

## 2. Datenmodellierung mit dem Entity-Relationship-Modell (E/R-Modell, ERM)

### Zielsetzung des Kapitels:

- (Nicht nur „intuitive“) Einführung von Begriffen wie **Entity**, **Entitytyp**, **Wertebereich**, **Attribut**, **Schlüssel**(-kandidat), **Beziehung**, **Beziehungstyp**, Komplexität (**Kardinalität**) von **Beziehungstypen**
- Einführung von E/R-Diagramm, d.h. der **graphischen Darstellungsform** von E/R-Modellen: Syntax und Semantik
- Einüben des Lesens von E/R-Diagrammen und – vor allem – des selbständigen Erstellens von E/R-Modellen / E/R-Diagrammen (Übungen!!!)
- Verstehen, dass es nicht nur **eine** („die“) E/R-Modellierungsmethodik gibt, sondern **viele Varianten und Ausbaustufen** (sog. erweiterte E/R-Modelle, semantische Datenmodelle, SDM) sowie Verstehen einiger dieser Erweiterungen

## Literatur zum E/R-Modell\*

\* besser: zur **Modellierungsmethodik**

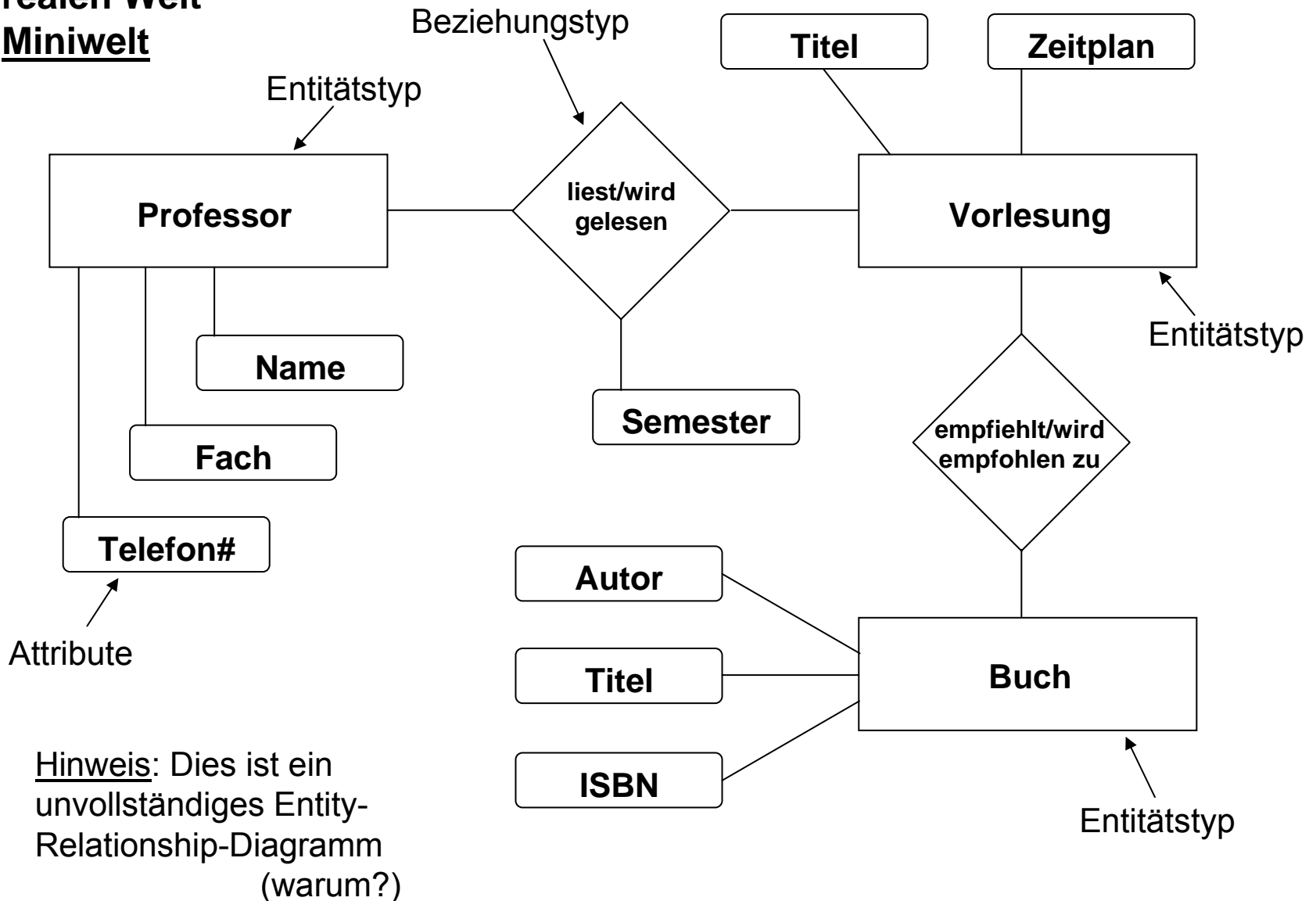
- a) Originalliteratur (vom „Erfinder“ der E/R-Modellierungsmethodik):  
P.P.Chen: The Entity-Relationship Model - Toward a Unified View of Data. ACM Transactions on Database Systems (TODS), Bd. 1, Nr. 1, März 1976, S. 9-36
- b) Praktisch in **jedem Datenbanklehrbuch**, „Urschleim“ der Datenbankforschung und -anwendung

## Warum überhaupt Modellierung auf „höherem Abstraktionsniveau“ (E/R-Diagramme) vs. direkter Modellierung in Datenbankmodellnotation?

Konkret am Bsp.: E/R-Modellierung vs. relationale Tabellendarstellung

1. E/R-Modell **unabhängig** von späterer Umsetzung in best. Form eines **Datenbankmodells** (hierarchisch, netzwerkorientiert, relational) und erst recht konkreten **Datenbanksystems** (DB2, Oracle, Informix ...) → **Flexibilität/Portabilität**
2. E/R-Modell **leichter verständlich/übersichtlicher**, auch wg. graphischer Notation, als z.B. relationale (tabellarische) Darstellung
3. **Mehr Semantik** in offensichtlicher Weise darstellbar im E/R-Modell als in relationalem Modell (Entity-Typen, Beziehungstypen, Kardinalitäten ...)   
↳ in relationalem Modell teils nur „auf Umwegen“ darstellbar (siehe auch nächste Folie)
4. E/R-Modell kann – viel besser als z.B. relationales Modell („Tabellensammlung“) – **Gesprächsgrundlage** bilden zwischen Anwendern/Fachabteilung und Informatikern („**anwendungsnah und trotzdem schon nah an der Datenbank**“)   
↳ hängt natürlich mit 2. und 3. zusammen

**Ausschnitt der  
realen Welt  
Miniwelt**



Aber merke: Datenbanksysteme unterstützen in aller Regel E/R-Modell **nicht direkt** („E/R-Modell = Datenbankmodell, ‘E/R-Sprache’ als Datenbanksprache“), sondern es muss nachfolgende **Umsetzung** E/R-Modell → Datenbankmodell erfolgen

- Relationales Modell „war schon da“, als E/R-Modell Mitte der 1970er erfunden wurde
- ‘E/R-Sprache’ u.U. schwierig zu erlernen, komplex; Vorsicht, dass nicht doch navigierend!



Jedoch: Forschung hat in vielfältiger Weise ‘E/R-Sprachen’ vorgeschlagen (meist sogar für erweiterte E/R-Modell), selten aber implementiert, noch seltener in Produkten Einzug gehalten (in Deutschland vor allem: TU BS; Produkt ehemals(?): Software AG)

## Entity/Entitytyp

- „Unter **Entität** (engl. entity) versteht man ein **bestimmtes**, d.h. von anderen **wohlunterscheidbares!** Objekt der realen Welt oder unserer Vorstellung. Dabei kann es sich um ein **Individuum**, einen **Gegenstand**, um einen **abstrakten Begriff** oder um ein **Ereignis** handeln.“  
[A. Meier: Relationale Datenbanken (1995), S. 14].
- Entitäten werden zu **Entitätstypen** (entity types) zusammengefasst. Entitytyp E ist Menge aller **möglichen** Entities e mit gleichen charakteristischen Merkmalen.
- Beispiel:  
Angestellter Meier und Angestellter Müller-Sockenschuss sind **Entities**.<sup>\*</sup> Haben gleiche charakterisierende Merkmale (Personalnummer, Name, Vorname, Anschrift, Gehalt ...) → bilde **Entitytyp** ANGEST, steht für Menge aller **möglichen** Angestellten

---

<sup>\*</sup> hier: Individuen

- Andere Beispiele:
  - Entitytyp ABTEILUNG: Menge aller **möglichen** Abteilungen  
(→ **abstrakter Begriff**)
  - Entitytyp GEBÄUDE: . . . (→ **Gegenstand**)
  - Entitytyp VERKAUFGSGESPRÄCH: . . . (→ **Ereignis**)
- **Verschiedene Entitytypen** müssen inhaltlich **nicht unbedingt disjunkt** sein: z.B. Entitytyp PERSON und Entitytyp STUDENT **oder**   
Entitytyp AUTOFAHRER und Entitytyp STUDENT 

Attribute/Attributwerte/Wertebereiche

Entitytypen besitzen Attribute, d.h. dem **Entitytyp E** wird eine nichtleere, endliche **Attributmenge A** zugeordnet (E:A)

Beispiele: E = ANGEST

A = {PNR, NAME, VORNAME, GEHALT, ANR ...}

oder

E = ABTEILUNG

A = {ANR, ABTEILUNGSBEZEICHNUNG, GEBÄUDE, ...}

dem einzelnen Entity e sind entsprechend **Attributwerte** zugeordnet, z.B.

Meier : {17, Meier, Alfons, 5000, 4 ...}

Allgemein:  $A = \{a_1, a_2, a_3 \dots, a_n\}$

$a_i$  sind die Attribute

$n$  endlich ( $n \geq 1$ )

Auf der Entityebene (Ausprägungsebene):

Die Attribut**werte** beschreiben das Entity  $e$ .

## Wertebereiche/Domains/Domänen

- Jedem Attribut  $a$  ist ein **Wertebereich** (Domain, Domäne) zugeordnet, den wir als **dom( $a$ )** bezeichnen.
- Beispiele:
  - $\text{dom}(\text{PNR}) = \text{INTEGER}$
  - $\text{dom}(\text{GEHALT}) = \{g \mid 1000 \leq g \leq 10000\}$
  - $\text{dom}(\text{HAARFARBE}) = \{\text{blond, braun, schwarz, grün}\}$

  
Integritätsfrage

Wertebereichsfestlegungen  
stellen (einfache) semantische  
**Integritätsbedingungen** dar !



## Schlüssel

Ziel: Eindeutige **Identifizierung** eines Entity  $e$  innerhalb seines Entitytyps  $E$ , also innerhalb der **möglichen** Entities gleichen Typs

Beispiel: ANGEST : {PNR, NAME, VORNAME, GEHALT, ANR}

$\vdots$   
 $E$

$\underbrace{\hspace{15em}}$   
 $A$

Gesucht  $K \subseteq A$ ,  $K$  nicht leer ( $K$  Schlüssel, bestehend aus Schlüsselattributen), so dass die Schlüsselattributwerte ein Entity eindeutig identifizieren

Im obigen Beispiel:

**$K = \{PNR\}$ , Personalnummer (z.B. 17) identifiziert Entity**  
 (hier: Angestellter Meier) eindeutig

**!** Voraussetzung: Personalnummer eindeutig, innerhalb des gesamten Unternehmens! → d.h. es reicht für die Entscheidung, ob eine Attributmenge Schlüssel ist, NICHT aus, nur konkret vorliegende Entities zu betrachten (wo die Schlüsseleigenschaft zufällig erfüllt sein **könnte**), sondern die Frage muss grundsätzlich und „zukunftsicher“ anhand der Semantik des Entitytyps entschieden werden

Annahme: Personalnummer **nicht** eindeutig innerhalb des Unternehmens, aber eindeutig innerhalb jeder Abteilung.

**K = {PNR, ANR}**, Schlüssel besteht also hier aus zwei Attributen

Weitere Schlüssel in unserem ANGEST-Beispiel?

Man fordert oft **Minimalität** des Schlüssels, d.h.  
 $\nexists K' \subset K : K' \text{ Schlüssel}$

Bei den Beispielen, die wir anschauen werden, wird diese (Minimalitäts-) Eigenschaft von K stets erfüllt sein.

(N.B.: Später (im relationalen Datenmodell) **muss** sie „von der reinen Lehre her“ erfüllt sein.)

Abschließend zu den Begriffen:

Es kann, auch unter Berücksichtigung der Minimalitätseigenschaft, **mehrere Schlüssel** für einen Entitytyp geben. Man\* bezeichnet die Schlüssel auch als **Schlüsselkandidaten** und wählt genau einen davon als sog. **Primärschlüssel** aus (bspsweise aufgrund seiner Bedeutung für das Unternehmen / die Miniwelt). Bsp.: PNR versus Sozialversicherungsnummer

\* der Datenbankadministrator

## Thema Künstliche Schlüssel / Künstlicher Primärschlüssel

In vielen Fällen wird (zumindest) der **Primärschlüssel künstlich eingeführt**, d.h. fällt bei der Sammlung von Attributen für einen Entitytyp nicht auf „natürliche Weise an“, z.B. Personalnummer, Abteilungsnummer, Gebäudenummer, Sozialversicherungsnummer, Teilenummer, Auftragsnummer sind künstlich. Gründe:

- a) „natürlicher“ Schlüssel **existiert nicht** oder setzt sich aus sehr vielen Attributen zusammen  
{NAME, VORNAME, GEBURTSDATUM, GEBURTSORT ...}  
↳ **Kompaktheit** des Schlüssels erwünscht, „Handlichkeit“
- b) „natürlicher“ Schlüssel existiert aus heutiger Sicht, aber nicht sicher, ob auch **Eigenschaft noch in Zukunft**  
z.B. {NAME, VORNAME} für ANGEST-Entitytyp in „5-Mann-Firma“
- c) Schlüssel soll auch als Medium für **Referenzierung** zwischen Entities benutzt werden → kompakter Schlüssel wichtig (siehe aber Kap. 5 „Relationale Datenbanken“)

## Beziehungen / Beziehungstypen

**Beziehungen** **r** (relationships) bestehen zwischen **einzelnen Entities** **e** und **e'** verschiedener oder gleicher Entitytypen **E** und **E'** wird später verallgemeinert

Beispiel: Entitytypen ANGEST (Angestellte) und PROJ (Projekte):  
 Zwischen einem Angestellten **a** und einem Projekt **p** besteht dann eine **Beziehung** **r<sub>1</sub>**, wenn der Angestellte **an** dem Projekt arbeitet.  
 oder: ... **Beziehung** **r<sub>2</sub>**, wenn der Angestellte das Projekt **leitet**.  
 oder: Zwischen einem Angestellten **a'** und einem **a''** besteht dann die **Beziehung** **r<sub>3</sub>**, wenn **a'** Chef von **a''** ist.

**Beziehungstyp** **R** (relationship type) ist die Menge aller **möglichen Beziehungen** **r** zwischen **je einem** Entity **e<sub>i</sub>** der beteiligten Entitytypen **E<sub>i</sub>** (**i** = 1 ... **k**)

„eines Typs“

$$R = E_1 \times E_2 \times \dots \times E_k \quad (\text{kartesisches Produkt})$$

$$R = \{r = (e_1, e_2 \dots e_k) \mid e_1 \in E_1, \dots, e_k \in E_k\}$$

Beispiel:    ARBEITET\_AN    = ANGEST × PROJ  
                 LEITET                = ANGEST × PROJ  
                 IST\_CHEF               = ANGEST × ANGEST  
                 → **LIEFERT                = LIEFERANT × ARTIKEL × PROJ**

— **dreistelliger** Beziehungstyp;

allgemein können Beziehungstypen **k-stellig** sein,  $k \geq 2$

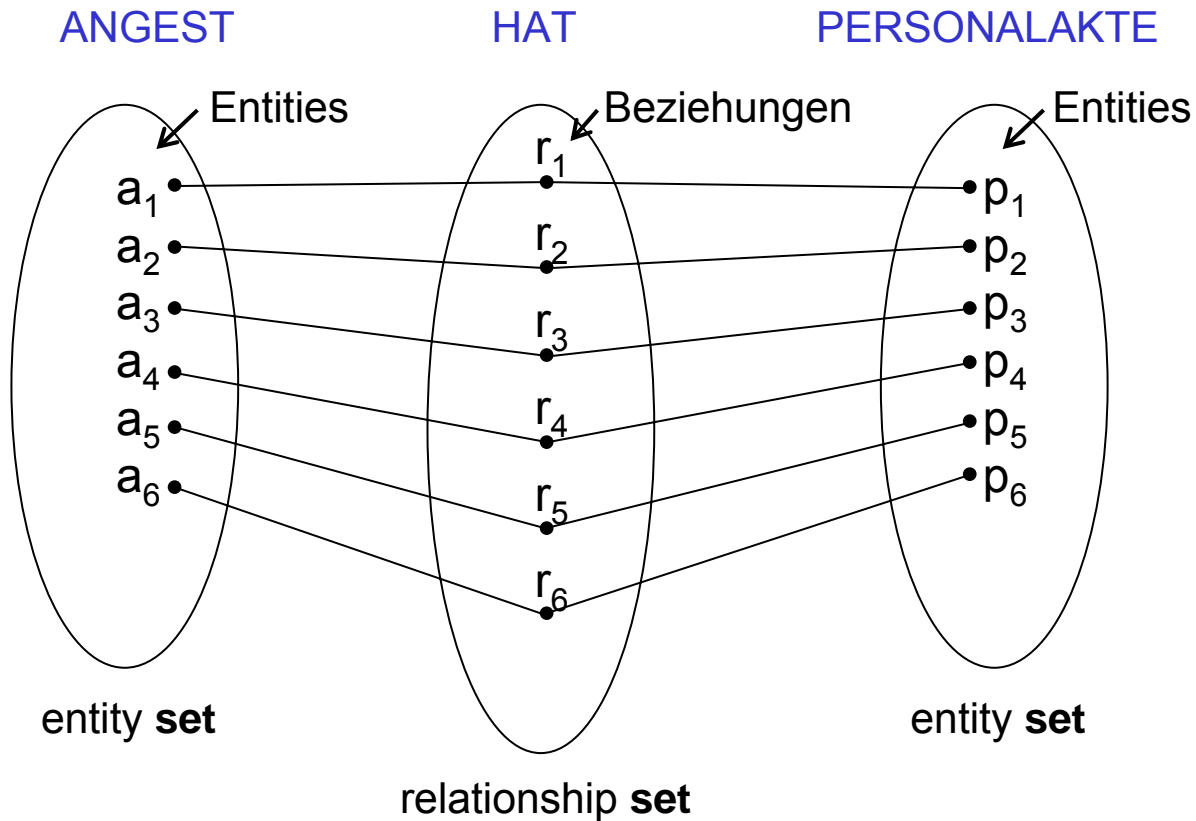
Hinweis: Für die Ablage in der Datenbank sind die **tatsächlich vorhandenen Entitymengen** (entity set) bzw. **Beziehungsmengen** (relationship set) von Interesse; diese sind Untermengen von E bzw. R:

entity set                 $ES \subseteq E$

relationship set     $RS \subseteq R$

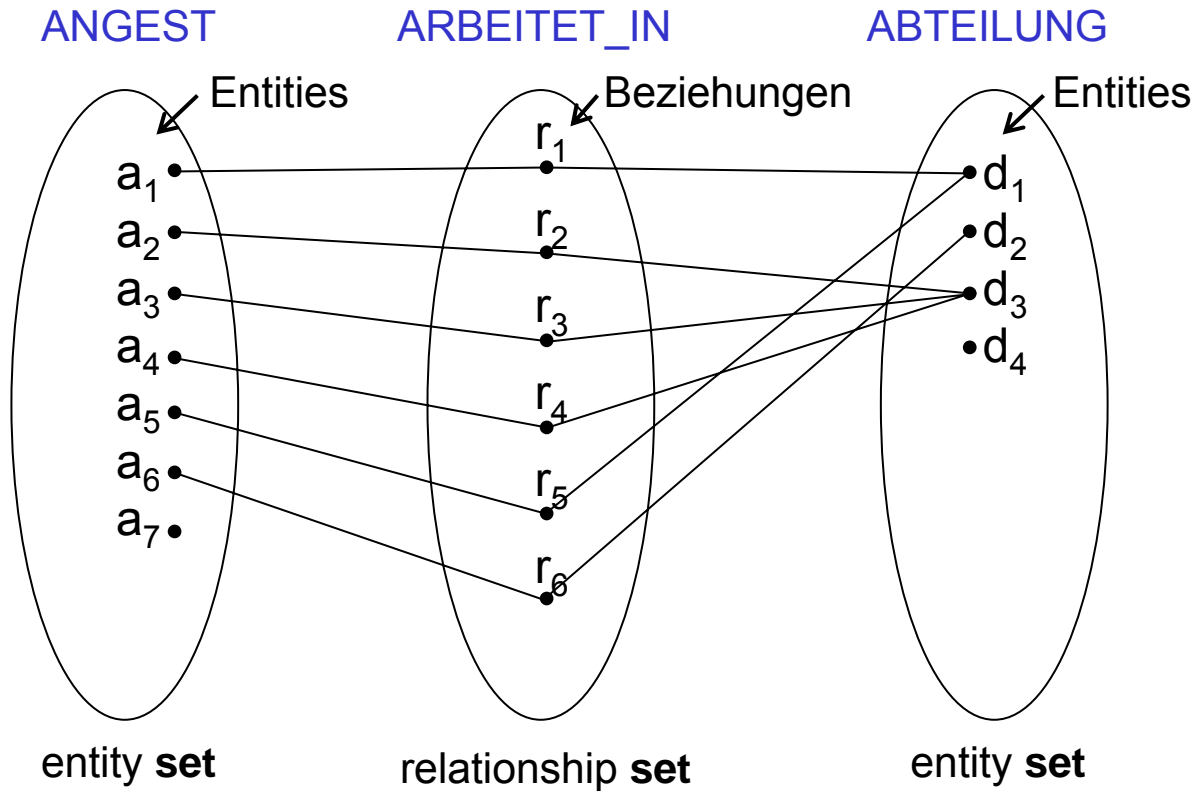
**ES und RS sind endliche Mengen** 😊

## Beispiele für Beziehungstypen (inkl. Klassifikation)



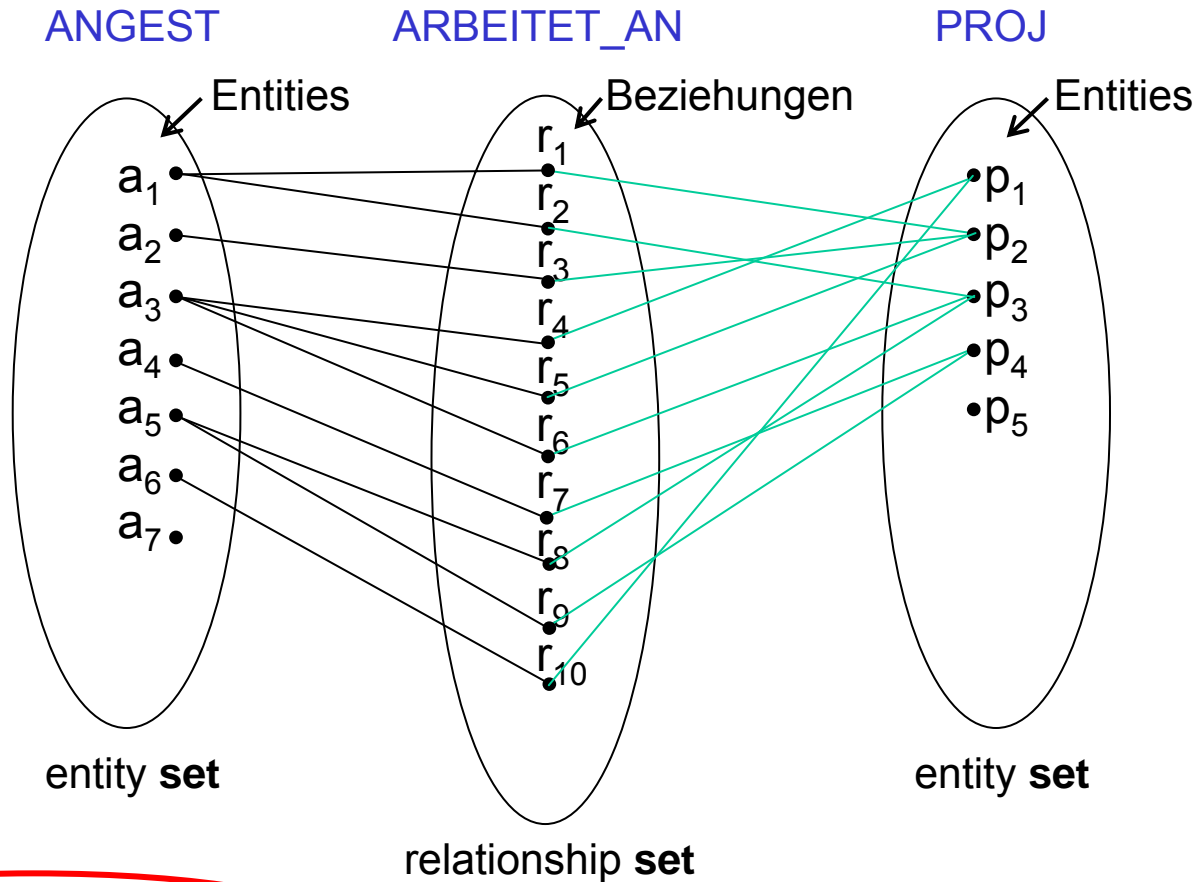
**1 : 1 – Beziehungstyp**

**1 : 1 relationship type**



**n : 1 – Beziehungstyp**

**n : 1 relationship type**



**$n : m$  – Beziehungstyp**

**$n : m$  relationship type**

many to many rel.ship type

⇒ Eine Beziehung  $r_i$  im Relationship Set für jedes real vorhandene „**Mitarbeitsverhältnis**“ zwischen einem Angestellten  $a_j$  und einem Projekt  $p_k$

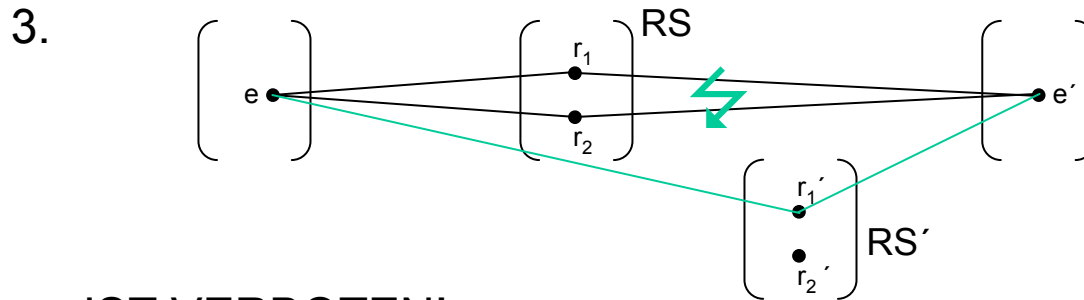


## Hinweis für Integritätsregeln bei dieser Darstellungsform (bzw. Beziehungstypen/-mengen allg.)

1. Von jeder Beziehung  $r_i$  geht **genau eine Kante** aus (beim zweistell. Beziehungstyp) „nach links und nach rechts“ (klar, da  $r_i = (e_j, e_k)$ )
2. **Nicht** von jedem Entity  $e$  **muss** eine Kante ausgehen innerhalb eines bestimmten Beziehungstyps:  $e$  steht aktuell (für **diesen** Beziehungstyp  $R$ !) in keiner Beziehung  $r$  zu einem  $e'$

Bsp.:

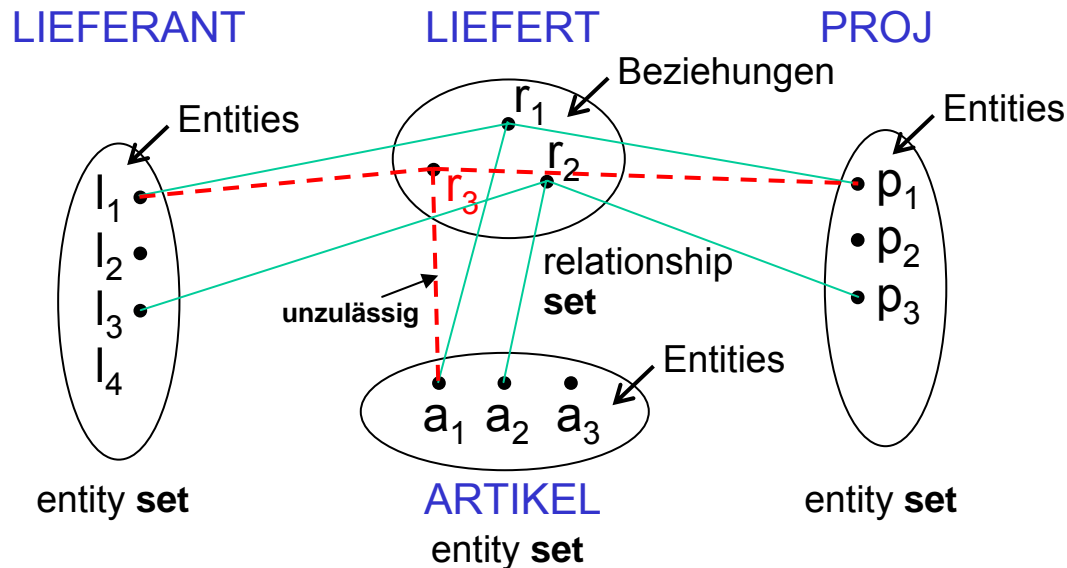
- Angestellter  $a_i$  arbeitet an keinem Projekt
- Projekt  $p_i$  hat keine Mitarbeiter



### IST VERBOTEN!

Würde bedeuten, dass **Mengeneigenschaft** des relationship set **verletzt** ist,  $r_1 = (e, e') = r_2$  beide in relationship set enthalten! (Oder, „nicht mathematisch“ gesprochen, wäre **ein Sachverhalt zweimal** dargestellt (redundant) → unerwünscht.)

## Mehrstellige Beziehungstypen ( $k \geq 3$ )



- **Lieferant**  $l_1$  liefert **Artikel**  $a_1$  für **Projekt**  $p_1$  etc. etc.
- Die Information, welche Artikel ein Lieferant liefert (d.h. **projektübergreifend**), ist hier implizit ebenfalls enthalten und **braucht nicht** zusätzlich separat dargestellt werden durch eigenen Beziehungstyp. → darf aber! **LIEFERT' = ARTIKEL  $\times$  LIEFERANT** (freilich redundant)

**Vorsicht:** Falls Semantik verschieden, benötigt man doch separate Beziehungstypen, z.B.: LIEFERT\* stellt dar, welche Artikel ein Lieferant konkret an welches Projekt liefert; LIEFERFÄHIGKEIT\* stellt dar, welche Artikel ein Lieferant zu liefern **in der Lage** ist („Artikelkatalog“)

\*dreistellig      \*zweistellig

## Abschließende Bemerkungen zu Beziehungstypen

- Die **Stelligkeit** eines Beziehungstyps R (Zahl **der an R beteiligten Entitytypen**) wird auch als **Grad** des Beziehungstyps bezeichnet:

$$R = E_1 \times E_2 \times \dots \times E_k \Rightarrow \text{grad}(R) = k \geq 2$$

- Beziehungen können Attributwerte besitzen, **Beziehungstypen dementsprechend Attribute**

Beispiele:

indiv.

≡ für konkrete Liefer-  
beziehung vereinbart

muss  
Attribut dem  
Bez.typ  
zuordnen

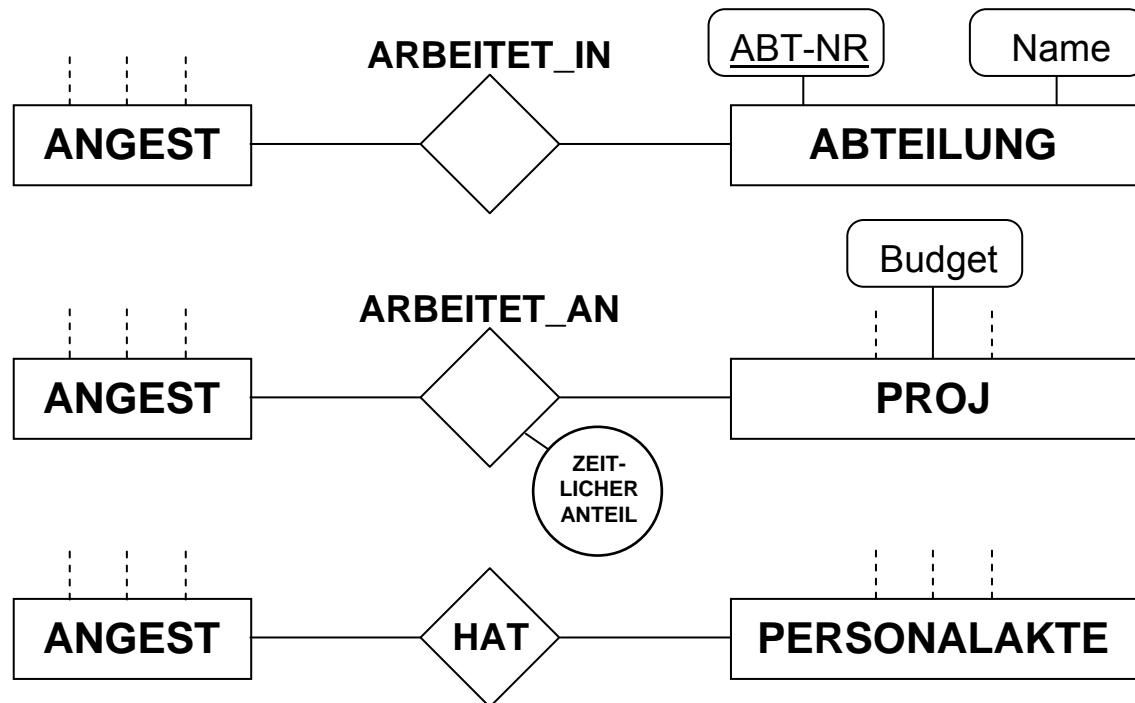
- Attribute MENGE, PREIS bei Beziehungstyp LIEFERT
- Attribut ZEITLICHER\_ANTEIL bei Beziehungstyp ARBEITET\_AN
- Attribut SEIT bei Beziehungstyp ARBEITET\_IN
- etc. etc.

## E/R-Diagramme



Als Entitytypnamen werden **Substantive** benutzt, als Beziehungstypnamen **Verben** (von links nach rechts bzw. oben nach unten gelesen)

### Beispiele:



In einem E/R-Diagramm müssen Entitytyp- und Beziehungstypnamen eindeutig sein

Hinweis: Zu den Beziehungstypnamen notiert man manchmal noch eine Art „Aliasnamen“ fürs Lesen von rechts nach links bzw. unten nach oben  
→ sorgt für größere Klarheit / besseres Verständnis der Semantik

<u>Beispiele:</u>	<b>Beziehungstypname</b>	<b>Aliasname</b>
	ARBEITET_IN	BESTEHT_AUS
	ARBEITET_AN	UMFASST
	HAT	GEHOERT_ZU

## Integritätsbedingungen, Komplexität/Kardinalität von Beziehungstypen

Was es zu betrachten gilt / zu unterscheiden:

- **Zwingende Beziehung (mandatory relationship)**

$R = E \times E'$ , betrachte  $e_1 \in E \Rightarrow$  es gibt stets mindestens ein  $e_2 \in E'$ , mit dem  $e_1$  innerhalb von  $R$  in Beziehung steht ( $r=(e_1, e_2)$ )

Beispiel: Angestellter gehört stets einer Abteilung an, steht also mit einer Abteilung in Beziehung ( $\rightarrow$  Beziehungstyp ARBEITET\_IN)

- **Optionale Beziehung (optional relationship)**

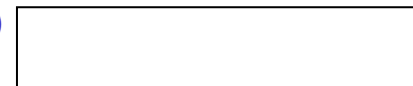
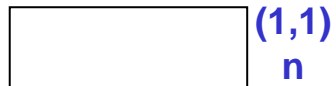
Entity  $e_1$  darf in  $E$  existieren, ohne dass es über  $R$  zu einem Entity  $e_2 \in E'$  in Beziehung steht

**Komplexität/Kardinalität** von Beziehungstypen:

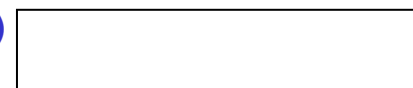
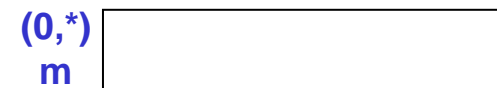
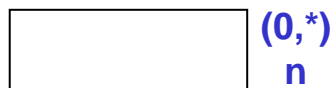
drückt aus, zu wie vielen Entities  $e_j \in E'$  ein Entity  $e_i \in E$  **minimal** in Beziehung stehen **muss** bzw. **maximal** in Beziehung stehen **darf**

$\Rightarrow$  (min,max)-Notation,  $\min \leq \max$

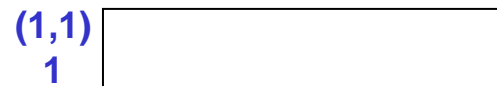
1:n  
hier.



many-to-many

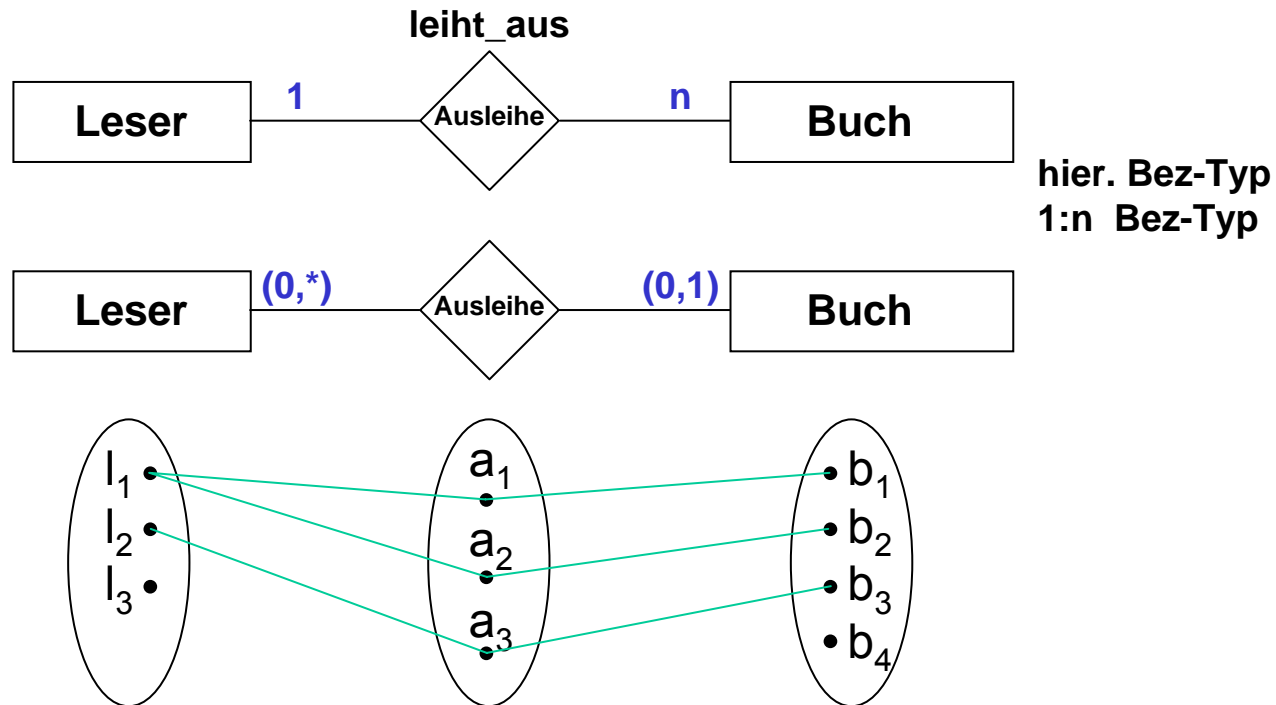


1:1



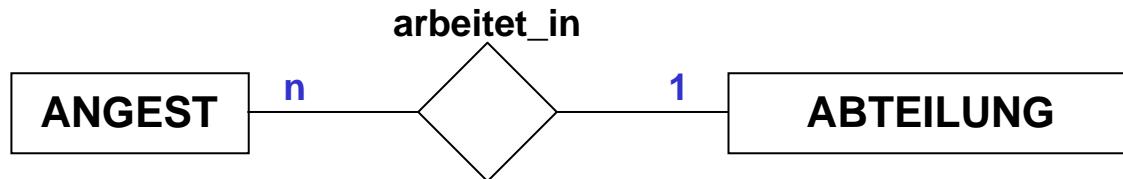
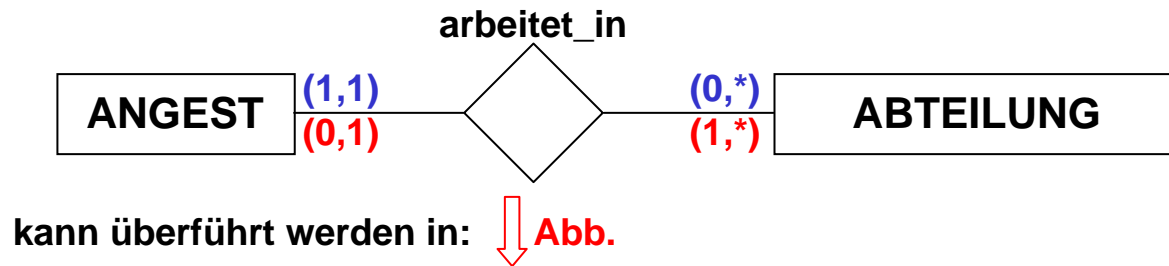
## Einige weitere Bemerkungen zur Komplexität / Kardinalität von Beziehungstypen

- Bisher betrachtet: (min,max)-Notation (vgl. Folie 73/74) für **k-stellige Beziehungstypen**,  $k \geq 2$
- für **2-stellige Beziehungstypen** (auch **binäre** Beziehungstypen genannt) wird oft vereinfachte Notation als Abkürzung vorgeschlagen, als 1:n-Notation bezeichnet. Beispiel:





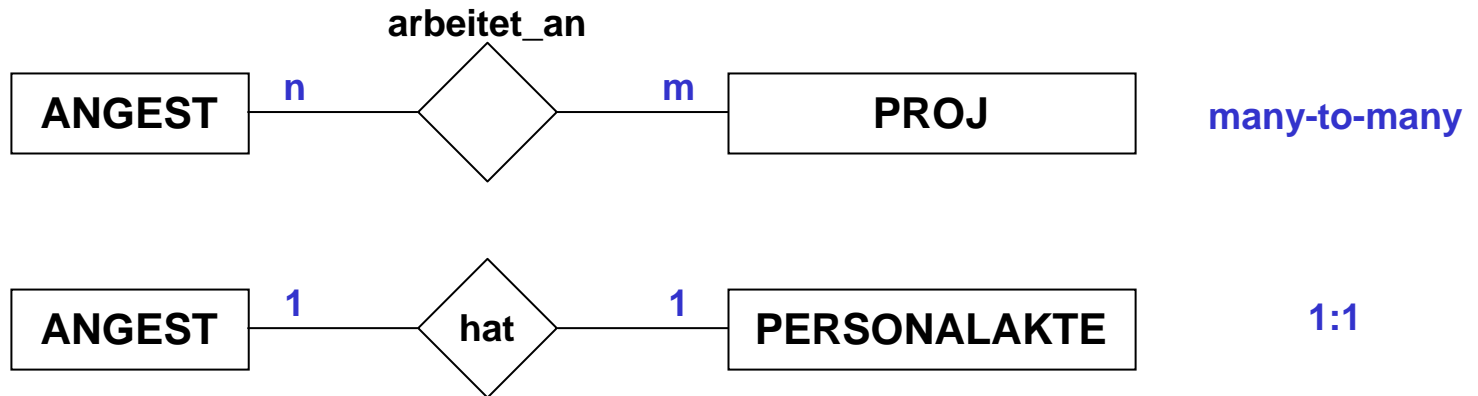
- Verbal:
- Jeder Leser kann beliebig viele ( $\geq 0$ ) Bücher ausgeliehen haben
  - Jedes Buch kann entweder an einen oder an keinen Leser ausgeliehen sein



1:n-Notation ist semantisch weniger genau (ärmer) als die (min,max)-Notation

Aber: Jeder Angestellter **muss** einer Abteilung angehören  $\Rightarrow$  Durch die 1:n-Notation geht diese Information (muss-Beziehung) verloren; es ist nicht mehr zu erkennen, ob das „n“ aus  $(0,1)$  oder aus  $(1,1)$  heraus entstanden ist, Rücküberführung somit nicht eindeutig möglich!!

Die 1:n-Notation umfasst den Spezialfall der **1:1-Beziehung** und kann zur **n:m-Beziehung** erweitert werden



Auch hier gilt wieder, dass bei Herleitung aus der (min,max)-Notation ein **Informationsverlust** auftritt, es ist konkret in obigen Beispielen nicht mehr zu erkennen,

- ob Angestellte erlaubt sind, die an keinem Projekt mitarbeiten
- ob „leere Projekte“ erlaubt sind
- ob Angestellte ohne Personalakte bzw. Personalakten ohne Angestellte erlaubt sind

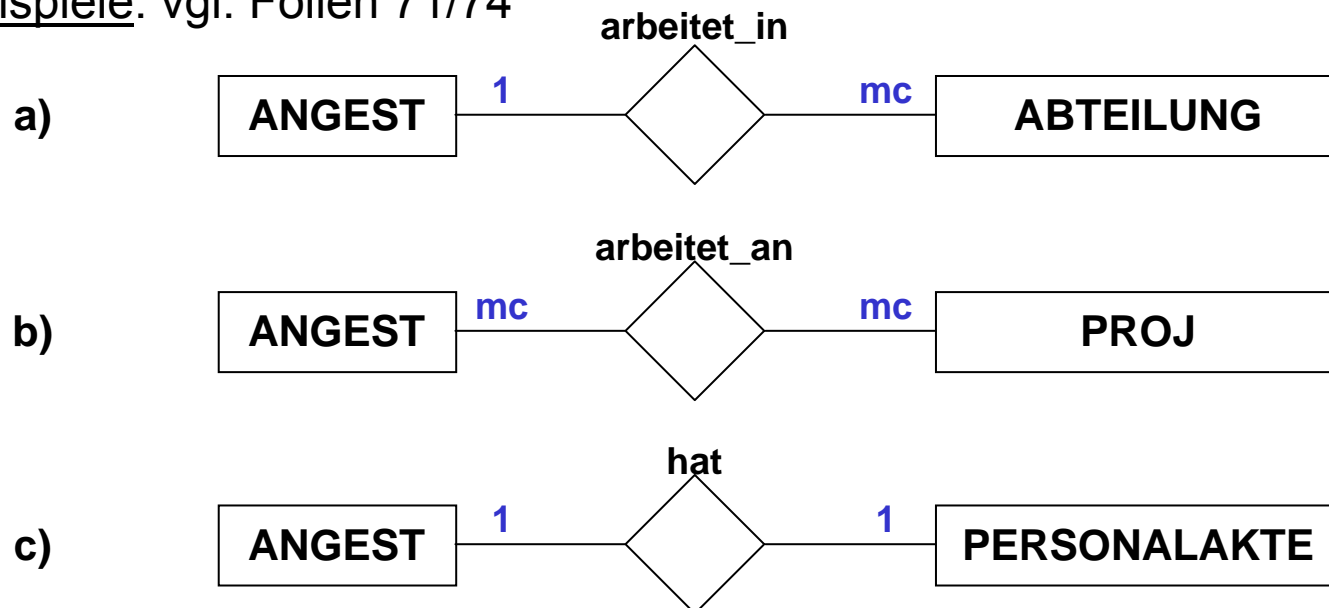
## Die (1,c,m)-Notation für Kardinalitäten / Komplexitäten

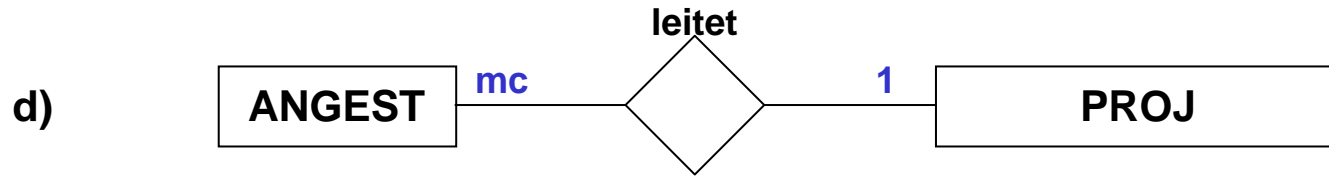
**Beschränkung der (min,max)-Notation auf vier (wesentliche) Fälle:**

(min,max)-Notation	(1,c,m)-Notation
(1,1)	1
(0,1)	c
(1,*)	m
(0,*)	mc

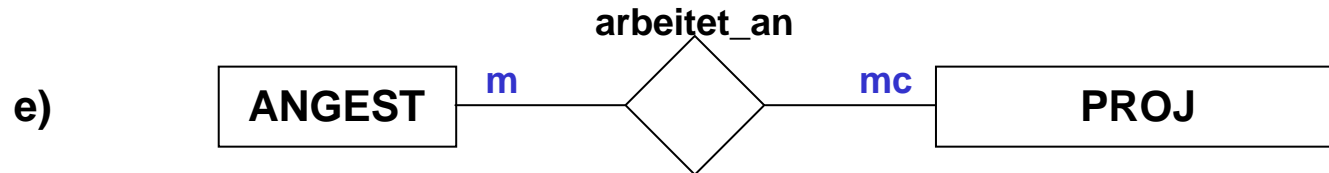
„c“ steht für „**choice**“  
(oder „**conditional**“)  
„m“ steht für „**multiple**“

Beispiele: vgl. Folien 71/74





Ein Angestellter kann beliebig viele ( $\geq 0$ ) Projekte leiten; jedes Projekt hat genau einen Projektleiter.



Im Unterschied zu b: Jeder Angestellte muss in **mindestens einem** Projekt mitarbeiten

### Bemerkungen:

(min,max)-Notation erlaubt **genauere Angaben** als (1,c,m)-Notation: Die Tatsache, dass ein Angestellter stets in **2 oder 3** Projekten mitarbeiten **muss**, ist in (min,max)-Notation als **(2,3)** genau darstellbar, in (1,c,m)-Notation dagegen nur als **m**  $\Rightarrow$  **Informationsverlust bei Überführung von (min,max)- nach (1,c,m)-Notation**

## Abschließende Hinweise zum Thema Kardinalitäten / Komplexitäten

- In der **Literatur** findet man die **unterschiedlichsten Darstellungsformen/Spezifikationsformen** für Kardinalitäten / Komplexitäten, auch über das hinaus, was wir hier angesprochen haben in den **3 genannten Varianten** ☹
- Man muss genau hinschauen, welche Semantik der jeweilige Autor zugrunde legt!
- Empfehlung: (Bei selbst zu erstellenden E/R-Modellen)  
**(min,max)-Notation** verwenden wegen großer Ausdrucksmächtigkeit oder **1:n-Notation**, falls genaue Kardinalitätsangaben (noch) nicht vorliegen

## Sog. schwache Entitytypen (auch abhängige Entitytypen genannt)

- Engl.: **weak entity** (bzw. weak entity type)

- Situation:

Instanzen eines bestimmten Entitytyps, also die zugehörigen Entities, können nur **existieren in Abhängigkeit von anderen Entities**

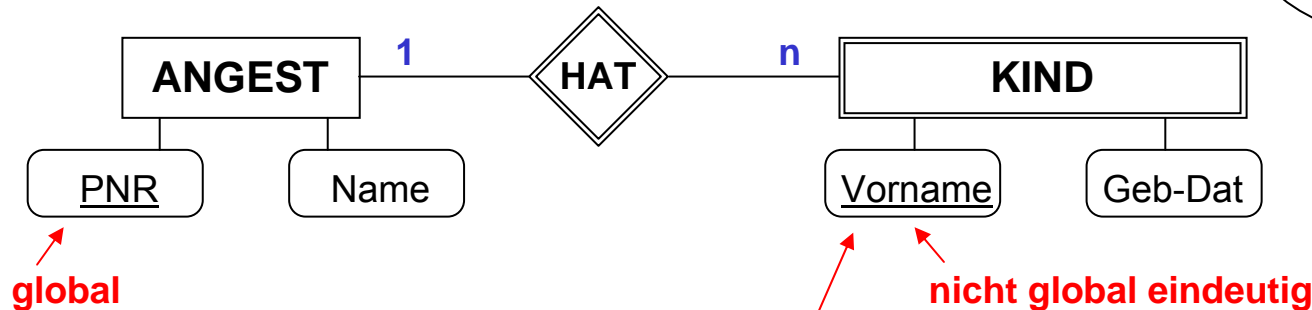
(zu denen sie in Beziehung stehen)

- Oder, mit anderen Worten: Zur **eindeutigen Identifizierung** einer Instanz eines schwachen Entitytyps muss zusätzlich die **Beziehung** mit herangezogen werden, d.h. erst **bestimmte Attributwerte des Entity plus Beziehung** identifizieren eindeutig!
- Bsp.: **Angestellte haben Kinder**, über die im Datenmodell / in der Datenbank ebenfalls Buch geführt werden soll. Die Kinder haben – im Datenmodell / in der Datenbank – aber nur **zusammen mit dem zugehörigen Elternteil** (Angest.) „Existenzberechtigung“, außerdem werden wir keinen eigenen global identifizierende Schlüssel à la „KPNR“ für Kinder einführen

⇒ KIND ist schwacher Entitytyp

WET  
nur in Zshg.  
mit hier.  
Bez.typ (1:n)

Darstellung im E/R-Diagramm:



- d.h. **doppelte Linien** beim Entitytyp und beim Beziehungstyp kennzeichnen schwache Entitytypen
- das (**nicht global**, unvollständig identifizierende) KIND-Attribut Vorname wird auch als **partieller Schlüssel** bezeichnet;  
**vollständig identifizierend** ist erst der partielle Schlüssel **zusammen** mit der Beziehung zum Angestellten (zugehöriges Elternteil)

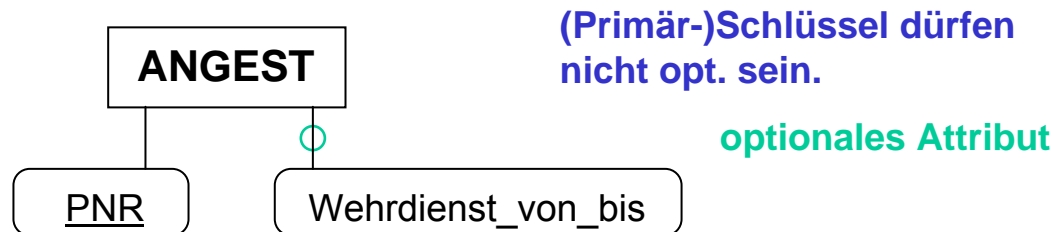
**EXISTENZABHÄNGIGKEITEN können über WET dargestellt werden**

## Zusätzliche Erweiterungen des E/R-Modells sowie der E/R-Diagrammdarstellung

### 1. Erweiterungen bei Attributen

#### a) Optionale Attribute

- **[Optionale Beziehungen** (Kann-Beziehung) (Folie 73): Entity  $e_1 \in E$  **darf** in Beziehung  $r$  stehen zu Entity  $e_2 \in E'$ , **muss** aber nicht;  
Bsp.: Mitarbeit von Angestellten in Projekten]
- **Optionales Attribut:**  
Attribut existiert für Entitytyp bzw. Beziehungstyp, **muss aber nicht für jedes Entity / jede Beziehung einen definierten Wert annehmen**  
(m.a.W.: als Wert existieren)
- **Graphische Notation (Bsp.):**



(Für eine **weibliche** Angestellte existiert die Wehrdienstangabe nicht.  
(war mal früher so in Deutschland..) Entsprechend bei Beziehungstypen



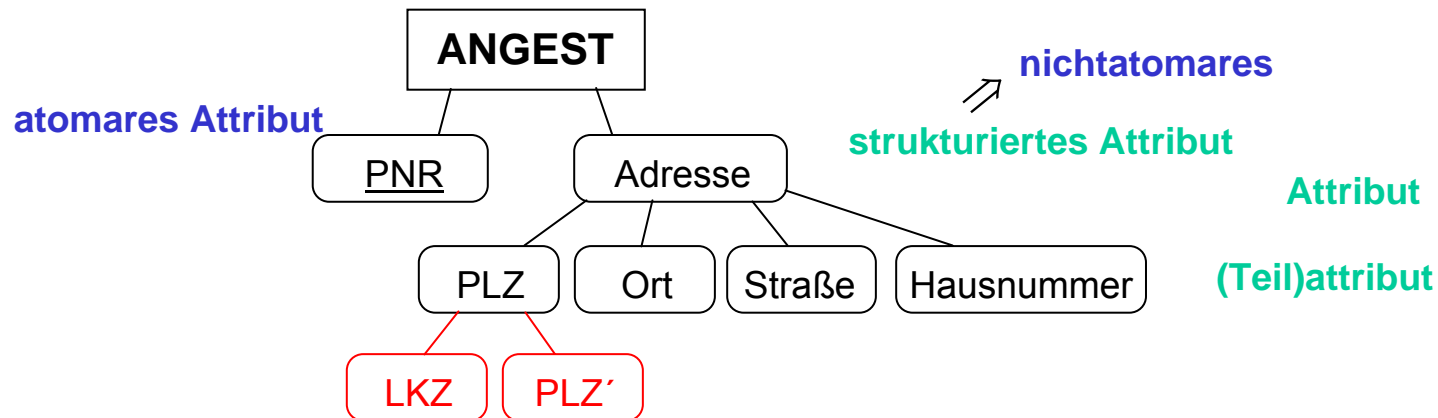
- Vorsicht: Die **Semantik** bei Optionalität sollte jeweils genau definiert/überlegt werden, meist werden **mehrere Bedeutungen** leichtfertig „in einen Topf geworfen“. Am Beispiel:
  - Für weibliche Angestellte **existierte** die Wehrdienstangabe (in Deutschland) nicht.
  - Für männliche Angestellte **existiert** sie nicht, wenn sie keinen Wehrdienst geleistet haben ...
  - ... oder wenn sie ihn zwar geleistet haben, der Von-bis-Zeitraum der Datenbank aber aktuell **unbekannt** ist („wurde bislang nicht eingegeben“)

⇒ die Tatsache, dass ein Attributwert aktuell undefiniert ist, kann verschiedene Ursachen haben (not known, does not exist ...).

Optionale Attribute sind primär für den Fall gedacht, wo **nicht** nur vorübergehend undefinierte Attributwerte zugelassen werden sollen, sondern wo **zulässige strukturelle Unterschiede zwischen den Entities eines Typs erlaubt sind** (~ CASE bei Typdefinition in höheren Programmiersprachen)

## b) Strukturierte Attribute

- **Strukturiertes Attribut** setzt sich aus anderen Attributen („einfacheren“ Attributen) zusammen  
(~ record- oder struct-Konstrukt in höheren Programmiersprachen)
- **Engl.: composite attribute**
- Die **Komponenten** eines strukturierten Attributs sind 1) **benannt** und 2) i.d.R. bezüglich ihrer Domänen (Wertebereiche) **inhomogen**
- **Graphische Notation (Bsp.):**



- Weitere **Schachtelung** möglich
- Auch als nicht atomares Attribut bezeichnet

### c) Mengenwertige Attribute

- **Mengenwertige Attribute: Attributwert = Wertemenge**  
(~ set-Konstrukt in höheren PS)
- **Engl.: multivalued attribute**
- Die **Elemente** der Wertemenge eines mengenwertigen Attributs sind  
1) **unbenannt** und 2) bezüglich ihrer Domäne (Wertebereichs) **homogen**
- **Graphische Notation (Bsp.):**



- Weitere **Schachtelung** möglich, auch in Verbindung mit **strukt. Attribut**  
(z.B.: Menge von Adressen)
- Auch als nichtatomares Attribut bezeichnet

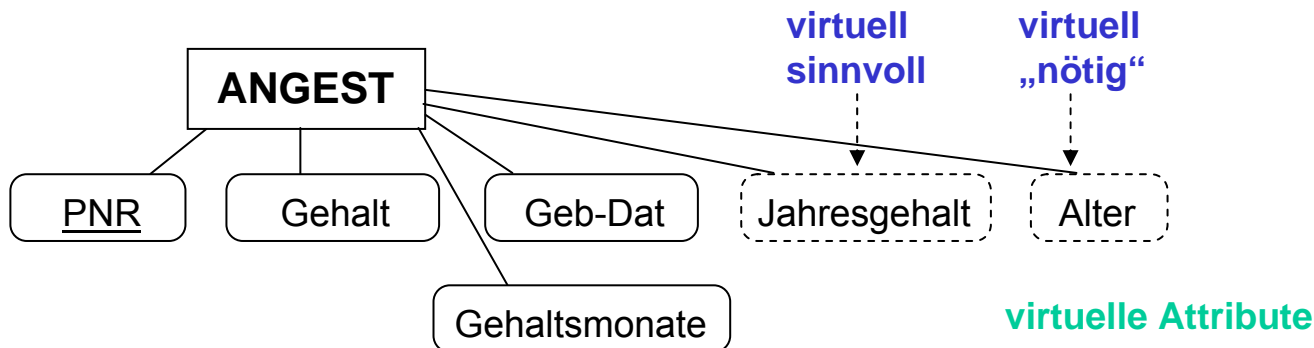
#### Hinweis zur Vorsicht:

Mit **strukturierten** und **mengenwertigen Attributen** sollte im E/R-Modell **wohlüberlegt** umgegangen werden, weil:

- a) „später“ **nicht 1:1 abbildbar** auf relationales Modell (Datenbankschema), da dieses in Reinkultur nur atomare Attribute kennt (Kap. 5, Abbildung trotzdem machbar, aber eben nicht 1:1)
- b) Entscheidung, ob etwas als (ggf. **schwacher**) **Entitytyp** modelliert wird oder als **nichtatomares Attribut**, nicht immer einfach (Bsp.: Kinder von Angestellten → schwacher Entitytyp oder mengenwertiges Attribut?) ⇒ Information nicht in Attributen „verstecken“, sondern (via Entitytyp/Beziehungstyp) **explizit darstellen** [oftmals betrachtet man bei E/R-Diagramm oberflächlich die Attribute gar nicht, sondern nur Entitytypen/Beziehungstypen → in nichtatomaren Attributen „versteckte“ Information bleibt Betrachter verborgen]
- c) (Nichtatomare) Attributwerte existieren nur, solange Entity existiert (natürlich!), d.h. können nicht eigenständig weiterleben → wie bei schwachem Entitytyp und dessen Instanzen. Berücksichtigen!

d) Virtuelle Attribute (auch  $\left\{ \begin{array}{l} \text{berechnete} \\ \text{abgeleitete} \end{array} \right\}$  Attribute genannt)

- **Nicht real gespeichert** („materialisiert“) als Wert, sondern durch Berechnungsvorschrift ermittelt
- Engl.: **derived attribute**
- **Graphische Notation (Bsp.):**



- Die **Berechnungsvorschrift** muss **mit angegeben** werden, z.B.
  - Jahresgehalt := Gehalt \* Gehaltsmonate
  - Alter := Aktuelles Datum – Geb-Dat
- Die **Berechnungsvorschrift** darf grundsätzlich **beliebig komplex** sein, sie muss sich vor allem nicht unbedingt nur auf Attributwerte des gleichen Entities beziehen (wie im obigen Beispiel zufällig gegeben),

Beispiele:

- Virtuelles Attribut **DurchschnGehaltAbt** im Entitytyp ABTEILUNG; Wert errechnet sich mittels Durchschnittsbildung über die **zugehörigen** ANGEST-Entities (also Entity Set eines anderen Entitytyps)
- Virtuelles Attribut **MaxGehalt** im Entitytyp ANGEST, Wert errechnet sich als Maximum der Gehälter aller Angestellten. In tabellarischer Darstellung:

ANGEST	<u>PNR</u>	Gehalt	Geb-Dat	...	MaxGehalt	virtuell
	4711	2000	15.5.62		3600	
	0815	3500	7.3.57		3600	
	1860	3300	29.2.76		3600	
	007	3500	2.12.55		3600	
	2483	1700	28.2.77		3600	
	1512	3600	11.5.70		3600	
	1618	2000	30.5.66		3600	

virtuelles Attribut

(Auf den ersten Blick vielleicht nicht sehr „sinnfällig“, gibt aber z.B. einfache Möglichkeit, zu bestimmen, wie weit das Gehalt eines Angestellten vom Maximalgehalt entfernt ist)

- (Berechnungsvorschrift darf allgemein Datenbankabfrage beinhalten)

erst später eigentlich relevant bei Datenbankabbildung

## Bemerkungen zu virtuellen Attributen (E/R)

- Wir betrachteten (zunächst, E/R) ausschließlich **Modellierungsebene**, noch nicht Umsetzung auf konkretes Datenbankmodell oder gar auf konkretes DBMS
- Kurzer **Ausblick** auf Realisierung in relationalem Datenbankmodell („Tabellenmodell“) später über **Views** = virtuelle Tabellen
- Über reale Tabellen (Basistabellen) werden virtuelle Tabellen (Views) „gestülpt“, d.h. man kann dort Dinge – Attribute, Tabellenspalten – „simulieren“ (berechnen), die es so real darunter (in der DB) nicht gibt
- Problem: Berechnungsaufwand bei jedem Zugriff auf „virtuelles“?!

### Möglichkeiten:

- a) Normale View** → beim Zugriff stets neue Berechnung, wie bei normalen DB-Abfragen
- b) Materialized View** („materialisierte virtuelle Tabelle“), d.h. es wird der Eindruck einer virtuellen Tabelle vermittelt, in Wahrheit ist sie aber vom DBMS **materialisiert**

Vorteil: - Performance-Gewinn beim Lesen

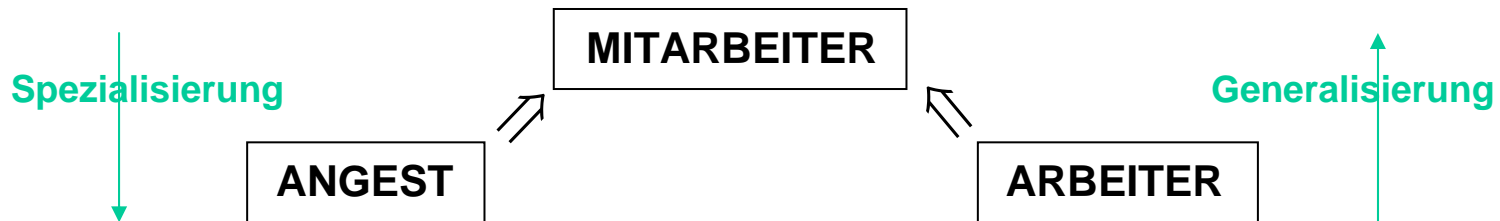
Nachteil: - Performance-Verlust beim Ändern (der Basistabellen) → komplette oder inkrementelle Anpassung der Mat. View nötig

## 2. Erweiterung um Spezialisierung und Generalisierung, IS-A-Beziehung, Sub- und Supertypen

- Hinweis: **Spezialisierung** und **Generalisierung** sind – wie auch schon optionale/strukturierte/mengenwertige/berechnete Attribute – Bestandteil sog. **erweiterter E/R-Modelle**

( { **Semantische Datenmodelle**  
( { „extended entity-relationship models“ } )

- Generalisierung ist die Bildung eines allgemeineren („generischen“) Entitytyps aus einer Anzahl „ähnlicher“ Entitytypen
- Beispiel:



Der umgekehrte Vorgang wird als Spezialisierung bezeichnet.

- Die entstehende Beziehung „ $\Rightarrow$ “ wird auch als **IS-A-Beziehung** bezeichnet: Jeder Angestellte **ist ein** Mitarbeiter, jeder Arbeiter **ist ein** Mitarbeiter.



- Außerdem sprechen wir in dem Zusammenhang auch von der Bildung von **Sub- und Supertypen**

In obigem Beispiel:

- MITARBEITER ist **Supertyp** von ANGEST und ARBEITER
- ANGEST und ARBEITER SIND **Subtypen** von MITARBEITER

### Warum Bildung von Sub- und Supertypen?

Realwelt-Entities sind einerseits oftmals zwar „ähnlich“, andererseits liegen doch so viele Unterschiede vor, dass sie nicht in einen einzigen Entitytyp abgebildet werden sollten.

Beispiel:

- Eine Firma beschäftigt Verkäufer, Schreibkräfte und Mechaniker **als Mitarbeiter**
- Für alle Mitarbeiter sollen jeweils Personalnummer, Name, Adresse, Abteilung und Gehalt in der Datenbank gespeichert werden
- Außerdem sollen bei den Verkäufern noch Umsatz-Soll und Umsatz-Ist, bei den Schreibkräften Anschläge pro Minute sowie Fremdsprachen und bei den Mechanikern die Zuständigkeit festgehalten werden.

Problem/Betrachtungen:

1. **Attributmengen** von Verkäufern, Schreibkräften und Mechanikern **nicht identisch**



↪ teilüberlappend

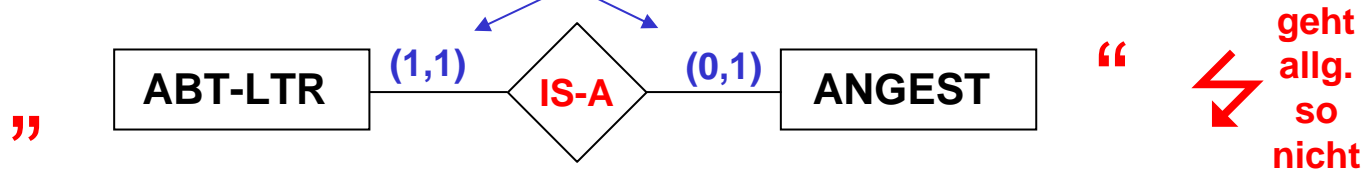
2. **Zusammenfassung** zu einem Entitytyp **problematisch**, da viele optionale Attribute und die Zusammenhänge zwischen diesen optionalen Attributen unklar (z.B.: Bei Schreibkräften und Mechanikern sind Umsatz-Soll **und** –Ist undefiniert; Mitarbeiter, bei denen **nur eines** dieser beiden Attribute undefiniert ist, darf es dagegen nicht geben)

3. **Trennung** in einzelne Entitytypen **problematisch**, da ja alle Firmenangehörigen Mitarbeiter sind mit (wie oben angegeben) etlichen gemeinsamen Eigenschaften

was fehlt:  
separate Integritätsregel

## Darstellung der IS-A-Beziehung im E/R-Diagramm

Kardinalitäten in (min,max)-Notation



Notationelle Erweiterung der einfachen E/R-Diagrammdarstellung

### Bemerkungen:

1. Subtyp hat normalerweise **zusätzliche Attribute** gegenüber Supertyp, etwa bei Abteilungsleiter: Ernennungsdatum, AnzahlUntergebene etc.  
→ Abteilungsleiter ist ein **spezieller** Angestellter
2. Die Generalisierung/Spezialisierung kann über **beliebig viele Stufen** erfolgen, etwa im Beispiel:  
Angestellte sind **spezielle** Mitarbeiter (denn es existieren daneben auch noch Arbeiter), Abteilungsleiter sind **spezielle** Angestellte



## Begriff der Vererbung

- Subtyp **erbt** die Attribute des Supertyps oder m.a.W.  
Supertyp **vererbt** seine Attribute an seine Subtypen
- Konsequenz: Die Attribute des Supertyps brauchen beim Subtyp nicht nochmals aufgeführt werden, sie sind dort vielmehr implizit vorhanden.
- Im Bsp.: Attribute PersNr, Name, Adresse, Gehalt ... werden vom Supertyp Mitarbeiter an die Subtypen Verkäufer, Schreibkraft und Mechaniker vererbt
- Was man nun noch vorsehen/betrachten könnte (wir tun aber nicht):  
Möglichkeit der Festlegung, **welche** Attribute vererbt/geerbt werden sollen und welche nicht sowie Möglichkeit, Attribute auf Subtypebene zu **überschreiben**

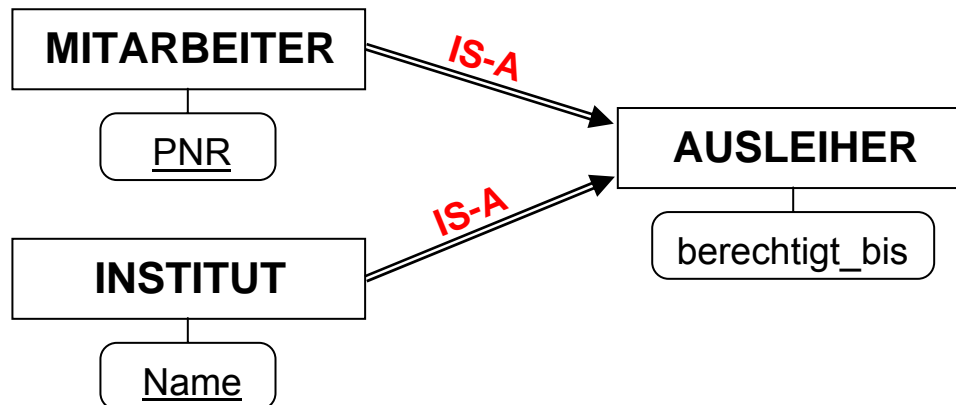
3. Subtyp und Supertyp haben meist **denselben Primärschlüssel**, dies muss aber nicht stets so sein.

Beispiele:



Der Primärschlüssel PNR (Personalnummer) gilt für Entitytyp MITARBEITER **und** die Subtypen ABT-LTR und ANGEST

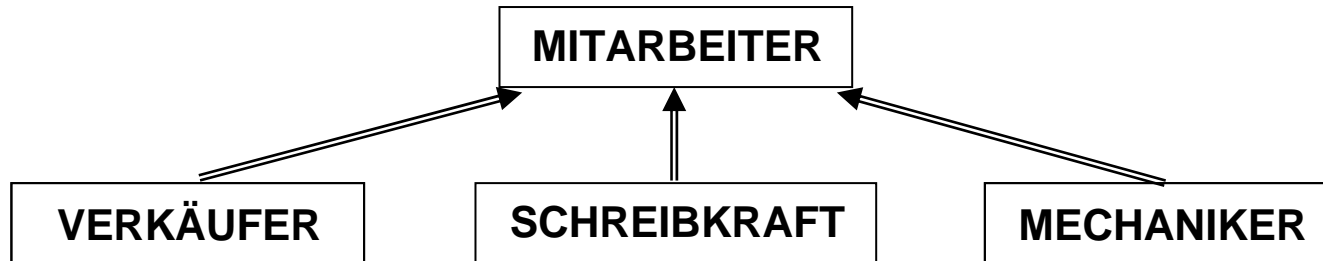
- b) Bibliotheksinformationssystem



Die Entitytypen Mitarbeiter und Institut haben jeweils **eigenen**, unterschiedlichen **Primärschlüssel**

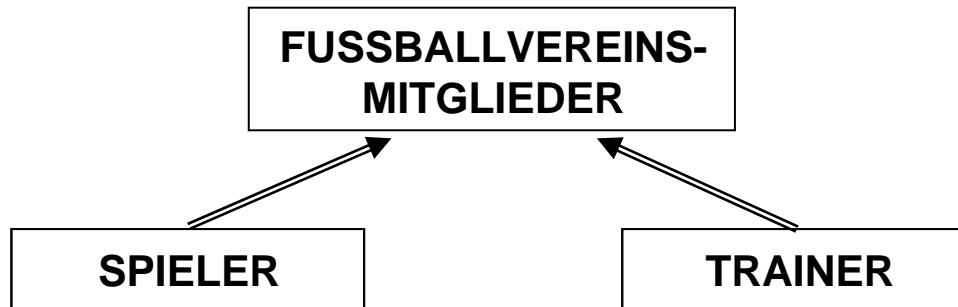
## Arten von Generalisierung / Spezialisierung

### 1. Disjunkte Subtypen



Jeder Mitarbeiter gehört (höchstens) einem dieser Subtypen an.

### 2. Nichtdisjunkte Subtypen (überlappende S.)



„Spielertrainer“ gehören beiden Subtypen an.

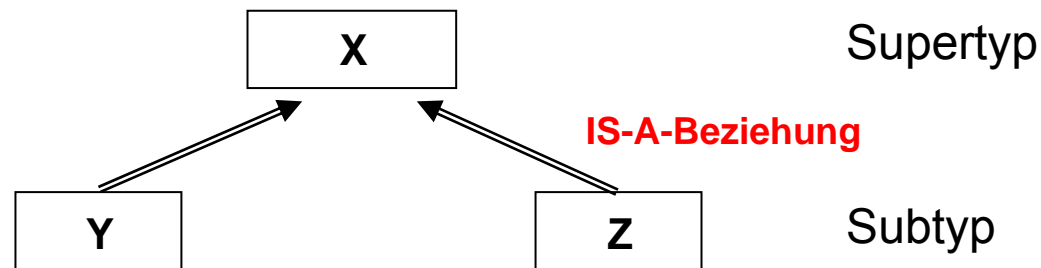
### 3. Vollständigkeit der Subtypbildung

Es gibt **keine** Fußballvereinsmitglieder, die nicht **entweder** Spieler **oder** Trainer sind (realistisch?)      fertium non datur

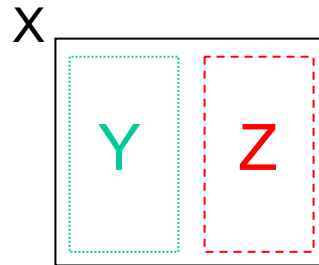
#### 4. Unvollständigkeit der Subtypbildung

Es gibt – neben Spielern und Trainern – noch passive Mitglieder, Funktionäre u.dgl., für die aber keine Subtypen existieren  
⇒ Spezialisierung ist nur **partiell**, nicht vollständig

Wichtig: Disjunkt/überlappend und vollständig/partiell sind **orthogonale Konzepte**, können also **beliebig** miteinander **kombiniert** werden



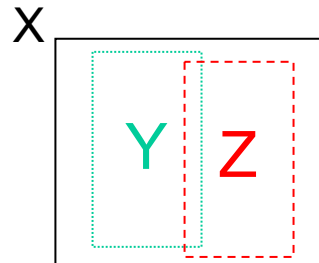
1. Spezialisierungstyp **pd**  
(**p**artiell & **d**isjunkt)



Beispiel:

X = Universitätsangehörige  
Y = Professoren  
Z = Studenten

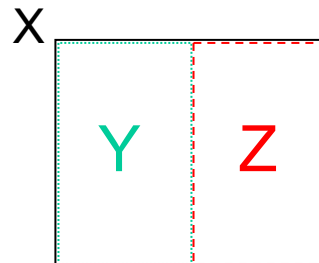
2. Spezialisierungstyp **pü**  
(**p**artiell & **ü**berlappend)



Beispiel:

X = Universitätsangehörige  
Y = Hiwis  
Z = Studenten

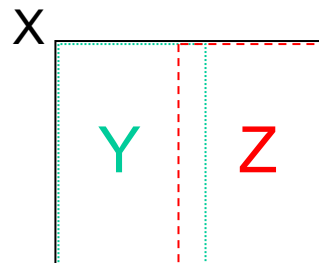
3. Spezialisierungstyp **vd**  
(**v**ollständig & **d**isjunkt)



Beispiel:

X = Personen  
Y = Männer  
Z = Frauen

4. Spezialisierungstyp **vü**  
(**v**ollständig & **ü**berlappend)



Beispiel:

X = Bauteile  
Y = Eigenteile  
Z = Fremdteile

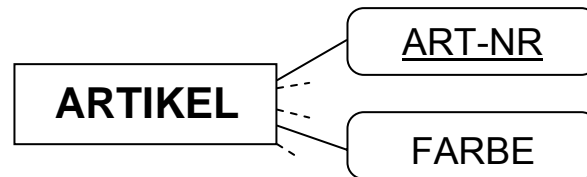


## Darstellung von Attributen und von Beziehungstypen als Entitytypen

### a) Attribut → Entitytyp

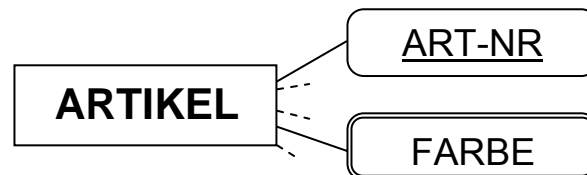
Beispiel: Entitytyp ARTIKEL : {ART-NR, ..., FARBE, ...}

mit dem Attribut FARBE und  $\text{dom}(\text{FARBE}) = \{\text{weiß, rot, gelb, grün, blau}\}$



### Wie lässt sich ein mehrfarbiger Artikel modellieren?

#### 1. FARBE in ein mengenwertiges Attribut überführen

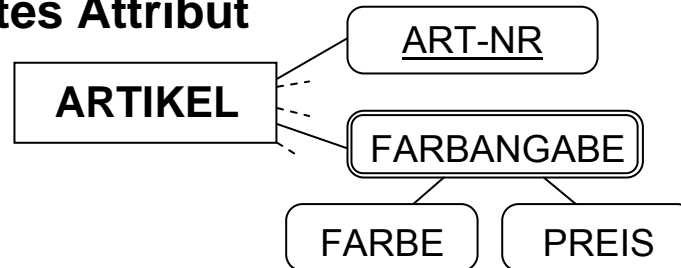


Probleme dabei:

- **Mengenwertiges** Attribut später nicht 1:1 abbildbar auf relationales Modell (Datenbankschema), da dort nur **atomare** Attribute erlaubt (vgl. Folie 87)
- Angenommen, zu jeder Farbe gäbe es noch einen (**farbspezifischen**) **Preis** („Artikel, die silbermetallisch sind, sind teurer als schwarze oder weiße Artikel“) ⇒

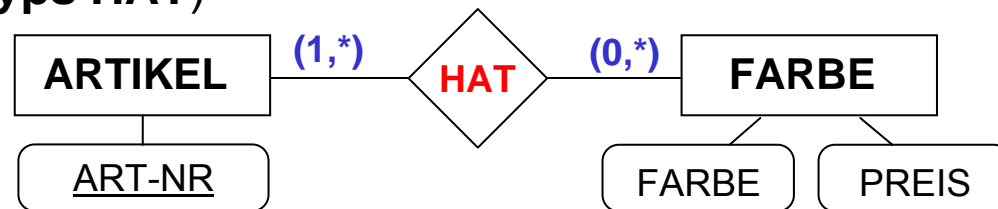
diesen Preis jeweils **zusätzlich** im Attribut mitführen?

→ **strukturiertes Attribut**



⇒ Preis, obwohl farbspezifisch und nicht artikelspezifisch gemeint, in jedem ARTIKEL-Entity vorhanden ⇒ Redundanz ⚡

2. Alternative deshalb: Einführung eines **Entitytyps FARBE** (sowie eines **Beziehungstyps HAT**)



Allgemeine Bemerkungen dazu:

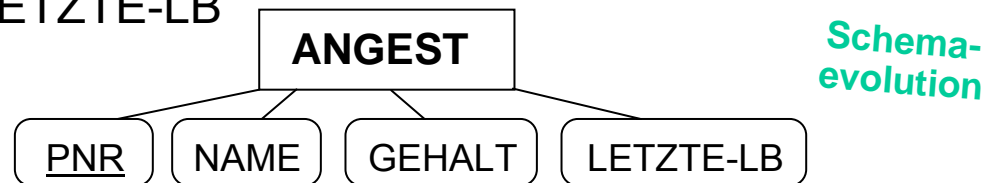
- Bei **mengenwertigen** Attributen – insbesondere wenn gleichzeitig **strukturiert** – muss vom Entwerfer eines E/R-Modells entschieden werden, ob statt dessen eigenständiger **Entitytyp** eingeführt wird (+ Beziehungstyp)

- Eigenständiger Entitytyp hat zudem Vorteil der **symmetrischen Darstellung** der Informationen:

ARTIKEL und FARBE stehen in Alternative 2 (s.o.) **gleichberechtigt** nebeneinander, während in Alternative 1 FARBE dem ARTIKEL untergeordnet ist

- Weiterer Aspekt: Redundanzfreie Darstellung kann auch bei atomaren Attributen Überführung in eigenständigen Entitytyp nahelegen.

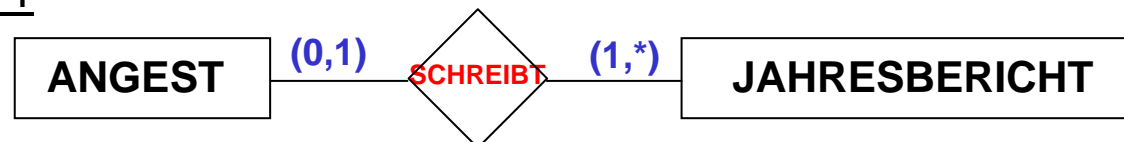
Beispiel: Es möge ein ANGEST-Entitytyp existieren mit Attributen PNR, NAME, GEHALT, LETZTE-LB



Dann möge hinzukommen Attribut JAHRESBERICHT

Anschließend wird festgelegt, dass **mehrere** Mitarbeiter, die im Team zusammenarbeiten, ihren Jahresbericht **gemeinsam** schreiben dürfen  
→ Redundanz ↗

Lösungsvariante 1





JAHRESBERICHT

## Lösungsvariante 2

Angenommen, es gibt bereits einen weiteren Entitytyp TEAM der Art:



Was passiert aber, wenn (1,1)-Kardinalitätsangabe durch (0,1) ersetzt wird  
→ eigenständiger Entitytyp wäre doch die bessere (und langfristig tragfähigere!) Lösung gewesen

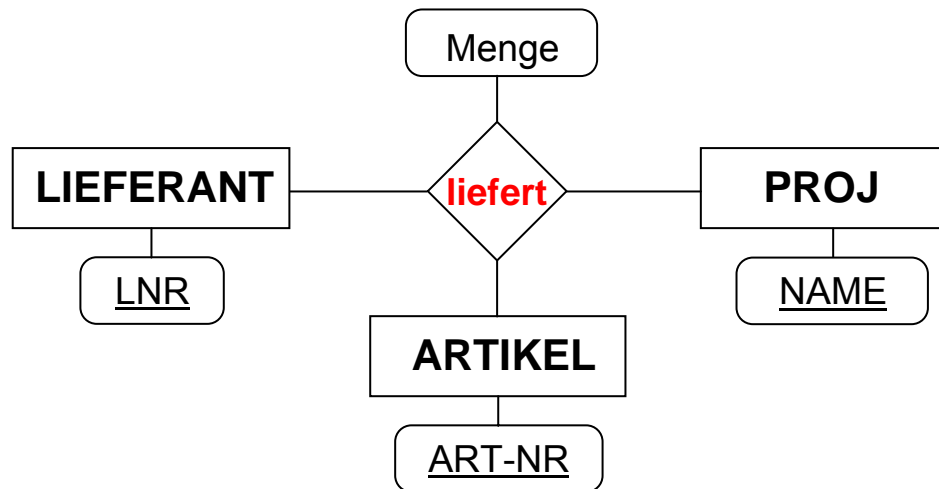
(N.B.:) Ein **E/R-Modell** nachträglich zu ändern, ist trivial; ein daraus bereits abgeleitetes **Datenbankmodell** (z.B. relationales Modell) nachträglich zu ändern, ist meist sehr schwierig\* ⇒ **gleich nach langfristig stabilem, flexiblem (E/R-)Modell streben** (vorauss. kommende Änderungen möglichst von vornherein berücksichtigen)

\* warum wohl?

JAHRESBERICHT

## b) Beziehungstyp → Entitytyp

Beispiel:



Aus Beziehungstyp liefert wird **Entitytyp**  
**LIEFERUNG : {LNR, ART-NR, NAME, Menge}**

Was ist Primärschlüssel in LIEFERUNG?

Prinzipielles Vorgehen:

Die **Primärschlüssel** der an den (k-stelligen) Beziehungstyp beteiligten **Entitytypen** werden – zusammen mit etwaigen Attributen des Beziehungstyps – zu **Attributen des neuen Entitytyps** gemacht

Gründe für eine solche Überführung:

- Kompakte (E/R-)Darstellung aller mit einer Lieferung zusammenhängenden Information in **einem** Entitytyp  $\Rightarrow$  die Informationen über Lieferungen „stehen **im Mittelpunkt**“
- [ • Größere Nähe der (E/R-)Modelldarstellung zur späteren **relationalen** Darstellung  $\Rightarrow$  im relationalen Modell (Kap. 5) werden alle Informationen in Tabellenform („Relationen“) dargestellt  
 $\Rightarrow$  viele Beziehungstypen werden dort zu eigenständigen Tabellen. ]



## Bekannte Stolpersteine bei E/R-Modellierungen

- Attribute bei **Beziehungstypen** vergessen / falsch verwendet
- Umgang mit k-stelligen ( $k \geq 3$ ) Beziehungstypen: Übersehen, falsch verwendet
- Kardinalitäten ((min, max) etc.) bei Beziehungstypen: Falsch den Kanten zugeordnet („vertauscht“), 1:n (n:m)-Notation richtige Kantenzuordnung
- Logische und physische Aspekte vermengt
- Semantische Möglichkeiten der E/R-Methodik wird oft nicht hinreichend genutzt („Porsche im 1. Gang“) → d.h. Modellierung versehentlich ohne optionale Attribute, virt. Attr. ..., Generalisierung/Spez.
- Merke: Es gibt nicht (i.d.R. nicht) DAS EINE, EINZIG RICHTIGE E/R-Diagramm für eine gegebene Problemstellung.  
ABER: Es gibt sehr wohl falsche, **unvollständige** E/R-Diag. und es gibt **gute** oder schlechte.

## Zusammenfassung / 'Hilites' zu Kap. 2 (E/R)

- E/R-Modellierungsmethodik zur Entwicklung eines logischen (konzeptuellen) Datenbankmodells **unabhängig** von konkreter DBMS-Technologie (welche Art von Daten**bank**modell, welches DBMS-Produkt ...)
  - **Semantisch reiches Modell**, d.h. relativ viel Semantik der betrachteten Anwendungswelt (Ausschnitt, Miniwelt) kann im Modell - E/R-Diagramm – dargestellt werden
  - Initial „erfunden“ von Chen Mitte 70er Jahre, später erweitert durch mehr semantische Möglichkeiten (Generalisierung/Spezialisierungen/...) → sog. **semantisches Datenmodell**
- Schnittstelle/Gesprächsbasis (Notation) zwischen Anwendern und Informatikern, „gemeinsame Sprache“
- **Reale Probleme:**
    - Für eine Miniwelt gibt es nie DAS eine richtige E/R-Diagramm → Modellierungsfreiheitsgrade! nicht unbedingt „Problem“
    - Komplexität (Größe) real existierender E/R-Diagramme
      - Ausschnittbildung
      - Abstraktionsebenen („geschachtelte E/R-Diagramme“)
      - Modifizierte E/R-Methodiken, etwa SER (Strukturiertes E/R), z.B. um Leserichtung in E/R-Diagrammen einzuführen