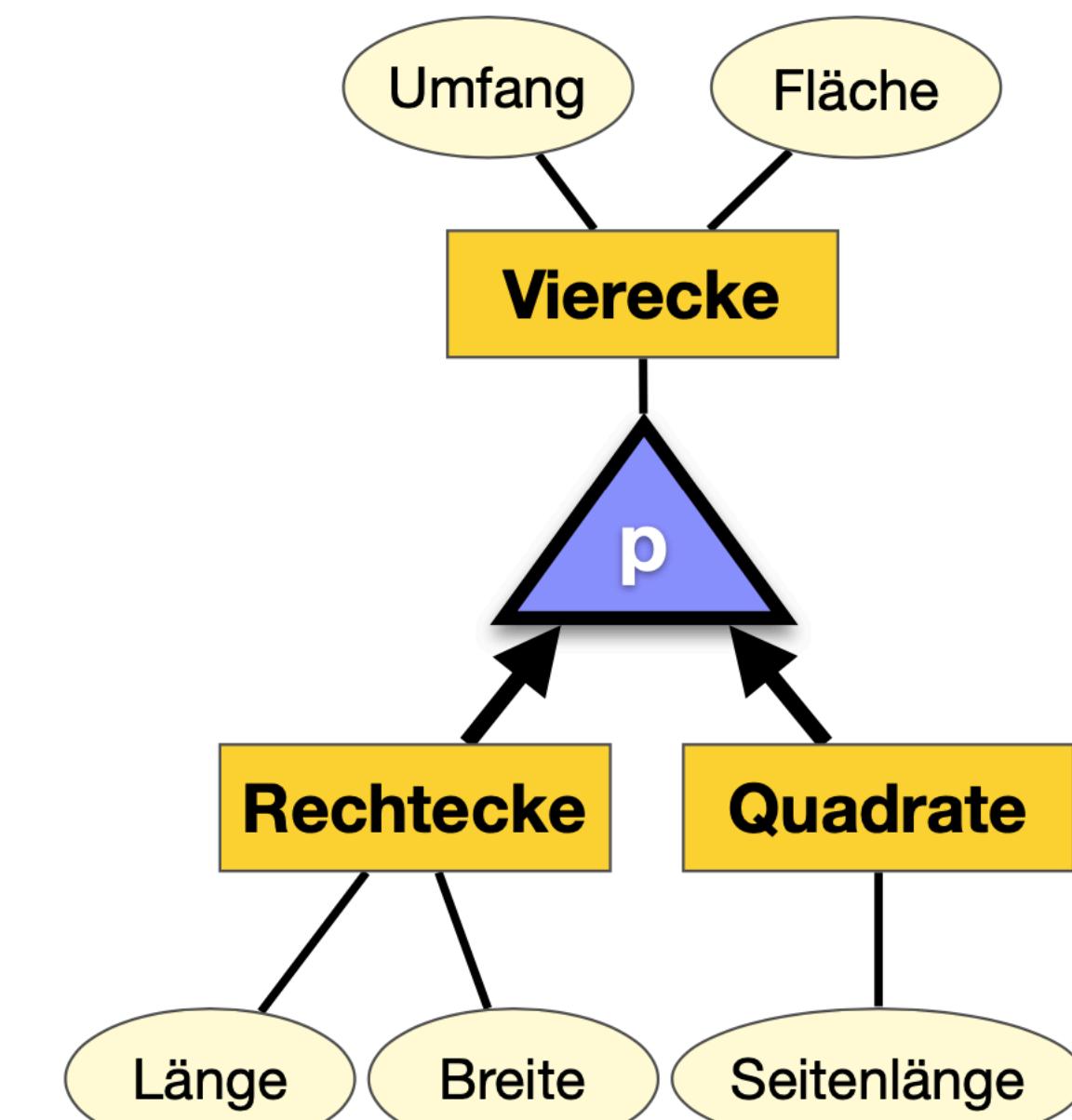


Diagramm  
Prof. Striegnitz

## Generalisierung, Spezialisierung

### Anmerkungen

- Behandlung von Spezialisierungshierarchien möglich, z.B. Modellierung typischer Objektwelten, aber:
- Abbildung in ein (relationales) DBMS nicht trivial.  
Mgl. Ansatz hier: sog. ORM (Objekt-relationale Mapper), ggf. sind aber nicht-relationale DBMS besser?

#### Q&A

- Wo liegt das Problem?

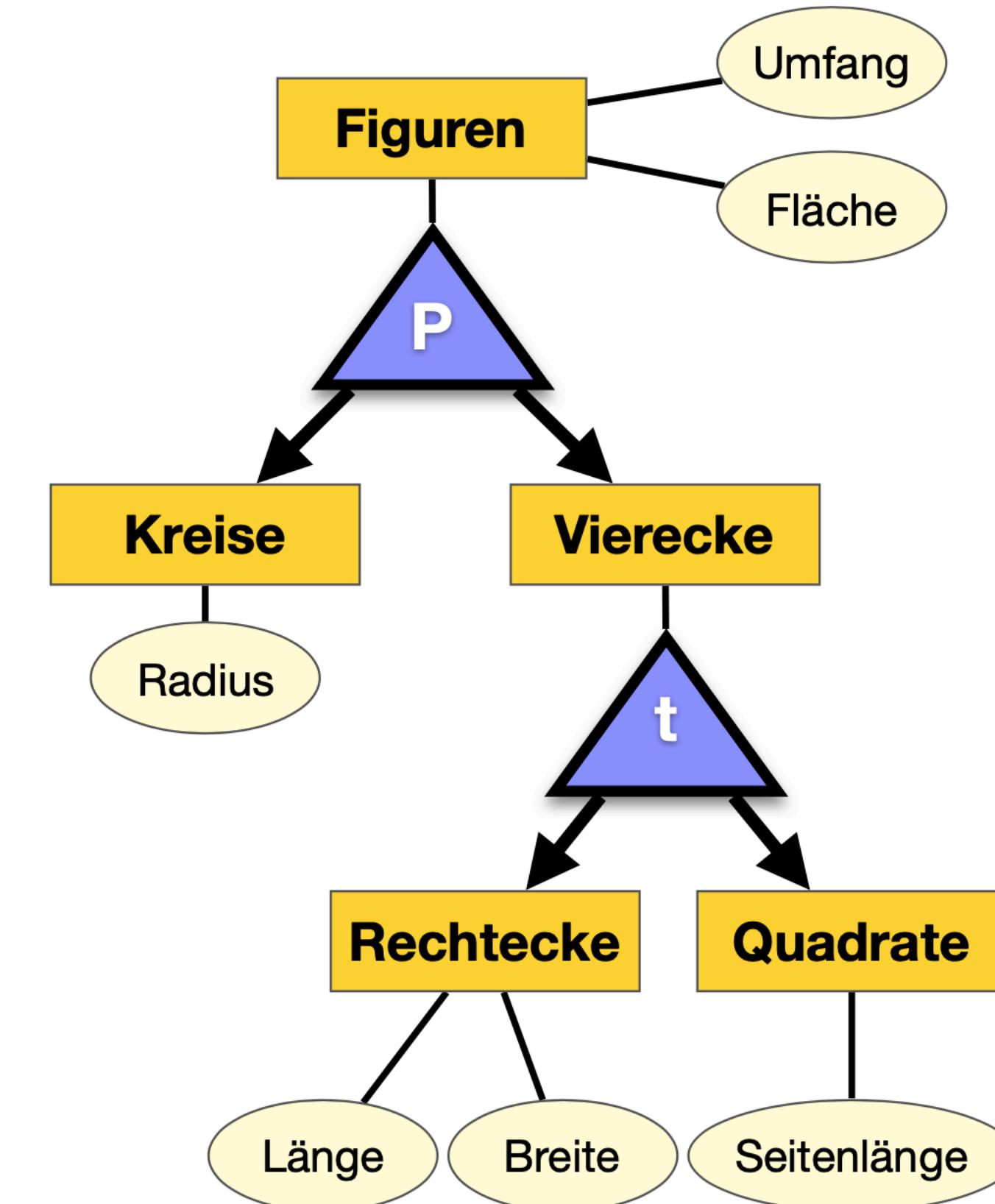


Diagramm  
Prof. Striegnitz

## Zusammenfassung

---

### Was ist wichtig?

- Weitere Elemente ER-Diagramme
  - Kardinalität in Chen-, Min-Max- und UML-Notation.
  - n-stellige Relationen
- Generalisierung, Spezialisierung

### Was folgt?

- Implementationsentwurf und 'Relationales Modell'.

## Fallbeispiel (Quelle Kemper/Eickler: Datenbanksysteme)

---

### Informationen aus Interviews

- Prüfungsamt: “Studierende erstellen Bachelorarbeiten zu einem bestimmten Thema und werden dabei von Assistenten betreut, die in der Regel an Ihrer Dissertation arbeiten. Das Thema der Dissertation wird mit dem betreuenden Professor abgestimmt.”
- Fachbereichsbibliothek: “Wir besitzen zahlreiche Dokumente, für die wir in einer Kartei Titel, Autoren und Jahr speichern. Zur eindeutigen Zuordnung besitzt jedes Dokument eine eindeutige Signatur. Geleitet wird die Bibliothek von Mitgliedern der Universität - das ist auch die Personengruppe, an welche Dokumente ausgeliehen werden können.”
- Dozenten: “Wir empfehlen für unsere Vorlesungen oft begleitende Literatur. Das sind meist Bücher. Den Studenten nennen wir jeweils den Titel, die Autoren, den Verlag und das Erscheinungsjahr.”

## Fallbeispiel

---

### Prüfungsamt

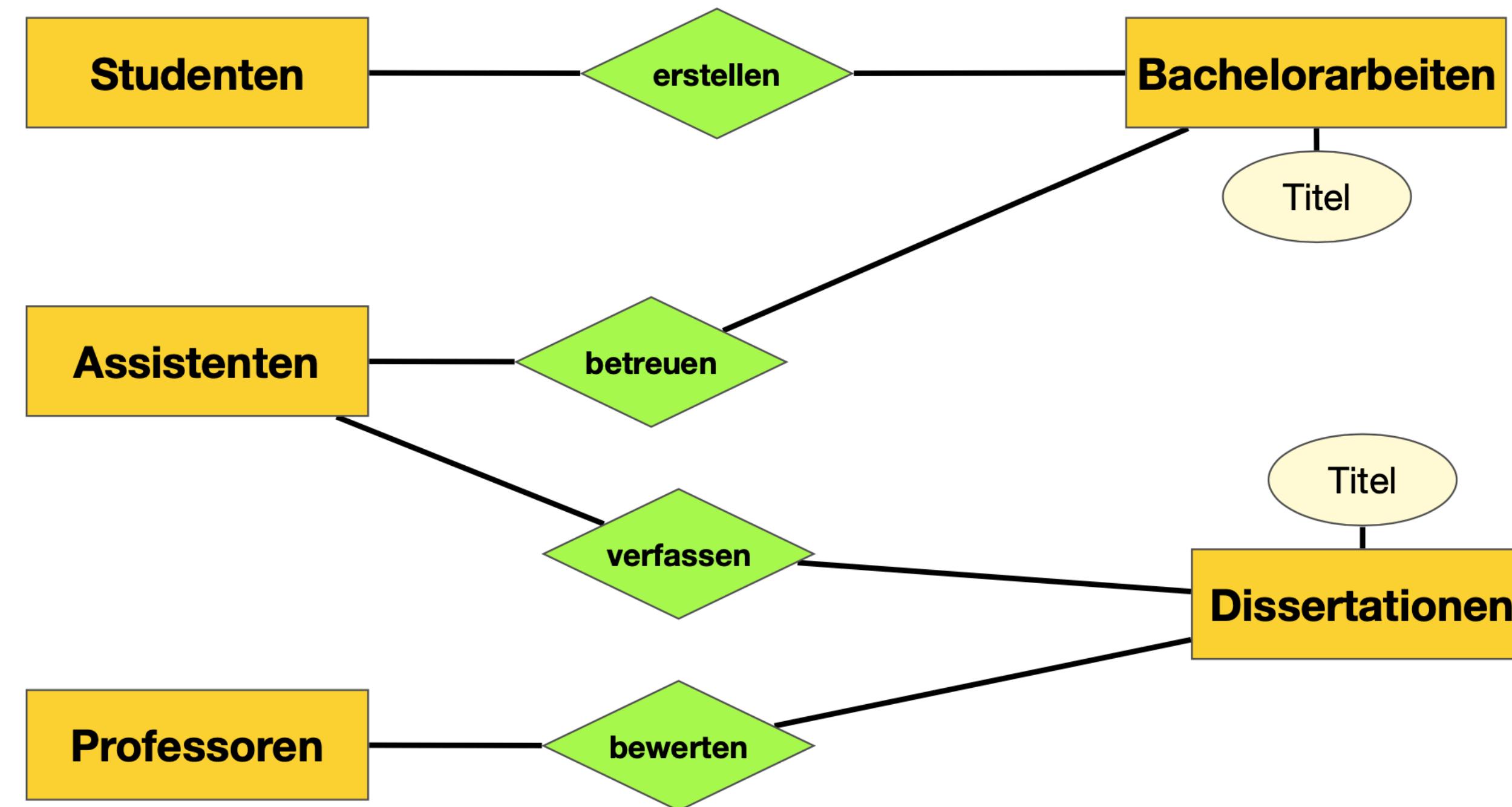


Diagramm Prof. Striegnitz  
Quelle Kemper/Eickler

## Fallbeispiel

---

### Fachbereichsbibliothek

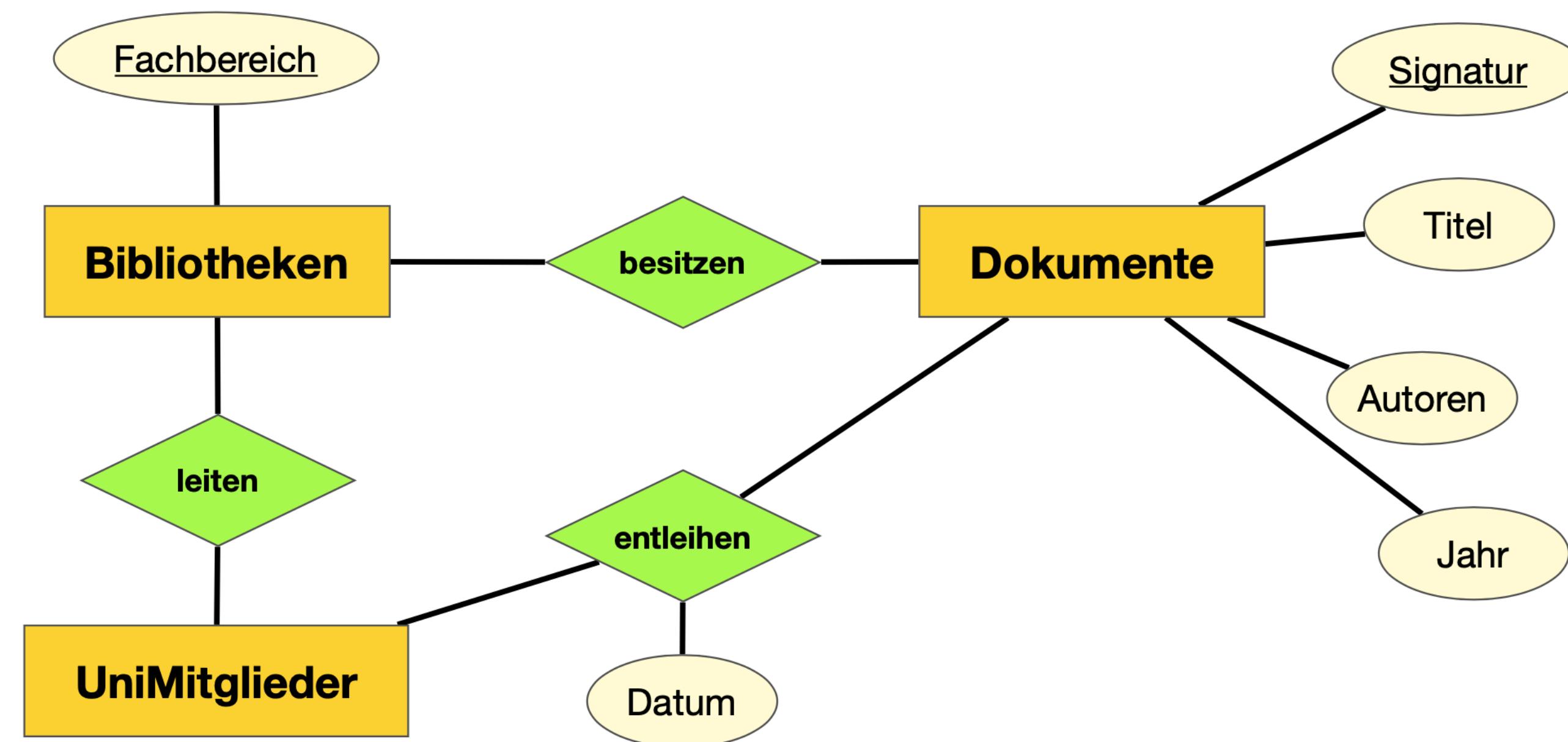


Diagramm Prof. Striegnitz  
Quelle Kemper/Eickler

## Fallbeispiel

---

### Dozenten

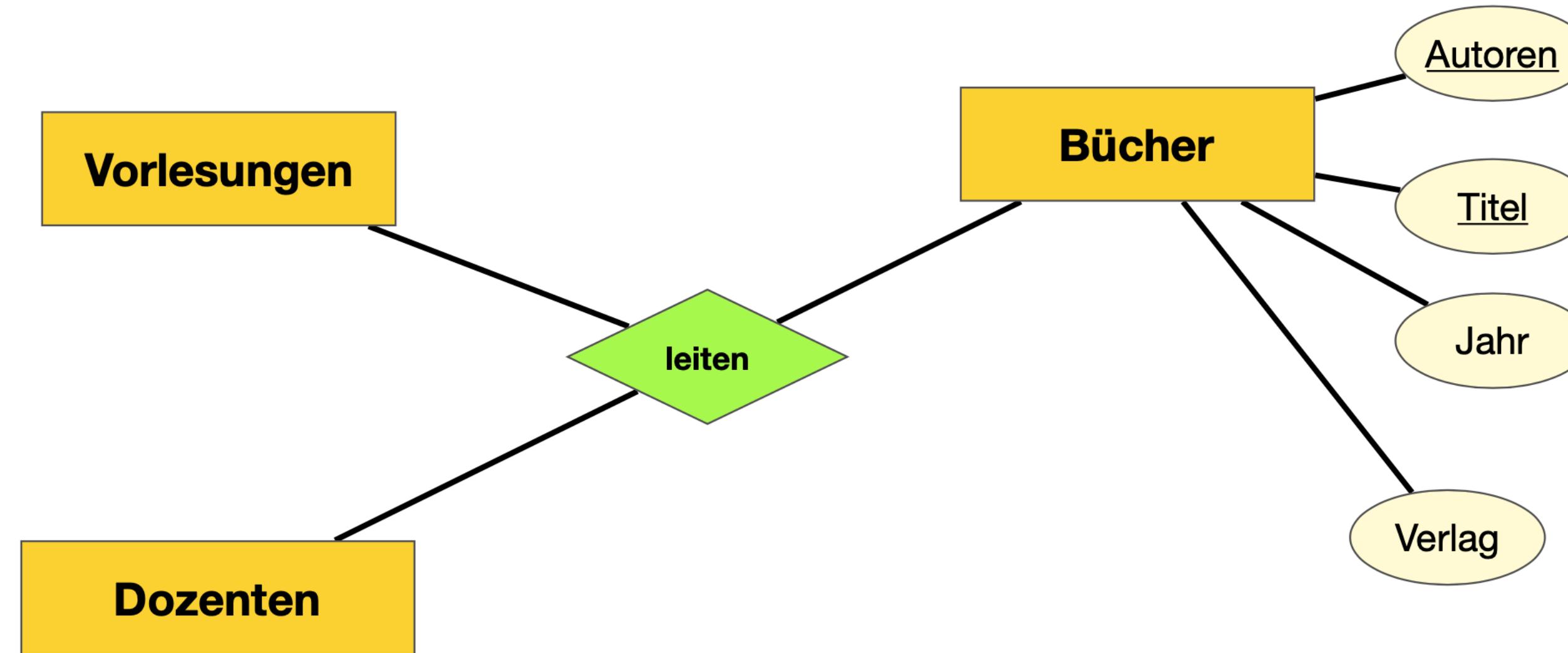


Diagramm Prof. Striegnitz  
Quelle Kemper/Eickler

## Fallbeispiel

---

### Beobachtungen I

- Begriffe *Dozenten* und *Professoren* sind synonym verwendet worden.
- Der Entitytyp *UniMitglieder* ist eine Generalisierung von *Studenten*, *Professoren* und *Assistenten*.
- Fakultätsbibliotheken werden von Angestellten geleitet.
- Die Beziehung *leiten* der Fachbereichsbibliothek ist revisionsbedürftig, sobald wir im globalen Schema ohnehin eine Spezialisierung von *UniMitglieder* in *Studierende* und *Angestellte* vornehmen - Studierende leiten keine Bibliotheken.
- Dissertationen, Diplomarbeiten und Bücher sind Spezialisierungen von Dokumenten, die in den Bibliotheken verwaltet werden.
- Wir können davon ausgehen, dass alle an der Universität erstellten Abschlussarbeiten und Dissertationen in Bibliotheken verwaltet werden.

## Fallbeispiel

---

### Beobachtungen II

- Die durch das Prüfungsamt festgelegten Beziehungen erstellen und verfassen modellieren denselben Sachverhalt wie das Attribut *Autoren* von *Büchern* der Sicht der Dozenten.
- Alle in einer Bibliothek verwalteten Dokumente werden durch eine Signatur identifiziert.



In einem konsolidierten Schema müssen die Begriffe eindeutig sein und es müssen sich die 'Partner' wiederfinden!

## Fallbeispiel

### Konsolidiertes Schema

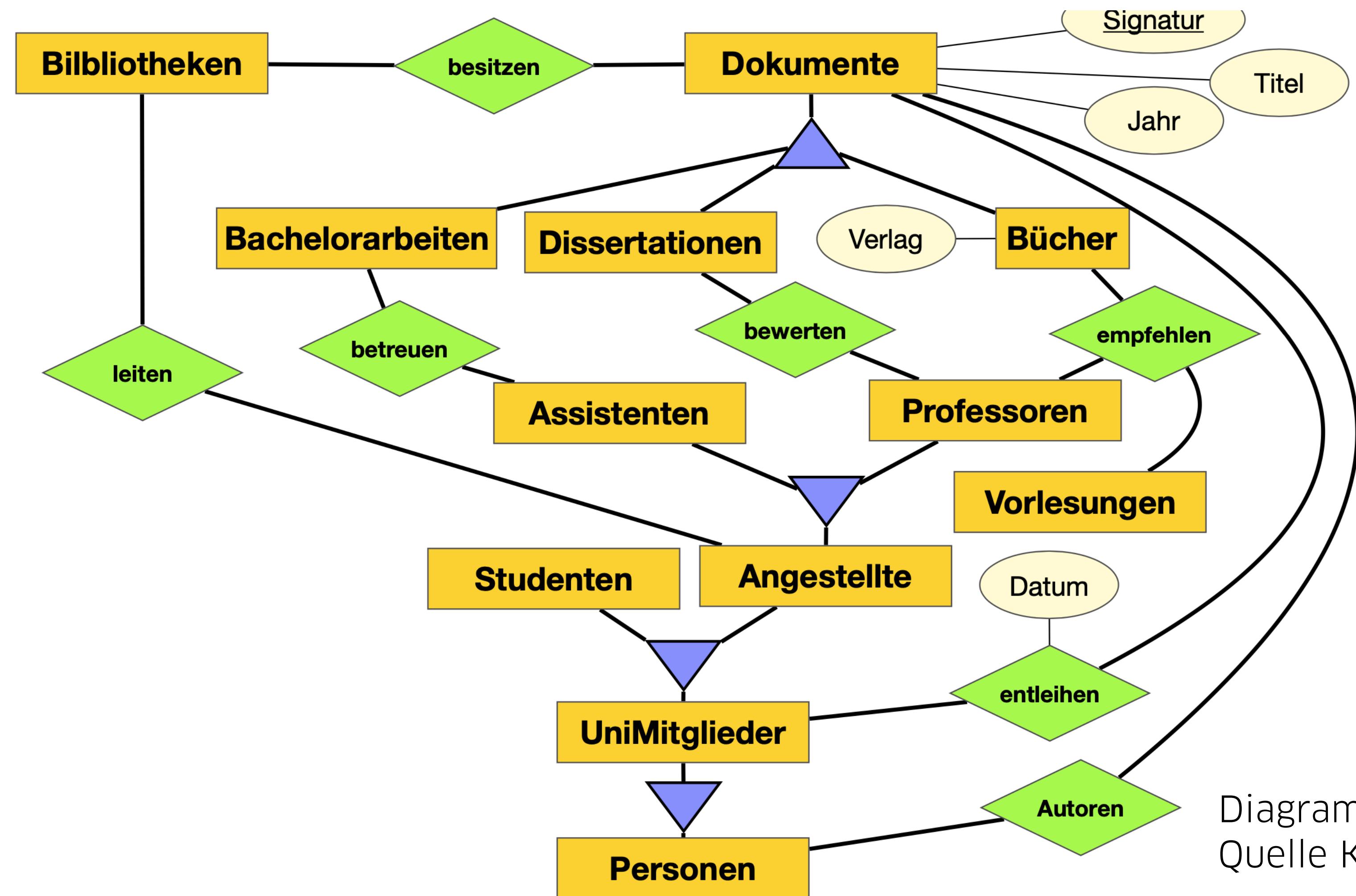


Diagramm Prof. Striegnitz  
Quelle Kemper/Eickler

## Vorschau

---

### **Relationen im RDBMS**

Bei der Umsetzung von 1:1, 1:N und N:M-Relationen eines ER-Diagramms (Modellierung Mini-Welt) in eine Datenbankstruktur gibt es, nimmt man mal hierarchische Strukturen aus, nicht so viele Möglichkeiten.

Diese sollen, für das Verständnis auch des folgenden Kapitels, im Folgenden kurz beispielhafte an Relationen aus `matse_mhist` vorgestellt werden. Hier ist noch wichtig, dass wir Entitäten immer über eine `id` identifizieren.

Hintergründe dann in den kommenden Kapiteln.

## Vorschau

---

### 1:N-Relation im RDBMS

- Eine Möglichkeit der Umsetzung sind zwei Tabellen für die Entitätstypen und die Realisierung der Relation als *Fremdschlüssel* in der Tabelle mit der N-Kardinalität !



	id	name
1	1	Mia
2	2	Ben
3	3	Emma
4	4	Paul
5	5	Hannah

mitarbeiter

	id	name	mitarbeiter_id
1	1	Petra	4
2	2	Mini	18
3	3	Rosa	<null>

tier

### Q&A

- Wieso ist der Fremdschlüssel beim Tier?

## Vorschau

### 1:N-Relation im RDBMS

- Eine andere Möglichkeit der Umsetzung sind zwei Tabellen für die Entitätstypen und eine eigene Tabelle für die Realisierung der Relation mit zwei Fremdschlüsseln.



	id	name
1	1	Vorstand
2	2	HR/Buchhaltung
3	3	Vertrieb
4	4	Marketing
5	5	Einkauf

abteilung

	abteilung_id	standort_id
1	1	1
2	2	1
3	3	2
4	3	1
5	3	3

sitzt\_am

	id	ort
1	1	Aachen
2	2	Jülich
3	3	Köln
4	4	Berlin

standort

### Q&A

- Hier gibt es einen Fehler...

## Vorschau

---

### N:M-Relation im RDBMS

- Die Relation enthält z.B. (Vorstand,Aachen) und (Buchhaltung, Aachen), d.h. ein Standort kann *mehrere* Abteilungen beherbergen → N:1, Richtig.
- Sie enthält aber auch (Vertrieb,Aachen) und (Vertrieb,Jülich)...  
→ N:M



	id	name
1	1	Vorstand
2	2	HR/Buchhaltung
3	3	Vertrieb
4	4	Marketing
5	5	Einkauf

abteilung

	abteilung_id	standort_id
1	1	1
2	2	1
3	3	2
4	3	1
5	3	3

sitzt\_am

	id	ort
1	1	Aachen
2	2	Jülich
3	3	Köln
4	4	Berlin

standort

## Vorschau

### N:M-Relation im RDBMS

- Die einzige Möglichkeit der Umsetzung sind zwei Tabellen für die Entitätstypen und eine eigene Tabelle für die Realisierung der Relation mit zwei Fremdschlüssen.



	id	vom
1	1	2014-09-12 00:00:00
2	2	2014-08-30 00:00:00
3	3	2014-09-11 00:00:00

bestellung

	bestellung_id	produkt_id
1	2	11
2	1	15
3	3	15
4	1	16
5	3	17

besteht\_aus

	id	bezeichnung
3	3	Spinatpizza
4	4	Fischstäbchen
5	5	Nudelpfanne
6	6	Möhren

produkt

## Vorschau

---

### 1:1-Relation im RDBMS

- Eine Möglichkeit der Umsetzung ist über eine Fremdschlüsselbeziehung in *einer der beiden Tabellen* (wie im ersten Beispiel).
- Oder über eine eigene Relationen-Tabelle (wie im letzten Beispiel). Hier darf aber natürlich dann jede Entität nur einmal vorkommen!
- In `matse_mhist` kommt keine 1:1-Relation vor, da man die Attribute auch immer zu einem Entitätstyp zusammenfassen kann, wenn es inhaltlich gerechtfertigt ist.

### Zusammenfassung der Grundidee

- Die Fälle 1:1 und 1:N (N:1) können über eine eigene Relationen-Tabelle abgebildet werden oder nicht. Eine N:M-Relation benötigt immer eine eigene Relationen-Tabelle.
- Vor- und Nachteile besprechen wir noch.

# UNIT 0x05

## RELATIONALES MODELL

## RELATIONALE ALGEBRA I

## Motivation/Erinnerung

---

### Aufgabe

- Entwicklung einer 'guten' Datenbank zu einem realen Problem (z.B. Mini-Welt), d.h. u.a. vollständig, korrekt, minimal, aber auch redundanzfrei und effizient...

### Vorgehen

- Konsolidierung und Konzeptueller Entwurf in Form eines ER-Modells ✓ (Beispiel)
- Überführung in eine 'gute' Datenbank... dazu benötigen wir:
  - Implementationsentwurf, z.B. mit Tabellen und Relationen  
→ Relationales Modell
  - Mathematisches Modell der Datenbankoperationen, z.B. zur Anfrageoptimierung  
→ Relationale Algebra
  - Gütemaß für den Implementationsentwurf  
→ Normalformen

# Wiederholung Modellierung

---

## Aufgabe GaiaZOO

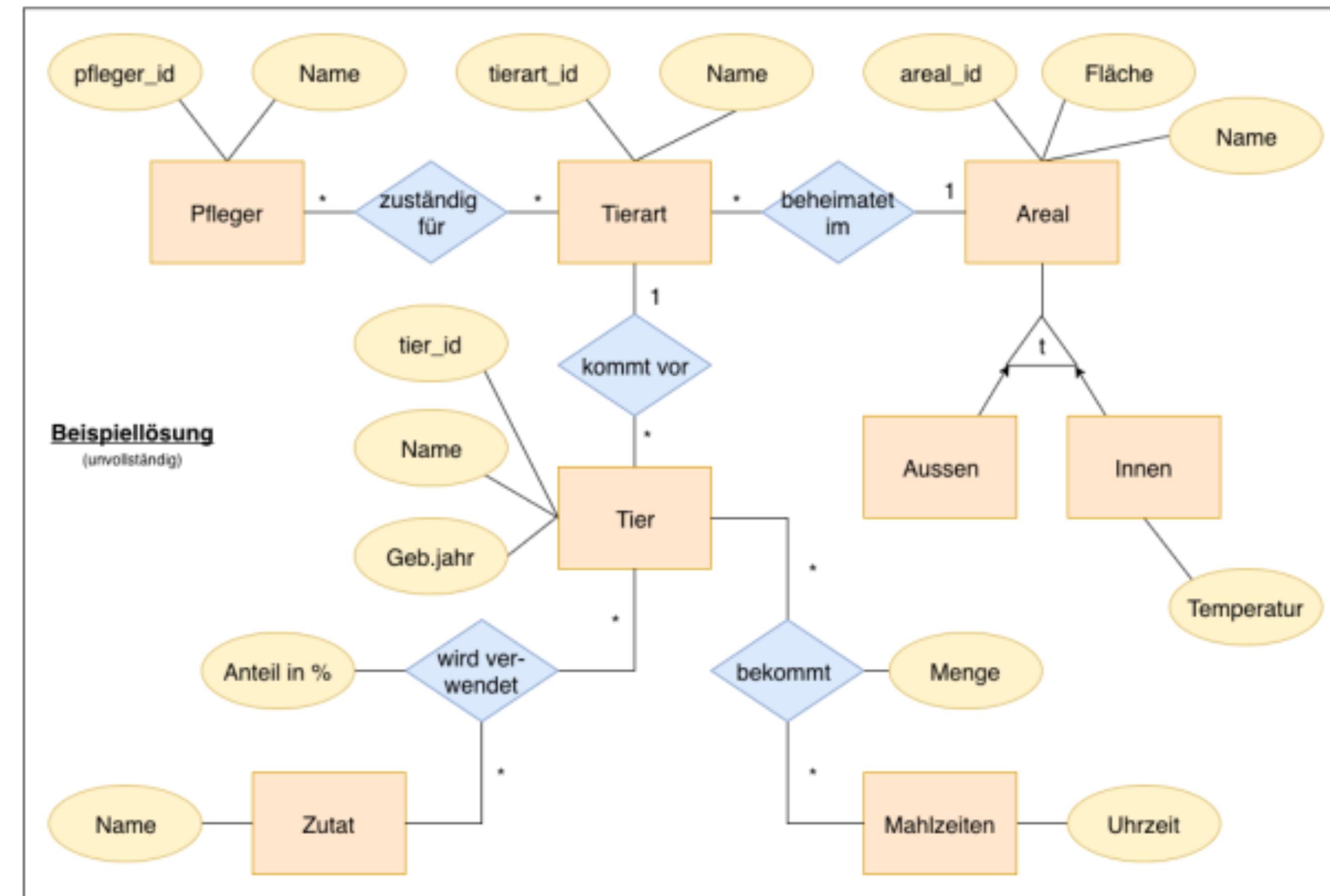
[...] Er beherbergt verschiedene Tierarten, die von Pflegern betreut werden. Alle Tiere einer Art befinden sich in genau einem ganz bestimmten Areal im Zoo. Dort können aber natürlich mehrere Tierarten leben. Aufgrund unterschiedlicher Allergien bekommt jedes Tier eine individuelle fest vorgegebene Futtermischung mit bestimmten Zutaten. Weiter gilt:

- Pfleger kümmern sich nicht um einzelne Tiere, sondern allgemein um alle Tiere einer Tierart. D.h. Ihnen sind bestimmte Tierarten zugeordnet. In der Regel sind das je Tierart mindestens zwei.
- Falls das letzte Tier einer Tierart sterben sollte, wird umgehend ein neues Tier dieser Art erworben. Dennoch kommt es vor, dass zeitweilig ein Gehege auch leer sein kann.
- Die Areale sind in genau zwei Kategorien eingeteilt: Innen- und Aussengehege. Für jedes Areal ist die Fläche in m<sup>2</sup> wichtig, aber die Innengehege besitzen ausserdem noch eine Temperaturvorgabe. Vereinzelt kommt es vor, dass Tiere innen und aussen leben können.
- Für die Zutaten einer individuellen Futtermischung eines Tieres ist gewünscht, sie in Prozent der Gesamtmenge einer Mahlzeit anzugeben. Jedes Tier bekommt verschiedene Mahlzeiten am Tag, wobei dann eine konkrete Mengenangabe in kg je Mahlzeit ausreicht, um die Mahlzeit zusammenzustellen. Mögliche Mahlzeiten sind Morgens 6 Uhr, Mittags 12 Uhr, Abends 18 Uhr und gegen Mitternacht 24 Uhr.

## Wiederholung Modellierung

---

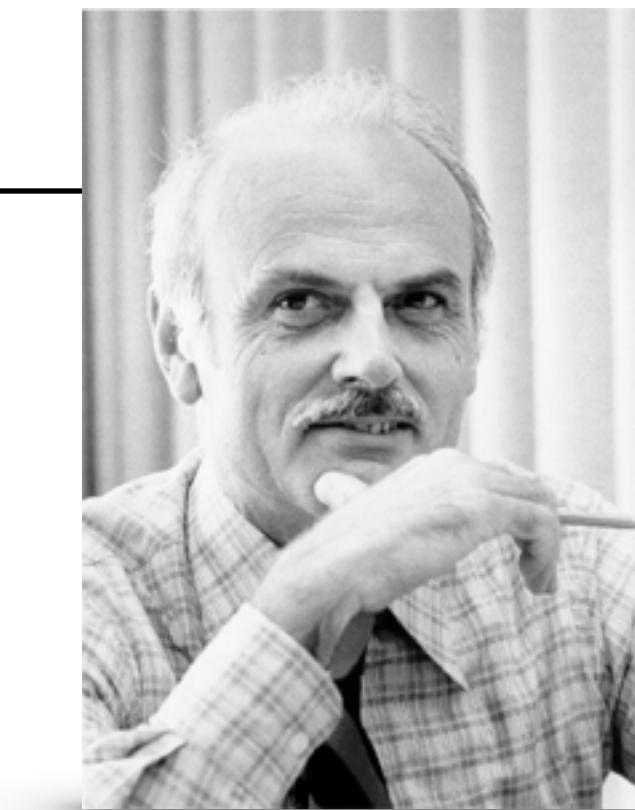
### Lösungsvorschlag GaiaZoo



# Historische Entwicklung

## Entwickelt von Edgar (Ted) Codd, IBM

- Relational Model of Data for Large Shared Data Banks,  
Comm. ACM, Juni 1970
- Formale Grundlage DBMS, Relationale Algebra, Normalform



### Information Retrieval

#### A Relational Model of Data for Large Shared Data Banks

E. F. CODD

IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation

**2.1.2. Projection.** Suppose now we select certain columns of a relation (striking out the others) and then remove from the resulting array any duplication in the rows. The final array represents a relation which is said to be a *projection* of the given relation.

A selection operator  $\pi$  is used to obtain any desired permutation, projection, or combination of the two operations. Thus, if  $L$  is a list of  $k$  indices<sup>7</sup>  $L = i_1, i_2, \dots, i_k$  and  $R$  is an  $n$ -ary relation ( $n \geq k$ ), then  $\pi_L(R)$  is the  $k$ -ary relation whose  $j$ th column is column  $i_j$  of  $R$  ( $j = 1, 2, \dots, k$ ) except that duplication in resulting rows is removed. Consider the relation *supply* of Figure 1. A permuted projection of this relation is exhibited in Figure 4. Note that, in this particular case, the projection has fewer  $n$ -tuples than the relation from which it is derived.

**2.1.3. Join.** Suppose we are given two binary relations, which have some domain in common. Under what circumstances can we combine these relations to form a

<sup>7</sup> When dealing with relationships, we use domain names (role-qualified whenever necessary) instead of domain positions.

$R * S$	(supplier)	part	project
1	1	1	1
1	1	1	2
2	1	1	1
2	1	1	2
2	2	2	1

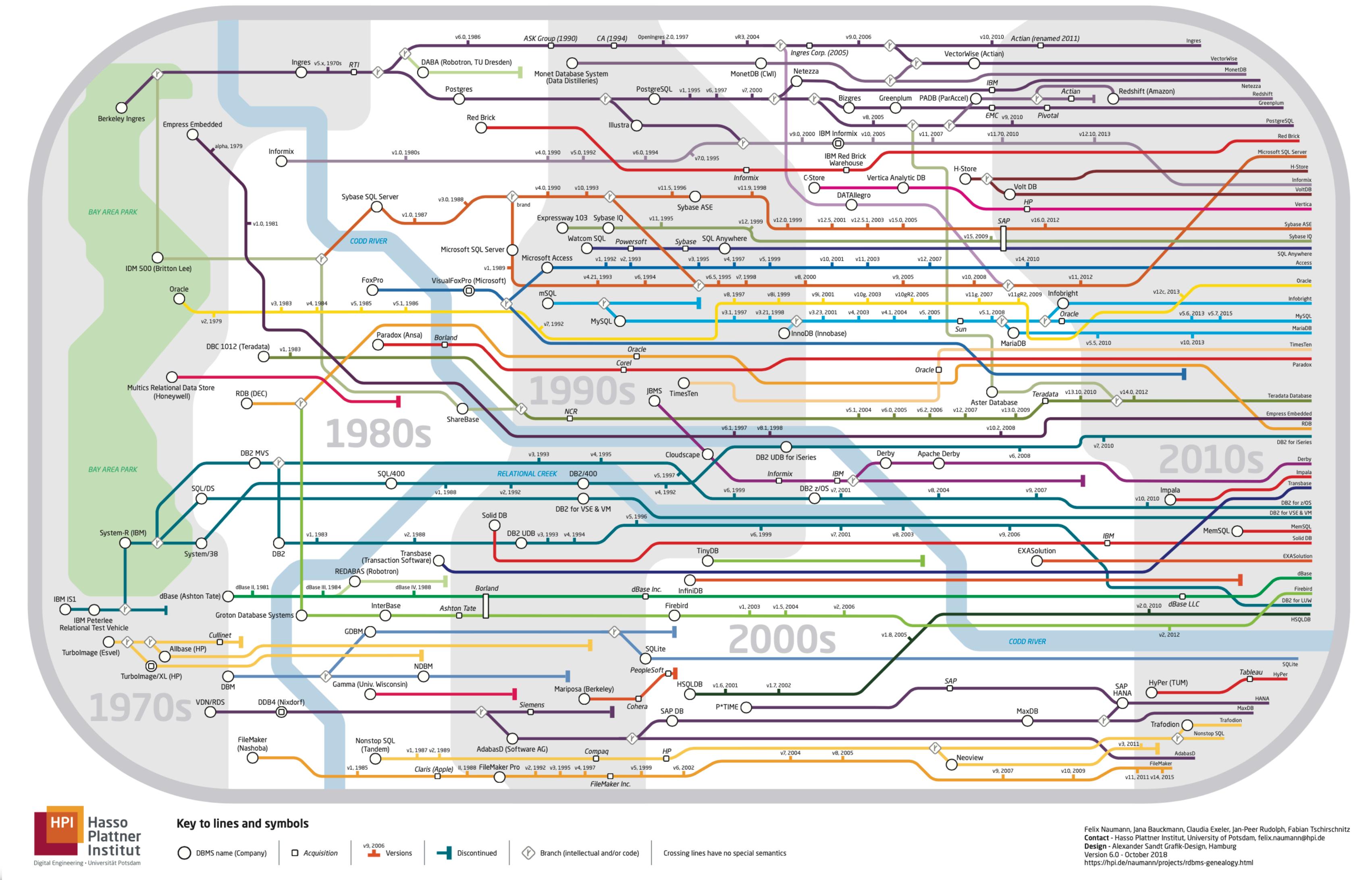
FIG. 6. The natural join of  $R$  with  $S$  (from Figure 5)

$U$	(supplier)	part	project
1	1	1	2
2	1	1	1
2	2	2	1

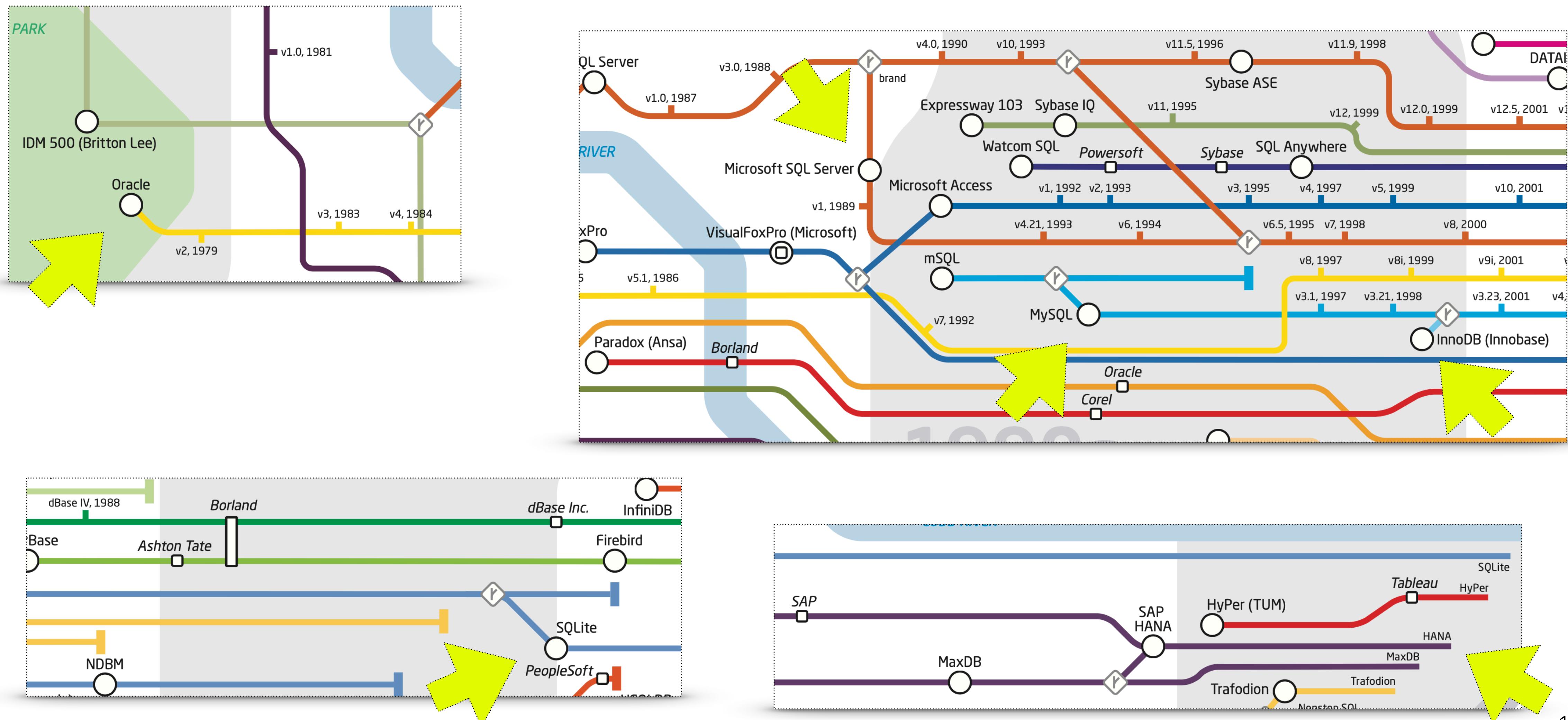
FIG. 7. Another join of  $R$  with  $S$  (from Figure 5)

Inspection of these relations reveals an element (element 1) of the domain *part* (the domain on which the join is to be made) with the property that it possesses more than one relative under  $R$  and also under  $S$ . It is this ele-

# Historische Entwicklung - Genealogy of RDBMS



# Historische Entwicklung - Genealogy of RDBMS



## ER-Modell → Relationales Modell

---

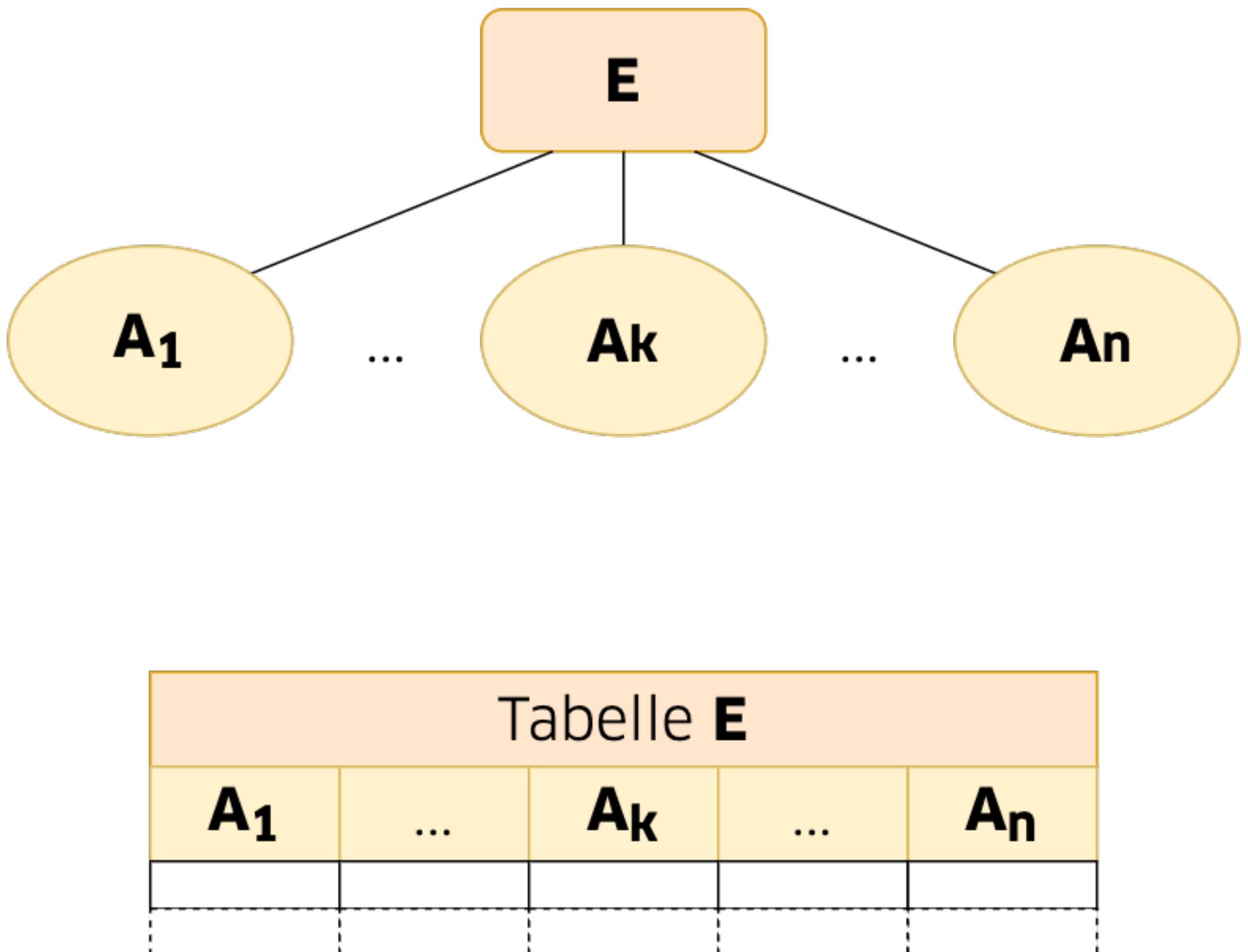
### Überführung der Elemente im ER-Diagramm in Datenbankstrukturen

- Entitätstypen
- Attribute
  - zusammengesetzt
  - mehrwertig
- Relationen
  - 1:1
  - 1:N
  - N:M
- Hierarchien (später)

## ER-Modell → Relationales Modell

### Entitätstyp E: {[A<sub>1</sub>:D<sub>1</sub>,...,A<sub>n</sub>:D<sub>n</sub>]}

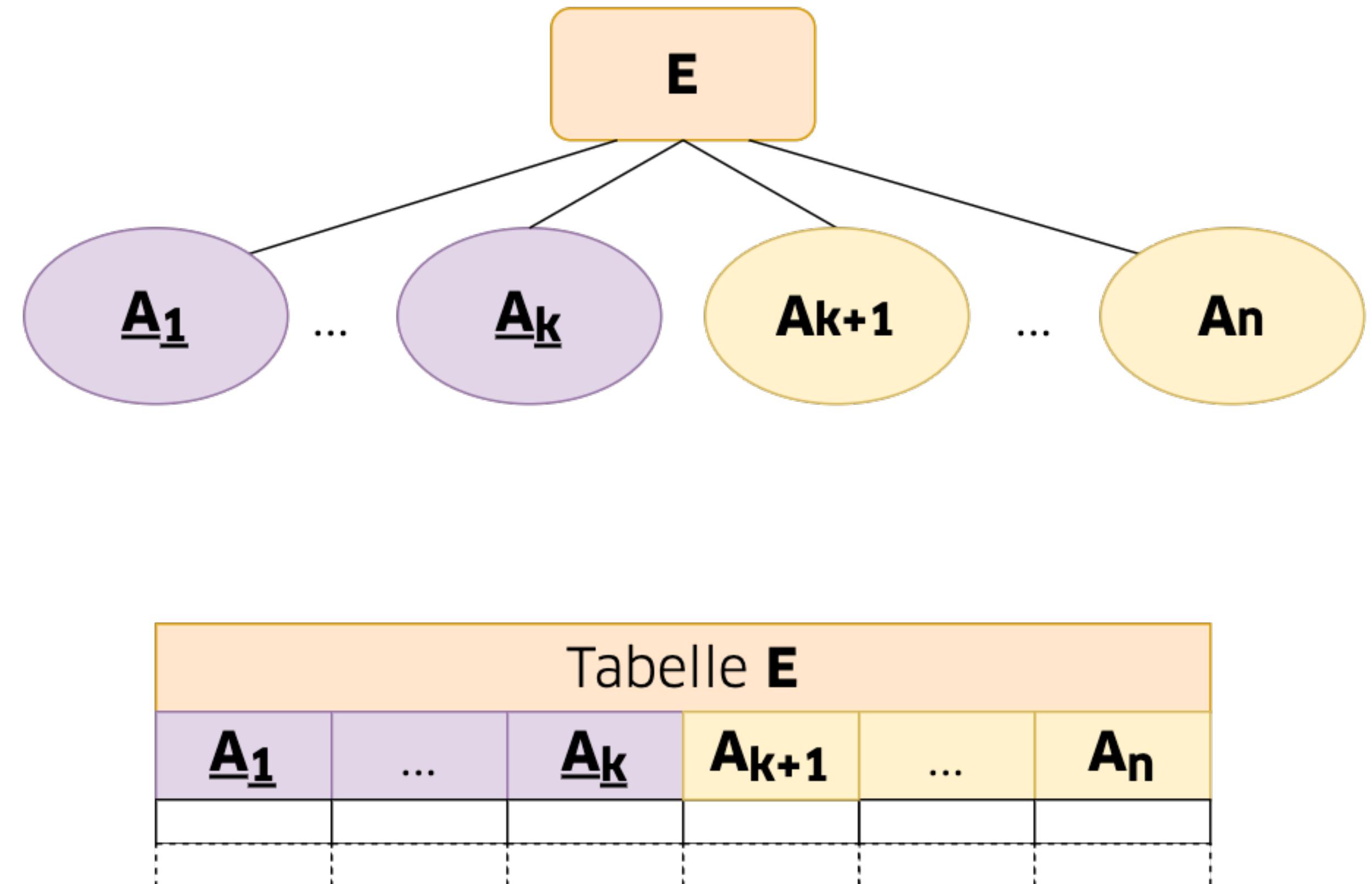
- E wird zur Tabelle E.
- Attribute A<sub>k</sub> sind Spalten mit Datentyp D<sub>k</sub>.
  - Alle Attribute sind *atomar*, also weder zusammengesetzt noch mehrwertig.  
Letztere werden in eine andere Darstellung überführt (folgt).
  - Schlüsselattribute werden zu 'primary keys' (folgt).
- Die Reihenfolge der Zeilen und Spalten ist nicht relevant.
- Enthaltene Informationen werden nur durch Datenwerte ausgedrückt.



## ER-Modell → Relationales Modell

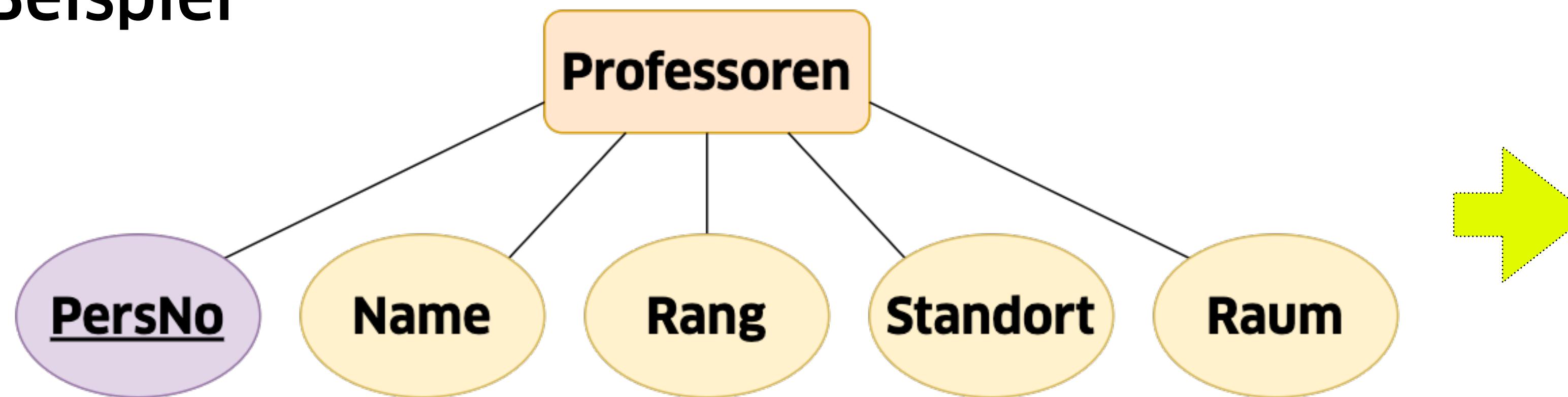
### (Primär)Schlüsselattribute

- Zeile/Tuple/Entität in der Tabelle beschreibt ein eindeutiges Objekt!
  - Achtung: Technisch sind Tabellen ohne Schlüsselattribut und Einträge(Zeilen), die in allen Werten identisch sind, denkbar... aber
  - erst die Verwendung eindeutiger Schlüsselattribute garantiert die Identifikation unterschiedlicher Objekte, selbst bei gleichen Werten der restlichen Attribute!
- Schlüsselattribute werden 'primary keys'.



## ER-Modell → Relationales Modell

### Beispiel



### Tabellenstruktur

Table:					Comment
Professoren					
Columns (5)	Keys (2)	Indices (1)	Foreign Keys		
PersNr int(11) - part of primary key					
Name varchar(30)					
Rang varchar(10)					
Standort varchar(20)					
Raum int(11)					

Tabellenstruktur  
{ [PersNo:int,  
Name:varchar,  
Rang:varchar,  
Standort:varchar,  
Raum:varchar] }

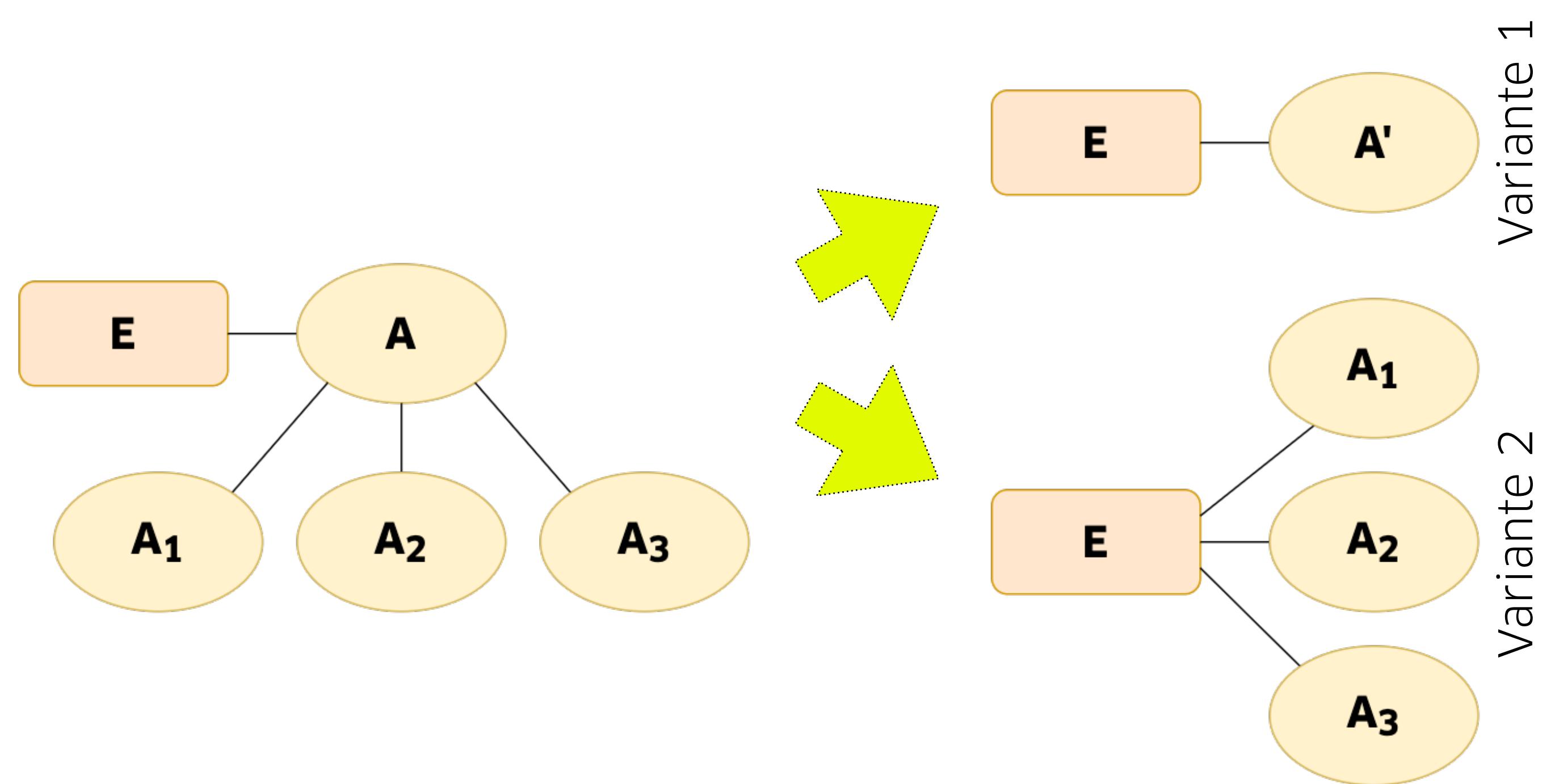
PersNr	Name	Rang	Standort	Raum
2125	Sokrates	C4	Jülich	226
2126	Russel	C4	Jülich	232
2127	Kopernikus	C3	Aachen	310
2133	Popper	C3	Aachen	52
2134	Augustinus	C3	Aachen	309
2136	Curie	C4	Jülich	36

Entitäten

## ER-Modell → Relationales Modell

### Zusammengesetzte Attribute

- Die Informationen von A liegen in den  $A_k$ , d.h. A enthält selber keine Informationen. Deswegen kann das Modell in eine der zwei Möglichkeiten mit nur *atomaren* Attributen überführt werden:
  - Zusammenfassung der  $A_k$  zu einem neuen gemeinsamen Attribut  $A'$ .
  - 'Umhängen' der Attribute  $A_k$  direkt an den Entitätstyp.



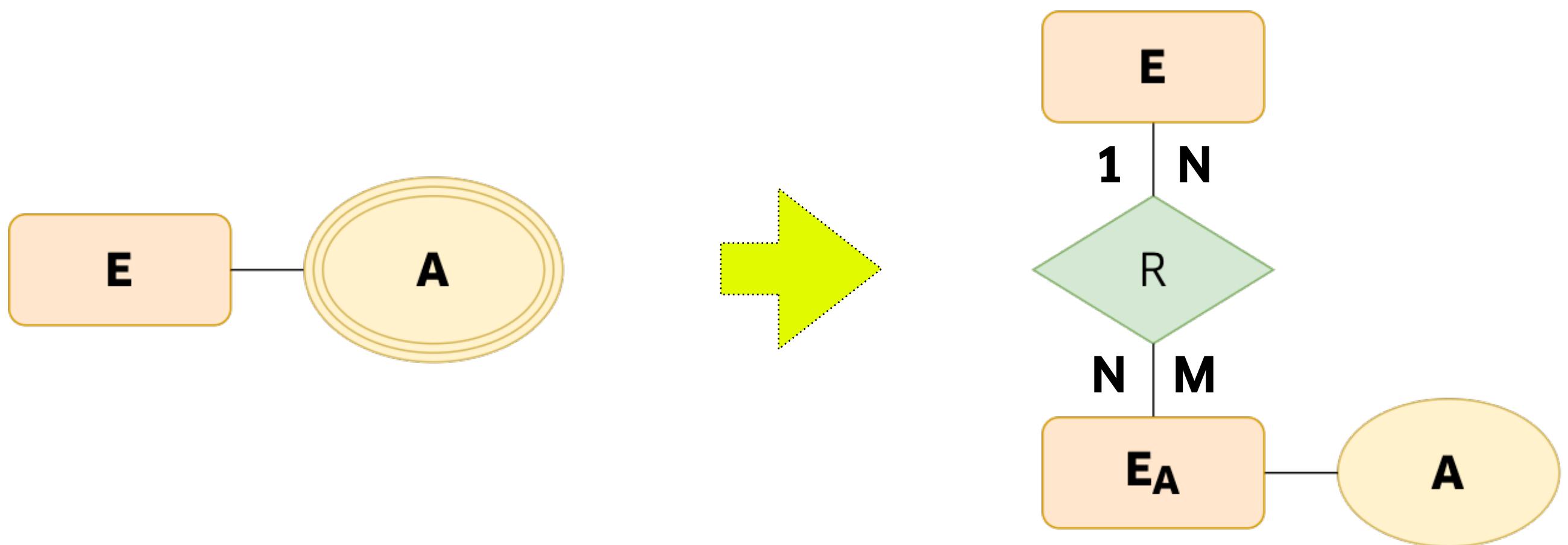
### Q&A

- Pro/Con der Varianten?

## ER-Modell → Relationales Modell

### Mehrwertige Attribute

- Die verschiedenen Ausprägungen von A werden üblicherweise in eine 1:N-Relation mit einem weiteren Entitätstyp  $E_A$  abgebildet. Eine allgemeinere Rolle von  $E_A$  und eine N:M-Relation ist auch denkbar (siehe Beispiel).
- Schlüsselattribute von E können so auch zu Schlüsselattributen von  $E_A$  werden. Das hängt von der genauen Umsetzung der Relation ab (folgt).



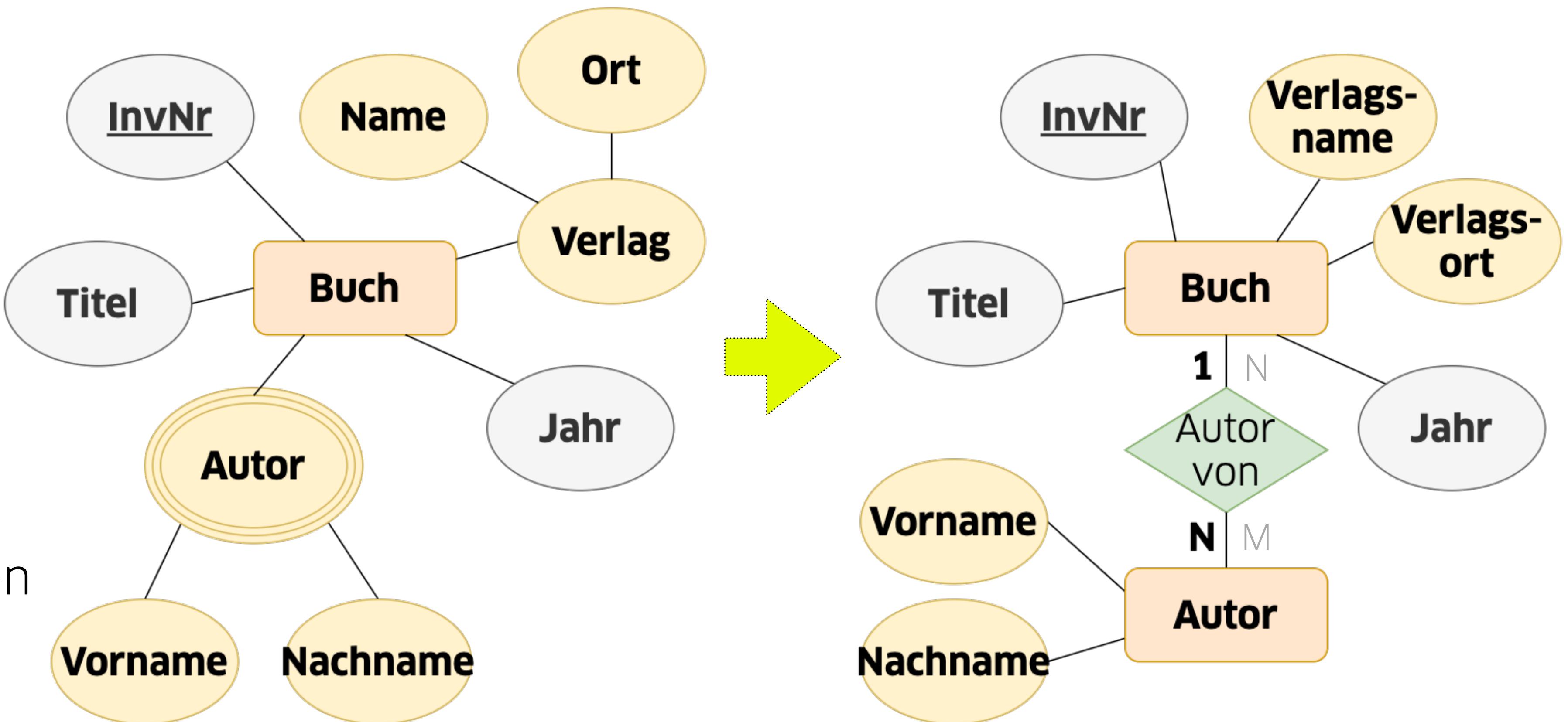
**Q&A**

- Pro/Con?

## ER-Modell → Relationales Modell

## Beispiel

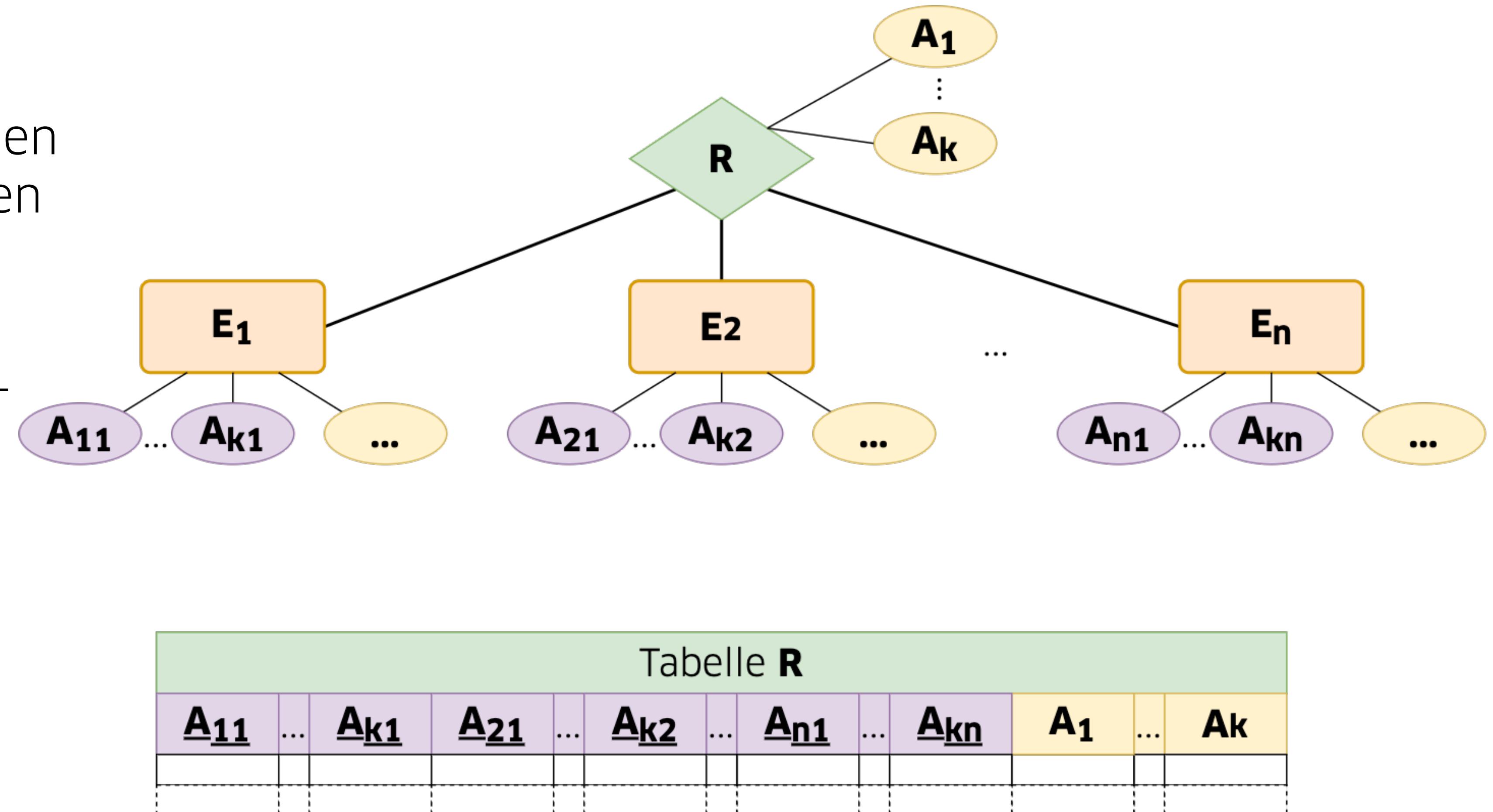
Hier ist auch eine allgemeinere Rolle von 'Autor', also eine N:M-Relation möglich, denn ein Autor kann viele Bücher schreiben. Das hätte in der Modellierung dann auch schon auffallen sollen...



## ER-Modell → Relationales Modell

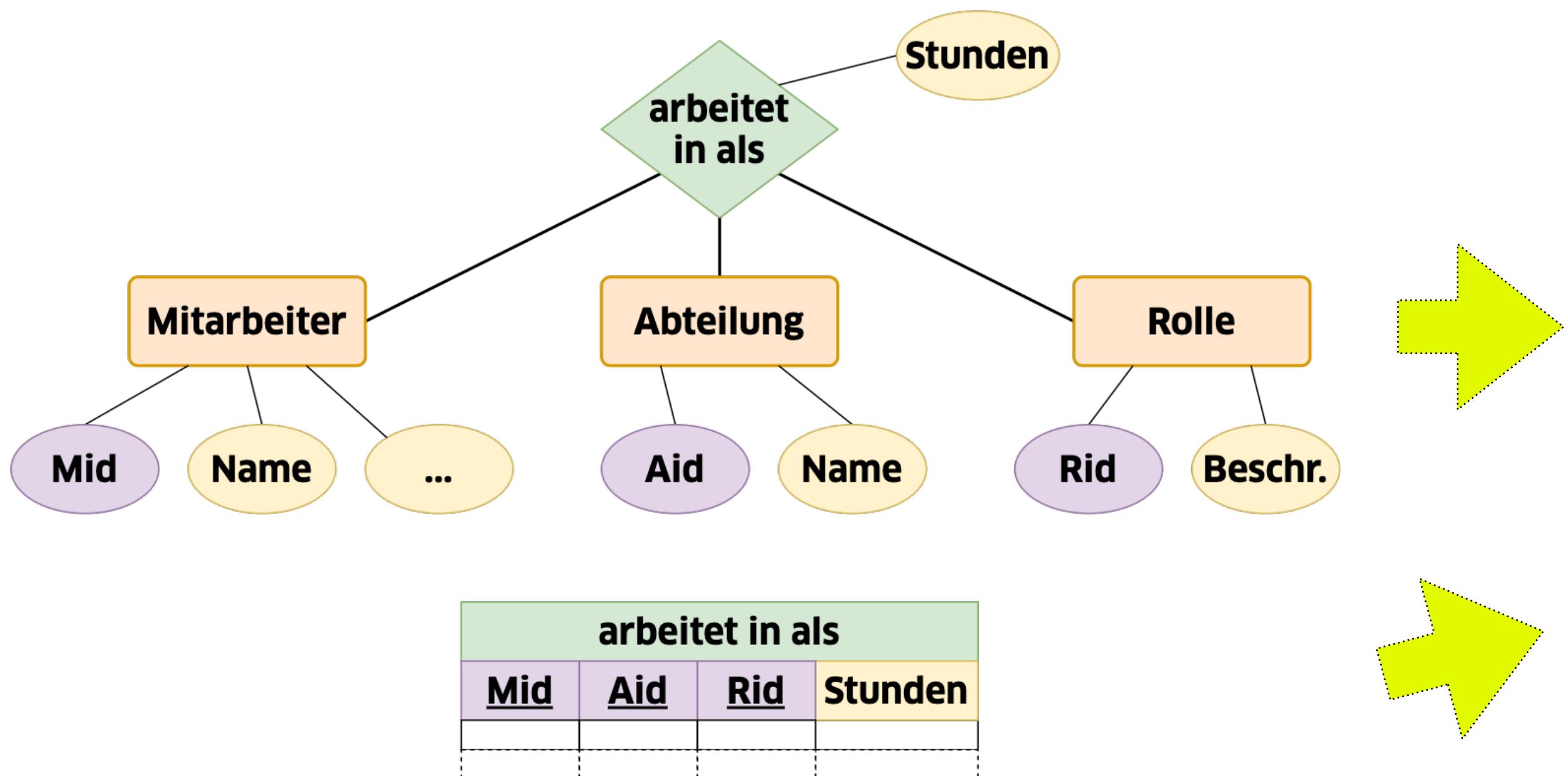
**N:M-Relationen**

- N:M-Relationen werden *immer* in einer eigenen Tabelle R realisiert.
- Diese Tabelle R enthält *alle* Schlüsselattribute der  $E_k$ , die vereint Schlüsselattribute von R sind ('Fremdschlüssel'),
- sowie die Attribute von R.



## ER-Modell → Relationales Modell

### Beispiel 3-stellige N:M-Relation



The relational database schema consists of four tables:

- mitarbeiter** (Employee): id, name, jahresgehalt
- abteilung** (Department): id, name
- rolle** (Role): id, beschreibung
- arbeitet\_in\_als** (Works In As): mitarbeiter\_id, abteilung\_id, rolle\_id, wochenstunden

Data extracted from the tables:

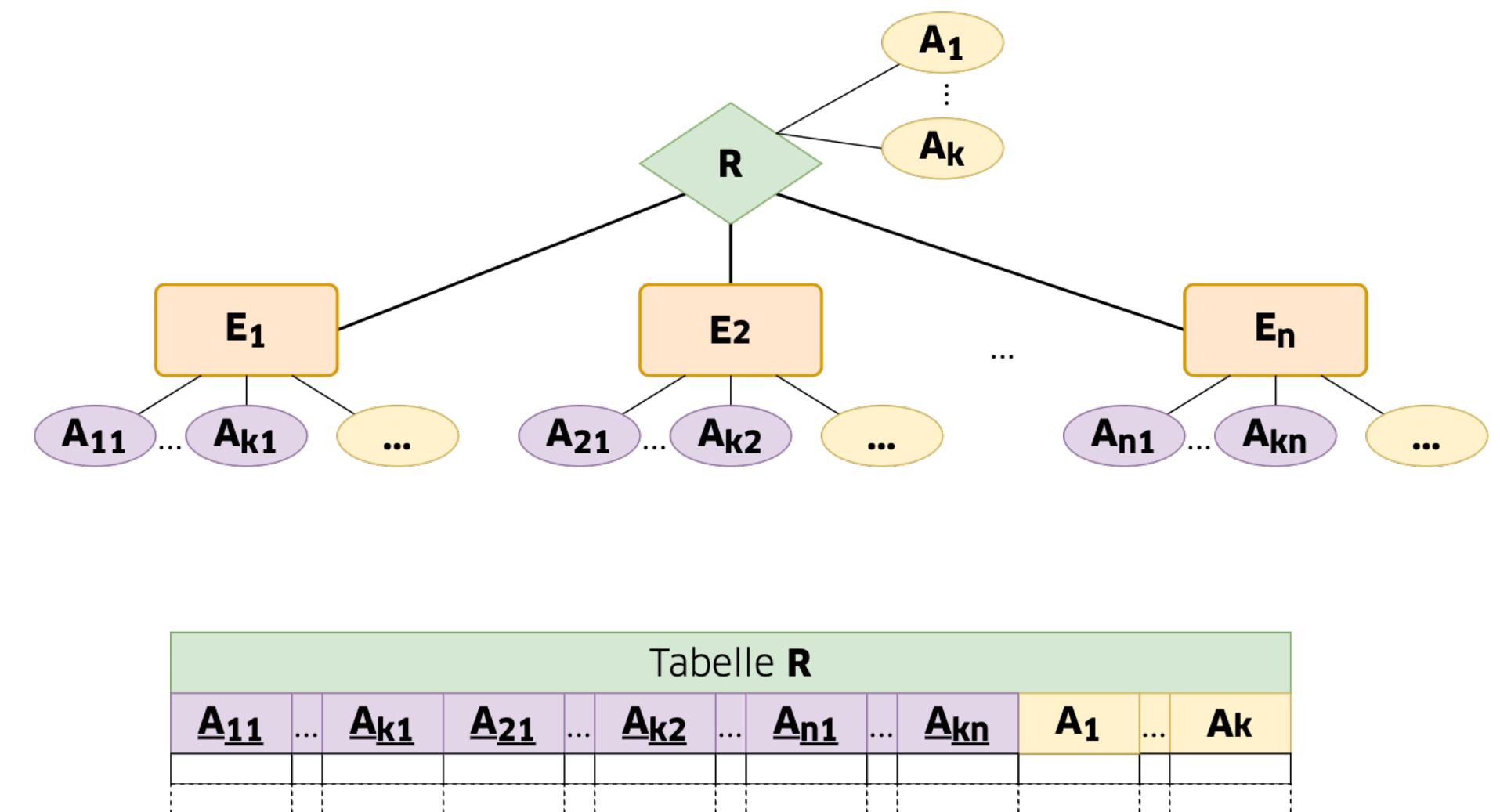
mitarbeiter	abteilung	rolle	arbeitet_in_als
1 Mia 110000.00	1 Vorstand	3 Leitung	1 1 3 10.00
2 Ben 90000.00	2 HR/Buchhaltung	4 Mitarbeiter	1 1 4 5.00
3 Emma 70000.00	3 Vertrieb	7 Student	1 2 3 10.00
4 Paul 5000.00	4 Marketing	8 Auszubildender	1 2 4 15.00
5 Hannah 5000.00	5 Einkauf	9 Schüler	2 1 4 5.00

'Mia' (id=1) arbeitet im Vorstand (id=1) in der Leitung (id=3).

## ER-Modell → Relationales Modell

### Anmerkungen zu N:M-Relationen

- 1:1- und 1:N-Relationen können auch über eine eigene Tabelle realisiert werden, aber
- Relationen so abzubilden verursacht Kosten!
- Die Kombination der Schlüsselattribute ist natürlicherweise ein eindeutiger Schlüssel für R. Es ist aber auch möglich, ein künstliches Schlüsselattribut ('id') zu verwenden.



### Q&A

- Pro/Con einer Relationen-Tabelle?
- Pro/Con eines künstlichen Schlüsselattributs?

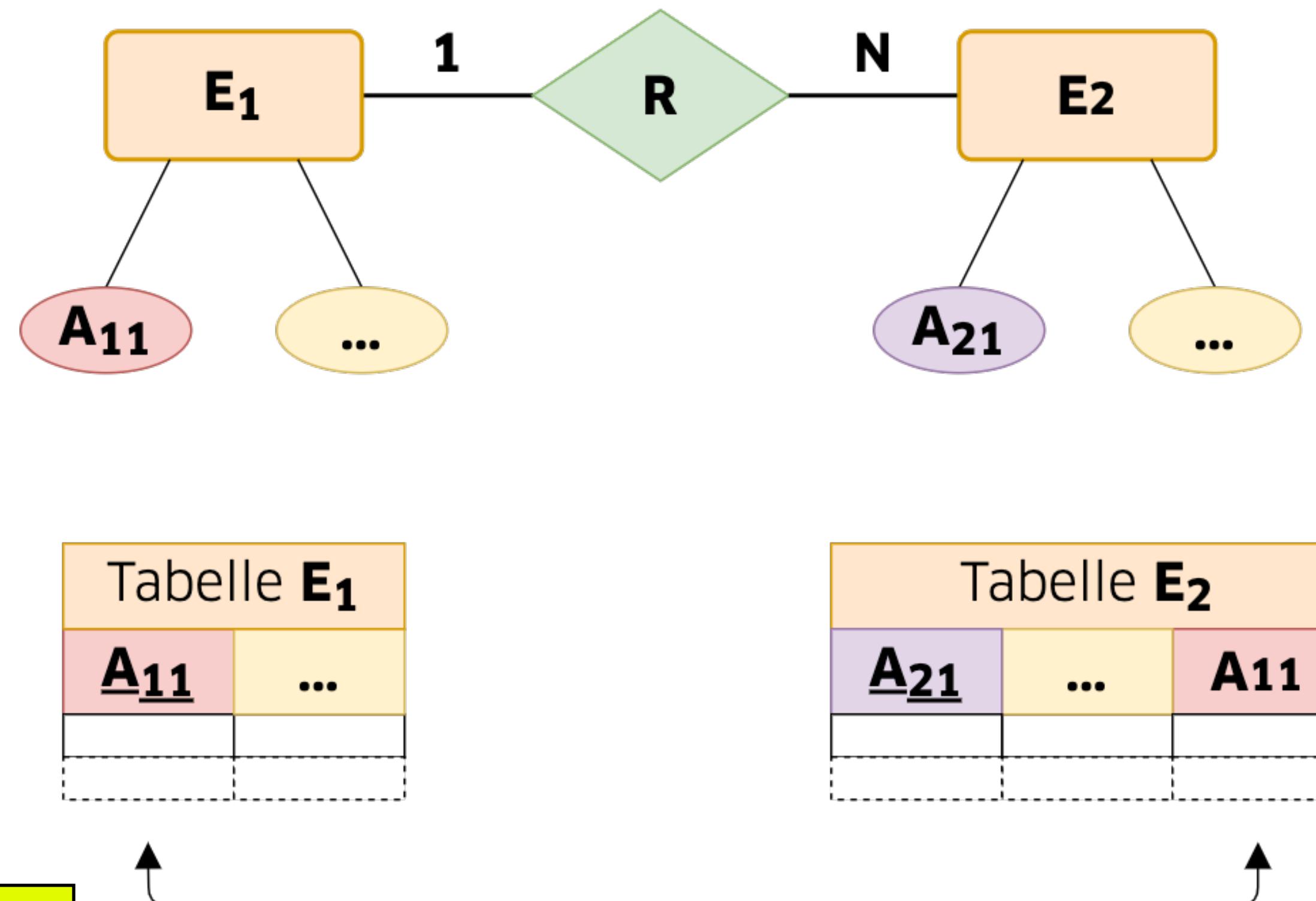
## ER-Modell → Relationales Modell

**1:N-Relationen**

- Da wir eine partielle Funktion  $f_2:E_2 \rightarrow E_1$  haben (Abb. auf 1), kann das Schlüsselattribut als sogenannter 'Fremdschlüssel' in  $E_2$  als weiteres Attribut eingebettet werden.
- Dort ist es *kein* Schlüssel von  $E_2$ , es referenziert nur die assoziierte Entität in  $E_1$ .

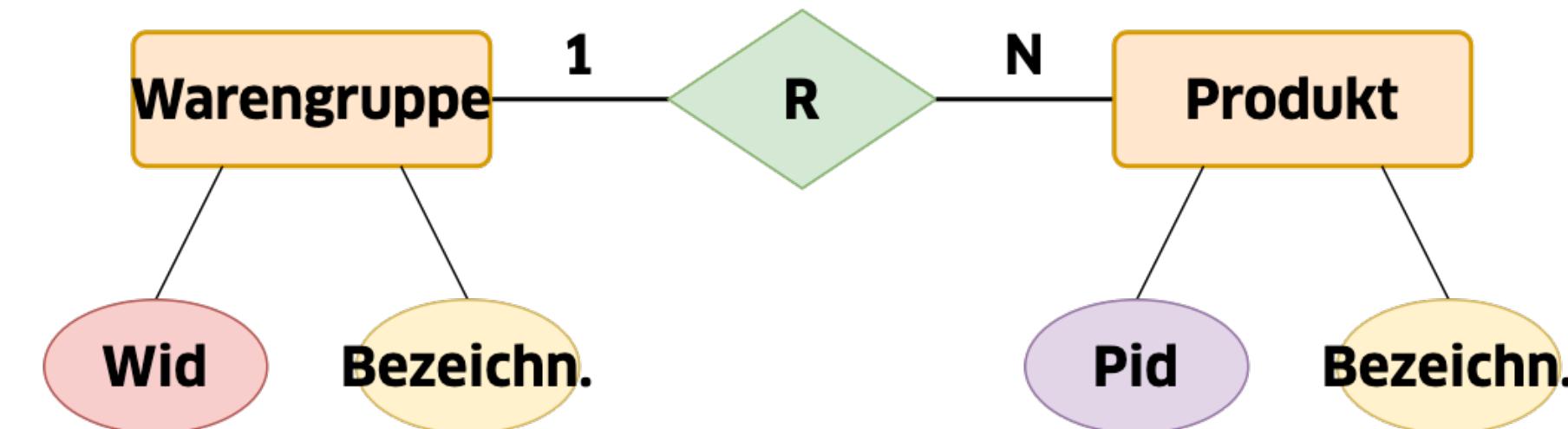


Es funktioniert nur mit Fremdschlüssel in  $E_2$ !  
Umgekehrt müsste man 'viele' Fremdschlüssel in  $E_1$  ablegen... Widerspruch zur Atomarität!



## ER-Modell → Relationales Modell

## Beispiel 1:N-Relation



Warenguppe	
<u>Wid</u>	...

Produkt		
<u>Pid</u>	...	<u>Wid</u>

<u>id</u>	<u>bezeichnung</u>
1	TK
2	Obst, Gemüse
3	Wurst, Aufschnitt
4	Milchprodukte
5	Kaltgetränke

warengruppe

<u>id</u>	<u>bezeichnung</u>	<u>stueckpreis</u>	<u>warengruppe_id</u>
3	Spätzlepizza	2.27	1
4	Fischstäbchen	1.99	1
5	Nudelpfanne	3.29	1
6	Möhren	0.39	2
7	Zwiebeln	0.29	2

produkt

## ER-Modell → Relationales Modell

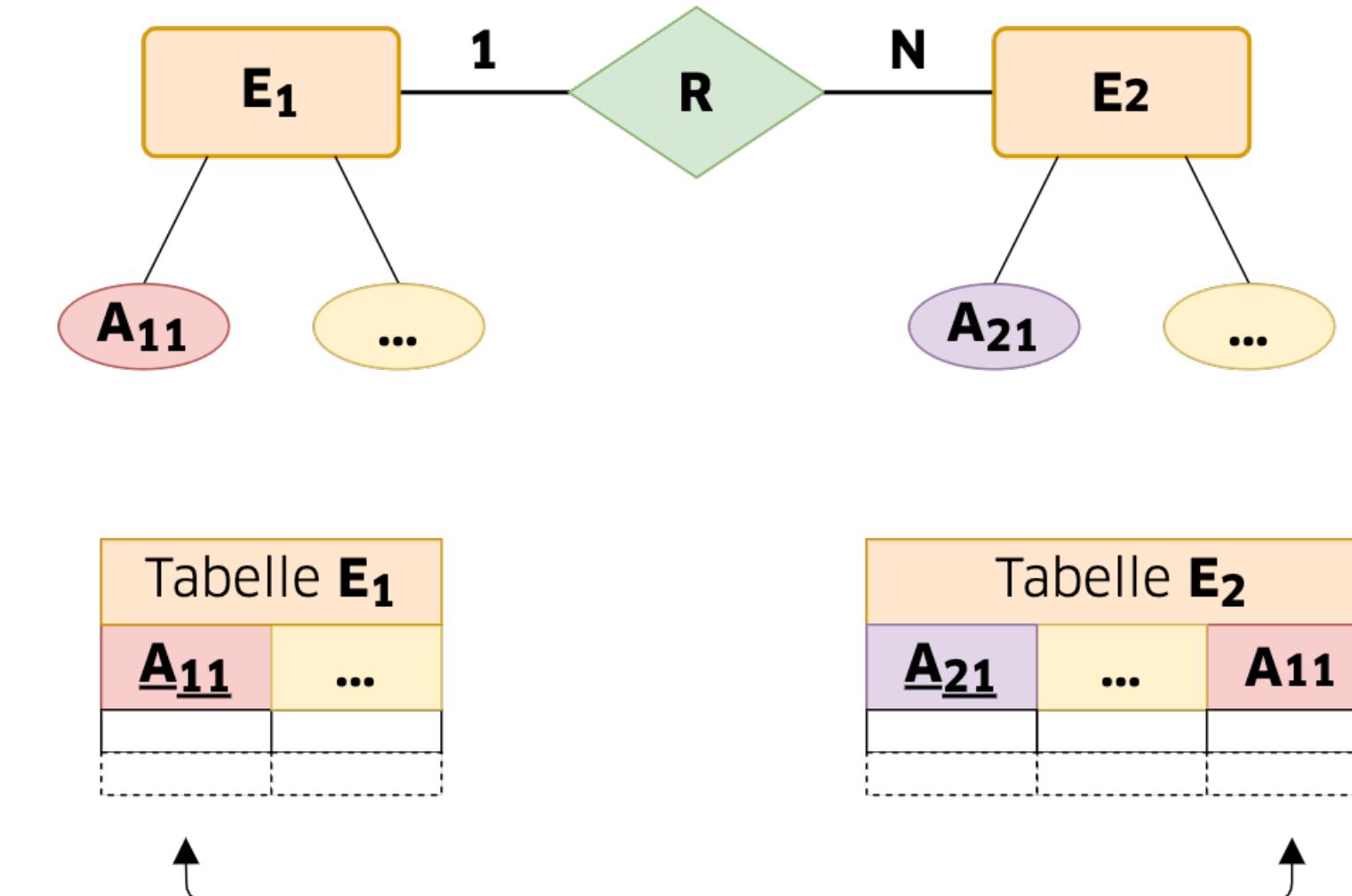
## Anmerkungen zu 1:N-Relationen

- In dem Beispiel ist auf beiden Seiten jeweils nur ein Schlüsselattribut gezeigt.

Gibt es mehrere Schlüsselattribute in  $E_1$ , so müssen sie offenbar *alle* in  $E_2$  auftauchen, denn nur gemeinsam referenzieren sie die assozierte Entität in  $E_1$ .

- Mögliche Attribute der Relation R können ebenfalls in  $E_2$  eingebettet werden. Hier ist aber drauf zu achten, insbesondere bei mehreren 1:N-Relationen, ob  $E_2$  immer noch *genau einen* Aspekt modelliert...
- Manchmal reicht eine kleine Änderung in den Anforderungen und aus einer 1:N-Relation R wird eine N:M-Relation. Bei einer eigenen Tabelle für R ist das kein Problem, ansonsten (wie hier) ändert sich das Schema.

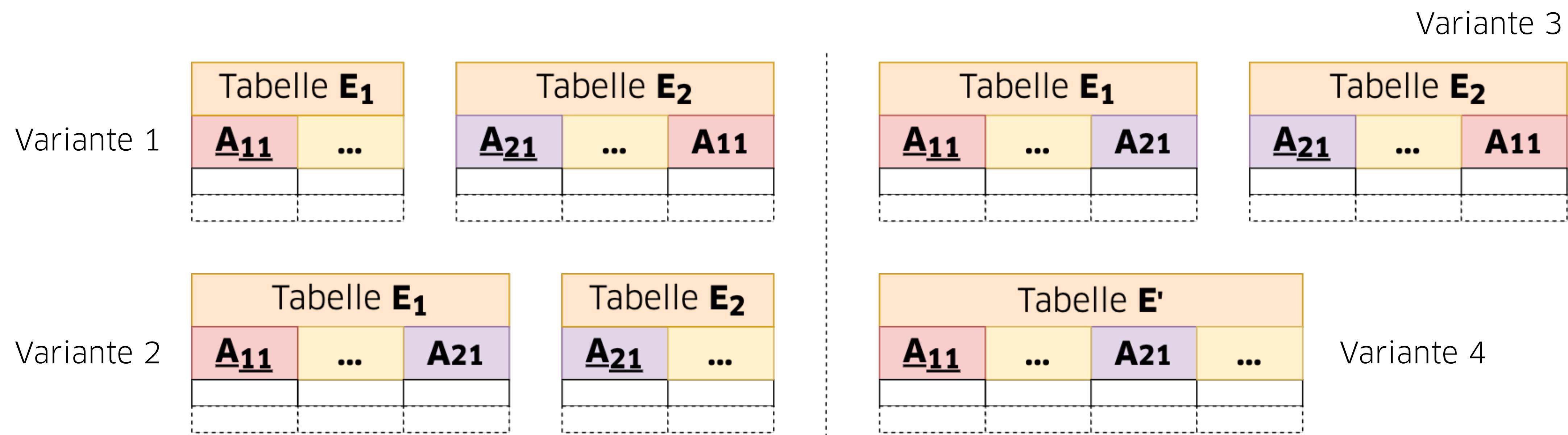
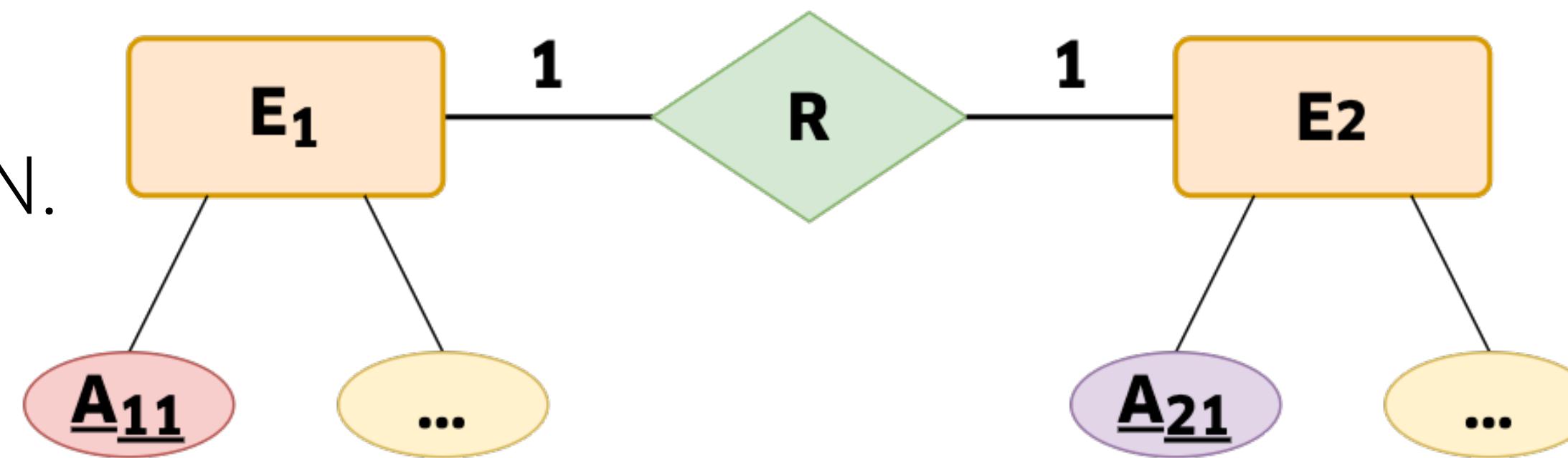
Stabilität der Anforderungen!



## ER-Modell → Relationales Modell

**1:1-Relationen**

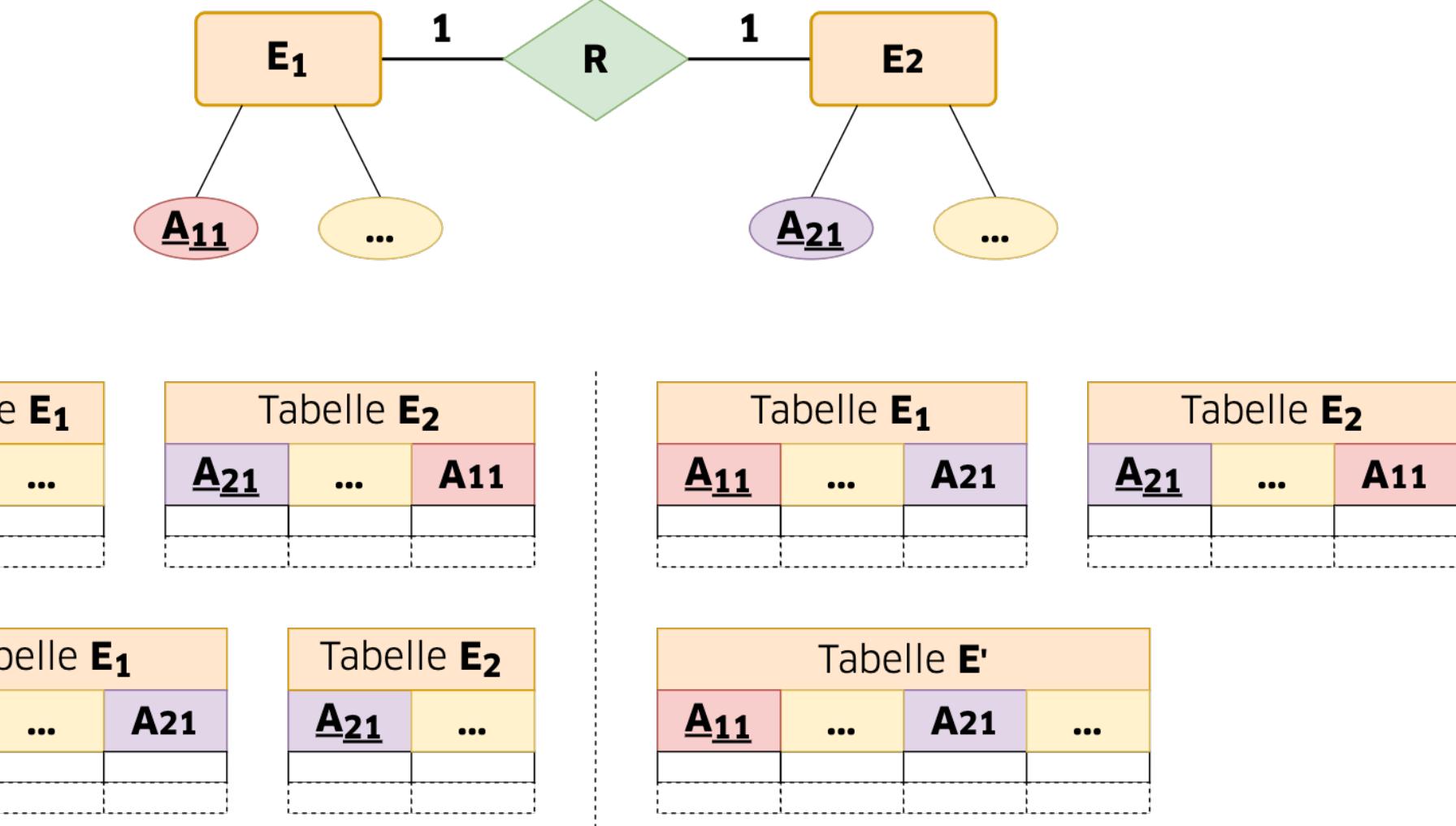
- Variante 1-3: Idee wie 1:N.
- Variante 4: Tabelle E'.
- Variante 5: Tabelle R.



## ER-Modell → Relationales Modell

### Anmerkungen zu 1:1-Relationen

- Welche der Möglichkeiten am geeignetsten ist, ist am Ende von den Anforderungen und der Verwendung der Relation in der Praxis abhängig:
  - Daten aus  $E_1$  und  $E_2$  werden häufig gemeinsam benötigt und man sucht ein  $e_2$  zu einem  $e_1$  - oder umgekehrt.
  - Auch wenn eine gemeinsame Tabelle  $E'$  die Trennung der modellierten Aspekte möglicherweise aufhebt, kann diese Lösung sehr effizient sein.
  - Auch wenn eine eigene Relationetabelle  $R$  die Datenbankoperationen komplexer macht, ist diese Lösung stabil gegenüber Änderungen zu einer 1:N- oder N:M-Rel.
  - ... Kein triviales Problem!



## Zwischenstand

---

### Erinnerung Vorgehen [...]

- Überführung in eine 'gute' Datenbank... dazu benötigen wir:
  - Implementationsentwurf, z.B. mit Tabellen und Relationen  
→ Relationales Modell ✓
  - Mathematisches Modell der Datenbankoperationen, z.B. zur Anfrageoptimierung  
→ Relationale Algebra (folgt)

### Zusammenfassung

- Sie sind jetzt in der Lage, ein Problem zu modellieren und das resultierende ER-Diagramm (ohne Spezialisierungen) in einen Datenbankentwurf zu überführen!



Das ist doch schonmal etwas 👍

## Motivation

---

### Warum ein formaler (mathematischer) Ansatz?

- In den matse\_mhist Produkten befinden sich Pizzen und wir fragen, ob welche weniger als 2.30€ kosten.
- Sind nebenstehende SQL-Befehle äquivalent?  
Alle liefern
 

■ pizza	■ price
Spinatpizza	2.29
- Kann z.B. das DBMS entscheiden, ob es eine Anfrage (select) 'umformen' kann? Z.B. weil es dann effizienter oder mit weniger Zwischendaten arbeiten kann?
- Dafür brauchen wir ein formales Gerüst → Relationale Algebra!

■ id	■ bezeichnung	■ stueckpreis
1	Spinat	1.99
2	Vier Käse Pizza	2.39
3	Spinatpizza	2.29
4	Fischstäbchen	1.99

```

SELECT P.bezeichnung as 'pizza',P.stueckpreis as 'price'
FROM produkt P
WHERE P.bezeichnung like '%pizza%' and P.stueckpreis<2.30;

SELECT Q.*
FROM (
  SELECT P.bezeichnung as 'pizza',P.stueckpreis as 'price'
  FROM produkt P WHERE P.bezeichnung like '%pizza%'
) Q WHERE Q.price<2.30;

SELECT Q.*
FROM (
  SELECT P.bezeichnung as 'pizza',P.stueckpreis as 'price'
  FROM produkt P WHERE P.stueckpreis<2.30
) Q WHERE Q.pizza like '%pizza%';

```

Das Ergebnis eines 'Subselects' (Q) verhält sich wie eine eigene Tabelle.

## Definitionen

---

### Begriffe der relationalen Algebra

- Im mathematischen Kontext der Relationalen Algebra verstehen wir, im Einklang mit Unit 0x03, unter einer Relation R primär eine konkrete *Tupelmenge* und weniger die Beziehung zwischen Entitätstypen:
  - Ein Tupel ist eine geordnete Menge von Attributwerten, wobei D den Wertebereich eines Attributs  $a:T$  mit Datentyp T beschreibt (E.F.Codd nutzt später ebenfalls Attribute statt der Ordnung des Tupels).
  - Eine Relation  $R \subseteq D_1 \times \dots \times D_n$  ist eine Menge von Tupeln  $t = [a_1, \dots, a_n] \in R$ ,  $a_k \in D_k$ .
- Eine Tabelle ist wiederum eine *visuelle Repräsentation* einer Relation und *eine Zeile* in einer Tabelle repräsentiert ein Tuple. Steht die Relation für einen Entitättyp, so sind die Tupel konkrete Ausprägungen bzw. Entitäten.

## Definitionen

---

### Begriffe der relationalen Algebra

- Die nachfolgend diskutierten Operationen der relationalen Algebra, etwa *Selektion*, *Projektion* und *Join*, sind mathematisch Mengenoperationen auf Relationen, die als Ergebnis ebenfalls Relationen liefern. Daher bezeichnet man die Relationenalgebra als abgeschlossen.
- Achtung Praxis...
  - SQL-Kommandos sind so gesehen Umsetzungen der Operationen auf Tabellen.
  - Je nach Kommando/Operation im DBMS können *doppelte Zeilen* entstehen – das Ergebnis ist mathematisch gesehen keine Menge mehr. Praktisch gesehen können Duplikate entfernt bzw. die Operationen direkt geeignet ausgeführt werden.
  - Die mathematische Behandlung des speziellen Attributwertes `NULL`, z.B. in einer dreiwertigen Logik, lassen wir hier aussen vor. Es ist aber wichtig, sich bei jeder Operation klar zu machen, wie diese mit `NULL`-Werten umgeht!

## Definitionen

---

### Begriffe der relationalen Algebra

- Ist man statt an einer Relation  $R$ , also der konkreten Tupelmenge, nur am Aufbau der Relation, d.h. den Attributen und Datentypen, interessiert, helfen diese Funktionen:
  - ♦  $\text{ident}(R) = \{a_i\}_i$ ,
  - ♦  $\text{schema}(R) = \{[a_1:T_1, \dots, a_n:T_n]\}$ .
- Ein Schlüsselkandidat  $K$  mit  $K \subseteq \text{ident}(R)$ , ist eine Menge von Attributen, deren Werte jeweils alle Tupel der Relation eindeutig identifizieren ('identifier').
  - So ist implizit garantiert, dass es keine zwei gleichen Tupel in der Relation gibt.
  - Man kann auch sagen, dass aus der Kenntnis eines Schlüsselkandidaten, genauer den entsprechenden Werten, die anderen Attributwerte des Tupels in der Relation folgen. Diese Sichtweise werden wir im Kontext von 'funktionalen Abhängigkeiten' und 'Normalformen' vertiefen.

## Definitionen

---

### Begriffe der relationalen Algebra

- Grundsätzlich kann es mehrere Schlüsselkandidaten geben, aus denen dann *der Schlüssel* ('primary key') ausgewählt wird. Im Sprechgebrauch wird oftmals der konkrete Wert eines Schlüssels, der 'identifier', gemeint.
  - Kurz: Kennt man den primary key, kennt man die Entität.
- Falls geboten, kann man den Entitätstyp  $E = \langle R, K \rangle$  als Kombination von Relation R und Schlüssel K angeben. So haben wir es in Unit 0x03 vorbereitet und das entspricht auch dem, was man in DBMS bei der Definition einer Tabelle angibt.
- Achtung Praxis...
  - Wenn ein Schlüsselkandidat aus mehreren Attributen besteht, so ist diese Situation im konkreten DBMS hinsichtlich Effizienz und Speicherbedarf zu prüfen. Ggf. kann die Einführung eines neuen künstlichen Schlüssels, bestehend aus nur *einem* Attribut, Sinn machen.

## Operationen der relationalen Algebra

---

### Mengenoperationen

- Um Mengenoperationen auf Relationen durchzuführen, müssen diese kompatibel sein. Das bedeutet, Attributanzahl und Wertebereiche müssen übereinstimmen – auch *Vereinigungsverträglichkeit* oder *Typkompatibilität* genannt.
- Für zwei Relationen S und T könnte man auch etwas unscharf fordern, dass  $\text{schema}(S) = \text{schema}(T)$ , gilt, ohne dass die Attribute formal gleich heißen müssen. Allerdings wäre eine verträgliche Bedeutung (Semantik) schon sinnvoll...
- Es existiert eine minimale Menge von (Grund)Operationen, die mindestens notwendig ist, um alle Ausdrücke der relationalen Algebra bilden zu können. Alle anderen Operationen lassen sich dadurch nachbilden.

# Operationen der relationalen Algebra

---

## Grundoperationen

- Vereinigung
- Differenz
- Kartesisches Produkt
- Selektion
- Projektion
- Umbenennung

Wir betrachten im Folgenden  
*ausgewählte* Operationen mit  
Bezug zu den entsprechenden  
SQL-Kommandos.



## Keine Grundoperationen

- Schnittmenge
- Symmetrische Differenz
- Join
- Equi-Join
- Natural Join
- Semi Join
- Outer Join
- Division

## Operationen der relationalen Algebra

---

### Klassische Mengenoperationen

- Seien  $S$  und  $T$  zwei kompatible Relationen. Dann sind definiert

$$\downarrow S \cup T = \{ r \mid r \in S \vee r \in T \}, \quad \text{SQL: union}$$

$$\downarrow S - T = S \setminus T = \{ r \mid r \in S \wedge r \notin T \} \quad \text{SQL: minus ...}$$

$$\downarrow S \cap T = \{ r \mid r \in S \wedge r \in T \} \quad \text{SQL: intersect ...}$$

Achtung: keine Grundoperation, da  $S \cap T = S \setminus (S \setminus T)$ .

- Leider ist es so, dass nicht alle DBMS alle SQL-Befehle unterstützen. MySQL etwa hat keine Implementierung von `minus` und `intersect` und man muss diese anders umsetzen.



Soviel zum Thema 'Standard'.

## Operationen der relationalen Algebra

### Beispiele Klassische Mengenoperationen

$S \subseteq \text{produkt}$ ,  $T \subseteq \text{produkt}$

- $S \cup T$ :

```
✓ ┌─┐ SELECT P.*  
   ├─┐ FROM produkt P WHERE P.stueckpreis<0.5  
   └─┐ UNION  
   ┌─┐ SELECT P.*  
   ├─┐ FROM produkt P WHERE P.stueckpreis>5.99;
```

id	bezeichnung	stueckpreis
7	Zwiebeln	0.29
6	Möhren	0.39
43	GameStar	6.50

- aber auch  
 $S \cup T$ :

```
✓ ┌─┐ SELECT P.einheit  
   ├─┐ FROM produkt P WHERE P.stueckpreis<0.5  
   └─┐ UNION ALL  
   ┌─┐ SELECT P.einheit  
   ├─┐ FROM produkt P WHERE P.stueckpreis>5.99;
```

einheit
KG
KG
ST
1x
nx

## Operationen der relationalen Algebra

---

### Kartesisches Produkt

- Für zwei Relationen S und T mit  $\text{ident}(S) \cap \text{ident}(T) = \emptyset$  ist das kartesische Produkt (auch Mengen- oder Kreuzprodukt)  $S \times T$  definiert durch
  - ♦ 
$$S \times T = \bigcup_{(s_1, \dots, s_n) \in S} \left[ \bigcup_{(t_1, \dots, t_k) \in T} \{(s_1, \dots, s_n, t_1, \dots, t_k)\} \right]$$
- Achtung: Im üblichen kartesischen Produkt entstehen Paare  $(s, t)$  mit  $s \in S$  und  $t \in T$ . Hier hingegen entstehen  $|S \times T| = |S| \cdot |T|$  Tupel, bestehend aus allen Attributen einer Entität  $s \in S$  verbunden mit den Attributen einer Entität  $t \in T$ .
- Weiter: Im Fall  $\text{ident}(S) \cap \text{ident}(T) \neq \emptyset$  würde obige Definition 'doppelte' Attribute erzeugen, was mathematisch wiederum ein Problem ist, aber in der Praxis durch 'Umbenennung' der Attribute gelöst werden kann.
- Merkregel: Jedes Element von S mit jedem von T.

## Operationen der relationalen Algebra

### Beispiel Kartesisches Produkt

$S \subseteq \text{abteilung}$ ,  $T \subseteq \text{standort}$ ,  $S \times T$ :

Umbenannt  
zu  $S.id$ ,  $T.id$

SELECT S.id, S.name FROM abteilung S;	
id	name
1	Vorstand
2	HR/Buchhaltung
3	Vertrieb
4	Marketing
5	Einkauf
6	F&E
7	AR

SELECT T.id, T.ort FROM standort T;	
id	ort
1	Aachen
2	Jülich
3	Köln
4	Berlin

Jedes Element von S  
mit jedem von T

SELECT S.id, S.name, T.id, T.ort FROM abteilung S, standort T;			
S.id	name	T.id	ort
1	Vorstand	1	Aachen
1	Vorstand	2	Jülich
1	Vorstand	3	Köln
1	Vorstand	4	Berlin
2	HR/Buchhaltung	1	Aachen
2	HR/Buchhaltung	2	Jülich
2	HR/Buchhaltung	3	Köln
2	HR/Buchhaltung	4	Berlin
3	Vertrieb	1	Aachen
3	Vertrieb	2	Jülich
3	Vertrieb	3	Köln
3	Vertrieb	4	Berlin

## Operationen der relationalen Algebra

---

### Selektion

- Für eine Relation  $S$  und eine logische Bedingung  $\Theta$  ist die Selektion definiert durch
  - ♦  $\sigma_\Theta(S) = \{ s \in S \mid s \text{ erfüllt Bedingung } \Theta \}$
- Selektionsbedingungen sind häufig Vergleichsoperationen ( $=, \neq, \leq, <, >, \geq$ ) auf den Attributen und logische Verknüpfungen ( $\wedge, \vee, \neg$ ). Auf eine formale Definition verzichten wir hier.
- Die Selektion wirkt wie ein Filter/Auswahlkriterium auf der Relation  $S$ , wo sie auf jedes Element angewandt wird (siehe Definition).
- Es ist leicht zu sehen, dass der SQL-Befehl `select` mit einer Bedingung  $\Theta$  genau die Operation  $\sigma_\Theta$  abbildet und die Frage nach der günstige Pizza aus der Motivation darauf hinausläuft, ob  $\sigma_{\Theta_1 \wedge \Theta_2}(S) = \sigma_{\Theta_1}(\sigma_{\Theta_2}(S)) = \sigma_{\Theta_2}(\sigma_{\Theta_1}(S))$  gilt.

# Operationen der relationalen Algebra

## Beispiele Selektion

- $\sigma_{\text{bez. like 'getränke'}}(\text{warengruppe})$

```
SELECT * FROM warengruppe
WHERE bezeichnung like '%getränke';
```

Output matse\_mhist.warengruppe

id	bezeichnung
5	Kaltgetränke
8	Heissgetränke

- $\sigma_{\text{gehalt}>50000}(\text{mitarbeiter})$

```
SELECT * FROM mitarbeiter
WHERE jahresgehalt>50000;
```

Output matse\_mhist.mitarbeiter

id	name	jahresgehalt
1	Mia	110000.00
2	Ben	90000.00
3	Emma	70000.00
12	Leon	70000.00
14	Luis	70000.00
16	Lukas	70000.00
20	Leonie	70000.00

# Operationen der relationalen Algebra

---

## Beispiele Selektion

- $\sigma_{(\text{bez. like 'pizza'}) \wedge (\text{preis} < 2.30)}(\text{produkt})$
- $\sigma_{\text{preis} < 2.30}(\sigma_{\text{bez. like 'pizza'}}(\text{produkt}))$
- $\sigma_{\text{bez. like 'pizza'}}(\sigma_{\text{preis} < 2.30}(\text{produkt}))$

```

SELECT P.bezeichnung as 'pizza', P.stueckpreis as 'price'
FROM produkt P
WHERE P.bezeichnung like '%pizza%' and P.stueckpreis<2.30;

SELECT Q.*
FROM (
    SELECT P.bezeichnung as 'pizza', P.stueckpreis as 'price'
    FROM produkt P WHERE P.bezeichnung like '%pizza%'
) Q WHERE Q.price<2.30;

SELECT Q.*
FROM (
    SELECT P.bezeichnung as 'pizza', P.stueckpreis as 'price'
    FROM produkt P WHERE P.stueckpreis<2.30
) Q WHERE Q.pizza like '%pizza%';

```

▪ pizza	▪ price
Spinatpizza	2.29

Wir lassen hier die Benennung der Attribute und die Projektion auf bezeichnung und stueckpreis zunächst aussen vor.

## Operationen der relationalen Algebra

---

### Projektion

- Die Projektion  $\Pi_{a_{i_1} \dots a_{i_k}}$  wählt aus einer Relation S mit  $\text{ident}(S) = \{a_1, \dots, a_n\}$  die Attribute  $a_{i_1}$  bis  $a_{i_k}$  aus:
  - ♦  $\Pi_{a_{i_1} \dots a_{i_k}}(S) = \{ (a_{i_1}, \dots, a_{i_k}) \mid a \in S \}$
- Achtung: Die mathematische Menge eliminiert Duplikate, die Ergebnismenge in SQL aber nicht.
- Der SQL-Befehl select, verbunden mit der Angabe der Attribute einer Relation, realisiert die obige Projektion. Das Kommando ist bekanntlich mächtiger.

# Operationen der relationalen Algebra

## Beispiele Projektion

- $\Pi_{\text{id}, \text{name}, \text{jahresgehalt}}(\text{mitarbeiter})$
- $\Pi_{\text{stueckpreis}, \text{umsatzsteuer}}(\text{produkt})$

```
SELECT id, name, jahresgehalt  
FROM mitarbeiter;
```

Output matse\_mhist.mitarbeiter

id	name	jahresgehalt
1	Mia	110000.00
2	Ben	90000.00
3	Emma	70000.00
4	Paul	5000.00

```
SELECT stueckpreis, umsatzsteuer  
FROM produkt order by stueckpreis;
```

Output matse\_mhist.produkt

stueckpreis	umsatzsteuer
0.29	0.07
0.39	0.07
0.79	0.07
0.90	0.07
0.98	0.07
0.98	0.07
0.99	0.07

## Operationen der relationalen Algebra

---

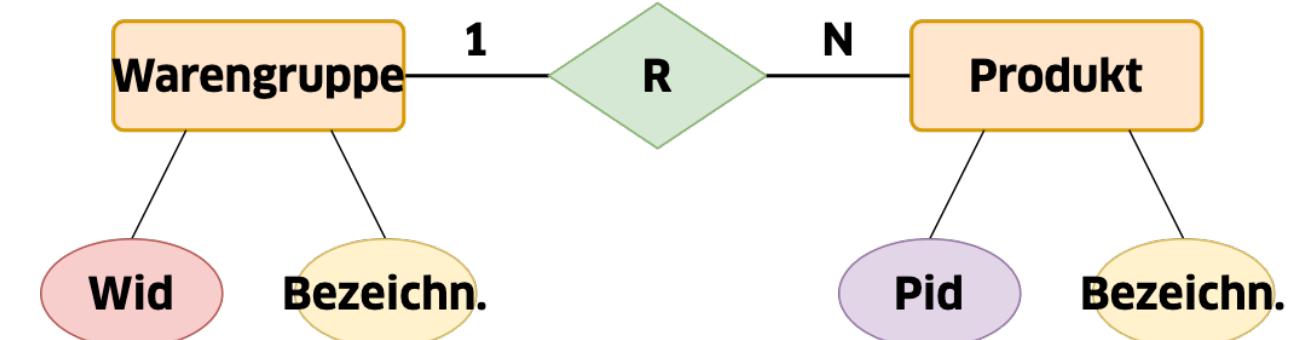
### Theta-Verbund/-Join $\bowtie_\Theta$

- Ganz zu Beginn haben wir in der Diskussion der Artikelsammlung die große Tabelle in mehrere Entitätstypen, Beziehungen und Fremdschlüsselbeziehungen aufgeteilt, um z.B. Redundanzen und Anomalien zu eliminieren. Diese so verteilten Daten wollen wir wieder zusammen führen und das leistet der sog. Theta-Verbund bzw. Theta-Join, oder kurz (und unpräzise) Join.
- Der Theta-Verbund  $S \bowtie_\Theta T$  für zwei Relationen  $S$  und  $T$  und Selektionsbedingung  $\Theta$  ist definiert durch:
  - ♦  $S \bowtie_\Theta T = \sigma_\Theta(S \times T)$
- Achtung: Hier entstehen theoretisch zunächst enorm große Datenmengen durch das kartesische Produkt, die dann durch die Selektion wieder reduziert werden. In der Praxis kann ein DBMS diese sehr häufigen Join-Operationen effizient durchführen.
- Achtung: Es gibt u.a. Inner, Outer, Cross, Self, Natural, Semi-, und Anti-Semi-Joins.

## Operationen der relationalen Algebra

### Beispiel Theta-Verbund

- Zur Motivation der Definition gucken wir zunächst auf die 1:N-Relation zwischen Warengruppe W und Produkt P, die über Fremdschlüssel realisiert ist.
- Das kartesische Produkt kombiniert zunächst jedes Produkt mit jeder Warengruppe ( $\text{produkt} \times \text{warengruppe}$ ).
- Wenn wir genau die Zeilen selektieren, in denen  $P.\text{warengruppe\_id}$  mit  $W.\text{id}$  übereinstimmt ( $\Theta$ ), haben wir zu jedem Produkt die jeweils assozierte Warengruppe → Voilà, die Definition  $\sigma_{\Theta}(P \times W)$ .



produkt

P.id	P.bezeichnung	warengruppe_id
4	Fischstäbchen	1
5	Nudelpfanne	1
6	Möhren	2

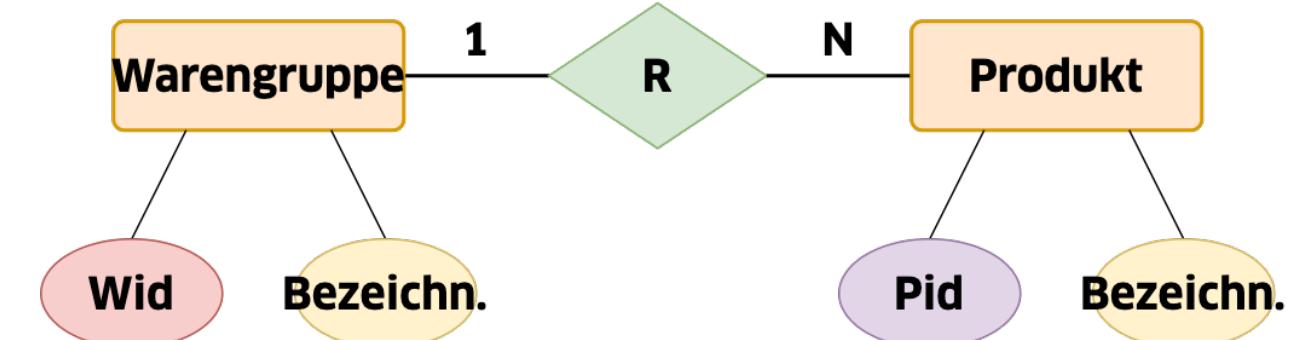
warengruppe

W.id	bezeichnung
1	TK
2	Obst, Gemüse
3	Wurst, Aufschnitt

produkt × warengruppe

P.id	P.bezeichnung	warengruppe_id	W.id	W.bezeichnung
9	5 Nudelpfanne	1	1	TK
0	5 Nudelpfanne	1	2	Obst, Gemüse
1	5 Nudelpfanne	1	3	Wurst, Aufschnitt
2	5 Nudelpfanne	1	4	Milchprodukte
3	5 Nudelpfanne	1	5	Kaltgetränke
4	5 Nudelpfanne	1	6	Grillgut
5	5 Nudelpfanne	1	7	Snacks, Süßwaren
6	5 Nudelpfanne	1	8	Heissgetränke
7	5 Nudelpfanne	1	9	Fertiggerichte
8	5 Nudelpfanne	1	10	Brot, Backwaren
9	5 Nudelpfanne	1	11	Zeitschriften
0	5 Nudelpfanne	1	12	Dienstleistung
1	6 Möhren	2	1	TK
2	6 Möhren	2	2	Obst, Gemüse

## Operationen der relationalen Algebra



### Beispiel Theta-Verbund

- 1:N-Relation
- P=produkt, W=warengruppe,  
 $\Theta=P.\text{warengruppe\_id}=W.\text{id}$
- $P \bowtie_{S.\text{warengruppe\_id}=T.\text{id}} W = \sigma_\Theta(P \times W)$

```

SELECT P.id, P.bezeichnung, P.warengruppe_id, W.id, W.bezeichnung
FROM produkt P, warengruppe W
WHERE P.warengruppe_id=W.id;
    
```

produkt  $\times$  warengr.  
 $\sigma_{S.\text{warengruppe\_id}=T.\text{id}}$

P.id	P.bezeichnung	warengruppe_id	W.id	W.bezeichnung
1	Spinat	1	1	TK
2	Vier Käse Pizza	1	1	TK
3	Spinatpizza	1	1	TK
4	Fischstäbchen	1	1	TK
5	Nudelpfanne	1	1	TK
6	Möhren	2	2	Obst, Gemüse
7	Zwiebeln	2	2	Obst, Gemüse
8	Bananen	2	2	Obst, Gemüse
9	Knoblauch	2	2	Obst, Gemüse
10	Fleischwurst	3	3	Wurst, Aufschnitt
11	Frikadellen	3	3	Wurst, Aufschnitt



Dieser (Inner) Join ist ein zentrales Ergebnis!  
 So bekommen wir Daten aus 1:1-, 1:N- und N:M-Relationen wieder zusammen.

# Operationen der relationalen Algebra

---

## Beispiel Theta-Verbund

- N:M-Relation
- A=abteilung, R=sitzt\_am, S=standort
- $A \bowtie_{A.id=R.abteilung\_id} R \bowtie_{R.standort\_id=S.id} S$

```
SELECT A.name, A.id, R.abteilung_id, R.standort_id, S.id, S.ort
FROM abteilung A, sitzt_am R, standort S
WHERE A.id=R.abteilung_id and R.standort_id=S.id;
```

name	A.id	abteilung_id	standort_id	S.id	ort
Vorstand	1		1	1	1 Aachen
HR/Buchhaltung	2	2		1	1 Aachen
Vertrieb	3	3		1	1 Aachen
F&E	6	6	1	1	Aachen
Vertrieb	3	3	2	2	Jülich
F&E	6	6	2	2	Jülich
Vertrieb	3	3	3	3	Köln
Einkauf	5	5	3	3	Köln
Marketing	4	4	3	3	Köln
Vertrieb	3	3	4	4	Berlin

## Q&A

- Was passiert, wenn es keine assoziierten Entitäten gibt?

# Operationen der relationalen Algebra

---

## Beispiel Theta-Verbund

- N:1-Relation, Tabelle tier hat 3 Entitäten
- T=tier, M=mitarbeiter
- $T \bowtie_{T.\text{mitarbeiter\_id}=M.\text{id}} M$
- Achtung: Der fehlende Eintrag (NULL) bei 'Rosa' führt zu einer kleineren Ergebnismenge, da der Verbund nur assoziierte Entitäten enthält.
- Möchte man *alle* Tiere im Join beachten, so führt das auf den Begriff des *Outer Joins*, hier konkret des *Left Outer Joins*. Dort werden etwaige NULL-Referenzen explizit mit berücksichtigt (praktische Schritte im SQL-Praktikum, Definitionen folgen).

name	mitarbeiter_id
Petra	4
Mini	18
Rosa	<null>

tier

```
SELECT T.name, T.mitarbeiter_id, M.id, M.name
FROM tier T, mitarbeiter M
WHERE T.mitarbeiter_id=M.id;
```

T.name	mitarbeiter_id	id	M.name
Petra	4	4	Paul
Mini	18	18	Max

## Operationen der relationalen Algebra

---

### Anmerkungen Theta-Verbund bzw. Join

- Die bisher gezeigten Joins sind sog. *Inner Joins* und SQL hat eigene (äquivalente und optimierte) Befehle dafür (`join`).
- Dies, und auch einen Teil der anderen *Joins*, besprechen wir schon einmal im SQL-Praktikum, damit man in der SQL-Praxis weiter kommt. Die jeweiligen formalen Definitionen folgen dann in Teil 2 der relationalen Algebra.

## Zwischenstand Operationen der relationalen Algebra

---

### Grundoperationen

- Vereinigung ✓
- Differenz (✓)
- Kartesisches Produkt ✓
- Selektion ✓
- Projektion ✓
- Umbenennung

### Keine Grundoperationen

- Schnittmenge (✓)
- Symmetrische Differenz
- Inner Join ✓
- Equi-Join
- Natural Join
- Semi Join
- Outer Join
- Division

# UNIT 0X06

# RELATIONALE ALGEBRA II

## Operationen der relationalen Algebra

---

### Wiederholung

- Die bisher gezeigten Joins sind sog. *Inner Joins* und SQL hat eigene (äquivalente und optimierte) Befehle dafür (`join`).
- Dies, und auch einen Teil der anderen *Joins*, besprechen wir schon vorab im SQL-Praktikum, damit man in der SQL-Praxis weiter kommt. Die jeweiligen formalen Definitionen folgen dann hier im zweiten Teil der relationalen Algebra.

# Operationen der relationalen Algebra

## SQL-Einschub with

- Um die folgenden Operationen zu verdeutlichen, konstruieren wir bei Bedarf spezielle Relationen mit jeweils 'passenden' Attributen und Werten. Hierzu sehr hilfreich sind sog. 'Common Table Expressions (CTE)' mit `with`.
- Bei CTEs handelt es sich, kurz gesagt, um temporäre Ergebnismengen, die unter einem eigenen Namen in darauf folgenden SQL-Befehlen zur Verfügung stehen.

## Beispiel

```
SELECT * FROM produkt;
```

id	bezeichnung	einheit
1	Spinat	PK
2	Vier Käse Pizza	ST
3	Spinatpizza	ST

Bekannte Struktur von `produkt`

```
WITH Prods as (
    SELECT id as 'prod_id', bezeichnung as 'bez' FROM produkt
)
SELECT * FROM Prods WHERE bez like '%pizza%';
```

prod_id	bez
2	Vier Käse Pizza
3	Spinatpizza

Neue Relation `Prods` nur mit Spalten `prod_id` und `bez`, nutzbar in `select`

## Operationen der relationalen Algebra

---

### Theta-Verbund/-Join vs Equi-join

- Der Join (Verbund) bezeichnet allgemein die beiden Operationen kartesisches Produkt mit anschliessender Selektion und Selektionsbedingung  $\Theta$ , daher auch Theta-Verbund:
  - ♦  $S \bowtie_{\Theta} T = \sigma_{\Theta}(S \times T)$
- Ein Spezialfall ist der *Equi Join* mit der Bedingung, dass der Inhalt bestimmter Attribute, z.B.  $a_1$  und  $a_2$ , identisch sein muss, d.h. der speziellen Form  $a_1 = a_2$  genügt:
  - ♦  $S \bowtie_{a_1=a_2} T = \sigma_{a_1=a_2}(S \times T)$

## Operationen der relationalen Algebra

---

### Natürlicher Verbund

- Ausgehend von der Idee, dass Primär- und Fremdschlüssel gleich heißen (mögen), nutzt der natürliche Verbund dies aus und verbindet Entitäten, die in gleich benannten Attributen gleiche Werte besitzen - ein Equi-Join auf gleichen Attributen.
- Im Unterschied zum Theta-Verbund enthält der Natural Join die gleichen Attribute nur einmal, eliminiert so also ungewünschte Redundanz.
- Für zwei Relationen  $S, T$  mit  $\text{schema}(S) = \{[a_1, \dots, a_n, b_1, \dots, b_k]\}, \text{schema}(T) = \{[b_1, \dots, b_k, c_1, \dots, c_m]\}$  (für die Übersicht ohne Typangabe) ist der natürliche Verbund  $S \bowtie T$  definiert durch
  - ♦  $S \bowtie T = \Pi_{a_1..a_n, S.b_1..S.b_k, c_1..c_m} \sigma_{S.b_1=T.b_1 \wedge \dots \wedge S.b_n=T.b_n} (S \times T)$
- Das entspricht dem Schema  $\{[a_1, \dots, a_n, b_1, \dots, b_k, c_1, \dots, c_m]\}$  (wieder ohne Typangabe, ggf. ergänzen) und man sieht, dass die 'doppelten' Attribute  $S.b_1$  bzw.  $T.b_1$  etc. durch die Projektion  $\Pi$  weggefallen sind.

# Operationen der relationalen Algebra

## Beispiel Natürlicher Verbund

- Vorbereitung der Relationen produkt und warenguppe ZU wg bzw. pg mit umbenannten Attributten.

Man sieht, dass das 'neue' gemeinsame Attribut *nur* wid, also der Fremdschlüssel der Warenguppe ist und insbesondere auch die Bezeichnung jetzt wbez bzw. pbez, also unterschiedlich, ist! So verwenden wir wg und pg mit with im Beispiel für Join bzw. Natural Join.

- Beim Natural Join fällt die 'doppelte' Spalte weg.

```
SELECT id as 'wid', bezeichnung as 'wbez' FROM warengruppe;
SELECT id as 'pid', bezeichnung as 'pbez', warengruppe_id as 'wid' FROM produkt;
```

	wid	wbez	pid	pbez	wid
wg	1	TK	1	Spinat	1
	2	Obst, Gemüse	2	Vier Käse Pizza	1
	3	Wurst, Aufschnitt	3	Spinatpizza	1
	4	Milchprodukte	4	Fischstäbchen	1
	5	Kaltgetränke	5	Nudelpfanne	1

```
WITH wg AS (SELECT id as 'wid', bezeichnung as 'wbez' FROM warengruppe)
      pd AS (SELECT id as 'pid', bezeichnung as 'pbez', warengruppe_id as 'wid' FROM produkt)
SELECT * FROM pd JOIN wg ON pd.wid=wg.wid;
```

pid	pbez	pd.wid	wg.wid	wbez
1	Spinat	1	1	TK
2	Vier Käse Pizza	1	1	TK
3	Spinatpizza	1	1	TK

```
WITH wg AS (SELECT id as 'wid', bezeichnung as 'wbez' FROM warengruppe)
      pd AS (SELECT id as 'pid', bezeichnung as 'pbez', warengruppe_id as 'wid' FROM produkt)
SELECT * FROM pd NATURAL JOIN wg;
```

wid	pid	pbez	wbez
1	1	Spinat	TK
1	2	Vier Käse Pizza	TK
1	3	Spinatpizza	TK

## Operationen der relationalen Algebra

---

### Anmerkungen Natürlicher Verbund

- Von der Verwendung von *Natural Joins* wird abgeraten!
- Beim *Natural Join* werden immer *alle* gleichnamigen Attribute verwendet, d.h. Hinzufügen neuer Attribute oder Umbenennen vorhandener Attribute führt schnell zu einer Änderung der Abfrage!
- Achtung: Erweitern einer Tabelle ist nichts ungewöhnliches und passiert ggf. sogar automatisiert, wenn z.B. die zu persistierende Objektstruktur erweitert wird und die Tabellen die Daten widerspiegeln.

### Beispiel

- Hier wurden Bezeichnungen gleich gewählt. Damit muss *wid* und *bez* für den *Natural Join* übereinstimmen... Oops.

```
WITH wg AS (SELECT id AS 'wid', bezeichnung AS 'bez' FROM warengruppe,
                    pd AS (SELECT id AS 'pid', bezeichnung AS 'bez', warengruppe_id AS 'wid' FROM produkt)
SELECT * FROM pd NATURAL JOIN wg;
```

wid	pid
1	1

## Operationen der relationalen Algebra

### Semi-Join bzw. Halbverbund

- Manchmal interessiert nur die Existenz, nicht aber die Attributwerte der assoziierten Entität beim (Natural) Join → Halbverbund bzw. Semi-Join.
- Der Halbverbund  $R \ltimes S$  ist für zwei Relationen S und T definiert durch:
  - ♦  $S \ltimes T = \Pi_{\text{ident}(S)} (S \bowtie T)$

### Beispiel

- Projiziert man das Beispiel des *Natural Joins* zuvor auf die Attribute von pd, so ergibt das den Halbverbund.

```
WITH wg AS (SELECT id AS 'wid', bezeichnung AS 'wbez' FROM warengruppe,
                    pd AS (SELECT id AS 'pid', bezeichnung AS 'pbez', warengruppe_id AS 'wid' FROM produkt)
SELECT pd.pid, pd.pbez, pd.wid FROM pd NATURAL JOIN wg;
```

pid	pbez	wid
1	Spinat	1
2	Vier Käse Pizza	1
3	Spinatpizza	1
4	Fischstäbchen	1

## Operationen der relationalen Algebra

---

### Anti-Semi-Join

- Beim Anti-Semi-Join  $S \triangleright T$  zweier Relationen S und T werden die Tupel aus S selektiert, die am natürlichen Verbund *nicht* teilnehmen:
  - ♦  $S \triangleright T = S - S \bowtie T = S - \Pi_{\text{ident}(S)}(S \bowtie T)$

### Beispiel

- Folgt noch, da wir den Minus-Operator benötigen, der wiederum in MySQL nicht definiert ist aber über einen Outer Join abbildbar ist.

## Operationen der relationalen Algebra

---

### Outer Join

- 'Zusammengehörige' Daten *und Daten*, zu denen *kein Pendant beim Join existiert*, abfragen. Das sind: *Left Outer Join*, *Right Outer Join* und *Full Outer Join*.
- Bei einem *Left Outer Join* zweier Relationen S und T, in Zeichen
  - ♦  $S \bowtie T$  werden *alle* Entitäten der Entitätenmenge *links* der Relation, also S, berücksichtigt, auch wenn es keine zugehörigen Entitäten in Entitätenmenge T gibt. Diese Attribute sind dann `NULL`.
- Bei einem *Right Outer Join*, in Zeichen
  - ♦  $S \bowtie T$  gilt das analog, nur mit vertauschten Rollen. D.h. es werden alle Entitäten der *rechten* Relation T berücksichtigt, auch wenn es keine zugehörigen Entitäten in S gibt.

# Operationen der relationalen Algebra

---

## Outer Join

- Der *Full Outer Join* ist die Vereinigung von *Left Outer Join* und *Right Outer Join*. Das bedeutet, es sind alle Entitäten beider Seiten dabei, nur ggf. mit NULL-Einträgen in den Attributen der anderen Seite, wenn es keine zugehörige Entität gibt. In Zeichen
  - $S \bowtie T$
- Bei den *Outer Joins* sind nicht nur *Natural Joins*, sondern allgemein Theta-Verbünde gemeint. Im Gegensatz zu *Outer Joins* spricht man auch explizit von *Inner Joins*.

## Beispiel Outer Join

- Tiere mit und ohne 'Besitzer' aus der Menge der Mitarbeiter.
- Zur Erinnerung: 'Rosa' ist beim *Inner Join* nicht dabei:

id	name	mitarbeiter_id
1	Petra	4
2	Mini	18
3	Rosa	<null>

```

SELECT T.name, M.name
FROM tier T INNER JOIN mitarbeiter M
ON M.id = T.mitarbeiter_id;

```

T.name	M.name
Petra	Paul
Mini	Max

Beispiele matse\_mhist

## Operationen der relationalen Algebra

### Beispiel Outer Join Fortsetzung

- Beim *Left Outer Join* ist 'Rosa' dabei, da *alle* Entitäten des Typs Tier (links des Joins) berücksichtigt werden, auch wenn sie nicht in Relation stehen.  
Die zugehörigen Attribute des (nicht vorhandenen) Partners sind `NULL`.
- Betrachten wir das gleiche SQL-Kommando als *Right Outer Join*, so kommen *alle* Mitarbeiter (rechte Seite des Joins) vor und die zugehörigen Attribute sind `NULL`, wenn der Mitarbeiter nicht in Relation zu einem Tier steht.  
Hier ist 'Rosa' nicht dabei!

The screenshot shows two database query results side-by-side.

**Top Query (Left Outer Join):**

```
SELECT T.name, M.name
FROM tier T LEFT OUTER JOIN mitarbeiter M
ON M.id = T.mitarbeiter_id;
```

**Output:**

T.name	M.name
Petra	Paul
Mini	Max
Rosa	<null>

**Bottom Query (Right Outer Join):**

```
SELECT T.name, M.name
FROM tier T RIGHT OUTER JOIN mitarbeiter M
ON M.id = T.mitarbeiter_id;
```

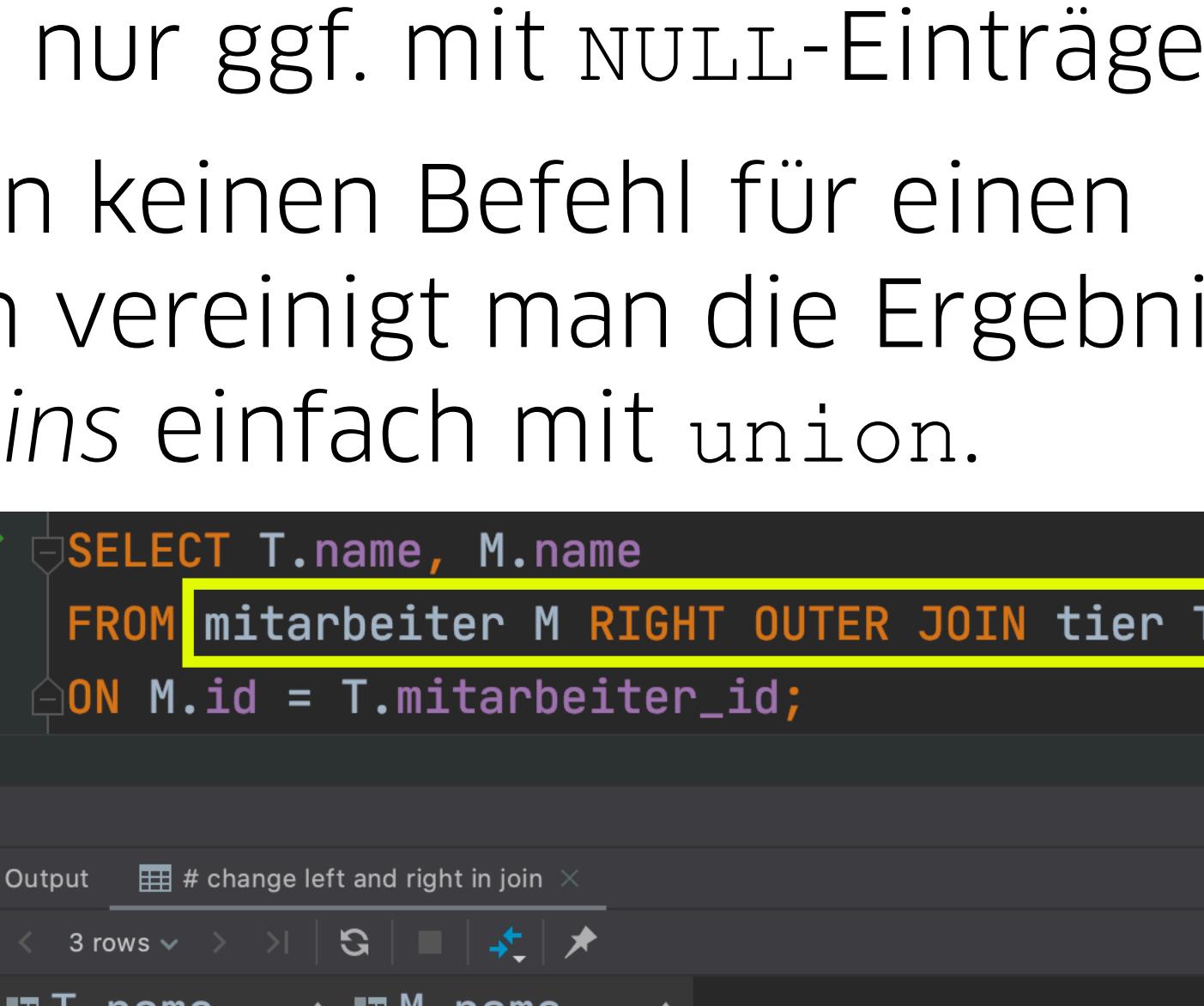
**Output:**

T.name	M.name
Petra	Paul
Mini	Max
<null>	Mia
<null>	Ben
<null>	Emma
<null>	Hannah
<null>	Luka

Beispiele matse\_mhist

# Operationen der relationalen Algebra

# Beispiel Outer Join Fortsetzung

- Der Full Outer Join ist die Vereinigung von *Left* und *Right Outer Join*. Das bedeutet, es sind alle Entitäten beider Seiten dabei, nur ggf. mit NULL-Einträgen.
  - Manche DBMS haben keinen Befehl für einen *Full Outer Join*, dann vereinigt man die Ergebnisse der beiden *Outer Joins* einfach mit *union*.
  - Vertauschen von Entitätstypen und *Left* und *Right Outer Joins*, ergibt wieder den ersten Outer Join.

The screenshot shows a code editor with a SQL query. The query is:

```
SELECT T.name, M.name
FROM mitarbeiter M RIGHT OUTER JOIN tier T
ON M.id = T.mitarbeiter_id;
```

The 'RIGHT OUTER JOIN' clause is highlighted with a yellow box. Below the code editor is a results table titled 'Output' with the following data:

T.name	M.name
Petra	Paul
Mini	Max

```
SELECT T.name, M.name  
FROM mitarbeiter M RIGHT OUTER JOIN tier T  
ON M.id = T.mitarbeiter_id;
```

```
SELECT T.name, M.name
FROM tier T LEFT OUTER JOIN mitarbeiter M
ON M.id = T.mitarbeiter_id
UNION
SELECT T.name, M.name
FROM tier T RIGHT OUTER JOIN mitarbeiter M
ON M.id = T.mitarbeiter_id;
```

Output    Result 82 ×

T.name	M.name
Petra	Paul
Mini	Max
Rosa	<null>
<null>	Mia
<null>	Ben
<null>	Emma
<null>	Hannah

# Beispiele matse mhist

# Operationen der relationalen Algebra

---

## Anmerkungen Inner / Outer Join

- Nur die Kunden, zu denen eine Bestellung existiert, d.h. die in Relation stehen – somit *Inner Join*:
- Kunden, zu denen eine Bestellung existiert und die, die noch keine Bestellung aufgegeben haben, d.h. faktisch *alle* Kunden, mit und ohne Partner – somit *Outer Join*:
  - Der Entitätstyp *kunde* steht *rechts* am *Join*, daher hier *Right Outer Join*.
- Da beim *Inner Join* nur Entitäten teilnehmen, die einen Partner haben, gibt es keinen *Left* oder *Right Inner Join*.

```
SELECT B.id, B.kunde_id, K.id, K.name
FROM bestellung B INNER JOIN kunde K
ON B.kunde_id=K.id;
```

B.id	kunde_id	K.id	name
1	30	30	Sparkasse
2	30	30	Sparkasse
3	32	32	E.ON

```
SELECT B.id, B.kunde_id, K.id, K.name
FROM bestellung B RIGHT OUTER JOIN kunde K
ON B.kunde_id=K.id;
```

B.id	kunde_id	K.id	name
<null>	<null>	29	Deutsche Bank
1	30	30	Sparkasse
2	30	30	Sparkasse
<null>	<null>	31	Börse
3	32	32	E.ON
<null>	<null>	33	Infinion

Beispiele matse\_mhist

## Operationen der relationalen Algebra

---

### Differenz/Minus, Schnittmenge, Symmetrische Differenz

- Diese beiden Operationen sind beispielsweise in MySQL bzw. MariaDB nicht vorhanden, können über Joins aber leicht abgebildet werden.
- Für zwei Relationen S und T mit  $\text{schema}(S)=\text{schema}(T)$  und Schlüsselattribut  $K$  gilt:
  - ◆  $S - T$ : `SELECT S.* FROM S LEFT OUTER JOIN T USING(K) WHERE isnull(T.K)`
  - ◆  $S \cap T$ : `SELECT S.* FROM S JOIN T USING(K);`
- Die Symmetrische Differenz ist somit auch definiert:
  - ◆  $S \Delta T = (S - T) \cup (T - S)$

## Operationen der relationalen Algebra

---

### Umbenennung von Relationen oder Attributen

- Motivation: Bei einem Join oder kartesischen Produkt kommt es zuweil vor, dass in der Ergebnisrelation eigentlich Attribute gleich heissen würden – was mathematisch und technisch ein Problem ist. Analog kann es notwendig sein, ganzen Relationen einen eigenen Namen zu geben, um etwa bei einem Self-Join diese zu unterscheiden.
- Idee: Umbenennen einer Relation  $S$  zu  $S'$  mittels:
  - ♦  $R[S \rightarrow S']$  oder  $\rho_{S'}(S)$
- und Umbenennen eines Attributes  $a$  zu  $a'$  einer Relation  $T$ :
  - ♦  $\rho_{a \rightarrow a'}(T)$

### Beispiel

- $\rho_{id \rightarrow no, bezeichnung \rightarrow bez}(\rho_{pr}(produkt))$

no	bez
1	Spinat
2	Vier Käse Pizza
3	Spinatpizza
4	Fischstäbchen

## Operationen der relationalen Algebra

---

### Division

- *Hintergrund:* Fragen, in denen es um 'für alle' (Allquantor  $\forall$ ) geht.
- Für zwei Relationen  $S$  und  $T$  mit  $\text{schema}(S) = \{[a_1, \dots, a_n, b_1, \dots, b_k]\}$ ,  $\text{schema}(T) = \{[b_1, \dots, b_k]\}$  (hier ohne Typangabe) ist die Division  $S \div T$  definiert durch
  - ♦  $S \div T = \{ (a_1, \dots, a_n) \mid \forall (b_1, \dots, b_k) \in T : (a_1, \dots, a_n, b_1, \dots, b_k) \in S \}$ , oder
  - ♦  $S \div T = \Pi_{(S-T)}(S) - \Pi_{(S-T)}((\Pi_{(S-T)}(S) \times T) - S)$
- Division  $\div$  ist 'Umkehroperation' zum kartesischen Produkt, denn es gilt (symbolisch):
  - ♦  $(S \times T) \div T = S$
- Achtung: Ergebnismenge ist nur der 'a'-Anteil von  $S$ , also  $\Pi_a(S)$

## Operationen der relationalen Algebra

---

### Beispiel Foto-Weihnachten

- Bestimme alle Fotos (Datum, Zeit), auf denen 'Vadder' und 'Mutti' zu sehen sind. Oder anders formuliert:
- Bestimme alle Fotos einer Menge R, für die gilt, dass *alle* Personen einer Teilmenge S ('Vadder', 'Mutti') zu sehen sind, d.h. wir suchen  $R \div S$ , mit R und S gegeben durch

$R = \{[\text{Datum}, \text{Zeit}, \text{Name}]\}$

Datum	Zeit	Name
24.12.18	09:00	Mutti
24.12.18	09:00	Omma
24.12.18	09:00	Vadder
24.12.18	10:00	Vadder
25.12.18	12:00	Mutti
25.12.18	12:00	Filius
25.12.18	12:00	Vadder
26.12.18	15:00	Omma

$S = \{[\text{Name}]\}$

Name
Mutti
Vadder

## Operationen der relationalen Algebra

---

### Fortsetzung Beispiel Foto-Weihnachten

- Die Division (siehe Definition) kann man schrittweise durchführen

$$R \div S = \Pi_{(R-S)}(R) - \Pi_{(R-S)}((\Pi_{(R-S)}(R) \times S) - R)$$

$\pi_{R-S}(R)$

Datum	Zeit
24.12.18	09:00
24.12.18	10:00
25.12.18	12:00
26.12.18	15:00

$\pi_{R-S}(R) \times S$

Datum	Zeit	Name
24.12.18	09:00	Mutti
24.12.18	09:00	Vadder
24.12.18	10:00	Mutti
24.12.18	10:00	Vadder
25.12.18	12:00	Mutti
25.12.18	12:00	Vadder
26.12.18	15:00	Mutti
26.12.18	15:00	Vadder

# Operationen der relationalen Algebra

---

## Fortsetzung Beispiel Foto-Weihnachten

- $R \div S = \Pi_{(R-S)}(R) - \Pi_{(R-S)}((\Pi_{(R-S)}(R) \times S) - R)$

 $\pi_{R-S}(R) \times S - R$ 

Datum	Zeit	Name
24.12.18	10:00	Mutti
26.12.18	15:00	Mutti
26.12.18	15:00	Vadder

 $\pi_{R-S}(\pi_{R-S}(R) \times S - R)$ 

Datum	Zeit
24.12.18	10:00
26.12.18	15:00

 $\pi_{R-S}(R) - \pi_{R-S}(\pi_{R-S}(R) \times S - R)$ 

Datum	Zeit
24.12.18	09:00
25.12.18	12:00

 $= R \div S$

## Operationen der relationalen Algebra

### Fortsetzung Beispiel Foto-Weihnachten

$R$	$S$	$R \div S$	
Datum	Name	Datum	Zeit
24.12.18	Mutti		
24.12.18	Omma		
24.12.18	Vadder		
24.12.18	Vadder		
25.12.18	Mutti	24.12.18	09:00
25.12.18	Filius	25.12.18	12:00
25.12.18	Vadder		
26.12.18	Omma		

$R \div S$  sind alle die Datensätze, die mit allen Daten aus S kombiniert vorkommen – aber nur die Attribute R-S!

- Alternativ (*überschaubare Datenmenge*): Es sind alle Datensätze in R gesucht, die mit allen S vorkommen...

## Operationen der relationalen Algebra

---

### Aggregation und Gruppierung

- Hintergrund: Gruppierung von Tupeln, d.h. Tupel mit identischen Werten in Attributen  $a_1, \dots, a_n$  (Gruppierungsattribute) werden zu einer Gruppe zusammengefasst und ggf. Aggregatfunktionen  $f_1, \dots, f_k$  angewendet. Notation:
  - ♦  $\gamma_{a_1, \dots, a_n; f_1, \dots, f_k}(S)$
- Jede Aggregatfunktion  $f_1, \dots, f_k$  ergibt eine Spalte in der Ergebnistabelle.
- Aggregatfunktion sind z.B. count, avg, sum, min, max, siehe SQL-Praktikum.

### Beispiel

- Gruppiere Studierende nach Semesterzahl und zähle sie pro Gruppe  
 $\gamma_{\text{Semester}; \text{count}(\ast)}(\text{studenten})$

## Übersicht Operationen der relationalen Algebra

---

### Grundoperationen

- Vereinigung ✓
- Differenz ✓
- Kartesisches Produkt ✓
- Selektion ✓
- Projektion ✓
- Umbenennung ✓

### Erweiterte Grundfunktionen

- Gruppierung / Aggregation ✓

### Keine Grundoperationen

- Schnittmenge ✓
- Symmetrische Differenz ✓
- Theta-Verbund, Inner Join ✓
- Equi-Join ✓
- Natural Join ✓
- Semi Join ✓
- Anti-Semi-Join ✓
- Outer Join ✓
- Division ✓

# Übersicht Operationen der relationalen Algebra

## SQL-Beispiele

- Synthetische Mengen R und S, wobei C ein gemeinsames Attribut ist und S.FB einen Fremdschlüssel C aus R darstellen soll.

Rid	A	B	C
1	a1	b1	c1
2	a2	b2	c2
3	a3	b3	c3
4	a4	b4	<null>

Sid	C	D	E	FB
11	c1	d1	e1	b1
13	c3	d3	e3	b3
14	c4	d4	e4	b4

- Die Beispiele nutzen zum Teil with zur Zusammenstellung geeigneter Teilmengen.

```

SELECT *
FROM R RIGHT OUTER JOIN S ON R.C = S.C;
# full outer join
SELECT *
FROM R LEFT OUTER JOIN S ON R.C = S.C
UNION
SELECT *
FROM R RIGHT OUTER JOIN S ON R.C = S.C;

# with, minus
WITH
    A1 AS (select * from R where Rid<4),
    A2 AS (select * from R where Rid>1)
SELECT A1.* FROM A1 LEFT OUTER JOIN A2 USING (Rid)
WHERE isnull(A2.Rid);

A1
+-----+
| Rid | A   | B   | C   |
+-----+
| 1   | a1 | b1 | c1 |
+-----+

```

Auszug SQL,  
Schema matse\_algebra

## Alternativen zur relationalen Algebra

---

### Tupelkalkül

- Idee: Ergebnis einer Anfrage wird als Menge von Tupeln beschrieben, die einer prädikatenlogischen Formel  $\psi$  entsprechen (wie bei mathematischen Mengen)

♦  $\{ s \in S \mid \psi(s) \text{ wahr} \}$ ,

die wiederum aus sog. Atomen  $a$ ,  $a$  Attribut in  $S$ , zusammengesetzt ist.

Beispiel:  $\{ s \in \text{studenten} \mid s.\text{Semester} > 5 \}$

### Domänenkalkül

- Grundidee wie zuvor, aber Variablen stehen hier für Tupelkomponente, d.h.

♦  $\{ [a_1, \dots, a_n] \in S \mid \psi(a_1, \dots, a_n) \text{ wahr} \}$ ,

Beispiel:  $\{ [\text{matrnr}, \text{name}, \text{sem}] \in \text{studenten} \mid \text{sem} > 5 \}$

Drei Sprachen sind gleich  
mächtig, aber nicht Turing-  
vollständig.



# UNIT 0x07

## NORMALFORMEN I

## FUNKTIONALE ABHÄNGIGKEITEN

## Motivation/Erinnerung

---

### Aufgabe

- Entwicklung einer 'guten' Datenbank zu einem realen Problem (z.B. Mini-Welt), d.h. u.a. vollständig, korrekt, minimal, aber auch redundanzfrei und effizient...

### Vorgehen

- Konsolidierung und Konzeptueller Entwurf in Form eines ER-Modells ✓
- Überführung in eine 'gute' Datenbank... dazu benötigen wir:
  - Implementationsentwurf, z.B. mit Tabellen und Relationen
    - Relationales Modell ✓
  - Mathematisches Modell der Datenbankoperationen, z.B. zur Anfrageoptimierung
    - Relationale Algebra ✓
  - Gütemaß für den Implementationsentwurf
    - Normalformen

## Qualität eines Datenbankentwurfs

### Anforderungen an unsere Artikel-Sammlung, u.a.

- Widerspruchsfreie Speicherung von Daten jeder Art
- Erinnerung  
erster  
Entwurf:

Nr (EAN,ISBN)	Zeitschrift	Verlags- gründung	Themen
...aa11	Heise - c't - 11/18	1949	S.15 C++, S.24 C#
...bb22	Heise - c't - 12/18	1949	S.12 Python, S.29 C#
...cc33	Heise - ix - 08/18	1949	S.9 RAID, S.23 gpio
...dd44	Computec - buffed - 08/18	1989	S.11 WoW
...ee55	Webedia - GameStar - 08/18	2007	S.13 LoL, S.20 WoW

### Q&A

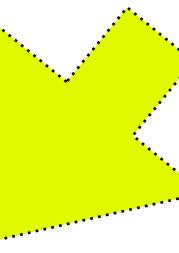
- Was war noch  
gleich die Kritik?

## Qualität eines Datenbankentwurfs

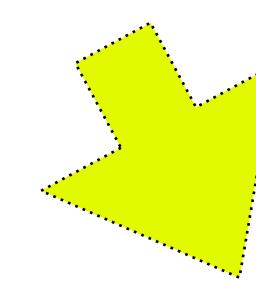
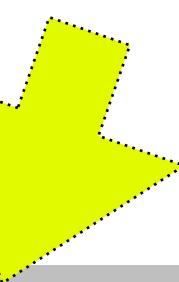
### **"Besserer" Entwurf, Kernideen**

- Redundanzen verringern
- Sachverhalte separieren

Nr (N)	Verlag (V)	Magazin (M)	Ausgabe (A)
aa11	Heise	c't	11/18
bb22	Heise	c't	12/18
cc33	Heise	iX	08/18
dd44	Computec	buffed	08/18
ee55	Webedia	GameStar	08/18



Nr (N)	Seite (S)	Thema (T)
aa11	15	C++
aa11	24	C#
bb22	12	Python
bb22	29	C#
cc33	9	RAID
cc33	23	gpio
dd44	11	WoW



Verlag (V)	Gründung (G)
Heise	1949
Computec	1989
Webedia	2007

### Q&A

- Wie ist die Qualität eines Entwurfs zu messen?
- Wie erhält man systematisch eine Verbesserung?
- Kann man direkt einen guten Entwurf erstellen?

## Qualität eines Datenbankentwurfs

---

### Wie ist die Qualität eines Entwurfs zu messen?

- Gütekriterien, genannt *Normalformen*

### Wie erhält man systematisch eine Verbesserung?

- Definitionen: Funktionale Abhängigkeiten, Attributhüllen, Kanonische Überdeckung
- Mathematische Werkzeuge: Armstrong-Axiome
- Algorithmen: Synthese- und Dekompositionsalgorithmus

### Kann man 'direkt' einen guten Entwurf erstellen?

- Spoiler: Ja! Aber dafür müssen die Kriterien verstanden sein...

## Attribute und Funktionale Abhangigkeit

### Beispiel Artikel-Datenbank

- Ähnlich wie bei Attributen im ER-Diagramm gehen wir von einer sinnvollen Aufteilung, d.h. separierten Sachverhalten, aus.
- Die Attribute/Spalten seien wie folgt abgekürzt:

Nr (N)	Verlag (V)	Magazin (M)	Ausgabe (A)	Gründung (G)	Seite (S)	Thema (T)
aa11	Heise	c't	11/18	1949	15	C++
aa11	Heise	c't	11/18	1949	24	C#
bb22	Heise	c't	12/18	1949	12	Pvthon

N=Nr, V=Verlag, M=Magazin, G=Grundung,  
A=Ausgabe, S=Seite, T=Thema  
kurz: R={N, V, M, A, S, T, G}

### Q&A

- Welche Attribute kann man aus anderen ableiten? Bzw.
- Für welche Attribute gilt: Kennt man dieses, kennt man auch jenes?

# Attribute und Funktionale Abhängigkeit

---

## Beispiel Artikel-Datenbank

- Abhängige Information/Attribute:
  - Wenn die (ISBN)Nr. (N) feststeht, dann sind der Verlag (V) mit Gründung (G), das Magazin (M) und die Ausgabe (A) klar.
  - Aber das Thema (T) steht erst fest, wenn z.B. die Nr. (N) und Seite (S) genannt sind.
  - Umgekehrt: Wenn der Verlag (V) bekannt ist, dann auch die Gründung (G).
  - Und: Wenn das Magazin (M) genannt wird, dann leitet sich aus dieser Information der Verlag (V) ab.
- (Funktionale) Abhängigkeiten in Kurzform:  $N \rightarrow VMAG$ ,  $NS \rightarrow T$ ,  $V \rightarrow G$ ,  $M \rightarrow V$ .



$X \rightarrow Y$  bedeutet: Wenn X bekannt ist, dann auch Y.

Nr (N)	Verlag (V)	Magazin (M)	Ausgabe (A)	Gründung (G)	Seite (S)	Thema (T)
aa11	Heise	c't	11/18	1949	15	C++
aa11	Heise	c't	11/18	1949	24	C#
bb22	Heise	c't	12/18	1949	12	Pvthon

## Attribute und Funktionale Abhangigkeit

---

### Ausgangssituation der folgenden Untersuchungen

- Gegeben ist eine Relation R mit (vereinfachtem) Schema { A, B, C,... }, und
- eine Menge von Abhangigkeiten der Form  $X \rightarrow Y$ , d.h. kennt man X, dann auch Y, wobei X und Y jeweils fur eins der Attribute des Schemas stehen.



Die konkrete Bedeutung von X bzw. Y ist *nicht relevant*, man stelle sich den Verlag oder ein Thema vor!  
Es geht vielmehr um die *Zusammenhange* der Informationen.

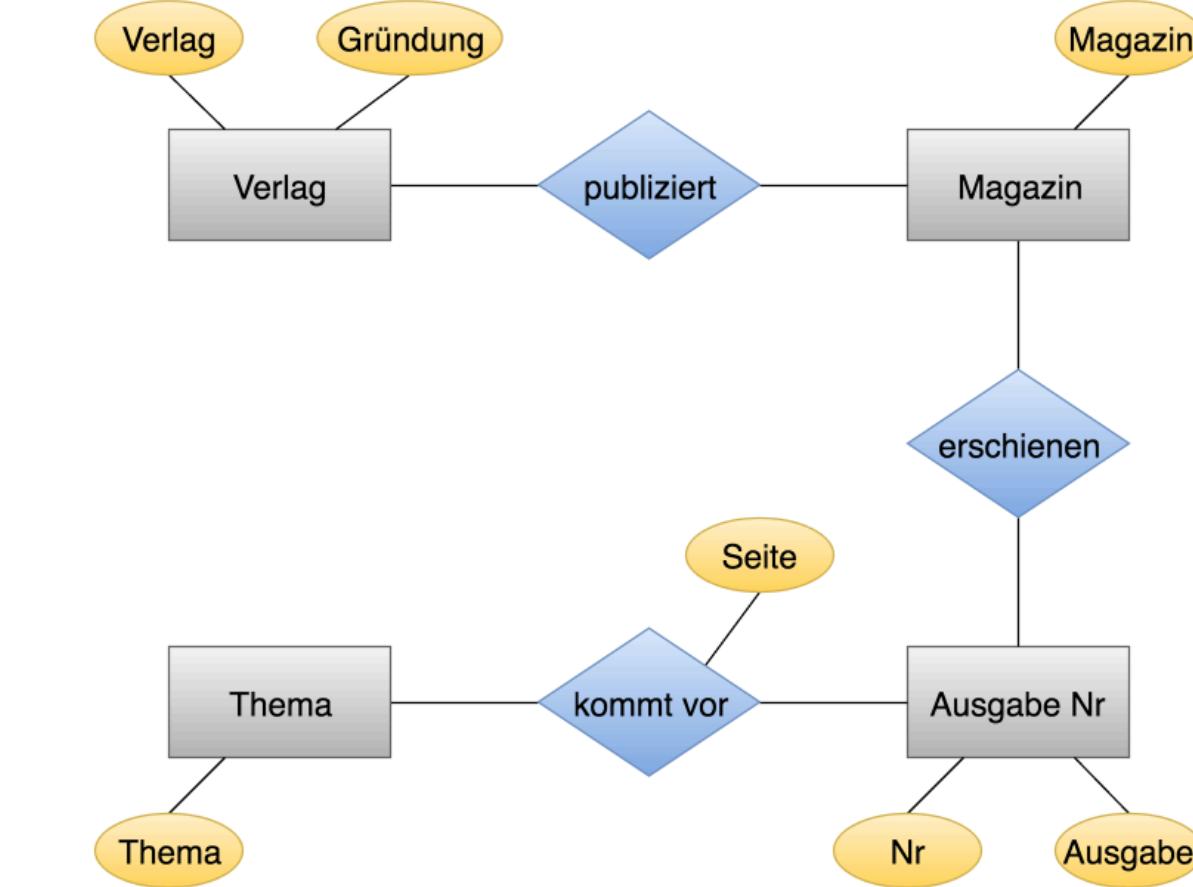
#### Q&A

- Aus welchen Attributen folgen alle anderen?
- Kann man die Menge der Abhangigkeiten vereinfachen?
- Wie sehen die Relationen bzw. Tabellen am Ende aus?

## Wiederholung/Ziel

### Zwei Ansätze

- Zu Beginn haben wir die Artikeldatenbank (intuitiv) in verschiedene Tabellen überführt, nur mit Überlegungen zu Informationsseparation und Redundanzreduktion. Es gab keine Mini-Welt, sondern nur diesen Ansatz.
- Eine Mini-Welt können wir in ein ER-Diagramm und dann in einen Implementationsentwurf überführen.



Die Überlegungen zu Normalformen zeigen im Folgenden, dass beide Ansätze systematisch zum gleichen Ergebnis kommen – einer 'guten' Datenbank.



### DB->normalizer (TUM)

- Tool, um diese Überführung zu demonstrieren und zum Üben! (Link am Ende).

## DB->normalizer (TUM)

Die 'Relation' sind die verwendeten Attribute und die 'FDs' die Abhängigkeiten aus unserem Beispiel. Details und Definitionen folgen, jetzt erstmal der Effekt:

Relation eingeben

NVMASTG

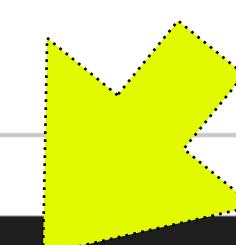
FDs/MVDs eingeben

N->VMAG

V->G

M->V

NS->T



Ergebnis anzeigen

Quiz

Schema speichern

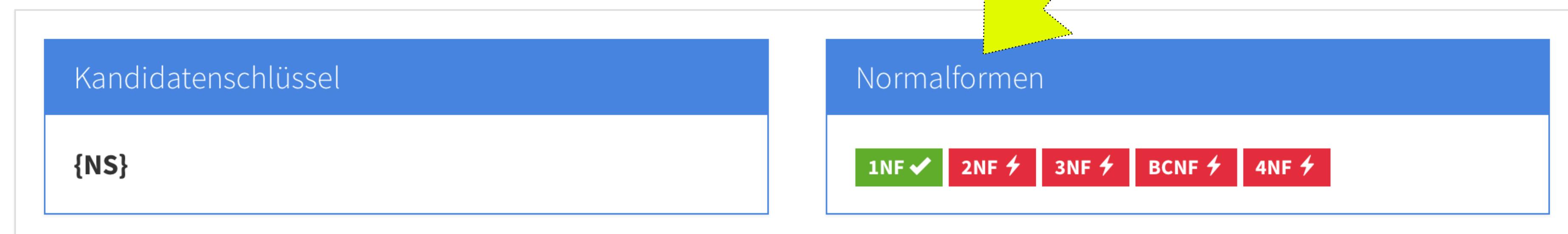
Schema laden ▾

## DB->normalizer (TUM)

### Eingabe



### Ergebnis



## DB->normalizer (TUM)

➤ Kanonische Überdeckung

### 1 Linksreduktion

$N \rightarrow AGMV$   
 $V \rightarrow G$   
 $M \rightarrow V$   
 $NS \rightarrow T$

### 3 $a \rightarrow \emptyset$ entfernen

$N \rightarrow AM$   
 $V \rightarrow G$   
 $M \rightarrow V$   
 $NS \rightarrow T$

### 2 Rechtsreduktion

$N \rightarrow AM$   
 $V \rightarrow G$   
 $M \rightarrow V$   
 $NS \rightarrow T$

### 4 FDs zusammenfassen

**$N \rightarrow AM$**   
 **$V \rightarrow G$**   
 **$M \rightarrow V$**   
 **$NS \rightarrow T$**

Algorithmen



Ergebnis

## DB-&gt;normalizer (TUM)

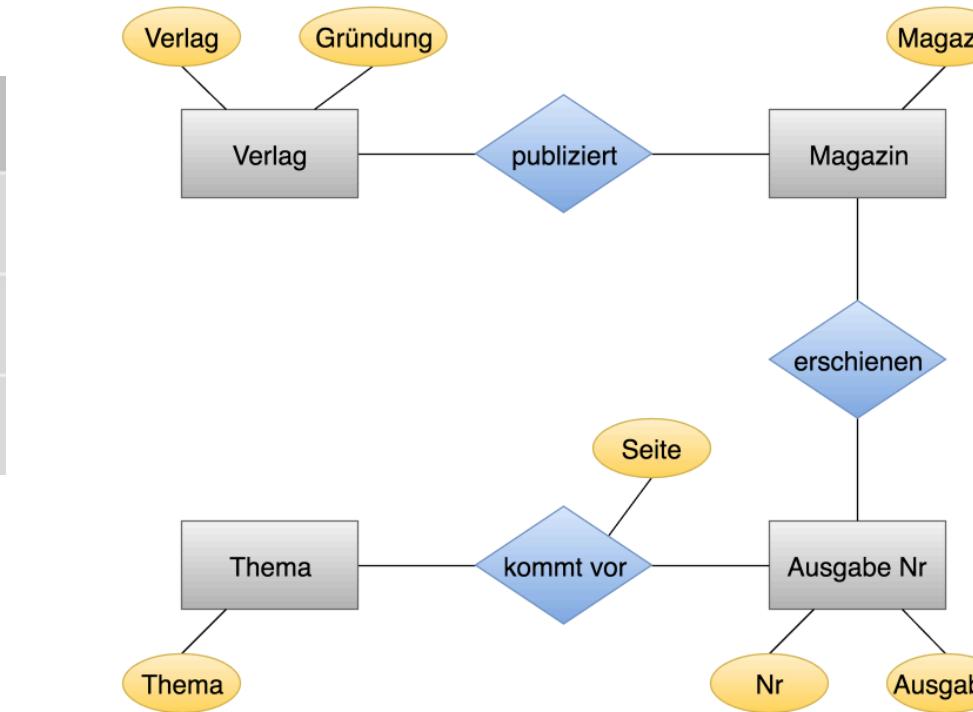
- Die Darstellung fokussiert weniger die IDs und Relationen sondern primär die Aufteilung der Informationen.
- Eine Abhängigkeit, etwa  $V \rightarrow G$ , ist dann realisiert, wenn beide Attribute in einer Tabelle stehen (mit geeigneten Schüsseln).

4 FDs zusammenfa

Algorithmen

**N->AM****V->G****M->V****NS->T****[V,G]**

Verlag (V)	Gründung (G)
Heise	1949
Computec	1989
Webedia	2007

**Nachdenken****[M,V]**

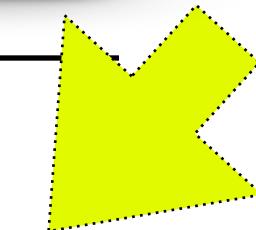
V-ID	Magazin (M)
101	c't
101	iX
102	buffed
103	GameStar

M-ID	Nr (N)	Ausgabe (A)
301	aa11	11/18
301	bb22	12/18
302	cc33	08/18
303	dd44	08/18
304	ee55	08/18

**[N,S,T]**

T-ID	Thema (T)
201	C++
202	C#
203	Python
204	RAID
205	gpio
206	WoW
207	LoL

N-ID	T-ID	Seite (S)
401	201	15
401	202	24
402	203	12
402	202	29
403	204	9
403	205	23
404	206	11
405	207	13
405	206	20

**[N,M,A]**

# Funktionale Abhangigkeit

## Definition

- Sei  $R=\{a_1, \dots, a_n\}$  ein vereinfachtes Schema,  $\alpha \subseteq \text{ident}(R)$ ,  $\beta \subseteq \text{ident}(R)$  und  $D \subseteq \text{dom}(R)$ . Dann ist  $\beta$  **funktional abhangig von**  $\alpha$  genau dann, wenn fur alle zulassigen  $D$  gilt:

$$\forall s, t \in D : s.\alpha = t.\alpha \Rightarrow s.\beta = t.\beta,$$

in Zeichen  $\alpha \rightarrow \beta$ . Hierbei meint  $s.\alpha = t.\alpha$  die Gleichheit in allen Attributen  $a \in \alpha$ .

$\alpha \rightarrow \beta$  heit, fur alle Tupel  $s, t$  mit gleichen  $\alpha$ -Attributen gilt, dass auch ihre  $\beta$ -Attribute gleich sind. Die Werte der Attribute aus der Attributmenge  $\alpha$  bestimmen also eindeutig die Werte der Attribute aus der Attributmenge  $\beta$ .

Bisher Verlag → Grundung: Kennt man den Verlag ( $\alpha$ ) so auch das Grundungsjahr ( $\beta$ ).  
Obige Definition: bei gleichem Verlag ( $s.\alpha$ ) ist das Grundungsjahr ( $t.\beta$ ) gleich.



# Funktionale Abhängigkeit

---

## Anmerkungen zur Definition

- *Funktionale Abhängigkeit*, oder FD für *Functional Dependencies*, beschreibt eine Eigenschaft der *realen Welt*, nicht der gerade vorliegenden Daten. Diese dürfen dazu aber natürlich nicht im Widerspruch stehen!

Beispiel: Betrachte zu  $R=\{A,B,C\}$  und den funktionalen Abhängigkeiten  $f_1: \{A\} \rightarrow \{B,C\}$  und  $f_2: \{C\} \rightarrow \{B\}$  diese Datenmenge D. Dann ist D verträglich mit  $f_1$ , aber  $f_2$  kann nicht gelten, da einmal b1 und einmal b2 aus c1 folgt. Weiter kann nicht einfach  $\{B\} \rightarrow \{A\}$  geschlossen werden, nur weil D gerade diese Ausprägung hat.

A	B	C
a1	b1	c1
a2	b3	c2
a3	b2	c1
a1	b1	c1

$D \subseteq \text{dom}(R)$

- Statt ' $\{A\} \rightarrow \{B,C\}$ ' schreibt man auch kurz ' $A \rightarrow BC$ ', statt ' $R=\{A,B,C\}$ ' auch ' $R=ABC$ '.

$AB \rightarrow CD$ : Wenn A und B bekannt, dann auch C und D.  
Nur Kenntnis von A oder B alleine sagt nichts aus!



## Funktionale Abhangigkeit

---

### Definition

- Sei  $R=\{a_1, \dots, a_n\}$  ein vereinfachtes Schema und  $\alpha \rightarrow \beta$  eine funktionale Abhangigkeit.  
Dann ist  $\beta$  **voll funktional abhangig von  $\alpha$**  wenn

$$\forall \gamma \in \mathcal{P}(\alpha) - \alpha: \alpha - \gamma \rightarrow \beta$$

in Zeichen:  $\alpha \rightarrow \beta$ . Das bedeutet,  $\alpha$  ist minimal.

Volle funktionale Abhangigkeit bedeutet, dass  $\alpha$  minimal ist, man also kein Attribut aus  $\alpha$  weglassen kann, ohne dass man die funktionale Abhangigkeit verliert.



## Schlüssel

---

### Definition

- Sei  $R=\{a_1, \dots, a_n\}$  ein vereinfachtes Schema und  $\alpha \subseteq \text{ident}(R)$ . Dann ist
  - $\alpha$  ein **Superschlüssel**, falls  $\alpha \rightarrow \text{ident}(R)$ , und
  - $\alpha$  ein **Schlüsselkandidat**, falls  $\alpha \dot{\rightarrow} \text{ident}(R)$ .
- Ein **Primärschlüssel** ist ein ausgesuchter/festgelegter Schlüsselkandidat.
- Ein Attribut  $a \in \text{ident}(R)$  heißt **prim**, falls  $a$  Attribut eines Schlüsselkandidaten von  $R$  ist, sonst **nicht prim**.



Funktionale Abhängigkeit aller Attribute vom Primärschlüssel bedeutet nichts anderes als dass man die gesamte Entität kennt, wenn man den Schlüssel hat... in volliger Übereinstimmung mit unserer Idee einer id.

## Schlüssel

---

### Anmerkungen zur Definition

- Es macht Sinn, aus den möglichen Schlüsselkandidaten einer Relation R genau *einen* Primärschlüssel festzulegen, da Verweise auf Tupel aus R über sog. *Fremdschlüssel* in anderen Relationen realisiert werden, also Schlüsselattribute dieses Primärschlüssels.
- Im DBMS werden Primärschlüssel explizit gekennzeichnet und führen dazu, dass sie in der Tabelle nicht `null` sein oder doppelt vorkommen dürfen. Daraus ergibt sich insbesondere, dass in der Tabelle keine zwei Tupel identisch sind (Eindeutigkeit).
- Nicht jedes  $\alpha$  mit  $\alpha \rightarrow \text{ident}(R)$  ist ein *Schlüsselkandidat*, denn  $\alpha$  muss nicht minimal sein! Insbesondere ist  $\alpha = \text{ident}(R)$  der triviale Superschlüssel.



Achtung: Es kann beispielsweise Schlüsselkandidaten AB und BCD geben, auch wenn BCD 'länger' als AB ist... die *Minimalität* bedeutet hier, dass man weder A oder B aus AB noch B,C oder D aus BCD weglassen kann, ohne die Schlüsseleigenschaft zu verlieren!

## Kurzer Halt

---



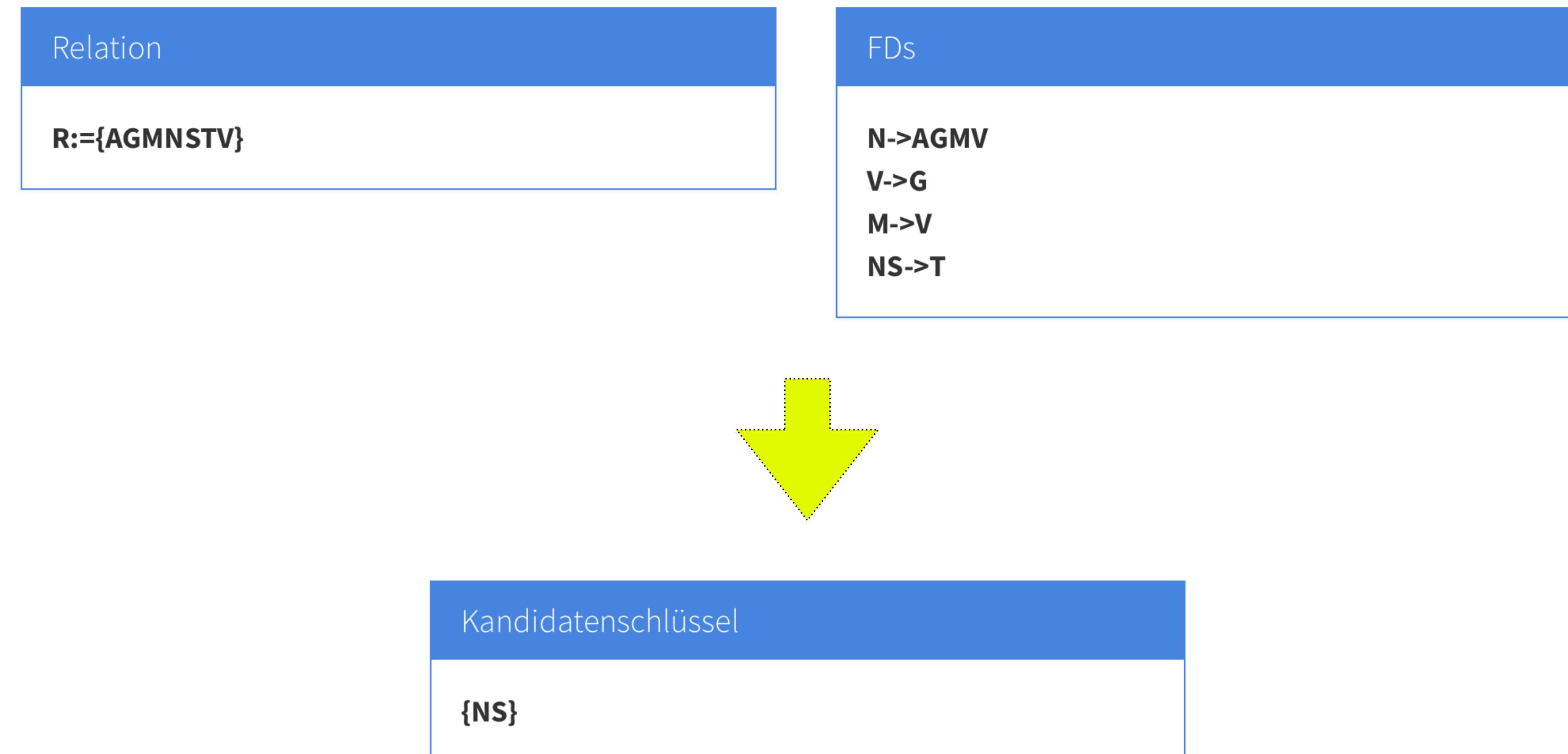
Die bisherigen Definitionen sind wichtig, ohne sie versteht man den Rest nicht mehr!

- A oder B stehen für reale Attribute wie der Verlag (V) oder die Gründung (G) in der Artikeldatenbank.
- R=ABCD, FDs={ A→B, AB→CD }.
- Superschlüssel, Schlüsselkandidat, Primärschlüssel, prim, minimal.

## Kurzer Halt

---

Nochmal ein Blick auf den Normalizer mit den Definitionen im Kopf:



## Armstrong-Axiome

---

### Motivation

- Gegeben sei ein Schema  $R=\{A,B,C,D\}$  mit FDs= $\{ A \rightarrow B, B \rightarrow C, AB \rightarrow CD \}$

#### Q&A

- Gilt  $A \rightarrow C$ ?
- Aus welchen Attributen folgen alle anderen Attribute?  
Anders: Wie lauten die Schlüsselkandidaten?
- Kann man die FDs 'vereinfachen'?  
Anders: Kann man FDs weglassen oder 'kürzen' bei gleicher 'Aussagekraft'?

## Armstrong-Axiome

---

### Motivation

- Gegeben sei ein Schema  $R=\{A,B,C,D\}$  mit FDs= $\{ A \rightarrow B, B \rightarrow C, AB \rightarrow CD \}$

### Gilt $A \rightarrow C$ ?

- Ja, denn ist A bekannt, dann auch B (wg.  $A \rightarrow B$ ) und dann auch C (wg.  $B \rightarrow C$ ); genannt *Transitivität*
- Es gibt offenbar Regeln, nach denen man aus den FDs oben weitere ableiten kann - die sog. **Armstrong-Axiome** (folgen).

## Armstrong-Axiome

---

### Motivation

- Gegeben sei ein Schema  $R=\{A,B,C,D\}$  mit FDs= $\{ A \rightarrow B, B \rightarrow C, AB \rightarrow CD \}$

### Wie lauten die Schlüsselkandidaten?

- Wir suchen grundsätzlich einzelne Attribute oder Attributkombinationen, aus denen alle anderen Attribute folgen und wo wir nichts weglassen können.
- Dazu starten wir mit einem Attribut, z.B. A, und sammeln zusammen, was folgt. Danach probieren wir weitere Attribute und ggf. Kombinationen, bis alle Schlüsselkandidaten bekannt sind.
- Dieses qualifizierte Raten kann man auf unterschiedliche Weise und optimiert durchführen, aber am Ende muss man ein paar Kombinationen ausprobieren...

## Armstrong-Axiome

---

### Motivation

- Gegeben sei ein Schema  $R=\{A,B,C,D\}$  mit FDs= $\{ A \rightarrow B, B \rightarrow C, AB \rightarrow CD \}$

### Wie lauten die Schlüsselkandidaten? Fortsetzung

- Zu A: Aus A folgt B ( $A \rightarrow B$ ), und dann C ( $B \rightarrow C$ ). Da wir mit A auch B kennen, folgen C und D ( $AB \rightarrow CD$ ) und insgesamt  $A \rightarrow ABCD$ . Also ist A ein (erster) Schlüsselkandidat.
- B: Aus B folgt nur C, nicht A oder D.
- C, D: Weder aus C noch aus D folgen weitere Attribute.
- Weitere Kombinationen mit A, etwa ABC, sind Superschlüssel aber nicht Schlüsselkandidat, denn man kann alles ausser A weglassen und somit sind sie nicht minimal.
- Weitere Kombinationen ohne A, also BC,CD und BCD, führen nie zu A.

A ist einziger Schlüsselkandidat!



## Armstrong-Axiome

---

### Motivation

- Gegeben sei ein Schema  $R=\{A,B,C,D\}$  mit FDs= $\{ A \rightarrow B, B \rightarrow C, AB \rightarrow CD \}$

### Kann man die FDs 'vereinfachen'?

- Ja, z.B. die 'kürzeren' FDs
    - $\{ A \rightarrow B, B \rightarrow C, A \rightarrow CD \}$  (da  $A \rightarrow B$  gilt, ist  $A \rightarrow CD$  'so gut wie'  $AB \rightarrow CD$ )
    - $\{ A \rightarrow B, B \rightarrow C, A \rightarrow D \}$  (da  $A \rightarrow B$  und  $B \rightarrow C$  gilt, ist  $A \rightarrow D$  'so gut wie'  $AB \rightarrow CD$ )
    - $\{ A \rightarrow BD, B \rightarrow C \}$  (da zuvor  $A \rightarrow B$  und  $A \rightarrow D$  gilt, auch  $A \rightarrow BD$ )
- sagen das gleiche aus.

### Q&A

- Wie ist 'so gut wie' definiert?

## Armstrong-Axiome

---

Mit Hilfe der **Armstrong-Axiome** lassen sich funktionale Abhangigkeiten aus einer Menge von FDs ableiten (wie zuvor bei der Transitivitat). Die folgenden drei Regeln reichen aus, um alle funktionalen Abhangigkeiten herzuleiten:

### Definition

- Sei  $R=\{a_1, \dots, a_n\}$  ein vereinfachtes Schema und  $\alpha, \beta, \gamma \subseteq \text{ident}(R)$ . Die **Armstrong-Axiome** sind gegeben durch:
  - **Reflexivitat**: fur alle  $\beta \subseteq \alpha$  gilt:  $\alpha \rightarrow \beta$  (triviale Abhangigkeit).
  - **Transitivitat**: gilt  $\alpha \rightarrow \beta$  und  $\beta \rightarrow \gamma$ , so auch  $\alpha \rightarrow \gamma$ .
  - **Anreicherung/Verstarkung**: gilt  $\alpha \rightarrow \beta$ , so auch  $\alpha\gamma \rightarrow \beta\gamma$ .
- Strenggenommen sind es keine 'Axiome' im mathematischen Sinne, denn sie lassen sich aus der Definition der funktionalen Abhangigkeit ableiten – heissen aber so.

## Armstrong-Axiome

---

Aus den gegebenen Armstrong-Axiomen können weitere Regeln abgeleitet werden, um ggf. Herleitungen zu vereinfachen.

### Definition

- Sei  $R=\{a_1, \dots, a_n\}$  ein vereinfachtes Schema und  $\alpha, \beta, \gamma, \delta \subseteq \text{ident}(R)$ . Es gilt weiter:
  - **Vereinigung:** gilt  $\alpha \rightarrow \beta$  und  $\alpha \rightarrow \gamma$ , so auch  $\alpha \rightarrow \beta\gamma$ .
  - **Dekomposition:** gilt  $\alpha \rightarrow \beta\gamma$ , so auch  $\alpha \rightarrow \beta$  und  $\alpha \rightarrow \gamma$ .
  - **Pseudotransitivität:** gilt  $\alpha \rightarrow \beta$  und  $\beta\gamma \rightarrow \delta$ , so auch  $\alpha\gamma \rightarrow \delta$ .

## Armstrong-Axiome

---

### Erinnerung

- Gegeben war ein Schema  $R=\{A,B,C,D\}$  mit FDs= $\{ A \rightarrow B, B \rightarrow C, AB \rightarrow CD \}$ .

### Wie lauten die Schlüsselkandidaten?

- Wir haben A hergenommen und überlegt, was aus den FDs folgt. Das ist aber nichts anderes, als die Regeln, also die Armstrong-Axiome, anzuwenden.
- Wir haben hier die sog. Attributhülle von A bzgl. FD berechnet (folgt sofort).

### Kann man die FDs 'vereinfachen'?

- Wir haben uns Regeln überlegt, die 'so gut wie' andere sind - ebenfalls Armstrong-Axiome. Das führt auf die sog. Kanonische Überdeckung (folgt in Teil II).

# Attributhülle

---

## Definition

- Zu einem Schema R und einer Menge FD (manchmal F) funktionaler Abhängigkeiten bestimmt die **Attributhülle**
  - ♦  $\text{AttrHülle}(FD, \alpha)$ , oder  $\alpha^+$ , wenn FD klar, alle Attribute, die bzgl. FD funktional abhängig von  $\alpha$  sind.
- Achtung: Algorithmus terminiert spätestens, wenn Ergebnismenge alle Attribute enthält. In diesem Fall ist  $\alpha$  sogar Superschlüssel.
- Achtung: Initiale Menge ist immer  $\alpha$  selbst (Reflexivität).

```

// ----- AttrHülle(F, α) -----
// Eingabe:
//   F: Menge von funktionalen Abhängigkeiten
//   α: Menge von Attributen
// Ausgabe:
//   transitiver Abschluss von α bzgl. F → α+

```

```

AttrHülle(F,α) {
    Erg = {α} ; ErgAlt = ∅
    while (Erg <> ErgAlt) {
        ErgAlt = Erg;
        foreach (β → γ in F)
            if (β ⊆ Erg) then Erg = Erg ∪ {γ}
    }
    return Erg
}

```

## Attributhülle

---

### Beispiel

Schema  $R=\{A,B,C,D\}$ ,  $FD=\{ A \rightarrow B, B \rightarrow C, AB \rightarrow CD \}$ , gesucht sind:

- AttrHülle( $FD, A$ ):  $A \rightarrow A \rightarrow AB \rightarrow ABC \rightarrow ABCD$  fertig

wg. Start  $A \rightarrow B \quad B \rightarrow C \quad AB \rightarrow CD$

d.h.  $A^+ = ABCD = \text{ident}(R)$ .

- AttrHülle( $FD, B$ ):  $B \rightarrow B \rightarrow BC$  fertig

wg. Start  $B \rightarrow C$

d.h.  $B^+ = BC \neq \text{ident}(R)$ .

- AttrHülle( $FD, C$ ):  $C \rightarrow C$  fertig

d.h.  $C^+ = C$ .

- AttrHülle( $FD, D$ ):  $D \rightarrow D$  fertig

d.h.  $D^+ = D$ .

### Q&A

- Wie schreibt man das auf, falls mal jemand fragt... ?

## Attributhülle

---

### Beispiel

Schema  $R=\{A,B,C,D\}$ ,  $FD=\{ A \rightarrow BCD, CD \rightarrow A \}$ , gesucht sind alle Attributhüllen:

- $\text{AttrHülle}(FD,A)$ :  $A \rightarrow A \rightarrow ABCD$  fertig, also  $A^+ = ABCD = \text{ident}(R)$  (Schlüsselkandidat).
- $AB^+$ ,  $AC^+$ ,  $AD^+$  alles Superschlüssel.
- $B^+$ ,  $C^+$ ,  $D^+$ ,  $BC^+$ ,  $BD^+$  alle trivial und insbes.  $\neq \text{ident}(R)$ .
- $\text{AttrHülle}(FD,CD)$ :  $CD \rightarrow CD \rightarrow ACD \rightarrow ABCD$ , also  $CD^+ = \text{ident}(R)$  (Schlüsselkandidat).

Der Algorithmus zur Bestimmung der Attributhülle liefert, auf alle Kombinationen angewandt, *alle* Schlüsselkandidaten.



Achtung: A und CD sind Schlüsselkandidaten, obwohl CD 'länger' als A ist... aber beide sind minimal.

## Attributhülle

---

### Bestimmung der Schlüsselkandidaten

- Es gibt im Wesentlichen zwei Möglichkeiten:
  - Alle Möglichkeiten ausprobieren.
  - Mit  $\text{ident}(R)$  anfangen und sinnvoll streichen (folgt bei *Kanonischer Überdeckung*).
- Problem ist, dass *alle Möglichkeiten* sehr viele sein können... aber:
  - Man kann geschickt anfangen, indem man zuerst alle Attribute berücksichtigt, die nicht gefolgt werden können (falls es sie gibt, Beispiel folgt).
  - Im Laufe der Berechnung einer Attributhülle ergeben sich Attributkombinationen, die z.B. einen Schlüsselkandidaten enthalten - dann ist man auch fertig, oder die man zuvor schon berücksichtigt hat.

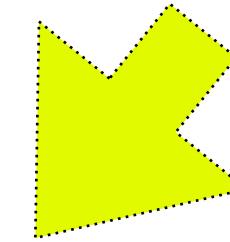


Insgesamt sieht es nach mehr Arbeit aus, als es ist.

## Attributhülle

---

### Beispiel



Schema  $R=\{A,B,C,D\}$ ,  $FD=\{ A \rightarrow BC \}$ , gesucht sind alle Schlüsselkandidaten:

- Grundidee: Wir benötigen  $\alpha \rightarrow \text{ident}(R)$ , d.h. *alle* Attribute müssen gefolgt werden.

#### Q&A

- Wie soll D in die Attributhülle gelangen?

- Kurz gesagt: Gar nicht, wenn es nicht schon drin ist.



Alle Attribute, die *nicht* auf einer rechten Seite vorkommen, müssen in die Schlüsselkandidaten!

- Es kann passieren, dass alle auf einer rechten Vorkommen... das ist dann Pech.

## Attributhülle

---

### Fortsetzung Beispiel

Schema  $R=\{A,B,C,D\}$ ,  $FD=\{ A \rightarrow BC \}$ , gesucht sind alle Schlüsselkandidaten:

- Es ist  $D \in \text{ident}(R)$ , aber D kommt auf keiner rechten Seite vor. D.h. alle Kombinationen ohne D, insbes. A, B, C, können kein Schlüsselkandidat sein.
- Der einzige mögliche Schlüsselkandidat mit nur einem Attribut ist D, aber man sieht, dass das wegen  $D^+=D$  nicht reicht.
- Der nächstbeste Kandidat ist folglich AD:  $AD^+ = AD \rightarrow ABCD = \text{ident}(R)$ , also Schlüsselkandidat.
- Da in FD nur aus A etwas folgt, ist man dann hier auch fertig.
- Insgesamt ist somit AD einziger Schlüsselkandidat und A und D sind *prim*, B und C sind *nicht prim*.

## Attributhülle

---

### Beispiel

Schema  $R=\{A,B,C,D\}$ ,  $FD=\{ A \rightarrow BCD, CD \rightarrow A \}$ , gesucht sind alle Schlüsselkandidaten. Man kann wie folgt argumentieren (statt alles blind durchzuprobieren):

- Zunächst: Kein Attribut fehlt auf der rechten Seite – Schade.
- A ist offensichtlicher Schlüsselkandidat, CD aufgrund der Transitivität ebenso.
- Man sieht in FD, dass weder aus B, noch aus C oder D alleine  $ABCD=\text{ident}(R)$  folgt, also B,C und C keine Schlüsselkandidaten.
- Jede Obermenge von A, also AB, AC etc., ist Superschlüssel, aber nicht minimal, da A reicht. Gleiches gilt für Obermengen von CD und es bleibt nichts mehr übrig zu untersuchen.
- Insgesamt sind A und CD Schlüsselkandidaten; A,C,D prim, B nicht prim.

## Attributhülle

---

### Beispiel

Schema  $R=\{A,B,C\}$ ,  $FD=\{ A \rightarrow B, B \rightarrow C, C \rightarrow A \}$ , gesucht sind alle Schlüsselkandidaten:

- Zunächst: Kein Attribut fehlt auf der rechten Seite – Schade.
- Man sieht:  $A^+=B^+=C^+=ABC=\text{ident}(R)$ .
- Mehr Kombinationen müssen nicht probiert werden, da alle Kombinationen Obermengen von A, B oder C sind.
- Insgesamt sind A, B und C Schlüsselkandidaten und alle Attribute *prim*.

## Zusammenfassung

---

### Roter Faden

- Armstrong-Axiome und Attributhülle: Grundlage für Umformungen der FDs.
- Kanonische Überdeckung, Synthesealgorithmus, Dekompositionsalgorithmus, Gütekriterien Normalformen folgen in Teil II und III.
- Link zum Normalizer: <https://normalizer.db.in.tum.de/index.py>

# UNIT 0x08

## NORMALFORMEN II

## KANONISCHE ÜBERDECKUNG

## Motivation/Erinnerung

---

### Aufgabe

- Entwicklung einer 'guten' Datenbank [...] u.a. minimal, redundanzfrei [...]

### Vorgehen

- Konzeptueller Entwurf [...] ER-Modells ✓
- Überführung in eine 'gute' Datenbank [...] dazu benötigen wir:
  - Implementationsentwurf [...] → Relationales Modell ✓
  - Mathematisches Modell [...] → Relationale Algebra ✓
  - Gütemaß → Normalformen
    - ♦ Kanonische Überdeckung (funktionale Abhängigkeiten ✓, Attributhülle ✓),
    - ♦ Synthesealgorithmus,
    - ♦ Dekompositionsalgorithmus

## Äquivalente Menge funktionaler Abhängigkeiten

### Motivation

- Gegeben sei ein Schema  $R=\{A,B,C,D\}$  mit  $FD=\{ A \rightarrow B, B \rightarrow C, AB \rightarrow CD \}$

### Kann man die FDs 'vereinfachen'?

- 'vereinfachen' bedeuten i.d.R., so viel wie möglich zu streichen, ohne die Bedeutung zu ändern bzw. hier einen 'äquivalenten' ('so gut wie') Ausdruck mit weniger oder 'kleineren' funktionalen Abhängigkeiten zu erhalten. 'kleiner' bedeutet wiederum, Attribute auf den *rechten und linken Seiten* der funkt. Abhängigkeiten weglassen zu können → Rechts- und Linksreduktion (folgen).
- Hier: Da  $A \rightarrow B$  gilt, ist  $AB \rightarrow CD$  'so gut wie'  $A \rightarrow CD$ , und daher  $FD'=\{ A \rightarrow B, B \rightarrow C, A \rightarrow CD \}$  'äquivalent' zu obiger Menge FDs.

### Q&A

- Wie formal konkretisieren?

## Äquivalente Menge funktionaler Abhängigkeiten

---

### Definition

- Zu einem Schema R und einer Menge FD funktionaler Abhängigkeiten bezeichne
  - ♦  $\mathbf{FD}^+$   
die Menge aller funktionalen Abhängigkeiten, die von FD impliziert werden.
- Anstatt die Armstrong-Axiome zu bemühen, überlege man sich, dass gilt
  - ♦  $\alpha \rightarrow \beta \in \mathbf{FD}^+$  genau dann, wenn  $\beta \subseteq \text{AttrHülle}(\mathbf{FD}, \alpha)$ .
- Zwei Mengen funktionaler Abhängigkeiten  $\mathbf{FD}_1$  und  $\mathbf{FD}_2$  heißen **äquivalent**, falls
  - ♦  $\mathbf{FD}_1^+ = \mathbf{FD}_2^+$ ,        in Zeichen  $\mathbf{FD}_1 \equiv \mathbf{FD}_2$ .
- Die kanonische Überdeckung ist in einem gewissen Sinne eine *minimale äquivalente* Menge funktionaler Abhängigkeiten (Definition folgt).
- Technisch: Manchmal schreiben wir auch etwas ungenau  $\beta \in \text{AttrHülle}(\mathbf{FD}, \alpha)$ .

## Äquivalente Menge funktionaler Abhängigkeiten

---

### Beispiel FD<sup>+</sup>

- Geg. sei Schema  $R=\{A, B, C, D\}$ ,  $FD=\{ A \rightarrow B, B \rightarrow C, AB \rightarrow CD \}$ , dann ist
  - $FD^+ = \{ A \rightarrow ABCD, B \rightarrow BC, C \rightarrow C, D \rightarrow D, AB \rightarrow ABCD, AC \rightarrow ABCD, AD \rightarrow ABCD, BC \rightarrow BC, BD \rightarrow BD, CD \rightarrow CD, ABC \rightarrow ABCD, ABD \rightarrow ABCD, BCD \rightarrow BCD, ABCD \rightarrow ABCD \}$   
(hier sind  $A \rightarrow B$  und  $A \rightarrow C$  zu  $A \rightarrow BC$  etc. zusammengefasst).
- Spaltet man die nicht-trivialen Abhängigkeiten ab, ergeben sich die Darstellungen
  - $FD^+ = \{ A \rightarrow BCD, B \rightarrow C \} \cup \{ \text{triviale Abhängigkeiten} \}$ , bzw.
  - $FD^+ = \{ A \rightarrow BD, B \rightarrow C \} \cup \{ \text{triviale Abhängigkeiten} \} \cup \{ \text{transitive Abhängigkeiten} \}$ .
- $FD^+$  ergibt sich, per Definition, durch die Betrachtung aller Attributhüllen bzw. der Vereinigung der dazugehörigen Abhängigkeiten.

### Q&A

- Wie sieht wohl eine kleinste, äquivalente Menge aus...?

## Äquivalente Menge funktionaler Abhängigkeiten

---

### Beispiel Äquivalenz

- Geg. sei Schema  $R=\{A, B, C, D\}$ ,  $FD=\{ A \rightarrow B, B \rightarrow C, AB \rightarrow CD \}$ , dann ist
  - $FD'=\{ A \rightarrow B, B \rightarrow C, A \rightarrow CD \}$  äquivalent zu  $FD$ , da gilt  $FD'^+=FD^+$  (nachrechnen), oder aber durch Betrachtung der im Vergleich relevanten Attributhüllen, also hier
    - ♦  $CD \subseteq \text{AttrHülle}(FD, AB)$ , und
    - ♦  $CD \subseteq \text{AttrHülle}(FD', A)$ .
- Anders ausgedrückt: Da die Attributhüllen in beiden Fällen gleich sind, sind auch die funktionalen Abhängigkeiten, die sich einmal aus  $AB \rightarrow CD$  und einmal aus  $A \rightarrow CD$  ergeben, dieselben. Die restlichen funktionalen Abhängigkeiten sind unberührt, also gilt wiederum  $FD'^+=FD^+$ .
- Generell: Wenn wir Attribute weglassen, so untersuchen wir eine Menge  $FD'$ , primär gegeben durch  $FD' = FD \setminus \{\text{untersuchte funkt. Abh.}\} \cup \{\text{modifizierte funkt. Abh.}\}$ .

## Äquivalente Menge funktionaler Abhängigkeiten

### Äquivalente Minimierung

- Wir möchten eine 'kleinere' aber äquivalente Menge mit weniger Attributen finden.  
**D.h. wir versuchen, Attribute wegzulassen und sehen, ob das, was übrig bleibt, äquivalent ist. Das kann man für die rechten und linken Seiten machen.**
- Geg. sei Schema  $R=\{A, B, C, D\}$ ,  $FD=\{ A \rightarrow B, B \rightarrow C, AB \rightarrow CD \}$ . Wir starten *links* und betrachten nur die funktionalen Abhängigkeiten, wo wir links auch Attribute weglassen können. Folglich untersuchen wir, ob wir  $AB \rightarrow CD$  durch  $A \rightarrow CD$  oder  $B \rightarrow CD$  ersetzen können. Es ergibt sich *mit der Argumentation von zuvor*:
  - $\{ A \rightarrow B, B \rightarrow C, A \rightarrow CD \}$  äquivalent zu FD, da  $CD \subseteq \text{AttrHülle}(FD, A)$ .
  - $\{ A \rightarrow B, B \rightarrow C, B \rightarrow CD \}$  nicht äquivalent zu FD, da:  $CD \not\subseteq \text{AttrHülle}(FD, B) = BC$ .

Da wir links Attribute weggelassen haben, ist es eine Linksreduktion.

In den Überlegungen haben wir in der Attributhülle das zweite Argument variiert.



## Äquivalente Menge funktionaler Abhängigkeiten

### Äquivalente Minimierung

- Geg. sei das reduzierte Schema  $R=\{A, B, C, D\}$ ,  $FD=\{ A \rightarrow B, B \rightarrow C, A \rightarrow CD \}$ . Wir testen nun Attribute auf der *rechten* Seite. Folglich untersuchen wir u.a., ob wir  $A \rightarrow CD$  durch  $A \rightarrow C$  oder  $A \rightarrow D$  äquivalent ersetzen können.
  - $FD'=\{ A \rightarrow B, B \rightarrow C, A \rightarrow D \}$  ist äquivalent zu  $FD$ , da  $C \subseteq \text{AttrHülle}(FD', A)$ .  
Wir folgern also  $C$  aus den *anderen* funktionalen Abhängigkeiten und können es somit hier rechts auch weglassen.
  - $FD''=\{ A \rightarrow B, B \rightarrow C, A \rightarrow C \}$  ist nicht äquivalent zu  $FD$ , da  $D \not\subseteq \text{AttrHülle}(FD'', A)$ .  
Wir können hier  $D$  *nicht* aus den *anderen* funktionalen Abhängigkeiten folgern und es somit auch hier rechts nicht weglassen.

Da wir rechts Attribute weggelassen haben, ist es eine Rechtsreduktion.



In den Überlegungen haben wir in der Attributhülle das erste Argument variiert.

## Äquivalente Menge funktionaler Abhängigkeiten

---

### Äquivalente Minimierung

- Das ursprüngliche Schema  $R=\{A, B, C, D\}$ ,  $FD=\{ A \rightarrow B, B \rightarrow C, AB \rightarrow CD \}$  haben wir durch *Links- und Rechtsreduktion* in eine äquivalente und kürzere Darstellung
  - $\{ A \rightarrow B, B \rightarrow C, A \rightarrow D \}$überführt.
- Fasst man die funktionalen Abhängigkeiten noch zusammen (Vereinigung), ergibt sich die **kanonische Überdeckung**:
  - $\{ A \rightarrow BD, B \rightarrow C \}$ . (vgl. Frage zu Beginn)
- In dem Beispiel haben wir nur einzelne funktionale Abhängigkeiten von FD betrachtet. Allgemein muss man bei der kanonischen Überdeckung *alle* betrachten.

## Kanonischen Überdeckung

---

### Definition

- Zu einem Schema R und einer Menge FD funktionaler Abhangigkeiten bezeichne
  - ♦  $FD^C$die **kanonischen Überdeckung** von FD, falls folgende Kriterien erfüllt sind:
  - $FD^C \equiv FD$ , d.h.  $FD^{C+} = FD^+$ .
  - Alle  $\alpha \rightarrow \beta \in FD^C$  sind minimal in dem Sinne, dass
    - $\forall a \in \alpha$  gilt:  $FD^C \setminus (\alpha \rightarrow \beta) \cup (\alpha - a \rightarrow \beta) \not\equiv FD^C$
    - $\forall b \in \beta$  gilt:  $FD^C \setminus (\alpha \rightarrow \beta) \cup (\alpha \rightarrow \beta - b) \not\equiv FD^C$
  - alle linken Seiten kommen nur einmal vor (einzigartig, Vereinigungsregel).

## Algorithmus zur Berechnung der Kanonischen Überdeckung

---

### Berechnung der kanonischen Überdeckung $FD^c$

- Gegeben sind ein Schema R und eine Menge FD funktionaler Abhängigkeiten.
- **Linksreduktion:** Für alle  $\alpha \rightarrow \beta \in FD$  überprüfe, ob ein  $a \in \alpha$  überflüssig ist, d.h. ob
  - $\beta \subseteq \text{AttrH\"ulle}(FD, \alpha - a)$ .gilt. In diesem Fall wird  $\alpha \rightarrow \beta$  durch  $\alpha - a \rightarrow \beta$  in FD ersetzt.
- **Rechtsreduktion:** Für alle  $\alpha \rightarrow \beta \in FD$  überprüfe, ob ein  $b \in \beta$  überflüssig ist, d.h. ob
  - $\beta \subseteq \text{AttrH\"ulle}(FD \setminus \{\alpha \rightarrow \beta\} \cup \{\alpha \rightarrow \beta - b\}, \alpha)$gilt. In diesem Fall wird ebenfalls  $\alpha \rightarrow \beta$  durch  $\alpha \rightarrow \beta - b$  in FD ersetzt.
- **Entfernung** mglw. entstandener  $\alpha \rightarrow \emptyset$ .
- **Vereinigung** mglw. entstandener  $\alpha \rightarrow \beta_1$  und  $\alpha \rightarrow \beta_2$  zu  $\alpha \rightarrow \beta_1 \beta_2$  (Vereinigung).

# Algorithmus zur Berechnung der Kanonischen Überdeckung

## Ein erstes Beispiel...

- $R = \{ A, B, C, D, E, F \}$ ,  
 $FD = \{ ABC \rightarrow DEF, A \rightarrow BC, D \rightarrow EF \}$
- Vorgehen:
  - erst Links- dann Rechtsreduktion
  - jeweils Attribute weglassen und 'stur' die Bedingungen testen

**B Beispiel: Kanonische Überdeckung**

$F = \{ABC \rightarrow DEF, A \rightarrow BC, D \rightarrow EF\}$

**Linksreduktion**

- **ABC** überflüssig? **Nein**, denn **AttrHülle(F, BC)**: {BC}.
- **B** überflüssig? **Ja**, denn **AttrHülle(F, AC)**: {AC}  $\rightarrow$  {ABC}  $\rightarrow$  {ABCDEF}  
 – Erhalte  $F = \{AC \rightarrow DEF, A \rightarrow BC, D \rightarrow EF\}$
- **C** überflüssig? **Ja**, denn **AttrHülle(F, A)**: {A}  $\rightarrow$  {ABC}  $\rightarrow$  {ABCDEF}  
 – Erhalte  $F = \{A \rightarrow DEF, A \rightarrow BC, D \rightarrow EF\}$

**nicht verwechseln:  
Attribut F und Menge FD**

**gleiche  
Menge FD,  
veränderte  
Attribute**

**neu, aber  
äquivalent**

A  $\rightarrow$  BC      ABC  $\rightarrow$  DEF

A  $\rightarrow$  BC      ABC  $\rightarrow$  DEF

Unterlagen Prof. Striegnitz

# Algorithmus zur Berechnung der Kanonischen Überdeckung

## Ein erstes Beispiel...

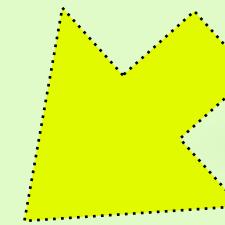
- $R = \{ A, B, C, D, E, F \}$ ,  
 $FD = \{ ABC \rightarrow DEF, A \rightarrow BC, D \rightarrow EF \}$
- $FD^C = \{ A \rightarrow BCD, D \rightarrow EF \}$

### B Beispiel: Kanonische Überdeckung

$$F = \{A \rightarrow DEF, A \rightarrow BC, D \rightarrow EF\}$$

#### Rechtsreduktion

- **$A \rightarrow DEF$** 
  - ▷ **D** überflüssig? **Nein**, denn  
 $\text{AttrHülle}(\{A \rightarrow EF, A \rightarrow BC, D \rightarrow EF\}, A) : \{A\} \rightarrow \{AEF\} \rightarrow \{AEFBC\}$
  - ▷ **E** überflüssig? **Ja**, denn  
 $\text{AttrHülle}(\{A \rightarrow DF, A \rightarrow BC, D \rightarrow EF\}, A) : \{A\} \rightarrow \{ADF\} \rightarrow \{ADFB\} \rightarrow \{ADFBCE\}$   
- Erhalte  $F = \{A \rightarrow DF, A \rightarrow BC, D \rightarrow EF\}$
  - ▷ **F** überflüssig? **Ja**, denn  
 $\text{AttrHülle}(\{A \rightarrow D, A \rightarrow BC, D \rightarrow EF\}, A) : \{A\} \rightarrow \{AD\} \rightarrow \{ADBC\} \rightarrow \{ADBCF\}$   
- Erhalte  $F = \{A \rightarrow D, A \rightarrow BC, D \rightarrow EF\}$



veränderte  
Menge FD,  
gleiche  
Attribute

Die anderen Regeln lassen sich nicht reduzieren ....  
Es ergibt sich:  $F^C = \{A \rightarrow BCD, D \rightarrow EF\}$

## Algorithmus zur Berechnung der Kanonischen Überdeckung

---

### Anmerkungen Teil I

- **Linksreduktion:** Hier sind nur die funktionalen Abhängigkeiten zu überprüfen, die rechts mind. zwei Attribute besitzen, denn eine Abhängigkeit der Form  $\emptyset \rightarrow \beta$  (ein weggelassenes Attribut bei nur einem Attribut) macht keinen Sinn.
- **Rechtsreduktion:** Hier sind *alle* funktionalen Abhängigkeiten, auch  $\alpha \rightarrow b$  mit nur einem Attribut  $b$  rechts, zu überprüfen. Die veränderte Menge für die Attributhülle sieht dann formal so aus:  $FD \setminus \{\alpha \rightarrow b\} \cup \{\alpha \rightarrow \emptyset\} \dots$

Wenn aber in der letzten funktionalen Abhängigkeit nichts aus  $\alpha$  folgt, kann man diese Abhängigkeit auch direkt weglassen. Also insgesamt heisst das, man überprüft direkt ohne diese Abhängigkeit die Menge  $FD \setminus \{\alpha \rightarrow b\}$ .

Wenn dann  $\beta = b$  aus  $\alpha$  folgt und man die funktionale Abhängigkeit  $\alpha \rightarrow b$  direkt weglässt, dann entstehen erst gar keine Abhängigkeiten der Form  $\alpha \rightarrow \emptyset$ .

## Algorithmus zur Berechnung der Kanonischen Überdeckung

---

### Anmerkungen Teil II

- **Rechtsreduktion:** Man beachte, dass beispielsweise die Mengen
  - ♦  $FD = \{ \alpha \rightarrow \beta, A \rightarrow BC \}$  und  $FD' = \{ \alpha \rightarrow \beta, A \rightarrow B, A \rightarrow C \}$offensichtlich äquivalent sind und daher die Rechtsreduktion generell etwas abgeändert werden kann (persönlicher Stil):
  - Anstatt FD zu modifizieren und  $A \rightarrow BC$  erst durch  $A \rightarrow C$  (Test, ob B notwendig) und dann durch  $A \rightarrow B$  (Test C) zu ersetzen, kann man auch zunächst FD zu einer äquivalenten Menge  $FD'$  ändern, in der alle funktionalen Abhängigkeiten mit mehr als einem Attribut auf der rechten Seite aufgespalten werden (Dekomposition).
  - Das führt zu einer größeren Menge  $FD'$ , die es zu untersuchen gilt, aber insgesamt bleibt die Menge der Tests mit der Attributhülle gleich und jeder einzelne Test ist etwas weniger komplex, weil man die untersuchte funktionale Abhängigkeit nicht modifiziert, sondern wegläßt. Im letzten Schritt werden alle wieder vereinigt.

## Algorithmus zur Berechnung der Kanonischen Überdeckung

### Anmerkungen Teil III

- **Beispiel Rechtsreduktion:** Die Alternativen sind

♦  $FD = \{ \alpha \rightarrow \beta, A \rightarrow BC \}$  und  $FD' = \{ \alpha \rightarrow \beta, A \rightarrow B, A \rightarrow C \}$

Klassisch würde man u.a. untersuchen, ob  $\beta \subseteq \text{AttrHülle}(FD \setminus \{\alpha \rightarrow \beta\} \cup \{\alpha \rightarrow \beta - b\}, \alpha)$  gilt, d.h.

- $B \subseteq \text{AttrHülle}(FD \setminus \{A \rightarrow BC\} \cup \{A \rightarrow C\}, A)$  ?
- $C \subseteq \text{AttrHülle}(FD \setminus \{A \rightarrow BC\} \cup \{A \rightarrow B\}, A)$  ?

In der anderen Variante untersucht man u.a.

- $C \subseteq \text{AttrHülle}(FD' \setminus \{A \rightarrow B\}, A)$  ?
- $B \subseteq \text{AttrHülle}(FD' \setminus \{A \rightarrow C\}, A)$  ?

Zunächst (FD') größer, aber  
Modifikation ist einfacher.



#### Q&A

- Ist die kanonische Überdeckung eindeutig?

# Algorithmus zur Berechnung der Kanonischen Überdeckung

---

## Beispiel

- $R = \{ A, B, C, D \}$ ,  
 $FD = \{ A \rightarrow B,$   
 $BD \rightarrow C,$   
 $ABC \rightarrow AB \}$

Links red.

$$\begin{array}{l} a \rightarrow \beta \\ BD \rightarrow C \end{array}$$

$$\text{Att. Hülle } (\overline{FD}, D) = D \quad , \quad C \in \{ D \} \text{ Nein } \times$$

$$\text{Att. Hülle } (\overline{FD}, B) = B \quad , \quad C \in \{ B \} \text{ Nein } \times$$

$$\begin{array}{l} a \rightarrow \beta \\ ABC \rightarrow AB \end{array}$$

$$\text{Att. Hülle } (\overline{FD}, BC) = BC \quad AB \in \{ BC \} \quad \checkmark$$

$$\text{Att. Hülle } (\overline{FD}, AC) = ACB \quad AB \in \{ ACB \} \quad \checkmark$$

also B überfl.

$$\overline{FD} = \{ A \rightarrow B, BD \rightarrow C, AC \rightarrow AB \}$$

$$\text{Att. Hülle } (\overline{FD}, A) = AB \quad \checkmark$$

also C überfl.

$$\overline{FD} = \{ A \rightarrow B, BD \rightarrow C, A \rightarrow AB \}$$

## Algorithmus zur Berechnung der Kanonischen Überdeckung

### Beispiel

Hilfslnd.

$$\begin{array}{l} A \rightarrow B \\ A \rightarrow B \end{array}$$

$$\text{sth. Hülle } (\{BD \rightarrow C, A \rightarrow AB\}, A) = AB \\ B \in \{AB\}$$

also  $B$  überfl.

$$\overline{FD} = \{ \underset{\substack{\uparrow \\ \text{weglassen}}}{A \rightarrow \emptyset}, BD \rightarrow C, A \cancel{\rightarrow} AB \}$$

$$\begin{array}{l} A \rightarrow B \\ BD \rightarrow C \end{array}$$

$$\text{sth. Hülle } (\{ A \rightarrow AB \}, BD) = BD, C \in \{BD\}$$

- $R = \{A, B, C, D\}$ ,  
 $\boxed{FD^C = \{A \rightarrow B, BD \rightarrow C\}}$

$$\begin{array}{l} A \rightarrow B \\ A \rightarrow AB \end{array}$$

$$\text{sth. Hülle } (\{BD \rightarrow C, A \rightarrow B\}, A) = AB, AB \in \{AB\} \\ \text{also } A \text{ überfl.}$$

$$\begin{array}{l} A \rightarrow B \\ A \rightarrow B \end{array}$$

$$\overline{FD} = \{ BD \rightarrow C, A \rightarrow B \}$$

$$\text{sth. Hülle } (\{ BD \rightarrow C \}, A) = A, B \in \{AB\}$$

## Kanonische Überdeckung

---

### Roter Faden - Wozu das alles?

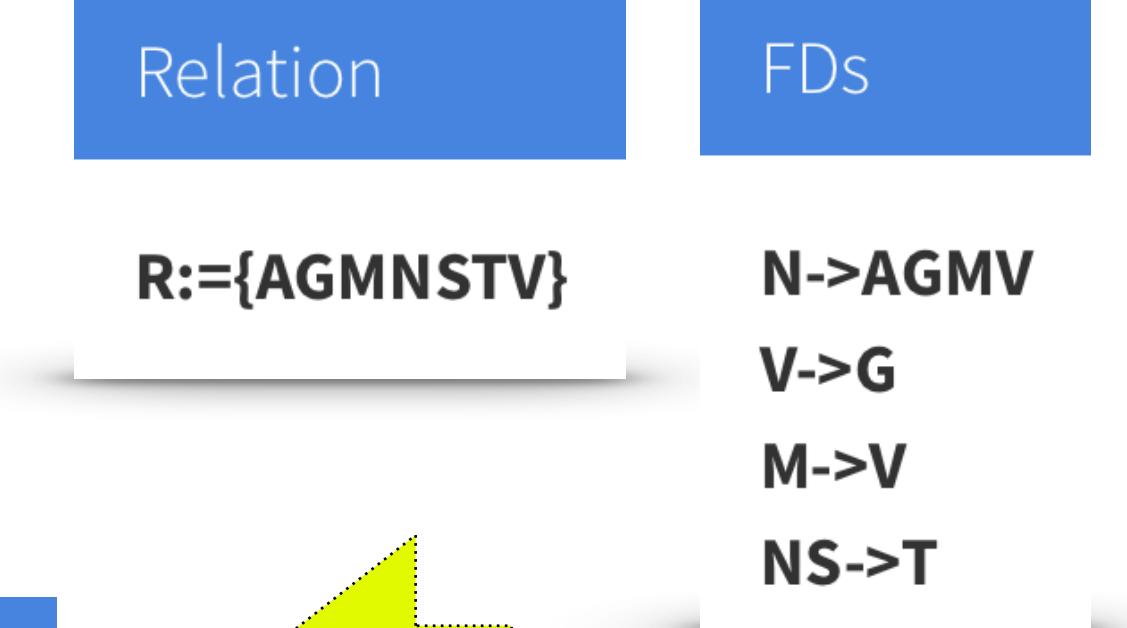
- Die Kanonische Überdeckung ist der Hauptbestandteil des *Synthesealgorithmus*, der eine Relation in die sogenannte *dritte Normalform* (Gütekriterium) überführt.
- Relationen, die einer gewissen Normalform entsprechen, vermeiden bestimmte Anomalien.
- Anomalien bezeichnen Zustände in der Datenbank, bei denen Daten falsch bzw. inkonsistent sind, z.B. ermöglicht durch Redundanzen (Beispiel 'Heise' vs. 'Heihse').

### Was folgt:

- *Teaser*: Artikeldatenbank inkl. *Synthesealgorithmus* und Normalformen
- *Unit 0x09*: Normalformen, Anomalien, Synthese- und Dekompositionsalgorithmus

## Beispiel Kanonische Überdeckung / Synthesealgorithmus

Nr (N)	Verlag (V)	Magazin (M)	Ausgabe (A)	Gründung (G)	Seite (S)	Thema (T)
aa11	Heise	c't	11/18	1949	15	C++
aa11	Heise	c't	11/18	1949	24	C#
bb22	Heise	c't	12/18	1949	12	Pvthon



- 1 Linksreduktion
- 2 Rechtsreduktion
- 3  $\alpha \rightarrow \emptyset$  entfernen
- 4 FDs zusammenfassen

$N \rightarrow AGMV$	$N \rightarrow AM$	$N \rightarrow AM$
$V \rightarrow G$	$V \rightarrow G$	$V \rightarrow G$
$M \rightarrow V$	$M \rightarrow V$	$M \rightarrow V$
$NS \rightarrow T$	$NS \rightarrow T$	$NS \rightarrow T$

Verlag (V)	Gründung (G)
Heise	1949
Computec	1989
Wedia	2007
...	

$\{[N, A, M]\}, \{[M, V]\}, \{[N, S, T]\}$

Kanonische Überdeckung / Hauptteil Synthesealgorithmus

## Vorschau

---

### Synthesealgorithmus

- Gegeben sind ein Schema R und eine Menge FD funktionaler Abhängigkeiten.
1. Bestimme Kanonische Überdeckung  $FD^C$  zu FD, d.h.
    - Links- und Rechtsreduktion, Entfernung, Zusammenfassung
  2. Für alle  $\alpha \rightarrow \beta \in FD^C$  wird Schema  $R_k = \alpha \cup \beta$ , bzw. Relation  $\{[\alpha, \beta]\}$ , erzeugt. Die funktionalen Abhängigkeiten, die Attribute aus  $R_k$  enthalten, werden  $R_k$  zugeordnet.
  3. Ein Schlüsselkandidat muss in den  $R_k$  aus Schritt 2 enthalten sein. Ansonsten wird eine Relation, die die Attribute eines Schlüsselkandidat enthält, zusätzlich erzeugt.
  4. Einzelne  $R_k$  aus Schritt 2 können andere  $R_m$  enthalten. Diese Schemata  $R_m$  braucht man dann nicht mehr, denn alle Informationen sind in  $R_k$ .



**Die funktionalen Abhängigkeiten, geeignet reduziert, bestimmen die Relationen im DBMS.**

## Vorschau

---

### Normalformen

- Die erste Normalform ist gegeben, wenn es keine zusammengesetzten oder mehrwertigen Attribute gibt.
- Die zweite Normalform ist erfüllt, wenn die erste erfüllt ist und alle nicht-prim Attribute voll funktional abhängig von jedem Schlüsselkandidaten sind.
- Die dritte Normalform liegt vor, wenn die zweite erfüllt ist und jedes nicht-prim Attribut direkt und nicht transitiv von einem Schlüsselkandidaten abhängt.

### Anmerkungen

- Beispiele und Anomalien, die verhindert werden, folgen!
- Für die zweite und dritte Normalform benötigen wir die Schlüsselkandidaten.
- Bedingungen beziehen sich auf nicht-prim Attribute. Redundanzen bei prim Attributen sind weiter möglich...

# UNIT 0x09

## NORMALFORMEN III

## SYNTHESEALGORITHMUS

## DEKOMPOSITIONSALGORITHMUS

## Motivation/Erinnerung

---

### Aufgabe

- Entwicklung einer 'guten' Datenbank [...] u.a. minimal, redundanzfrei [...]

### Vorgehen

- Konzeptueller Entwurf [...] ER-Modells ✓
- Überführung in eine 'gute' Datenbank [...] dazu benötigen wir:
  - Implementationsentwurf [...] → Relationales Modell ✓
  - Mathematisches Modell [...] → Relationale Algebra ✓
  - Gütemaß → Normalformen (Anomalien, Verlustlosigkeit, Abhängigkeitserhaltung)
    - ♦ Kanonische Überdeckung ✓, funktionale Abhängigkeiten ✓, Attributhülle ✓,
    - ♦ Synthesealgorithmus (✓),
    - ♦ Dekompositionsalgorithmus

## Anomalien

---

### CRUD

- Unter CRUD versteht man die fundamentalen Datenbankoperationen:
  - **Create:** Datensatz anlegen,
  - **Read/Retrieve:** Datensatz lesen,
  - **Update:** Datensatz aktualisieren, und
  - **Delete/Destroy:** Datensatz löschen.
- Da der Lesezugriff (Read) die Daten nicht ändert, betrachtet man die folgenden Anomalien primär im Zusammenhang mit den anderen, die Daten verändernden, Operationen Create (Einfügung), Update und Delete (Lösung).
- Anmerkung: Die CRUD Operationen sind auch die, die in Persistenz-Frameworks häufig die konkreten Operationen auf dem DBMS abstrahieren.

## Anomalien

---

### Definition

- Es gibt:
  - **Update-Anomalien**
  - **Einfüge-Anomalien**
  - **Löschen-Anomalien.**
- Ausgangspunkt der Beispiele ist immer eine Relation, die diese Anomalien auch zuläßt und in diesem Sinne keinen "guten Datenbankentwurf" darstellt. Ein Datenbankentwurf, der bestimmte Normalformen erfüllt, vermeidet diese.

# Anomalien

---

## Update-Anomalie

- Angenommen, Vorlesungen und Professoren sind in einer Tabelle realisiert.
- Durch das Update einer falschen Raumnr. werden die Daten inkonsistent, da Sokrates kein eindeutiger Raum mehr zugeordnet ist.
- Generell tritt diese Anomalie auf, wenn redundante Daten in einem Tupel nur teilweise (falsch) aktualisiert werden.

ProfVorl						
PersNr	Name	Rang	Raum	VorlNr	Titel	SWS
2125	Sokrates	C4	226	5041	Ethik	4
2125	Sokrates	C4	226	5049	Mäeutik	2
2125	Sokrates	C4	226	4052	Logik	4
...	...	...	...	...	...	...
2132	Popper	C3	52	5259	Der Wiener Kreis	2
2137	Kant	C4	7	4630	Die 3 Kritiken	4

- **Update-Anomalie**

```
update ProfVorl
  set Raum=111
where Titel="Ethik"
```

Unterlagen Prof. Striegnitz

# Anomalien

---

## Einfüge-Anomalie

- Angenommen, Vorlesungen und Professoren sind in einer Tabelle realisiert *und* VorlNr darf nicht NULL sein!
- Wenn Fr. Curie noch keine Vorlesung hält, ist Einfügen nicht möglich.
- Generell sind bei dieser Anomalie Daten so verbunden, dass sie nicht ohne andere (not NULL) eingegeben werden können.

ProfVorl						
PersNr	Name	Rang	Raum	VorlNr	Titel	SWS
2125	Sokrates	C4	226	5041	Ethik	4
2125	Sokrates	C4	226	5049	Mäeutik	2
2125	Sokrates	C4	226	4052	Logik	4
...	...	...	...	...	...	...
2132	Popper	C3	52	5259	Der Wiener Kreis	2
2137	Kant	C4	7	4630	Die 3 Kritiken	4

- **Einfüge-Anomalie**

```
insert into ProfVorl
values (2239,Curie,C4,112)
```

Unterlagen Prof. Striegnitz

## Anomalien

---

### Löschen-Anomalie

- Angenommen, Vorlesungen und Professoren sind in einer Tabelle realisiert.
- Beim Löschen der Vorlesung "Der Wiener Kreis" wird auch Hr. Popper gelöscht, wenn das seine einzige Vorlesung ist.
- Generell müssen bei dieser Anomalie alle Daten eines Tupels gelöscht werden, obwohl eine Teilmenge gereicht hätte.

ProfVorl						
PersNr	Name	Rang	Raum	VorlNr	Titel	SWS
2125	Sokrates	C4	226	5041	Ethik	4
2125	Sokrates	C4	226	5049	Mäeutik	2
2125	Sokrates	C4	226	4052	Logik	4
...	...	...	...	...	...	...
2132	Popper	C3	52	5259	Der Wiener Kreis	2
2137	Kant	C4	7	4630	Die 3 Kritiken	4

- **Löschen-Anomalie**

```
delete ProfVorl
where Titel="Der Wiener Kreis"
```

Unterlagen Prof. Striegnitz

# Verlustlosigkeit und Abhängigkeitserhaltung

## Lösungsansatz

- Offensichtlich wurden in der Relation zwei Sachverhalte modelliert.
- Trennt man die Relation auf (Dekomposition), so muss natürlich später die Zusammenführung möglich sein (Rekomposition). Dazu benötigt man einen gemeinsamen, ggf. neuen, Bereich, der in beiden Relationen vorkommt.

ProfVorl						
PersNr	Name	Rang	Raum	VorlNr	Titel	SWS
2125	Sokrates	C4	226	5041	Ethik	4
2125	Sokrates	C4	226	5049	Mäeutik	2
2125	Sokrates	C4	226	4052	Logik	4
...	...	...	...	...	...	...
2132	Popper	C3	52	5259	Der Wiener Kreis	2
2137	Kant	C4	7	4630	Die 3 Kritiken	4

Professoren

PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Vorlesungen

VorlNr	Titel	SWS	gelesenVon
4052	Logik	4	2125
4630	Die 3 Kritiken	4	2137
5001	Grundzuege	4	2125
5022	Glaube und Wissen	2	2134
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Maeeutik	2	2125

## Verlustlosigkeit und Abhängigkeitserhaltung

### Definition

Gegeben sei eine Relation R mit einer Menge funktionaler Abhängigkeiten. Diese Relation soll in neue Relationen  $R_i$  aufgeteilt werden, z.B. um Anomalien zu vermeiden.

- **Verlustlosigkeit/Verbundtreue:** Die in der ursprünglichen Relation R enthaltenen Informationen müssen aus den neuen Relationen  $R_1, \dots, R_n$  mittels natürlichen Verbunds (Natural Join) rekonstruierbar sein.
- **Abhängigkeitserhaltung:** Die ursprünglich geltenden funktionalen Abhängigkeiten müssen auch auf der Zerlegung, also den neuen Relationen, gelten.

#### Q&A

- Fallen Ihnen Aufteilungen ein, wo man Informationen oder Abhängigkeiten verliert?

Relation ist hier im Sinne eines *Schemas* gemeint, also nicht auf eine konkrete Ausprägung bezogen.



## Verlustlosigkeit und Abhängigkeitserhaltung

---

### Hinreichendes Kriterium für Verlustlosigkeit

- Eine Zerlegung einer Relation R in zwei Relationen  $R_1$  und  $R_2$  ist verlustlos, wenn man mind. eine der beiden aus dem gemeinsamen Bereich (Überlappung der Attribute) wieder ableiten kann. D.h. es gilt:
  - ♦  $R_1 \cap R_2 \rightarrow R_1$ , oder
  - ♦  $R_1 \cap R_2 \rightarrow R_2$ .

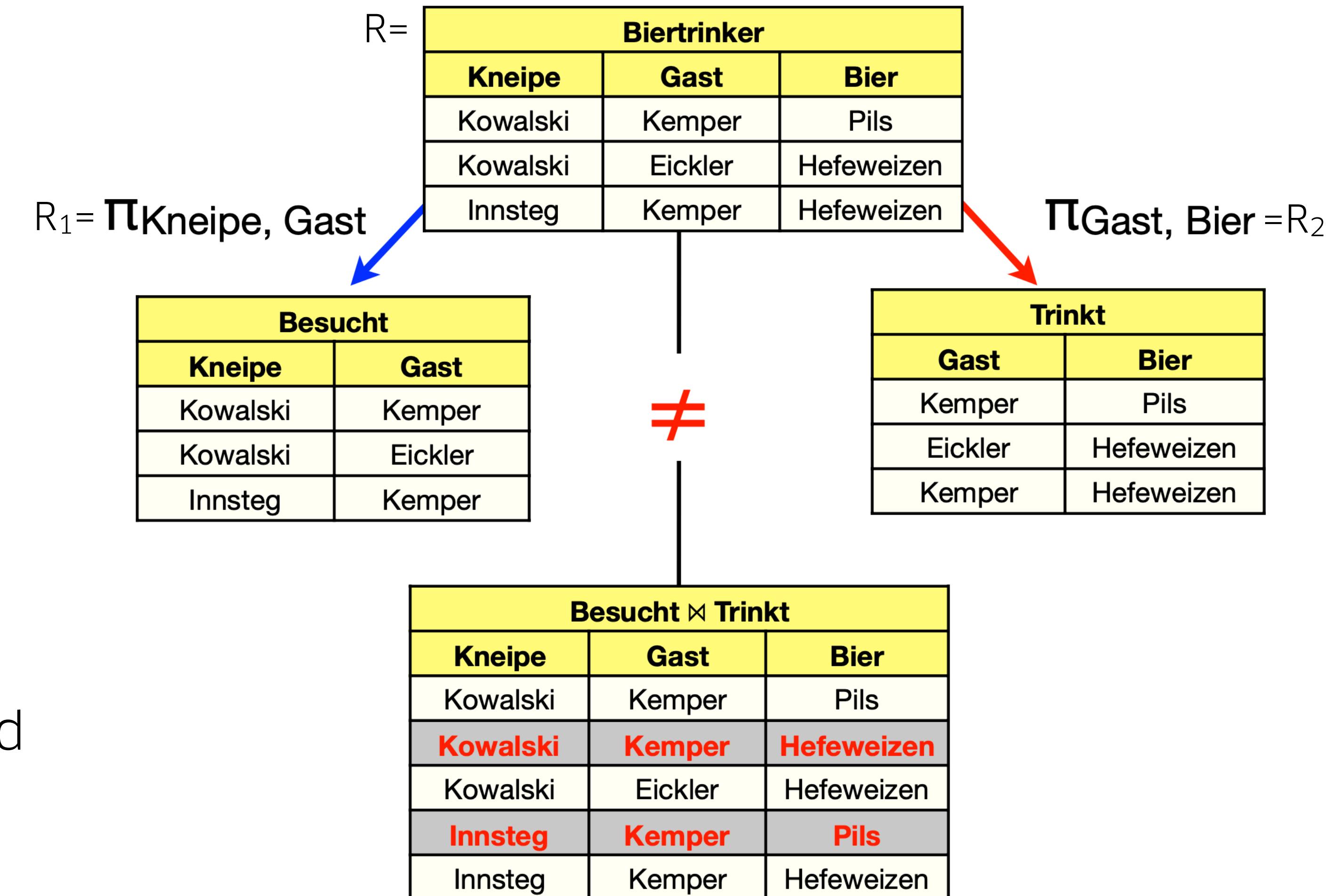


So findet man es gerne in der Literatur. Gemeint sind hier wieder die Schemata und nicht eine konkrete Ausprägung der Daten, denn die funktionalen Abhängigkeiten gelten für alle Ausprägungen. Vgl. Diskussion um funktionale Abhängigkeiten.

# Verlustlosigkeit und Abhängigkeitserhaltung

## Beispiel Verletzung der Verlustlosigkeit

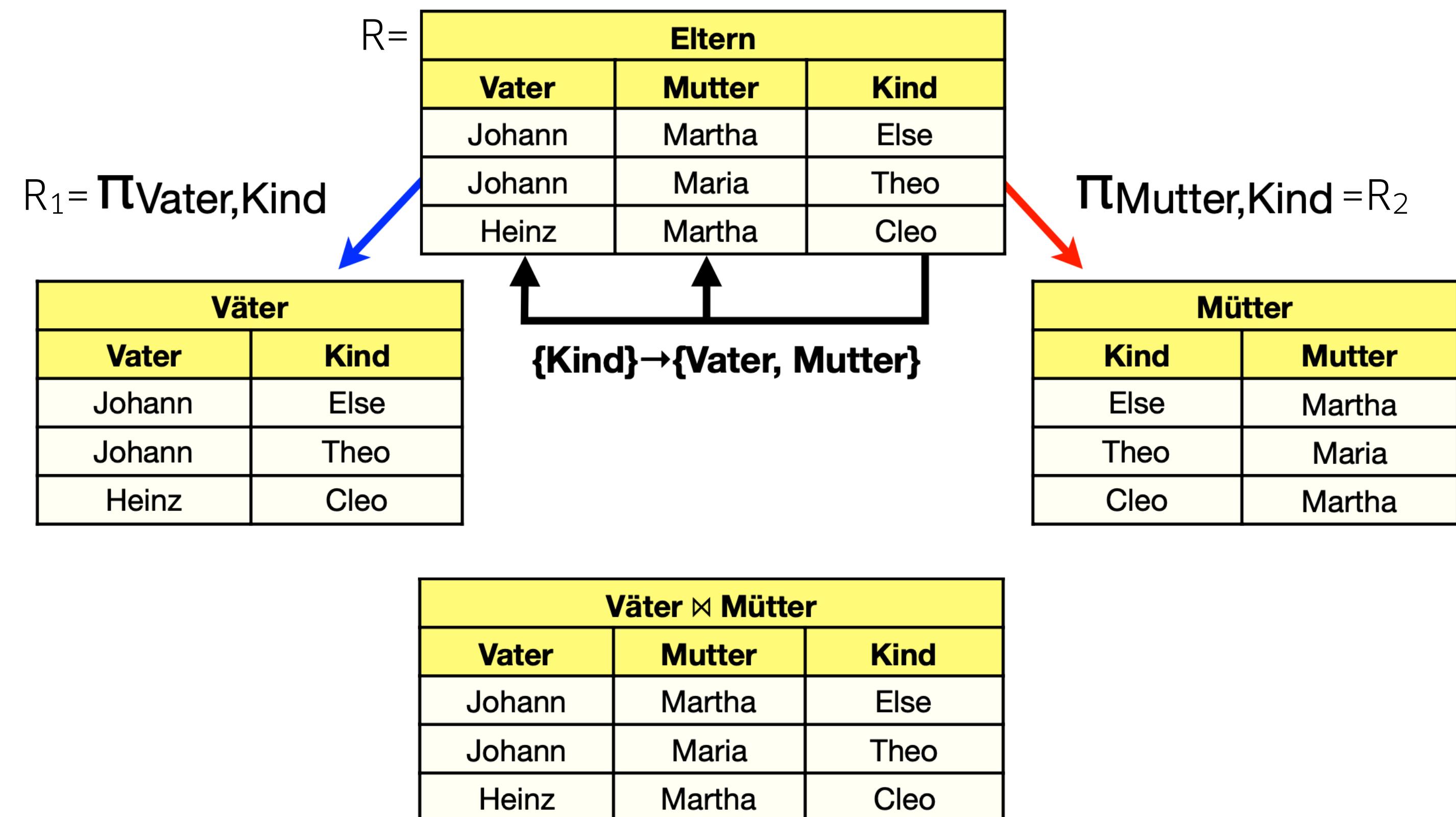
- Die Aufteilung in  $R_1$  und  $R_2$  ist *nicht verlustlos*, denn die Rekomposition durch den natürlichen Verbund enthält hier *mehr* Daten als in der ursprünglichen Relation!
- Insbesondere gilt:
  - $R_1 \cap R_2 = \text{Gast} \not\rightarrow R_1$ , und
  - $R_1 \cap R_2 = \text{Gast} \not\rightarrow R_2$ .



# Verlustlosigkeit und Abhängigkeitserhaltung

## Beispiel Verlustlosigkeit

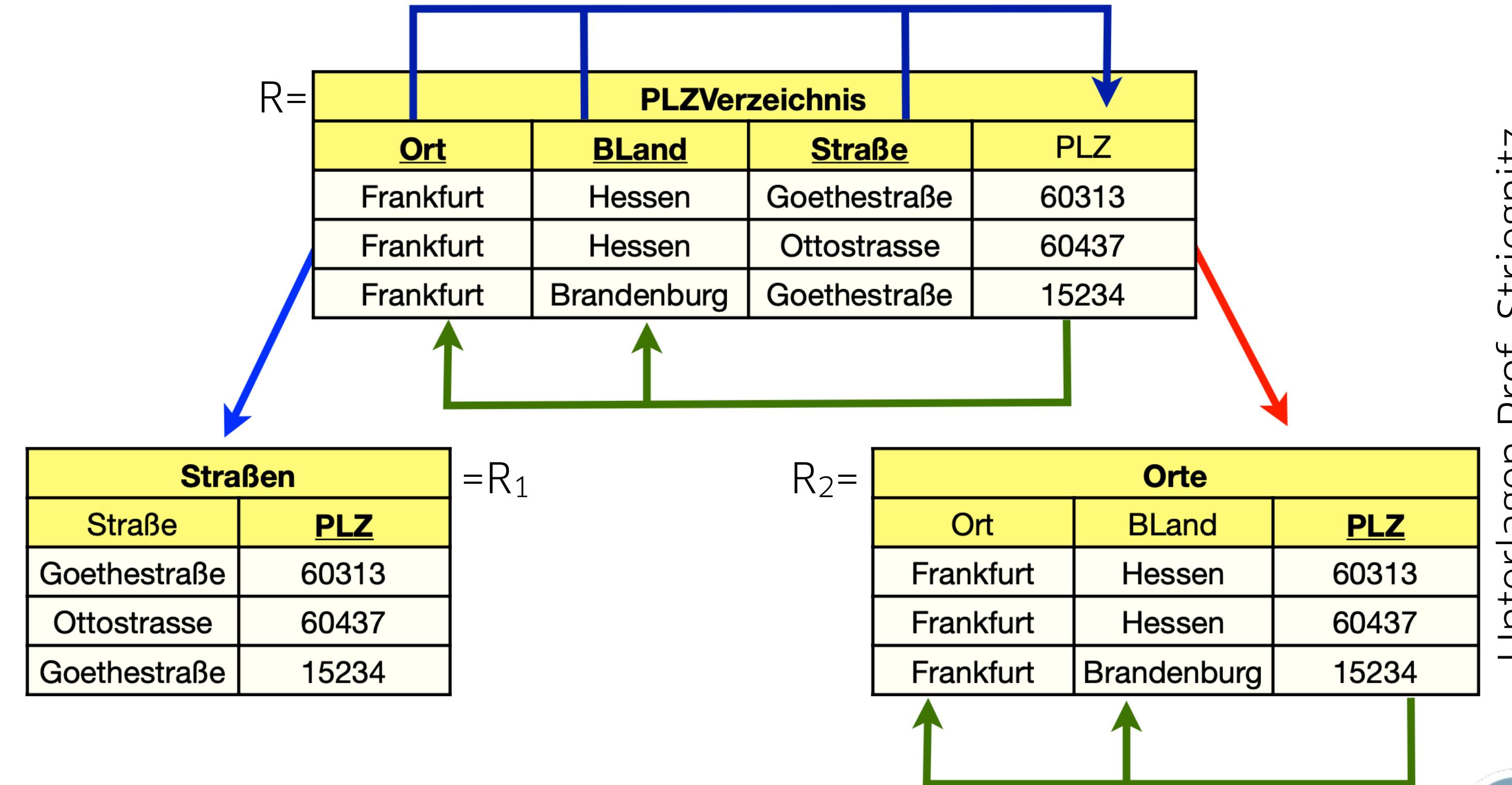
- Die Aufteilung in  $R_1$  und  $R_2$  ist *verlustlos*, denn die Rekomposition durch den natürlichen Verbund ergibt hier wieder die ursprünglichen Relation!
- Insbesondere gilt:
  - $R_1 \cap R_2 = \text{Kind} \rightarrow R_1$ , und
  - $R_1 \cap R_2 = \text{Kind} \rightarrow R_2$ .



# Verlustlosigkeit und Abhängigkeitserhaltung

## Beispiel Verlust der Abhängigkeitserhaltung

- Orte sind durch Namen und Bundesland eindeutig identif.
- Innerhalb einer Straße ändert sich die Postleitzahl nicht.
- Postleitzahlengebiete gehen nicht über Ortsgrenzen.
- Funktionale Abhängigkeiten:
  - PLZ → Ort,BLand
  - Straße, Ort, BLand → PLZ



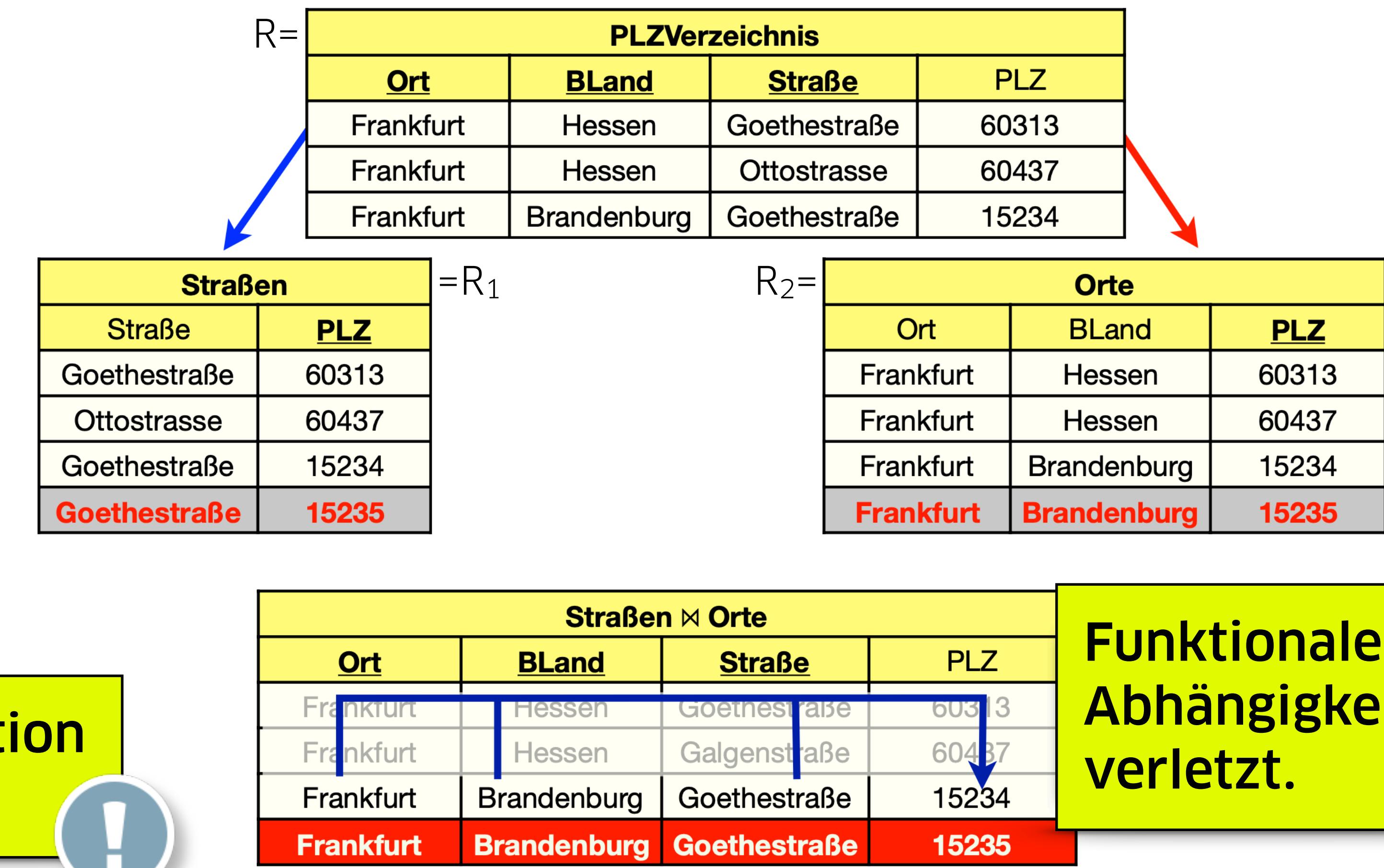
Die Zerlegung ist verlustfrei, aber die funktionale Abhängigkeit  $\text{Straße}, \text{Ort}, \text{BLand} \rightarrow \text{PLZ}$  geht verloren.



# Verlustlosigkeit und Abhängigkeitserhaltung

## Beispiel Verlust der Abhängigkeitserhaltung

- Durch den Verlust der funktionalen Abhängigkeit sind diese neuen Einfügungen in  $R_1$  und  $R_2$  möglich, führen aber zur Verletzung bei der Rekombination.



Offenbar kann man eine Relation nicht beliebig aufspalten... !



## Normalformen

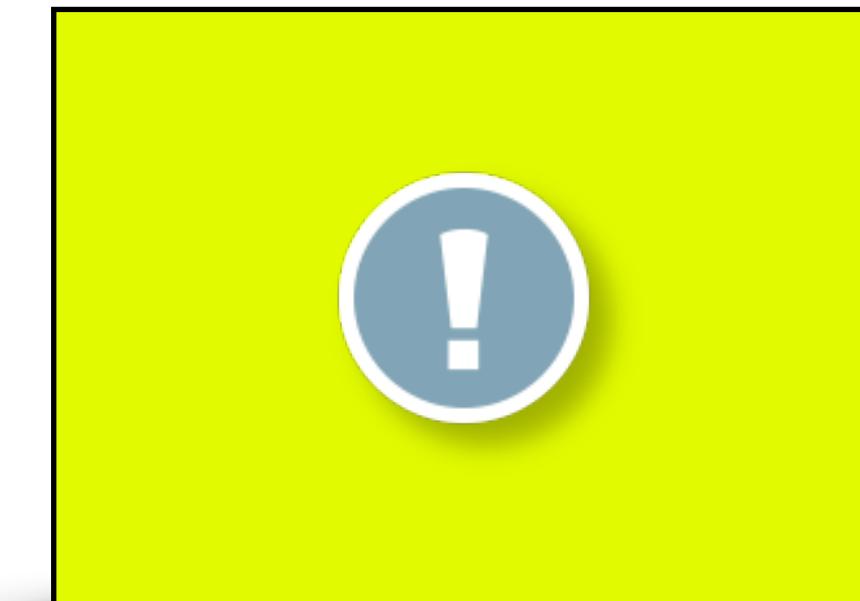
---

### Roter Faden

- Zur Motivation der Normalformen betrachtet man *Anomalien*, die durch die jeweilige Normalformen verhindert werden.
- Verändert man ein Schema, um einer Normalform zu genügen, muss man auf *Verlustlosigkeit und Abhängigkeitserhaltung* achten. Synthese- und Dekompositions-algorihmus überführen jeweils eine Schema in eine bestimmte Normalform.

### Normalformen

- Erste Normalform (1NF, manchmal auch NF1),
- Zweite Normalform (2NF/2NF),
- Dritte Normalform (3NF/NF3),
- Boyce-Codd-Normalform (BCNF),
- Vierte und Fünfte Normalform (hier nicht relevant).



## Erste Normalform

---

### Definition

- Ein Schema in erster Normalform (1NF, auch NF1) besitzt nur atomare/elementare Attribute, d.h. kein Attribut ist zusammengesetzt oder mehrwertig.
- Funktionale Abhängigkeiten spielen keine Rolle.

### Anmerkungen

- Die Atomarität hatten wir schon beim Übergang vom ER-Diagramm in das Implementationsmodell gefordert.
- Man findet auch die Bezeichnung 'Nullte Normalform' für ein Schema nicht in 1NF.
- Die erste Normalform *ist immer gegeben*, wenn das Schema in der Form anonymer Attribute (A,B,C,...) vorliegt. Nur bei Attributen mit Bedeutung können diese zusammengesetzt oder mehrwertig sein.

## Erste Normalform

---

### Beispiel zu Erster Normalform

Nr (EAN,ISBN)	Zeitschrift	Verlags- gründung	Themen
...aa11	Heise - c't - 11/18	1949	S.15 C++, S.24 C#
...bb22	Heise - c't - 12/18	1949	S.12 Python, S.29 C#
...cc33	Heise - ix - 08/18	1949	S.9 RAID, S.23 gpio
...dd44	Computec - buffered - 08/18	1989	S.11 WoW
...ee55	Webedia - GameStar - 08/18	2007	S.13 LoL, S.20 WoW

Nicht in erster Normalform

Nr (N)	Verlag (V)	Magazin (M)	Ausgabe (A)	Gründung (G)	Seite (S)	Thema (T)
aa11	Heise	c't	11/18	1949	15	C++
aa11	Heise	c't	11/18	1949	24	C#
bb22	Heise	c't	12/18	1949	12	Python
bb22	Heise	c't	12/18	1949	29	C#
cc33	Heise	iX	08/18	1949	9	RAID
cc33	Heise	iX	08/18	1949	23	gpio
dd44	Computec	buffered	08/18	1989	11	WoW

Erfüllt erste Normalform

### Überführung in erste Normalform

- Um 1NF zu erreichen, müssen Sachverhalte in mehrere Attribute getrennt und Mehrwertigkeiten, typischerweise in eine 1:N-Relation, aufgespalten werden.

## Zweite Normalform

---

### Motivation

- Das nebenstehende Schema besitzt den Schlüssel
  - MatrNr,VorlNr. → ident aber es gilt
  - MatrNr → Name, Semester d.h. Attribute sind abhängig von *einem Teil* des Schlüssels.
- Hier sind Redundanzen sichtbar und Update-Anomalien möglich, etwa die Änderung eines Names in nur einem Tupel.

<b>StudentenBelegung</b>			
<b>MatrNr</b>	<b>VorlNr</b>	<b>Name</b>	<b>Semester</b>
26120	5001	Fichte	10
27550	5001	Schopenhauer	6
27550	4052	Schopenhauer	6
28106	5041	Carnap	3
28106	5216	Carnap	3
28106	5259	Carnap	3
...	...	...	...

## Zweite Normalform

---

### Definition

- Ein Schema R ist in zweiter Normalform (2NF, auch NF2), wenn es in 1NF vorliegt und jedes Attribut entweder
  - prim ist, oder
  - voll funktional abhängig von jedem Schlüsselkandidaten ist.

### Anmerkungen

- Für den Test, ob 2NF vorliegt, benötigt man *alle* Schlüsselkandidaten.
- Wenn jedes Attribut in irgendeinem Schlüsselkandidaten vorkommt, ist 2NF erfüllt, da es nur prim Attribute gibt.
- 2NF kann nur verletzt sein, wenn ein Schlüsselkandidat zusammengesetzt ist.

## Zweite Normalform

### Überführung in zweite Normalform

- Die Relation kann verlustfrei und abhängigkeitsbewahrend in zwei Relationen aufgeteilt werden.

Man sieht, dass eine Relation hören sowohl aus einer Überführung in 2NF als auch aus einer guten Modellierung folgt.

StudentenBelegung			
<u>MatrNr</u>	<u>VorlNr</u>	Name	Semester
26120	5001	Fichte	10
27550	5001	Schopenhauer	6
27550	4052	Schopenhauer	6
28106	5041	Carnap	3
28106	5052	Carnap	3
28106	5216	Carnap	3
28106	5259	Carnap	3
...	...	...	...

Nicht in zweiter Normalform

hören	
<u>MatrNr</u>	<u>VorlNr</u>
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
...	...

Studenten		
<u>MatrNr</u>	Name	Semester
26120	Fichte	10
27550	Schopenhauer	6
28106	Carnap	3
...	...	...

Erfüllt zweite Normalform



Wieviele Attribute sind es?

## Zweite Normalform

### Beispiele

- Welche Normalform liegt vor?
  - Schlüsselkandidaten
  - Bedingungen NF
- Normalisieren...
   
(später)

Bsp.1

$$\mathcal{U} = A B C D$$

$$\mathcal{F}D = \{ A \rightarrow BC, B \rightarrow A \}$$

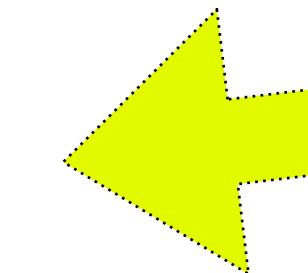
Von welcher NF?

$\Rightarrow$  Schlüsselkand.: D fällt auf rechter Seite (nur in Schl.)

$$\Rightarrow AD, BD$$

$\Rightarrow$  Nicht-Prin: C

$\Rightarrow$  vgl.  $A \rightarrow C$  und A Teil einer Schl. kand. Nicht NF2 oder ?NF



Nur diese für 2NF...

Bsp.2

$$\mathcal{U} = A B C D, \mathcal{F}D = \{ A \rightarrow B, B \rightarrow A, AD \rightarrow C \}$$

$\Rightarrow$  Schl. kand.: AD, BD

$\Rightarrow$  Nicht-Prin: C aber vgl.  $AD \rightarrow C$  liegt 2NF vor



## Dritte Normalform

### Motivation

- Das nebenstehende Schema besitzt die funktionalen Abhängigkeiten
  - $\text{Name} \rightarrow \text{Plz}, \text{Ort}, \text{Strasse}$
  - $\text{Plz} \rightarrow \text{Ort}$
 mit Name als Schlüssel.
- Das Schema ist in 2NF, da der Schlüssel nur aus einem Attribut besteht.
- Hier sind ebenfalls Redundanzen sichtbar und Update-Anomalien möglich, etwa die Änderung eines Ortes in nur einem Tupel (vgl. Motivation Zweite Normalform). Das Problem sind funktionale Abhängigkeiten von Attributen, die nicht prim sind (Ort) und nur transitiv vom Schlüssel abhängen (über Plz).

$\mathcal{R}$			
Name	Plz	Ort	Strasse
Kurt Hanf	52080	Aachen	Von-Coels-Str.
Heidi Panne	52080	Aachen	Karlsstr.

**Die 2NF verhindert offenbar nicht alle Redundanzen.**



## Dritte Normalform

---

### Definition

- Ein Schema R ist in dritter Normalform (3NF, auch NF3), wenn es in 2NF vorliegt und jedes nicht-prim Attribut direkt, also nicht transitiv, von einem Schlüsselkandidaten abhängt.

### Anmerkungen

- Gemeint ist, dass bei einer vorliegenden transitiven Abhängigkeit b von  $\beta$ , also  $\beta \rightarrow a \rightarrow b$ , wobei  $\beta$  ein Schlüsselkandidat ist, hier a kein Schlüsselkandidat ist.

### Überführung in dritte Normalform

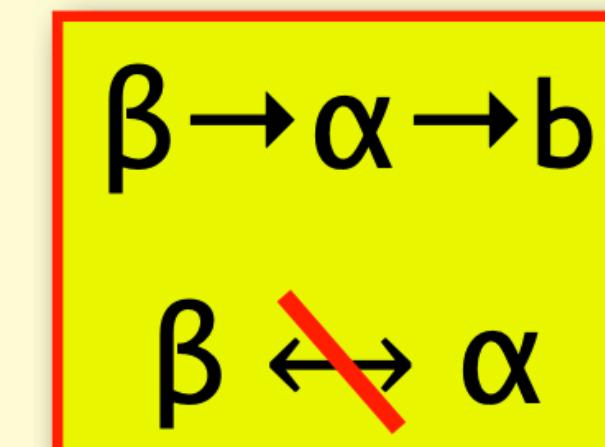
- Der Synthesealgorithmus überführt R verlustlos und abhängigkeits-erhaltend in 3NF.

## Dritte Normalform

## Definition Transitive / direkte Abhangigkeit

Seien  $\mathcal{R} = \{b_1, \dots, b_n\}$  ein vereinfachtes Schema,  $F$  eine Menge von funktionalen Abhangigkeiten zu  $\mathcal{R}$ ,  $\beta \subseteq \mathcal{R}$  und  $b \in \mathcal{R}$ .  $b$  ist **transitiv abhangig** von  $\beta$ , falls gilt

- $\beta \rightarrow b \in F^+ \wedge b \notin \beta$
- $\exists \alpha \subseteq \mathcal{R} :$ 
  - $\alpha \rightarrow b \in F^+ \wedge b \notin \alpha \wedge$
  - $\beta \rightarrow \alpha \in F^+ \wedge$
  - $\alpha \rightarrow \beta \notin F^+$



Gilt  $\beta \rightarrow b \in F^+$  und  $b$  ist nicht transitiv von  $\beta$  abhangig, so heit  $b$  **direkt abhangig** von  $\beta$ .

## Dritte Normalform

---

### Alternative Definition

- Ein Schema R mit einer Menge funktionaler Abhängigkeiten FD ist in dritter Normalform (3NF, auch NF3), wenn für jede funktionale Abhängigkeit  $\alpha \rightarrow \beta \in FD^+$  mindestens eine der folgenden Bedingungen gilt:
  - $\alpha \rightarrow \beta$  ist trivial, also  $\beta \subseteq \alpha$ ,
  - $\beta$  enthält nur prim Attribute,
  - $\alpha$  ist Superschlüssel.

Auch wenn es technischer aussieht,  
beim Test auf 3NF testet man einfach  
alle Abhängigkeiten, ob eine der drei  
Bedingungen zutrifft.

### Anmerkungen

- Für den Test, ob 3NF vorliegt, benötigt man *alle* Schlüsselkandidaten.
- Die zweite Normalform ist eingeschlossen.
- Diese Definition ist auch für die BCNF-Normalform wichtig.



## Dritte Normalform

**Beispiele**

- Welche Normalform liegt vor?
  - Schlüsselkandidaten
  - Bedingungen NF
- Normalisieren...
   
(später)

$\mu = ABCD$        $F^1 = \{ A \rightarrow BC, C \rightarrow D \}$ , welche NF?

$\Rightarrow$  Schlüsselkand. A  $\checkmark$  Nicht-Prim B,C,D

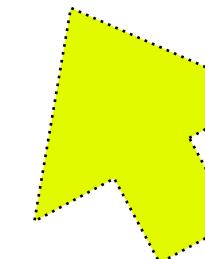
$\Rightarrow$  Trans. Abh. ? ja, sieht man:  $A \rightarrow C, C \rightarrow D$   
**Weg 1** also Nicht 3NF

--- alternativ / immer ---

$A \rightarrow BC$ :

**Weg 2** Bed. 1:  $BC \notin A$   $\checkmark$   
                 Bed. 2: B,C nur Prim?  $\checkmark$   
                 Bed. 3: A Superfl.  $\checkmark$

**Weg 1: nur nicht-prim**



(d.h.  $A \rightarrow BC$  reicht Bed. nicht)

$C \rightarrow D$ :



Bed. 1:  $\checkmark$   
                 Bed. 2: D Prim?  $\checkmark$   
                 Bed. 3: C Superfl?  $\checkmark$

**Weg 2 geht immer!**

also keine Bed. nifft m  $\Rightarrow$  Nicht 3NF



# Synthesealgorithmus

## Definition

- 1. Bestimme die kanonische Überdeckung  $F^C$  zu  $F$ . Wiederholung:**
  - Linksreduktion
  - Rechtsreduktion
  - Entfernung von FDs der Form  $\alpha$
  - Zusammenfassung gleicher linker Seiten
- 2. Für jede funktionale Abhängigkeit  $\alpha \rightarrow \beta \in F^C$ :**
  - Kreiere ein Relationenschema  $\mathcal{R}_\alpha := \alpha \cup \beta$
  - Ordne  $\mathcal{R}_\alpha$  die FDs  $F_\alpha := \{\alpha' \rightarrow \beta' \in F^C \mid \alpha' \cup \beta' \in \mathcal{R}_\alpha\}$  zu.
- 3. Falls eines der in Schritt 2. erzeugten Schemata einen Schlüsselkandidaten von  $\mathcal{R}$  bzgl.  $F^C$  enthält, sind wir fertig. Sonst wähle einen Schlüsselkandidaten  $\kappa \in \mathcal{R}$  aus und definiere folgendes Schema:**
  - $\mathcal{R}_\kappa = \kappa$
  - $F_\kappa = \emptyset$
- 4. Eliminiere diejenigen Schemata  $\mathcal{R}_\alpha$ , die in einem anderen Relationenschema  $\mathcal{R}_\alpha'$  enthalten sind, d.h.:**
  - $\mathcal{R}_\alpha \subseteq \mathcal{R}_\alpha'$

## Synthesealgorithmus

---

### Anmerkungen

- Grundlage ist die kanonische Überdeckung  $FD^C$  (1), sie ergibt die Relationen (2).
- Ein Schlüssel muss enthalten sein (3), vgl. dazu auch das kommende Beispiel.
- Relationen, deren Attribute komplett in anderen Relationen enthalten sind, sind überflüssig (4).



**Der Synthesealgorithmus überführt  
das Schema in die dritte Normalform!**

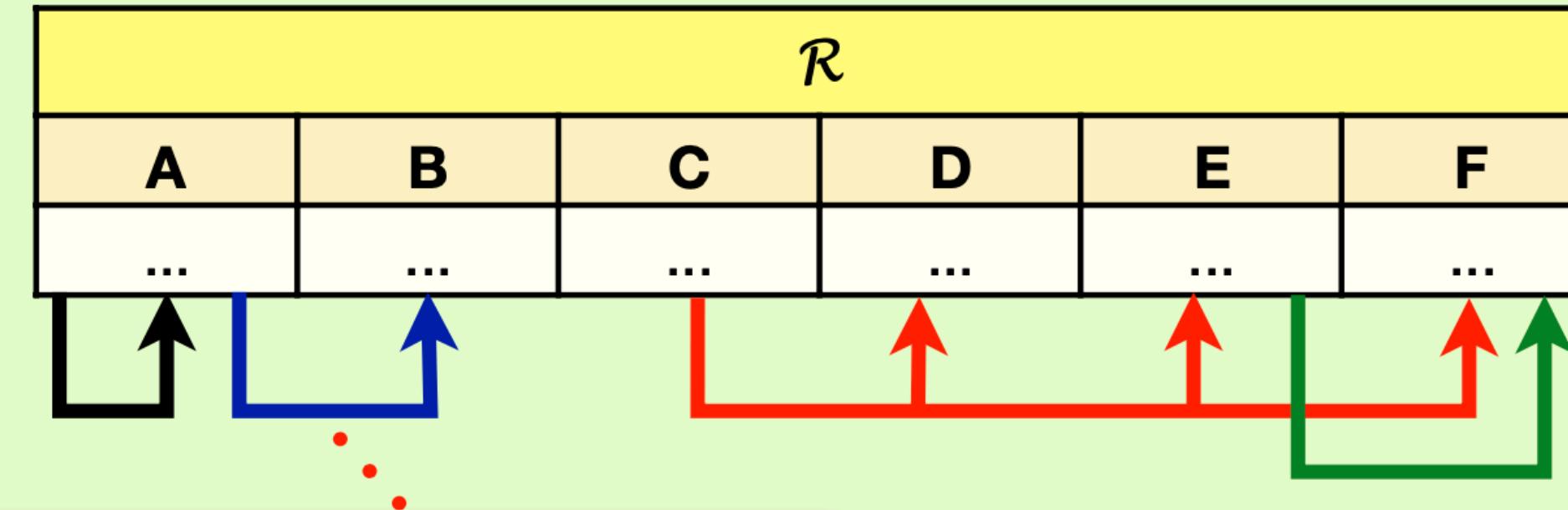
# Synthesealgorithmus

## Beispiel 1

Wir betrachten ein Beispiel zur Veranschaulichung der Notwendigkeit des 3. Schrittes im Synthesealgorithmus

Seien  $\mathcal{R} = \{A, B, C, D, E, F\}$  und folgende FDs gegeben:

- ▶ 1:  $\{A\} \rightarrow \{A\}$
- ▶ 2:  $\{A\} \rightarrow \{B\}$
- ▶ 3:  $\{C\} \rightarrow \{D, E, F\}$
- ▶ 4:  $\{E\} \rightarrow \{F\}$



**R nicht in 2. NF!**

**Schlüsselkandidaten:**

$L: \beta \subseteq \text{AttrHülle}(F, \alpha - \{a\})$

Linke Seiten der FDs betrachten:  $\{A, C, E\}$  ist offenbar Superschlüssel

**Linksreduktion:**

- Entfernen von A:  $\{C, E\} \xrightarrow{3} \{C, D, E, F\}$  - nicht möglich
- Entfernen von C:  $\{A, E\} \xrightarrow{2} \{A, B, E\} \xrightarrow{4} \{A, B, E, F\}$  - nicht möglich
- Entfernen von E:  $\{A, C\} \xrightarrow{2} \{A, B, C\} \xrightarrow{3} \{A, B, C, D, E, F\}$  - möglich!

**{A,C} ist einziger Schlüsselkandidat**

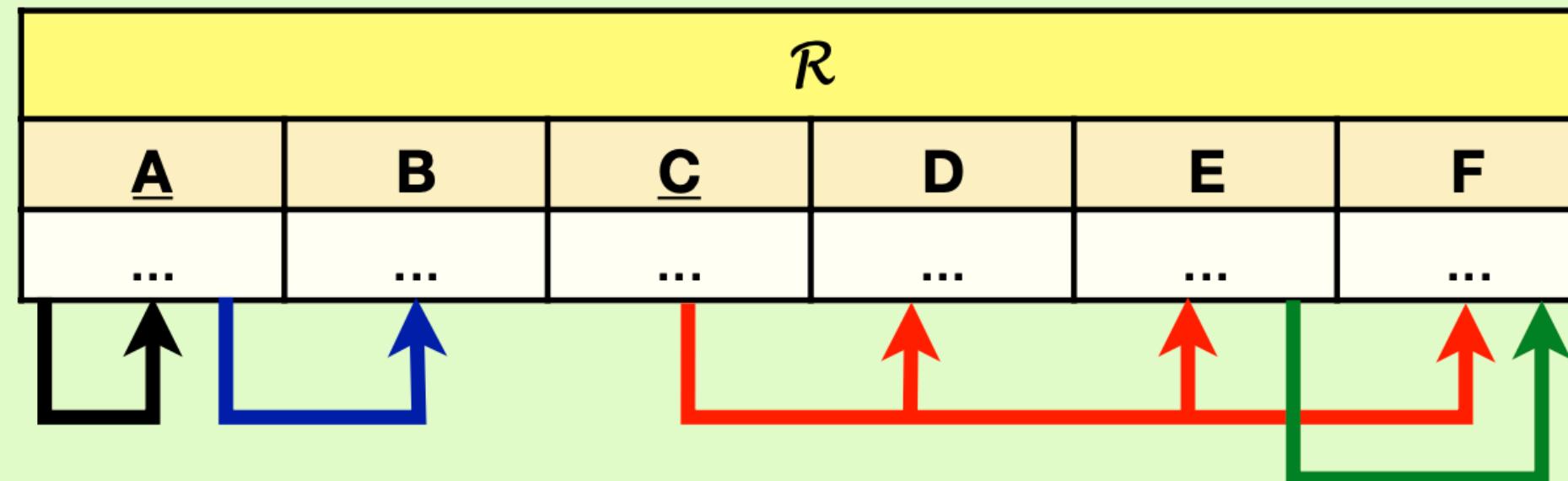
# Synthesealgorithmus

## Beispiel 1

Wir betrachten ein Beispiel zur Veranschaulichung der Notwendigkeit des 3. Schrittes im Synthesealgorithmus

Seien  $\mathcal{R}=\{A, B, C, D, E, F\}$  und folgende FDs gegeben:

- ▶ 1:  $\{A\} \rightarrow \{A\}$
- ▶ 2:  $\{A\} \rightarrow \{B\}$
- ▶ 3:  $\{C\} \rightarrow \{D, E, F\}$
- ▶ 4:  $\{E\} \rightarrow \{F\}$



### Kanonische Überdeckung:

#### Linksreduktion:

- nichts zu tun

#### Rechtsreduktion

- $\{A\} \rightarrow \emptyset$  (trivial)
- $\{A\} \rightarrow \{B\}$  - nichts zu tun
- $\{C\} \rightarrow \{D, E, F\}$  entferne F, denn  $\{C\} \rightarrow_3 \{D, E\} \rightarrow_4 \{D, E, F\}$
- $\{E\} \rightarrow \{F\}$  - nichts zu tun

$$F^C = \{A \rightarrow B, C \rightarrow DE, E \rightarrow F\}$$

# Synthesealgorithmus

## Beispiel 1

### Synthesealgorithmus:

$$F^C = \{A \rightarrow B, C \rightarrow DE, E \rightarrow F\}$$

Schlüsselkandidaten = { {A,C} }

### Schemata erstellen:

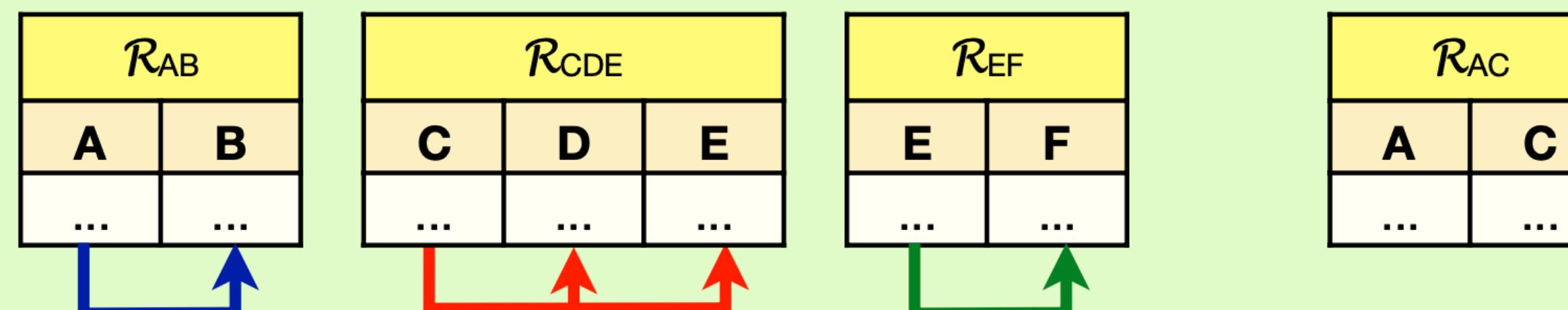
- $R_{AB} = \{A, B\}$  mit  $F_{AB} = \{A \rightarrow B\}$
- $R_{CDE} = \{C, D, E\}$  mit  $F_{CDE} = \{C \rightarrow DE\}$
- $R_{EF} = \{E, F\}$  mit  $F_{EF} = \{E \rightarrow F\}$

### Test auf Schlüsselkandidat

- Keine Relation enthält den einzigen Schlüsselkandidaten {A,C}
- $R_{AC} = \{A, C\}$  mit  $F_{AC} = \emptyset$

### Streichen von überflüssigen Relationen

- Hier nicht notwendig



## Synthesealgorithmus

### Beispiel 2

$$\mathcal{R} = \{A, B, C, D, E\}$$

$$\mathcal{F}_n = \{AD\bar{E} \rightarrow BE, B \rightarrow AC\bar{D}, \bar{D} \rightarrow AC, \\ ACE \rightarrow BD, C \rightarrow B\}$$

Schlüssel

- alle Attribut Werte ... (rechte Seite), d.h.

teste:

- wir brauche:  $\exists \rightarrow ABCDE$  min.

- wir betrachten Attrib. Menge ( $\mathcal{F}_n, \exists$ ) =  $\exists^+$   
 also  $A^+ = A$

$$\exists^{\#} \rightarrow BACD, \text{ also } \exists^+ = ABCD$$

$$C \rightarrow CB \rightarrow ABCD$$

$$D \rightarrow DAC \rightarrow DACB \rightarrow ABCD$$

$$E \rightarrow E$$

"

- Fest steht:  $E$  folgt nicht...  
 normalerweise hinreichend:  $AD\bar{E} \rightarrow BE$ , d.h.  
ohne  $E$  kein  $E$ , also nur  $E$  in der Schlüssel.

$$\begin{aligned} AE &\rightarrow AE \\ BE &\rightarrow ADCDE \quad \checkmark \\ CE &\rightarrow ABCDE \quad \checkmark \\ DE &\rightarrow ABCDE \quad \checkmark \end{aligned}$$

- $BE, CE, DE$  Schlüsselnd. (nicht zu reduzieren)  
 Prim:  $B, C, D, E$ , Nicht-Prim:  $A$  (w.  $D \rightarrow A$  nicht zuf)

# Synthesealgorithmus

## Beispiel 2

Links red.

- Beinhaltet nur FDs mit mehr als einer Attributstr.

$$\underline{A \rightarrow DE} \rightsquigarrow \underline{BE}$$

A weglassen:  $D\bar{E} \rightarrow D\bar{E}AC \rightarrow D\bar{E}ACB$   
also  $BE \subset D\bar{E}^+$

d.h. A nicht notwendig

$$\text{neue } F_n = \{ D\bar{E} \rightarrow BE, \dots \}$$

E weglassen:  $D \rightarrow DAC \rightarrow DACB$  und  $BE \not\subset ABC$   
d.h. E notwendig

D weglassen:  $E \rightarrow E$  und  $BE \not\subset E$   
d.h. D notwendig

- $\underline{ACE} \rightarrow BD$

A weglassen:  $C\bar{E} \rightarrow C\bar{E}B \rightarrow C\bar{E}BA\bar{D}$  und  $BD \subset ABC$   
d.h. A nicht notwendig

neue  $F_n = \{ \dots, C\bar{E} \rightarrow BD, \dots \}$

C weglassen:  $E \rightarrow E$ , d.h.  $BD \not\subset E$  aber notwendig

E weglassen:  $C \rightarrow CB \rightarrow CBA\bar{D}$  d.h.  $BD \subset CAB$   
also E nicht notwendig

neue  $F_n = \{ \dots, C \rightarrow BD, \dots \}$

- flesant:  $F_n = \{ D\bar{E} \rightarrow BE, B \rightarrow ACD, D \rightarrow AC,$   
 $C \rightarrow BD, C \rightarrow B \}$

# Synthesealgorithmus

## Beispiel 2

Reduktion

betrachte  $\tilde{F}_n = \{ DE \rightarrow B, DE \rightarrow E, B \rightarrow A, B \rightarrow C, \underline{B \rightarrow D}, \underline{D \rightarrow A}, \underline{D \rightarrow C}, C \rightarrow B, C \rightarrow D, C \rightarrow B \}$

$DE \rightarrow B$

Akh. Wille ( $\tilde{F}_n \setminus \{ DE \rightarrow B \}, DE \right)$ :

\*  $DE \rightarrow DEAC \rightarrow DEACB$  und  $B \subset ABCDE$   
 also  $DE \rightarrow B$  nicht notwendig

neuer  $\tilde{F}_n = \{ DE \rightarrow E, B \rightarrow A, \dots \}$  (ohne  $DE \rightarrow B$ )

$DE \rightarrow E$

Akh. Wille ( $\tilde{F}_n \setminus \{ DE \rightarrow E \}, DE \right)$ :

$DE \rightarrow DEAC \rightarrow DEACB \quad \text{und} \quad E \subset DEACB$

(also  $DE \rightarrow E$  nicht notwendig)

(oft erschöpft)

neuer  $\tilde{F}_n = \{ B \rightarrow A, B \rightarrow C, \dots \}$

$B \rightarrow A$

Akh. Wille ( $\tilde{F}_n \setminus \{ B \rightarrow A \}, B \right)$ :

$B \rightarrow BCD \rightarrow BCD\underline{A}$  also  $A \subset BCDA$   
 und  $B \rightarrow A$  nicht notw.

neuer  $\tilde{F}_n = \{ B \rightarrow C, B \rightarrow D, \dots \}$

## Synthesealgorithmus

### Beispiel 2

$B \rightarrow C$

Ath. Wölle ( $\tilde{F}_n \setminus \{B \rightarrow C\}, B$ ):

$B \rightarrow B \rightarrow B \rightarrow A \rightarrow C$  d.h.  $C \in BDAC$

und  $B \rightarrow C$  nicht wahr.

neu:  $\tilde{F}_n = \{B \rightarrow D, D \rightarrow A, \dots\}$

$B \rightarrow D$

Ath. Wölle ( $\tilde{F}_n \setminus \{B \rightarrow D\}, B$ ):

$B \rightarrow B \rightarrow D$  als  $B \rightarrow D$  wahr.

$D \rightarrow A$

Ath. Wölle ( $\tilde{F}_n \setminus \{D \rightarrow A\}, D$ ):

$D \rightarrow D \rightarrow C \rightarrow D \rightarrow B$  also  $D \rightarrow A$  wahr. ( $A \notin DCB$ )

$D \rightarrow C$

Ath. Wölle ( $\tilde{F}_n \setminus \{D \rightarrow C\}, D$ ):

$D \rightarrow D \rightarrow A$  also  $D \rightarrow C$  wahr. ( $C \notin DA$ )

$C \rightarrow B$

doppelt, also strikt

neu:  $\tilde{F}_n = \{B \rightarrow D, D \rightarrow A, D \rightarrow C, C \rightarrow D, C \rightarrow B\}$

$C \rightarrow B$

Ath. Wölle ( $\tilde{F}_n \setminus \{C \rightarrow B\}, C$ ):

$C \rightarrow CB \rightarrow CBD$  also nicht wahr. ( $D \subset CB$ )

neu:  $\tilde{F}_n = \{B \rightarrow D, D \rightarrow A, D \rightarrow C, C \rightarrow B\}$

$C \rightarrow B$

Ath. Wölle ( $\tilde{F}_n \setminus \{C \rightarrow B\}, C$ ):

$C \rightarrow C$  also wahr.

## Synthesealgorithmus

### Beispiel 2

- nach Multimod.

$$\widehat{\mathcal{F}}_n = \{ B \rightarrow D, \overset{\uparrow}{D} \rightarrow AC, C \rightarrow B \}$$

mit. geprägt

= Kanonisch Überdeckung

- M. schreibe

$$\mathcal{R}_1 = \{ \cancel{B \rightarrow D} \} \quad \mathcal{F}_1 = \{ B \rightarrow D \}$$

$$\mathcal{R}_2 = \{ \cancel{D \rightarrow AC} \} \quad \mathcal{F}_2 = \{ D \rightarrow AC \}$$

$$\mathcal{R}_3 = \{ BC \} \quad \mathcal{F}_3 = \{ C \rightarrow B \}$$

- mind. ein Sch. Kandidat erh. ? ( $BE, CE, DE$ )

Nei, also himmfig, d.h.

$$\mathcal{R}_4 = \{ BE \}, \quad \mathcal{F}_4 = \{ \} \quad (\text{Sch. Kandidat beliebig})$$

## Boyce-Codd Normalform

### Motivation

- Alle Attribute sind prim. Damit ist 3NF erfüllt...

$\mathcal{R}$		
PLZ	Ort	Strasse
52080	Aachen	Von-Coels-Str.
52080	Aachen	Karlsstr.



$\mathcal{R}$  ist in 3. Normalform mit folgenden FDs

- ▶ 1:  $\{PLZ\} \rightarrow \{Ort\}$
- ▶ 2:  $\{Ort, Strasse\} \rightarrow \{PLZ\}$

**alle Attribute sind prim!**

**Schlüsselkandidaten**

Linke Seiten der FDs:  $\{Ort, Strasse, PLZ\}$  ist offenbar Superschlüssel

**Linksreduktion (Beachte:** nicht deterministisch!)

- Variante 1:  $\{Strasse, PLZ\} \xrightarrow{1} \{Strasse, PLZ, Ort\}$  - keine weitere Reduktion möglich
- Variante 2:  $\{Ort, Strasse\} \xrightarrow{2} \{Ort, Strasse, PLZ\}$  - keine weitere Reduktion möglich

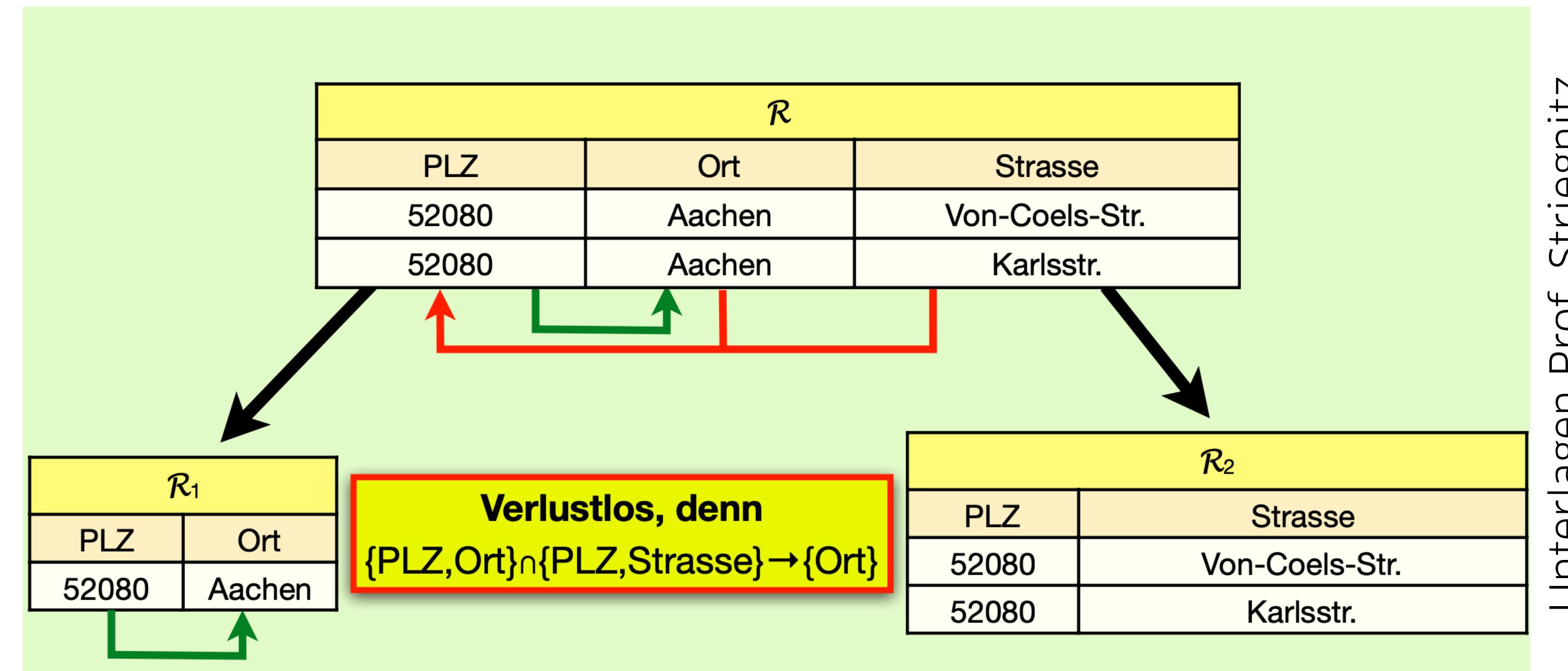
**2 Schlüsselkandidaten:  $\{Strasse, PLZ\}$  und  $\{Ort, Strasse\}$**

**Die 3NF verhindert offenbar nicht alle Redundanzen.**



## Boyce-Codd Normalform

## Motivation



## Boyce-Codd Normalform

---

### Definition

- Ein Schema R mit einer Menge funktionaler Abhängigkeiten FD ist in Boyce-Codd Normalform (BCNF), wenn für jede funktionale Abhängigkeit  $\alpha \rightarrow \beta \in FD^+$  mindesten eine der folgenden Bedingungen gilt:
  - $\alpha \rightarrow \beta$  ist trivial, also  $\beta \subseteq \alpha$ ,
  - $\alpha$  ist Superschlüssel.

Beim Test auf BCNF testet man, wie bei 3NF, einfach alle Abhängigkeiten, ob eine der zwei Bedingungen zutrifft.



### Anmerkungen

- Vgl. 3NF, bei BCNF fällt die zweite Bedingung weg.
- Die dritte Normalform ist eingeschlossen.

## Boyce-Codd Normalform

---

### Anmerkungen

- Man kann jedes Schema R mit funktionalen Abhängigkeiten FD so in Schemata  $R_1, \dots, R_n$  zerlegen, dass BCNF erfüllt ist.



**Die Zerlegung nicht notwendigerweise abhängigkeitserhaltend!**

- Der folgende *Dekompositionsalgorithmus* setzt genau das um und durch die Aufteilung der Relationen können funktionale Abhängigkeiten, wie im motivierenden Beispiel, verloren gehen.
- Die Voraussetzung  $\alpha \rightarrow \beta \in FD^+$  ist wichtig, denn nicht immer sind alle Abhängigkeiten explizit gegeben - siehe Beispiele.

# Dekompositionsalgorithmus

## Definition

- **Starte mit  $Z = \{\mathcal{R}\}$**
- **Solange es noch ein Relationenschema  $\mathcal{R}_i$  in  $Z$  gibt, das nicht in BCNF ist, mache folgendes:**

**Anmerkung:** Es gibt also eine für  $\mathcal{R}_i$  geltende funktionale Abhängigkeit  $\alpha \rightarrow \beta$  mit  $\alpha \cap \beta = \emptyset$  (nicht trivial) und  $\neg(\alpha \rightarrow \mathcal{R}_i)$  ( $\alpha$  kein Superschlüssel)

  - ▶ **Finde eine solche FD**

**Anmerkung:** Man sollte die FD so wählen, dass  $\beta$  alle von  $\alpha$  funktional abhängigen Attribute  $b \in (\mathcal{R}_i - \alpha)$  enthält, damit der Dekompositionsalgorithmus möglichst schnell terminiert.
  - ▶ **Zerlege  $\mathcal{R}_i$  in  $\mathcal{R}_{i1} = \alpha \cup \beta$  und  $\mathcal{R}_{i2} := \mathcal{R}_i - \beta$** 

Die FDs verteilen sich entsprechend - kommen die Attribute in einer FD in keiner Relation mehr geschlossen vor, entfällt diese FD → Verlust von Abhängigkeiten!
  - ▶ **Entferne  $\mathcal{R}_i$  aus  $Z$  und füge  $\mathcal{R}_{i1}$  und  $\mathcal{R}_{i2}$  ein**

also setze  $Z = (Z - \{\mathcal{R}_i\}) \cup \{\mathcal{R}_{i1}\} \cup \{\mathcal{R}_{i2}\}$

# Dekompositionsalgorithmus

## Beispiel 1

Bsp. BCNF

$$\mathcal{R} = \{ A, B, C, D \}$$

$$\mathcal{F}_R = \{ BC \rightarrow D, AB \rightarrow A \}$$

Schlüssel?  $ABC$ 

Dekompositionsalg. (keine kan. Übereinst.)

Test ob für  $\alpha \rightarrow \gamma$ entweder 1.  $\gamma \subseteq \alpha$ oder 2.  $\alpha$  SuperschlüsselFinde  $\alpha \rightarrow \gamma$ , die eine Bed. reicht ...

$$BC \rightarrow D$$

1.  $D \not\subseteq BC$ 2.  $BC$  kein SuperSl.

also

$$\mathcal{R}_1 = \underbrace{BCD}_{\alpha \quad \gamma}$$

$$\mathcal{F}_{\mathcal{R}_1} = \{ BC \rightarrow D \}$$

$$\mathcal{R}_2 = ABCD \setminus D = ABC$$

$$\mathcal{F}_{\mathcal{R}_2} = \{ AB \rightarrow A \}$$

## Dekompositionsalgorithmus

## Beispiel 2

Bsp. BCNF

$$\mathcal{R} = \{ A, B, C, D \}$$

$$\mathcal{F}_R = \{ C \rightarrow A, BD \rightarrow AC, C \rightarrow B \}$$

keine Schlüsselknd.

$$\begin{matrix} CD \\ BD \end{matrix}$$

Test BCNF:

$$\begin{aligned} C \rightarrow A : & \quad A \notin C \text{ also } \times \\ & C \text{ ke Superkl. also } \times \end{aligned}$$

also

$$\mathcal{R}_1 = AC$$

$$\mathcal{F}_{\mathcal{R}_1} = \{ C \rightarrow A \}$$

$$\mathcal{R}_2 = BD$$

achtg

$$\mathcal{F}_{\mathcal{R}_2} = \{ BD \rightarrow C \\ C \rightarrow B \}$$

↑ BCNF?

 $\mathcal{R}_2$ : End. Schlüssel BD,also für  $C \rightarrow B$  ist C ke Superkl.d.h.  $\mathcal{R}_2$  reicht BCNF

also

$$\mathcal{R}_{21} = BC$$

$$\mathcal{F}_{\mathcal{R}_{21}} = \{ C \rightarrow B \}$$

$$\mathcal{R}_{22} = \{ C \}$$

$$\mathcal{F}_{\mathcal{R}_{22}} = \{ \}$$

wert

# Dekompositionsalgorithmus

## Beispiel 3

Bsp.       $\overline{FD} = \{ B \rightarrow A, D \rightarrow ACE, AC \rightarrow DE, D \rightarrow AB \}$   
 $R = ABCDE$

$B \rightarrow A$  verletzt BCNF

$$\begin{array}{ll} \downarrow & \longrightarrow \\ R_1 = A \underline{B} & R_2 = R - A = \underline{BCDE} \\ FD_1 = \{ B \rightarrow A \} & FD_2 = \{ D \rightarrow BCE \} \end{array}$$

Verlust:  $AC \rightarrow DE$

# Dekompositionsalgorithmus

## Beispiel 4

Bsp.

$$\text{FD} = \{ACD \rightarrow E, ABE \rightarrow CD, AC \rightarrow DE, BC \rightarrow E\}$$

$$\mathcal{R} = ABCDE$$

$$\text{SL. knd. : } A\bar{B}C, A\bar{B}E$$

$$\text{Syllese : } \text{FD}^c = \{ABE \rightarrow C, AC \rightarrow DC, BC \rightarrow E\}$$

$$\mathcal{R}_1 = \underline{ABC}\bar{E} \text{ ob } \underline{ABC}\bar{E}, \quad \text{FD}_1 = \{A\bar{B}E \rightarrow C\}$$

$$\mathcal{R}_2 = \underline{ACD}\bar{E}, \quad \text{FD}_2 = \{AC \rightarrow DC\}$$

$$\mathcal{R}_3 = \underline{BC}\bar{E}, \quad \text{FD}_3 = \{BC \rightarrow E\}$$

hier  $\mathcal{R}_3 \subset \mathcal{R}_1$  also  $\mathcal{R}_3$  weglassen (d  $\text{FD}_3$  in  $\text{FD}_1$ )

Dekomp :

$ACD \rightarrow E$  verletzt BCNF ( $ACD$  ist Superfl.)



$$\mathcal{R}_1 = \underline{ACD}\bar{E}$$

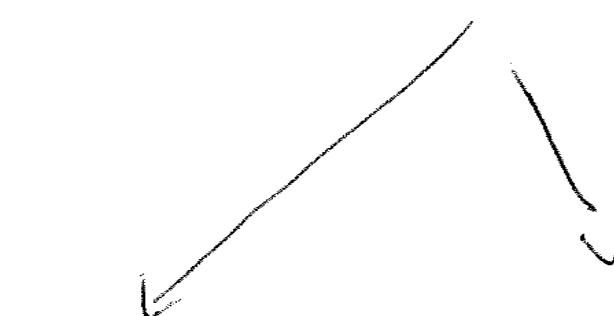
$$\text{FD}_1 = \{ACD \rightarrow E, AC \rightarrow DE\}$$

$$\mathcal{R}_2 = \mathcal{R} - E = \underline{ABCD}$$

$$\text{FD}_2 = \{AC \rightarrow D\}$$

(neue knd. SL. folgt aus  
neuer Syllese)

hier:  $AC \rightarrow D$  verletzt  
BCNF  
da  $AC$  ist  
Superfl.



$$\mathcal{R}_{21} = \underline{ACD}$$

$$\text{FD}_{21} = \{AC \rightarrow D\}$$

$$\mathcal{R}_{22} = ABCD - D = \underline{ABC}$$

$$\text{FD}_{22} = \{\}$$

## Dekompositionsalgorithmus

### Beispiel 5

Bsp.

$$FD = \{ A \rightarrow B, B \rightarrow DE, C \rightarrow B, E \rightarrow F, AC \rightarrow BD \}$$

bcd. s.t. AC

$A \rightarrow B$  verletzt BCNF

$$R_1 = \underline{AB}$$

$$FD_1 = \{ A \rightarrow B \}$$

$$R_2 = \underline{ACDEF}$$

$$FD_2 = \{ \underline{E} \rightarrow F, AC \rightarrow D, \\ A \rightarrow DEF, \\ C \rightarrow DEF, \\ AC \rightarrow EF \}$$

sub (e FD)

hau'kr

$E \rightarrow F$  verletzt BCNF

$$R_{21} = \underline{EF}$$

$$FD_{21} = \{ E \rightarrow F \}$$

$$R_{22} = \underline{ACDE}$$

$$FD_{22} = \{ AC \rightarrow D, \\ A \rightarrow DE, \\ C \rightarrow DE, \\ AC \rightarrow E \}$$

$$R_{21} = \underline{EF}$$

$$FD_{21} = \{ E \rightarrow F \}$$

$$R_{22} = \underline{ACDE}$$

$$FD_{22} = \{ AC \rightarrow D, \\ A \rightarrow DE, \\ C \rightarrow DE, \\ AC \rightarrow E \}$$

$A \rightarrow DE$  verletzt BCNF

$$R_{221} = \underline{ADE}$$

$$FD_{221} = \{ A \rightarrow DE \}$$

$$R_{222} = \underline{AC}$$

$$FD_{222} = \{ \}$$

## Finale

---

### Aufgabe

- Entwicklung einer 'guten' Datenbank [...] u.a. minimal, redundanzfrei [...]

### Vorgehen

- Konzeptueller Entwurf [...] ER-Modells ✓
- Überführung in eine 'gute' Datenbank [...] dazu benötigen wir:
  - Implementationsentwurf [...] → Relationales Modell ✓
  - Mathematisches Modell [...] → Relationale Algebra ✓
  - Gütemaß → Normalformen (Anomalien, Verlustlosigkeit, Abhängigkeitserhaltung)
    - ♦ Kanonische Überdeckung ✓, funktionale Abhängigkeiten ✓, Attributhülle ✓,
    - ♦ Synthesealgorithmus ✓,
    - ♦ Dekompositionsalgorithmus ✓.

## Finale

---

### Schlussbemerkungen

- Wie am Beispiel der Artikeldatenbank gezeigt, kann man *mit Verständnis für die Problematik von Anomalien und Redundanzen* sowohl aus einer großen Tabelle, funktionalen Abhängigkeiten und Normalisierung (Universal Relation Assumption), als auch über gut modellierte ER-Diagramme und der Umsetzung in die Relationen, einen guten Datenbankentwurf, d.h. die konkreten Tabellen, ableiten!
- Nicht immer ist eine Normalisierung bis ins Kleinste anzustreben:
  - Durch die Aufteilung der Daten entsteht Overhead im Datenhandling (Joins), der Durchsatz kann leiden, etc. Hier können bewusst eingesetzte Redundanzen zur Performancesteigerung beitragen.
  - SQL-Integritätsbedingungen können ebenfalls helfen, Anomalien zu vermeiden.

**Wie immer Sie modellieren - beide Ansätze zu Kennen ist wichtig!**



## Typische Prüfungsaufgabe

---

### Aufgabe

- Ermitteln Sie für das vorliegende Schema alle Schlüsselkandidaten.
- Prüfen Sie, in welcher Normalform das Schema vorliegt.
- Entweder: Überführen Sie es mit dem Synthesealgorithmus in 3NF, falls notwendig.
- Oder: Überführen Sie es mit dem Dekompositionsalgorithmus in BCNF, falls notwendig.

### Anmerkung

Man findet im Internet viele "Algorithmen" zur "schnellen und unkomplizierten" Berechnung aller Schlüsselkandidaten... Folgt man aber der Literatur, etwa Lucchesi, Osborn (1978). *Candidate keys for relations*. *Journal of Computer and System Sciences*, 17, so stellt sich heraus, dass das Ermitteln aller Schlüsselkandidaten im worst case exponentielle Komplexität besitzt. Also Vorsicht vor falschen Rezepten.

## Beispiel

$$R = \{A, B, C, D, E, F\}$$

①

$$FD_S = \{BCE \rightarrow D, B \rightarrow CEF, DE \rightarrow BC\bar{F}\}$$

immer benötigt: Schlüsselkandidaten  $\rightsquigarrow$  A fehlt auf rechter Seite,  
d.h. kann nicht gefolgt werden,  
muss also in Schlüssel  $\triangleright$

- Alt. Menge ( $FD, AB$ ) =  $ABC\bar{D}EF \vee$   
(kurz:  $AB \rightarrow ABC\bar{E}F \rightarrow ABCDEF$ ) also  $AB$  Schlüsselkandidat  $\triangleright$
- $AC, AD, \dots$  braucht nicht präsent zu werden, da aus C, D, etc. nichts folgt.
- $ADE$  ist die nächste sinnvolle Kombination (siehe  $FD$ )  
also  $ADE \rightarrow ABCDEF \vee$  also  $ADE$  Schlüsselkand.  $\triangleright$
- $ABC\bar{E}$  Superschlüssel
- wg Struktur von  $FD$  keine weiteren Schlüsselkandidaten
- also  $AB, ADE$  Schlüsselkand.,  $A, B, D, E$  prim  
 $C, F$  nicht-prim

## Beispiel

Welche NF liegt vor?

NF1 ✓

NF2: wg.  $B \rightarrow CEF$  und  
 $AB$  Schlüsselkandidat,  
 insbes.  $B \rightarrow C$ , mit  
 $C$  nicht-prim, liegt  
 NF2 nicht vor.

NF3, BCNF wg. fehlede NF2  
 auch nicht ?

Kanonische Überdeckung für Synthesealg.:

Linksnormalisierung:

$BCE \rightarrow D$ : formal naheinander  $C, E^-$   
 streichen, aber wg.  $B \rightarrow CEF$   
 sieht man darunter  
 beide streichen kann ✓  
 $B \rightarrow BCEF \rightarrow BCD^+EF$   
 also durch  $B \rightarrow D$  ersetzen

- |  $B \rightarrow CEF$  und nicht  
unterstellt werden (2)
- |  $DE \rightarrow BCF$ : wg.  $D \rightarrow D$ ,  
 $E \rightarrow E^-$   
leider nichts geschieht werden
- | also  $FD' = \{B \rightarrow D, B \rightarrow CEF,$   
 $DE \rightarrow BCF\}$
- | Reduktionsreduzibilität:
  - |  $B \rightarrow D$ : formal  
Alt. Nullte ( $FD' \setminus \{B \rightarrow D\}$ ,  $B$ )  
 $= BCEF \not\ni D$
  - | oder: ohne  $B \rightarrow D$  leidet  $D$  nicht  
unter, also  $B \rightarrow D$  bleibt
- |  $B \rightarrow CEF$ :
  - | Alt. Nullte ( $FD' \setminus \{B \rightarrow CEF\} \cup \{B \rightarrow CE\}$ ,  $B$ )  
 $= BCDEF \ni CEF$
  - | also  $F$  überflüssig,  $FD'' = \{B \rightarrow D,$   
 $B \rightarrow CE,$   
 $DE \rightarrow BCF\}$
- |  $B \rightarrow CE$ : ...

## Beispiel

$B \rightarrow CE$ :

$$\text{Attr. Hülle } (FD)^H \setminus \{B \rightarrow CE\} \cup \{B \rightarrow C\}, B \\ = BCD \not\Rightarrow CE$$

also  $E$  notwendig

$$\text{Attr. Hülle } (FD)^H \setminus \{B \rightarrow CE\} \cup \{B \rightarrow E\}, B \\ = BCDEF \not\Rightarrow CE$$

also  $C$  überflüssig

$$FD^H = \{B \rightarrow D, B \rightarrow E, DE \rightarrow BCF\}$$

$DE \rightarrow BCF$ :

$$\text{Formal Attr. Hülle } (FD)^H \setminus \{DE \rightarrow BCF\} \\ \cup \{DE \rightarrow \dots\}, DE \\ = \dots$$

Oder: Weder  $B$ , noch  $C$ , noch  $F$

folgt aus  $\overline{FD}^H \setminus \{DE \rightarrow BCF\}$

also alle notwendig

$$\Rightarrow \overline{FD}^C = \{B \rightarrow DE, DE \rightarrow BCF\}$$

| Synthesearc.

$$|\bullet R_1 = \{BDE\}, FD_1 = \{B \rightarrow DE, DE \rightarrow B\}$$

$$|\bullet R_2 = \{BCDEF\}, FD_2 = \{B \rightarrow DE, DE \rightarrow BCF\}$$

$$|\bullet \text{ Weder } AD \text{ noch } ADE \text{ in Schenata,} \\ |\bullet \text{ also } R_3 = \{AB\}, FD_3 = \{\}$$

|  $R_1 \subset R_2$ , also final:

$$R_1 = \{BCDEF\}$$

$$FD_1 = \{B \rightarrow DE, DE \rightarrow BCF\}$$

$$R_2 = \{AB\}$$

$$FD_2 = \{\}$$

(3)

## Beispiel

### Dekompositionsalg.

$$FD = \{ BCE \rightarrow D, B \rightarrow CEF, DE \rightarrow BCF \}$$

- Teste fktl. Abh. auf BCNF Kriterie:

$BCE \rightarrow D$ :  $BCE$  kein Superkey,  
also  $BCNF$  verletzt

$$R_1 = \{ BCDE \}, FD_1 = \{ BCE \rightarrow D, B \rightarrow CE, DE \rightarrow BC \}$$

$$R_2 = \{ ABCEF \}, FD_2 = \{ B \rightarrow CEF \}$$

- Anmerkung: weitere fktl. Abh. aus da DB-Normalisierung unnötig

- unterscheide nur  $R_1, R_2$

- $R_1$ : Schlüsselknd.:  $B$  und  $DE$

$\Rightarrow$  nur Superkey auf  
linker Seiten, also  
 $BCNF$  liegt vor

$R_2$ : Schlüsselknd.  $AB$   
↑ nicht rezipise  
 $\Rightarrow B \rightarrow CEF$  verletzt  $BCNF$   
da  $B$  kein Superkey

• Auflösung von  $R_2$ :

$$R_{21} = \{ BCEF \}, FD_{21} = \{ B \rightarrow CEF \}$$

Schlüssel  $B$ , und somit  $BCNF$

$$R_{22} = \{ AB \}, FD_{22} = \{ \}$$

Final:  $R_1, R_{21}, R_{22}$   
mit  $FD_1, FD_{21}, FD_{22}$



# UNIT 0X0A

## SQL I

## Hintergrund

---

### **Einordnung der Kapitel SQL I und II**

- Die praktischen Übungen zu SQL und etwas Hintergrund dazu gibt es regulär im SQL-Praktikum.
- Die Folien hier greifen einzelne Aspekte und Variationen etwas breiter auf, dennoch sind sie kein Nachschlagewerk. Dazu bemühe man die aktuelle Dokumentation der jeweiligen DBMS.

## Hintergrund SQL

---

### **Ursprung**

- Entwickelt bei IBM von Donald Chamberlin und Raymond Boyce 1970er im Zusammenhang mit System-R.
- Ursprünglicher Name SEQUEL (Structured English Query Language).

### **SQL besteht aus**

- Datendefinitions (DDL)-,
- Datenmanipulations (DML)- und der
- Abfrage (Query)-Sprache.

## Hintergrund SQL

### Standard

- 1986 erstmalig standardisiert durch ANSI/ISO
- 1992 - neue Datentypen, neue JOIN-Syntax, ...
- 1999 - BOOLEAN, Objekt-relationale Erweiterungen, stored procedures
- 2003 - xml, Sequenz-Generatoren
- ...

<https://www.iso.org/committee/45342/x/catalogue/p/1/u/0/w/0/d/0>

#### © ISO/IEC 9075-1:2016

Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework)

#### © ISO/IEC 9075-2:2016

Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation)

#### © ISO/IEC 9075-2:2016/COR 1:2019

Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation) -

#### © ISO/IEC 9075-3:2016

Information technology – Database languages – SQL – Part 3: Call-Level Interface (SQL/CLI)

#### © ISO/IEC 9075-4:2016

Information technology – Database languages – SQL – Part 4: Persistent stored modules (SQL)

#### © ISO/IEC 9075-4:2016/COR 1:2019

Die verschiedenen DBMS implementieren, trotz "Standardisierung", häufig im Detail unterschiedliche Features. Man sollte generell *nicht* davon ausgehen, dass SQL-Code bzw. -Skripte, die etwa für MySQL laufen, auch auf einem Oracle DBMS laufen... Bestenfalls kann man sie einfach adaptieren.



## Hintergrund SQL

---

### Kritik

- Abweichung vom theoretischen Modell
  - Mengen vs. Listen
  - Reihenfolge von Tupeln relevant (z.B. LIMIT)
  - NULL im theoretischen Modell nicht existent
- Inkompatible Implementierungen (trotz Standardisierung!)
  - Darstellung von Zeit / Datum
  - Behandlung von NULL
  - nur PostgreSQL hat den Anspruch standard-nah zu sein, ist aber nicht standard-konform (es fehlt z.B. der Typ BLOB).

# Übersicht

---

## Themen

- Datentypen und -strukturen
- Anfragen/Select
- Ausdrücke und Funktionen
- Gruppieren/Aggregatfunktionen
- Optimierung
- Umbenennung
- Mengenoperationen
- Unterabfragen/Subselect
- With
- Korrelierte und unkorrelierte Anfragen
- IN/ALL/ANY/EXISTS
- Fallunterscheidung/CASE

# Datentypen und -strukturen

## Zeichenketten (*character string types*)

- **character[ (n) ] | char[ (n) ]**  
Zeichenkette fester Länge  $n$ ; ggf. mit Leerzeichen aufgefüllt
- **character varying[ (n) ] | varchar[ (n) ] | char varying[ (n) ]**  
Zeichenkette variabler Länge  $n$
- **character large object [ (n) ] | clob [ (n) ]**  
Zeichenkette variabler Länge  $n$

Wird die Längenangabe weggelassen, so ist die Länge 1

- PostgreSQL, MS-SQL: Typ **text** anstelle von **clob**
- PostgreSQL: **varchar** ohne Längenangabe ist Zeichenkette beliebiger Länge
- Zulässige Maximallänge kann je nach Implementierung variieren  
z.B. PostgreSQL: **text** max. 2GB; Oracle: **clob** max 4GB;  
DB2: **varchr** max. 32Kb; **clob** max. 2 GB
- SQL-Typen wie "**national char**, **national varchar**" werden so gut wie von keiner Implementierung unterstützt.

# Datentypen und -strukturen

## Exakte numerische Typen

Ganze Zahlen / Zahlen mit fester Zahl an Nachkommastellen

- **numeric[p ,s] ] | decimal[p ,s]] | integer | int | smallint**  
p bezeichnet die Gesamtzahl der Stellen  
s die Anzahl der Nachkommastellen, falls weggelassen gilt s=0

- **numeric** und **decimal** sind synonym
- **int** ist Kurzform zu **integer**
- **integer**: typischerweise 4 Byte
- **smallint**: typischerweise 2 Byte
- Maximale Genauigkeit von **decimal** / **numeric** implementierungsabhängig
  - PostgreSQL: 131.072 Vor- und 16.383 Nachkommastellen
  - MySQL: 65 Stellen
  - DB2: 31 Stellen
  - MS-SQL: 38 Stellen
  - Oracle: 38 Stellen - Oracle kennt nur **number**, nicht die SQL'99 Typen!

# Datentypen und -strukturen

## Annähernd numerische Typen

Fliesskommazahlen (Exponent, Mantisse)

- **real | double precision | float(*n*)**  
*n* ist Genauigkeit in Bits

- **real** typischerweise 4 Byte (PostgreSQL, DB2, MySQL)
- **double precision** typischerweise 8 Byte (PostgreSQL, DB2, MySQL)
- Bereiche für *n*:  
Oracle: 1-126  
MS-SQL, MySQL,DB2,PostgreSQL:  
1-23 Abbildung auf **real**; 24-53 Abbildung auf **double precision**
- Oracle kennt die Typen **real** und **double precision** nicht  
**real** entspricht float(63), **double precision** entspricht float(126)

# Datentypen und -strukturen

---

## Bit Strings

Zeichenkette aus Bits ('0' und '1')

- **bits[ (n) ] | bits varying[ (n) ]**  
*n* exakte (!) Länge bei **bits**, maximale Länge bei **bits varying**

- Bitstrings sind keine Zeichenketten (z.B.ASCII) - Abbildung auf Folge von Bytes
- kaum unterstützter SQL-Typ:  
PostgreSQL, MS-SQL unterstützen Bitstrings,  
Oracle, DB2 und MySQL jedoch nicht

# Datentypen und -strukturen

## Binäre Zeichenketten

- **binary large object[ (n) ] | blob[ (n) ]**

*n* bezeichnet maximale Größe in byte

- Längenangabe kaum unterstützt
- nicht durchgängig unterstützt;  
**Alternativen:**  
PostgreSQL: `bytea`  
MS-SQL: `binary` bzw. `var binary`
- Maximale Größe implementierungsabhängig  
DB2,MS-SQL: 2GB  
Oracle: 4GB

# Datentypen und -strukturen

## Datums- und Zeittypen

- **date**  
Jahr, Monat und Tag
- **time[ (n) ] [ with timezone | without timezone ]**  
Stunde, Minute und Sekunde;  
*n* Nachkommastellen der Sekunden-Komponente  
alternativ: Zeitzonenumwandlung in Stunden (zu UTC)
- **timestamp [ (n) ] [ with timezone | without timezone ]**  
Jahr, Monat, Tag, Stunde, Minute, Sekunde  
*n* Nachkommastellen der Sekunden-Komponente  
alternativ: Zeitzonenumwandlung in Stunden (zu UTC)

- with / without timezone kaum unterstützt  
PostgreSQL: kompatibel
- Besonderheiten bei Oracle:  
`date` entspricht `timestamp(0) without timezone` (also incl. Zeitangabe)  
`time` ist nicht bekannt

# Datentypen und -strukturen

---

## Zeitintervalle

Zeiträume, Zeitdifferenzen

- **interval range**

range bezeichnet die Genauigkeit

YEAR | MONTH | DAY | HOUR | MINUTE | SECOND[ (*n*) ]

YEAR TO MONTH, DAY TO HOUR, DAY TO MINUTE, DAY TO SECOND [ (*n*) ]

- kaum unterstützt  
PostgreSQL und Oracle kennen **interval**

# Datentypen und -strukturen

---

## Boole'scher Typ

- **boolean**  
kann Werte **true** und **false** annehmen

- gut unterstützt  
PostgreSQL, Oracle, DB2 und MySQL kennen **boolean**

# Abfragen / Select

- **select entspricht (fast) der Projektion, where der Selektion**
  - **select PersNr, Name from Professoren where Rang='C4'**
- **Wichtig: SQL eliminiert keine Duplikate**
  - **select Rang from Professoren**  
7 Ergebnisse in SQL, da Duplikate nicht eliminiert werden
  - $\Pi_{\text{Rang}}(\text{Professoren})$   
liefert hingegen nur 2 Ergebnisse
- **Elimination von Duplikaten mit distinct**
  - **select distinct Rang from Professoren**  
liefert (wie gewünscht) 2 Ergebnisse

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

# Abfragen / Select

- **Ergebnis kann nach Spalte(n) sortiert werden:**
  - `select PersNr, Name, Rang, Raum  
from Professoren  
order by Rang desc, Name asc;`



PersNr	Name ↑ <sub>2</sub>	Rang ↓ <sub>1</sub>	Raum
2136	Curie	C4	36
2137	Kant	C4	7
2126	Russel	C4	232
2125	Sokrates	C4	226
2134	Augustinus	C3	309
2127	Kopernikus	C3	310
2133	Popper	C3	52

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

## Abfragen / Select

---

- **select-Klausel darf Ausdrücke (Berechnungen) enthalten**
  - Arithmetische Ausdrücke
  - Ausdrücke über Zeichenketten
  - Ausdrücke über Datum und Zeit
- **Vorgaben aus Standard eher "sparsam"**
  - viele Erweiterungen seitens der Hersteller
  - z.B. keine trigonometrischen Funktionen in SQL'99, SQL'03, aber in MySQL, PostgreSQL, DB2, MS-SQL

# Ausdrücke und Funktionen

---

- **Arithmetische Ausdrücke**

- Operatoren +,- (auch als Präfix-Operator), \* und /

```
select 1+1 from table
select 355/113.0 as PI from table
select preis + (preis*19)/100 as Brutto from Waren
```

- vordefinierte Funktionen in SQL'99
    - **abs(x)** Absolutwert
    - **mod(x,y)** Divisionsrest
  - zusätzlich in SQL'03
    - **floor(x)** Abrunden
    - **ceiling(x)** Aufrunden
    - **sqrt(x)** Quadratwurzel
    - **exp(x)**  $e^x$
    - **ln(x)** natürlicher Logarithmus
    - **power(x,y)**  $x^y$

# Ausdrücke und Funktionen

- **Ausdrücke über Zeichenketten**

- Operatoren: `||` (Konkatenation)
    - SQL'99 setzt Zeichenkettentypen voraus; viele Implementierungen konvertieren ggf. automatisch

```
select 'An ' || Name from Studenten
```

- vordefinierte Funktionen in SQL'99
    - **position(a in b)** Position von Zeichenkette **a** in Zeichenkette **b**
    - **lower(s)** Zeichenkette **s** in Kleinbuchstaben
    - **upper(s)** Zeichenkette **s** in Grossbuchstaben
    - **substring(s from start [for len])**  
Unterzeichenkette von **s** ab Position **start** mit Länge **len**
    - **char\_length(s)** Länge der Zeichenkette **s**
    - **trim([leading | trailing | both] [characters] from s)**  
Entfernt führende (**leading**), am Ende stehende (**trailing**), oder führende und am Ende stehende (**both**) Zeichen in der Liste **characters** aus der Zeichenkette **s**

# Ausdrücke und Funktionen

- **Syntax**  
**string [NOT] LIKE pattern**
- **pattern darf Sonderzeichen (Wildcards) enthalten**
  - '%' als Platzhalter für beliebig viele Zeichen
  - '\_' als Platzhalter für ein Zeichen

- **Beispiele:**

```
'abc' LIKE 'abc'    true
'abc' LIKE 'a%'    true
'abc' LIKE '_b_'   true
'abc' LIKE 'c'     false
```

- **Vorsicht mit führenden Wildcards:**

```
select distinct s.Name
from Vorlesungen as v, hören as h, Studenten as s
where s.MatrNr = h.MatrNr and
      h.VorlNr = v.VorlNr and
      v.Titel like '%thik%' // Kein Index nutzbar!
```

# Ausdrücke und Funktionen

---

- **Ausdrücke über Datum und Zeit**
  - Literale
    - **DATE** '*year-month-day*'
    - **TIME** '*hour:minute:seconds*'
    - **TIMESTAMP** '*year-month-date hour:minute:second*'
  - Konstanten für aktuelles Datum / aktuelle Zeit
    - **CURRENT\_DATE**
    - **CURRENT\_TIME**
    - **CURRENT\_TIMESTAMP**
  - Operatoren + und -

# Ausdrücke und Funktionen

---

- **Automatische Konvertierung**
  - innerhalb der Zeichentypen, der exakten und der annähernd numerischen Typen
- **`cast(x as type)`** konvertiert Wert **x** in Typ **type**

		nach													
		EN	AN	VC	FC	VB	FB	D	T	TS	YM	DT	BO	CL	BL
von	EN	■	■	■	■								■		
	AN	■											■		
	C	■						■					■		
	B						■						■		
	D							■		■			■		
	T								■	■			■		
	TS							■	■	■			■		
	YM									■			■		
	DT										■		■		
	BO											■	■		
	BL	■											■		

**EN** exakt numerisch

**AN** annähernd numerisch

**VC** varchar

**FC** char

**VB** bits varying

**FB** bits

**D** Date

**T** time

**TS** timestamp

**YM** year-month interval

**DT** day-time interval

**BO** boolean

**CL** clob

**BL** blob

**C** character (fixed, variable)

**B** bit string (fixed, variable)

# Ausdrücke und Funktionen

- bei Vergleichen und boole'schen Operationen wird **null** wie ein undefinierter Wert behandelt:

=	0	1	null
0	true	false	null
1	false	true	null
null	null	null	null

^	0	1	null
0	0	0	null
1	0	1	null
null	null	null	null

- **Konsequenz:**
  - Test auf null immer mit **is null** bzw. **is not null!**

## Ausdrücke und Funktionen

---

- **COALESCE( $v_1, \dots, v_n$ )**  
gibt den ersten Wert ungleich NULL zurück
- **NULLIF( $a, b$ )**  
gibt NULL zurück, falls  $a$  und  $b$  gleich sind

# Gruppierungen / Aggregatfunktionen

- **Aggregatfunktionen fassen alle Werte einer Spalte zusammen**
  - **min([ distinct ] A )**  
Berechnung des Minimalwerts der Spalte **A** (optionales Schlüsselwort **distinct** hier bedeutungslos)
  - **max([ distinct ] A )**  
Berechnung des Maximalwerts der Spalte **A** (optionales Schlüsselwort **distinct** hier bedeutungslos)
  - **avg([ distinct ] A )**  
Berechnung des Durchschnittswerts der Spalte **A** (mit **distinct** gehen gleiche Werte nur einmal in die Berechnung ein)
  - **sum([ distinct ] A )**  
Berechnung der Summe aller Werte in Spalte **A** (mit **distinct** gehen gleiche Werte nur einmal in die Berechnung ein)
  - **count(\*)**  
Zählen der Tupel der betrachteten Relation
  - **count([ distinct ] A )**  
Zählen der Tupel der betrachteten Relation, bei **distinct** nach Duplikateliminierung bezüglich Spalte **A**

# Gruppierungen / Aggregatfunktionen

- **Lediglich count(\*) berücksichtigt NULL-Werte**
- **Vor Anwendung einer Aggregatfunktion werden NULL-Werte eliminiert**
  - ggf. soll eine Warnung ausgegeben werden
- **Sollen NULL-Werte berücksichtigt werden, sollte mit COALESCE gearbeitet werden**

## B Aggregatfunktionen und NULL

- Gegeben sei die Tabelle **testAgg**

```
select avg(A),avg(B),count(*),count(A),count(B)  
from testAgg;
```



avg(A)	avg(B)	count(*)	count(A)	count(B)
100	133.333	4	4	3

testAgg	
A	B
150	150
0	null
200	200
50	50

# Gruppierungen / Aggregatfunktionen

---

- **Aggregatfunktionen fassen alle Zeilen einer Tabelle zusammen**
- **Häufig sinnvoll: Aggregate über Teilmengen einer Relation berechnen**
  - => alle Tupel mit gleichen Attributwerten “landen im selben Topf”.
  - => Anwendung der Aggregatfunktion pro Topf
  - Beispiel:
    - Es soll gezählt werden, wie viele Studierende sich in jedem Semester befinden
- **Es wird ein Konstrukt benötigt, das**
  - Teilmengen einer (möglicherweise berechneten) Relation zu Gruppen zusammenfasst
  - weitere Operationen auf diesen Gruppen ausführt
  - entstandene Guppen ggf. filtert

# Gruppierungen / Aggregatfunktionen

- **Syntax:**

```
select A1,..,An
from R1,...,Rk
where Pw
group by B1,...,Bm
having Ph
```

- **Fasst Zeilen, die in Spalten B<sub>1</sub>,...,B<sub>m</sub> identische Werte haben, zu Gruppe zusammen**
  - B<sub>1</sub>,...,B<sub>m</sub> bestimmen, wie 'Töpfe' gebildet werden - Ergebnis der Anfrage sind 'Töpfe', also Zusammenfassungen von Zeilen
  - **Daher:** Spalten, die nicht zur Gruppierung beitragen, dürfen nur im Kontext von Aggregatfunktionen verwendet werden! Aggregatfunktion wirkt dann innerhalb einer Gruppe
- **Gruppen werden abschließend mit Prädikat P<sub>h</sub> gefiltert**
  - Beachte: Prädikat P<sub>w</sub> der where-Klausel wird vor Gruppierung angewendet!

# Gruppierungen / Aggregatfunktionen

## B GROUP BY / HAVING

- ohne Angabe von **group by** werden **alle** Zeilen **zusammengefasst**:

```
// Durchschnittliche Semesterzahl aller Studenten  
select avg(Semester) from Studenten
```

- Gruppierung der Vorlesungen nach Professoren, die sie lesen:

```
// Lehrleistung der Professoren  
select gelesenVon,sum(SWS) from Vorlesungen group by gelesenVon
```

- Anschließende Filterung - Professoren, die mindestens 8 SWS leisten

```
// Professoren mit Lehrleistung >= 8 SWS  
select name,sum(SWS) from vorlesungen, professoren  
where (PersNr=gelesenVon)  
group by name  
having sum(SWS)>=8
```

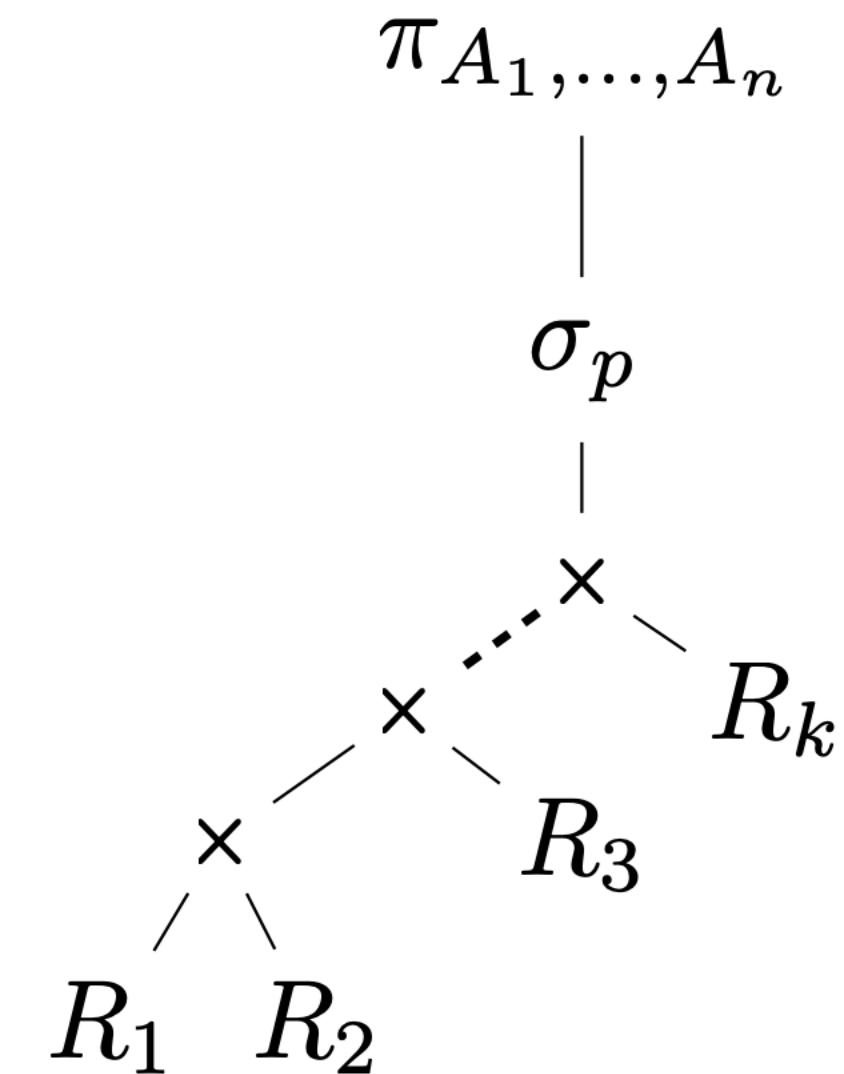
# Optimierung

- **SQL-Anfragen können in Ausdrücke der rel. Algebra übersetzt werden:**

- **select** → Projektion
- **where** → Selektion
- **Komma** → Kreuzprodukt

$$\pi_{A_1, \dots, A_n} (\sigma_P (R_1 \times \dots \times R_k))$$

**select distinct**  $A_1, \dots, A_n$   
**from**  $R_1, \dots, R_k$   
**where**  $P;$



# Optimierung

---

- **Abarbeitung der `from`-Klausel**

Kartesisches Produktes über  $R_1, \dots, R_k$ :

```
Table cross_product(Table R1, Table R2)
  Table res(R1.cols concat R2.cols);
  foreach row1 in R1 // O(|R1| * |R2| )
    foreach row2 in R2
      res.appendRow( row1 concat row2 )
  return res;
```

```
select distinct A1,...,An
from R1,...,Rk
where P;
```

$$O\left(\prod_{i=1}^k |R_i|\right)$$

**Laufzeit und Platz!**

**Zeit- / platzkritische Operation**

- **Filterung mit Prädikat in `where`-Klausel (optional):**

```
Table filter(Table R, Predicate P)
  Table res(R.cols);
  foreach row in R // O(|R| )
    if ( P(row) )
      res.appendRow( row )
  return res;
```

$$O\left(\prod_{i=1}^k |R_i|\right)$$

**Best case:**  $O\left(\log\left(\prod_{i=1}^k |R_i|\right)\right)$   
(binäre Suche mit Index)

- **Projektion auf Spalten der `select`-Klausel (optional):**

```
Table project(Table R, Columns C)
  Table res(C);
  foreach row in R // O(|R| )
    res.appendRow( πC(row) )
  return res;
```

$$O\left(\prod_{i=1}^k |R_i|\right)$$

# Umbenennung

---

Studenten		
MatrNr	Name	Semester
...	...	...

hören	
MatrNr	VorlNr
...	...

Vorlesungen			
VorlNr	Titel	SWS	gelesenVon
...	...	...	...

- **Erhöht die Lesbarkeit**

- `select Name, Titel // Welche Studenten hören welche Vorlesungen?  
from Studenten, hören, Vorlesungen  
where Studenten.MatrNr = hören.MatrNr and  
hören.VorlNr = Vorlesungen.VorlNr;`
  - ohne **Kontext** ist unklar, welcher Tabelle die Spalten **Name** und **Titel** zuzuordnen sind.
- `select s.Name, v.Titel  
from Studenten as s, hören as h, Vorlesungen as v  
where s. MatrNr = h. MatrNr and  
h.VorlNr = v.VorlNr`
  - s,h und v entsprechen den Tupelvariablen
  - **Beachte:** implizit verwendet SQL den Namen einer Relation als Tupelvariable (s.o. - hören.VorlNr)

- **Ist bei Doppeldeutigkeiten evtl. notwendig!**

# Umbenennung

- **Welche Studenten kennen sich aus Vorlesungen?**
  - geben Sie alle Paare aus!

```
select s1.Name, s2.Name
from Studenten as s1,
     hoeren as h1, hoeren as h2,
     Studenten as s2
where h1.VorlNr = h2.VorlNr and
      h1.MatrNr = s1.MatrNr and
      h2.MatrNr = s2.MatrNr
```

26120 (Fichte),  
 27550 (Schopenhauer) und  
 29120 (Theophrastos) kennen sich

<b>Studenten</b>		
<b>MatrNr</b>	<b>Name</b>	<b>Semester</b>
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

<b>hören</b>	
<b>MatrNr</b>	<b>VorlNr</b>
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022

# Mengenoperationen

- **Kombination von zwei SFW-Abfragen**
- **SQL unterstützt Mengenoperationen**
  - Vereinigung:
    - query<sub>1</sub> **UNION [ALL]** query<sub>2</sub>
  - Differenz:
    - query<sub>1</sub> **EXCEPT [ALL]** query<sub>2</sub>
  - Durchschnitt:
    - query<sub>1</sub> **INTERSECT [ALL]** query<sub>2</sub>
- **Schachtelungen und Verkettungen sind möglich**
  - z.B. query<sub>1</sub> UNION query<sub>2</sub> UNION (query<sub>3</sub> INTERSECT query<sub>4</sub>)
- **Durch den Zusatz **ALL** werden Duplikate **nicht** eliminiert**
  - MySQL: **EXCEPT** und **INTERSECT** nicht unterstützt

// Alle Studenten, die keine Hiwis sind:  
(select Name from Studenten)  
**except**  
(select Name from Hiwis)

## Mengenoperationen

---

- INTERSECT und EXCEPT können leicht simuliert werden:

// Alle Studenten, die gleichzeitig Hiwis sind:

(select Name from Studenten)

**intersect**

(select Name from Hiwis)

select Name **from** Studenten **join** Hiwis  
**using** Name

// Alle Studenten, die keine Hiwis sind:

(select Name from Studenten)

**except**

(select Name from Hiwis)

select Name **from** Studenten **left join** Hiwis  
**using** Name  
**where** Hiwis.Name **IS NULL**

## Unterabfragen / Subselect

---

- **Anfragen nicht nur über Mengenoperation kombinierbar**
- **Unterabfragen**
  - in **select**-Klausel
  - in **from**-Klausel
  - in **where**-Klausel
- **Operationen auf Unterabfragen die Mengen zurückliefern**
  - **Existenzquantor:** exists / not exists
  - **Inklusion:** in / not in
  - **Vergleich:** all / any / some

## Unterabfragen / Subselect

- Unteranfrage in der select-Klausel
- Für jedes Ergebnistupel wird die Unteranfrage ggf. neu ausgeführt
- Beispiel:

```
// Lehrleistung der Professoren
select PersNr, Name, ( select sum (SWS)
    from Vorlesungen
    where gelesenVon=PersNr ) as Lehrbelastung
from Professoren;
```

**Korrelation (später)**

## Unterabfragen / Subselect

---

- Unterabfrage kann einen Wert oder eine Liste liefern
- Sofern sie **einen Wert** liefert, kann Sie im where-Ausdruck wie eine Konstante eingesetzt werden:

```
// Prüfungen, die über dem Durchschnitt liegen
select * from prüfen
where Note < ( select avg (Note) from prüfen )
```

# Unterabfragen / Subselect

- Verwertung der Ergebnismenge einer Unterabfrage

```
// Studenten, die mehr als zwei Vorlesungen hören
select tmp.MatrNr, tmp.Name, tmp.VorlAnzahl
from (select s.MatrNr, s.Name, count(*) as VorlAnzahl
      from Studenten as s, hören as h
     where s.MatrNr=h.MatrNr
   group by s.MatrNr, s.Name) as tmp
  where tmp.VorlAnzahl > 2
```

Wie viele  
Vorlesungen hört ein  
Student?



MatrNr	Name	VorlAnzahl
27550	Schopenhauer	2
25403	Jonas	1
28106	Carnap	4
29120	Theophrastos	3
26120	Fichte	1
29555	Feuerbach	2

# Unterabfragen / Subselect

## B "Marktanteil" der Vorlesungen 1 -> funktionale Dekomposition

- Anzahl der Studenten

```
select count(*) as GesamtAnz from Studenten
```

1

- Anzahl der Hörer pro Vorlesung

```
select VorlNr, count(*) as AnzProVorl from hören  
group by VorlNr
```

2

- Beides durch Kreuzprodukt auf eine Zeile bringen

```
select *  
from (select count(*) as GesamtAnz from Studenten) as g ,  
(select VorlNr, count(*) as AnzProVorl from hören  
group by VorlNr) as h
```

3

- Ergebnis in select berechnen

```
select h.VorlNr, cast(h.AnzProVorl as decimal(6,2)) / g.GesamtAnz as Marktanteil  
from (select count(*) as GesamtAnz from Studenten) as g ,  
(select VorlNr, count(*) as AnzProVorl from hören  
group by VorlNr) as h
```

4

## WITH

### B "Marktanteil" der Vorlesungen 2 -> Mehr Übersicht mit "with"

#### Marktanteil der Vorlesung - Zerlegung in Teilaufgaben / Lösungen dazu

- Anzahl der Studenten

```
select count(*) as GesamtAnz from Studenten
```

- Anzahl der Hörer pro Vorlesung

```
select VorlNr, count(*) as AnzProVorl from hören  
group by VorlNr
```

#### Teillösungen in "with"-Block; Synthese der Gesamtlösung:

```
with SummeStudenten as (select count(*) as GesamtAnz from Studenten),  
HörerProVorlesung as (select VorlNr, count(*) as AnzProVorl from hören  
group by VorlNr)
```

```
select VorlNr, cast( AnzProVorl as decimal(5,2)) / GesamtAnz  
from SummeStudenten,HörerProVorlesung
```

Unterabfragen in "with"-Block werden einmal ausgeführt

# WITH

- In with eingeführte Bezeichner können um Schema-Definition erweitert werden
  - Dadurch liest sich eine Tabellendefinition fast wie eine Funktionsdefinition

```
with SummeStudenten as (select count(*) as GesamtAnz from Studenten),
      HörerProVorlesung as (select VorlNr, count(*) as AnzProVorl from hören
                            group by VorlNr)

select VorlNr, cast( AnzProVorl as decimal(5,2)) / GesamtAnz
from SummeStudenten,HörerProVorlesung
```

Typen ergeben sich aus  
Ergebnisspalten des zugehörigen  
Select!



```
with SummeStudenten(GesamtAnz)
      HörerProVorlesung(VorlNr,AnzProVorl) as (select count(*) from Studenten),
                                      as (select VorlNr, count(*) from hören
                                          group by VorlNr)

select VorlNr, cast( AnzProVorl as decimal(5,2)) / GesamtAnz
from SummeStudenten,HörerProVorlesung
```

# WITH

- **Bekanntheitsgrad eines Professors**
  - Studenten kennen einen Professor, wenn Sie eine seiner Vorlesungen besuchen

## B Bekanntheitsgrad

**with**

```
/* Studenten kennen Professor aus der Vorlesung.  
Beachte: distinct, da Student mehrere Vorlesungen eines Professors besuchen kann! */  
kennenSich(ProfessorId,MatrNr) as (select distinct gelesenVon, MatrNr  
from hören natural join Vorlesungen),  
/* Anzahl der unterschiedlichen Studenten, die einen Professor kennen  
Beachte: bereits eingeführter Bezeichner in folgenden with-Definitionen erlaubt */  
anzStudenten(ProfessorId,Anzahl) as (select ProfessorId, count(*)  
from kennenSich group by ProfessorId),  
/* Gesamtzahl der Studenten */  
gesamtStudenten(Gesamt) as (select count(*) from Studenten)  
  
select name, Anzahl *1.0 / Gesamt as Bekanntheitsgrad  
from anzStudenten, gesamtStudenten, Professoren  
where ProfessorId = Professoren.PersNr  
order by Bekanntheitsgrad desc
```

## WITH

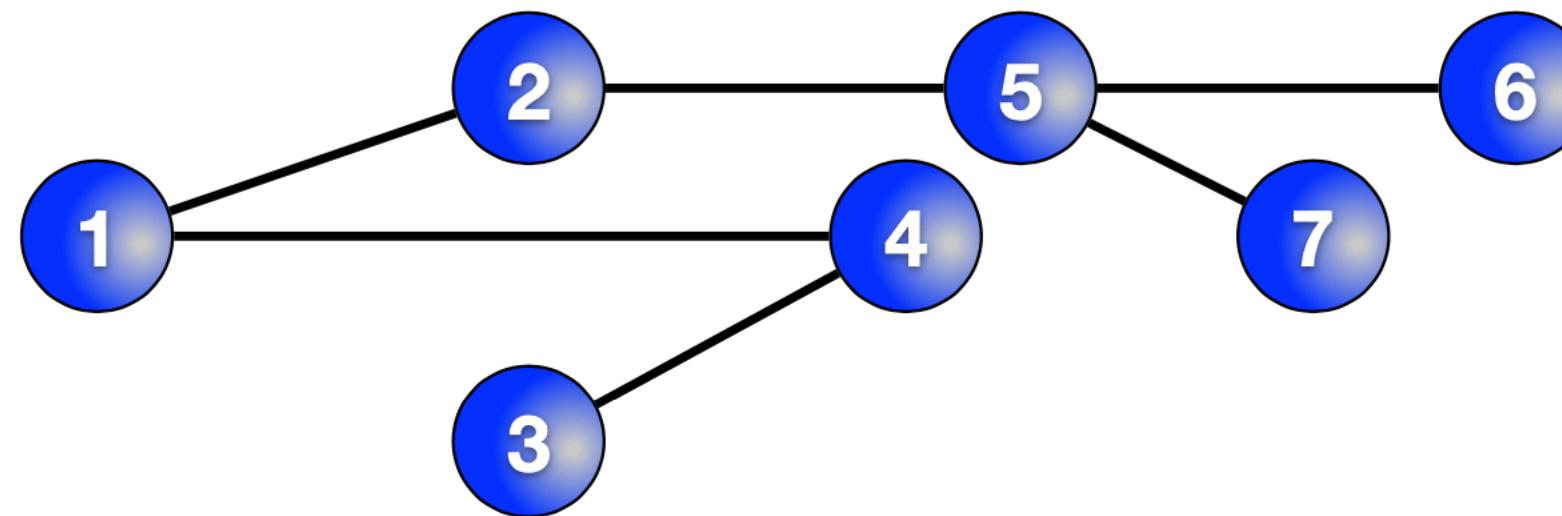
---

- **with erlaubt rekursive Definitionen**
- **Syntax:**  
**WITH RECURSIVE *non\_recursive\_term* UNION [ ALL | DISTINCT ]  
  *recursive\_term***
- **rekursiver Aufruf nur in *recursive\_term* erlaubt**
  - Konsequenz: durch den Zusatz "RECURSIVE" kann auf with-Bezeichner zugegriffen werden, die später im with-Block definiert werden
- **nur ein rekursiver Aufruf ist erlaubt!**
  - Fixpunktsemantik der Rekursion

## WITH

- Transitive Hülle eines azyklischen Graphen

- Modellieren des Graphen durch Relation "Kanten"



Kanten	
V	N
1	2
2	5
1	4
4	3
5	6
5	7

- Zwei Knoten A und B sind miteinander verbunden, wenn
  - eine Kante sie direkt verbindet, also Kanten(A,B) existiert
  - es einen Zwischenknoten Z gibt, so dass A mit Z verbunden ist **und** eine Kante von Z nach B existiert

Verbunden(A,B) :- Kanten(A,B)

Verbunden(A,B) :- Verbunden(A,Z) , Kante(Z,B)

## WITH

---

`Verbunden(A,B) :- Kanten(A,B)`

`Verbunden(A,B) :- Verbunden(A,Z) , Kante(Z,B)`

- **Umsetzung in SQL:**

```
with recursive Verbunden(A,B) as ( (select V,N from Kanten)
    union all
    (with V(A,Z) as (select * from Verbunden),
     K(Z,B) as (select * from Kanten)
      select A, B from V, K
      where V.Z = K.Z
    )
  )
select * from Verbunden;
```

# WITH

```
with recursive Verbunden(A,B) as ( (select V,N from Kanten)
                                    union all
                                    (with V(A,Z) as (select * from Verbunden),
                                     K(Z,B) as (select * from Kanten)
                                     select A, B from V, K
                                     where V.Z = K.Z
                                    )
                                    )
select * from Verbunden;
```

- Operationelle Abarbeitung (vereinfacht):

```
working_table = execute non_recursive_query
while (! working_table.empty) {
    tmp_table = execute recursive_query substitute Verbunden ↪ working_table
    working_table = tmp_table
}
```

- Konsequenz: terminiert nicht für zyklische Graphen

## IN / ALL / ANY / EXISTS

---

- **Können in where-Klausel weiter bearbeitet werden**
  - Existenzquantor: exists / not exists
  - Inklusion: in / not in
  - Vergleich: all / any / some
- **Führen oft zu korrelierten Unterabfragen ...**

# IN / ALL / ANY / EXISTS

- **Syntax:**  
**exists( subquery )**
- **unärer Operator**
  - liefert true, falls *subquery* nicht die leere Tabelle liefert
  - *subquery* darf Variablen der umschließenden Anfrage referenzieren
  - dann spricht man von einer **korrelierten Unterabfrage**

```
// Professoren, die Vorlesungen halten  
select p.Name  
from Professoren p  
where exists ( select *  
               from Vorlesungen as v  
               where v.gelesenVon = p.PersNr )
```

**Korrelation**

Korrelierte Unterabfrage muss

- für jeden Eintrag neu ausgeführt werden  
 $O(|\text{Professoren}| * |\text{Vorlesungen}|)$
- nicht vollständig ausgewertet werden  
**select-Klausel bedeutungslos**

Wir werden diese Anfrage gleich optimieren ....

# IN / ALL / ANY / EXISTS

- **Syntax:**

***expression in ( subquery )***

- **binärer Operator**

- prüft, ob Ergebnis von *expression* in *subquery* vorkommt
- Voraussetzung: *subquery* liefert exakt eine Spalte
- **Vorsicht!** Ergebnis ist **null**, falls
  - *expression null* liefert, oder Ergebnis der *subquery null* enthält, denn:

$$a \text{ in } \{v_1, \dots, v_k\} := (a=v_1) \vee \dots \vee (a=v_k)$$

```
// Professoren, die Vorlesungen halten (nicht korreliert!)
```

```
select Name  
from Professoren  
where PersNr in ( select gelesenVon  
                  from Vorlesungen  
                )
```

- nicht korrelierte Unterabfrage wird einmal ausgewertet  
evtl. nicht vollständig
- Inklusionstest (bei vorhandenem Index) in konstanter Zeit!  
 $O(|\text{Professoren}|)$

# IN / ALL / ANY / EXISTS

- **Korrelierte Formulierung**

```
select *  
from Studenten as S  
where exists ( select *  
               from Professoren as P  
               where P.GebDatum > S.GebDatum )  
// Studenten, die älter sind als mind. einer der Professoren
```

$$O(|S| * |P|)$$

- **Schlecht:** Student wird mit jedem Professor verglichen
- **Besser:** Finde zuerst den jüngsten Professor - Elimination der Korrelation

```
select *  
from Studenten as S  
where S.GebDatum < ( select max(GebDatum)  
                      from Professoren )
```

$$O(|S|)$$

## IN / ALL / ANY / EXISTS

---

- Manchmal ist die Beseitigung einer Korrelation nicht ganz so einfach

```
select *  
from Assistenten as A  
where exists ( select *  
    from Professoren as P  
    where A.Chef = P.PersNr and P.GebDatum > A.GebDatum )  
  
// Assistenten, die älter als Ihre Chefs sind
```

- max hilft hier nicht, aber wir könnten einen join versuchen:

```
select *  
from (Assistenten as A join Professoren as p on A.Chef=P.PersNr)  
where P.GebDatum > A.GebDatum
```

- relationale Datenbanken sind für joins hochoptimiert
- join mit Hashtabellen in linearer Zeit möglich

# IN / ALL / ANY / EXISTS

---

- **Syntax:**

*expression operator any( subquery )*

- **ternärer Operator**

- prüft, ob *subquery* einen Wert *value* enthält, so dass *expression operator value true* ergibt
- *subquery* muss Tabelle mit exakt einer Spalte zum Ergebnis haben
- *operator* muss im Ergebnis einen Wahrheitswert liefern (z.B. =, >=, <>)
- *any* und *some* sind synonym
- 'in' ist äquivalent zu '= any'
- **Vorsicht!** Ergebnis ist **null**, falls *expression* null oder Ergebnis von *subquery* null enthält, denn

$$a \text{ op any } \{v_1, \dots, v_k\} := (a \text{ op } v_1) \vee \dots \vee (a \text{ op } v_k)$$

# IN / ALL / ANY / EXISTS

---

- **Syntax:**

***expression op all( subquery )***

- **ternärer Operator**

- prüft, ob *subquery* nur Werte *value* enthält, so dass *expression operator value true* ergibt
- *subquery* muss Tabelle mit exakt einer Spalte zum Ergebnis haben
- *operator* muss im Ergebnis einen Wahrheitswert liefern (z.B. =, >=, <>)
- 'not in' ist äquivalent zu '<> all'
- **Vorsicht!** Ergebnis ist **null**, falls *expression* null oder Ergebnis von *subquery* null enthält.

```
// dienstälteste Studenten
select Name
from Studenten
where Semester >= all ( select Semester
                           from Studenten
                         )
```

## IN / ALL / ANY / EXISTS

---

- **SQL-92 kennt keinen Allquantor**
- **Allquantifizierung muss also durch eine äquivalente Anfrage mit Existenzquantifizierung ausgedrückt werden**

$$\forall x.P(x) \Leftrightarrow \neg\exists x.\neg P(x)$$

# IN / ALL / ANY / EXISTS

## B Umformung im Tupelkalkül

- **Wer hat alle vierstündigen Vorlesungen gehört?**

$$\{s \mid s \in \text{Studenten} \wedge \forall v \in \text{Vorlesungen} : (v.\text{SWS} = 4 \Rightarrow \exists h \in \text{hören} : (h.\text{VorlNr} = v.\text{VorlNr} \wedge h.\text{MatrNr} = s.\text{MatrNr}))\}$$

- **Elimination  $\forall$**

$$\{s \mid s \in \text{Studenten} \wedge \neg(\exists v \in \text{Vorlesungen} : \neg(v.\text{SWS} = 4 \Rightarrow \exists h \in \text{hören} : (h.\text{VorlNr} = v.\text{VorlNr} \wedge h.\text{MatrNr} = s.\text{MatrNr}))))\}$$

- **Elimination  $\Rightarrow$  mit  $a \Rightarrow b = \neg a \vee b$**

$$\{s \mid s \in \text{Studenten} \wedge \neg(\exists v \in \text{Vorlesungen} : \neg(\neg(v.\text{SWS} = 4) \vee \exists h \in \text{hören} : (h.\text{VorlNr} = v.\text{VorlNr} \wedge h.\text{MatrNr} = s.\text{MatrNr}))))\}$$

- **Regel von DeMorgan  $\neg(a \vee b) = \neg a \wedge \neg b$**

$$\{s \mid s \in \text{Studenten} \wedge \neg(\exists v \in \text{Vorlesungen} : (v.\text{SWS} = 4 \wedge \neg(\exists h \in \text{hören} : (h.\text{VorlNr} = v.\text{VorlNr} \wedge h.\text{MatrNr} = s.\text{MatrNr}))))\}$$

# IN / ALL / ANY / EXISTS

## B Umsetzung in SQL

$$\{s \mid s \in \text{Studenten} \wedge \neg(\exists v \in \text{Vorlesungen} : (v.SWS = 4 \wedge \neg(\exists h \in \text{hören} : (h.VorlNr = v.VorlNr \wedge h.MatrNr = s.MatrNr))))\}$$

```
// Wer hat alle vierstündigen Vorlesungen gehört?  
select s.*  
from Studenten as s  
where not exists  
  (select *  
   from Vorlesungen as v  
   where v.SWS = 4 and  
         not exists (select * from hören as h  
                      where h.VorlNr = v.VorlNr and  
                      h.MatrNr=s.MatrNr  
    ))  
);
```

# IN / ALL / ANY / EXISTS

- Allquantifizierung kann immer auch durch eine count-Aggregation ausgedrückt werden
  - Idee: Zählen aller Elemente, die P(x) erfüllen

## B Allquantor und count

- Studenten, die mindestens 20% aller Vorlesungen besuchen

```
select MatrNr  
from hören  
group by MatrNr  
having count (*) >= 0.2*(select count (*) from Vorlesungen)
```

```
with /* Gesamtzahl aller Vorlesungen */  
    GesamtAnzahl(Gesamt) as (select count(*) from Vorlesungen),  
    /* Anzahl Vorlesungen, die ein Student hört (fast...) */  
    AnzahlProStudent(MatrNr,Anzahl) as (select MatrNr,count(*) from hören  
                                         group by MatrNr)  
select MatrNr,Name,Anzahl  
from (Studenten natural join AnzahlProStudent) cross join GesamtAnzahl  
where Anzahl >= 0.2*(select Gesamt from GesamtAnzahl)
```

# IN / ALL / ANY / EXISTS

- **Ermitteln Sie die Studenten, die weniger als 10% aus dem Vorlesungsangebot hören**
  - Denken Sie daran: es gibt auch faule Studenten oder solche die Urlaub machen

## B Allquantor und count

```
with GesamtAnzahl(Gesamt) as (select count(*) from Vorlesungen),
/* Studenten, die keine Vorlesung besuchen */
StudentenOhne(MatNr) as ((select MatrNr from Studenten)
except
(select MatrNr from hören)),
AnzahlProStudent(MatNr,Anzahl) as ( (select MatrNr,count(*) from hören
group by MatrNr)
union /* Hinzunahme der "Faulen ... */
(select MatrNr,0 from StudentenOhne))
select MatrNr,Name,Anzahl from (Studenten natural join AnzahlProStudent),GesamtAnzahl
where Anzahl <= 0.1*(select Gesamt from GesamtAnzahl)
```

# CASE

---

- **Fallunterscheidung von mit Case**

```
select MatrNr, ( case when Note < 1.5 then 'sehr gut'  
                      when Note < 2.5 then 'gut'  
                      when Note < 3.5 then 'befriedigend'  
                      when Note < 4.0 then 'ausreichend'  
                      else 'nicht bestanden'  
                  end )  
from prüfen
```

- **Achtung:** die **erste** passende when-Klausel wird ausgeführt
- möglichst so formulieren, dass Reihenfolge irrelevant (also nicht, wie oben!)

# UNIT 0X0B

# SQL II

## Hintergrund

---

### **Einordnung der Kapitel SQL I und II**

- Die praktischen Übungen zu SQL und etwas Hintergrund dazu gibt es regulär im SQL-Praktikum.
- Die Folien hier greifen einzelne Aspekte und Variationen etwas breiter auf, dennoch sind sie kein Nachschlagewerk. Dazu bemühe man die aktuelle Dokumentation der jeweiligen DBMS.

# Übersicht

---

## Themen

- Tabellen anlegen und löschen
- Bedingungen in Tabellen (Constraints)
- Domains
- Daten einfügen, modifizieren und löschen
- Sichten
- Generalisierung
- Datenintegrität
- Kaskadierende Operationen
- Check
- Assertions
- Stored Procedures
- Trigger
- Beispieldatenbank Kemper/Eickler

# Anlegen von Tabellen

- Umfangreiche Möglichkeiten, hier nur einzelne Beispiele.
- Bedingungen/Constraints können an Spalten oder an die Tabelle gestellt werden.

```
create table table_name [ if not exists ] (
    { column_name data_type {column_constraint}* }
    {, column_name data_type {column_constraint}* }*
);
```

## B Anlegen der Tabelle Professor

- Wir beschränken uns zunächst auf die korrekten Datentypen

```
create table Professoren (
    PersNr integer,
    Name varchar(50),
    Rang char(2),
    Raum integer
);
```

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

## Anlegen von Tabellen – Constraints

---

### Bedingungen an eine Spalte

- primary key  
Spalte ist Primärschlüssel, impliziert unique und not NULL.
- references ref\_table [ (ref\_column) ]  
Spalte ist Fremdschlüssel, verweist auf Spalte ref\_column in Tabelle ref\_table.
- not null  
NULL ist nicht erlaubt.
- unique  
Spaltenwert einmalig, NULL ist erlaubt.
- check (expression)  
Für einzufügende Werte muss expression true oder NULL liefern.
- default default\_expr  
Wird beim Einfügen einer neuen Zeile kein Wert für diese Spalte angegeben, so wird sie mit default\_expr vorbelegt.

# Anlegen von Tabellen – Beispiel

## B Eine verbesserte Professorentabelle

- **PersNr** ist der Primärschlüssel
- Für **Name** ist NULL nicht erlaubt
- Nur ein Professor pro **Raum**
- **Rang** ist C2,C3 oder C4 - default ist C2

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

```
create table Professoren (
    PersNr integer primary key,
    Name varchar(50) not null,
    Rang char(2) check( Rang in ['C2','C3','C4']) default 'C2',
    Raum integer unique
);
```

# Anlegen von Tabellen – Beispiel

## B Die Tabelle "Vorlesungen"

- **VorlNr** ist der Primärschlüssel
- Für **Titel** ist NULL nicht erlaubt
- **SWS** liegt zwischen 1 und 8
- **gelesen\_von** ist Fremdschlüssel in Professoren-Tabelle

Vorlesungen			
VorlNr	Titel	SWS	gelesen_von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

```
create table Vorlesungen (
    VorlNr integer primary key,
    Titel varchar(50) not null,
    SWS integer check( SWS between 1 and 8 ),
    gelesen_von integer references Professoren // (PersNr) optional, da Primärschlüssel
);
```

## Anlegen von Tabellen - Table Constraints

---

### Motivation

- Zusammengesetzte Schlüssel definieren.
- check Constraint über mehrere Spalten.
- unique für Tupel.

- **Syntax**

```
create table table_name [ if not exists ] (
    { column_name data_type {column_constraint}* }
    {, column_name data_type {column_constraint}* }*
    {table_constraint}
);
```

## Anlegen von Tabellen – Beispiel Table Constraints

### B Die Tabelle "Vorlesungen" mit *table constraints*

```
create table Vorlesungen (
    VorlNr integer primary key,
    Titel varchar(50) not null,
    SWS integer check( SWS between 1 and 8 ),
    gelesen_von integer references Professoren // (PersNr) optional, da Primärschlüssel
);
```

```
create table Vorlesungen (
    VorlNr integer,
    Titel varchar(50),
    SWS integer,
    gelesen_von integer,
    // table constraints:
    primary key( VorlNr ),
    foreign key( gelesen_von ) references Professoren,
    check( SWS between 1 and 8 )
);
```

# Anlegen von Tabellen – Beispiel Table Constraints

## B Die Tabelle "voraussetzen"

- **(Vorgänger,Nachfolger)** ist zusammengesetzter Primärschlüssel
- **Vorgänger** und **Nachfolger** sind Fremdschlüsselelemente mit Verweis in die Tabelle Vorlesungen

voraussetzen	
<u>Vorgänger</u>	<u>Nachfolger</u>
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

```
create table voraussetzen (
    Vorgänger integer references Vorlesungen, // column constraint
    Nachfolger integer references Vorlesungen, // column constraint
    primary key (Vorgänger, Nachfolger) // table constraint
);
```

## Löschen von Tabellen

---

- **Syntax zum Löschen**

`drop table table_name;`

PostgreSQL: `drop table [if exists] table_name;`

Löscht eine Tabelle und alle zugehörigen Indizes

- **Syntax zum Leeren - Tabelle selbst bleibt erhalten**

`delete from table_name;`

PostgreSQL: `truncate table_name;` als schnellere Alternative

## Domain

---

- Eine Domain beschreibt die Einschränkung eines Basistyps durch eine oder mehrere Bedingungen.
- Kann z.B. bei `create table` überall dort eingesetzt werden, wo Typ erwartet wird. Aber: definiert keinen echten neuen Typ, sondern entspricht vom Typ her immer dem Basistyp `data_type`.

- **Syntax:**

```
create domain domain_name as data_type  
[ default expression ]  
[{not null | null | check( expression ) }]
```

Wert in `expression` wird mit **value** angesprochen

# Beispiel Domain

## B Professorentabelle mit eigener domain

- **PersNr** ist der Primärschlüssel
- Für **Name** ist NULL nicht erlaubt
- Nur ein Professor pro **Raum**
- **Rang** ist C2,C3 oder C4 - default ist C2

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

```
create domain ProfRang as char(2)
default 'C2'
check( value in ['C2','C3','C4'] )

create table Professoren (
    PersNr integer primary key,
    Name varchar(50) not null,
    Rang ProfRang,
    Raum integer unique
);
```

# Einfügen von Daten

- **Einfügen einer Zeile:**
  - `insert into Studenten (MatrNr, Name)  
values (28121, 'Archimedes');`
  - nicht angegebene Spalten werden mit *default*-Wert belegt (siehe **create table**, **create domain**), falls für zugehörigen Typ keiner bekannt mit NULL.
  - `insert into Studenten default values; // Neuer Student aus Vorgabewerten`
- **Einfügen mehrerer Zeilen**
  - `insert into Studenten // ohne Spaltenangabe: Werte für alle Spalten erforderlich  
values (28121, 'Archimedes', 12), (25403,'Jonas',18), ...;`
- **Einfügen einer Tabelle:**
  - `insert into hören  
select MatrNr, VorlNr  
from Studenten, Vorlesungen  
where Titel= 'Logik' ;`

# Löschen und Ändern von Daten

---

- **Löschen**
  - **delete from Studenten where MatrNr = 197403** // Löschen eines Studenten
  - **delete from Studenten where Semester > 13;** // Löschen aller Studenten ab dem 14. Semester
  
- **Ändern**
  - **update Studenten set Nachname='von Neumann' where MatrNr=196704** // Student mit MatrNr.=196704 erhält neuen Namen
  - **update Studenten set Semester= Semester + 1;** // Alle Studenten erreichen nächstes Semester

## Sichten

---

### Idee

- Abstrahieren eine bestimmte 'Sicht' aus einer Rolle oder einem Anwendungsfall, d.h. in der Regel Ausschnitt oder bestimmte Aggregation aus Daten bzw. Modell.
- Nicht einfach zu entscheiden ist, ob Sicht besser in externer Business Logik oder im DBMS aufgehoben ist.
- Realität ist noch komplexer, z.B. Elasticsearch mit Beats.
- Manche Projekte verlangen ausschliesslich Sichten.

## Sichten

---

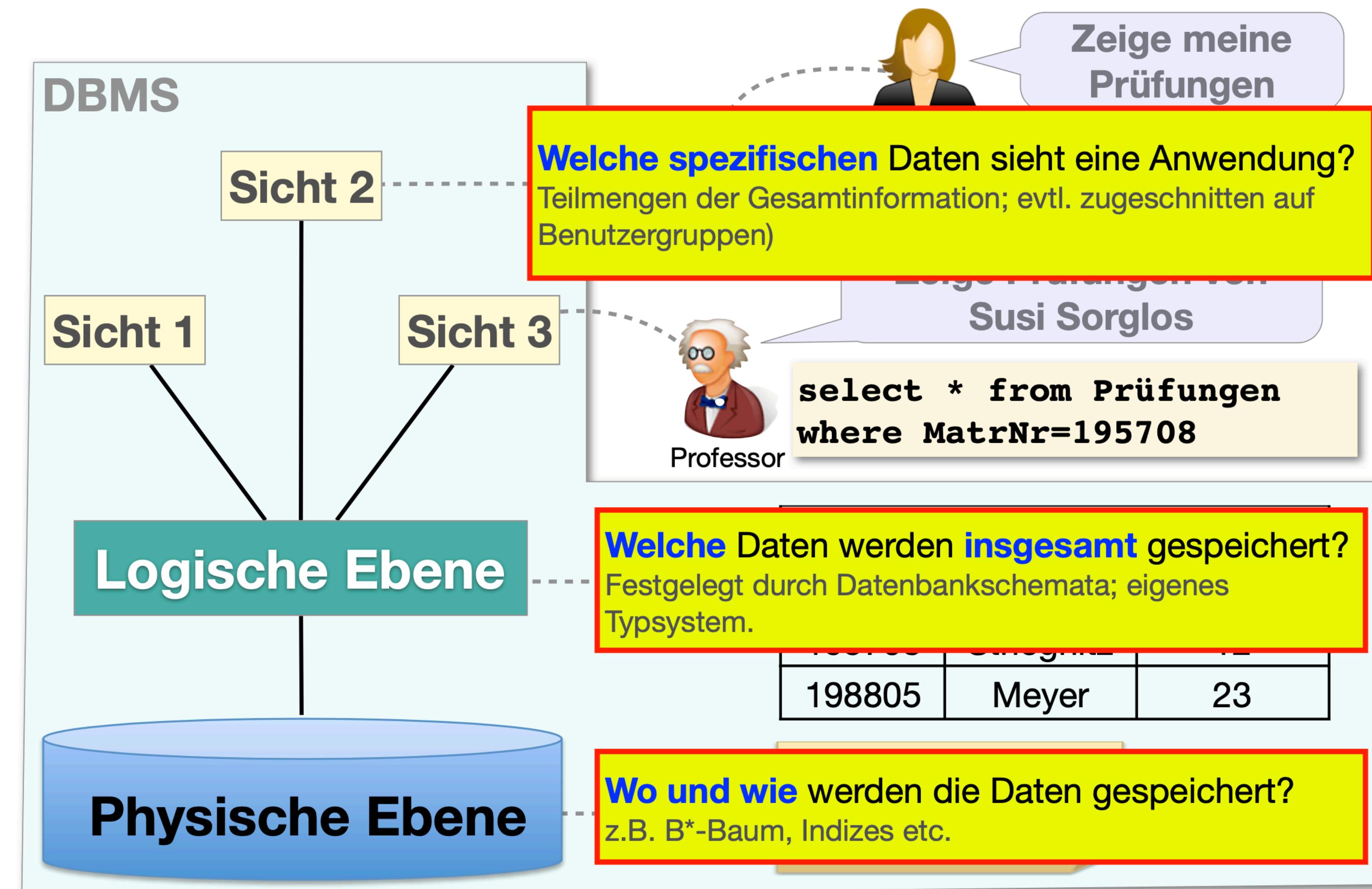
### Temporäre Sichten

- im Kontext von with-Definitionen
- überdauern die Anfrage nicht

### Statische Sichten

- Datenschutz
- Vereinfachung von Aufgaben
- Darstellung der Generalisierung

# Sichten und Abstraktionsebenen



## Sichten

---

### Idee

Oftmals benötigen Anwender speziell zugeschnittenen Ausschnitt der Daten.

### Externe Sichte

- sind Bestandteil der 3-Ebenen-Architektur.
- ermöglichen die Modellierung der Daten in einer für den Benutzer wünschenswerten Form.
- arbeiten auf den Daten der logischen Gesamtsicht. Somit sind entsprechende Transformationsregeln notwendig.

## Sichten - Beispiel

---

- **Datenbank für Kinobetriebe: häufig abgefragt werden die Filme des aktuellen Tages**
- **In Datenbank sind pro Kino alle Filme mit Angabe von Titel, Saal und Uhrzeit abgelegt**
- **Es kann sinnvoll sein, den Zugriff über eine spezielle Tabelle**

**PROGRAMM\_HEUTE (Name, Adresse, Telefon, Uhrzeit, Titel,  
Regisseur)**

**zu realisieren**

## Sichten - Beispiel

- **Möglichkeit:**
  - Deklaration einer **separaten Tabelle** und Befüllen mit der gewünschten Information:

```
create table PROGRAMM_HEUTE (
    Name  varchar(30) ,
    Adresse varchar(30),
    Telefon varchar(15),
    Uhrzeit timestamp,
    Titel  varchar(100) primary key,
    Saal   varchar(20)
)

insert into PROGRAMM_HEUTE
( select F.Name, F.Adresse, F.Telefon, F.Uhrzeit, F.Titel, F.Saal
  from  (PROGRAMM natural join KINO natural join FILM) as F
  where F.Datum = CURRENT_DATE )
)
```

- **Nachteile:** Redundante Datenhaltung, Tägliche Änderung der Tabelle

## Sichten - Beispiel

- **Sinnvolle Alternative:**
  - Vereinbarung der „Tabelle“ PROGRAMM\_HEUTE als **Sicht** (View)
- **Sicht entspricht Berechnung aus tatsächlich gespeicherten Relationen:**

```
create or replace view PROGRAMM_HEUTE as (
  select F.Name, F.Adresse, F.Telefon, F.Uhrzeit, F.Titel, F.Saal
  from   (PROGRAMM natural join KINO natural join FILM) as F
  where  F.Datum = CURRENT_DATE
)
```

## Sichten

---

### **Effekt einer Sichtenvereinbarung**

- DBMS speichert die Sichtdefinition, wertet sie aber nicht aus.
- Der Name der Sicht kann in Anfragen wie eine mit CREATE TABLE erzeugte Tabelle verwendet werden.
- Bei Zugriff auf die Sicht wird der Sichtename automatisch durch die mit der Sicht assoziierten Anfrage ersetzt. Die Sichtdefinition wirkt wie ein Makro

# Sichten - Beispiel

- **Anfrageauswertung für die Sicht PROGRAMM\_HEUTE:**

Anfrage eines Nutzers auf die Sicht PROGRAMM\_HEUTE: „Welcher Film läuft im Atrium-Saal?“:

```
select Name  
from PROGRAMM_HEUTE  
where Saal = ,Atrium'
```

Tatsächliche Anfrage nach Auswertung der Sichtendefinition:

```
select PROGRAMM_HEUTE.Name  
from ( select F.Name, F.Adresse, F.Telefon, F.Uhrzeit, F.Titel, F.Name  
      from (PROGRAMM natural join KINO natural join FILM) as F  
      where F.Datum = CURDATE()  
    ) as PROGRAMM_HEUTE  
where Saal = ,Atrium';
```

## Sichten - Beispiel

- **Beliebiges Anfrage zur Definition einer Sicht erlaubt**
  - Sichten können **mehrere Tabellen** verknüpfen
  - Sichten können Aggregatfunktionen nutzen
  - Sichten können auch Gruppierungen nutzen

```
create view PersonalkostenProStandort as
( select SUM(m.Gehalt) as Gehaltsbudget, m.Abteilungsnummer, a.Standort
  from Mitarbeiter as m natural join Abteilungen as a
  group by m.Abteilungsnummer, a.Standort
)
```

## Sichten

---

### Problem

- Änderungen müssen immer noch auf die Basisrelationen abgebildet werden. Dies ist je nach Sichtdefinition nicht immer möglich.

### Beispiel

- Änderung der Uhrzeit einer Vorstellung in PROGRAMM\_HEUTE ist unproblematisch: Änderung kann direkt in die benutzten Relationen eingebracht werden.
- Änderung des Datums eines Filmes in PROGRAMM\_HEUTE entfernt Film aus der Sicht.

### Allgemein

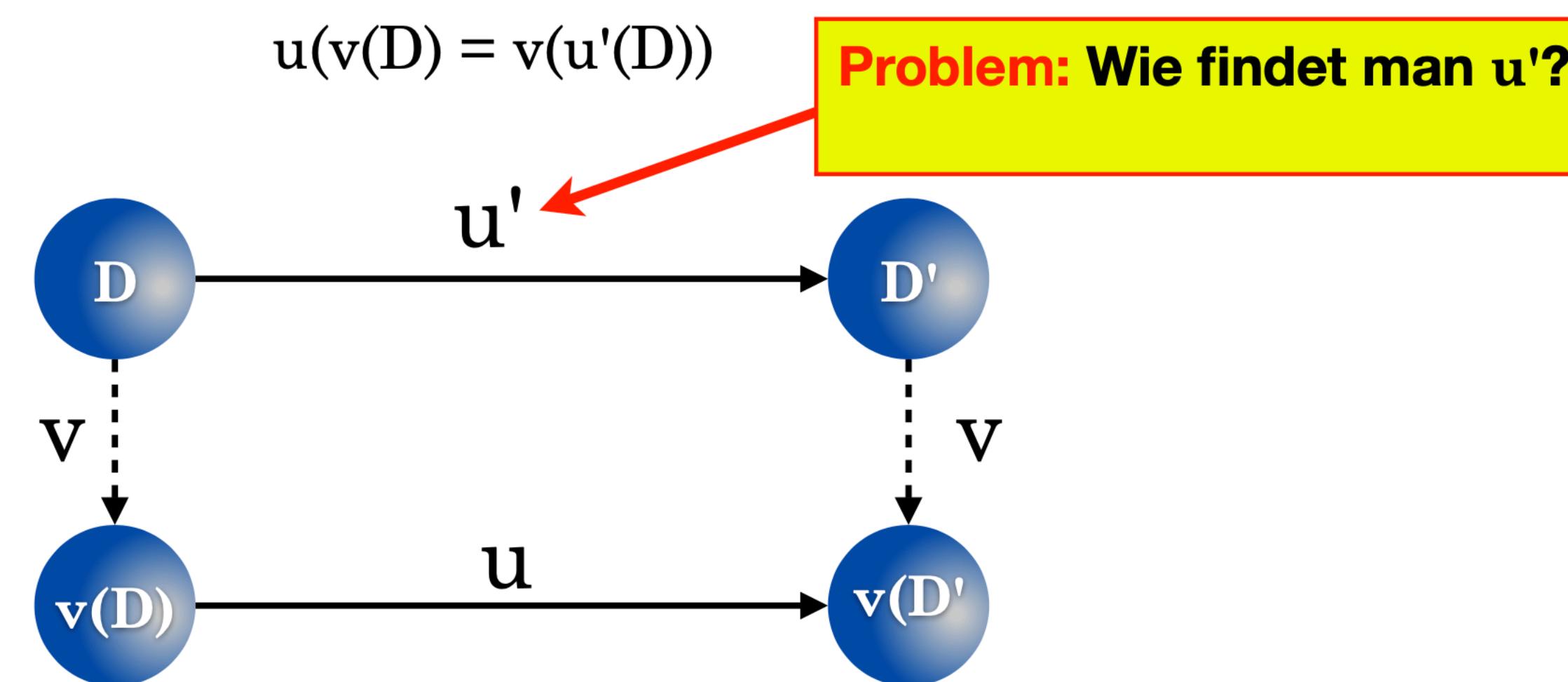
- Änderungen von Tupeln können ein „Löschen“ des Tupels aus der entsprechenden Sicht bewirken (Tupelmigration).

# Sichten

- Sei  $D$  die Menge aller Belegungen einer Datenbank, dann können Sichten und Änderungen als partielle Funktionen betrachtet werden:

$$v:D \rightarrow D \text{ bzw. } u:D \rightarrow D$$

- wobei  $v$  eigentlich keine Änderung vornimmt
- Eine Sicht  $v$  ist durch ein Update  $u$  änderbar, wenn es ein äquivalentes Update  $u'$  für den Ausgangsdatenbestand  $D \in D$  gibt, so dass



## Sichten

---

- **Die eben genannte Definition reicht zur Entscheidungsfindung nicht aus, denn es fehlt an Kenntnis über  $u'$** 
  - von DBMS ist  $u'$  nicht automatisch bestimmbar
- **Stattdessen: definiere syntaktische Einschränkungen für änderbare Sichten**
  - Syntax und statische Semantik von DBMS automatisch überprüfbar
  - Einschränkungen u.U. nicht vollständig

## Regeln für Sichten

---

- Die Sicht muss durch eine einzelne select-Anweisung definiert sein, d.h., kein join, union, etc.
- Die select-Klausel darf nur Attributnamen enthalten und jeden Namen nur einmal, keine Aggregatfunktionen, berechnete oder konstante Ausdrücke; ebenso kein distinct.
- Die from-Klausel darf nur einen einzigen Relationsnamen enthalten. Dieser muss eine Basisrelation oder eine änderbare Sicht bezeichnen.
- Falls die where-Klausel geschachtelte Anfragen beinhaltet, darf in deren from-Klauseln der Relationsname aus "from" nicht auftauchen, d.h. keine korrelierten Unterabfragen

## Beispiel Sichten

---

- **Gegeben seien die beiden Tabellen:**
  - MGA( Mitarbeiter, Gehalt, Abteilung)
  - AL( Abteilung, Leiter )
- **MGA speichert Daten über Zugehörigkeit von Mitarbeitern zu Abteilungen und deren jeweiliges Gehalt**
- **AL gibt für jede Abteilung den Abteilungsleiter an**

## Beispiel Sichten – Projektion

- **Wir definieren eine Sicht MA auf MGA, bei der das Gehalt ausgeblendet wird:**

```
create or replace view MA as (
    select Mitarbeiter, Abteilung
    from MGA
)
```

- **Diese Sicht gilt als änderbar, kann aber Probleme bereiten**
  - **insert into MA values ('Zuse','Informatik')** wird umgesetzt zu
  - **insert into MGA values ('Zuse',**null**, 'Informatik')** **null** evtl. in Schema ausgeschlossen
- **Beachte:**
  - implizite constraint-Verletzungen beim Einfügen Projektionssichten
  - auf default-Werte achten

## Beispiel Sichten – Selektion

- **Wir definieren eine Sicht Top von MGA, welche die Top-Verdiener enthält**

```
create view Top as (
    select Mitarbeiter, Gehalt
    from MGA
    where Gehalt > 20
)
```

- **Sicht gilt als änderbar, aber**
  - **update Top set Gehalt=15 where Mitarbeiter='Zuse'** bewegt Tupel aus der Sicht heraus und ist logisch somit fragwürdig
  - **check option:** Einfügungen und Änderungen an View müssen zu dessen Definition passen, ansonsten werden sie abgelehnt

```
create view Top as (
    select Mitarbeiter, Gehalt
    from MGA
    where Gehalt > 20
) with check option
```

## Beispiel Sichten – Verbund/Join

- **Wir definieren MGAL als Verbund:**

```
create or replace view MGAL as (
    select Mitarbeiter, Gehalt, MGA.Abteilung, Leiter
    from MGA natural join AL
)
```

- **View gilt als nicht änderbar!**
  - Grund: zwei Relationen in **from**-Klausel
- **Sinnvoll, da es sonst zu Inkonsistenzen kommen kann**
  - **insert into MGAL values('Müller', 3000, 001, 'Boss')** zieht nach sich
  - **insert into MGL values('Müller',3000, 001)**
  - Aber was soll in AL geschehen, wenn Tupel (001,'Boss') nicht existiert?
    - update, insert, Fehlermeldung?

## Beispiele

---

### Welche der folgenden Views sind änderbar?

**CREATE VIEW Aenderbar(...) AS ...**

**SELECT MAX (gehalt)**  
**FROM Personal**

**SELECT abtnr, budget \* 1,71**  
**FROM Projekt**

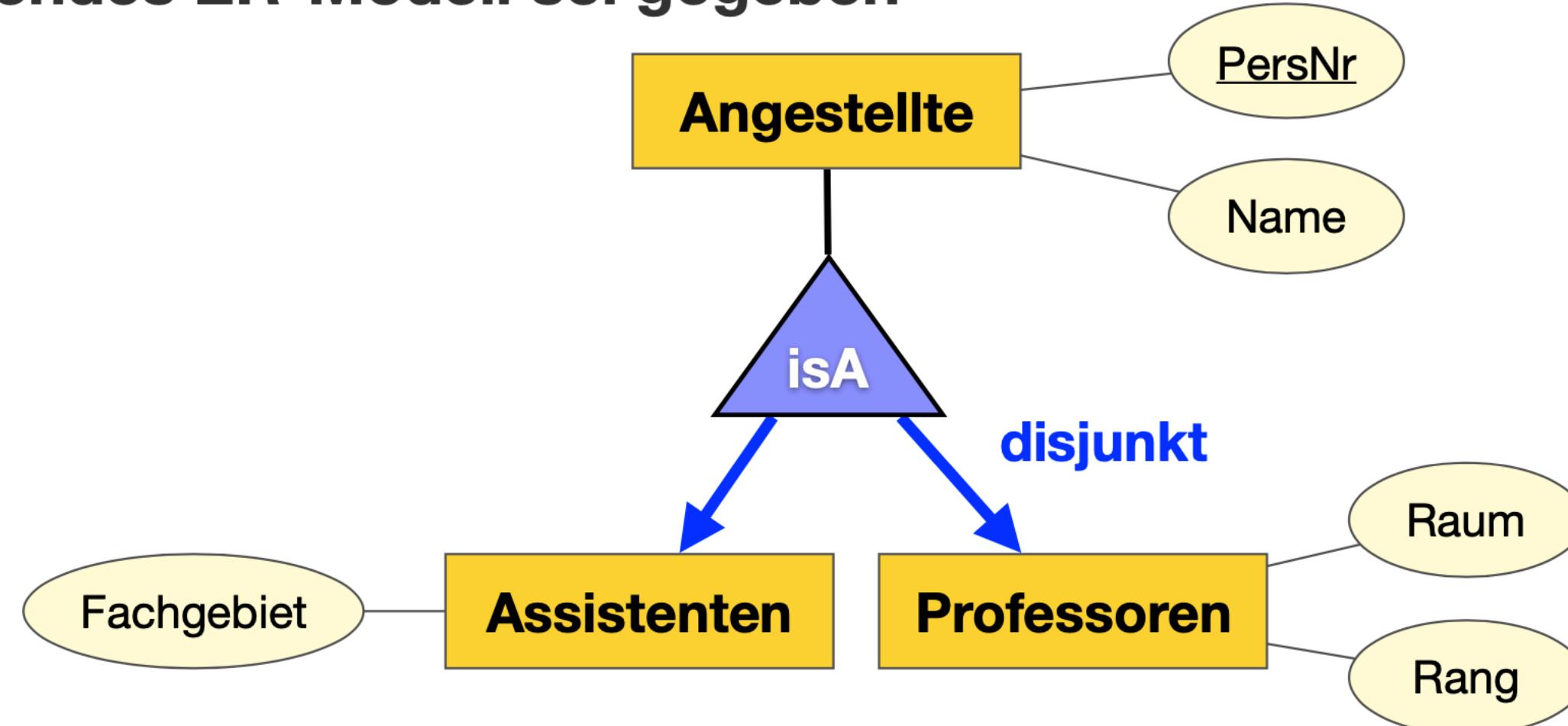
**SELECT persnr, name**  
**FROM Personal**  
**WHERE name LIKE 'M%'**

**SELECT Personal.name, Projekt.name **FROM** Personal **JOIN** Projekt**  
**USING (projnr)**

**SELECT gehalt**  
**FROM Personal**  
**WHERE gehalt > 5000**

# Views und Generalisierung – Variante 1

- Folgendes ER-Modell sei gegeben



- **Variante 1: Sichten für Unterklassen**
- **geerbte Attribute werden nicht repliziert:**
  - Angestellte: {[PersNr, Name]}
  - Professoren: {[PersNr, Rang, Raum]}
  - Assistenten: {[PersNr, Fachgebiet]}

# Views und Generalisierung - SQL 1

- Nur Oberklasse als echte Tabelle, Unterklassen als Sichten

```
create table Angestellte (
    PersNr integer not null,
    Name varchar (30) not null );
```

// Δ Professoren - Angestellte

```
create table ProfDaten (
    PersNr integer not null,
    Rang character(2),
    Raum integer );
```

// Δ Assistenten - Angestellte

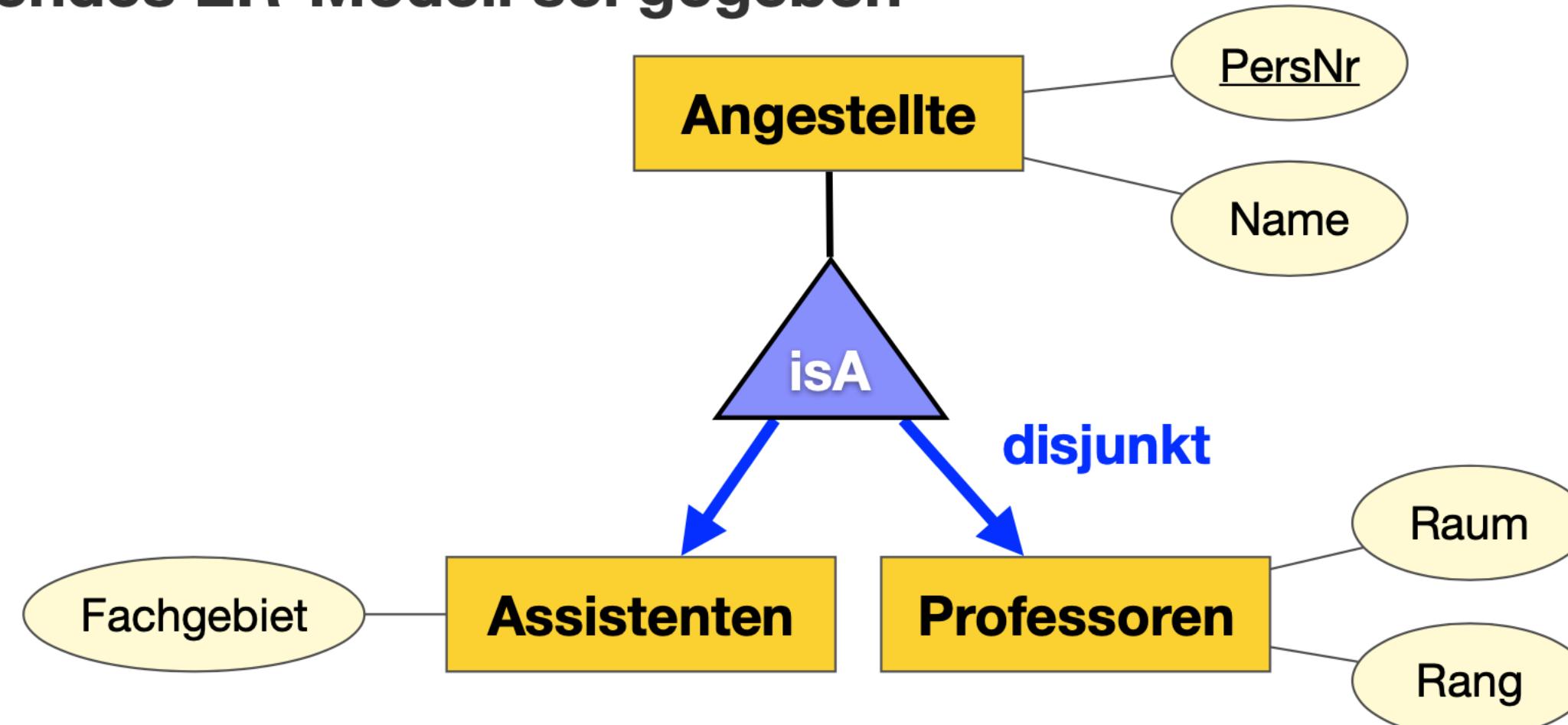
```
create table AssiDaten (
    PersNr      integer not null,
    Fachgebiet  varchar(30),
    Boss        integer );
```

```
create view Professoren as (
    select *
    from Angestellte a, ProfDaten d
    where a.PersNr=d.PersNr );
```

```
create view Assistenten as (
    select *
    from Angestellte a, AssiDaten d
    where a.PersNr=d.PersN )r;
```

# Views und Generalisierung – Variante 2

- Folgendes ER-Modell sei gegeben



- **Variante 2: Sicht für Oberklasse**
- **geerbte Attribute werden repliziert:**
  - AndereAngestellte: {[PersNr, Name]}
  - Professoren: {[PersNr, Name, Rang, Raum]}
  - Assistenten: {[PersNr, Name, Fachgebiet]}

# Views und Generalisierung - SQL 2

- Nur Oberklasse als echte Tabelle, Unterklassen als Sichten

```
create table Professoren (
    PersNr  integer not null,
    Name    varchar (30) not null,
    Rang    character (2),
    Raum    integer );
```

```
create table Assistenten (
    PersNr  integer not null,
    Name    varchar (30) not null,
    Fachgebiet  varchar (30),
    Boss    integer );
```

```
create table AndereAngestellte (
    PersNr  integer not null,
    Name    varchar (30) not null );
```

```
create view Angestellte as (
    ( select PersNr, Name
        from Professoren )
    union
    ( select PersNr, Name
        from Assistenten )
    union
    ( select*
        from AndereAngestellte);
)
```

## Zusammenhang Objekt-Relationale-Mapper

---

### Idee

- Abbildung von Vererbungshierarchien.
- Je nach Anwendungsfall mehrere sinnvolle Möglichkeiten der Implementierung.
- Wer entscheidet, wie modelliert wird?
- Probleme erscheinen u.a. auch beim Weiterentwickeln der Software und der Synchronisation von Klassen und Daten im DBMS.

## Datenintegrität

---

### Idee

- Daten sollen in sich schlüssig, konsistent sein.
- Auch hier die Frage: Besser in externer Business Logik oder im DBMS?
- Wir benötigen Mittel, um Funktionalität zu modellieren.
- Was passiert im Fehlerfall?

# Integrität

# Bisher: Column- und Table-Constraints



# Neu

- Komplexe Check-Constraints Über mehrere Tabellen mit Hilfe von Triggern.
  - Referentielle Integrität Vermeiden von Fremdschlüsselverweisen "ins Leere".

# Referentielle Integrität

## Fremdschlüssel

- verweisen auf Tupel einer Relation
- z.B. "gelesenVon" in "Vorlesungen" verweist auf Tupel in "Professoren"

## referentielle Integrität

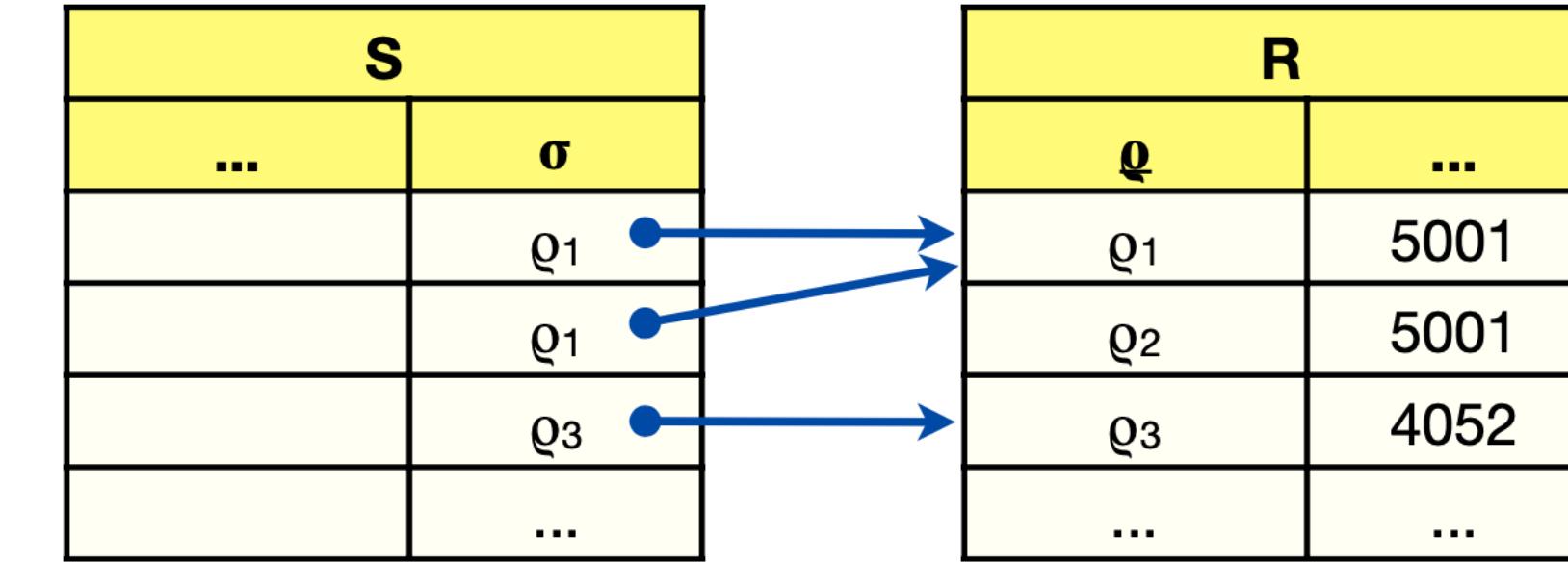
- Fremdschlüssel müssen auf existierende Tupel verweisen oder den Wert "NULL" haben

Zugehöriger Eintrag in Tabelle  
"Studenten" muss existieren!

hören	
MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
...	...

# Referentielle Integrität

```
// Tabelle R mit Schlüssel q
create table R (
    q integer primary key
    ...
)
// Tabelle S mit Schlüssel σ
create table S (
    σ integer references R
    ...
)
```



- In S dürfen für σ nur Werte eingetragen werden, für die es in R eine passende Zeile gibt (oder σ hat den Wert NULL)
- In R dürfen keine Zeilen gelöscht werden, für die es noch Verweise aus S heraus gibt
- Ein Schlüssel in R darf nur dann geändert werden, wenn es keinen Verweis darauf in S gibt

DBMS erzeugt Fehler, falls eine der Bedingungen verletzt ist!

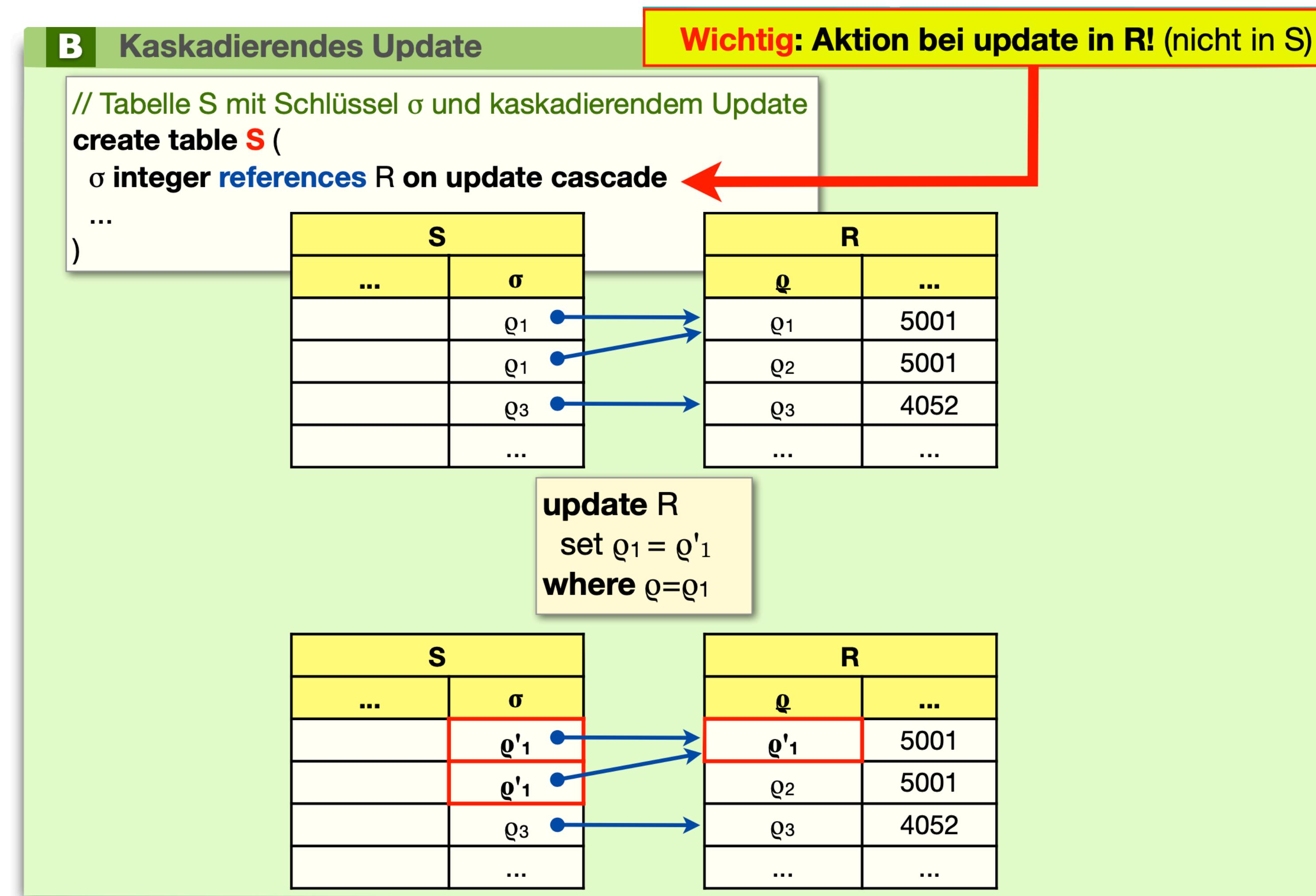
## Kaskadierende Operationen

---

**Für Tabellen können gesonderte Regeln für festgelegt werden:**

- Durch die Klausel **on update cascade** werden Veränderungen des Primärschlüssels auf den Fremdschlüssel propagiert
- Durch die Klausel **on delete cascade** zieht das Löschen eines Tupels in R das Entfernen des auf ihn verweisenden Tupels in S nach sich
- Durch die Klauseln **on update set null** und **on delete set null** erhalten beim Ändern bzw. Löschen eines Tupels in R die entsprechenden Tupel in S einen Nulleintrag

# Kaskadierende Operationen

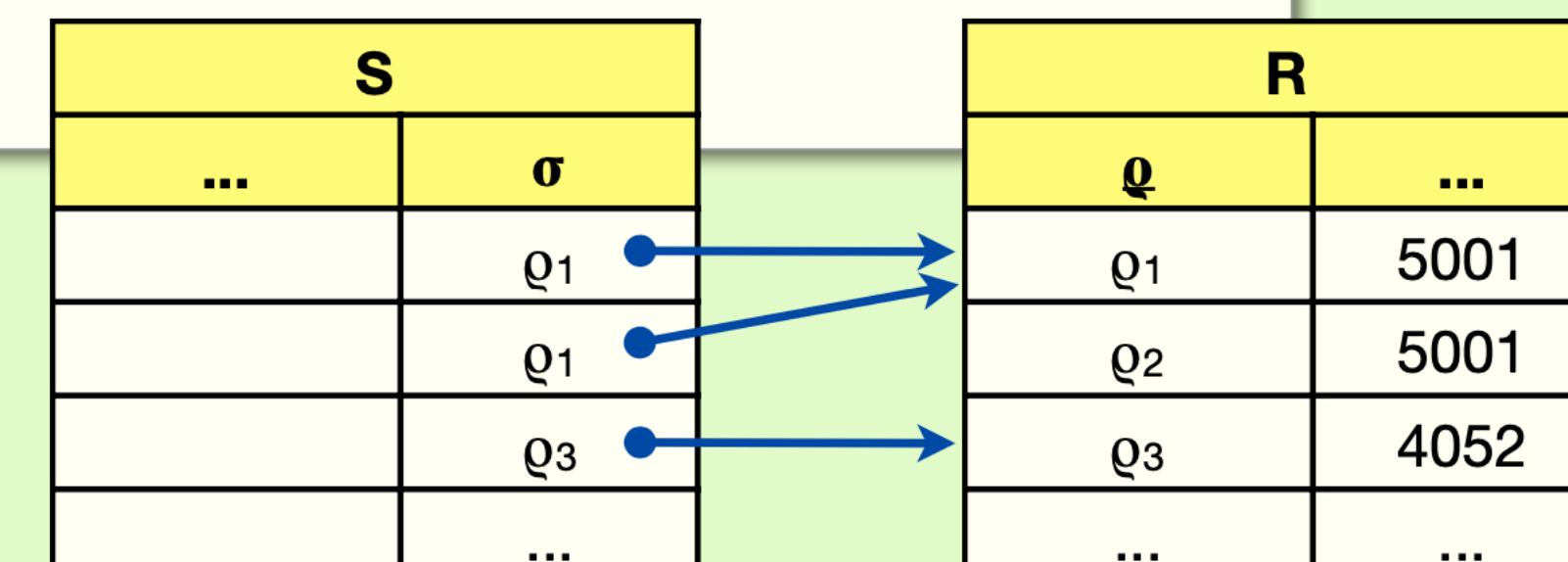


# Kaskadierende Operationen

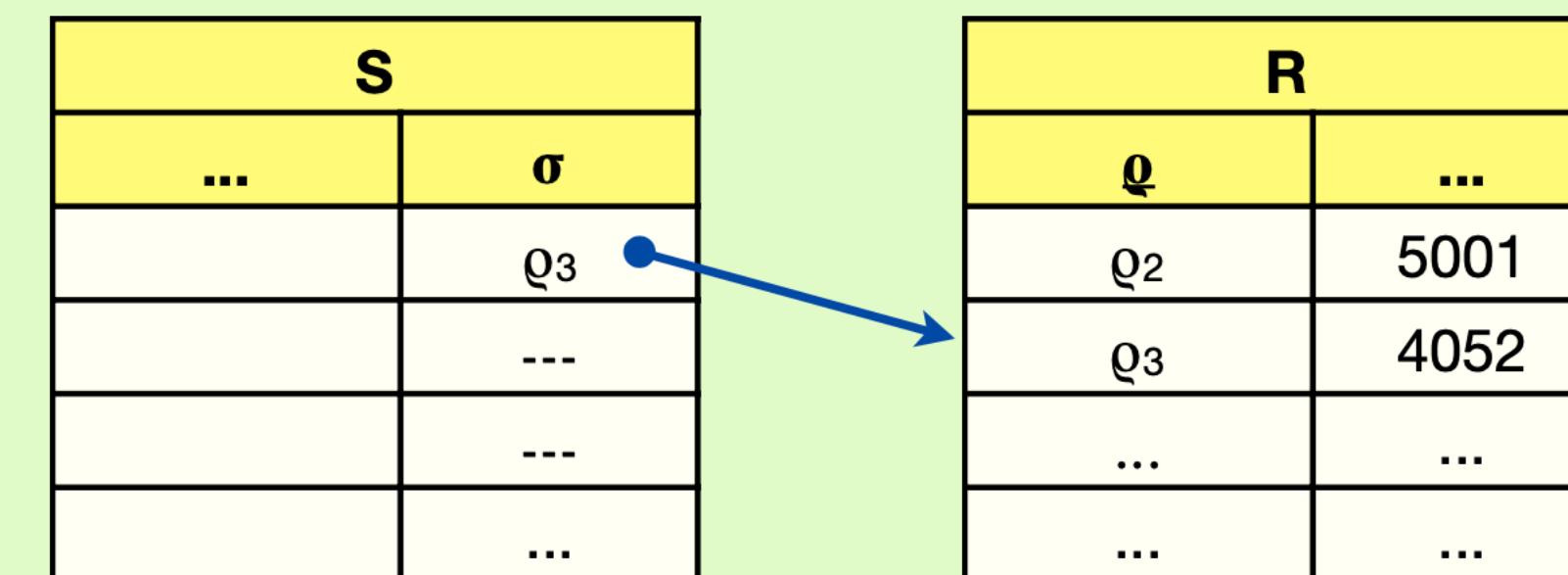
## B Kaskadierendes Update

// Tabelle S mit Schlüssel  $\sigma$  und kaskadierendem Löschen

```
create table S (
     $\sigma$  integer references R on delete cascade
    ...  
)
```



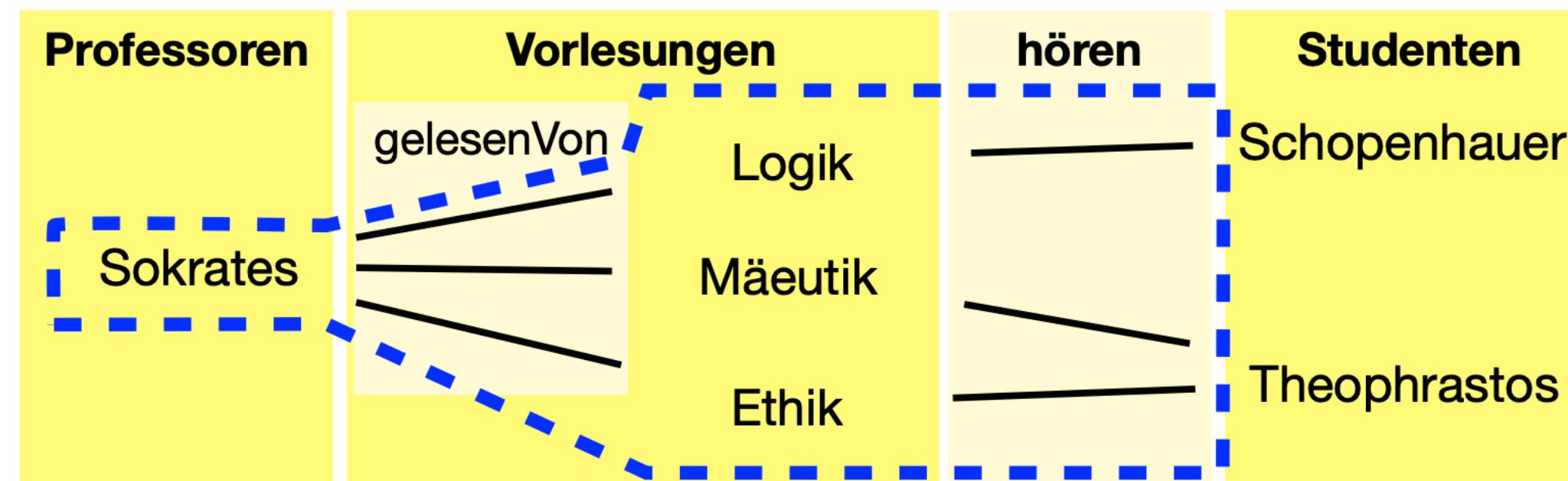
```
delete from R  
where Q=Q1
```



# Kaskadierende Operationen

**Es ist durchaus erlaubt mehrere Klauseln zu verwenden:**

```
// Tabelle S mit Schlüssel σ und kaskadierendem Löschen
create table S (
    σ integer references R on delete cascade
        on update cascade
    ...
)
```



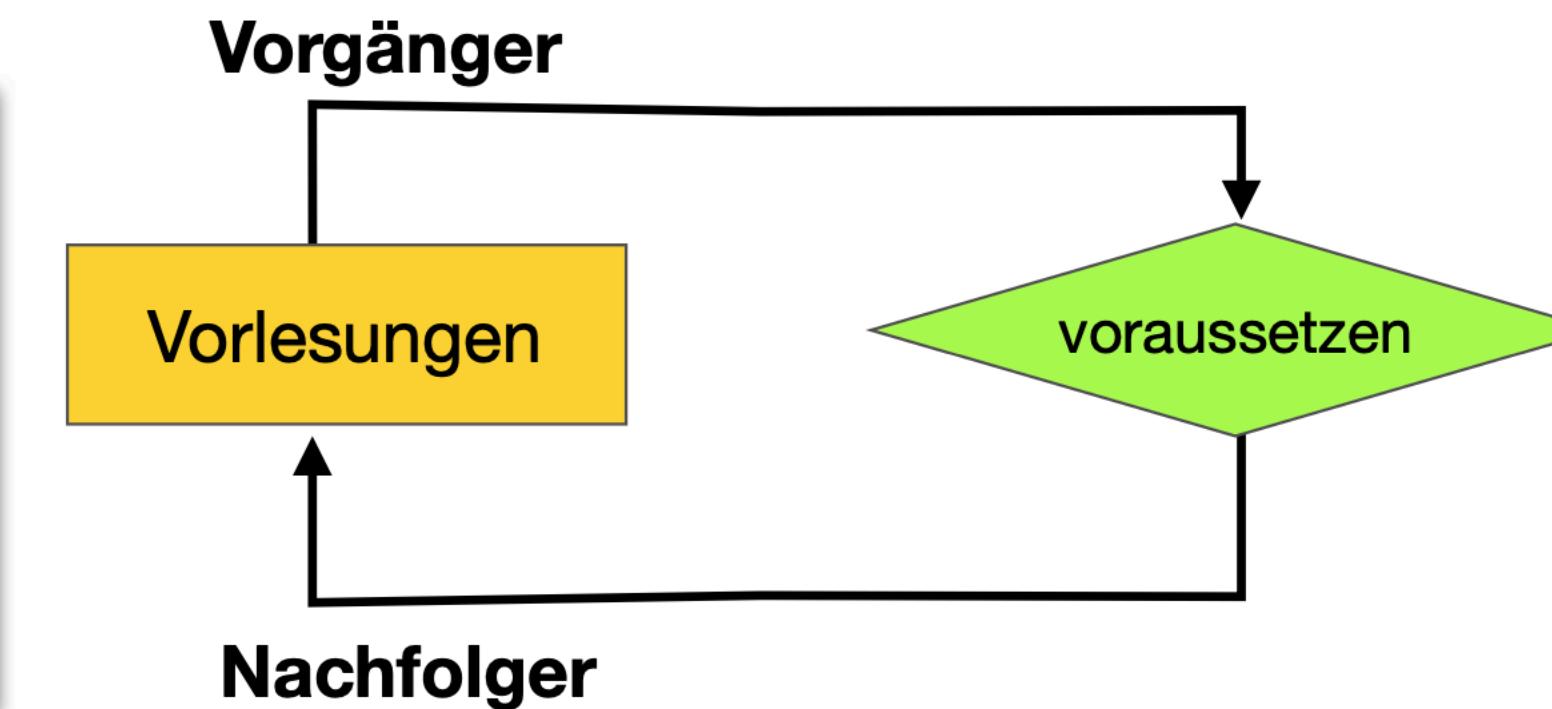
**Vorsicht bei kaskadierendem Löschen!**



## Kaskadierende Operationen

**Klausel "on ..." bezieht sich immer auf Vorgänge in Tabelle mit Primärschlüssel!**

```
create table voraussetzen (
    Vorgänger references Vorlesungen
        on delete cascade,
    Nachfolger references Vorlesungen
        on delete cascade,
    primary key (Vorgänger,Nachfolger)
)
```



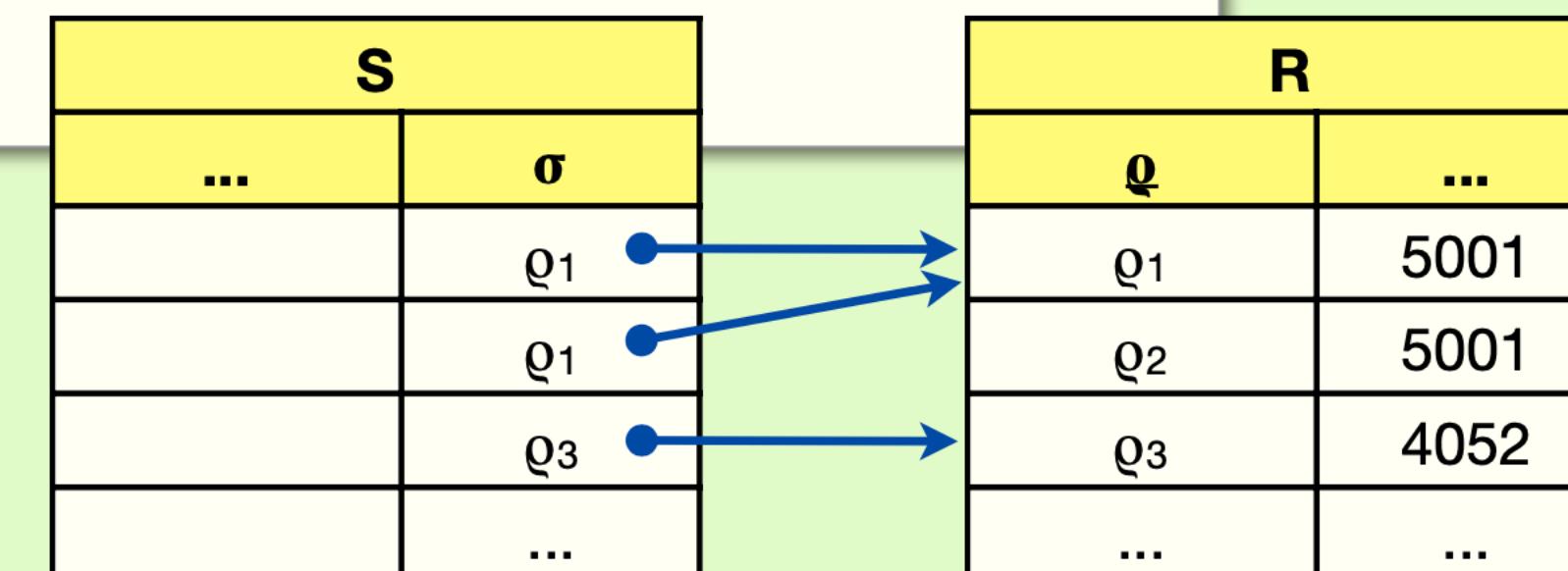
- Es wird nicht innerhalb von **voraussetzen** kaskadiert!

# Kaskadierende Operationen

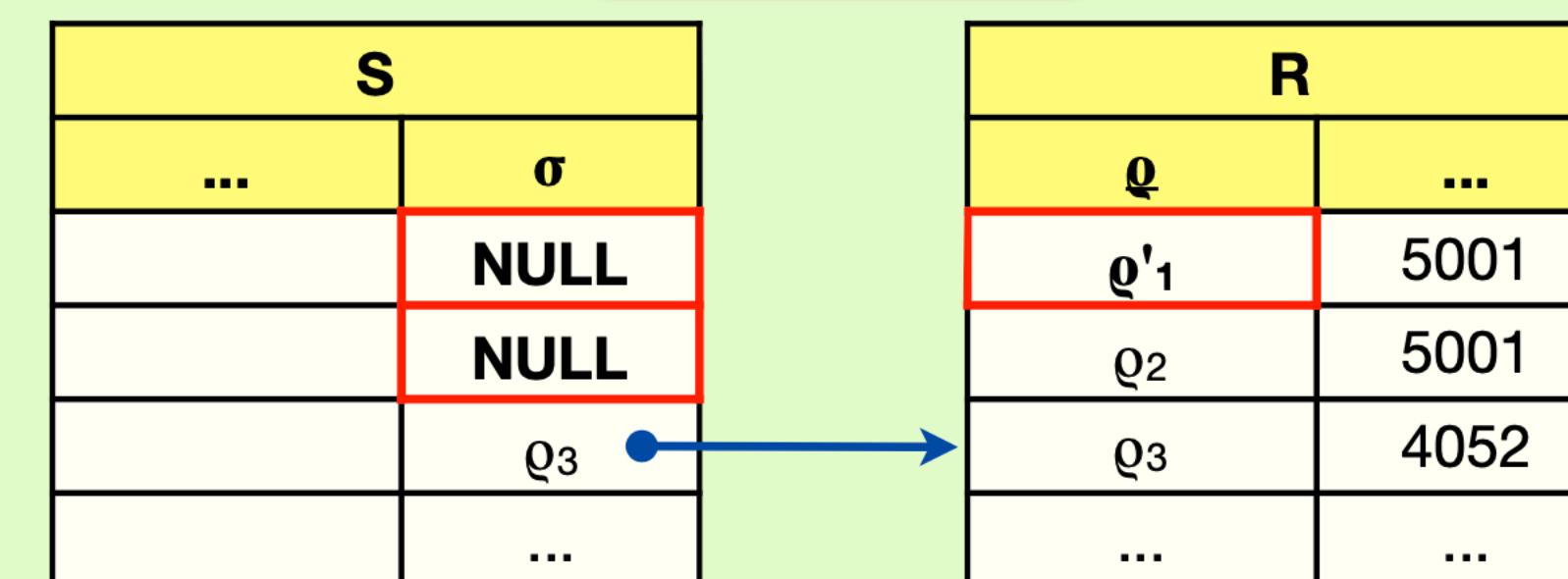
## B Kaskadierendes Update

// Tabelle S mit Schlüssel  $\sigma$  und kaskadierendem Update

```
create table S (
     $\sigma$  integer references R on update set null
    ...
)
```



```
update R
set Q1 = Q'1
where q=Q1
```



# Check

- **Falls als *table constraint* realisiert, kann Konsistenz einer Zeile geprüft werden**
  - Rückgriff nur auf Attribute dieser Zeile
- **Nicht weit verbreitet: Komplexere check-constraints**
  - Beispiel: Student darf nur Vorlesungen prüfen, die er besucht hat

```
create table prüfen (
    MatrNr integer references Studenten on update cascade
                                            on delete cascade,
    VorlNr integer references Vorlesungen on update cascade ,
    PersNr integer references Professoren,
    Note numeric (2,1) check (Note between 1.0 and 5.0),
    primary key (MatrNr, VorlNr),
    check( exists ( select * from hören as h
                    where h.VorlNr = prüfen.VorlNr and h.MatNR = prüfen.MatNr))
```

- **Problem:** *constraint* kann verletzt werden, wenn sich hören-Tabelle ändert!
- **Daher:** in Tupel und Spalten-*constraint* immer nur auf die Attribute der Zeile zugreifen!
  - nur dann: **verlässliche constraints!**

# Assertions

---

- **Relationsübergreifende Integritätsbedingungen**
  - nicht auf Operationen bzgl. einer Tabelle beschränkt
  - **Konsequenz:** muss bei jeder Änderung geprüft werden

- **Syntax:**

```
create assertion Name check( expression )
drop assertion Name
```

- **Werden von kaum einem DBMS unterstützt!**
  - Alternative: Auf Trigger zurückgreifen

## Beispiel Assertions

### B Einfache Stored Procedure als SQL-Kommando

- Student darf nur Vorlesung prüfen, wenn er diese gehört hat

```
create assertion check_prüfen
check( exists ( select * from hören as h
                 where h.VorlNr = prüfen.VorlNr and h.MatNR = prüfen.MatNr)
      );
```

#### Beachte:

- **nur boole'sche Ausdrücke möglich!**  
→ einfach in der Handhabung - *constraint programming*
- **Alle Operationen, die Assertion verletzen müssen abgelehnt werden!**  
→ **Problem:** Welche Operationen sind das?  
→ Für DBMS nicht einfach herauszufinden, daher kaum umgesetzt  
Alternative: Trigger - Programmierer legt fest, was wann geprüft werden muss

## Stored Procedures

---

### Idee

- Funktionalität, z.B. Geschäftslogik oder Massnahmen zur Erhaltung der Datenintegrität, im DBMS.

### Q&A

- DBMS oder Business-Logik im Programm?
- Vor- und Nachteile DBMS?

## Stored Procedures

---

- Grundlage für Trigger
- Mit Stored Procedures können Datenbankabläufe als definierte Prozessabläufe ohne Benutzerinteraktion hinterlegt werden
  - werden im Server in übersetzter Form gespeichert
  - werden vom Server ausgeführt (Vermeidung von Datentransporten zum Client)
  - Vorgefertigte Abläufe können das Sicherheitsniveau verbessern  
(z.B. Vermeidung von SQL-Injections)
- Neben SQL kommen auch andere Sprachen zum Einsatz
  - neben speziellen "Makrosprachen" z.B. auch - JAVA (z.B. Oracle, DB2), Perl, PHP (PostgreSQL), .NET-Sprachen (Microsoft) oder JAVA
  - wenige Implementierungen orientieren sich am SQL/PSM-Standard  
(PSM=Persistent Stored Modules), MySQL, DB2, PostgreSQL (mit zusätzlichem Modul PL/pgPSM)

# Stored Procedures

## B Einfache Stored Procedure als SQL-Kommando

- Funktion zur Berechnung des Durchschnitts zweier Zahlen

```
create or replace function average(IN numeric,IN numeric) returns numeric AS
  'select ($1 + $2) / 2.0;'    // Beachte den Makro-Character!
language SQL
immutable                      // Hinweis an Optimizer: keine Änderung an Tabellen
returns null on null input; // Umgang mit NULL-Werten

select average(3,5);
```

- Alle Vorlesungen nebst Dozent und SWS, die ein Student hört

```
create or replace function alleVorlesungen(IN integer) returns table (titel varchar,
                                                               name varchar,
                                                               sws integer) AS

'select titel,name,sws from (hören natural join Vorlesungen)
                           join Professoren on PersNr=gelesen Von
                           where MatrNr = $1;'

language SQL;

select * from alleVorlesungen(28106);
```

# Stored Procedures

## B Einfache Stored Procedures in anderen Sprachen

- Funktion zur Berechnung des Durchschnitts zweier Zahlen in PL/PGSQL

```
create or replace function average(IN a numeric,IN b numeric) returns numeric AS $$  
begin  
    return (a + b) / 2.0;  
end  
$$  
language plpgsql  
immutable  
returns null on null input;
```

**Start- bzw. End-Tag**

- Berechnung des Maximums in PL/PHP

```
create or replace function php_max(integer, integer) returns integer AS $$  
if ($args[0] > $args[1]) {  
    return $args[0];  
} else {  
    return $args[1];  
}  
$$  
strict // Kurzform für set null on null input  
language 'plphp';
```

# Trigger

- Vereinbarung einer Stored Procedure und der sie auslösenden Ereignisse, etwa
    - Events Durch Änderung ausgelöstes Ereignis.
    - Conditions Filtern und Kategorisieren der Änderung.
    - Actions Durch das DBMS durchgeführte Anweisungen.
  - Ein Trigger kann vor oder nach einer Operation ausgelöst werden.
    - Vor einer Aktion: Unterdrücken möglich; z.B. wenn neue Werte bestimmten Bedingungen nicht genügen -> komplexe check-constraints.
  - Mit einem Trigger können komplexe Aktionen mit Ereignissen kombiniert werden, z.B. besondere Maßnahmen bei bestimmten Fehlersituationen.
  - Häufig werden mit einem Trigger auch nicht Datenbank-spezifische Abläufe in einem Unternehmen angesprochen (falls das DBMS entsprechendes unterstützt), wie das Versenden einer SMS oder E-Mail bei Störfällen.

# Trigger

## B Trigger als komplexes check-constraint (PostgreSQL)

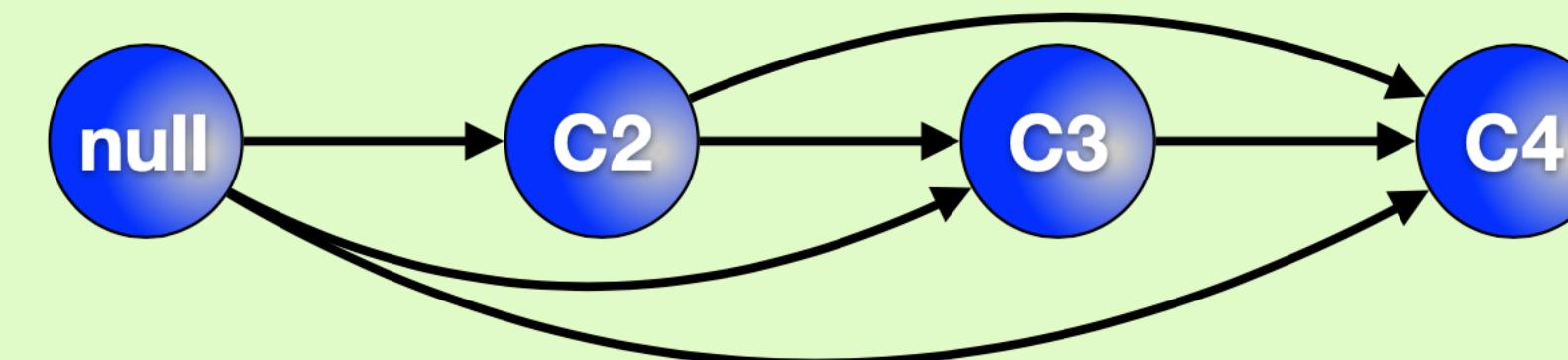
- Student darf nur Vorlesung prüfen, wenn er diese gehört hat

```
create or replace function mussVorlesungHoeren() returns trigger as $$  
declare hatGehoert boolean;  
    name varchar;  
    vorlesung varchar;  
begin  
    hatGehoert = exists (select * from hören as h  
                          where h.VorlNr = NEW.VorlNr AND h.MatrNr = NEW.MatrNr);  
    if (not hatGehoert) then  
        select s.name into name from studenten as s where MatrNr = NEW.MatrNr;  
        select v.titel into vorlesung from vorlesungen as v where VorlNr = NEW.VorlNr;  
        raise notice '% hat die Vorlesung % nicht gehört',name,vorlesung;  
        return OLD;  
    end if;  
    return NEW;  
end  
$$ LANGUAGE plpgsql;  
  
create trigger mussVorlesungHoeren  
before insert on prüfen  
for each row execute procedure mussVorlesungHoeren();
```

# Trigger

## B Trigger als komplexes check-constraint 2 (PostgreSQL)

- Rang eines Professors nicht beliebig änderbar
  - Degradierungen sind nicht erlaubt



```
create or replace function keineDegradierung() returns trigger AS $$  
begin  
    if (OLD.Rang is not null) then  
        if (NEW.Rang is NULL) then NEW.Rang = OLD.Rang;  
        elsif (NEW.Rang < OLD.Rang) then  
            raise exception '% darf nicht von % auf % degradiert werden',  
                NEW.name,OLD.Rang,NEW.Rang;  
        end if;  
    end if;  
    return NEW;  
end  
$$ language plpgsql;
```

```
create trigger keineDegradierung before update on Professoren  
for each row execute procedure keineDegradierung();
```

## Trigger

---

### Wann sind Trigger sinnvoll?

- Bedingung kann nicht einfach über SQL-Constraints formuliert werden, bzw. als Ersatz für Assertions.
- Die entsprechende Überprüfung müsste über verschiedene Anwendungsprogramme erfolgen.
- Es sollen nicht datenbankspezifische Aktionen getroffen werden.

### Anmerkungen

- Trigger sollten nur dann verwendet werden, wenn die Reaktion nicht mittels Check und Assertion realisiert werden kann.
- Vorsicht: Trigger können wieder weitere Trigger auslösen!
- Bei reinen 3-Ebenen-Architekturen werden diese Mechanismen oft in der mittleren Ebene implementiert (Geschäftsprozesslogik).

# Beispieldatenbank Kemper/Eickler

---

Tabellen werden u.a. bei Prüfungen im SQL-Teil verwendet.

Dabei geht es allerdings um die SQL-Kommandos, nicht um das Ergebnis. Das bedeutet, primär die Struktur der Tabellen ist relevant, nicht der Inhalt.

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

prüfen			
MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Studenten			
MatrNr	Name	Semester	
24002	Xenokrates	18	
25403	Jonas	12	
26120	Fichte	10	
26830	Aristoxenos	8	
27550	Schopenhauer	6	
28106	Carnap	3	
29120	Theophrastos	2	
29555	Feuerbach	2	

Vorlesungen			
VorlNr	Titel	SWS	gelesenVon
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

Assistenten			
PersNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

# UNIT 0x0c

## TRANSAKTIONEN

## ANFRAGEOPTIMIERUNG

## Motivation

---

### Transaktionen

- Situation: Mehrere Benutzer greifen auf das DBMS 'gleichzeitig' zu.

#### Q&A

- Was kann schon schiefgehen... ?
- Welche Aktionen sind betroffen?

### Anfrageoptimierung

- Situation: Abfragen dauern sehr lange.

#### Q&A

- Welche Engpässe gibt es?
- Wie könnte man Anfragen im DBMS optimieren?

## Transaktionen – Motivation

---

### Standardbeispiel

- Bei einer Überweisung wird Geld von einem Konto auf ein anderes transferiert.
- Nur beide Aktionen zusammen sind zulässig, also Abbuchen und Einzahlen.
- Schlägt nur eine der Aktionen fehl, so darf der Datenbestand nicht verändert sein bzw. muss eine vollständige Rückabwicklung gewährleistet werden.

### Konzept

- Aus logischer Sicht ist eine Transaktion *ein* (1) Arbeitspaket. So klein wie möglich aber so gross wie nötig, um alle Integritätsbedingungen einhalten zu können.
- Aus technischer Sicht ist eine Transaktion eine Folge von Datenbankoperationen mit einem definierten Beginn und einem definierten Abschluss.
- Die Transaktionsverwaltung ist eine nicht triviale Kernaufgaben eines Datenbanksystems mit Mehrbenutzerverwaltung.

## Transaktionen im DBMS

---

### Commit / Rollback(Abort) - Grundidee

- Es wird, explizit oder implizit, eine Transaktion gestartet (`START TRANSACTION`).
- Alle Befehle, die während einer laufenden Transaktion ausgeführt werden, verändern die lokale Sicht auf die Daten. Andere Benutzer oder andere Sessions sehen diese Änderungen erst einmal nicht.
- Läuft alles gut, werden alle Änderungen bestätigt (`COMMIT`). Tritt ein Fehler auf, wird die Transaktion abgebrochen und der Zustand vor Beginn der Transaktion wieder hergestellt (`ROLLBACK`).
- Je nach DBMS ist in der Standardeinstellung ein sog. `autocommit` aktiv, d.h. jeder Befehl wird implizit mit einem `COMMIT` oder `ROLLBACK` beendet, so dass alle Nutzer und Sessions alle erfolgreichen Änderungen automatisch sofort sehen oder aber im Fehlerfall nichts verändert wurde.
- Details dazu im SQL-Praktikum.

## ACID

---

### Anforderungen an ein DBMS - ACID-Prinzip

- Atomicity (Atomarität)
- Consistency (Konsistenz)
- Isolation (Nebenläufigkeit)
- Durability (Dauerhaftigkeit)

**Ein Verständnis der Ideen hinter den Begriffen ist wichtig.**  
**Wir stellen erst die Begriffe vor und danach die technische Umsetzung der Transaktionen im DBMS/in SQL und die Locking-Mechanismen.**



### Anmerkungen

- 'ACID' wird häufig zur Charakterisierung eines DBMS genannt, insbesondere auch für Nicht-Relationale DBMS. Dort gibt es aber auch 'analoge' Konzepte wie etwa 'BASE' (Basically Available, Soft state, Eventual Consistency)...
- 'Locking' und 'Transaktionen' sind wesentlich für Parallelität der Anwendung.
- Konzept der 'Transaktionen' ist nicht auf DBMS beschränkt, z.B. Daten im Speicher.

## ACID

---

### Anforderungen Atomicity / Atomarität

- Es gibt Transaktionen und eine Transaktion ist atomar in dem Sinne, dass entweder alle durch die Operationen der Transaktion bedingten Änderungen in der Datenbank umgesetzt werden oder gar keine.  
→ "alles-oder-nichts"-Prinzip.
- Als Konsequenz benötigen wir einen Mechanismus in dem DBMS, um Transaktionen zu definieren bzw. zu starten, zu beenden oder abzubrechen.
- Beobachtung bisher: Alle SQL-Befehle wurden direkt ausgeführt und die Folgen waren sofort sichtbar...

## ACID

---

### Anforderungen Consistency / Konsistenz

- Ausgangspunkt ist eine konsistente Datenbank vor dem Beginn einer Transaktion.
- Am Ende einer Transaktion ist eine Datenbank in einem konsistentem Zustand, d.h.
  - entweder jede Integritätsregel ist erfüllt, etwa referentielle Integrität (Fremdschlüssel), oder
  - die Transaktion wird komplett zurückgesetzt und die Datenbank befindet sich wieder in dem (konsistenten) Zustand wie zu Beginn.
- Achtung: Während der Abarbeitung einer Transaktion können bestimmte Inkonsistenzen erlaubt sein, da muss man im konkreten DBMS nachlesen, ob dem so ist.

## ACID

---

### Anforderungen Isolation / Nebenläufigkeit

- Nebenläufige, also parallele bzw. gleichzeitig ausgeführte Transaktionen (generell Operationen) beeinflussen sich nicht.
- Logisch betrachtet: jede Transaktion hat exklusiven Zugriff auf die Datenbank und erhält 'Ihre' Version während der Abarbeitung der Transaktion.
- Achtung: In so einem Szenario ist es sehr leicht, 'falsche' Daten zu erzeugen. Es werden Konzepte benötigt, z.B. Sperren, um Effekte wie
  - **'lost updates'**
  - **'dirty reads'**zu verhindern. Diese Situationen werden im Anschluss vorgestellt.

## ACID

---

### Anforderungen Durability / Dauerhaftigkeit

- Nach Abschluss einer Transaktion haben die von ihr ausgeführten Änderungen dauerhaft bestand. Das bedeutet, auch Prozessabstürze oder Plattenfehler dürfen nicht zu Datenverlust führen.
- Diese Anforderung bedeutet üblicherweise, dass Daten auf nicht-flüchtigen Speichermedien gespeichert (persistiert) werden.

## Nebenläufigkeit - Lost Update

- **Betrachte folgende, nebenläufige Transaktionen**
  - **T<sub>1</sub>**: Buche Betrag 300 von Konto A auf Konto B
  - **T<sub>2</sub>**: Schreibe Konto A 3% Zinsen zu

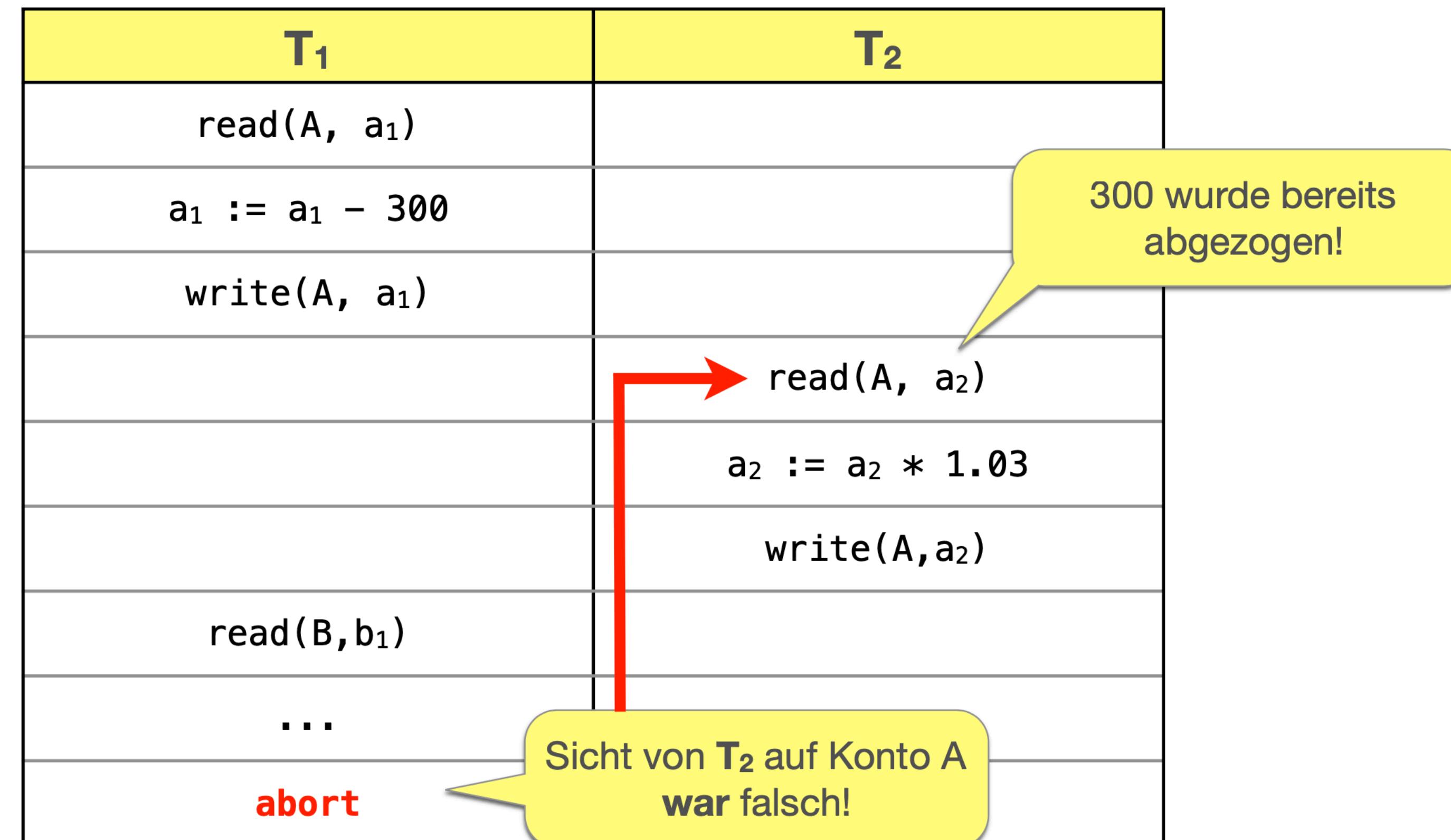
T <sub>1</sub>	T <sub>2</sub>
read(A, a <sub>1</sub> )	
a <sub>1</sub> := a <sub>1</sub> - 300	
	read(A, a <sub>2</sub> )
	a <sub>2</sub> := a <sub>2</sub> * 1.03
	write(A, a <sub>2</sub> )
write(A, a <sub>1</sub> )	
read(B, b <sub>1</sub> )	
b <sub>1</sub> := b <sub>1</sub> + 300	
write(B, b <sub>1</sub> )	

**Sicht von T<sub>1</sub> auf Konto A nach write falsch!**

**Zinsgutschrift geht hier verloren!**

## Nebenläufigkeit - Dirty Read

- **Betrachte folgende, nebenläufige Transaktionen**
  - $T_1$ : Buche Betrag 300 von Konto A auf Konto B
  - $T_2$ : Schreibe Konto A 3% Zinsen zu



# Nebenläufigkeit - Locking

## Je nach Operation (read / write) zwei Sperrmodi

- **S** (shared lock, Lesesperre):
  - S-Sperre auf Datum A ermöglicht Lesen mittels read. Mehrere S-Sperren auf ein Datum sind möglich
- **X** (exclusion lock, Schreibsperre):
  - X-Sperre auf Datum A ermöglicht Schreiben mittels write. Nur eine X-Sperre auf einem Datum ist möglich.
  - X-Sperre auf Datum A setzt voraus, dass keine S-Sperre auf A gesetzt ist!

		Aktuelle Locks auf Datum		
		kein Lock	S	X
Anforderung	S	✓	✓	✗
	X	✓	✗	✗

Verträglichkeitsmatrix

## Nebenläufigkeit – Locking

---

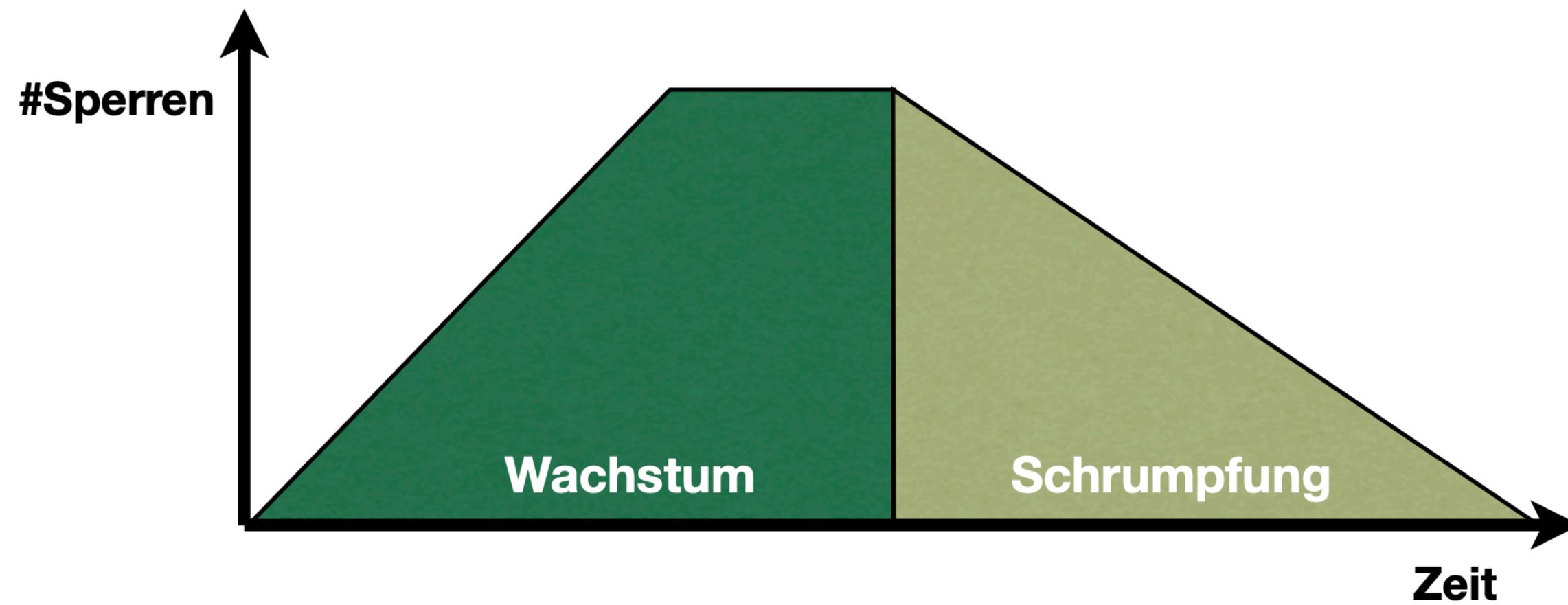
### Folgende Regeln müssen eingehalten werden:

- Jedes Objekt muss vor der Benutzung gesperrt werden.
- Eine Transaktion fordert eine Sperre, die sie schon besitzt, nicht erneut an.
- Eine Transaktion respektiert vorhandene Sperren gemäß der Verträglichkeitsmatrix und wird ggf. in eine Warteschlange eingereiht.
- Jede Transaktion durchläuft zwei Phasen:
  - → Wachstumsphase (nur Sperren anfordern)
  - → Freigabephase (nur Sperren freigeben)
- Spätestens bei Transaktionsende muss eine Transaktion all ihre Sperren zurückgeben

## Nebenläufigkeit - Locking

T <sub>1</sub>	T <sub>2</sub>	Bemerkung
BOT		
lockX(A)		
read(A)		
write(A)		
	BOT	
	lockS(A)	T <sub>2</sub> muss warten – T <sub>1</sub> hat X-Lock auf A
lockX(B)		
read(B)		
unlockX(A)		T <sub>2</sub> wecken
	read(A)	
	lockS(B)	T <sub>2</sub> muss warten – T <sub>1</sub> hat X-Lock auf B
write(B)		
unlockX(B)		T <sub>2</sub> wecken
	read(B)	
commit		
	unlockS(A)	
	unlockS(B)	
	commit	

## Nebenläufigkeit - Locking



- **Rollback:** Rücknahme der Operationen einer abgebrochenen Transaktion
- Problem mit 2PL: Kaskadierendes Rollback

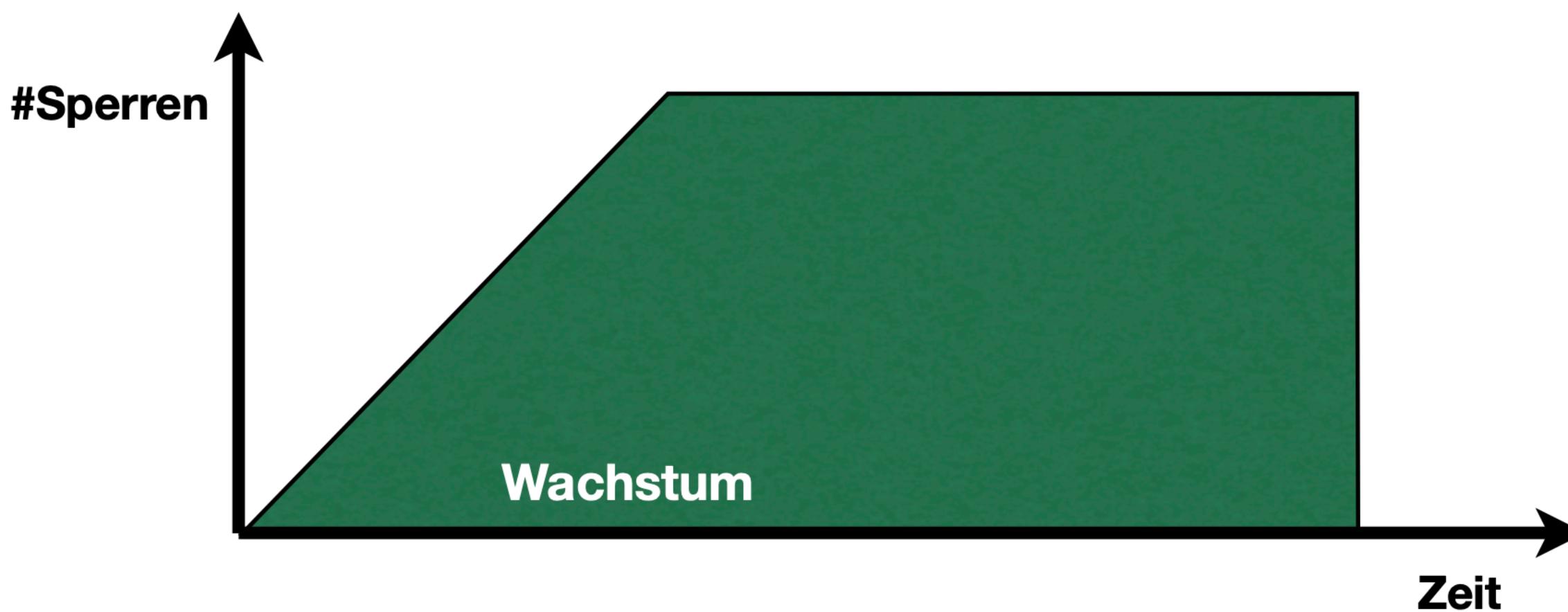
## Nebenläufigkeit - Locking

T <sub>1</sub>	T <sub>2</sub>	Bemerkung
BOT lockX(A) read(A) write(A)	<b>Rollback</b> Alle Änderungen zurücknehmen	
lockX(B) read(B) unlockX(A)	BOT lockS(A)	T <sub>2</sub> muss warten – T <sub>1</sub> hat X-Lock auf A
write(B) unlockX(B)	read(A) lockS(B)	T <sub>2</sub> wecken
<b>abort</b> <b>Dirty Reads!</b>	read(B)	T <sub>2</sub> muss warten – T <sub>1</sub> hat X-Lock auf A
	unlockS(A) unlockS(B)	T <sub>2</sub> wecken
	commit	<b>Kaskadiertes Rollback</b> Alle Änderungen zurücknehmen

## Nebenläufigkeit - Locking

---

- **Vermeidet kaskadierende Rollbacks**
- **Wie 2-Phasen-Sperrprotokoll, aber**
  - Keine Schrumpfungsphase
  - Alle Sperren werden erst zum Transaktionsende freigegeben



# Nebenläufigkeit - Locking

---

T <sub>1</sub>	T <sub>2</sub>	Bemerkung
BOT		
lockX(A)		
read(A)		
write(A)		
	BOT	
	lockS(A)	T <sub>2</sub> muss warten – T <sub>1</sub> hat X-Lock auf A
lockX(B)		
read(B)		
write(B)		
unlockX(A)		
unlockX(B)		
<b>abort</b>		T <sub>2</sub> wecken
 <b>Unproblematisch Keine Dirty Reads!</b>		
	read(A)	
	lockS(B)	
	read(B)	
	unlockS(A)	
	unlockS(B)	
	commit	

## Nebenläufigkeit - Locking

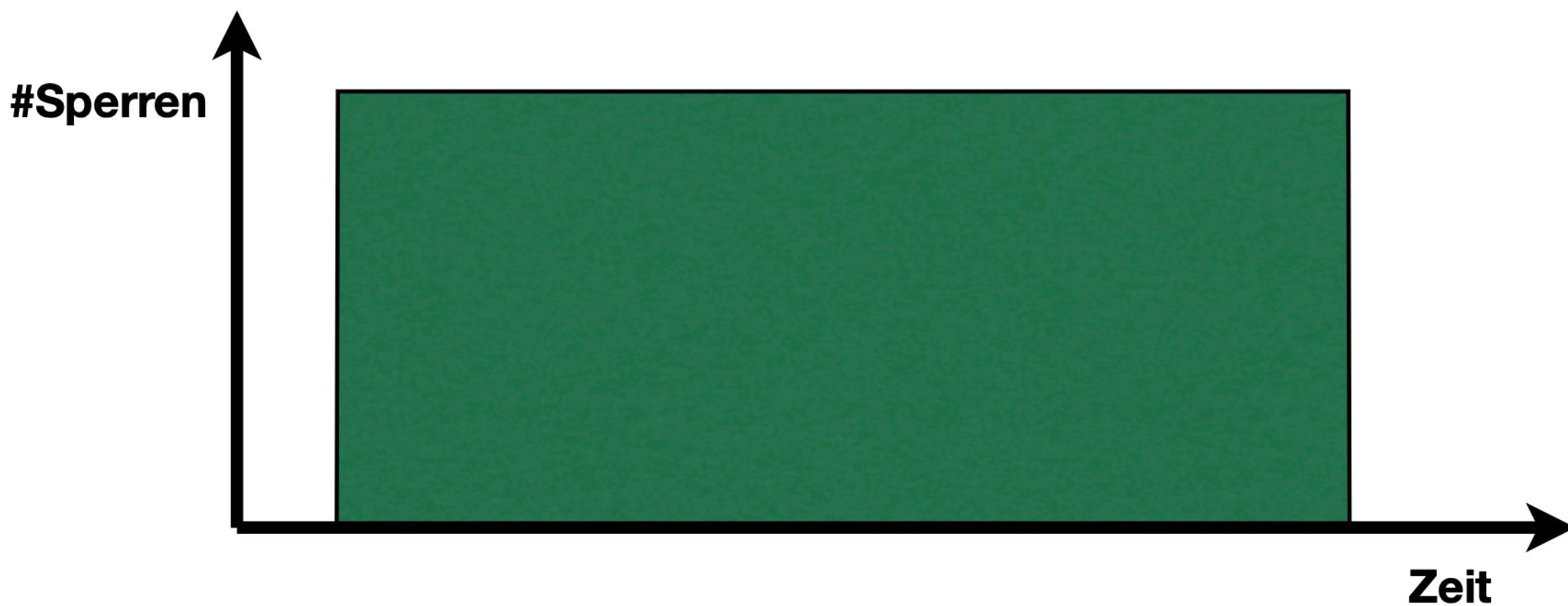
### Inherentes Problem der auf Sperren basierte Synchronisation

- Verklemmungen (Deadlocks)

T <sub>1</sub>	T <sub>2</sub>	Bemerkung
B0T		
lockX(A)		
	B0T	
	lockS(B)	
	read(B)	
read(A)		
write(A)		
lockX(B)		T <sub>1</sub> muss warten – T <sub>2</sub> hat S-Lock auf B
	lockS(A)	T <sub>2</sub> muss warten – T <sub>1</sub> hat X-Lock auf A
<b>Deadlock</b> T <sub>1</sub> und T <sub>2</sub> blockieren sich gegenseitig		

## Nebenläufigkeit - Locking

- **Idee: Notwendige Sperren mit BOT anfordern**
  - Konsequenz: keine ausgewiesene Wachstumsphase mehr



- **Problem:**
  - Notwendige Sperren nicht vorhersehbar (z.B. if .. then .. else ..)
  - **Daher:** Obermenge tatsächlich benötigter Sperren anfordern
    - das geht aber zu Lasten der Parallelität!

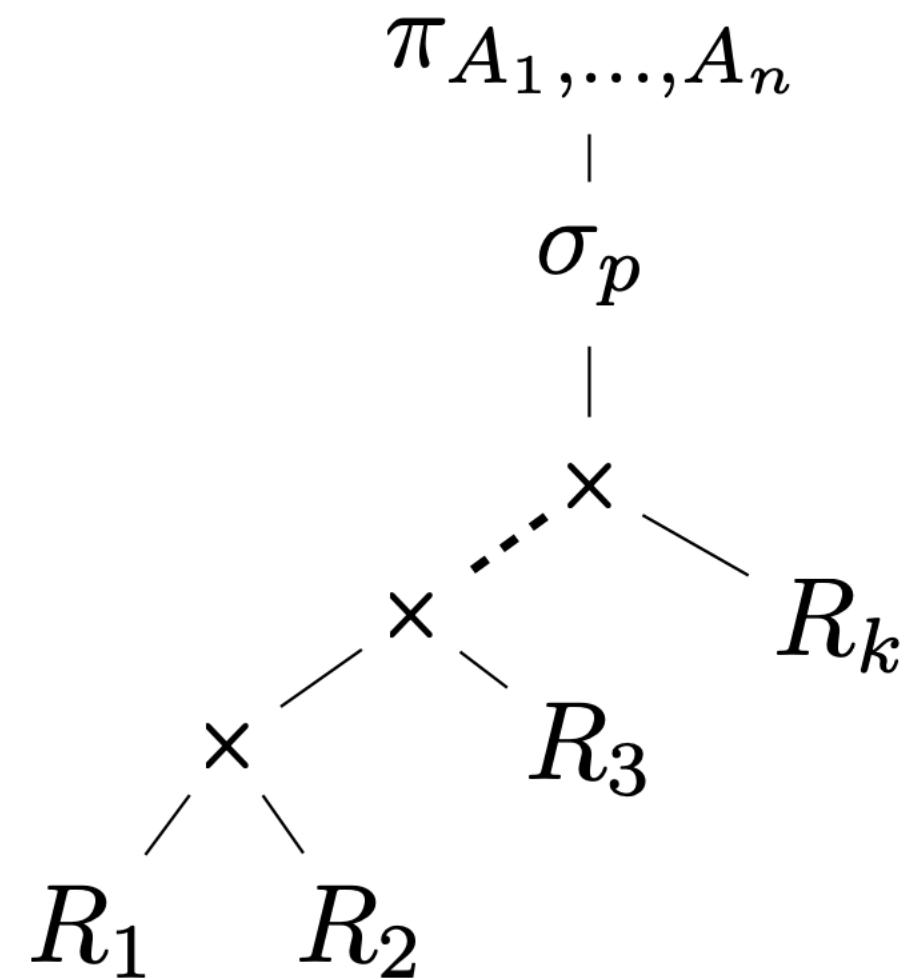
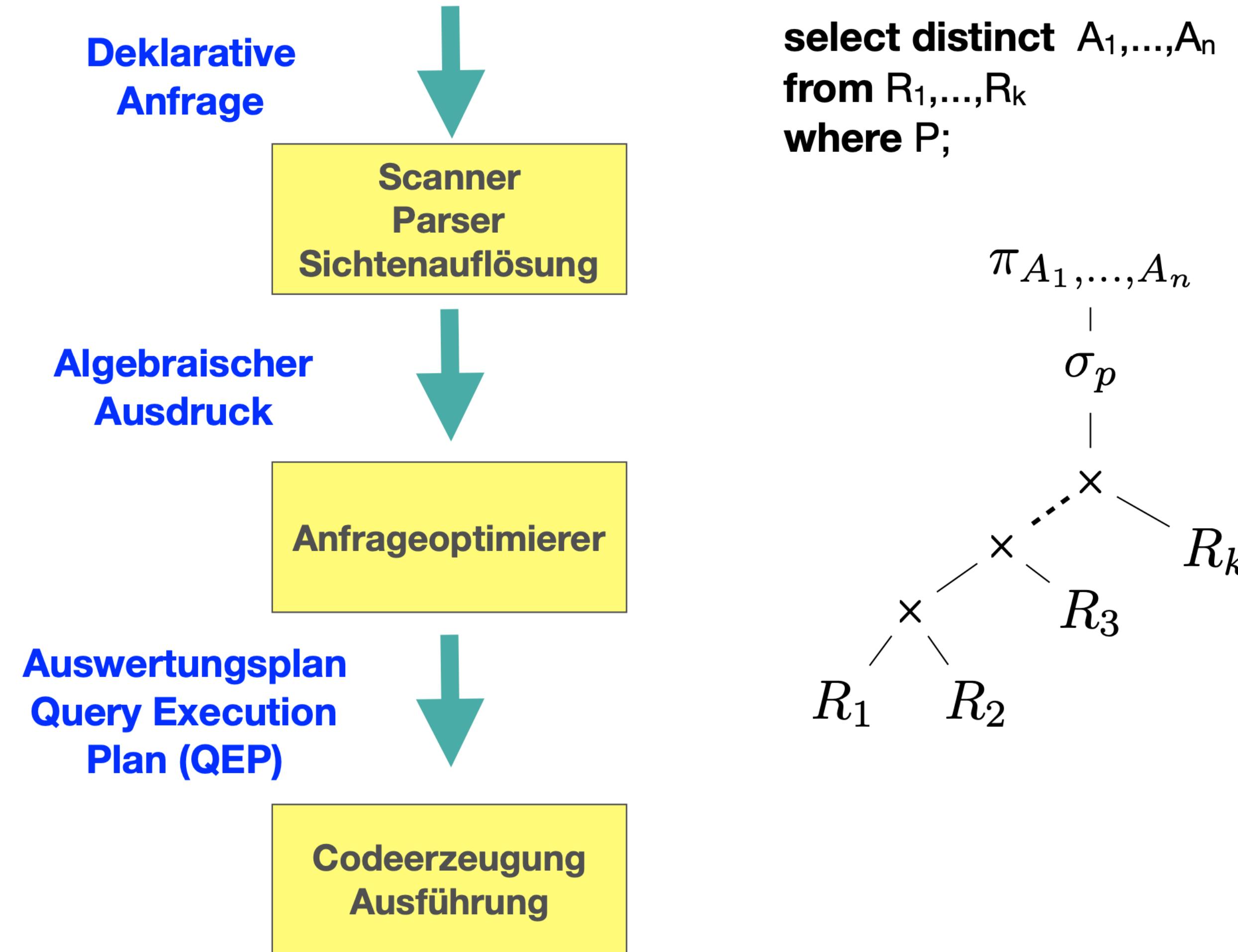
## Nebenläufigkeit - Locking

---

### Anmerkungen

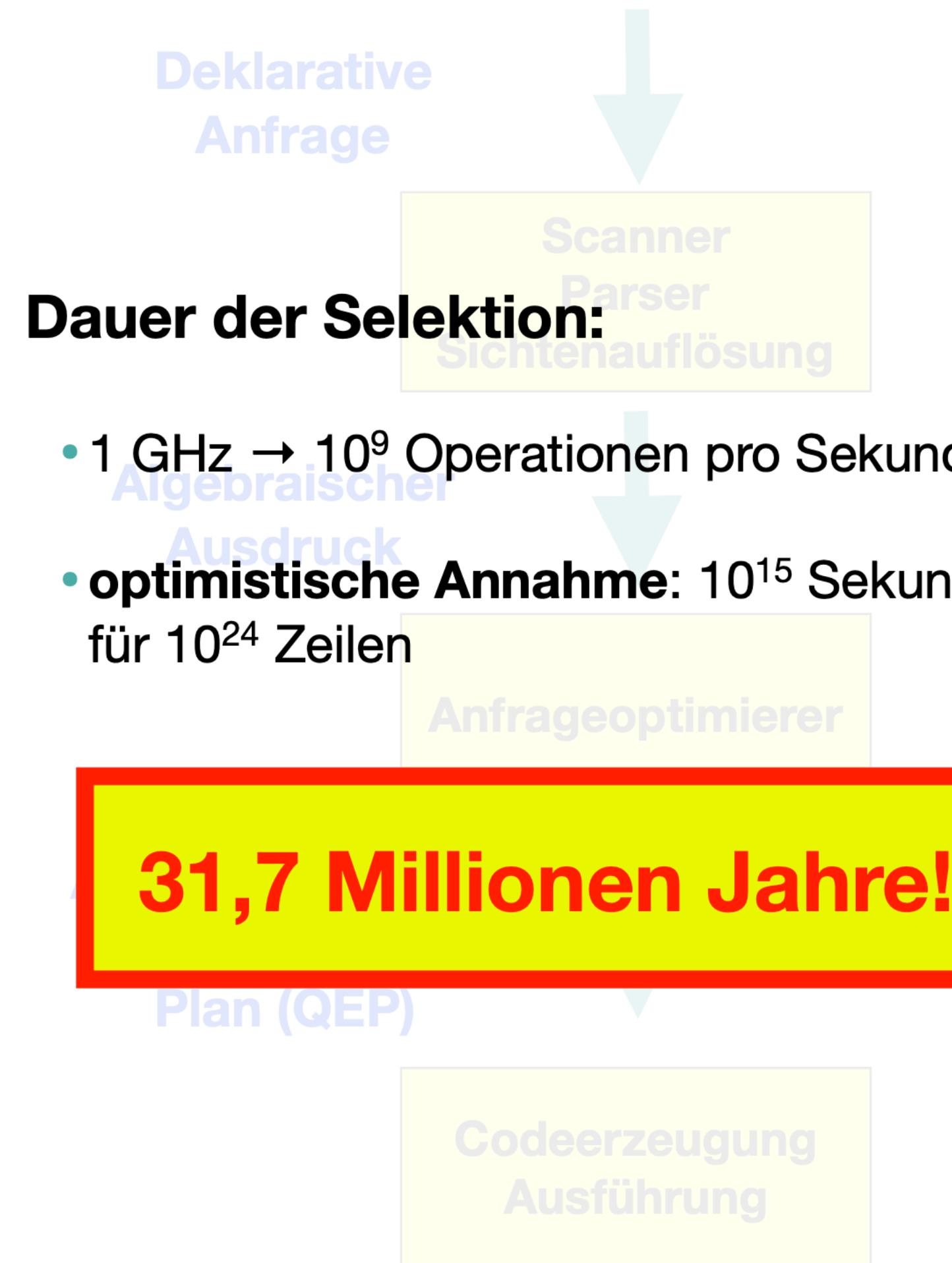
- Umgang mit Parallelität nicht DBMS-spezifisch.
- Beste Optimierung: 'Nachdenken' und z.B. Realisierung über möglichst unabhängige Aufteilung von Daten und/oder Aufgaben.
- Parallelität kostet!
- Zu restriktives Locking sealisiert de-fakto.

## Ablauf der Anfrageoptimierung



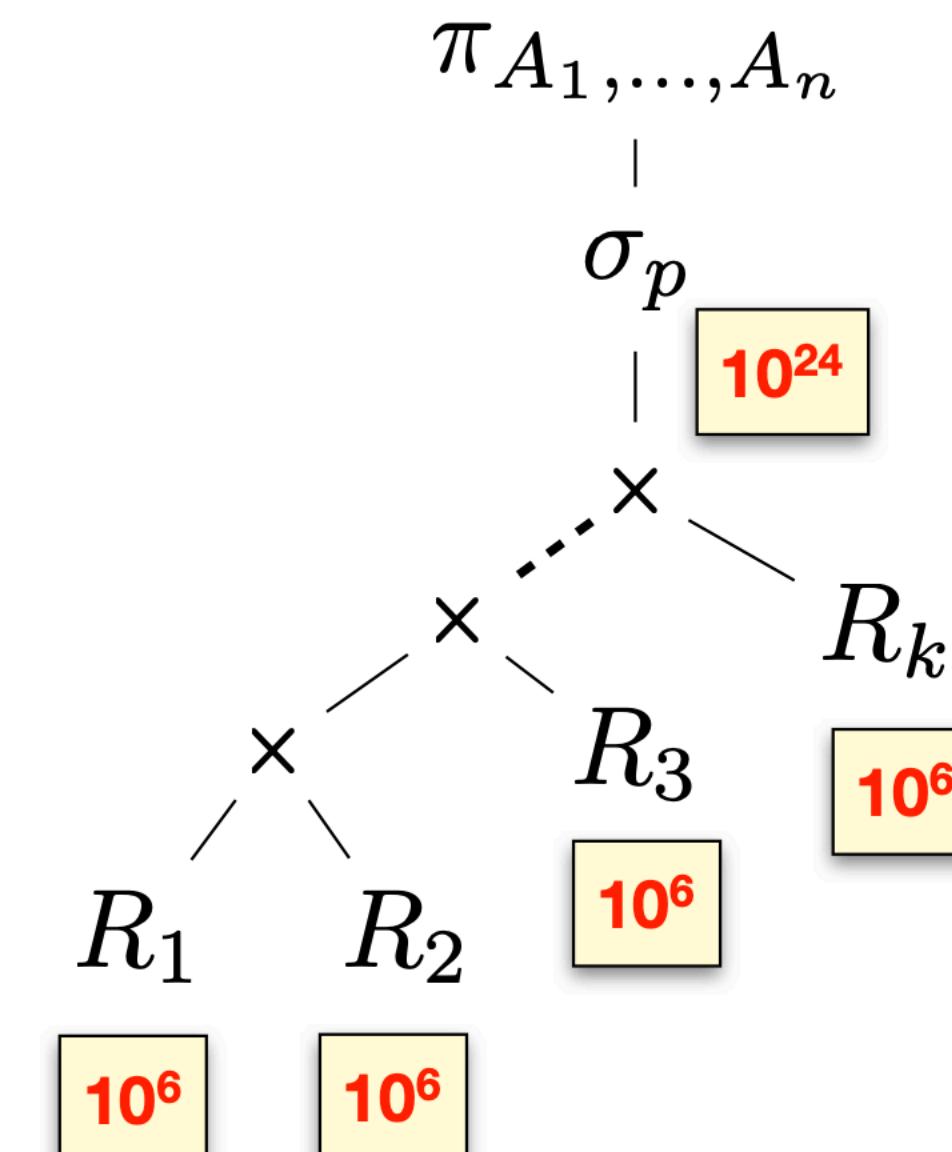
# Ablauf der Anfrageoptimierung

---



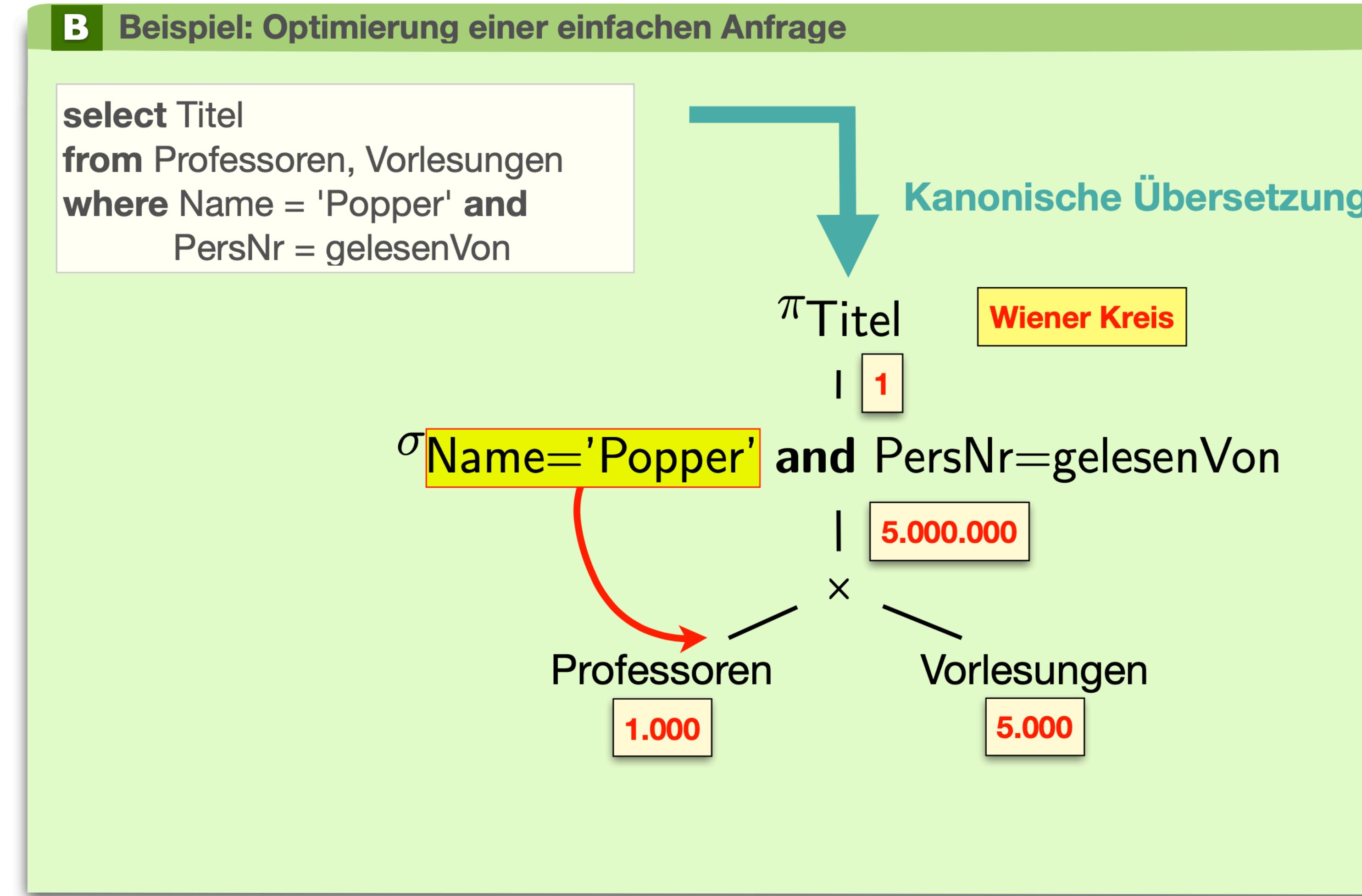
```

select distinct A1,...,An
from R1,...,Rk
where P;
  
```

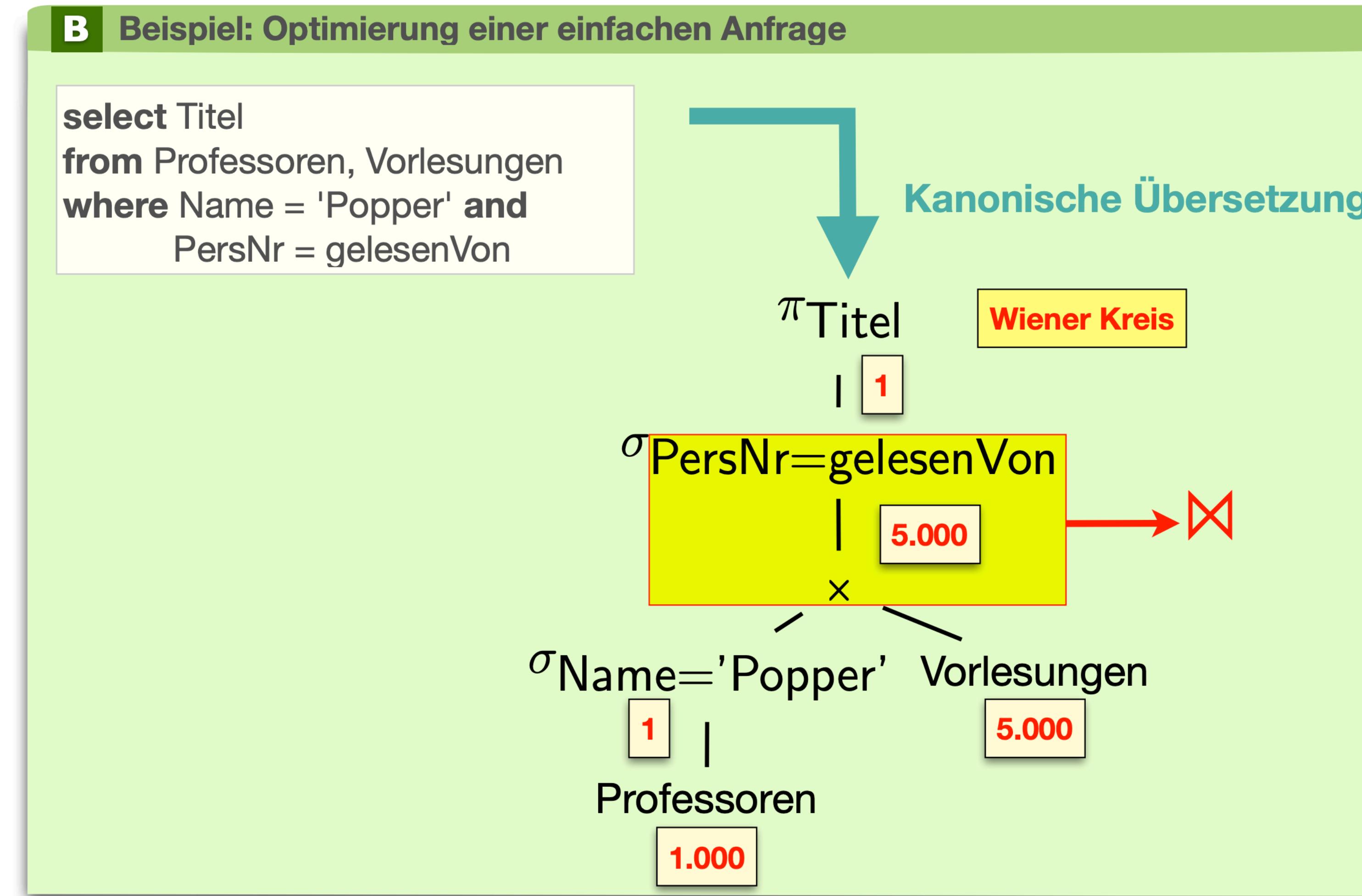


Kreuzprodukt von 4 Tabellen mit jeweils 1 Millionen Zeilen liefert Tabelle mit **1 Quadrillionen Spalten!**

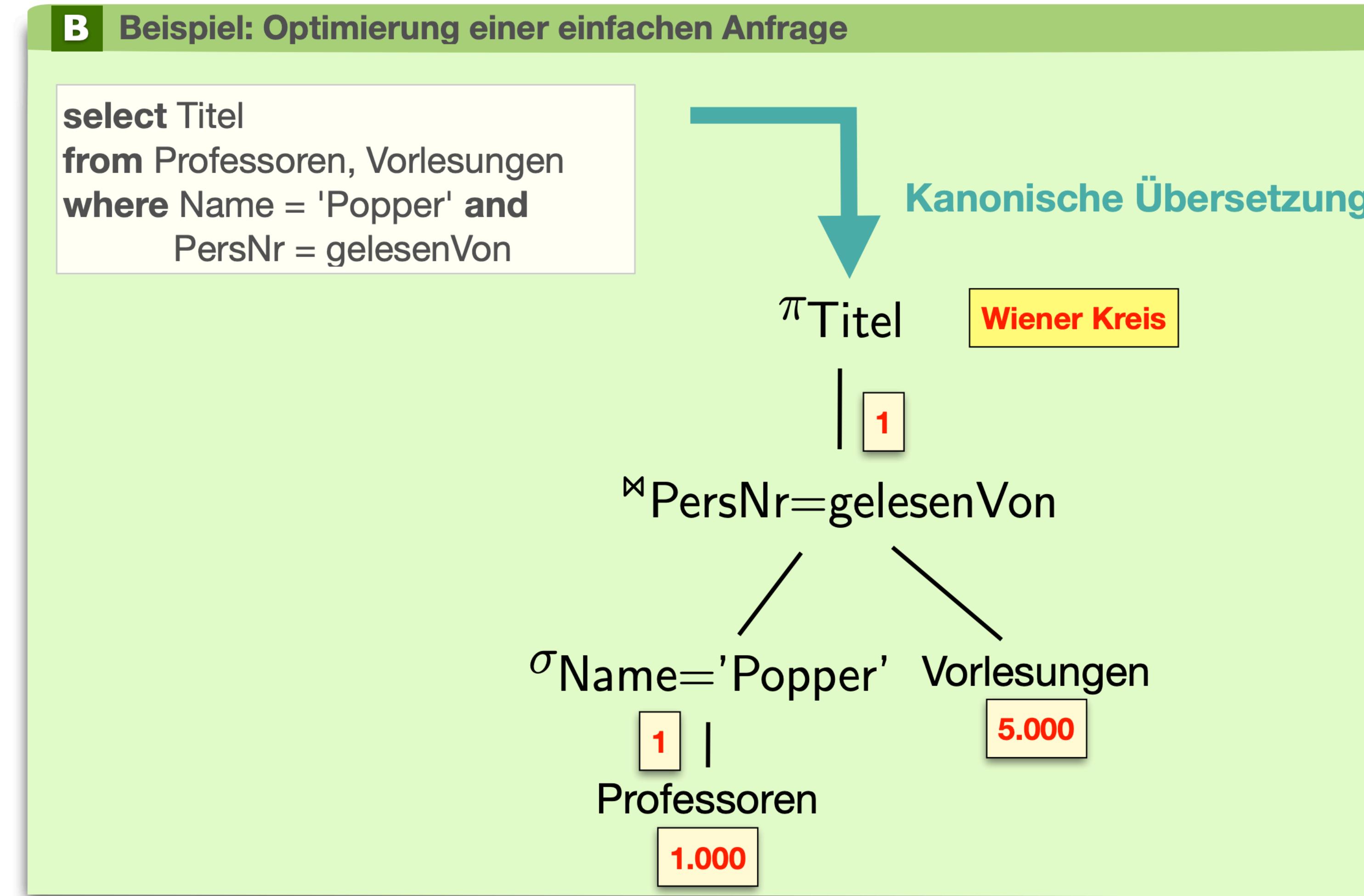
## Beispiel: Einfache Anfrageoptimierung



## Beispiel: Einfache Anfrageoptimierung



## Beispiel: Einfache Anfrageoptimierung



# Äquivalenzerhaltende Transformationen

## 1 Aufbrechen von Konjunktionen im Selektionsprädikat

$$\sigma_{C_1 \wedge C_2 \wedge \dots \wedge C_n} \equiv \sigma_{C_1}(\sigma_{C_2}(\dots(\sigma_{C_n}(R))\dots))$$

## 2 $\sigma$ ist kommutativ

$$\sigma_{C_1}(\sigma_{C_2}(R)) \equiv \sigma_{C_2}(\sigma_{C_1}(R))$$

## 3 $\pi$ -Kaskaden

- falls  $L_1 \subseteq L_2 \subseteq \dots \subseteq L_n$

$$\pi_{L_1}(\pi_{L_2}(\dots(\pi_{L_n}(R))\dots)) \equiv \pi_{L_1}(R)$$

## 4 Vertauschen von $\sigma$ und $\pi$

- Falls sich Selektionsprädikat  $C$  nur auf Attribute  $A_1, \dots, A_n$  der Projektionsliste bezieht:

$$\pi_{A_1, \dots, A_n}(\sigma_C(R)) \equiv \sigma_C(\pi_{A_1, \dots, A_n}(R))$$

# Äquivalenzerhaltende Transformationen

---

## 5 $\times$ , $\cup$ , $\cap$ und $\bowtie$ sind kommutativ

- se sei  $\Theta = \{\times, \cup, \cap, \bowtie\}$

$$R\Theta S \equiv S\Theta R$$

## 6 Pushing Selections

- Idee: Selektiere so früh wie möglich
- Falls Selektionsprädikat  $C$  sich nur auf Attribute von  $R$  bezieht:

$$\sigma_C(R \bowtie_{C_J} S) \equiv \sigma_C(R) \bowtie_{C_J} S$$

- Anmerkung: Falls Selektionsprädikat  $C$  eine Konjunktion der Form  $C_1 \wedge C_2$  ist und
  - $C_1$  sich nur auf Attribute von  $R$  bezieht und
  - $C_2$  sich nur auf Attribute von  $S$  bezieht

$$\sigma_{C_1 \wedge C_2}(R \bowtie_{C_J} S) \equiv \sigma_{C_1}(R) \bowtie_{C_J} \sigma_{C_2}(S)$$

# Äquivalenzerhaltende Transformationen

## 7 Vertauschen von $\pi$ mit $\bowtie$

- **Idee:** Projeziere so früh wie möglich!
- Sei die Projektionsliste  $L = \{A_1, \dots, A_n, B_1, \dots, B_m\}$ , wobei  $\{A_1, \dots, A_n\} \subseteq R$  und  $\{B_1, \dots, B_m\} \subseteq S$  und beziehe sich das Joinprädikat  $C_J$  **ausschließlich** auf Attribute aus  $L$ , dann

$$\pi_L(R \bowtie_{C_J} S) \equiv \pi_{A_1, \dots, A_n}(R) \bowtie_{C_J} \pi_{B_1, \dots, B_m}(S)$$

- Bezieht sich das Joinprädikat  $C_J$  auf weitere Attribute  $\{A'_1, \dots, A'_k\} \subseteq R$  und  $\{B'_1, \dots, B'_l\} \subseteq S$ , so müssen diese für den Join erhalten bleiben und können erst danach herausprojiziert werden:

$$\pi_L(R \bowtie_{C_J} S) \equiv \pi_L(\pi_{A_1, \dots, A_n, A'_1, \dots, A'_k}(R) \bowtie_{C_J} \pi_{B_1, \dots, B_m, B'_1, \dots, B'_l}(S))$$

# Äquivalenzerhaltende Transformationen

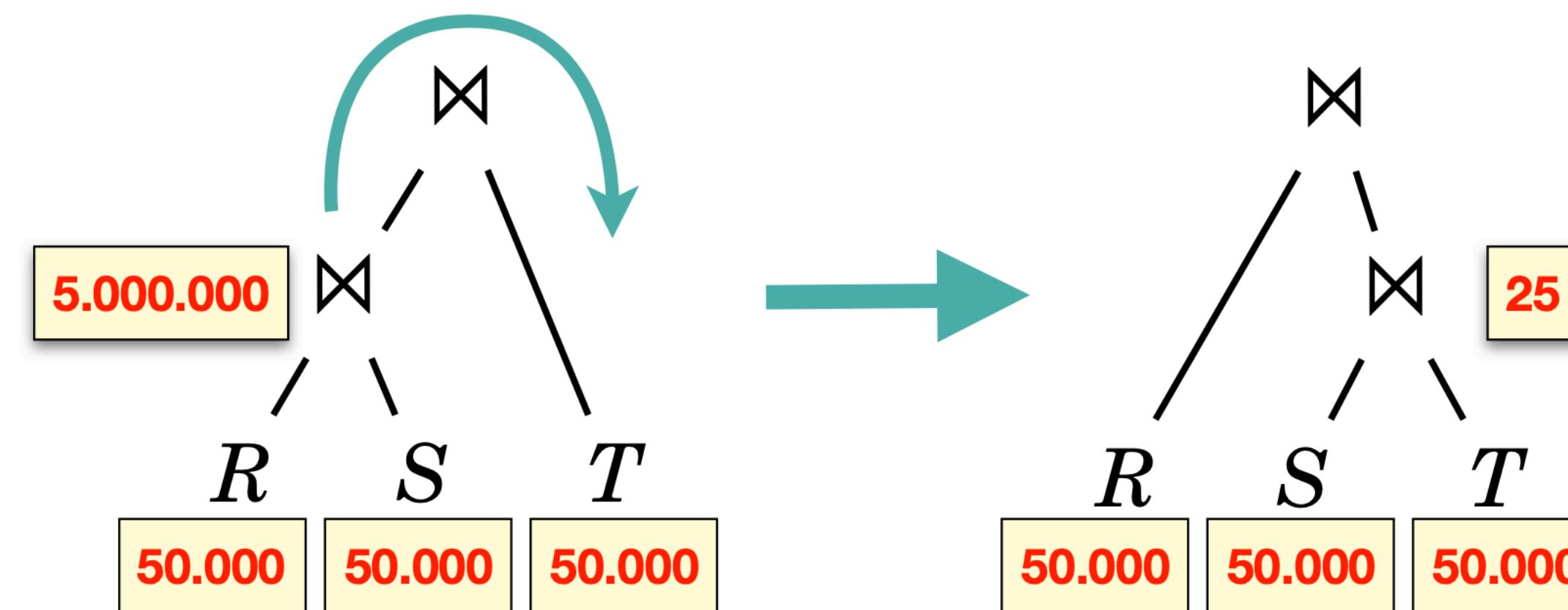
8

## Die Operationen $\bowtie, \times, \cup, \cap$ sind jeweils assoziativ

- seit  $\Theta = \{\times, \cup, \cap, \bowtie\}$

$$(R \Theta S) \Theta T \equiv R \Theta (S \Theta T)$$

- Anmerkung: Regel hat in Zusammenhang mit Join viel Potential!
  - Merke: DBMS führt intern Statistiken z.B. über Tabellengrößen



# Äquivalenzerhaltende Transformationen

## 9 $\sigma$ ist distributiv mit $\cup, \cap, \neg$

- se sei  $\Theta = \{\cup, \cap, \neg\}$

$$\sigma_C(R \Theta S) \equiv \sigma_C(R) \Theta \sigma_C(S)$$

## 10 $\pi$ ist distributiv mit $\cup$

$$\pi_C(R \cup S) \equiv \pi_C(R) \cup \pi_C(S)$$

## 11 De Morgans Regeln für Join- und Selektionsprädikate

- $\neg(a \wedge b) \equiv \neg a \vee \neg b$
- $\neg(a \vee b) \equiv \neg a \wedge \neg b$

Vorsicht bei NULL-Werten!

# Äquivalenzerhaltende Transformationen

---

12

## Zusammenfassung von $\sigma$ und $\times$ zu $\bowtie$

- Kartesisches Produkt  $R \times S$ , das von Selektionsoperation mit Prädikat  $C$  gefolgt wird und bei dem  $C$  nur Attribute aus  $R \cup S$  enthält, kann in Join überführt werden:

$$\sigma_C(R \times S) \equiv R \bowtie_C S$$

## Heuristik: Optimieren

---

### **Ziel: worst case vermeiden, nicht die optimale Anfrage!**

- Mittels Regel 1 werden konjunktive Selektionsprädikate in Kaskaden von  $\sigma$ -Operationen zerlegt.
- Mittels Regeln 2, 4, 6, und 9 werden Selektionsoperationen soweit „nach unten“ propagiert wie möglich.
- Mittels Regel 8 (Assoziativgesetz) werden die Blattknoten so vertauscht, dass derjenige, der das kleinste Zwischenergebnis liefert, zuerst ausgewertet wird.
- Eine  $\times$ -Operation, die von einer  $\sigma$ -Operation gefolgt wird, wird eine  $\bowtie$ -Operation.
- Mittels Regeln 3, 4, 7, und 10 werden Projektionen soweit wie möglich nach unten propagiert.
- Versuche Operationen zusammenzufassen, wenn sie in einem "Durchlauf" ausführbar sind (z.B. Anwendung von Regel 1, Regel 3, aber auch Zusammenfassung aufeinanderfolgender Selektionen und Projektionen zu einer „Filter“-Operation).

# Beispiel: Anfrageoptimierung

## B Beispiel: Anwendung der Heuristik auf einfache Anfrage

// In welchen Semestern befinden sich Studenten,  
// die eine Vorlesung von Sokrates besuchen

```
select distinct s.Semester
from Studenten s, Professoren p,
      Vorlesungen v,hören h
where p.Name = 'Sokrates' and
      v.gelesenVon = p.PersNr and
      v.VorlNr = h.VorlNr and
      h.MatrNr = s.MatrNr
```

Kanonische Übersetzung

$$\pi_{s.\text{Semster}} |$$

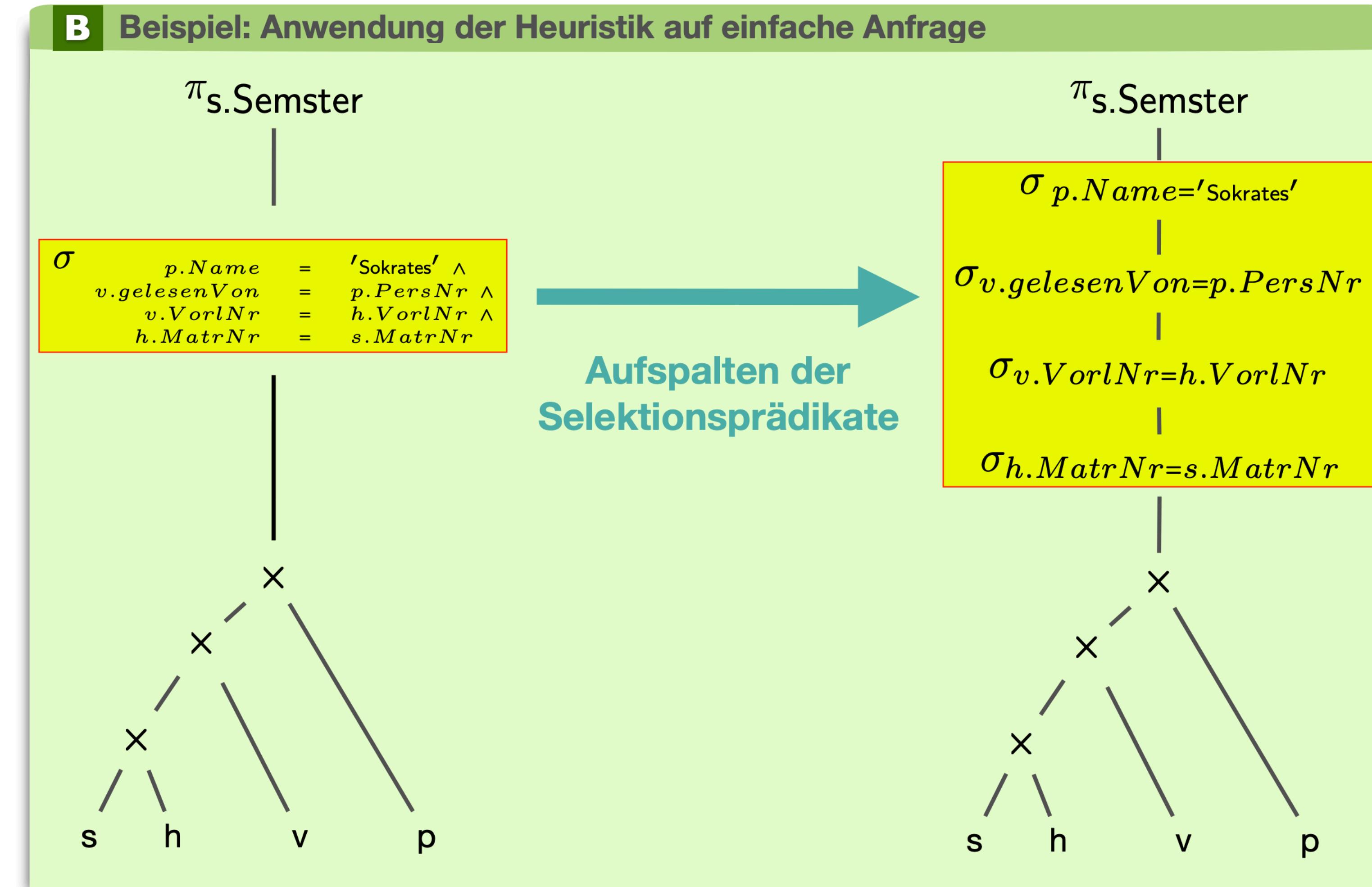
$$\sigma |$$

<i>p.Name</i>	=	'Sokrates'	$\wedge$
<i>v.gelesenVon</i>	=	<i>p.PersNr</i>	$\wedge$
<i>v.VorlNr</i>	=	<i>h.VorlNr</i>	$\wedge$
<i>h.MatrNr</i>	=	<i>s.MatrNr</i>	

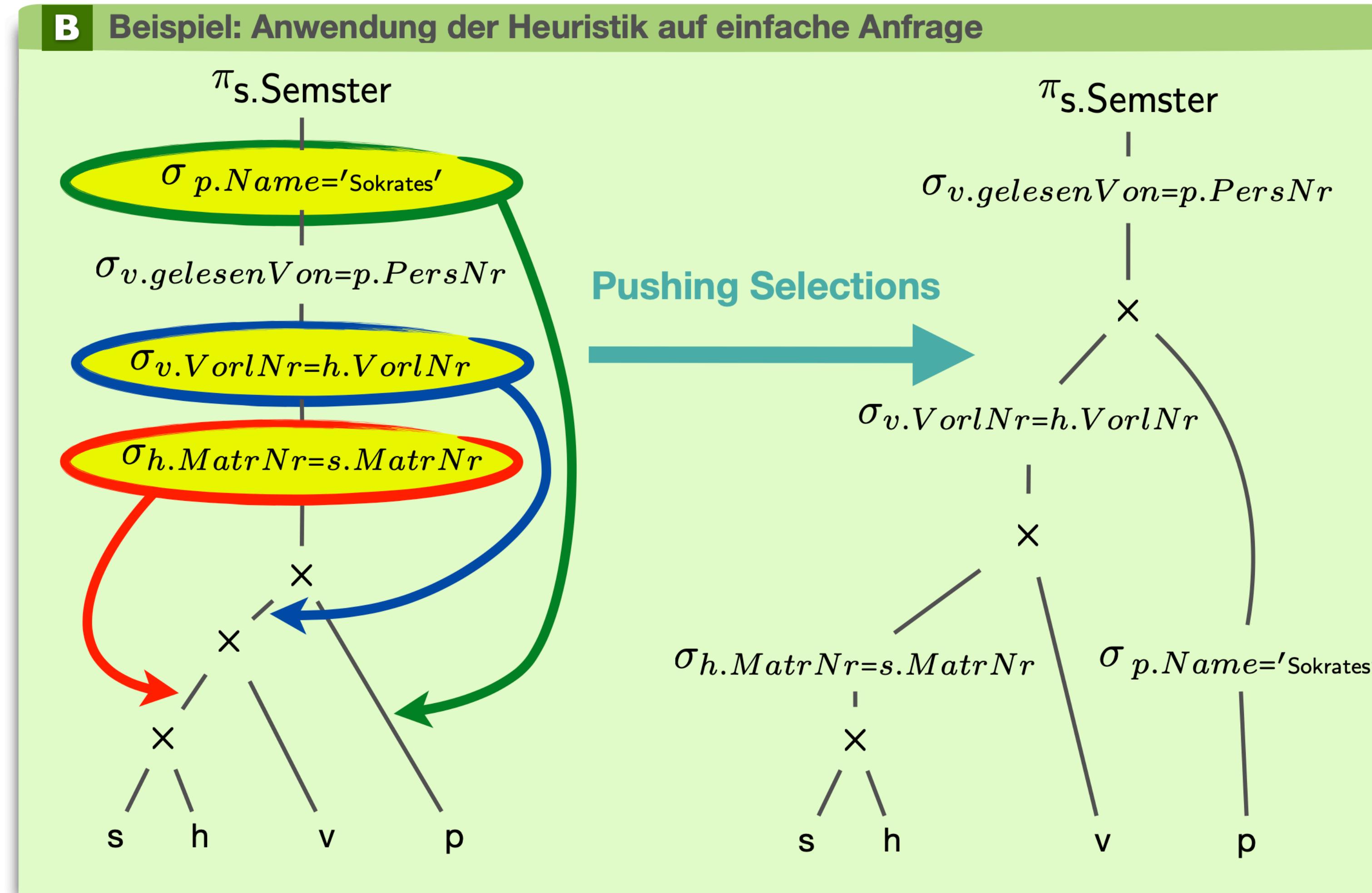
**750\*10<sup>15</sup>**

s      h      v      p  
 50.000    500.000    3.000    1.000

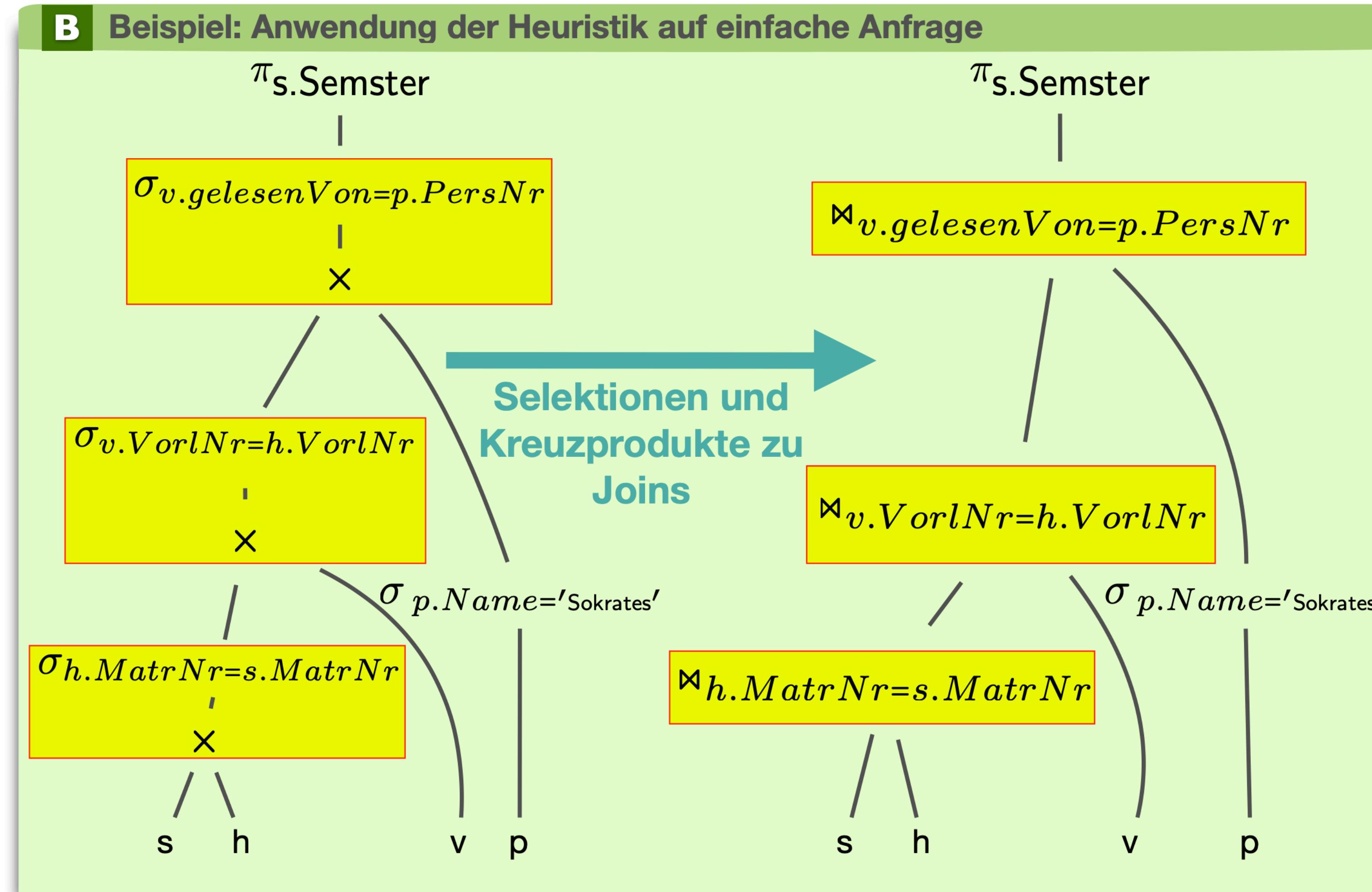
## Beispiel: Anfrageoptimierung



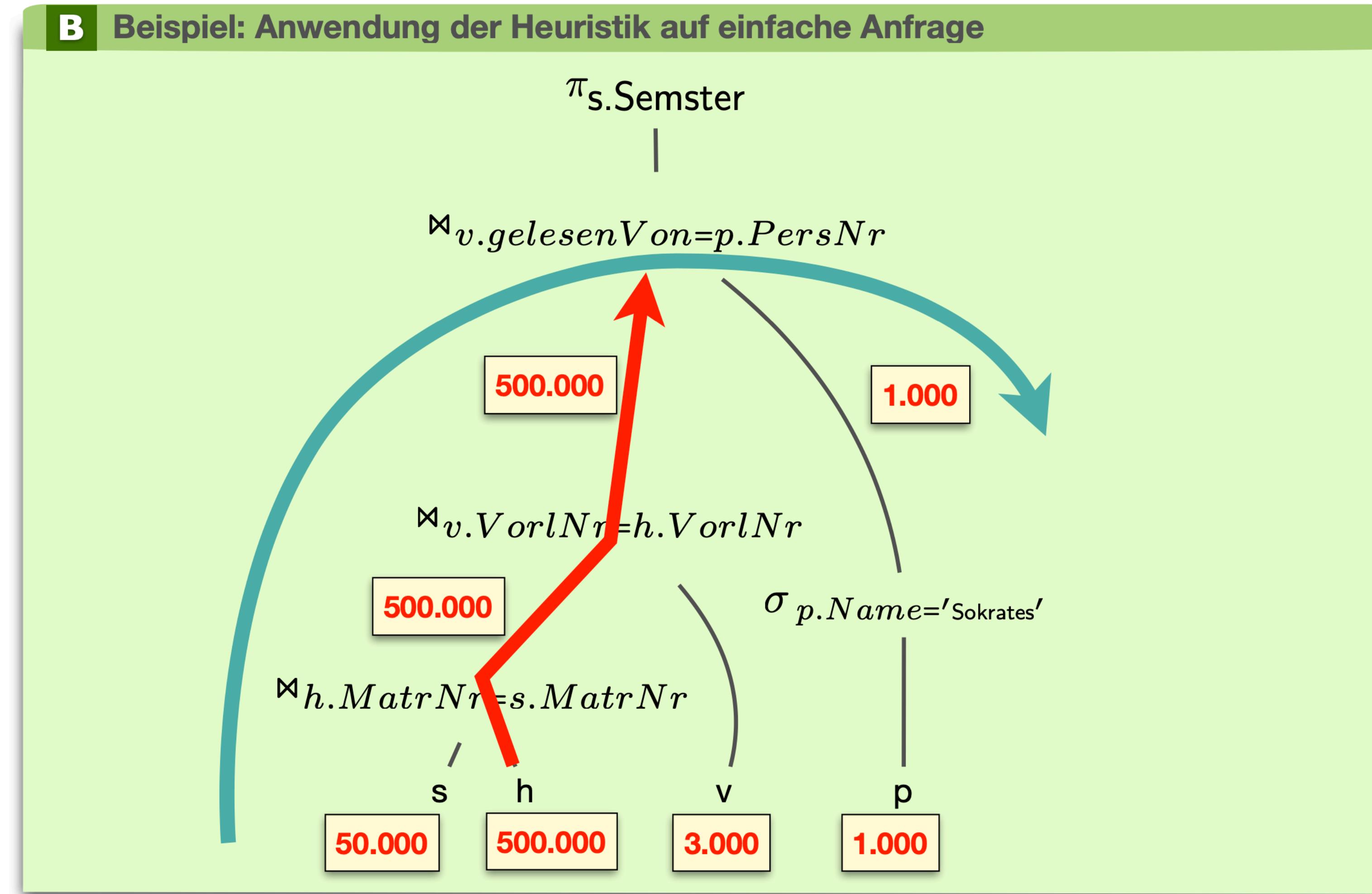
## Beispiel: Anfrageoptimierung



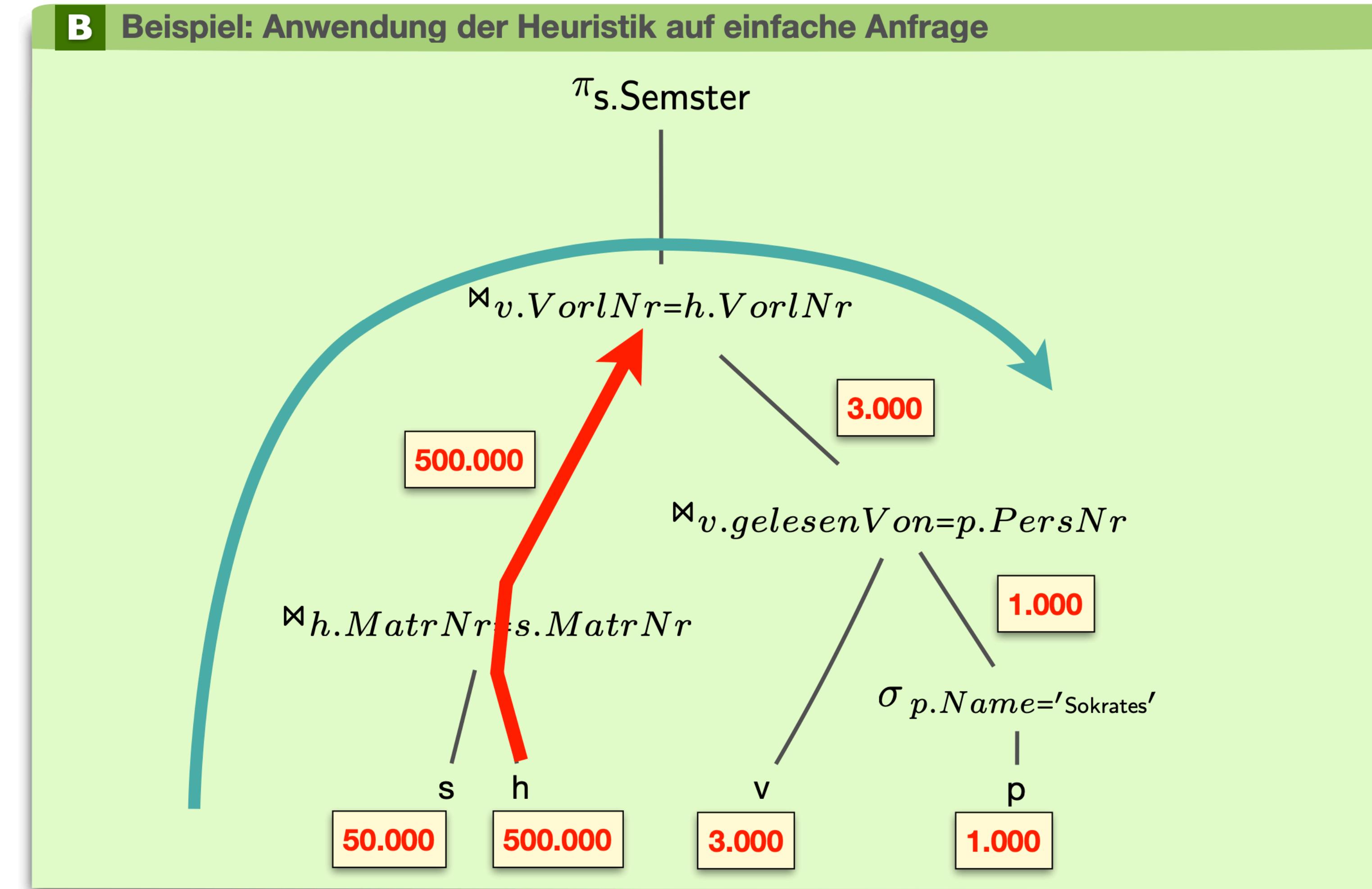
## Beispiel: Anfrageoptimierung



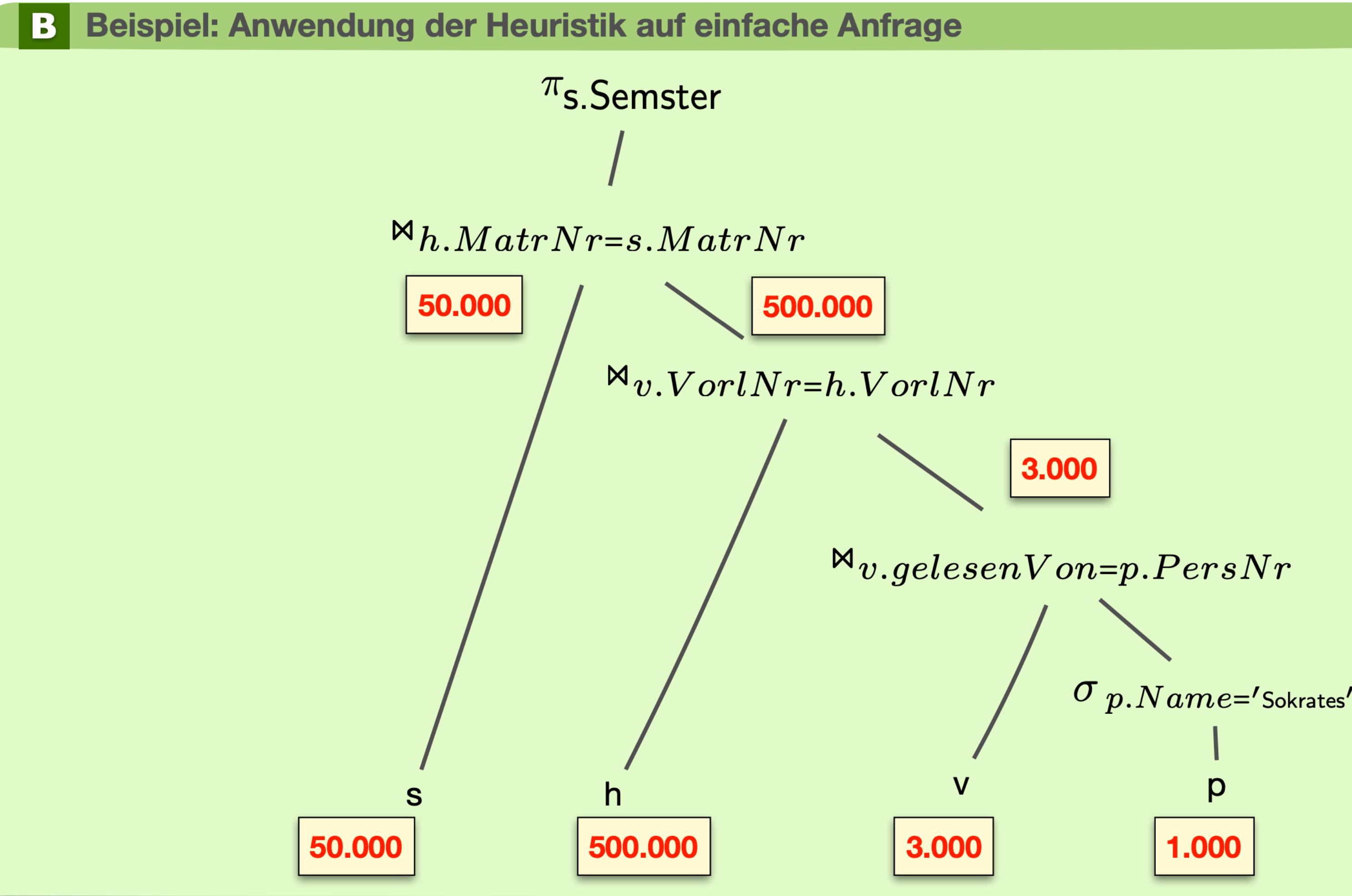
## Beispiel: Anfrageoptimierung



## Beispiel: Anfrageoptimierung



## Beispiel: Anfrageoptimierung



## Anfrageoptimierung

---

### Anmerkungen

- Optimierung ist Anwendungsfall-spezifisch.
- Optimierung nicht per Gefühl.
- Tools nutzen.
- Indizes nutzen.
- Normalformen dürfen in Frage gestellt werden.

# UNIT 0x0D

## ORM UND HIBERNATE

## Motivation

---

### Idee Object-Relational Mapping (ORM)

- Abbildung/Persistierung der Geschäftslogikobjekte in einem relationalen DBMS.

#### Q&A

- Vor- bzw. Nachteile?
- Wo liegen die Schwierigkeiten
- Wären andere DBMS besser?

### Bekannte ORM

- **Hibernate** – Ein Beispiel.
- Eine Liste aktueller ORMs für unterschiedliche Programmiersprachen findet sich schnell online.

## Hibernate

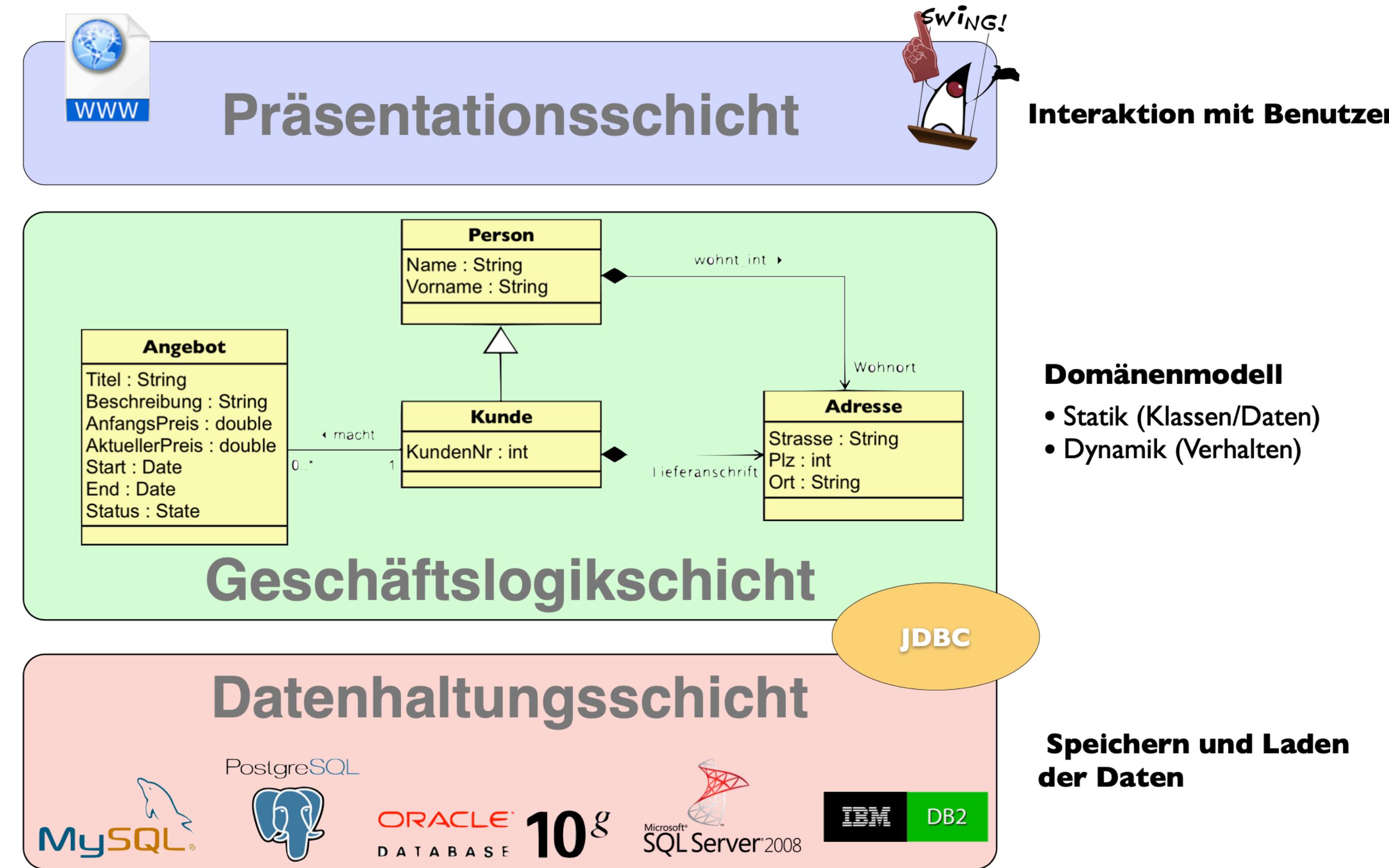
---

### Hintergrund

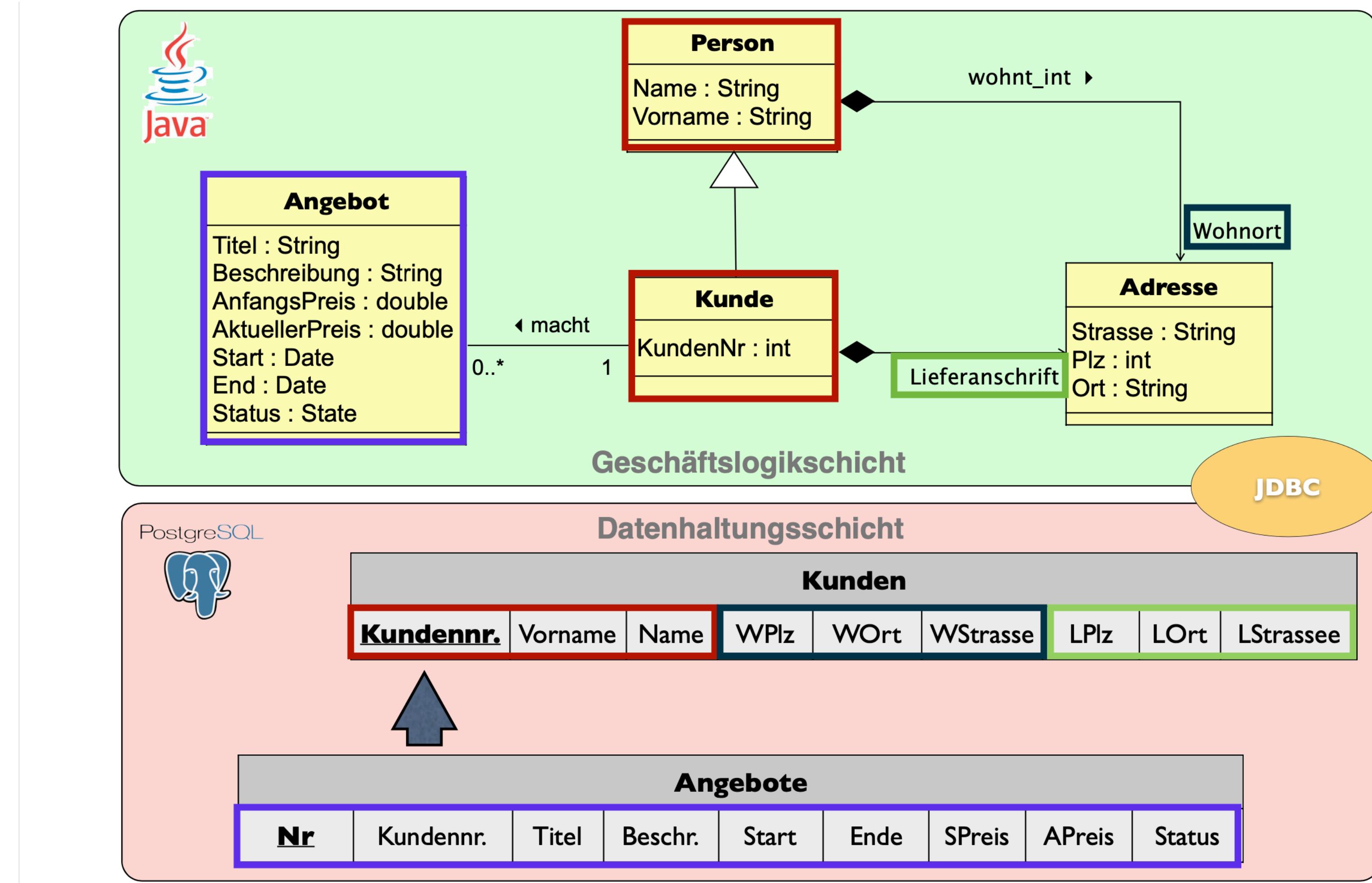
- Konfiguration
- Der Entity-Manager
- Lebenszyklus eines Entity-Objekts

## 3-Schichten-Modell

- Typisches Architekturmuster für JAVA-Applikationen



## 3-Schichten-Modell – Entwurf



## 3-Schichten-Modell – Implementierung

```

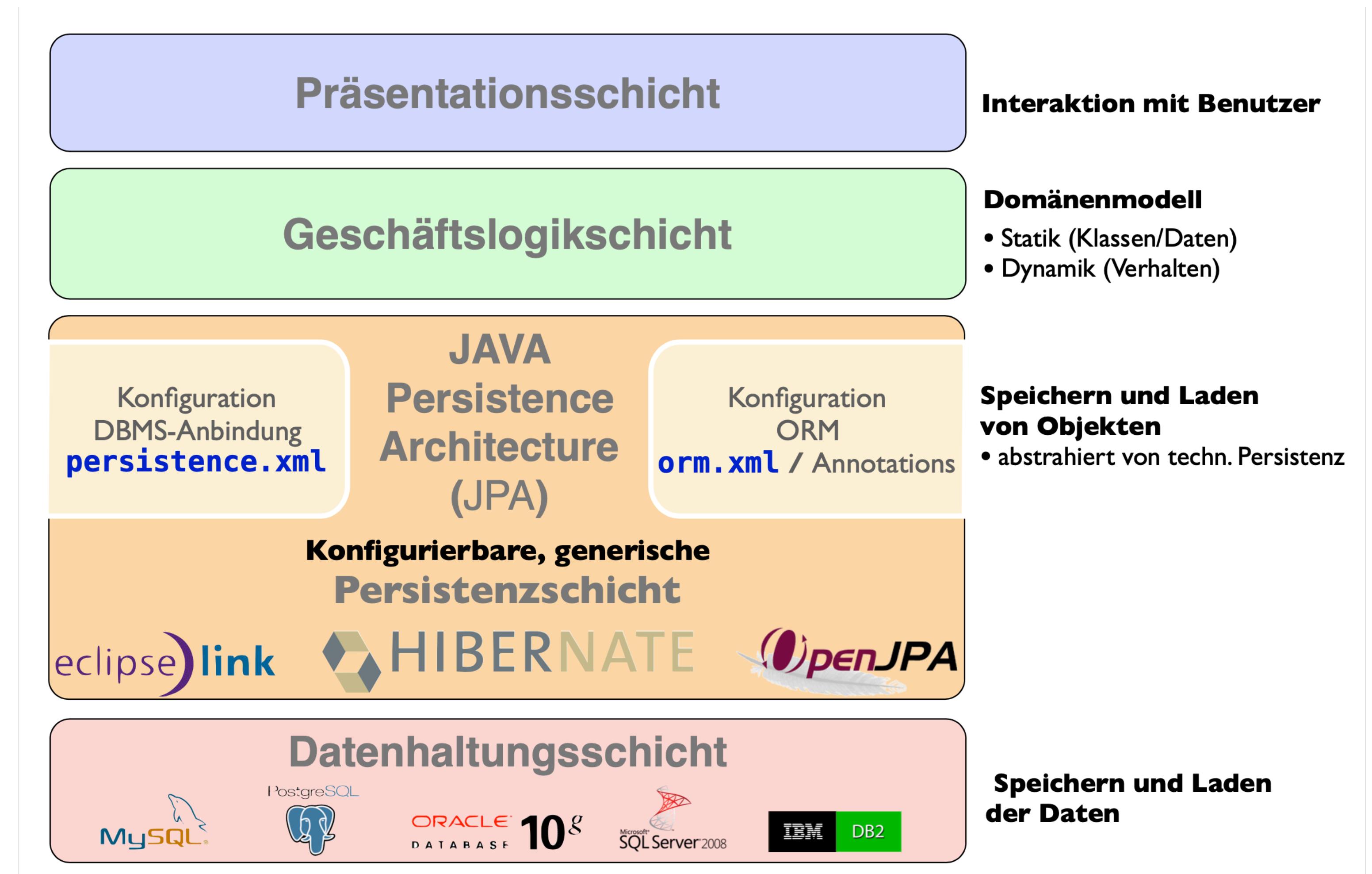
public class Kunde extends Person {
    protected Vector<Gegenstand> Angebote;
    protected Adresse LieferAnschrift;
    protected int KundenNr;
    /* Interaktion mit DBMS */
    public Kunde create(String Name,
                        String Vorname,
                        int KundenNr);
    public Kunde read(int KundenNr);
    public void update(Kunde p);
    public void delete(Kunde p);
}

```

ca. 750 Seiten

- **Aspekte der Datenhaltungsschicht jetzt in Geschäftslogik!**  
⇒ SQL in Geschäftslogik!  
Verstoss gegen Prinzip „**Separation of Concerns**“:
- **Konsequenz:**  
Pro **DBMS / Relationenmodell** eigene Geschäftslogikschicht  
Änderungen an eigentlicher Geschäftslogik bedeutet Änderung in mehreren Implementierungen der Geschäftslogikschicht  
⇒ Hohe Anforderungen an Konfigurationsmanagement

# JAVA Persistence Architecture (JPA)



## Hibernate – Konfiguration

---

### XML

- Pro: Trennung der „Geschäftslogik“ und „Datenhaltung“.
- Contra: Dateien können sehr groß und unübersichtlich werden.

### Annotationen

- Pro: Kürzer in der Formulierung, direkt am Code.
- Contra: Prinzip „Separation of Concerns“ verletzt, DBMS spezifisch.

# Hibernate – Konfiguration

## Beispiel

- **Konfiguration:**

Verzeichnis **META-INF** im Wurzelverzeichnis des **classpath** mit

- **persistence.xml** (Konfiguration der Persistenzschicht)
- **orm.xml** (optional - Beschreibung des Mappings)

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">

    <persistence-unit name="Persistence_Test">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <properties>
            <!-- SQL-Dialekt -->
            <property name="hibernate.dialect"
                value="org.hibernate.dialect.PostgreSQLDialect" />

            <!-- Verbindung zum DBMS -->
            <property name="hibernate.connection.url"
                value="jdbc:postgresql://localhost/TestDB" />
            <property name="hibernate.connection.driver_class"
                value="org.postgresql.Driver" />
            <property name="hibernate.connection.username" value="testuser" />
            <property name="hibernate.connection.password" value="123" />
        </properties>
    </persistence-unit>
</persistence>
```

# Hibernate – EntityManager

- **EntityManager:**

Verwaltet persistente Objekte (laden, speichern, ändern und löschen)

Stellt Persistenz-Kontext bereit (z.B. Cache)

Schnittstelle zur Persistenzschicht (z.B. Transaktionsmanagement)

```
EntityManagerFactory f = Persistence.createEntityManagerFactory("Persistence_Test");
EntityManager em      = f.createEntityManager();

/* Transaktion beginnen */
em.getTransaction().begin();
/* Laden von Objekten mithilfe von Primärschlüsselwerten */
Kunde k1 = em.find(Kunde.class, 1);
Kunde k2 = em.find(Kunde.class, 2);
/* Ändern persistenter Objekte -> Änderungen landen nach commit in DB */
k1.getAngebote().add( .... );
k1.getAngebote().get(2).setTitel( ... );

Person Jan = new Kunde( ... ); /* Neues Objekt anlegen ...
em.persist( Jan ); /* ... und unter Verwaltung des Entity-Managers stellen */
/* Löschen eines Objektes aus der Datenbank - k2 bleibt gültiges JAVA-Objekt */
em.remove( k2 );

/* Ändern: nach commit() werden alle Änderungen am Objektgraph persistent */
em.getTransaction().commit();

em.close(); f.close();
```

# Lebenszyklus persistenter Objekte

- Objekte können in vier Zuständen sein

**transient:** Objekte, die mit new erzeugt wurden.

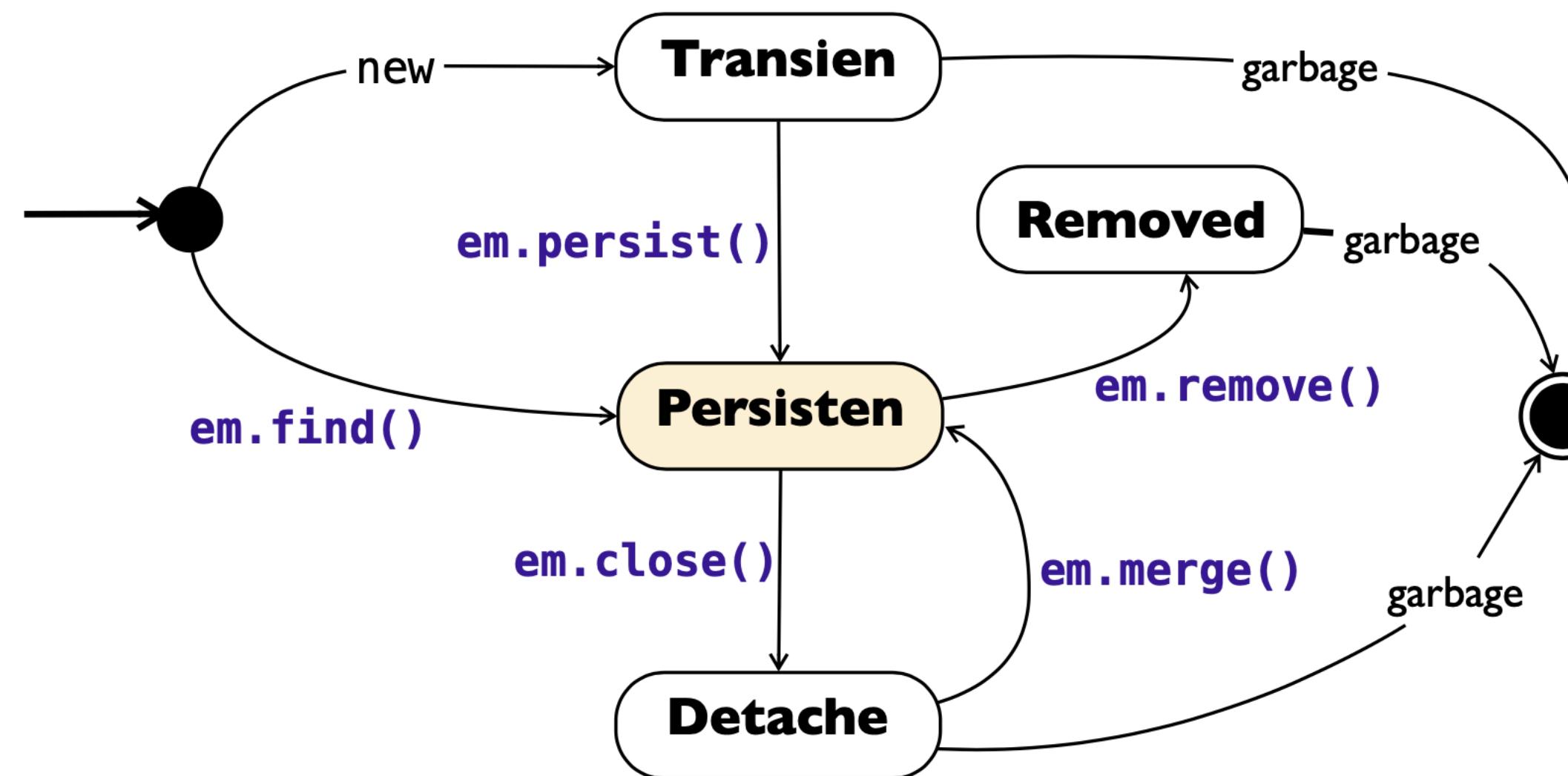
**persistent:** Objekte, die eindeutig einer Zeile in einer DB-Tabelle zugeordnet sind.

Änderungen am Objekt werden in der Datenbank sichtbar (und umgekehrt).

**detached:** Objekte, die eindeutig einer Zeile in einer DB-Tabelle zugeordnet sind.

Änderungen am Objekt werden **nicht** in Datenbank sichtbar

**removed:** Objekte, die aus Datenbank entfernt wurden.



In Anlehnung an: Christian Bauer, Gavin King: JAVA Persistence With Hibernate, S. 386

## ORM – Mapping

---

### Statisches Mapping

- Klassen
- Vererbung
- Aggregation, Komposition und Assoziation

### Dynamisches Mapping

- Constraints und Trigger
- Performance: Dirty Checking, Lazy Fetching und Caching
- hier nicht behandelt.

## ORM – Grundvoraussetzungen

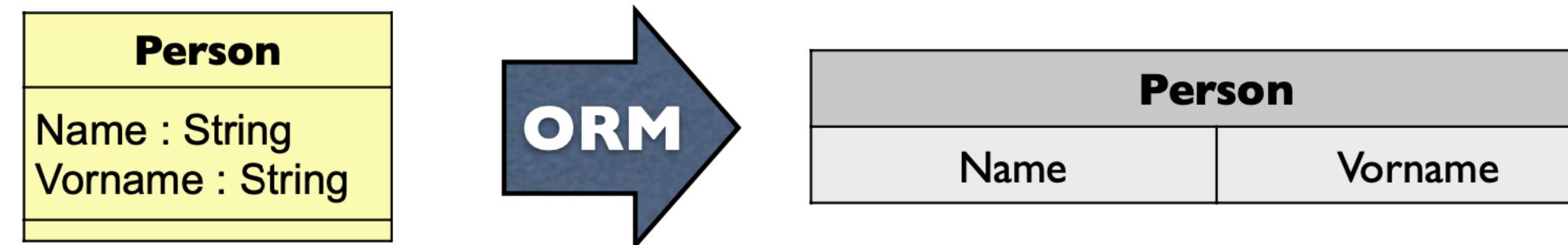
---

### Entity-Klassen

- sind Klassen, deren Objekte durch die Persistenzschicht verwaltet werden können.
- Muss-Bedingungen:
  - entweder Annotation @Entity oder <entity>-Eintrag in orm.xml-Konfiguration.
  - Top-Level-Klasse (keine inner-class, kein interface und kein enum).
  - Default-Konstruktor (Konstruktor ohne Argumente, public oder protected).
  - Klasse und Attribute dürfen nicht final sein.
  - Primärschlüssel muss definiert sein.
- Kann-Bedingungen:
  - Implementierung des Interfaces „Serializable“.
  - Get- und Set-Methoden für zur speichernde Attribute.

# Klassen – Basics

- Klassen werden direkt auf Tabellen abgebildet, wenn**  
sie von keiner anderen Klasse (ausser Object) abgeleitet sind  
sie nur primitive oder serialisierbare Attribute haben (keine Collections)



- Ein Objekt entspricht immer einer Zeile in einer Tabelle**
- Klassen- und Attributnamen werden übernommen**
- Typen werden automatisch zugeordnet**  
Abhängig vom eingestellten SQL-Dialekt

## B Type-Mapping in PostgreSQL

Java	PostgreSQL
<code>byte, short</code>	<code>int2</code>
<code>int</code>	<code>int4</code>
<code>long</code>	<code>int8</code>
<code>float</code>	<code>float4</code>
<code>double</code>	<code>float8</code>

Java	PostgreSQL
<code>boolean</code>	<code>bool</code>
<code>String</code>	<code>varchar(255)</code>
<code>Integer</code>	<code>int4</code>
<code>Double</code>	<code>float8</code>
<code>Serializable</code>	<code>bytea</code>

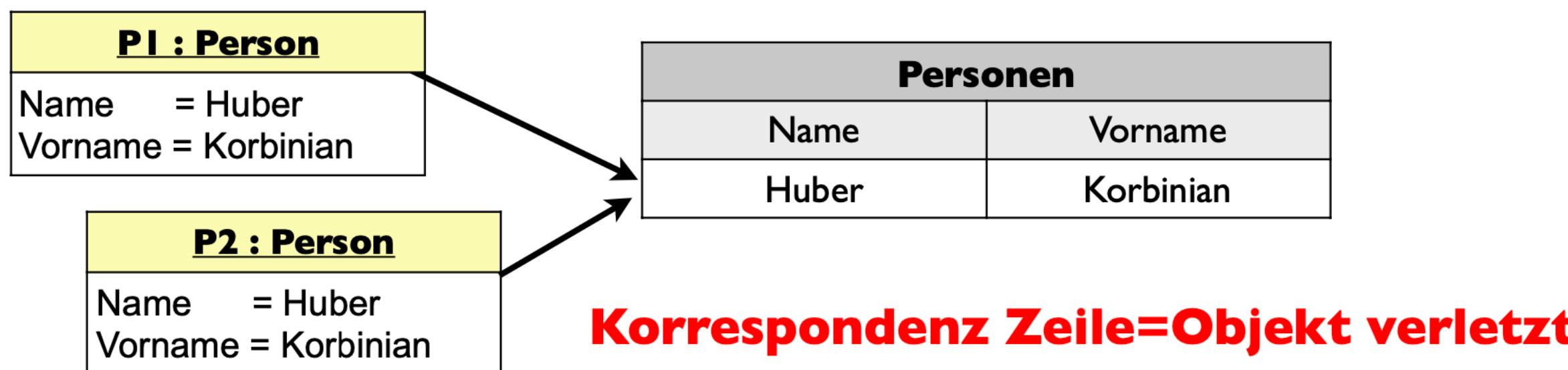
## Klassen - Objektidentität

- **Problem: Objektidentität**

JAVA: Verschiedene Objekte dürfen gleiche Attributwerte haben

DBMS: Zeilen mit identischen Attributwerten nicht erlaubt

Daher: persistente Objekte mit gleichen Attributwerten nicht mehr unterscheidbar



- **Lösung: definiere Attribute, die Objekt eindeutig auszeichnen**  
synthetische ID-Schlüssel (z.B. Sequenzen) mit neuem Attribut (z.B. int id)  
fachlicher ID-Schlüssel (auch zusammengesetzt - dazu später)

## Beispiel: ID-Schlüssel

### B Entity-Klasse beschrieben durch Annotationen

```
@Entity  
public class Person {  
  
    @Id  
    int Id;  
  
    String Nachname;  
    String Vorname;  
  
    public Person() {}  
}
```



Person		
Id	Vorname	Nachname

### B Entity-Klasse beschrieben in orm.xml

```
<entity-mappings  
    xmlns="http://java.sun.com/xml/ns/persistence/orm"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm  
    http://java.sun.com/xml/ns/persistence/orm_1_0.xsd" version="1.0">  
    <package>model</package>  
    <entity class="Person">  
        <attributes>  
            <id name="id" />  
        </attributes>  
    </entity>  
</entity-mappings>
```

# Klassen – Synthetische ID-Schlüssel

- Können von **Applikation** verwaltet werden

**Dann:** neben @Id keine weiteren Angaben nötig!

**Vorsicht!** Applikation ist u.U. für Erzeugung von Schlüsseln verantwortlich

Mit new erzeugtes Objekt kann nur dann „persistent“ werden, wenn @Id noch nicht in DB!

## B Probleme mit selbstverwalteten Schlüsseln

```
EntityManager em = ...;
em.getTransaction().begin();

/* Ok! Id 2 noch nicht in Tabelle */
Person Heidi = new Person("Heidi","von der Alm");
Heidi.id = 2;
em.persist( Heidi );

/* Fehler! Id 1 schon in Verwendung ... */
Person Sepp = new Person("Sepp","Huber");
Sepp.id = 1;
em.persist( Sepp );

em.getTransaction().commit();
```

Person		
ID	Vorname	Nachname
1	Hein	Bollo

- **Besser: Verwaltung durch DBMS oder Hibernate**

**@GeneratedValue:** mit Strategien Identity, Sequence, Table und Auto

## Klassen - Synthetische ID-Schlüssel

- Können von **DBMS** verwaltet werden

DBMS erzeugt Schlüssel und sorgt für Eindeutigkeit (z.B. Auto-Increment-Felder)

Dann: ID-Attribut als `@GeneratedValue` mit Strategie `IDENTITY` markieren

### B Datentyp „serial“ für synthetische Schlüssel mit PostgreSQL

Effekt der Verwendung von `serial`:

```
CREATE SEQUENCE person_id_seq;

CREATE TABLE Person (
    Id integer DEFAULT nextval(person_id_seq) NOT NULL,
    Vorname varchar(255),
    Nachname varchar(255),
    CONSTRAINT pk_Person PRIMARY KEY (Id)
);
```

```
@Entity
public class Person {
    @Id /* Nur kompatibel für echten „IDENTITY“-Typ in DB! */
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    int Id;
    String Nachname;
    String Vorname;

    public Person() {}
}
```

# Klassen - Synthetische ID-Schlüssel

- Können von **Hibernate** verwaltet werden

Hibernate erzeugt Schlüssel und sorgt für Eindeutigkeit

Strategie wird mit Annotation `@GeneratedValue` z.B. auf SEQUENCE oder TABLE gesetzt

## B Synthetischer Schlüssel mittels SEQUENCE

Ohne Verwendung von `serial`:

```
CREATE SEQUENCE person_id_seq;

CREATE TABLE Person (
    Id integer NOT NULL, /* Beachte! DEFAULT-Angabe fehlt */
    Vorname varchar(255),
    Nachname varchar(255),
    CONSTRAINT pk_Person PRIMARY KEY (Id)
);
```

```
@Entity
public class Person {
    @Id
    @SequenceGenerator(name="person_seq", sequenceName="person_id_seq")
    @GeneratedValue(strategy=GenerationType.SEQUENCE, generator= "person_seq")
    int Id;
    String Nachname;
    String Vorname;

    public Person() {}
}
```

# Klassen - Namen, Typen und transiente Attribute

- **Standardmäßig (ohne Annotation)**

Tabellenname entspricht Klassenname; Spaltenname entspricht Attributname

Typzuordnung automatisch (siehe Folie zu Basics)

Alle Attribute werden abgebildet - **Ausnahme:** Markierung mit @Transient

## B Mapping von Typen / Namen für Felder und Namen für Klassen

```
CREATE DOMAIN german_plz AS text  
CONSTRAINT german_plz_check CHECK ((VALUE ~ '^\d{5}$'::text));
```

Kunde			
<b>Id:serial</b>	Vorname:text	Nachname:text	<b>WPlz : german_plz</b>

```
@Entity @Table(name="Kunde")/* Name-Map! */  
public class Person {  
    @Id @GeneratedValue  
    int Id;  
    /* Name- und Type-Map! Plz kommt aus Feld WPlz */  
    @Column(name="WPlz", length=5, columnDefinition="german_plz")  
    String Plz;  
    String Nachname;  
    String Vorname;  
    /* Nur zur Laufzeit relevant */  
    @Transient  
    boolean activeTransaction;  
    public Person() {}  
}
```

## Klassen – Aufgespaltene Relationen

- Entities können Ergebnis eines Joins repräsentieren

Join-Attribut muss definiert sein (nur 1:1 Beziehung)

Für Attribute muss ggf. angeben werden, aus welcher Tabelle sie kommen

**B | 1:1 Beziehung mit @SecondaryTable**

Person		
<b>Id</b>	Vorname	Nachname

Adresse				
<b>Id</b>	Plz	Ort	Strasse	Person

**Fremdschlüssel**

```
@SecondaryTable(name="Adresse",
    pkJoinColumns=@PrimaryKeyJoinColumn(name="Person"))
public class Person {
    @Id @GeneratedValue
    int Id;
    String Vorname;
    String Nachname;
    @Column(table="Adresse")
    String Plz;
    @Column(table="Adresse")
    String Ort;
    @Column(table="Adresse")
    String Strasse;
    public PersonA() {}
}
```

**Vorsicht!**  
Geht so nur für 1:1 Beziehungen

# Vererbung – Einfachster Fall

- **Der einfachste Fall:**

Oberklasse nur zur Modellierung (z.B. abstrakte Klasse)

Objekte sollen **nicht** gespeichert werden

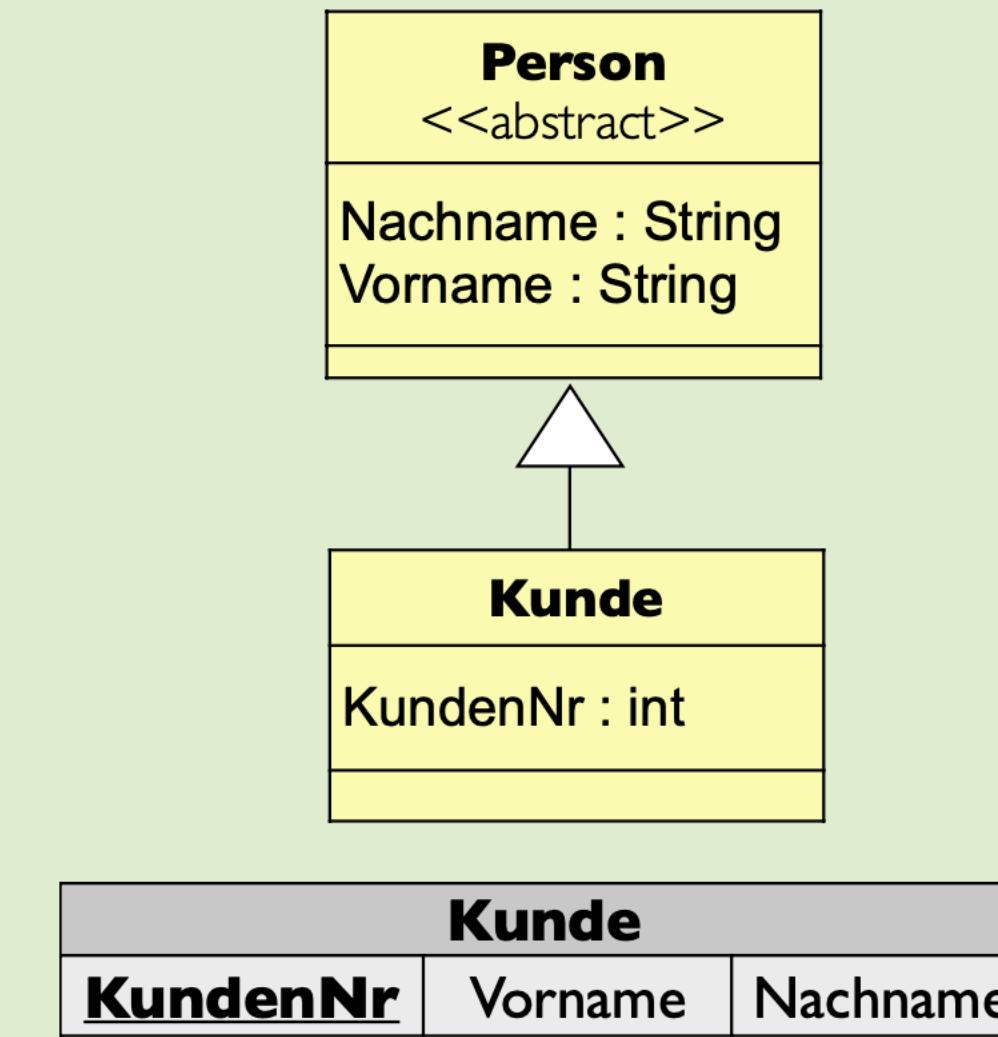
**Dann:** markiere Oberklasse mit `@MappedSuperclass` und Attribute wie gewohnt

## B Vererbung ohne Speicherung von Objekten der Superklasse

```
@MappedSuperclass
public abstract class Person {
    /* Beachte: @Id nicht erforderlich! */
    String Nachname;
    String Vorname;
}

/* „Normale“ Entity-Klasse */
@Entity
public class Kunde extends Person {
    @Id @GeneratedValue
    int KundenNr;

    public Kunde() {}
}
```



## Vererbung – Strategien

- **Strategie wird in Wurzel der Entity-Hierarchie gewählt**

Annotation hierzu: **@Inheritance**

Kann in abgeleiteten **nicht** mehr gewechselt werden

Darüber nur **@MappedSuperclass** erlaubt

**Entity-Hierarchie ist  
Auschnitt aus  
Klassenhierarchie**

- **Drei Varianten pro Klassenhierarchie**

**TABLE\_PER\_CLASS** Jede Klasse in eine Tabelle  
(**mit** geerbten Attributen)

**JOINED** Jede Klasse in eine Tabelle  
(**ohne** geerbte Attribute, Fremdschlüsselbeziehung zu Oberklasse)

**SINGLE\_TABLE** Alle Klassen in einer Tabelle  
(Diskriminator-Spalte zur Bestimmung des Objekt-Typs)

# Polymorphe Abfragen

- **Polymorphe Abfragen**

Polymorphe Abfragen berücksichtigen Objekte von Kindklassen

**B Polymorphe „select“**

```

classDiagram
    class Person {
        Id : int
        Name : String
        Vorname : String
    }
    class Freelancer {
        Available : bool
    }
    class Angestellter {
        Gehalt : double
        Eintritt : Date
    }
    Freelancer <|-- Person
    Angestellter <|-- Person
  
```

The diagram illustrates a polymorphic query scenario. It shows a base class **Person** with attributes **Id**, **Name**, and **Vorname**. Two subclasses inherit from it: **Freelancer** (with attribute **Available**) and **Angestellter** (with attributes **Gehalt** and **Eintritt**). A polymorphic query **select p from Person p;** is demonstrated, which returns three results corresponding to the objects Hein, Erik, and Jan.

<b>Hein : Person</b>	
<b>Id</b>	= 1
<b>Name</b>	= Bollo
<b>Vorname</b>	= Hein

<b>Erik : Freelancer</b>	
<b>Id</b>	= 2
<b>Name</b>	= Roth
<b>Vorname</b>	= Erik
<b>Available</b>	= true

<b>Jan : Angestellter</b>	
<b>Id</b>	= 3
<b>Name</b>	= Cux
<b>Vorname</b>	= Jan
<b>Gehalt</b>	= 3217,55
<b>Eintritt</b>	= 1.5.2003

**select p from Person p;**

<b>:Person</b>	
<b>Id</b>	= 1
<b>Name</b>	= Bollo
<b>Vorname</b>	= Hein

<b>:Person</b>	
<b>Id</b>	= 2
<b>Name</b>	= Roth
<b>Vorname</b>	= Erik

<b>:Person</b>	
<b>Id</b>	= 3
<b>Name</b>	= Cux
<b>Vorname</b>	= Jan

# Vererbung – TABLE\_PER\_CLASS

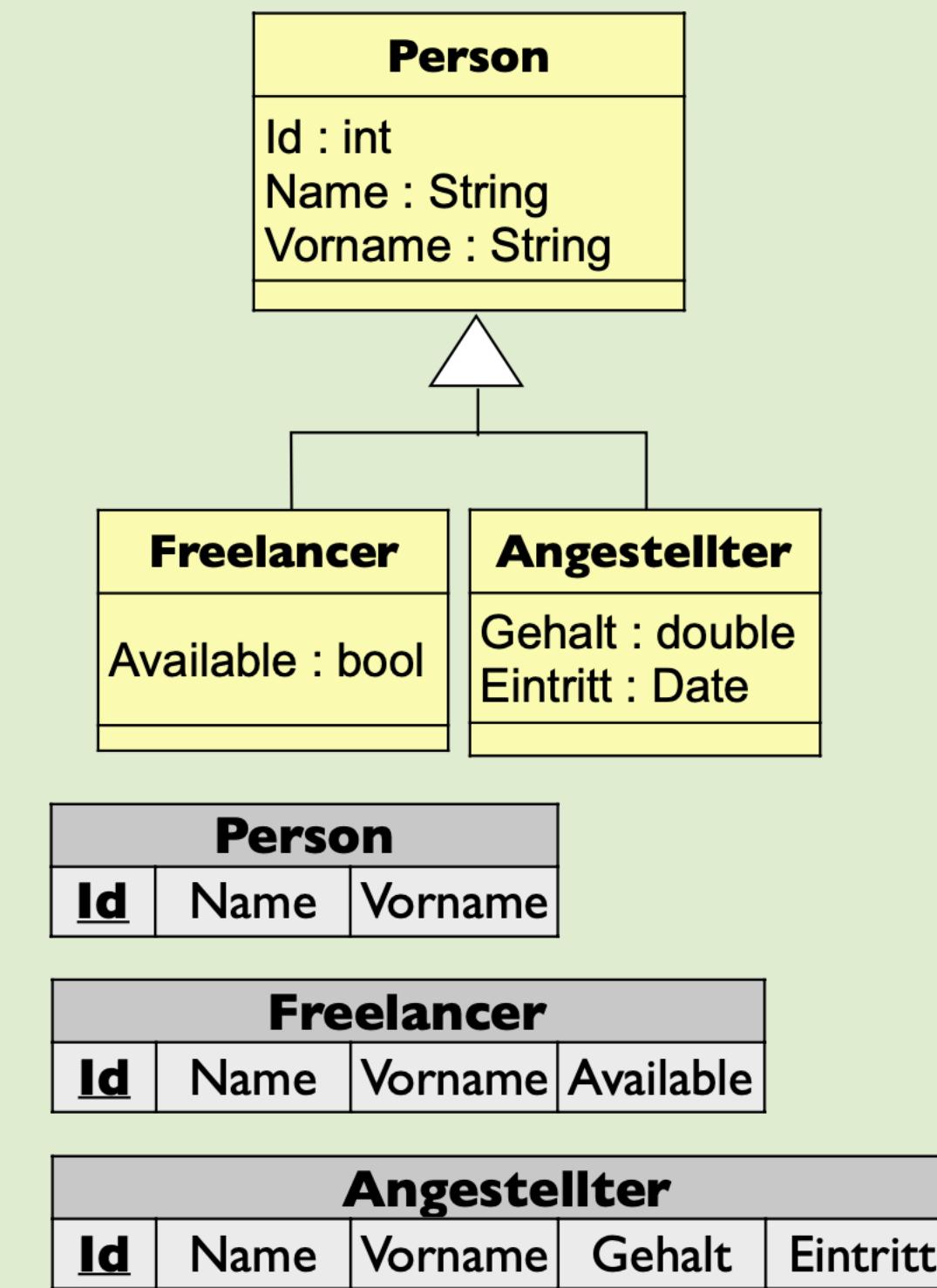
## B Anwendung von TABLE\_PER\_CLASS

```

@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Person {
    @Id @GeneratedValue
    int id;
    String Name;
    String Vorname;
    public Person() {}
}
/* Beachte: @Id wird geerbt */
@Entity
public class Freelancer extends Person {
    boolean Available;
    public Freelancer() {}
}

@Entity
public class Angestellter extends Person {
    double Gehalt;
    Date Eintritt;
    public Angestellter() {}
}

```



- Vererbung datenbankseitig nicht erkennbar

# Polymorphes Select – TABLE\_PER\_CLASS

**B Polymorphe Anfrage - TABLE\_PER\_CLASS**

<b>Hein : Person</b>	<b>Person</b>
Id = 1 Name = Bollo Vorname = Hein	Id Name Vorname 1 Bollo Hein

<b>Jan : Angestellter</b>	<b>Freelancer</b>
Id = 3 Name = Cux Vorname = Jan Gehalt = 3217,55 Eintritt = 1.5.2003	Id Name Vorname Available 2 Roth Erik true

<b>Erik : Freelancer</b>	<b>Angestellter</b>
Id = 2 Name = Roth Vorname = Erik Available = true	Id Name Vorname Gehalt Eintritt 3 Cux Jan 3217,55 152.003

```
select p from Person p;
```

```

SELECT Id,Name,Vorname FROM Person UNION
SELECT Id,Name,Vorname FROM Freelancer UNION
SELECT Id,Name,Vorname FROM Angestellter;

```

## Bewertung – TABLE\_PER\_CLASS

Person		
<b>Id</b>	Name	Vorname
1	Bollo	Hein

Freelancer			
<b>Id</b>	Name	Vorname	Available
2	Roth	Erik	true

Angestellter				
<b>Id</b>	Name	Vorname	Gehalt	Eintritt
3	Cux	Jan	3217,55	152.003

- **Polymorphes select und update**
  - Select: langsam, drei selects (u.U. mit Projektionen) oder ein select mit union
  - Update: langsam, mehrere Tabellen betroffen
- **Monomorpher Zugriff**
  - schnell, alle Daten in einer Tabelle
- **Änderungen am Design**
  - Neue Klasse: unproblematisch, falls Blatt in Hierarchie
  - Neues Attribut: Tabellen aller Kindklassen müssen geändert werden
- **Sonstiges**
  - passt gut zur Philosophie „Eine Entity-Klasse entspricht einer Tabelle“

# Vererbung – JOINED

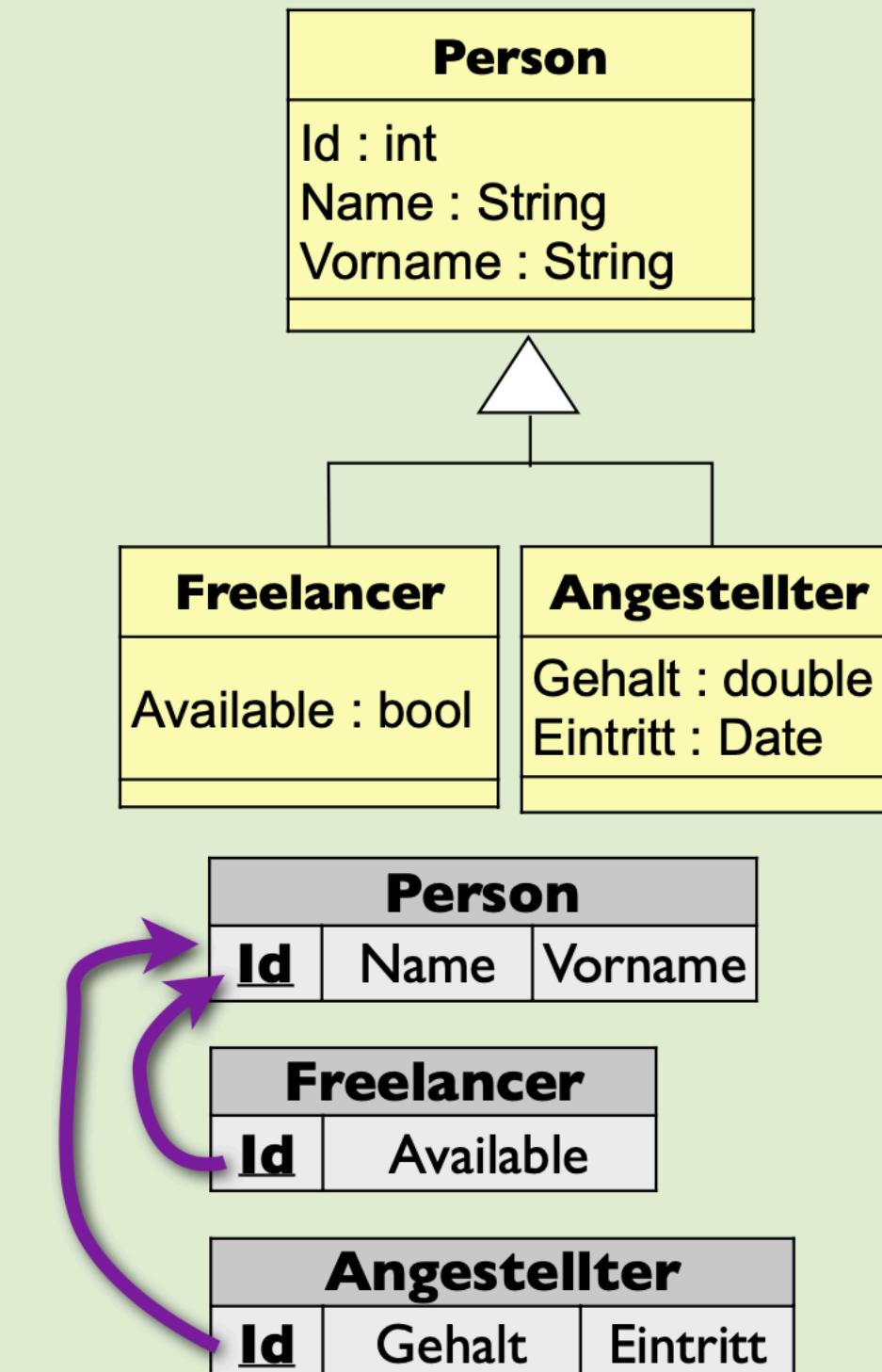
## B Anwendung von JOINED

```

@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class Person {
    @Id @GeneratedValue
    int id;
    String Name;
    String Vorname;
    public Person() {}
}
/* Beachte: @Id wird geerbt */
@Entity
public class Freelancer extends Person {
    boolean Available;
    public Freelancer() {}
}

@Entity
public class Angestellter extends Person {
    double Gehalt;
    Date Eintritt;
    public Angestellter() {}
}

```



- Vererbung datenbankseitig als Fremdschlüssel-Beziehung

# Polymorphes Select - JOINED

**B Polymorphe Anfrage - JOINED**

<b>Hein : Person</b>	<b>Jan : Angestellter</b>	<b>Erik : Freelancer</b>	<b>Person</b>	<b>Freelancer</b>	<b>Angestellter</b>
<b>Id</b> = 1 <b>Name</b> = Bollo <b>Vorname</b> = Hein	<b>Id</b> = 3 <b>Name</b> = Cux <b>Vorname</b> = Jan <b>Gehalt</b> = 3217,55 <b>Eintritt</b> = 1.5.2003	<b>Id</b> = 2 <b>Name</b> = Roth <b>Vorname</b> = Erik <b>Available</b> = true	<b>Id</b> Name Vorname 1 Bollo Hein 2 Roth Erik 3 Cux Jan	<b>Id</b> 2 true	<b>Id</b> Gehalt Eintritt 3 3217,55 152.003

**select p from Person p;**      **SELECT Id,Name,Vorname FROM Person;**

## Bewertung – JOINED

<b>Person</b>		
<b>Id</b>	Name	Vorname
1	Bollo	Hein
2	Roth	Erik
3	Cux	Jan

<b>Freelancer</b>	
<b>Id</b>	Available
2	true

<b>Angestellter</b>		
<b>Id</b>	Gehalt	Eintritt
3	3217,55	152.003

- **Polymorphes select und update**
  - Select: für Wurzel ein select, ansonsten Join mit allen Oberklassen
  - Update: langsam, mehrere Tabellen betroffen
- **Monomorpher Zugriff**
  - langsam, Join mit allen Oberklassen
- **Änderungen am Design**
  - Neue Klasse: unproblematisch, neue Tabelle
  - Neues Attribut: nur eine Tabelle betroffen
- **Sonstiges**
  - besonders geeignet für wechselnde Designs, polymorphe selects

# Vererbung – SINGLE\_TABLE

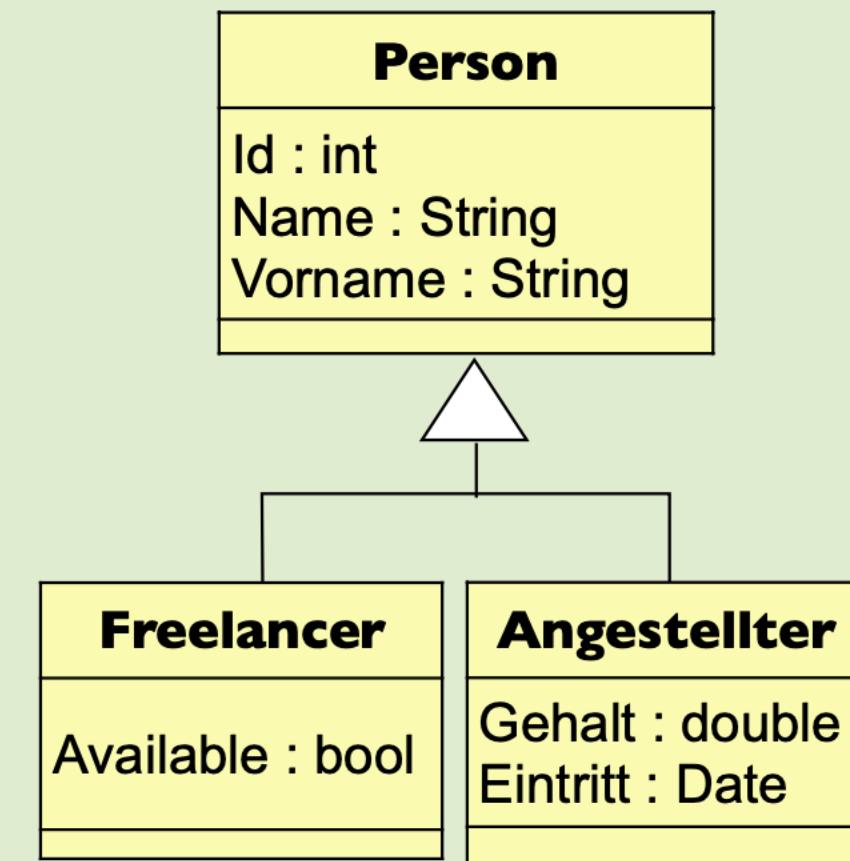
## B Anwendung von SINGLE\_TABLE

```

@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(
    name = "Art",
    discriminatorType = DiscriminatorType.STRING)
@DiscriminatorValue("P")
public class Person {
    @Id @GeneratedValue
    int id;
    String Name;
    String Vorname;
    public Person() {}
}
/* Beachte: @Id wird geerbt */
@Entity
@DiscriminatorValue("F")
public class Freelancer extends Person {
    boolean Available;
    public Freelancer() {}
}

@Entity
@DiscriminatorValue("A")
public class Angestellter extends Person {
    double Gehalt;
    Date Eintritt;
    public Angestellter() {}
}

```



Person						
<b>Id</b>	<b>Art</b>	Name	Vorname	Gehalt	Eintritt	Available

# Polymorphes Select – SINGLE\_TABLE

**B Polymorphe Anfrage - TABLE\_PER\_CLASS**

<b>Person</b>						
<b>Id</b>	<b>Art</b>	<b>Name</b>	<b>Vorname</b>	<b>Gehalt</b>	<b>Eintritt</b>	<b>Available</b>
1	P	Bollo	Hein			
2	F	Roth	Erik			true
3	A	Cux	Jan	3217,55	15.2.2003	

**Hein : Person**

Id = 1
Name = Bollo
Vorname = Hein

**Jan : Angestellter**

Id = 3
Name = Cux
Vorname = Jan
Gehalt = 3217,55
Eintritt = 1.5.2003

**Erik : Freelancer**

Id = 2
Name = Roth
Vorname = Erik
Available = true

**select p from Person p;**

**SELECT Id,Name,Vorname FROM Person;**

## Bewertung – SINGLE\_TABLE

Person						
<b>Id</b>	Art	Name	Vorname	Gehalt	Eintritt	Available
1	P	Bollo	Hein			
2	F	Roth	Erik			true
3	A	Cux	Jan	3217,55	15.2.200	

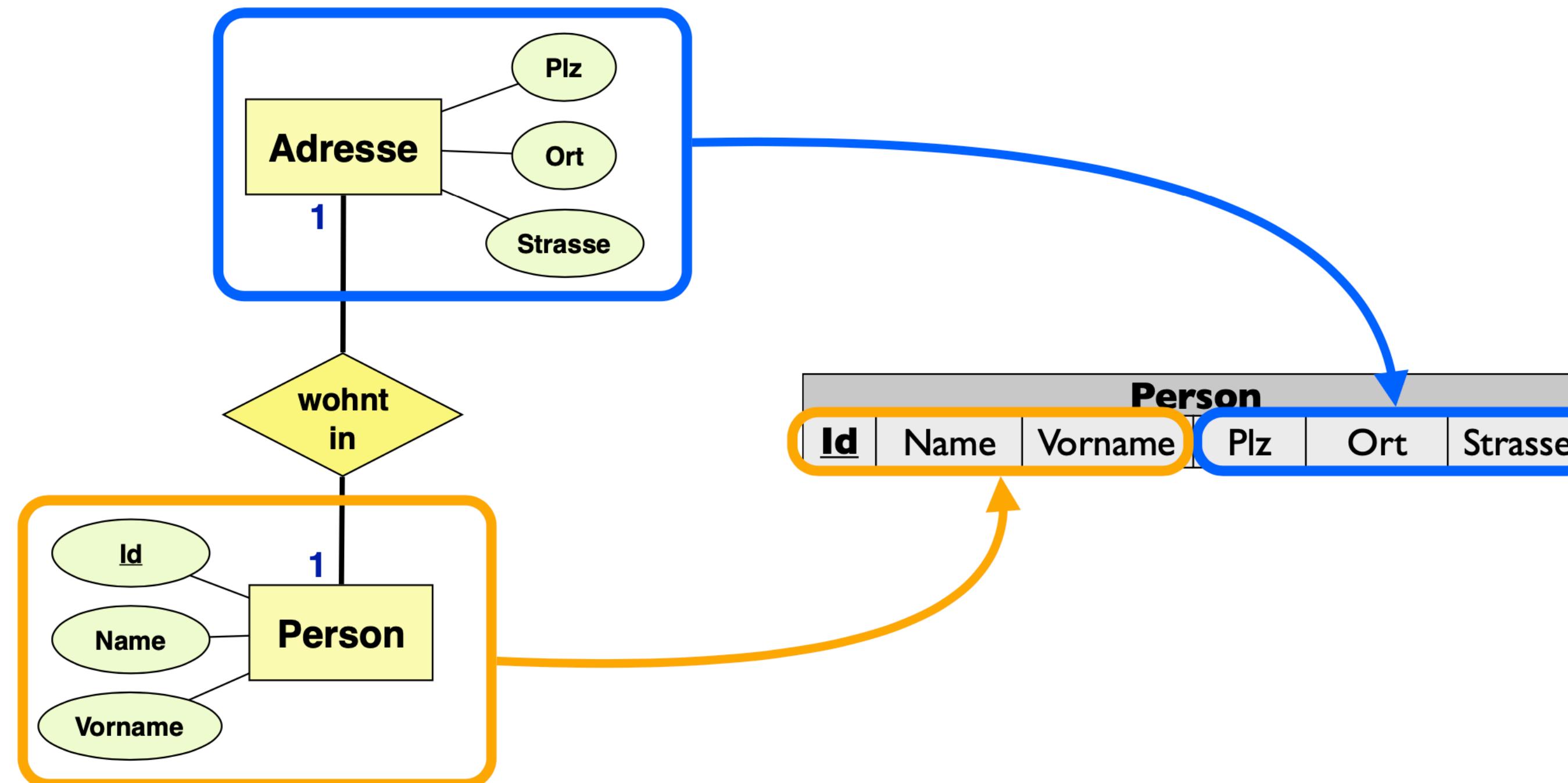
- **Polymorphes select und update**
  - Select: schnell, ein select mit Projektion
  - Update: schnell, eine Tabelle
- **Monomorpher Zugriff**
  - schnell: eine Tabelle
- **Änderungen am Design**
  - Neue Klasse: Umorganisation einer u.U. großen Tabelle
  - Neues Attribut: Umorganisation einer u.U. großen Tabelle
- **Sonstiges**
  - Attribute müssen nullable sein
  - Interne Darstellungsform in Hibernate

# Zusammenfassung Vererbung

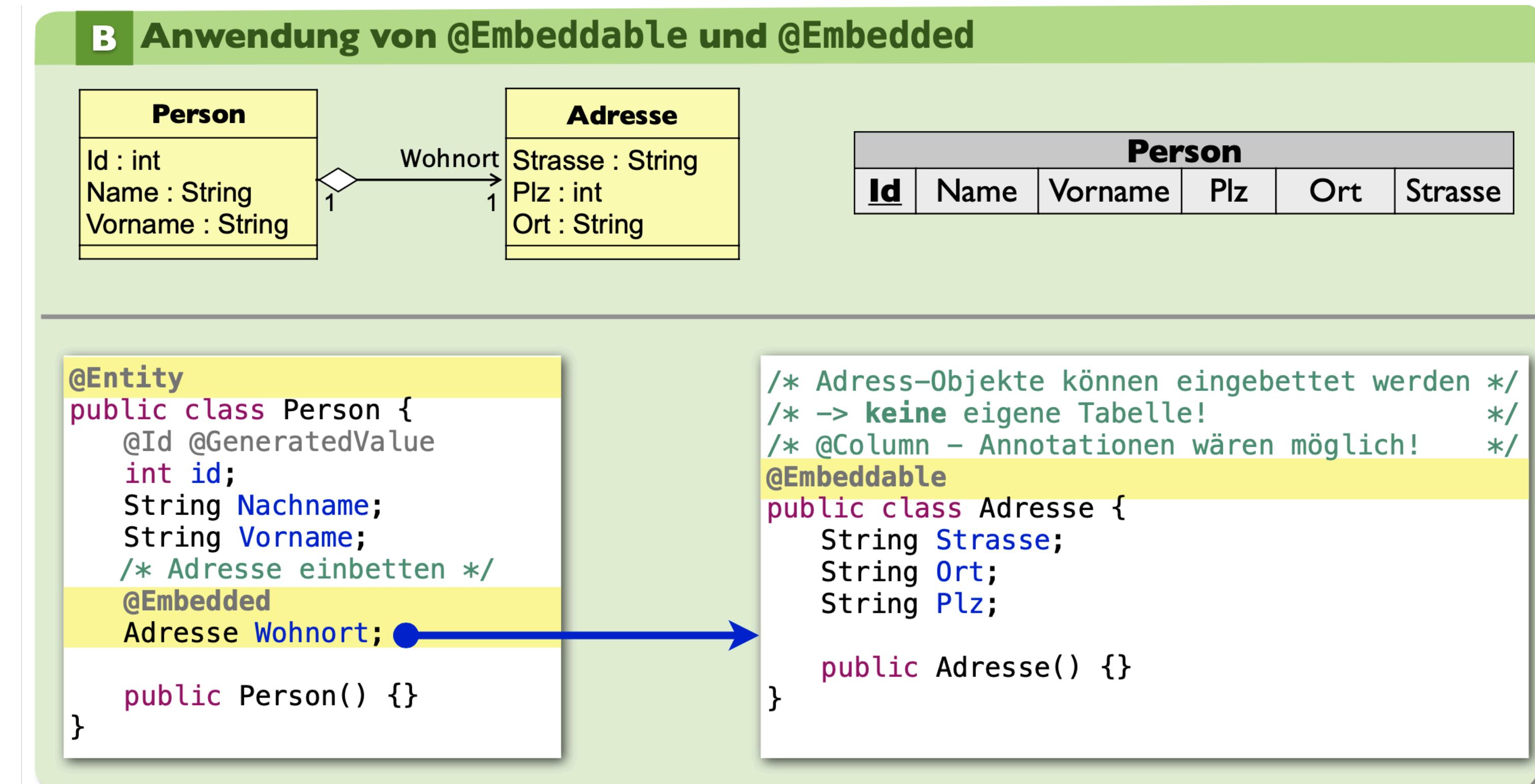
Aktion / Strategie	TABLE_PER_CLASS	JOINED	SINGLE_TABLE
Hinzunehmen von Klassen	Neue Tabelle.	Neue Tabelle; evtl. Fremdschlüsselbeziehung von Kindklassen anpassen.	Aufwändig, da neue Spalten einzufügen sind.
Ändern von Attributen	Alle Kindklassen bzw. deren Tabellen betroffen.	Nur eine Tabelle betroffen.	Nur eine Tabelle betroffen.
Polymorphes select	Aufwändig: SQL-UNION über alle Kindklassen.	Nur eine Tabelle betroffen	Nur eine Tabelle betroffen; aber: Projektion erforderlich
Monomorpher Zugriff auf einzelnes Objekt	Alle Daten in einer Tabelle.	Daten müssen aus Tabellen der Oberklassen gesammelt werden.	Alle Daten in einer Tabelle.
Sonstiges	Passt zum Konzept „Klasse entspricht Tabelle“		Attribute in Kindklassen müssen nullable sein.

# Einbettung

- 1:1-Beziehungen wird in eine Tabelle eingebettet



# Abbildung einer UML-Komposition

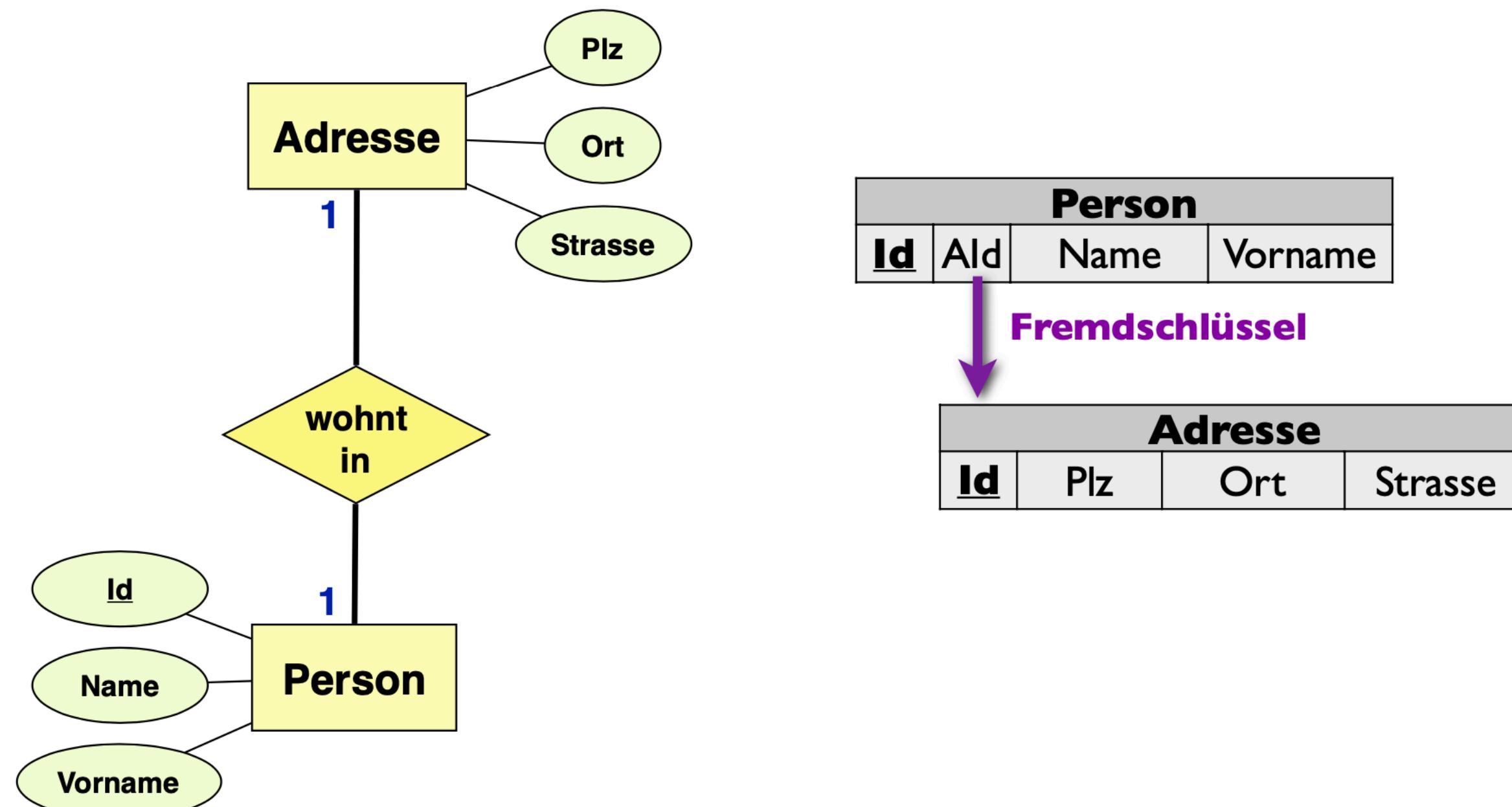


- **@EmbeddedId: Einbettung als Primärschlüssel sein, wenn**
  - Methoden **equal** und **hashCode** im Embeddable implementiert sind und
  - das **Serializable** interface realisiert wird

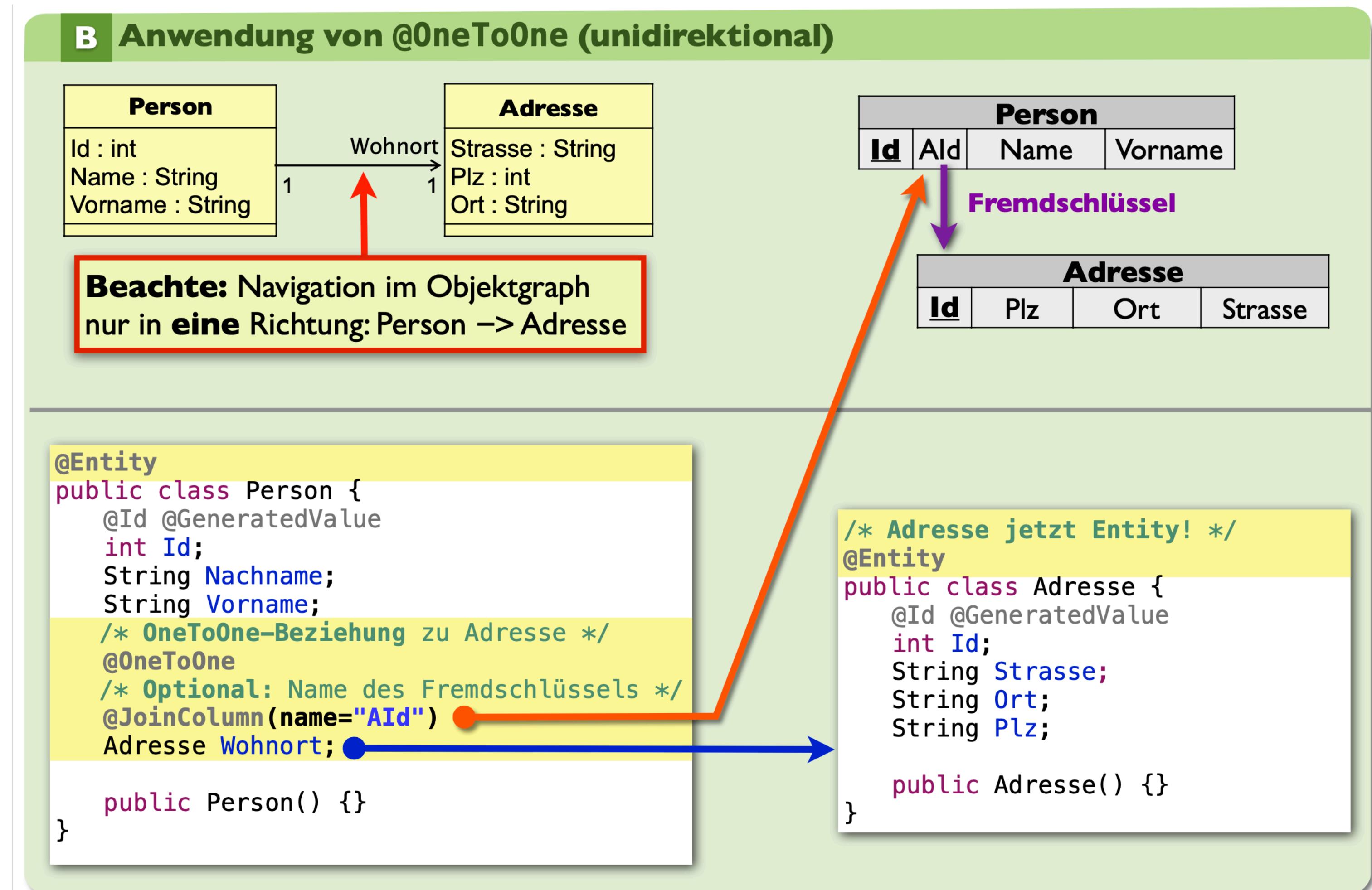
## 1:1-Beziehung über Fremdschlüssel

- **Jeder Teilnehmer in eine eigene Tabelle**
- **Beziehung wird über Fremdschlüssel hergestellt**

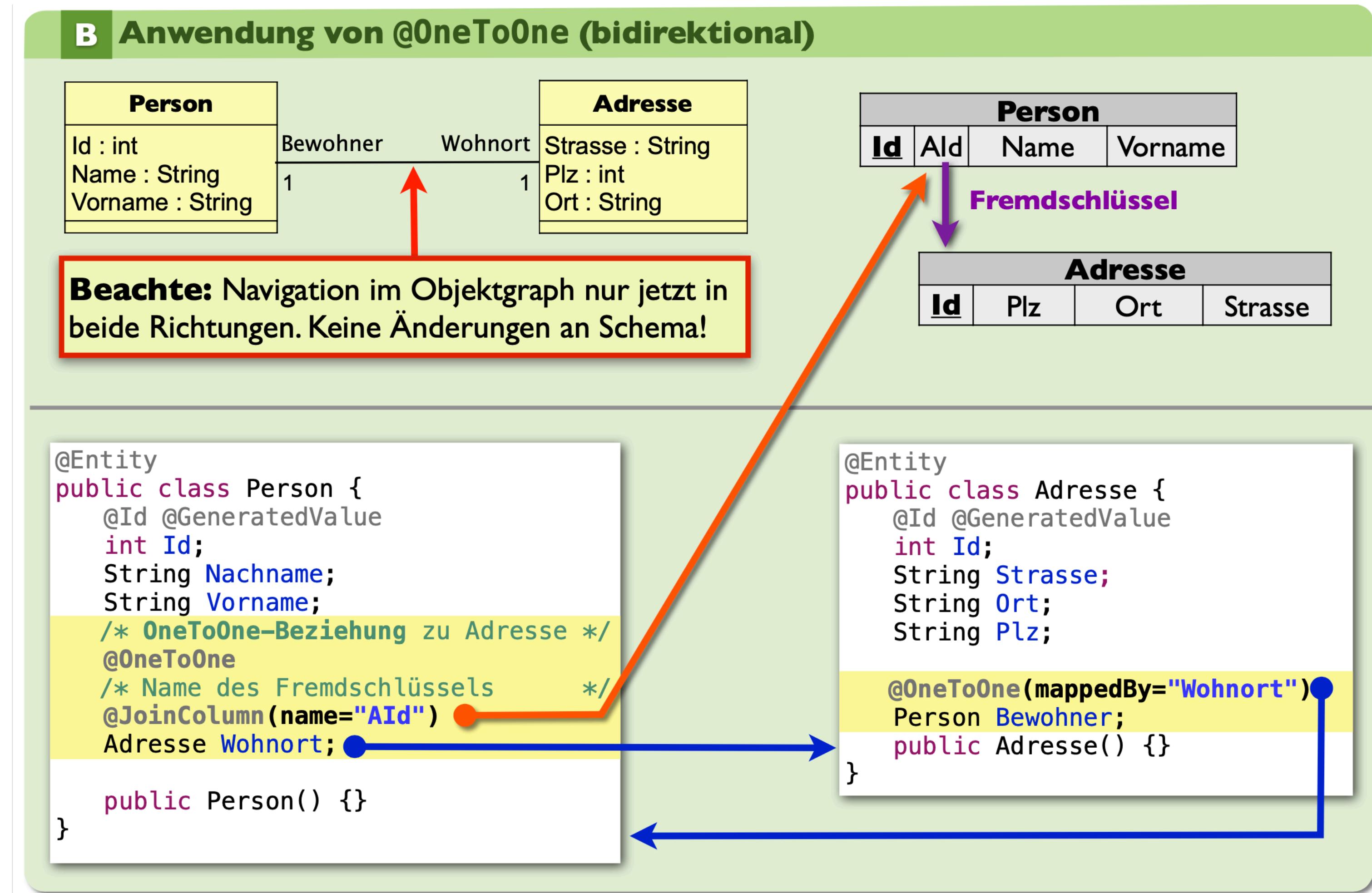
Re-Kombination via Join über **zwei** Tabellen



# 1:1-Beziehung über Fremdschlüssel



# 1:1-Beziehung über Fremdschlüssel

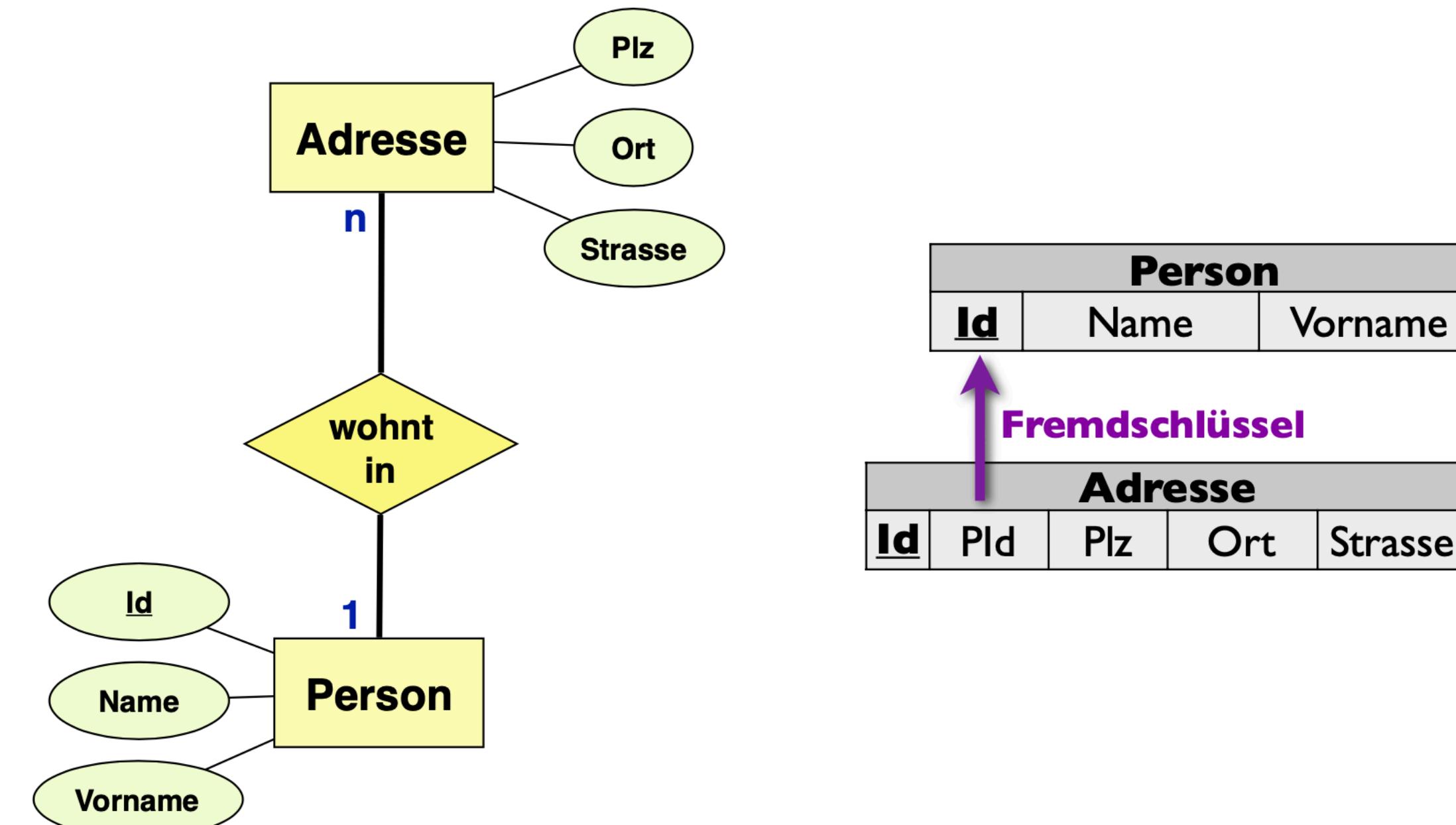


## 1:n-Beziehung

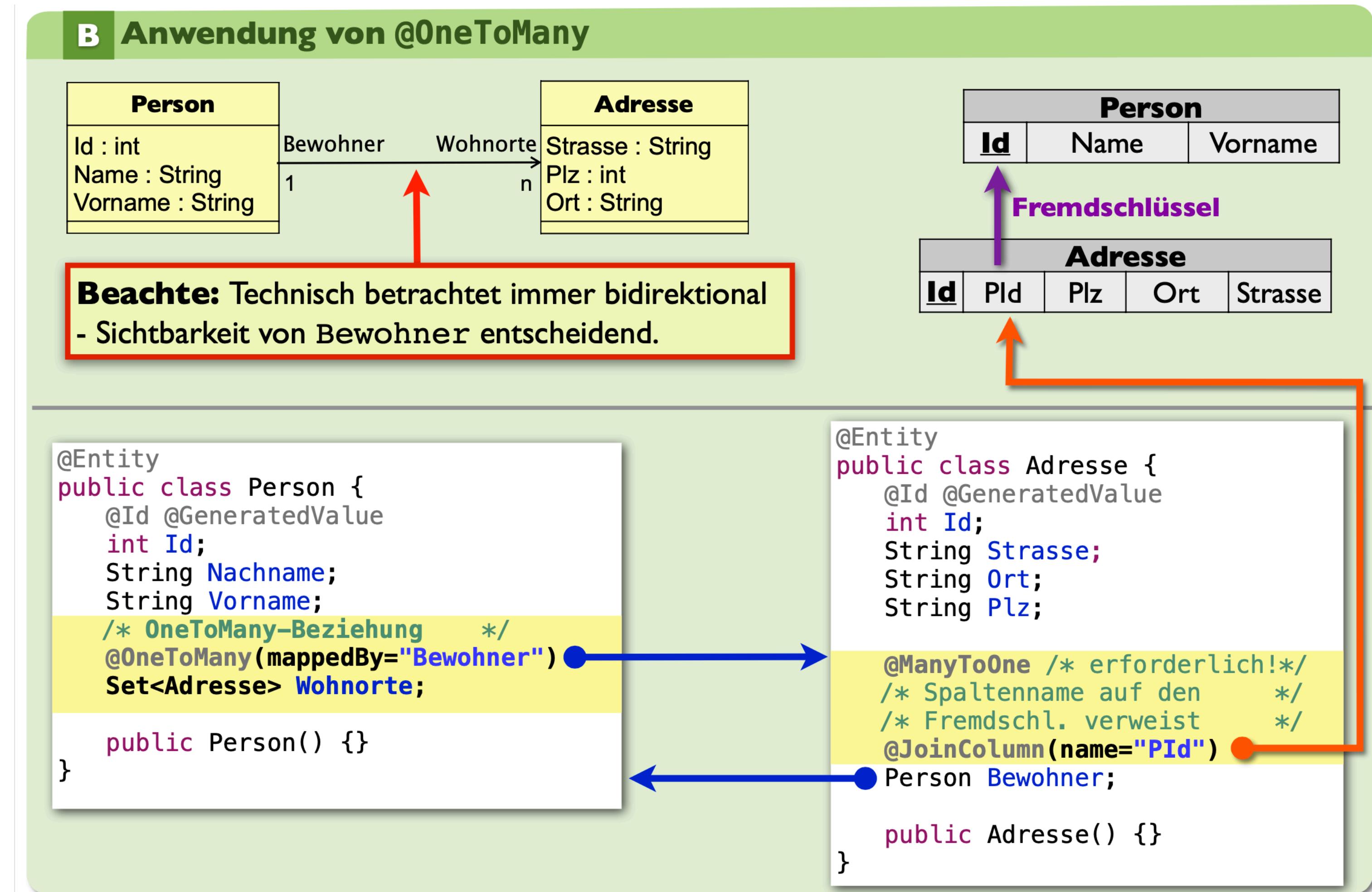
- **Jeder Teilnehmer in eigene Tabelle**
- **Beziehung direkt über Fremdschlüssel in n-Tabelle**

n-Tabelle ist Besitzer ; Re-Kombination via Join über **zwei** Tabellen

Am „1-Ende“ als interface-type ( Collection, Set, List, Map)



## 1:n-Beziehung

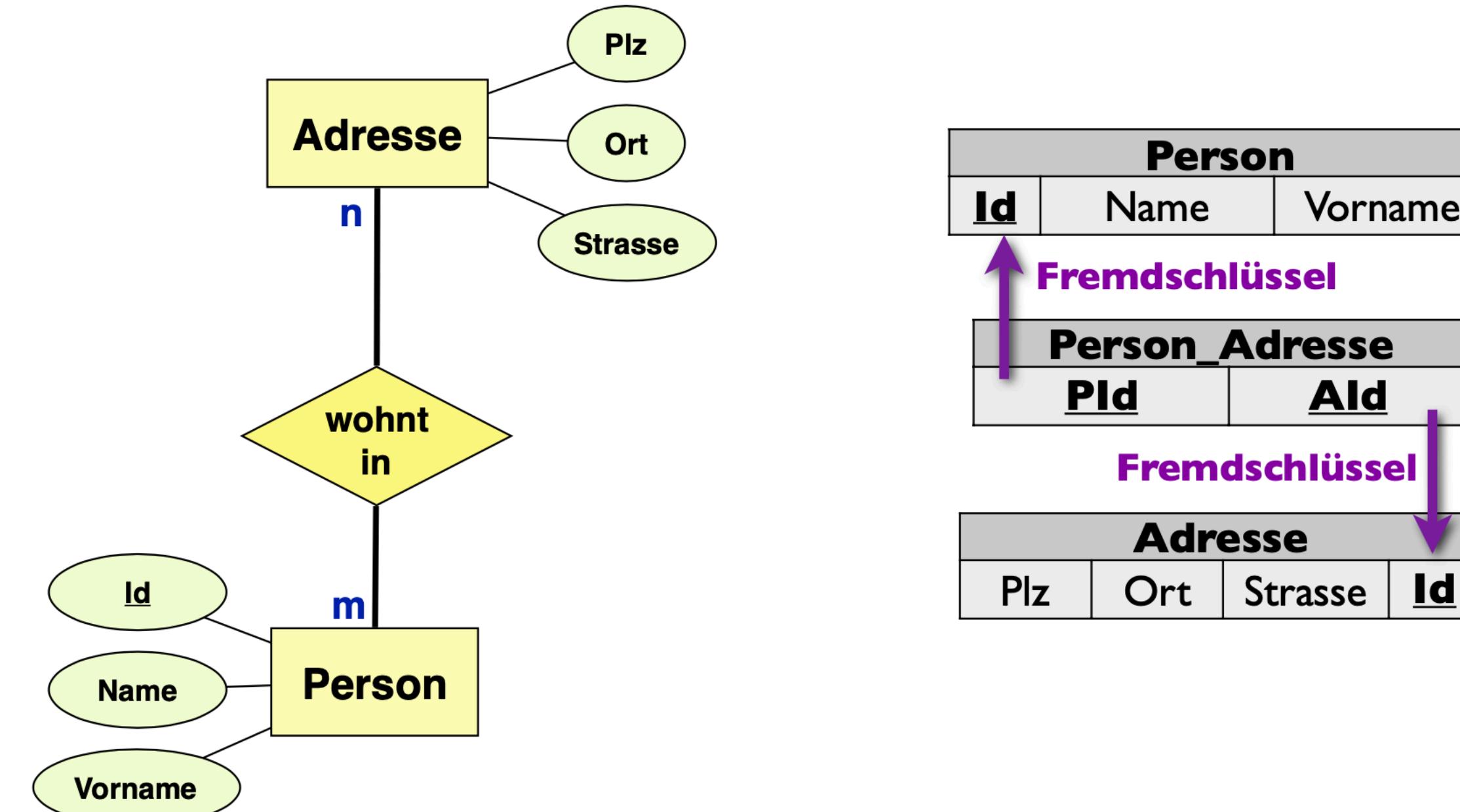


## n:m-Beziehung über Join

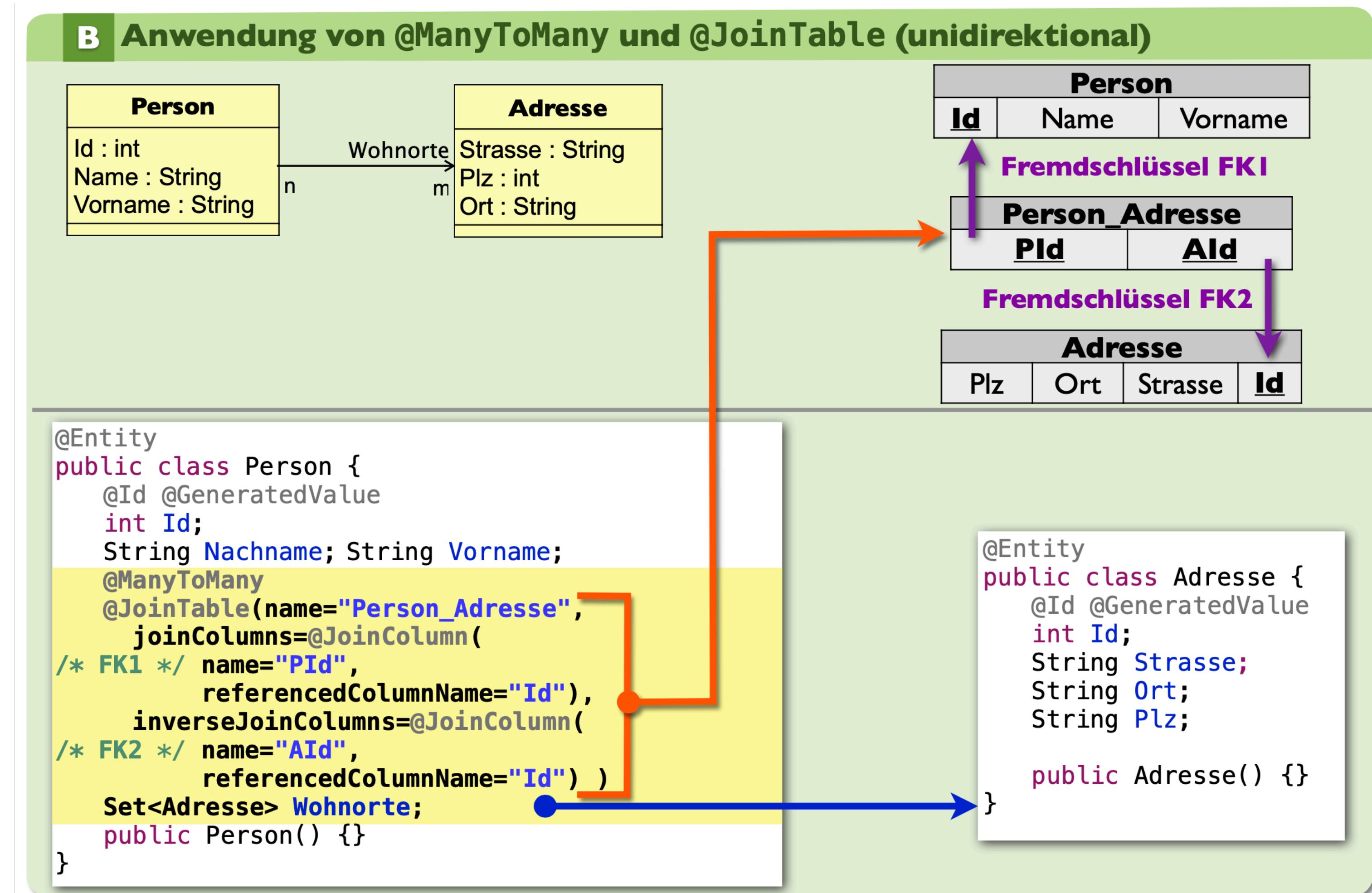
- **Jeder Teilnehmer in eine eigene Tabelle**
- **Beziehung über Assoziationsstabelle**

Re-Kombination via Join über **drei** Tabellen

Darstellung an beiden Enden als Collection (Set, ArrayList, Map etc.)



## n:m-Beziehung über Join



## ORM

---

### Anmerkungen

- Idee eines ORM sollte bekannt sein.
- Hibernate nur ein Beispiel. Die Wege, Datenbank und Klassen zu verbinden, also etwa über externe Beschreibungen oder Attribute, findet man auch in anderen ORM.
- Abstrahiert generell die Anbindung an verschiedene DBMS.
- Abwägung, ORM zu nutzen und wenn welcher, ist nicht trivial.

# UNIT 0X0E

## ODBC/JDBC

## Motivation

---

### Ziel

Bislang wurden SQL-Befehle per IDE, z.B. DataGrip, im Terminal oder Web-basiert abgesetzt. Wie kommuniziert aber die IDE oder das Terminal mit dem DBMS?

- Wir wollen Datenbankbefehle, z.B. SQL-Kommandos, programmatisch absetzen
- und auch die Ergebnisse verarbeiten.

### Ansätze/Ideen

- Einbettung von SQL in Code
- Zugriff auf das DBMS über spezielle Bibliotheken/APIs  
(Open Database Connectivity ODBC bzw. Java Database Connectivity JDBC)

### Q&A

- Wie sieht die Datenübertragung aus?
- Welche Systeme werden unterstützt?
- Welche Sprachen werden benötigt?

## Embedded SQL

### IDEE

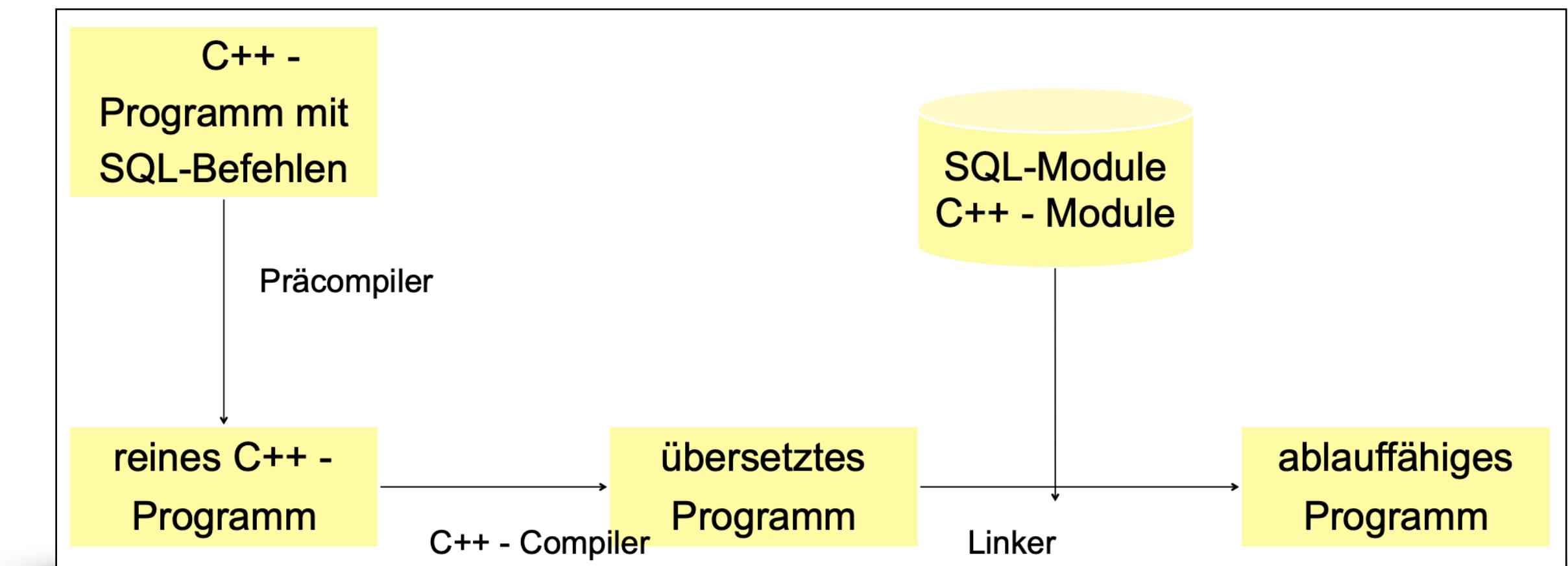
Einbettung von SQL-Anweisungen als eigenständiges Sprachkonstrukt in eine Programmiersprache

- Abgrenzung der SQL-Elemente durch spez. Präfix

```
EXEC SQL <sql-statement> <terminator>
#sql { <sql-statement> };
```

Nicht-Java, oder  
Java

- Umsetzung der Kommandos durch
  - spezielle Tools zur Compilezeit (Präcompiler) und/oder
  - Bibliotheken zur Laufzeit (Linker).



## Embedded SQL

---

### Beispiel

- Variablen aus dem Programm werden verwendet.
- Typische Aufgaben, als SQL-Befehl in das Programm eingebettet (hier C):
  - Daten abfragen (SELECT),
  - Daten anlegen (INSERT),
  - Verbindung aufbauen (CONNECT),
  - Fehlerbehandlung (SQLERROR).

```
/* connect to Oracle */
EXEC SQL CONNECT :userid;
printf("Connected.\n");
```

```
/* handle errors */
EXEC SQL WHENEVER SQLERROR do sql_error("Oracle error");
```

```
EXEC SQL SELECT ename, job, sal + 2000
INTO :emp_name, :job_title, :salary
FROM emp
WHERE empno = :emp_number;
```

```
int      emp_number;
char    temp[20];
VARCHAR emp_name[20];

/* get values for input host variables */
printf("Employee number? ");
gets(temp);
emp_number = atoi(temp);
printf("Employee name? ");
gets(emp_name.arr);
emp_name.len = strlen(emp_name.arr);

EXEC SQL INSERT INTO EMP (EMPNO, ENAME)
VALUES (:emp_number, :emp_name);
```

## Embedded SQL

### Anmerkungen

- Nebenstehend zwei Ausschnitte aus aktuellen Dokumentationen Oracle und IBM DB2, d.h. hier wird dieser Ansatz durchaus gepflegt.
- Erfordert weiteres Tool (Präcompiler) in der Entwicklungspipeline inkl. aller (DBMS-)Abhängigkeiten und Fehlerquellen.
- Sprachen C, Fortran, Cobol, REXX häufig in älteren Systemen verwendet (ohne Wertung).
- Weitere Details hierzu in den Docs.

+ <b>Changes in This Release for Pro*C/C++ Programmer's Guide</b>
- <u><a href="#">Part I Introduction and Concepts</a></u>
+ <b>1 Introduction</b>
+ <b>2 Precompiler Concepts</b>
+ <b>3 Database Concepts</b>
+ <b>4 Datatypes and Host Variables</b>
+ <b>5 Advanced Topics</b>
+ <b>6 Embedded SQL</b>
+ <b>7 Embedded PL/SQL</b>
+ <b>8 Host Arrays</b>
+ <b>9 Handling Runtime Errors</b>
+ <b>10 Precompiler Options</b>
+ <b>11 Multithreaded Applications</b>

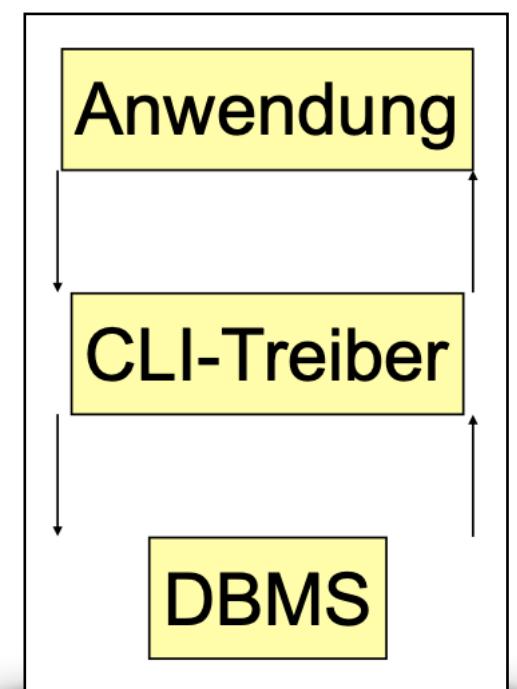
Oracle Doc.

The Db2 development environment
ADO.NET
OLE DB
enshot
DB2CI
<a href="#">Embedded SQL</a>
Embedding SQL statements in a host language
Embedded SQL statements in C and C++ applications
Embedded SQL statements in FORTRAN applications
Embedded SQL statements in COBOL applications
Embedded SQL statements in REXX applications
IBM DB2 Doc.

## Call Level Interface (CLI)

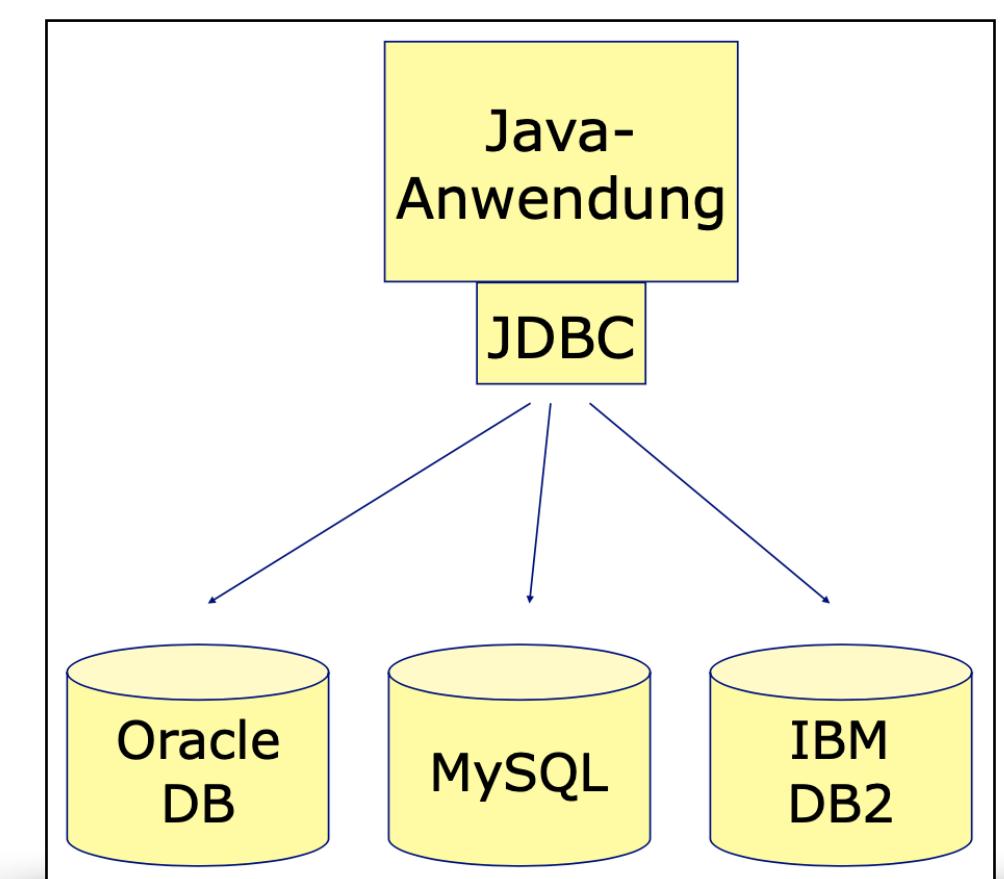
### Idee

- Anwendungen benutzen ein sprachkonformes API: Call Level Interface (CLI).
- SQL-Anweisungen werden in CLI-Calls als Zeichenkette eingebettet.



### Vorteile

- Integration in Programmiersprache, Bibliotheksnutzung, kein Präcompiler.
- Offene Schnittstelle (DBMS-unabhängig, d.h. durch Austausch der Treiber können DBMS verschiedener Hersteller angesprochen werden), z.B Open DataBase Connectivity (ODBC), Java DataBase Connectivity (JDBC).
- Weit verbreitet in der Praxis.



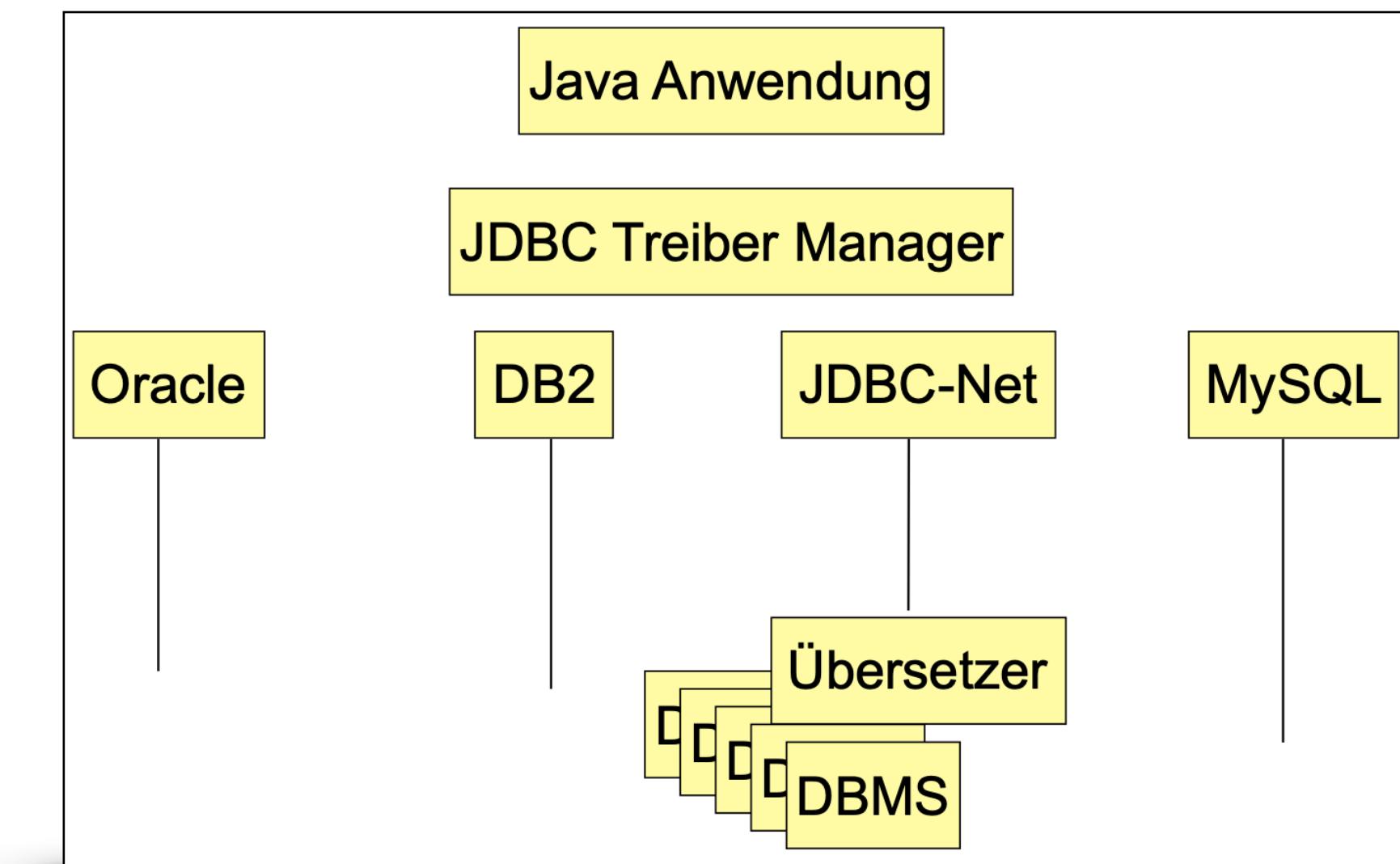
### Nachteile

- Ggf. schlechtere Performance wg. dynamischer SQL-Interpretation (zur Laufzeit).

## Java DataBase Connectivity (JDBC)

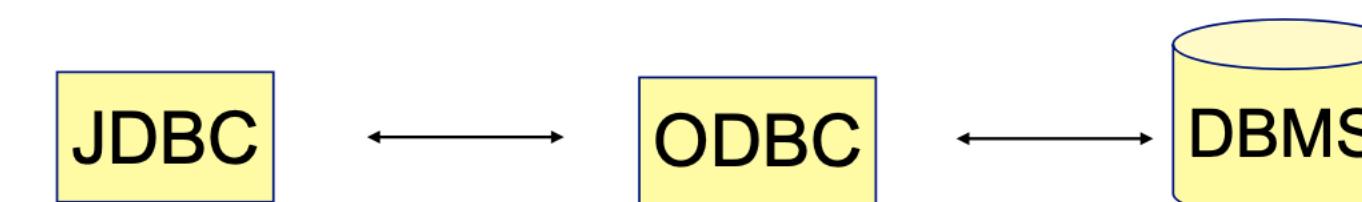
### Hintergrund

- Allgemeine Datenbank-Zugriffsschnittstelle für SQL.
- Übertragung der JAVA-Portabilität auf Datenbanken, d.h. JDBC ermöglicht einen SQL-basierten, plattform-unabhängigen Zugriff auf verschiedene relationale DBMS.
- Teil von Java 5 (Package `java.sql`).
- Basiert auf X/Open SQL CLI (wie auch ODBC).
- Übersichtlicher und einfacher benutzbar als ODBC.
- Unterstützt ANSI SQL92- Basisfunktionalität.
- Treiber sind vom DBMS-Hersteller als Bibliothek (z.B. jar-File) oder Artefakt (z.B. via maven mvnRepository) zu bekommen.

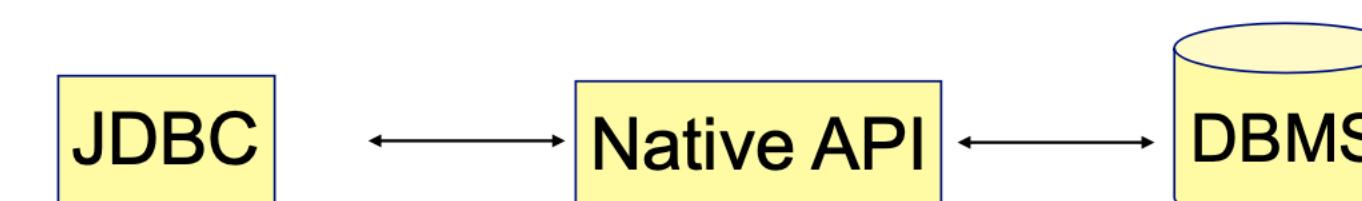


## JDBC-Treiber-Implementierung

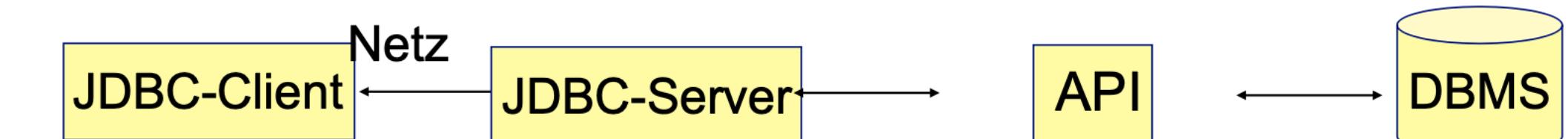
- **JDBC-ODBC-Bridge (Typ 1)**
  - Nutzen vorhandener ODBC-Treiber



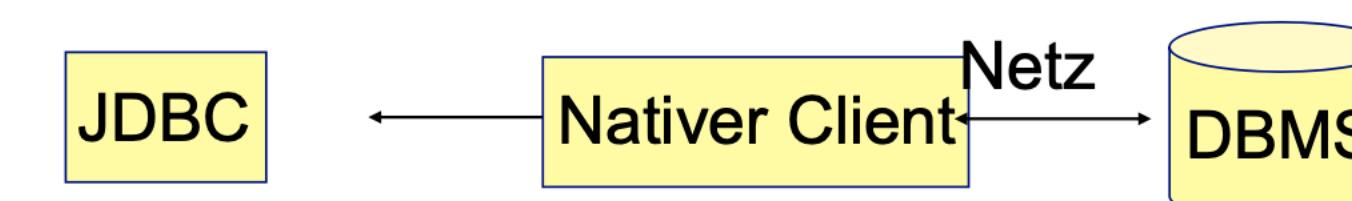
- **Native-API-Treiber (Typ 2)**
  - Basiert auf Bibliotheken des DBMS-Herstellers



- **JDBC-Netz-Treiber (Typ 3)**
  - Aufteilung Treiber in JDBC-Client und JDBC-Server



- **Native-Protokoll-Treiber (Typ 4)**
- Direkte Übersetzung der JDBC-Aufrufe in DBS-Aufrufe
  - Client direkt mit Datenbankserver verbunden
  - Nachteilig sind die i.A. proprietären DBS-Protokolle



# Abbildung SQL- auf Java-Datentypen

SQL-Datentyp	Java-Datentyp
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	<i>java.sql.Bignum</i>
DECIMAL	<i>java.sql.Bignum</i>
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONVARBINARY	byte[]
DATE	<i>java.sql.Date</i>
TIME	<i>java.sql.Time</i>
TIMESTAMP	<i>java.sql.Timestamp</i>

## Interaktion mit dem DBMS

---

### Prinzipielle Funktionsweise

- Datenbanktreiber laden (implizit oder explizit).
- Aufbau einer Verbindung (Connection) zur Datenbank.
- Erstellen und Ausführen von SQL-Anweisungen sowie Verarbeitung der Ergebnisse im Kontext der Verbindung
  - Cursor-Konzept notwendig - folgt.
- Alle Verbindungen (Connection, Statement, ResultSet) immer schließen (close).
- Programmbeispiele folgen ebenfalls.

## Einschub Cursor-Konzept

**Problem:**

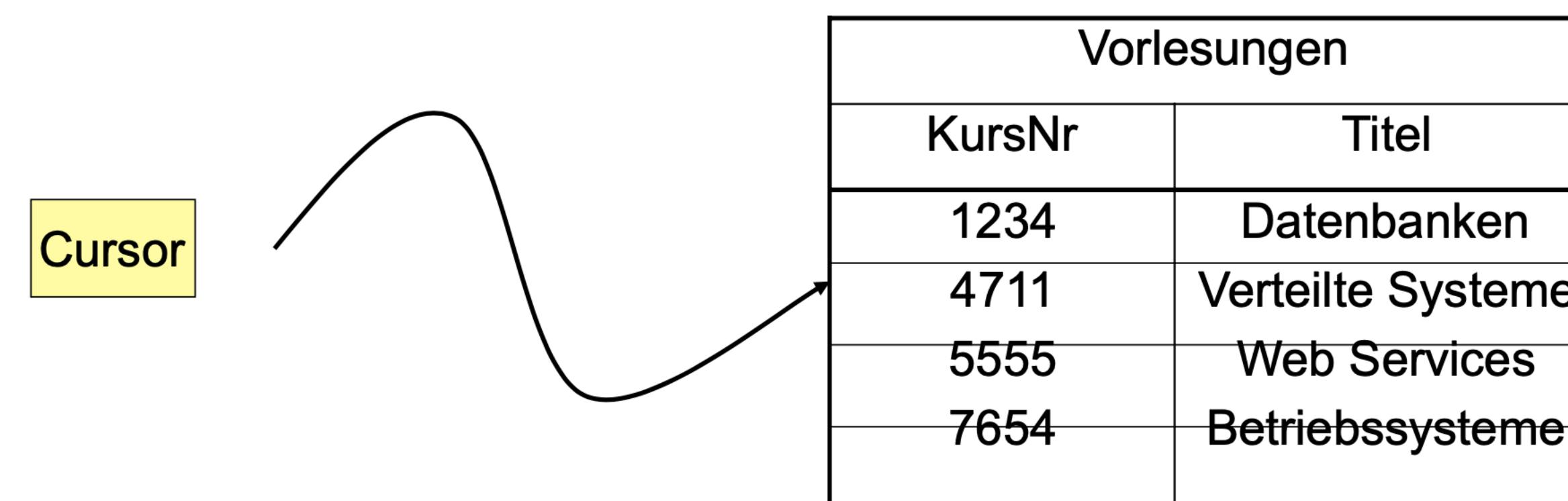
Fehlanpassung („impedance mismatch“) zwischen Datenmodell des DBMS und der Anwendung, da normale Programmiersprachen einen Typ „Relation“ nicht anbieten

**Aufgabe:**

Abbildung von Tupelmengen auf die Variablen der Programmiersprache

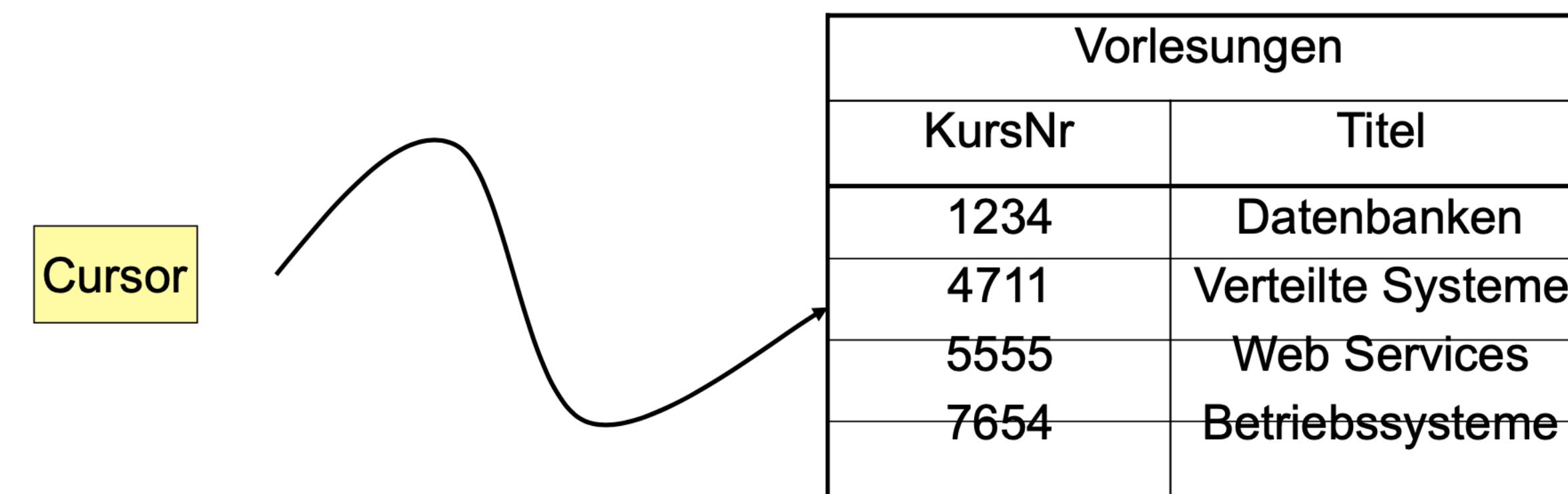
**Lösung:**

Cursor-Konzept; sukzessives traversieren der Tupel einer Relation mittels des Iterator-Entwurfsmuster



## Einschub Cursor-Konzept

- Durch das Iterator-Muster kann man sequentiell auf die Elemente einer Datenstruktur bzw. eines zusammengesetzten Objekts zugreifen, ohne dabei ihre interne Struktur offen zu legen



## Beispiel Java/Kotlin

### Projekt

- Mix aus Java und Kotlin.
- Projekt: db\_connector\_xxx

### Ablauf

- Verbindung aufbauen,
- alle Kunden ausgeben,
- Kunde einfügen, alle ausgeben,
- Kunde löschen, alle ausgeben,
- Metadaten Tabelle  
Kunde ausgeben,
- Verbindung schliessen.

```
fun main() {
    try {
        connectToMatseMhistOf(
            user = "root",
            password = "root",
            port = 3310
        ).use { connection -> connection.apply { this: Connection
            println("step 1: select all in Kunde")
            showAllCustomers()

            val kunde = "FH Aachen"
            println("step 2: insert kunde '$kunde'")
            insertCustomer(name = kunde, discount = 1.0)
            showAllCustomers()

            println("step 3: delete kunde '$kunde'")
            deleteCustomer(name = kunde)
            showAllCustomers()

            println("step 4: metadata table 'kunde'")
            showMetaDataCustomers()
        } }
    } catch (e: SQLException) {
        println("SQL-Error: $e")
    } catch (e: Exception) {
        println("Error: $e")
    }
}
```

## Beispiel Java/Kotlin

---

### Verbindung aufbauen (Kotlin)

```
fun connectToMatseMhistOf(user: String, password: String, port: Int) =  
    DriverManager.getConnection(  
        "jdbc:mysql://localhost:$port/matse_mhist?useSSL=false&allowPublicKeyRetrieval=true",  
        user,  
        password  
    )!!
```

- Aufruf über DriverManager
- Zusammenstellung eines Connection-Strings; genauer Aufbau variiert je Verbindungs-typ und DBMS (hier jdbc:mysql), URL, Port (hier localhost, port) und Parametern (hier useSSL etc.).
- Bei Erfolg ist Rückgabe eine Connection, sonst wird eine SQLException geworfen.
- Sieht in Java (fast) genau so aus.

## Beispiel Java/Kotlin

---

### Alle Kunden ausgeben (Kotlin)

- SQL-Befehl zusammenstellen (sql),
- Statement aus SQL-Befehl erstellen und ausführen,
- über Ergebnismenge/ResultSet nacheinander iterieren, die Werte der Spalten holen und ausgeben,
- das Statement wird implizit geschlossen (wg. use).
- In Java wird üblicherweise explizit ein Statement- und ein ResultSet-Objekt angelegt, evtl. mittels try-with-resources.

```
fun selectAllInKunde(connection: Connection) {  
    val sql = "SELECT id, name, rabatt_prozent FROM kunde"  
    connection.createStatement().apply { //statement ->  
        executeQuery(sql).use { resultSet ->  
            while (resultSet.next()) {  
                val id = resultSet.getInt("id")  
                val name = resultSet.getString("name")  
                val rabatt = resultSet.getDouble("rabatt_prozent")  
                println(" id:$id, name:'$name', rabatt:$rabatt")  
            }  
        }  
    }  
}
```

## Java DataBase Connectivity (JDBC)

---

### Einschub Statements

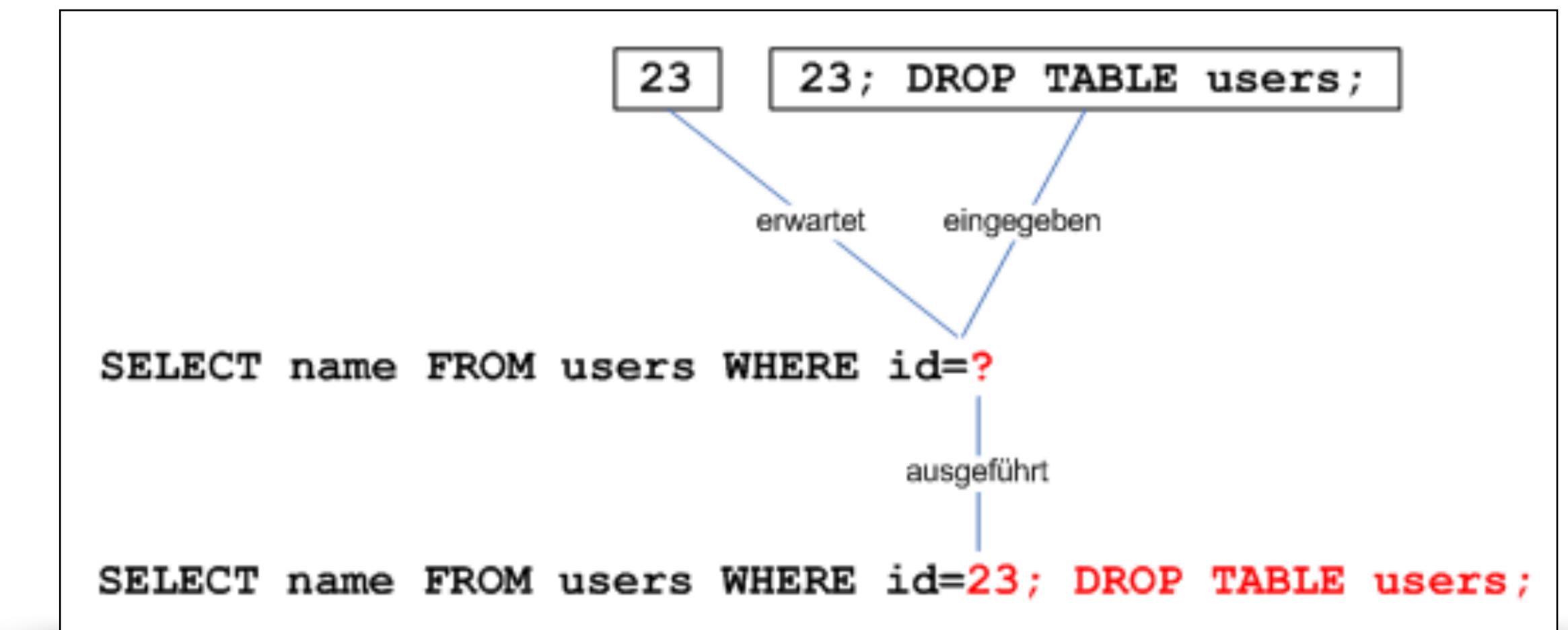
JDBC unterstützt drei Statement-Arten:

- **Statement:** Statische SQL-Anweisung, die von JDBC nicht verändert, sondern einfach weiter gereicht wird.
- **PreparedStatement:** Erlaubt die Formulierung von vorbereiteten SQL-Anweisungen, bei denen später bestimmte Attribute über Parameter belegt werden. Hierdurch können häufig verwendete, nur leicht variierende SQL- Anweisungen bereits vom DBMS übersetzt werden. Eine spätere Ausführung findet zumeist auf der Basis des schon erstellten Zugriffsplan statt. Gleichzeitig wird so ein Angriff via SQL-Injection verhindert.
- **CallableStatement:** Erlaubt das Ausführen einer in der Datenbank hinterlegten Prozedur (Stored Procedure) – hier nicht weiter relevant.

## Java DataBase Connectivity (JDBC)

### Einschub SQL-Injection

- Angriffsart, bei der durch mangelnde Validierung der Eingabedaten das Einschleusen fremden SQL-Codes in eine Datenbankabfrage möglich ist.
- Wichtiges Problem für Web-basierte Anwendungen, die SQL-Statements generieren.
- Erfolgt durch Modifizierung von übergebenden Parametern.
- Eine dynamisch generierte SQL-Query wird mit fremden Code „infiziert“. Basiert auf der Idee, das eigentliche Kommando abzubrechen und dafür als Parameter ein vollständiges weiteres SQL-Kommando abzusetzen.



## Java DataBase Connectivity (JDBC)

---

### ResultSets

- Nehmen Ergebnisse auf bzw. stellen Methoden zur Verfügung, mit denen diese abgerufen werden sowie Methoden zur Positionierung des Cursors.
- Was beim Positionieren des Cursors möglich ist, hängt am Statements-Objekt, der Anfrage selber und dem DBMS. Hier gibt es verschiedene Typen, z.B.
  - `ResultSet.TYPE_FORWARD_ONLY` (nur vorwärts),
  - `ResultSet.TYPE_SCROLL_INSENSITIVE` (alle Richtungen)
  - `ResultSet.CONCUR_UPDATABLE` (modifizierbar).
- Seit JDBC 2.0 können die in ResultSets gespeicherten Daten auch modifiziert werden.
- Achtung: Bevor Werte verarbeitet werden können, muss die `next`-Methode aufgerufen werden. Ist der Rückgabewert `TRUE`, so liegt ein Ergebnis vor.

# Java DataBase Connectivity (JDBC)

## ResultSets

Vorlesungen	
KursNr	Titel
1234	Datenbanken
4711	Verteilte Systeme
5555	Web Services
7654	Betriebssysteme

} rSet

- Schließlich kann das **ResultSet**-Objekt mittels des Cursor-Prinzips verarbeitet werden

```
while (resultSet.next()) {  
    val id = resultSet.getInt("id")  
    val name = resultSet.getString("name")  
    val rabatt = resultSet.getDouble("rabatt_prozent")  
    println(" id:$id, name:'$name', rabatt:$rabatt")  
}
```

## Beispiel Java/Kotlin

---

### Kunde einfügen (Java)

```
static void insertCustomer(Connection connection, String name, Double discount) throws
    // hier ist das PreparedStatement, man achte auf den Zusatz '(?,?)'
    // - auf die id wird verzichtet, da die Tabelle Kunde autoincrement id besitzt
    String sql = "INSERT INTO kunde (`name`, `rabatt_prozent`) VALUES (?,?)";
    PreparedStatement statement = connection.prepareStatement(sql);
    statement.setString(parameterIndex: 1, name);
    statement.setDouble(parameterIndex: 2, discount);
    statement.execute();
}
```

- SQL-Befehl als PreparedStatement zusammenstellen. Hierbei werden die '?' durch explizit anzugebende Werte ersetzt.

## Beispiel Java/Kotlin

---

### Kunde löschen (Java)

```
public static void deleteCustomer(Connection connection, String name) throws
    // hier ist das PreparedStatement, in einem try-with-resources
    String sql = "DELETE FROM kunde WHERE `name`=?;";
    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.setString(parameterIndex: 1, name);
        statement.execute();
    }
}
```

- Wie zuvor SQL-Befehl als PreparedStatement zusammenstellen. Hierbei werden wieder die '?' durch explizit anzugebende Werte ersetzt.

## Beispiel Java/Kotlin

---

### Metadaten Tabelle Kunde ausgeben (Kotlin)

```
fun showMetaDataCustomers(connection: Connection) {
    connection.metaData.getColumns(null, null, "kunde", "%").use { set ->
        while (set.next()) {
            val column = set.getString("COLUMN_NAME")
            val dataType = set.getString("DATA_TYPE")
            println(" column:$column, type-enum:$dataType")
        }
    }
}
```

- Wie beim select wird hier das ResultSet ausgegeben. Es enthält zeilenweise die Spalteninformationen, z.B. den Namen und den Datentyp.
- Auch hier wird das ResultSet wieder geschlossen.

## Beispiel Java/Kotlin

### Meanwhile in DataGrip

- Nach dem Einfügen war der neue Datensatz in der Tabelle Kunde auch in einer externen Sicht auf die Tabelle zu sehen (DataGrip).

id	name	rabatt_prozent
2	Bayer	1.00
26	BMW	1.00
27	Commerzbank	0.00
28	Daimler	1.00
29	Deutsche Bank	0.00
30	Sparkasse	10.00
31	Börse	0.00
32	E.ON	1.00
33	Infinion	2.00
34	Lufthansa	2.00
35	RWE	1.00
36	SAP	2.00

id	name	rabatt_prozent
2	Bayer	1.00
26	BMW	1.00
27	Commerzbank	0.00
28	Daimler	1.00
29	Deutsche Bank	0.00
30	Sparkasse	10.00
31	Börse	0.00
32	E.ON	1.00
33	Infinion	2.00
34	Lufthansa	2.00
35	RWE	1.00
36	SAP	2.00
51	FH Aachen	1.00

matse\_mhist

## Java DataBase Connectivity (JDBC)

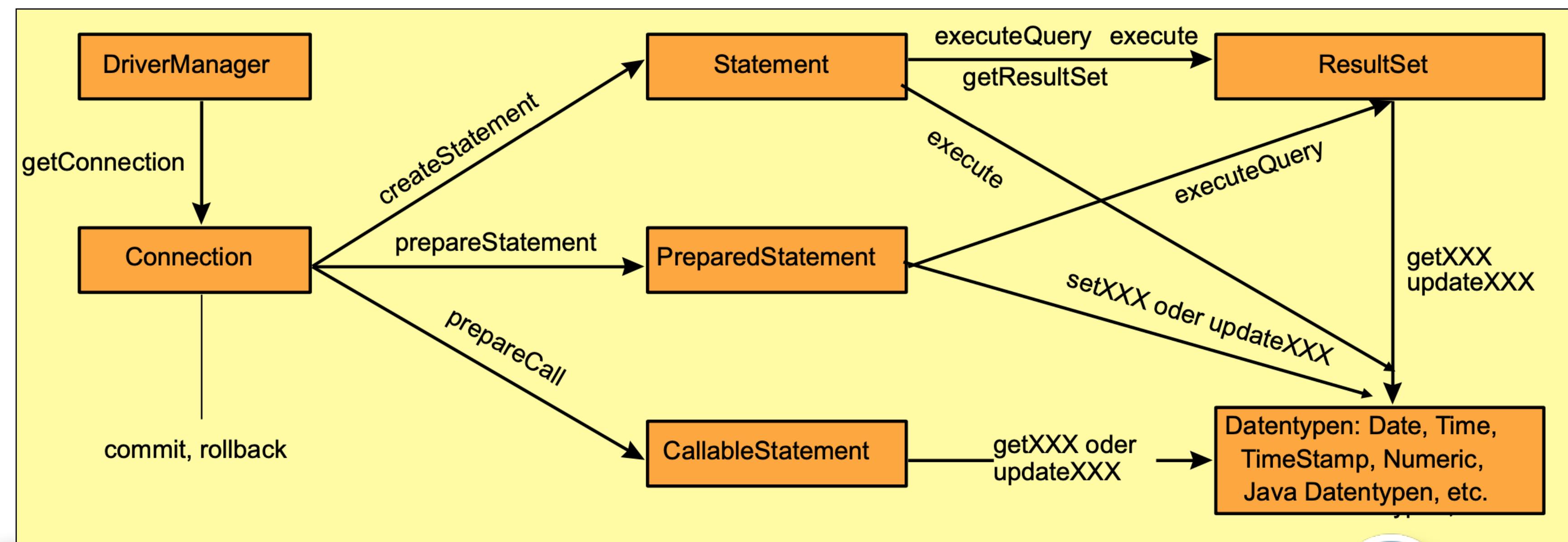
---

### Ergänzung

- Zur Realisierung von Transaktionen gibt es Befehle der Connection zum commit und rollback. Standardmäßig wird ein Auto-Commit bei jedem update durchgeführt. Dies gilt auch für Änderungen mittels ResultSets.
- Die Fehlerbehandlung wird in JDBC ausschließlich über Exceptions, genauer SQLExceptions, abgewickelt.
- Außer den Exceptions gibt es noch die Möglichkeit, Warnungen abzufangen. Diese stoppen nicht die Ausführung einer Anweisung, sie alarmieren nur den Benutzer, dass die Abwicklung nicht wie geplant ablaufen könnte. Sie werden von Connections, Statements und ResultSets über getWarnings bereitgestellt.

# Java DataBase Connectivity (JDBC)

## Zusammenfassung



Das kann hier nur ein grober Überblick über grundsätzliche Zusammenhänge sein.  
Details entnehme man bitte der Dokumentation!



# UNIT 0X0F

# NoSQL

## NoSQL = Not only SQL...

---

### Idee

- NoSQL bezeichnet Typen von DBMS, die einen nicht-relationalen Ansatz verfolgen
- ER-Modellierung nach wie vor sinnvoll

### Q&A

- Vor- bzw. Nachteile relationaler DBMS?
- Wo liegen ggf. Schwierigkeiten bei Alternativen?

## Überblick

---

### NoSQL Typen (unvollständig)

- Graphendatenbanken
- Objektorientierte Datenbanken
- Key-Value Datenbanken
- Dokumentenorientierte Datenbanken

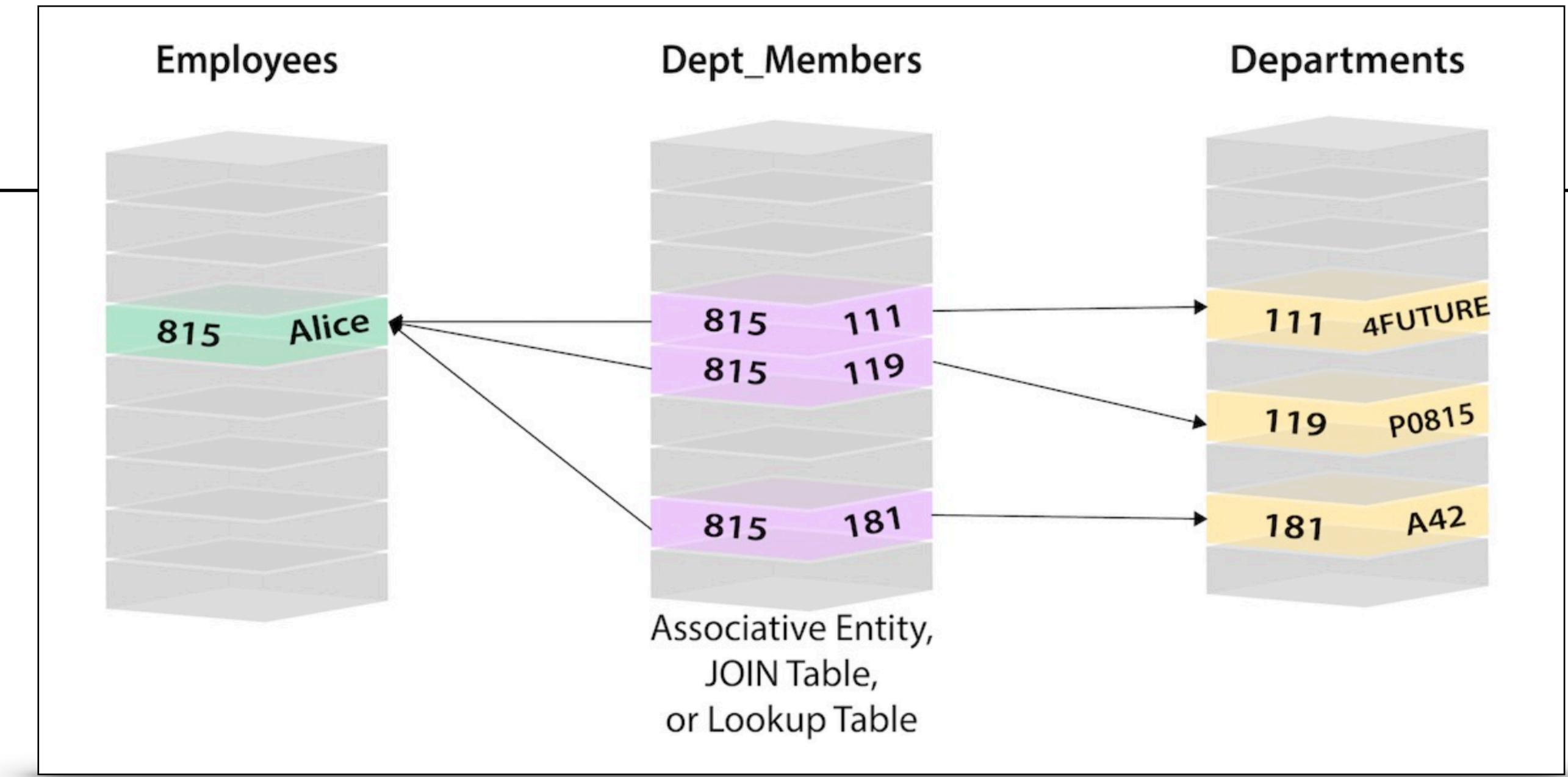
### Abgrenzung und Bezeichnung

- unscharf
- relationale DBMS bieten z.T. obige Tabellen- oder Spaltentypen an, beispielsweise unterstützt MySQL nativ JSON Daten (Dokumente) und implementiert einen effizienten Zugriff auf einzelne Attribute innerhalb eines solchen Datums

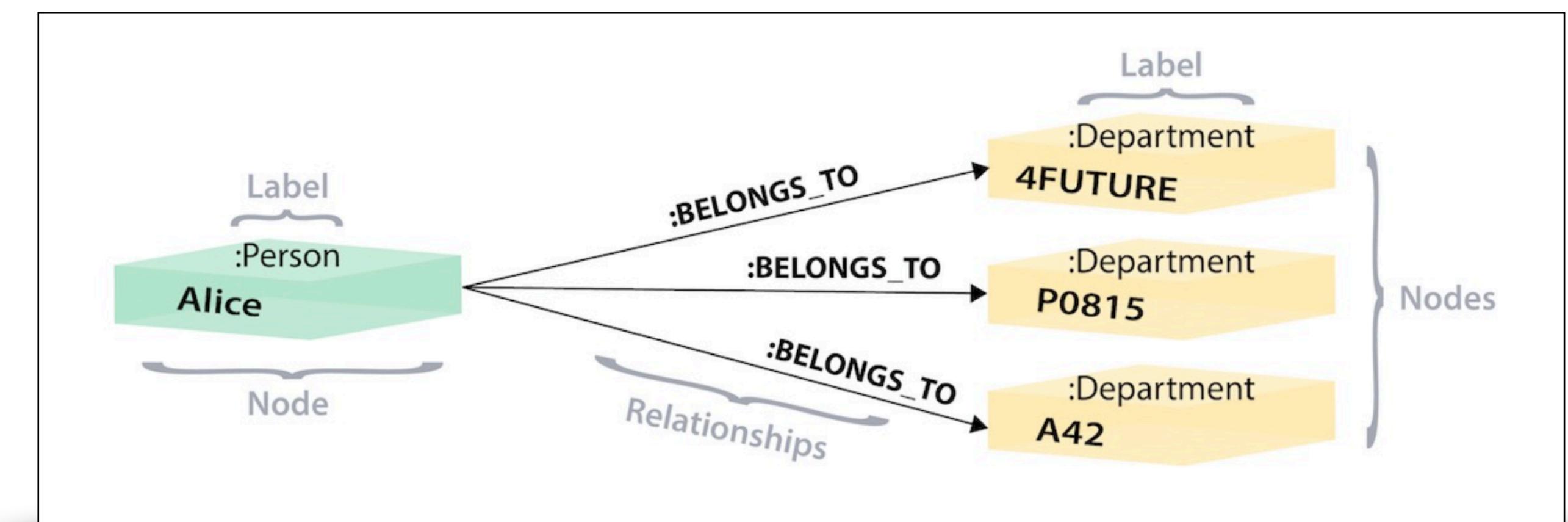
## NoSQL Typen

### Graphendatenbanken

- Stark vernetzte Daten (Netzwerk) können hier inkl. der Eigenschaften von Knoten und Kanten abgelegt werden
- Spezielle Variante: Triplestore (Bob kennt Fred), z.B. im Forschungsdatenmanagement
- Bsp. neo4j



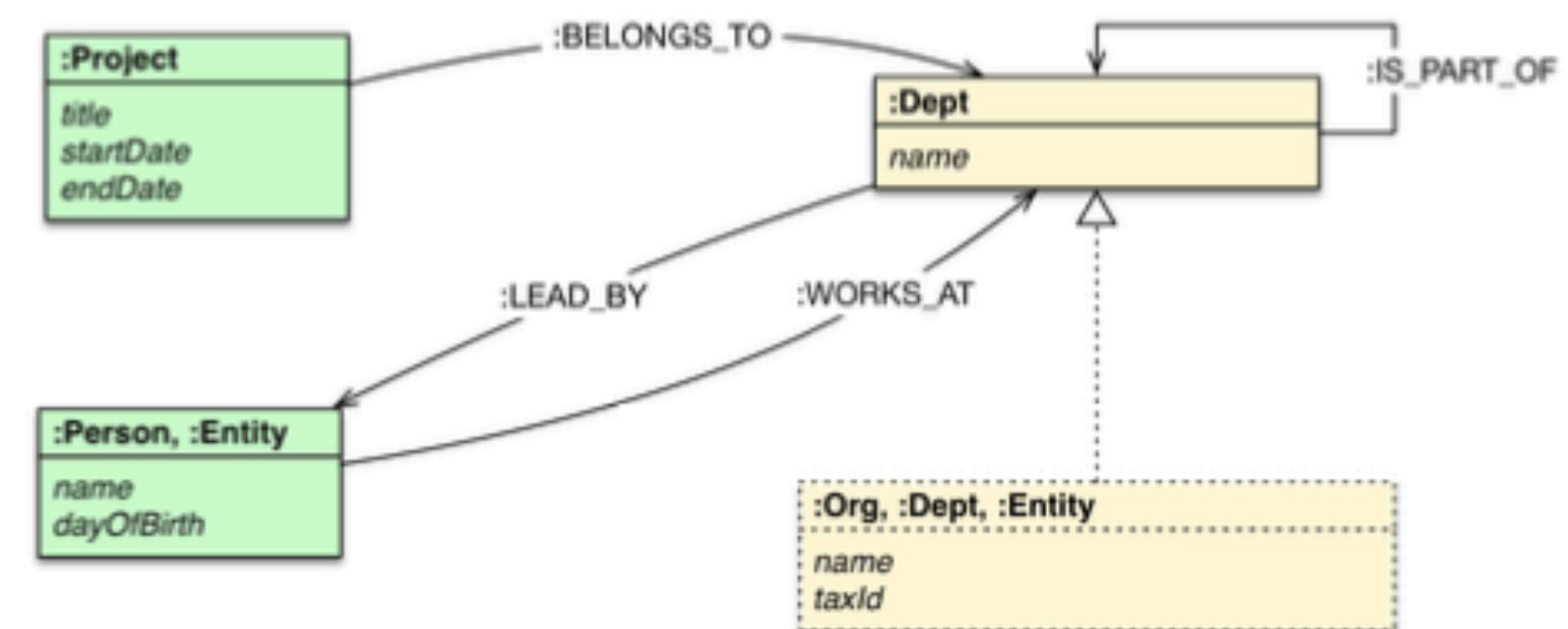
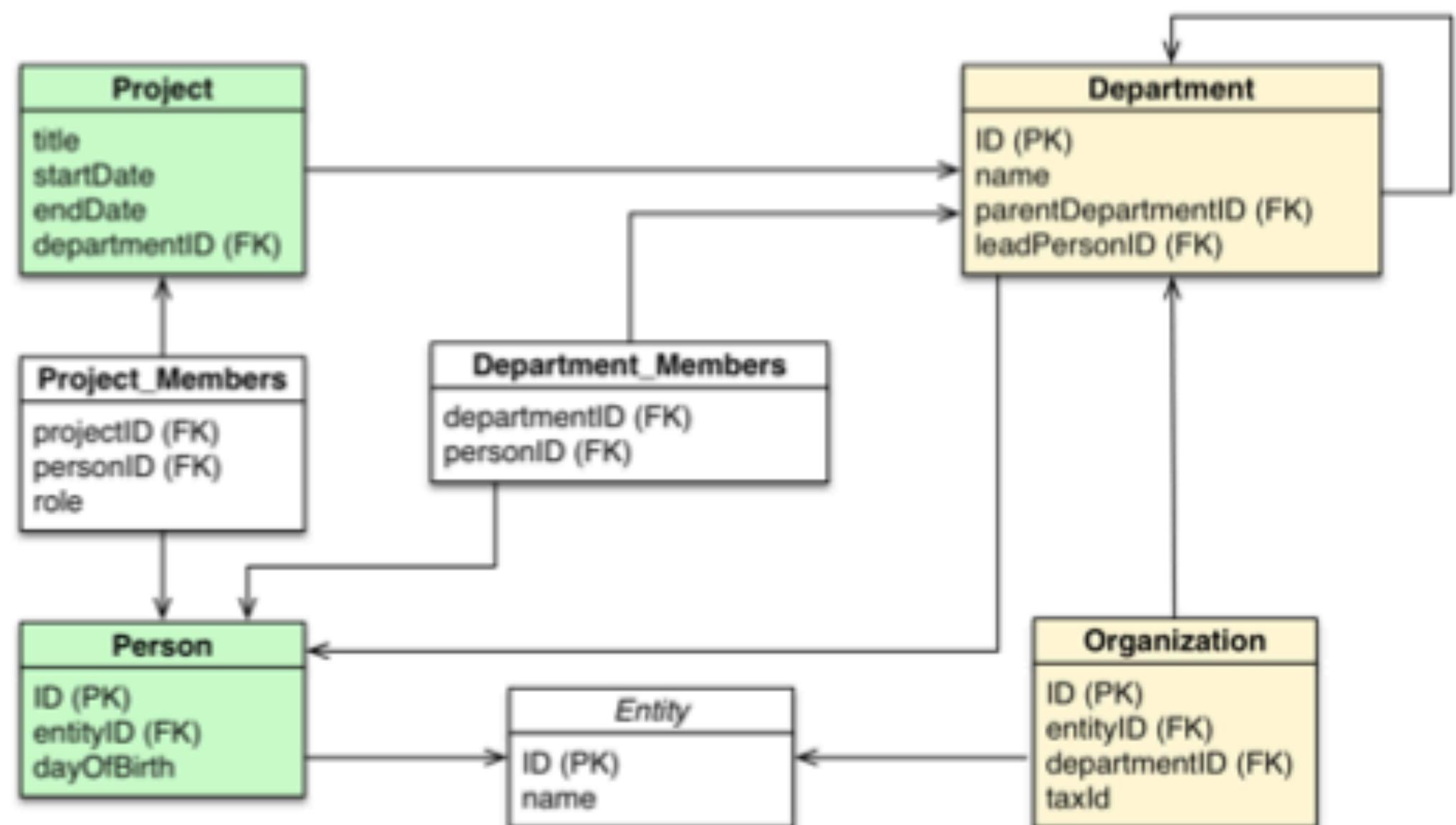
Rel. Sicht, Quelle neo4j



Nodes, Quelle neo4j

## NoSQL Typen

### Graphendatenbanken neo4j



Quelle neo4j

## NoSQL Typen

---

### Graphendatenbanken

#### SQL Statement Sql

```
SELECT name FROM Person
LEFT JOIN Person_Department
  ON Person.Id = Person_Department.PersonId
LEFT JOIN Department
  ON Department.Id = Person_Department.DepartmentId
WHERE Department.name = "IT Department"
```

#### Cypher Statement Cypher

```
MATCH (p:Person)-[:WORKS_AT]->(d:Dept)
WHERE d.name = "IT Department"
RETURN p.name
```

Quelle neo4j

## NoSQL Typen

---

### Objektorientierte Datenbanken (ODBMS)

- Verwaltung komplexer Objekte
- Unterstützung von Objektidentitäten
- Kapselung der Objekte entsprechend objektorientierter Programmierung
- Klassen sind in Klassenhierarchien angeordnet
- ODBMS stellt Manipulationssprache bereit
- Bsp. ZooDB
  - implementiert Apache JDO
- Bsp. db4o
  - unterstützt .NET und begrenzt LINQ
  - keine Unterstützung mehr seit 2014 ...

## NoSQL Typen

### Objektorientierte Datenbanken (ODBMS)

- Sehr kleine Bibliothek.
- Objekte können gespeichert werden ohne Metainformationen, z.B. Klasseninformationen, anzugeben.

```
// Öffne eine Datenbank
ObjectContainer db = Db4o.openFile("C:/beispiel.db");
try {
    // Speichere zwei Personen
    db.store(new Person("John Doe", "TOP INC", 45));
    db.store(new Person("Clara Himmel", "BB Radio", 30));
    // Iteriere über alle Personen
    ObjectSet result = db.queryByExample(new Person());
    while (result.hasNext()) {
        System.out.println(result.next());
    }
    // Verändere eine Person
    result = db.queryByExample(new Person("Clara Himmel"));
    Person found = (Person) result.next();
    found.setAge(25);
    db.store(found);
    // Lösche eine Person
    db.delete(found);
    // Schreibe die Änderungen fest
    db.commit();
}
finally {
    // Schließe die Datenbank
    db.close();
}
```

db4o

## NoSQL Typen

---

### **Key-Value Datenbanken**

- In Key-Value-Datenbanken werden Werte (Value) über einen Schlüssel (Key) eindeutig identifiziert und abgelegt.
- Werte bestehen aus Tupeln, Relationen oder Dokumenten.
- Trennung Key-Value- und Dokument-Datenbanken unscharf.
- Beispiel Google BigTable
  - baut auf Google File System auf (große Files, verteilt),
  - Anwendungen sind Google Maps, Google Bücher, YouTube, Google Earth oder Google Cloud Datastore als Teil der Google Cloud Platform.
- Beispiel Amazon Dynamo
  - Cloud Datenbank,
  - unterstützt auch Dokument- und Key-Value Datenbanken.

## NoSQL Typen

---

### Dokumentenorientierte Datenbanken

- Dokumente bilden Grundlage der Speicherung.
- Dokumente sind eindeutig identifizierbar, bestehen aus Key-Value-Paaren mit frei festlegbarem Schema.
- Beispiel CouchDB.
- Beispiel MongoDB.

## NoSQL Typen

---

### Dokumentenorientierte Datenbanken CouchDB

- Verwendet für Webseiten oder in Facebook-Anwendungen, in Ubuntu z.B. für Adressen und Lesezeichen.
- Speichert Daten als Dokumente, die aus JSON-Objekten bestehen (früher XML).
- Relationen können über eindeutige ids zwischen Dokumenten hergestellt werden  
→ daher sind ER-Modelle auch modellierbar.

```
{  
  "_id": "biking",  
  "_rev": "AE19EBC7654",  
  
  "title": "Biking",  
  "body": "My biggest hobby is mountain biking.",  
  "date": "2009/01/30 18:04:11"  
}
```

```
{  
  "_id": "bought-a-cat",  
  "_rev": "4A3BBEE711",  
  
  "title": "Bought a Cat",  
  "body": "I went to the pet store and bought a cat.",  
  "date": "2009/02/17 21:13:39"  
}
```

```
{  
  "_id": "hello-world",  
  "_rev": "43FBA4E7AB",  
  
  "title": "Hello World",  
  "body": "Well hello and welcome to my first document.",  
  "date": "2009/01/15 15:52:20"  
}
```

## NoSQL Typen

### Dokumentenorientierte Datenbanken CouchDB

- Queries via “views” (function) und map-reduce-functions.
- Ergebnisse in B-Bäume.

```
function(doc) {  
  if(doc.date && doc.title) {  
    emit(doc.date, doc.title);  
  }  
}
```



```
{  
  "total_rows": 3,  
  "offset": 0,  
  "rows": [  
    {  
      "key": "2009/01/15 15:52:20",  
      "id": "hello-world",  
      "value": "Hello World"  
    },  
    {  
      "key": "2009/01/30 18:04:11",  
      "id": "biking",  
      "value": "Biking"  
    },  
    {  
      "key": "2009/02/17 21:13:39",  
      "id": "bought-a-cat",  
      "value": "Bought a Cat"  
    }  
  ]  
}
```

UNIT 0x10

XML

## Lernziele

---

- **Herkunft bzw. Vorgänger von XML kennen**

Beides ist wichtig, da viele XML-Dokumente (auch die XML-Spezifikation selbst) Bezug darauf nehmen

- **Syntax von XML kennen**

XML-Dateien sind strukturierte Text-Dateien. Sie sollen die Strukturelemente und den Aufbau eines gültigen (wohlgeformten) XML-Dokumentes kennen.

- **Anwendungsgebiete für XML kennen**

XML wird in vielen Bereichen eingesetzt, hat dort aber einen anderen Namen

- **XML im Ansatz bewerten können**

Sie sollten Vor- und Nachteile von XML kennen



## Historie

---

- **XML = eXtensible Markup Language**
- **Ausgangspunkt: Textsatz**
- **Schreibmaschinen-Zeitalter:**

Manuskripte wurde auf Schreibmaschine getippt  
Anweisungen wie „Fettdruck“, „Kursiv“ an Setzer als handschriftliche Kommentare (bzw. als *Markup* - engl. für Textauszeichnung, abgeleitet von „*marking up*“)
- **Computer-Zeitalter:**

zunächst wie Schreibmaschine - nach Ausdruck handschriftliche *Markups*  
Prozess trägt Möglichkeiten der EDV aber nicht Rechnung!  
daher: spezielle *Markups* in Text integriert (formalisierte *Markup*)

# Historie

---

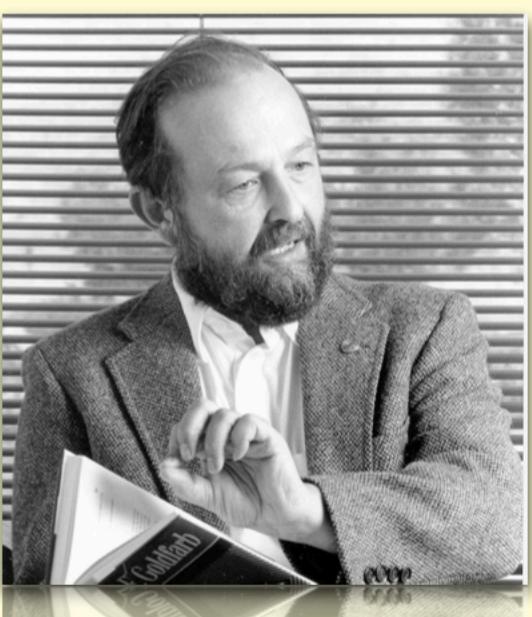
Jahrzehnt	Dokument-Formate	Programmierung	Netzwerke
60er 70er	Generalized Markup Language ( <b>GML</b> ) von IBM (Goldfarb, Mosher and Lorie)	Pascal, C, Simula 67	TCP/IP (Kahn und Cerf)
Lochkarten (80 Spalten), Assemblerprogrammierung, Bildschirme mit 80x12 Zeichen			

**GML**  
**:h1.** Erstes Kapitel: Einführung  
**:p.** GML unterstützt hierarchische Container, z.B.  
**:ol**  
**:li.** geordnete Listen,  
**:li.** ungeordnete Listen,  
**:li.** Definitionslisten  
**:eol.**  
 sowie einfache Strukturen.  
**:p.** Die vereinfachte Auszeichnung erlaubt es, abschließende Formatierungsbefehle wie zum Beispiel für die Elemente "h1" oder "p" auszulassen.

Strikte Trennung von

- **was** (Überschrift, Absatz, Aufzählung etc.) und
- **wie** (Art der Darstellung, z.B. Font, Zeilenabstand etc.)
- GML wurde von der *Document Composition Facility* (eine Art Compiler) in IBM's Druckersprache SCRIPT/VS übersetzt
- **Profile bestimmen, wie Dokument schlussendlich dargestellt wird**  
Profile gab es z.B. für Laser- und Nadeldrucker sowie diverse Bildschirme

# Historie



Charles F. Goldfarb

## Übergang zum Deskriptiven Markup

*That project required integrating a **text editing** application with an **information retrieval** system and a **page composition** program. The **documents** had to be kept in a repository from which they **could be selected by queries**. The selected documents could be revised with the text editor and returned to the data base, or rendered for presentation by the composition program.*

*Standard stuff for SGML systems today, perhaps, but far from the way most people thought about document processing in 1969. So far, in fact, that the applications we needed to integrate were not only not designed to work together, they couldn't even run on the same operating system. Fortunately, we had access to CP-67, an early hypervisor that allowed multiple concurrent operating systems to run on the same computer and share files. The problem was that, even when Ed Mosher, Ted Peterson, and I finally got the programs to talk to one another, we found they each required different **procedural markup** in the document files.*

**Quelle:** <http://www.sgmlsource.com/history/roots.htm>

# Prozedurales vs. Descriptives Markup

---

- **Prozedurales Markup**

*Domain Specific Language* (DSL) für den Textsatz. Instruktionen an einen „Word-Processor“ sind im Text eingebettet; Prozeduren, Schleifen und Makros sind oft möglich. Typische Beispiele sind **TeX** und **Postscript**.

- **Deskriptives Markup**

Ein bestimmter Text-Abschnitt wird mit einem *Markup* markiert, es wird nicht festgelegt, wie er konkret verarbeitet werden soll. Dadurch erreicht man eine **Entkopplung der Gliederung eines Dokumentes von seiner konkreten Darstellung**. Ein Typisches Beispiel ist HTML, wo die konkrete Darstellung in ein CSS-Dokument aus gegliedert wird.

# Historie

---

Jahrzehnt	Dokument-Formate	Programmierung	Netzwerke
80er	ISO-Standard Generalized Markup Language ( <b>SGML</b> ) <b>TeX</b> (Donald E. Knuth)	OOP, C++, GUIs	BITNET, EARN, CSNET, NSF-NET

PCs lösen Großrechner ab; Festplattenspeicher; Grafische Benutzeroberflächen (WYSIWIG)

```

<!-- Copyright (c) 2011 by Heini. This is free software. -->
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<article>
  <sect1 id="introduction" lang="en">
    <title>Introduction & Basics</title>
    <para>
      Hello world!
    </para>
  </sect1>
</article>

```

**SGML**

SGML beschreibt Syntax auf zwei Ebenen

- **Allgemeine Syntax** - Prinzipieller Aufbau von Dokumenten
- **Spezielle Syntax** - Document-Type-Definition (DTD)

Legt fest, welche Tags mit welchen Attributen erlaubt sind und wie diese geschachtelt werden dürfen

# Historie

---

Jahrzehnt	Dokument-Formate	Programmierung	Netzwerke
90er	<b>MS-Word</b> als de facto Standard, SGML nur Insidern bekannt	JAVA, komponenten-orientierte Programmierung	WWW wird Killer-applikation und ersetzt <i>ressource discovery systems</i> (Suchsysteme), Industrie entdeckt Internet:
<b>HTML</b> (Tim Berner-Lee): einfache Handhabung von Dokumenten im Netz			

```
<!-- Copyright (c) 2011 by Heini Hein. This is free software. -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>Introduction</title>
  </head>
  <body>
    <h1>Introduction</h1>
    <p align="left">This introduction covers the following topics:
      <ul>
        <li>XML
        <li>HTML
      </ul>
    Enjoy!
    </p>
  </body>
</html>
```

**HTML**

HTML ist eine Anwendung von SGML!

# Historie

---

Jahrzehnt	Dokument-Formate	Programmierung	Netzwerke
2000er	<b>XML (1998)</b> wird die technische Basis einer Integration von Dokumenten, Medien, Datenverarbeitung und Kommunikationsnetzen		

```

<?xml version="1.0"?>
<!-- Copyright (c) 2011 by Heini Hein. This is free software. -->
<!DOCTYPE MyFancyDoc SYSTEM "http://www.my.org/MyFancyDoc.dtd">
<hochschule name="FH Aachen">
  <fachbereiche>
    <fachbereich name="Medizintechnik und Technomathematik">
      <studiengaenge>
        <studiengang>
          <name>Scientific Programming</name>
          <grad>B.Sc.</grad>
          <dauer>6</dauer>
        </studiengang>
        <studiengang>
          <name>Technomathematik</name>
          <grad>M.Sc.</grad>
          <dauer>4</dauer>
        </studiengang>
      </studiengaenge>
    </fachbereich>
  </fachbereiche>
</hochschule>

```

**XML**

**XML ist Vereinfachung von SGML**

## Zentrale Anwendungen für XML

---

- **Stetig wachsende Nutzung des Internet für e-Commerce**
  - Business-to-Business (B2B)
  - Business-to-Computer (B2C)
- **Dadurch: zunehmende Bedeutung des elektronischen Datenverkehrs**
  - Hier hat sich XML zum de facto-Standard entwickelt
- **Datenquelle: meist Relationale Datenbankmanagementsysteme**
- **Zwei Techniken für uns besonders relevant**
  - **Definition und Darstellung von Sichten auf eine Datenbank mit XML**  
Technologien: Document Type Definition (DTD) und XML-Schema
  - **Definition eines kompletten Datenbank-Schemas in XML**  
Technologien: XPath und XQuery

## Standard

---

- **Herausgeber ist das *Word Wide Web Consortium (W3C)***
  - W3C spricht von „Recommendation“, nicht von Standard
- **Zwei Versionen sind relevant**
  - Version 1.0 vom 28.11.2008 - siehe <http://www.w3.org/TR/2008/REC-xml-20081126/>  
Auf diese Version stützt sich diese Vorlesung
  - Version 1.1 vom 16.08.2006 - siehe <http://www.w3.org/TR/2006/REC-xml11-20060816/>  
Änderungen sind eher marginal und werden hier nicht behandelt
- **Spezifikation beschreibt**
  - Klasse von Datenobjekten (XML-Dokumente) und
  - teilweise das Verhalten eines XML-Prozessors, einem Software-Modul zum Lesen von XML-Dokumenten und zum Zugriff auf dessen Inhalt und Struktur

# Syntax - Überblick

- Processing Instructions
- Kommentare
- Tags
- Text
- Entity-Referenzen
- weiteres .... (später)

```
<?xml version="1.0"?>
<!-- Copyright (c) 2011 by Heini. --&gt;
&lt;hs&gt;
  &lt;fachbereiche&gt;
    &lt;fb&gt;
      &lt;name&gt; Medizintechnik &amp; Technomathematik &lt;/name&gt;
      &lt;studiengaenge&gt;
        &lt;studiengang&gt;
          &lt;name&gt; Scientific Programming &lt;/name&gt;
          &lt;grad&gt; B.Sc. &lt;/grad&gt;
          &lt;dauer&gt; 6 &lt;/dauer&gt;
        &lt;/studiengang&gt;
        &lt;studiengang&gt;
          &lt;name&gt; Technomathematik &lt;/name&gt;
          &lt;grad&gt; M.Sc. &lt;/grad&gt;
          &lt;dauer&gt; 4 &lt;/dauer&gt;
        &lt;/studiengang&gt;
      &lt;/studiengaenge&gt;
    &lt;/fb&gt;
  &lt;/fachbereiche&gt;
&lt;/hs&gt;</pre>
```

## Syntax – Tags

---

- Ein XML-Dokument enthält **Tags**
- Tags beginnen mit "<" und enden auf ">"; man unterscheidet
  - öffnende Tags; z.B. `<fachbereich>`
  - schließende Tags; z.B. `</fachbereich>`
  - Kurztags; z.B. `<fachbereich/>`
- Tag-Namen dürfen aus
  - Buchstaben,
  - Ziffern,
  - Minuszeichen, Punkt, Doppelpunkt und Unterstrichen bestehen;
- Tags müssen mit einem
  - Buchstaben,
  - Unterstrich oder
  - Doppelpunkt beginnen
- Umlaute sind erlaubt -> Unicode!

**Exakte Festlegungen für gültige Namen in XML  
finde Sie hier: [http://www.w3.org/TR/2008/  
REC-xml-20081126/#sec-common-syn](http://www.w3.org/TR/2008/REC-xml-20081126/#sec-common-syn) (s.n.F)**

# Namensregeln

- Hier zum Nachschlagen die XML-Namensregel

```
NameStartChar ::= ":" | [A-Z] | "_" | [a-z] | [#xC0-#xD6] | [#xD8-#xF6] |
[ #xF8-#x2FF] | [#x370-#x37D] | [#x37F-#x1FFF] |
[#x200C-#x200D] | [#x2070-#x218F] |
[#x2C00-#x2FEF] | [#x3001-#xD7FF] | [#xF900-#xFDCF] |
[#xFDF0-#xFFFFD] | [#x10000-#xEFFFF]
NameChar      ::= NameStartChar | "-" | "." | [0-9] | #xB7 |
[#x0300-#x036F] | [#x203F-#x2040]
Name          ::= NameStartChar (NameChar)*
```

**Tipp:** Eine Übersicht über alle Unicode-Zeichen finden Sie hier:  
<http://www.utf8-zeichentabelle.de/>

## Syntax – Tags

- **Tags sind case-sensitive;**
  - d.h. `<b>` und `<B>` sind unterschiedliche Tags
- **Tags dürfen nicht mit xml beginnen** (auch nicht mit XML, xMI etc.)

```
<xml-test>Ein Text</xml-test>
```



- **Zu jedem öffnenden Tag existiert ein passendes schließendes Tag**
  - in HTML erlaubt!

```
<ul>  
  <li>Eins  
  <li>Zwei  
</ul>
```



```
<p>Erster Abschnitt  
<p>Zweiter Abschnitt  
</p> <!-- Welcher wird beendet? -->
```



- **Öffnende und schließende Tags sind korrekt balanciert**
  - in HTML erlaubt!

```
<b>  
  <it>fett und kursiv  
  </it>  
</b>
```



## Syntax – Elemente

- Als **Element** bezeichnet man den Bereich vom öffnenden zum schließenden Tag (einschl. der Tags selbst)
- Der Name des öffnenden Tags ist der **Typ** des Elements
- Ein XML-Dokument besteht aus verschachtelten Elementen:

```
<studiengaenge>
    <studiengang>
        <name>Scientific Programming</name>
    </studiengang>
</studiengaenge>
```

The XML code is shown with some parts highlighted in yellow boxes. The outermost tag, `<studiengaenge>`, has the word "studiengaenge" written twice to its right. The inner tag, `<studiengang>`, has the word "studiengang" written once to its right. Inside the `<studiengang>` tag, the `<name>` tag contains the text "Scientific Programming", which is highlighted in a yellow box, and has the word "name" written once to its right.

- Der Bereich zwischen den Tags wird **Inhalt** genannt
- Kurtztags sind folglich Elemente ohne Inhalt
- Der Inhalt eines Elements besteht aus Elementen oder Daten
  - Ein Element beschreibt daher einen Baum

## XML-Dokument als Baum

### B XML-Element als Baum (XML-Baum)

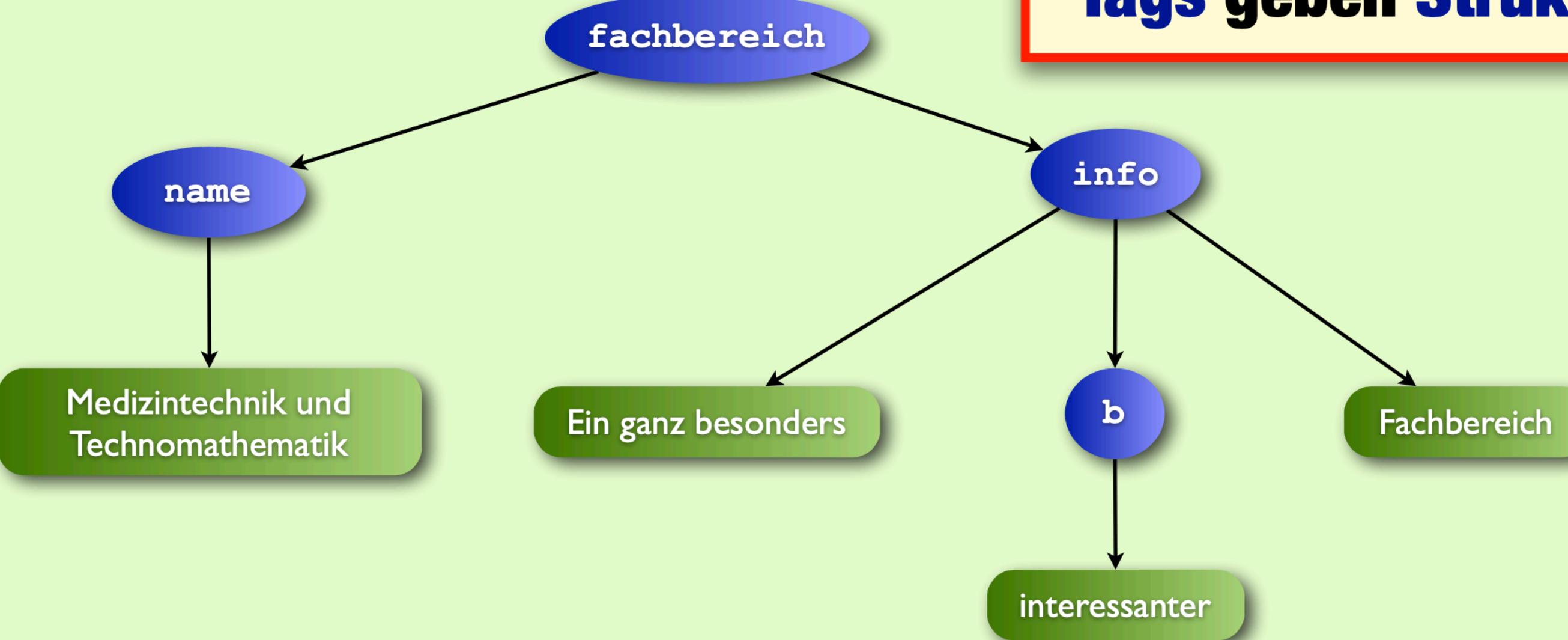
XML-Fragment:

```
<fachbereich>
  <name>Medizintechnik und Technomathematik</name>
  <info>Ein ganz besonders <b>interessanter</b> Fachbereich</info>
</fachbereich>
```

Inhalt

Zugehöriger Baum:

**Tags geben Struktur!**



## XML-Dokument als Baum

---

- **Wohlgeformte XML-Dokumente besitzen genau ein Element - das Wurzel-Element oder auch Dokumenten-Element**
  - Ein XML-Dokumente repräsentiert daher einen Baum (XML-Baum)
  - Auch Graphen sind möglich (über „Pointer“ - dazu später mehr)
- **An den Blättern eines XML-Baumes stehen**
  - **Kurztags** (leere Elemente),
  - **Text** (*PCDATA - parsed character data - analysierter Text*) oder
  - **CDATA-Knoten** (nicht analysierter Text)

# Syntax – Text

---

- **Textknoten enthalten beliebigen Text**
- **Aber: Symbole „<“ und „&“ sind nicht erlaubt stattdessen müssen Entity-Referenzen verwendet werden**
  - **Entities** sind Makro-Definitionen; Entity-Referenzen verweisen darauf
  - Entity-Referenzen werden mit "&" eingeleitet und enden auf ";" z.B. wird "&amp;" zum kaufmännischen Und "&" expandiert
  - Benutzerdefinierte Entities sind möglich (dazu später mehr)

Vordefinierte Entities	
Entity	Zeichen
<b>amp</b>	&
<b>gt</b>	>
<b>lt</b>	<
<b>quot</b>	"
<b>apos</b>	'

**D Entity-Referenzen**

`<name>Medizintechnik & Technomathematik</name>`

Ist illegal, da XML-Prozessor „&“ als Start einer Entity-Referenz interpretiert. Korrekt ist:

`<name>Medizintechnik &#amp; Technomathematik</name>`

- **Zeichenreferenzen verweisen auf spezifisches Unicode-Zeichen**
  - `&#x0030;` für das Zeichen „0“ bzw.
  - `&#50;`, falls der Zeichencode dezimal angegeben werden soll

## Syntax - CDATA

---

- **Der XML-Prozessor zerlegt eine XML-Datei in lexikalische Bestandteile - Inhalt von Elementen wird daher immer analysiert**
  - Dieser kann Mixtur aus Text, weiteren Elementen und Entity-Referenzen sein
  - Man spricht von **PCDATA** (*parsed character data*)
- **CDATA-Knoten werden vom XML-Prozessor niemals zerlegt**
- **Sie starten mit "<! [CDATA[ " und enden auf " ] ]>"**
  - Text zwischen diesen Markierungen wird 1:1 übernommen
  - "<" und "&" dürfen in CDATA-Knoten direkt verwendet werden
  - Text darf lediglich die Symbolfolge " ] ]>" nicht enthalten

## Syntax - CDATA

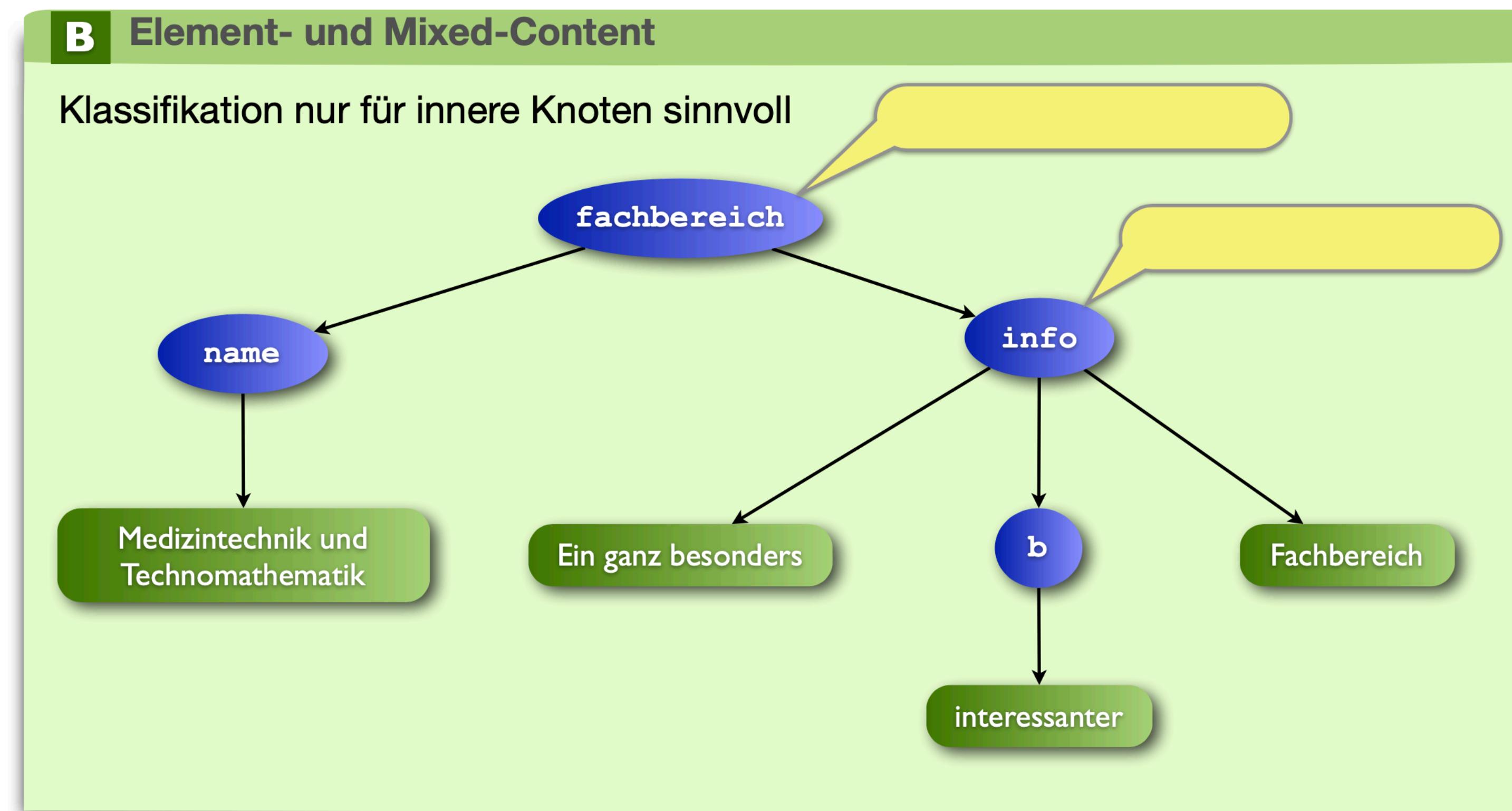
- Beispiel für einen CDATA-Knoten

### B CDATA

```
<algorithm>
  <name>Bubble Sort</name>
  <implementation language="JAVA">
    <![CDATA[
      public void bubbleSort(int feld[]) {
        for (int i=feld.length-1 ; i>=0 ; --i)
          for (int j=1; j<=i ; ++j)
            if (feld[j]<feld[j-1]) {
              int t = feld[j] ; feld[j]=feld[j-1] ; feld[j-1]=t;
            }
      }
    ]]>
  </implementation>
</algorithm>
```

# Inhalt von Elementen: Klassifikation

- Je nach Art der Kindknoten unterscheidet man Elemente mit
  - **Element Content**: Alle Kind-Knoten sind Element-Knoten
  - **Mixed Content**: Kind-Knoten sind Element-, Text- oder CDATA-Knoten



# Syntax – Attribute

- Elemente können mit **Attributen** versehen werden
  - Definitionen innerhalb des öffnenden Tags bzw. des Kurztags
  - Format:
    - **Attributname="Wert"** oder
    - **Attributname='Wert'**
    - nicht erlaubt ist Mischen von " und ' ; z.B. **Attributname='Wert'" X**
  - Entity-Referenzen innerhalb der Werte sind zulässig (ggf. notwendig!)
  - Attributname darf nur einmal verwendet werden
  - Attribute sind ungeordnet

## B Attribute

```
<fachbereich name="Medizintechnik & Technomathematik" nr="9">
  <dekan name="Volker Sander" raum="O1G31"/>
  <prodekane>
    <prodekan name='Andreas Terstegge' />
    <prodekan name="Karl Ziemons" raum="O1F19" />
  </prodekane>
</fachbereich>
```

# Attribut oder Element?

- **Daten besser als Attribut**

```
<dekan name="Volker Sander" raum="O1G31" />
```

oder als Element

```
<dekan>
  <name>Volker Sander</name>
  <raum>=O1G31</raum>
</dekan>
```

speichern?

- **Keine offizielle Festlegung durch XML-Standard, aber Hinweise:**

Attribut	Element
<b>ungeordnet</b> (Element-Knoten besitzt Attributmenge)	<b>geordnet</b> (Element-Knoten hat mehrere Element-Knoten als Kinder)
<b>1:1 - Beziehung</b> (Ein Attribut nur einmal pro Element)	<b>1:n - Beziehung</b> (Element kann wiederholt als Kindknoten auftreten)
<b>Wert unstrukturiert</b> (einfacher Text)	<b>Wert strukturiert</b> (Baum)

## Reservierte Attributnamen (Beispiele)

---

- **xml:lang**
  - Sprache des Inhalts
  - Beispiel: `<p xml:lang="de">Aufgabe 1</p>`
- **xml:space**
  - Leerräume im Inhalt B
  - Beispiel: `<p xml:space="[preserve/default]">Aufgabe 1</p>`
- **xml:id**
  - Elementbezeichner (dokumentweit eindeutig)
  - Beispiel: `<p xml:id="Sektion_1">Sektion 1</p>`
- **xml:base**
  - Basis-URL (für relative Links)
  - Beispiel: `<ul xml:base="http://www.example.com/docs/2012/">`  
`<li><a href="XML_1">XML-Attribute</a></li>`  
`</ul>`

# Leere Elemente?

## B Leere Elemente

```
<name>
  <first>Donald</first>
  <last>Knuth</last>
</name>
```

vs.

```
<name>
  <first>Donald</first>
  <middle/>
  <last>Knuth</last>
</name>
```

- **leere Elemente können Attribute haben; z.B.**

```
<name>
  <first>Donald</first>
  <middle status="unknown"/>
  <last>Knuth</last>
</name>
```

- **evtl. von DTD oder XML-Schema vorgeschrieben (dazu später)**

## Syntax – PIs

---

- **Procssing-Instructions sind Anweisungen an einen XML-Prozessor**
  - von daher u.U. spezifisch für bestimmte Implementierungen eines Prozessors
- **XML-Dokument sollte mit „XML-Instruction“ beginnen**
  - sie gibt Auskunft über die verwendete **Version** von XML und die **Zeichenkodierung**
  - ist die Zeichenkodierung weder UTF-8 noch UTF-16, so ist die XML-Instruction Pflicht!  

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```
  - für XML 1.1 obligatorisch!
- **xmlstylesheet ist eine weitere Processing Instruction**
  - mit ihr können stylesheets zur Darstellung von xml-Dateien spezifiziert werden
  - wird hier nicht behandelt!

## Syntax – Kommentare

- **Kommentare beginnen mit „<!--“ und enden auf „-->“**

- dürfen nicht vor der XML-Instruction stehen
- sind nur ausserhalb des Markups erlaubt

```
<name <!-- Ein Name --> >Heinrich</name>
```

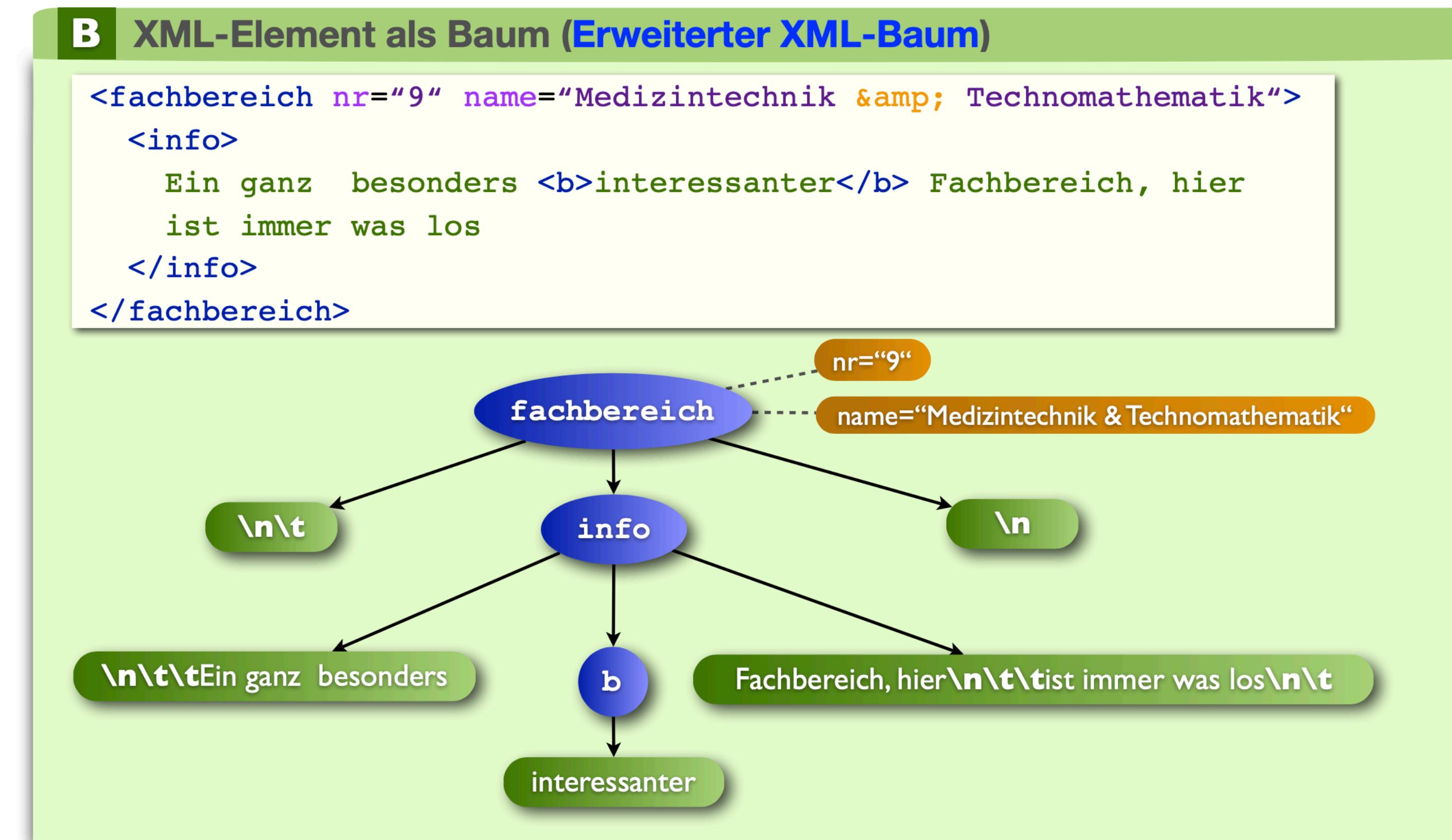


- dürfen die Zeichenfolge „--“ nicht enthalten
- Entity-Referenzen werden nicht expandiert - „<“ und „&“ als Zeichen sind erlaubt

```
<!--  
Ein Kommentar darf sich über mehrere  
Zeilen erstrecken. & wird nicht zu ✓  
& ersetzt.<ja!>  
-->
```

# XML und Whitespaces

- **Whitespaces** bleiben in XML erhalten
  - Whitespaces sind: Leerzeichen (#x20), Tabulator (#x9), Zeilenende (#xA) und Zeilenvorschub (#xD)



## XML und Zeilenende

---

- **Kodierung des Zeilenendes variiert von Betriebssystem zu Betriebssystem:**
  - UNIX: #xA (Zeilenvorschub)
  - OSX (Mac): #xD (Wagenrücklauf)
  - DOS und Windows: #xD#xA (Wagenrücklauf und Zeilenvorschub)
- **XML-Prozessor reicht an Anwendung nur Zeilenvorschub #xA weiter!**
  - Vorsicht! Wurde in XML 1.1 geändert

## Wohlgeformtes XML

---

- Ein Dokument beinhaltet wohlgeformtes XML (*well-formed XML*) wenn es die XML-Syntaxregeln respektiert; insbesondere muss gelten:
  - Es gibt genau ein Wurzel-Element
  - Element- und Wurzelnamen entsprechen den Festlegungen in <http://www.w3.org/TR/2008/REC-xml-20081126/#sec-common-syn>
  - Zu jedem Start-Tag gibt es ein korrespondierendes End-Tag  
(Beachte XML ist case-sensitive)
  - Elemente überlappen sich nicht (korrekte Schachtelung)
  - Ein Element enthält keine Attribute mit identischen Namen
  - Jedes Attribut hat einen Wert

## Flexibilität

---

- Dokumente, die sich an obige Syntaxregeln halten, nennt man **wohlgeformtes XML**
- Wohlgeformtes XML bildet nur einen groben Rahmen zur Darstellung von Daten (keine vordefinierten Elemente!)
- Um Daten wieder auffindbar bearbeiten zu können reichen diese Festlegungen nicht aus, denn
  - Man muss wissen, wo was steht - dafür muss Dokument-Struktur bekannt sein!
  - Man muss wissen, wie die Daten an den Blättern kodiert sind - dafür braucht man so etwas wie Daten-Typen!
- Reines XML oft Basis für andere Sprachen
  - spezielle Sprachen schränken XML bewusst ein (z.B. durch kontextfreie Grammatik)
  - Später: DTDs und XML-Schema -> XML als **Meta-Sprache**

# Anwendungen von und mit XML

---

- **Primär: Datenaustausch**
- **Textsatz (klassische, von SGML „geerbt“)**
  - DocBook
  - Hier auch MathML (mathematische-) und CML (chemische Formeln)
- **Word Wide Web**
  - XHTML als konsistenter Ersatz von HTML
  - SOAP - *Simple Object Access Protocol* - Nachrichtenaustausch zwischen Anwendungen
  - RSS - *Really Simple Syndication*
- **Austausch von Unternehmensinformationen** (z.B. Jahresabschlüsse)
  - XBRL - *Extensible Business Reporting Language*
- **Katalogisierung von Büchern, CDs etc.**
  - Dublin Core
- **Kodierung von Grafiken und Filmen**
  - SVG - *Scalable Vector Graphics* und SMIL - *Synchronized Multimedia Integration Language*
- **Dokumente konvertieren**
  - XSLT - *Extensible Stylesheet Language Transformations*
  - XSL-FO - *XSL Formatting Objects*
- **Konfiguration von Software (z.B. Hibernate, Android)**
- **Abfragen von XML-Dokumenten**
  - XPATH, X-Query

## Bewertung – positiv

---

- **XML-Dateien sind Textdateien**
  - mit einfachen Mitteln (Editor) lesbar
  - eventuell selbsterklärend (falls Namen von Tags und Attributen klaren Hinweis auf Semantik geben)
- **XML ist flexibel**
  - Eigene „Datensprache“ oder Adaption einer bestehenden
  - Einfache Erweiterung existierender Sprachen (z.B. Hinzufügen von Elementen oder Attributen)
- **XML eignet sich gut für den Datenaustausch**
  - XML ist ein offener Standard und lässt sich leicht transformieren
- **XML erfreut sich einer umfangreicher Werkzeugunterstützung**
  - Validierende XML Prozessoren (Parser), XSLT-Prozessoren, XML-Editoren etc.
  - Diverse Programmierbibliotheken bis hin zur vollständigen Integration in Programmiersprachen (z.B. via JAVA-Annotationen)

## Bewertung - negativ

---

- **Konflikte zwischen Spezifikation**
  - insbesondere z.B. XML-Schema vs. Relax NG vs. DTD
  - Welche Technologie einsetzen?
- **Spezifikationen gelten z.T. als zu kompliziert**
  - z.B. XML-Schema-Spezifikation des W3C oder ODF - das Open Document Format
- **Integration in Browser noch nicht perfekt**
- **XML birgt Overhead**
  - **Speicherplatz**: klar, daher werden XML-Dateien häufig gepackt
  - **Laufzeit**: XML-Dateien müssen in interne Programmstrukturen (z.B. Objektgraphen) überführt werden

# GESCHAFFT...

PROF. DR. RER. NAT. ALEXANDER VÖß

FH AACHEN  
UNIVERSITY OF APPLIED SCIENCES