Datenbanken als Basis betrieblicher Anwendungssysteme

2017

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
AUFSTIEG DURCH BILDUNG
OFFENE HOCHSCHULEN

Prof. Dr.-Ing. Manfred Beham

Teil 1: Konzeption und Gestaltung von Datenbanken

- Konzepte und Notation der objektorientierten Analyse -

(Basiskonzepte)

1. Objektorientierte Basiskonzepte

- OO Basiskonzepte:
  - Objekt
  - Klasse
  - Attribut
  - (Operation)
  - Klassendiagramm

Literatur

- Heide Balzert
  Lehrbuch der Objektmodellierung
  Spektrum Akademischer Verlag, 1999
- Bernd Oestereich
  Objektorientierte Softwareentwicklung
  Oldenburg, 1998
- Gabriele Bannert, Martin Weitzel
  Objektorientierter Softwareentwurf mit UML
  Addison-Wesley, 1999
- Dietmar Steinpichler und Horst Kargl
  Projektabwicklung mit UML und Enterprise Architect
  SparxSystems Software 2012

Anwendungssysteme: Business-Software

- Abgrenzung zu Systemsoftware / Middleware
  - Betriebssystem
  - Programmentwicklung (Compiler, Testwerkzeuge)
  - Netzwerktechnik / Infrastruktur
  - Datensicherung

- Installiert auf:
  - Arbeitsplatzrechner (Desktop-Anwendung)
  - Server
  - Mobiles Endgerät (App)
  - Webanwendung (Client-Server-Architektur)

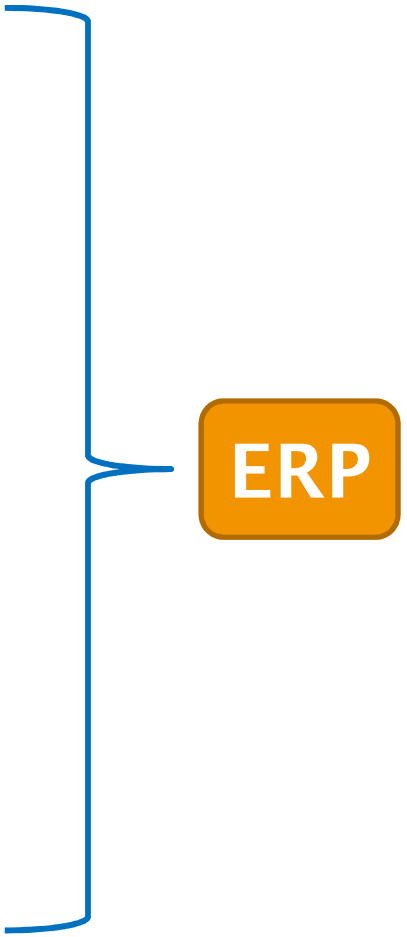
Anwendungssysteme: Unterstützung der Administration von Unternehmen und Behörden

- Ausführung der Systeme:
  - Standardsoftware
  - Individuallösungen
  - Branchenlösungen
  - (mobile Apps)

Arten und Funktionen betrieblicher Anwendersoftware

Eine Auswahl:

- Customer-Relationship-Management (CRM)
- E-Business
- Fakturierung
- Finanzbuchhaltung
- Operations Research (OR)
- Personalinformations-/-managementsystem (HRIS/HRMS)
- Produktionsplanungs- und Steuerungssystem (PPS)
- Produktdatenmanagementsysteme (PDM)
- Projektmanagementsoftware
- Warenwirtschaftssysteme
- Workflow-Management
- Management/Executive Informationssysteme (MIS/EIS)
- Datawarehouse
- Informationsmanagement
- Qualitätssicherungssysteme (QM)
- Klassische Bürodienste (Kommunikation, Kalender, …)



Definition der SW-Technik

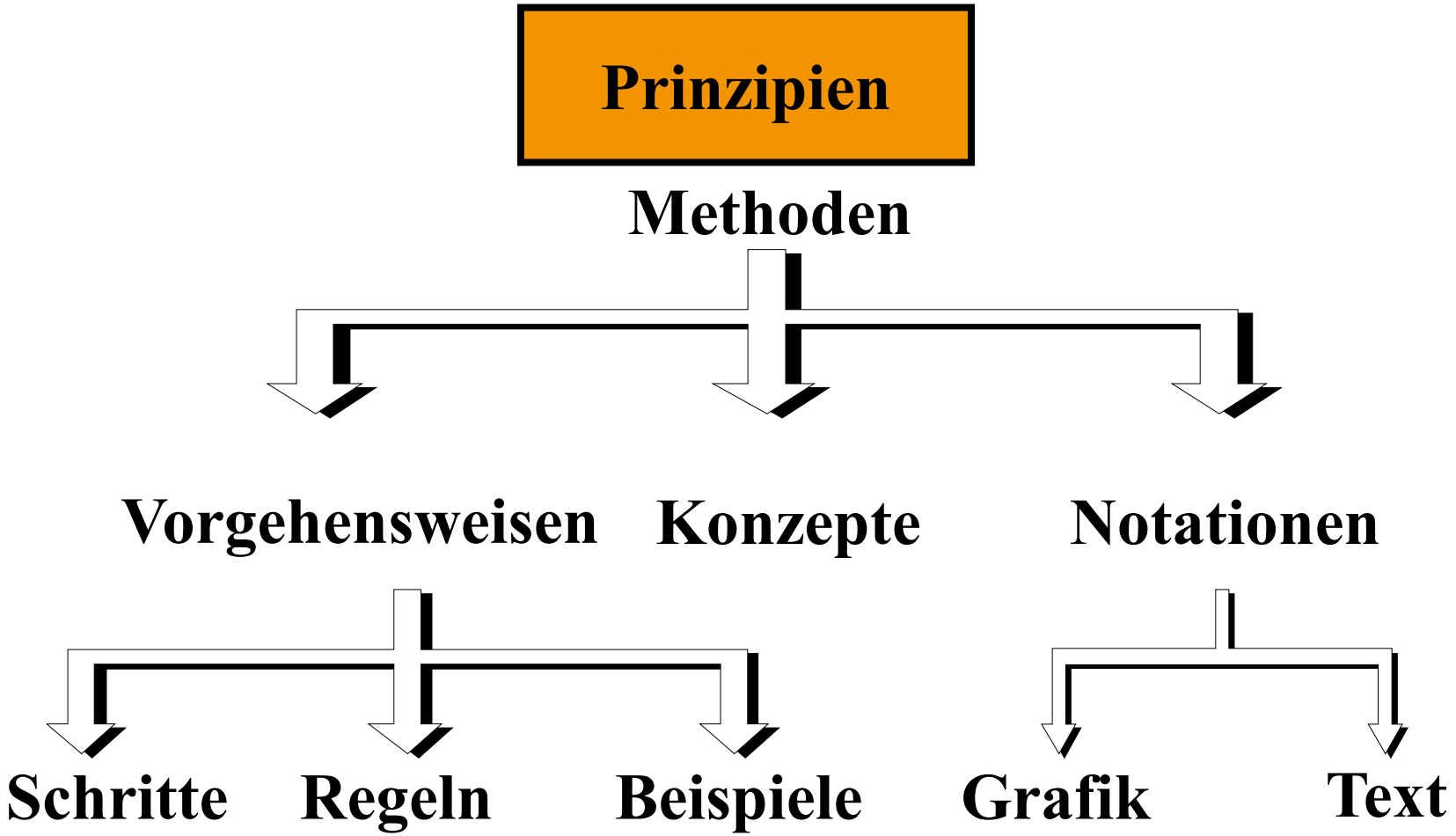
- **“The practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them.”, vgl. B.W. Boehm: Software Engineering, 1976**

- **“Das ingenieurmäßige Entwerfen, Herstellen und Implementieren von Software sowie die ingenieur-wissenschaftliche Disziplin, die sich mit Methoden und Verfahren zur Lösung der damit verbundenen Problemstellungen befasst”, vgl. Brockhaus-Enzyklopädie**

Herausforderungen in der SW-Entwicklung

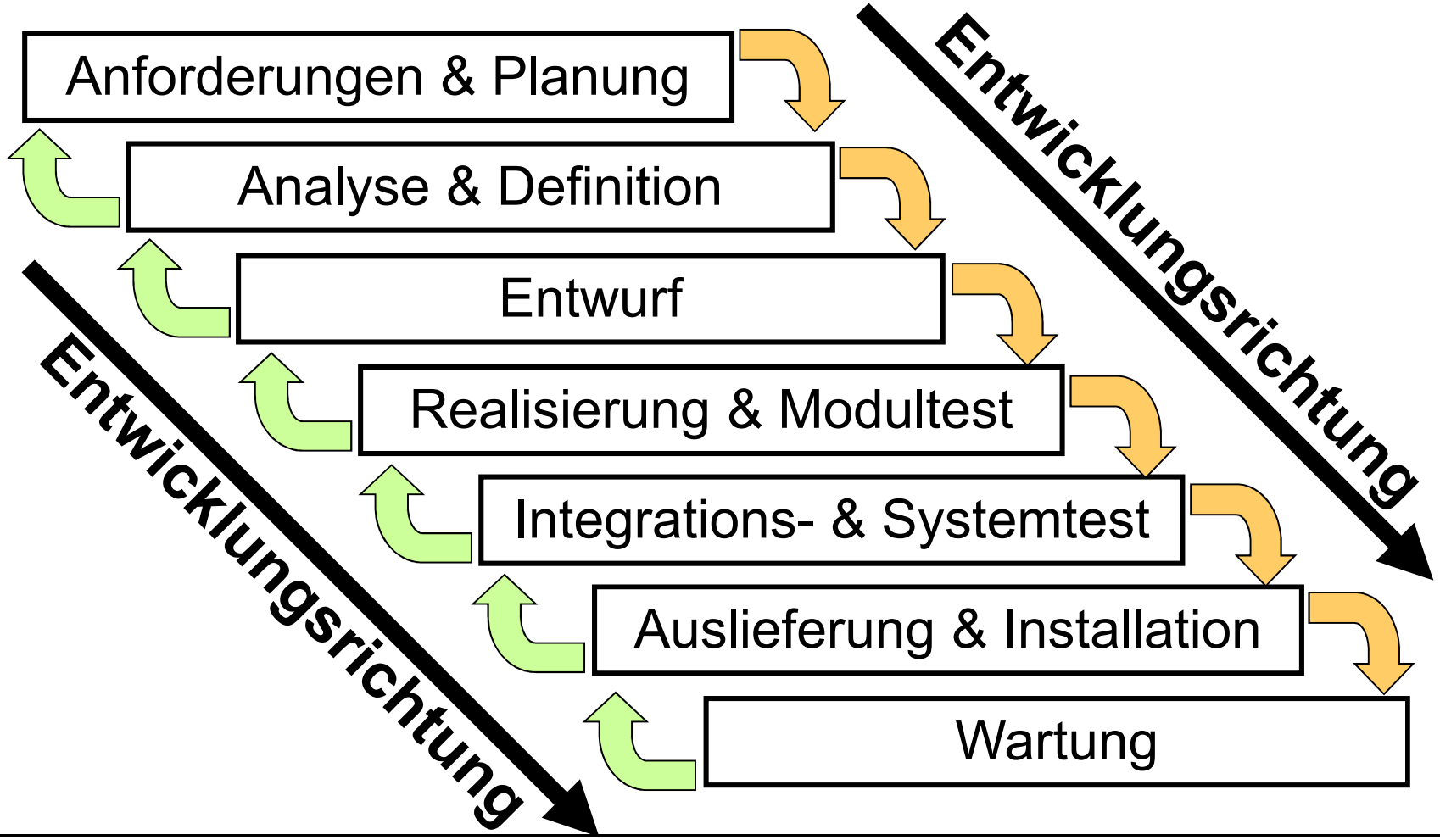
- Steigende Komplexität
- Bedeutungszunahme der Software
- Wachsende Qualitätsansprüche
- Softwarewartung und Softwareanpassung
- Leistungsstarke Entwicklungsteams
- Kürzere Entwicklungszeiten / -zyklen
- Heterogene, verteilte Systeme
- Fachübergreifende fragestellungen

Methodenkomponenten



vgl. H. Balzert: Lehrbuch der Software-Technik, 2000

Phasen der Softwareentwicklung



Planungsphase

- **Durchführbarkeits- /Machbarkeitsstudien**
  - ✓ **Lastenheft (grobes Pflichtenheft)**
  - ✓ **Glossar**
  - ✓ **Projektkalkulation**
  - ✓ **Projektplan**

Definitions- / Analysephase

- **Anforderungsanalyse und Verabschiedung der Anforderungen.**
  - ✓**Erweitertes Glossar**
  - ✓**Pflichtenheft**
  - ✓**Prototyp oder Pilotsystem**
  - ✓**Erste Version des Benutzerhandbuches**

Designphase

- **Rand- und Umgebungsbedingungen definieren und deren Einflussgrößen konzipieren**
  - ✓**Softwarearchitektur**
  - ✓**Spezifikation der Systemkomponenten**

Implementierungsphase

- **Programmieren im Kleinen**
  - **Datenstrukturen und Algorithmen**
  - **Dokumentation der Implementierung**
  - **Performancedokumentation**
  - **Testen und Verifizieren**
  - ✓ **Quell- und Objektcode**
  - ✓ **Test- und Prüfprotokolle**

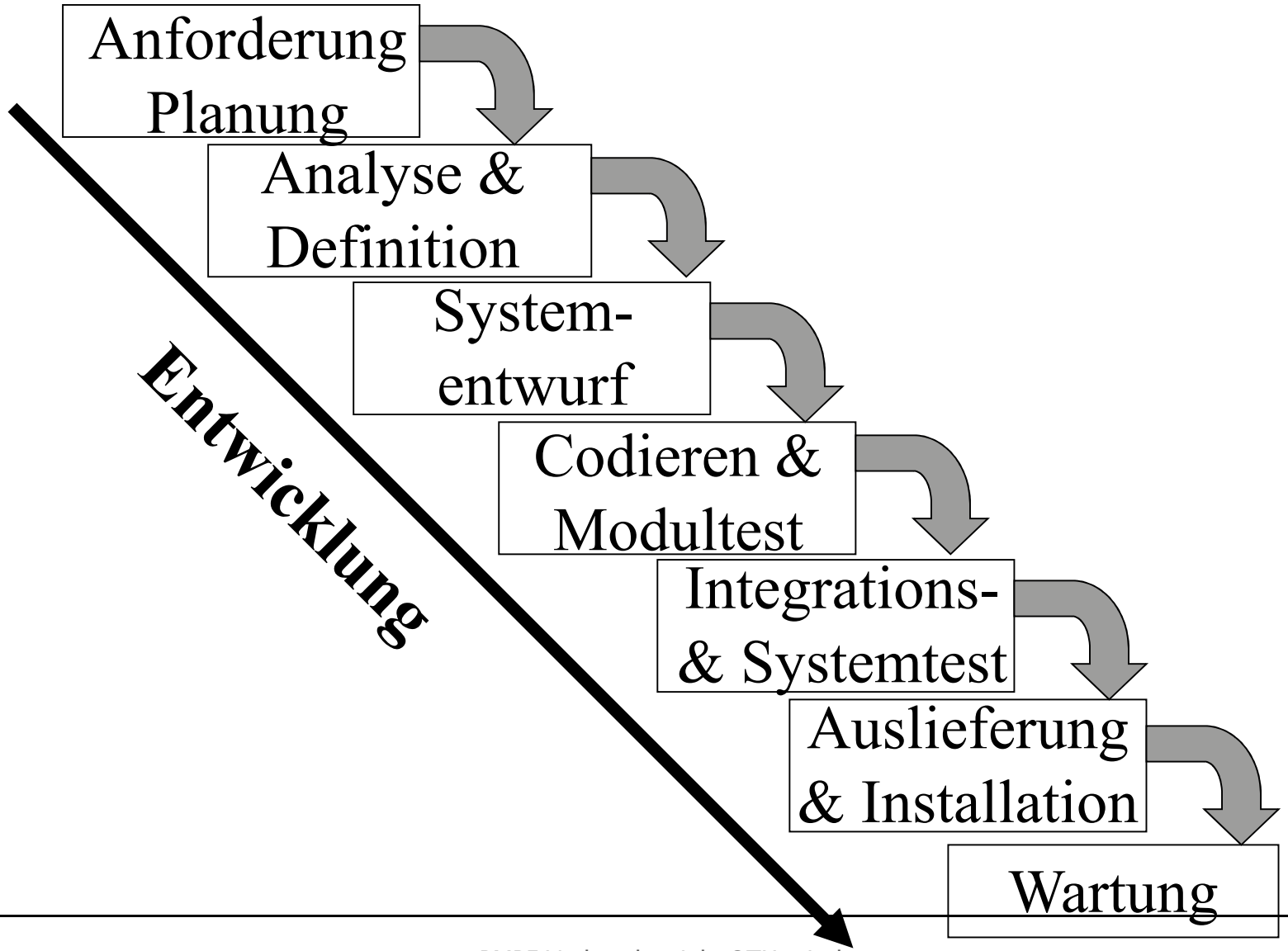
Abnahmephase

- **Übergabe der SW und Abnahmetest**
- **Installation und Schulung**
- **Inbetriebnahme**
  - ✓ **Gesamtdokumentation**
  - ✓ **Abnahmeprotokoll**
  - ✓ **Einführungsprotokoll**

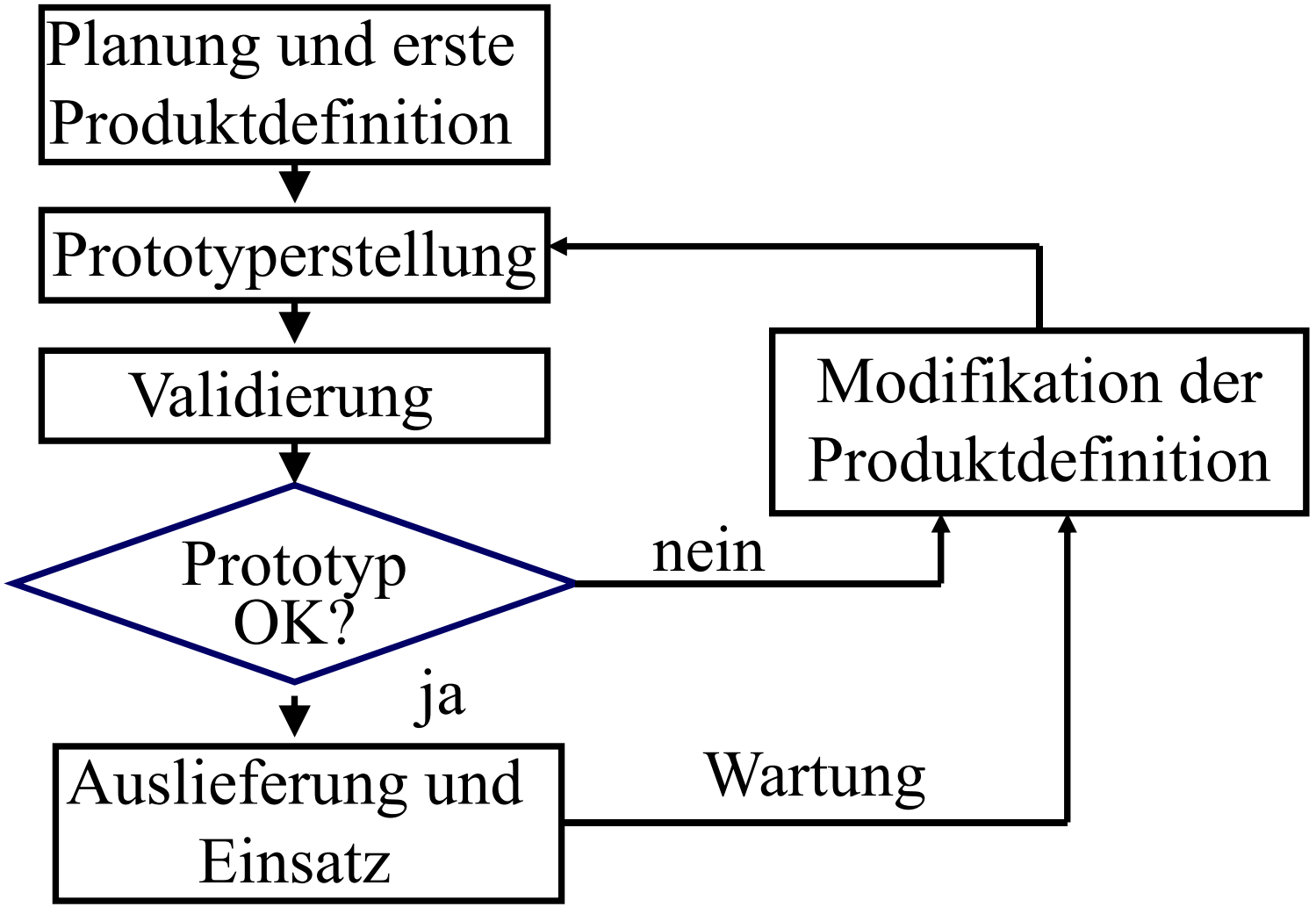
Wartungsphase

- **Stabilisierung / Korrektur**
- **Optimierung / Leistungsverbesserung**
- **Anpassung / Änderung**
- **Erweiterung**

Wasserfallmodell



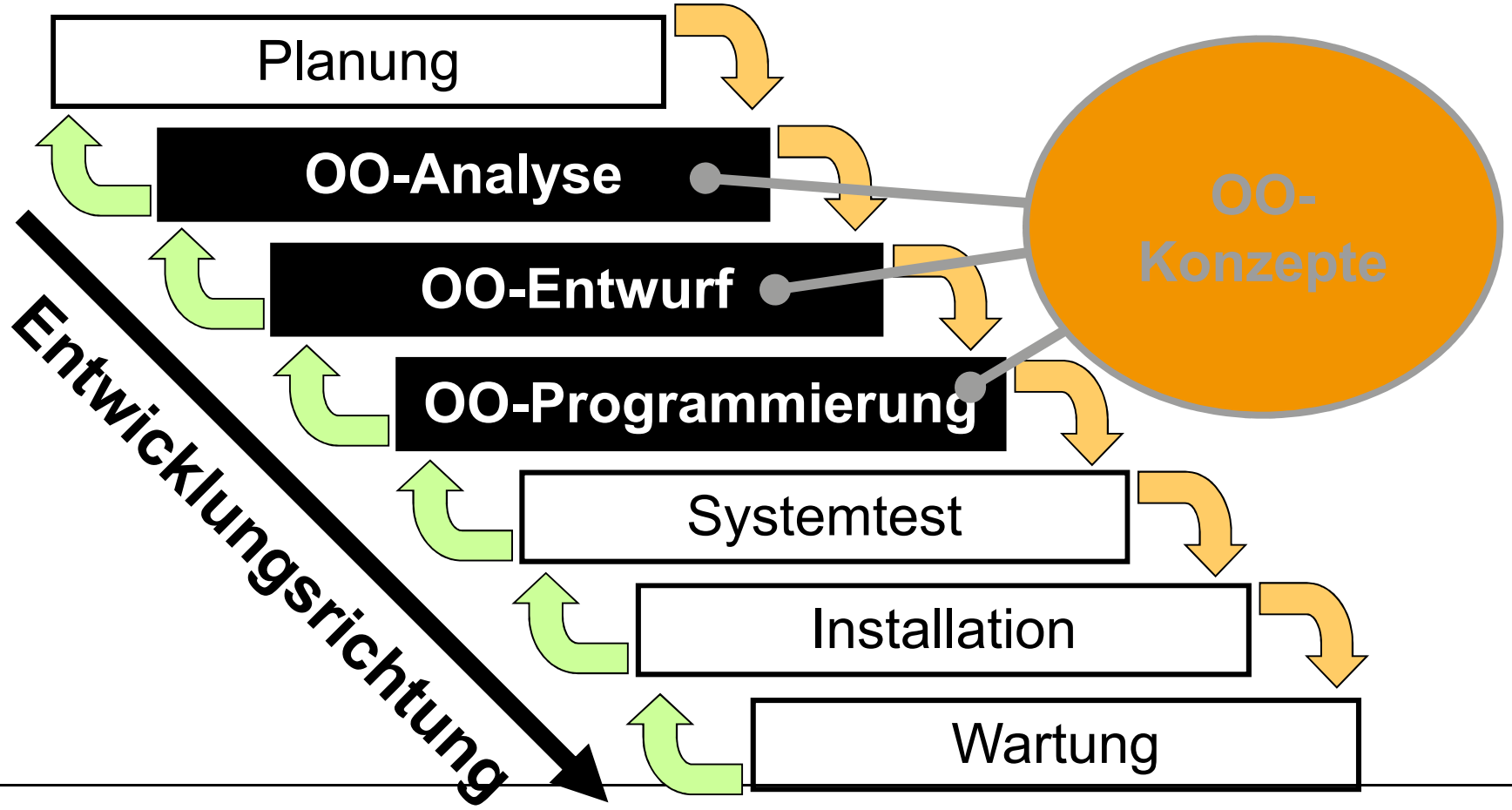
Evolutionäres Modell



Sichtweisen
der Softwareentwicklung

Sichtweise	Beispiel
algorithmisch	Transportsteuerung
funktional	Kundenverwaltung
datenorientiert	Berichtssysteme
regelbasiert	Planungssysteme
zustandsorientiert	Automatensteuerung
objektorientiert	Administrative Systeme
szenariobasiert	Optimierungssysteme

Phasen der OO-Softwareentwicklung



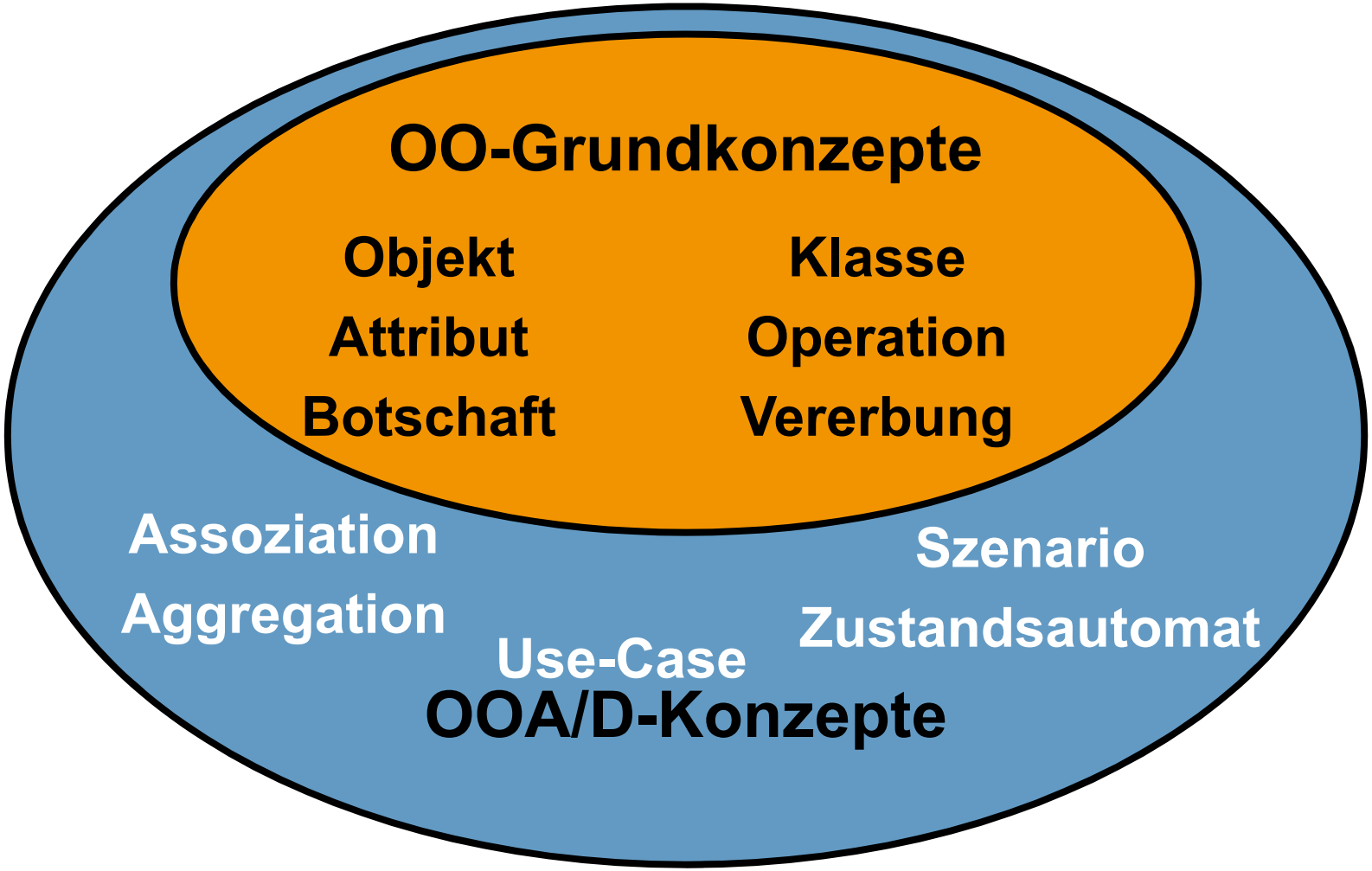
Vorteile der "durchgängigen" OO-Entwicklung

- Unterstützung der inkrementellen Erweiterung des Systems (Varianten)
- Unterstützung der systematischen Wiederverwendung vorhandener Software

⇒ **anpaßbar, erweiterbar, wiederverwendbar**

⇒ **Reduktion der Entwicklungskosten**

Objektorientierte Konzepte

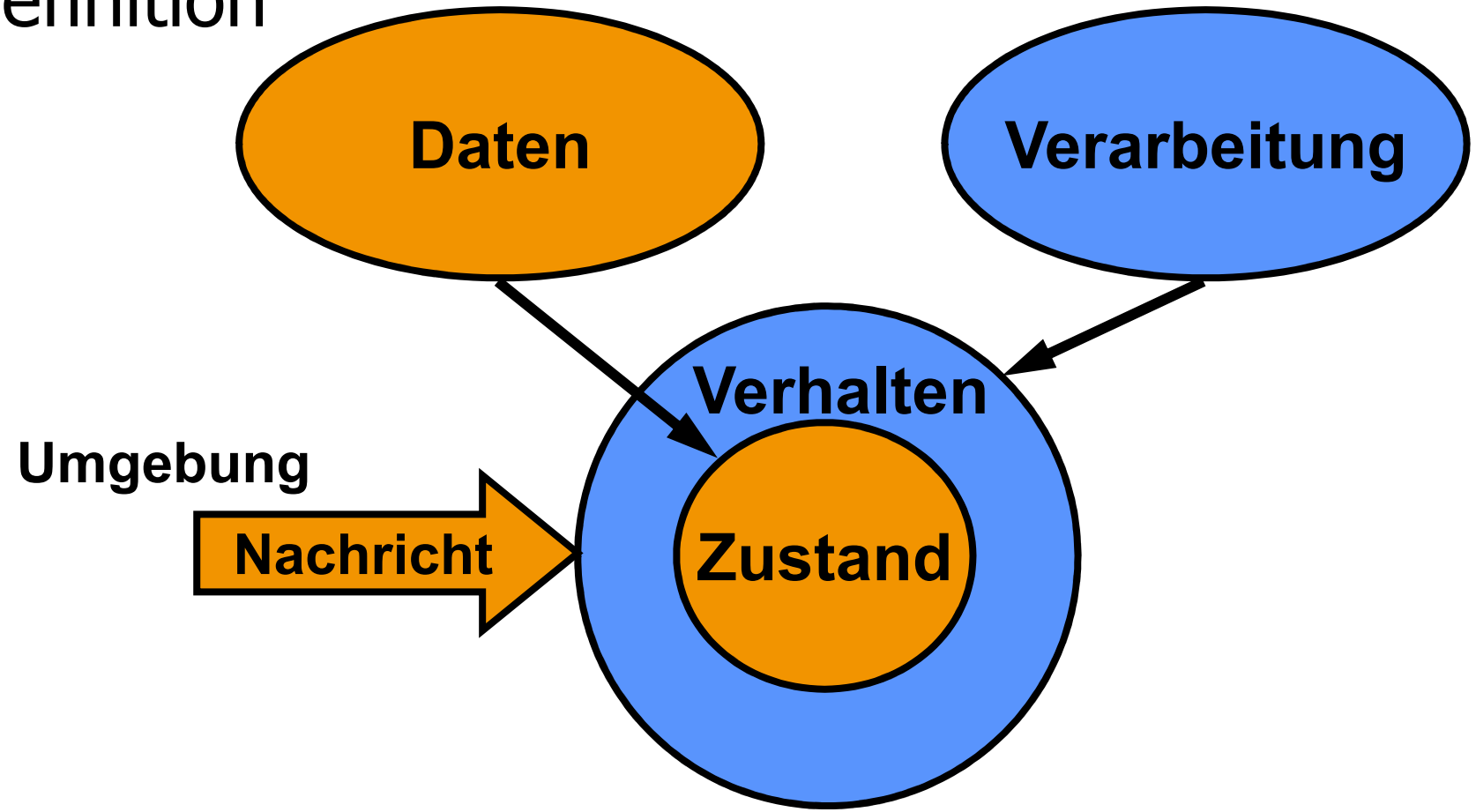


Was ist ein Objekt? (1) alg. Sprachgebrauch

- **ein Gegenstand des Interesses**,
  insbesondere einer Beobachtung, Untersuchung oder Messung

  - Dinge (Fahrrad, Büro)
  - Personen (Kunde, Mitarbeiter)
  - Begriffe (Krankheit, Farbe)

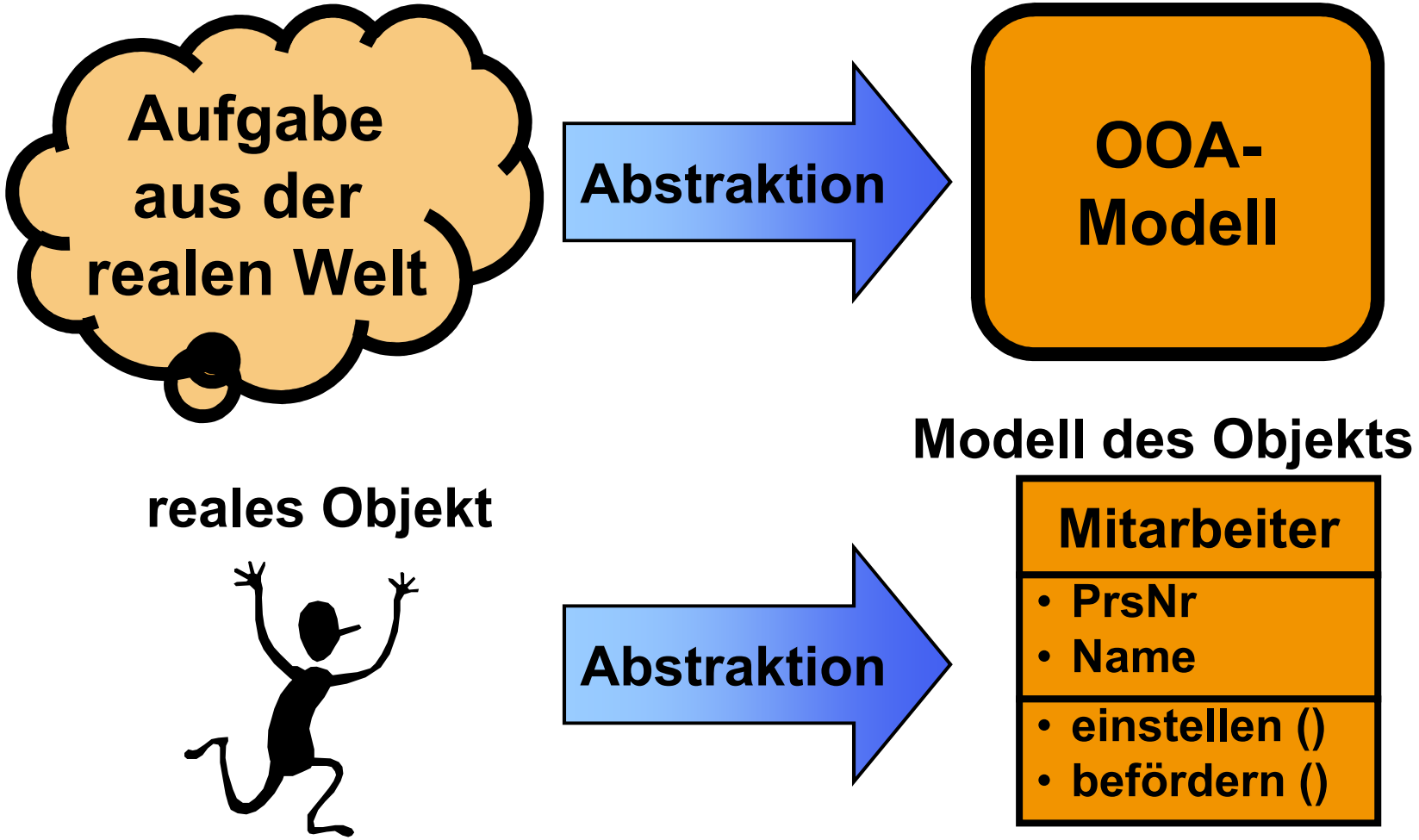
Was ist ein Objekt (2) IT-Definition



Vorteile von Objekten

- Jedes Objekt ist prinzipiell nur für seine eigenen Daten verantwortlich
- Datenmanipulationen erfolgen nur durch das Objekt
- Änderungsanforderungen betreffen meist Daten und Verarbeitung
- Für Begriffe der realen Welt gibt es in den Objekten eines objektorientierten Modells eine direkte, als natürlich empfundene Entsprechung

Objektorientierte Analyse



Abstraktion

"Ein Modell ist eine Abstraktion, die dazu dient, ein System zu verstehen, bevor es gebaut wird. Weil ein Modell auf unwesentliche Details verzichtet, läßt es sich leichter manipulieren [untersuchen, verändern, verstehen] als das Original."

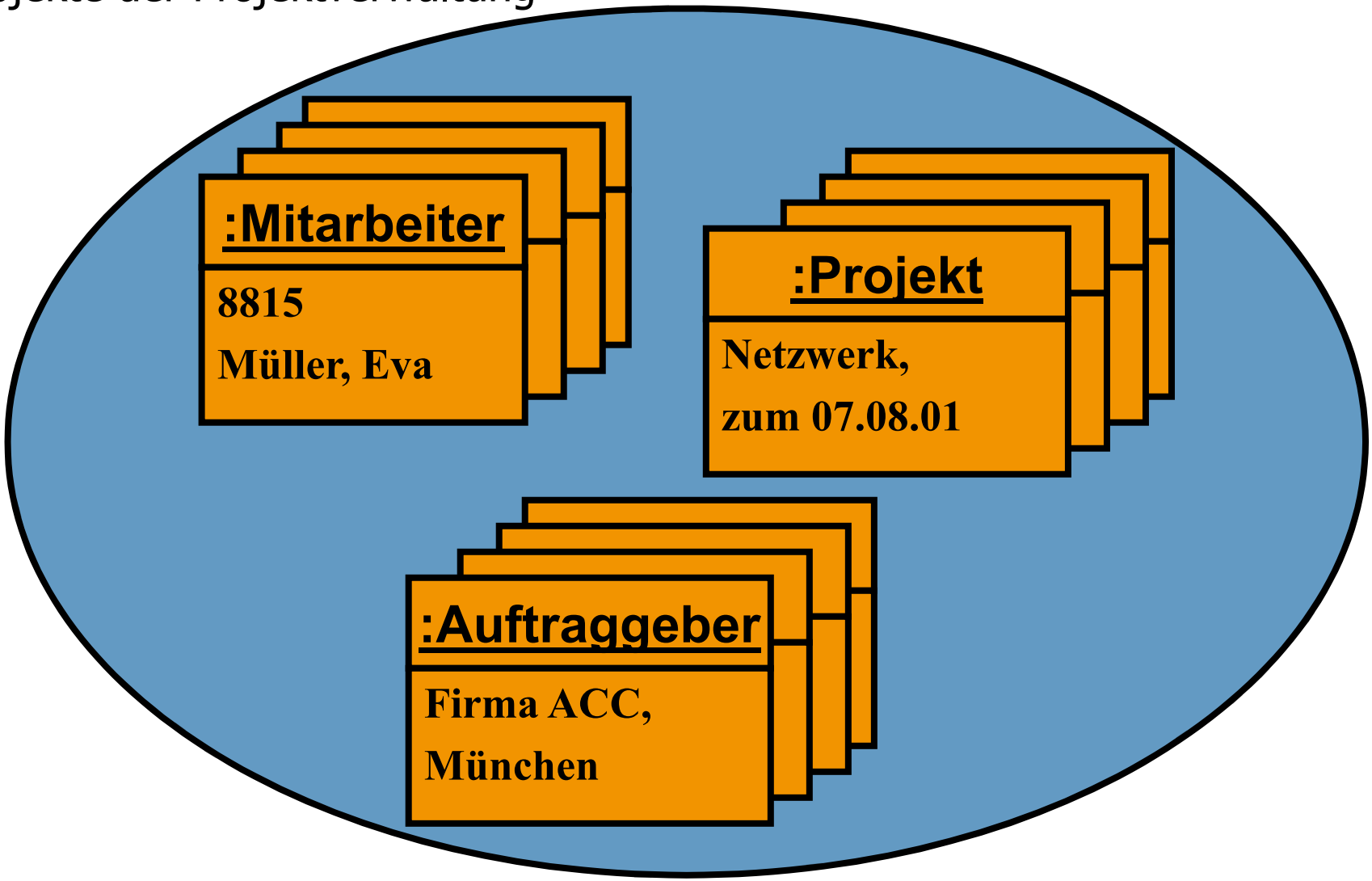
James Rumbaugh

Beispiel: Projektverwaltung

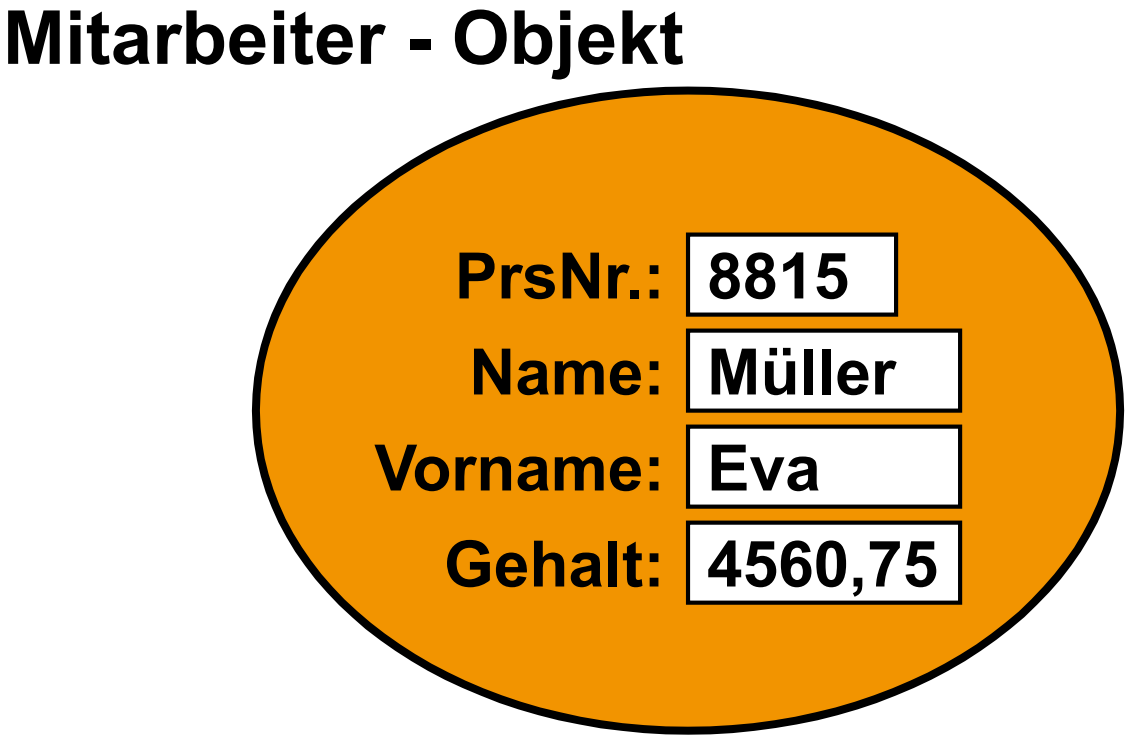
Anforderungen:

- Personalverwaltung
- Auftragserfassung
- Projektteam
- Kosten- / Terminkontrolle

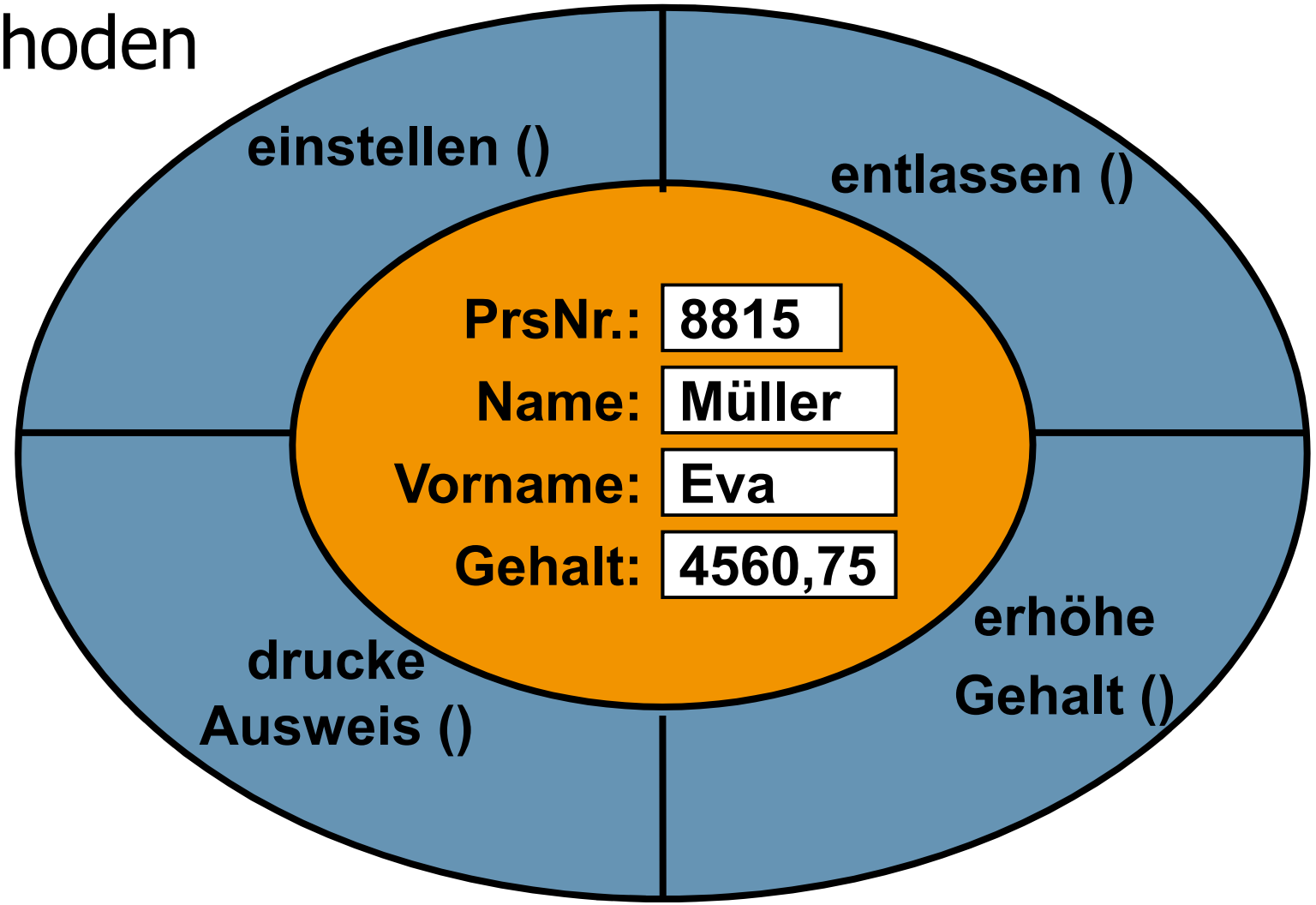
Objekte der Projektverwaltung



Zustand eines Objekts: Attribute

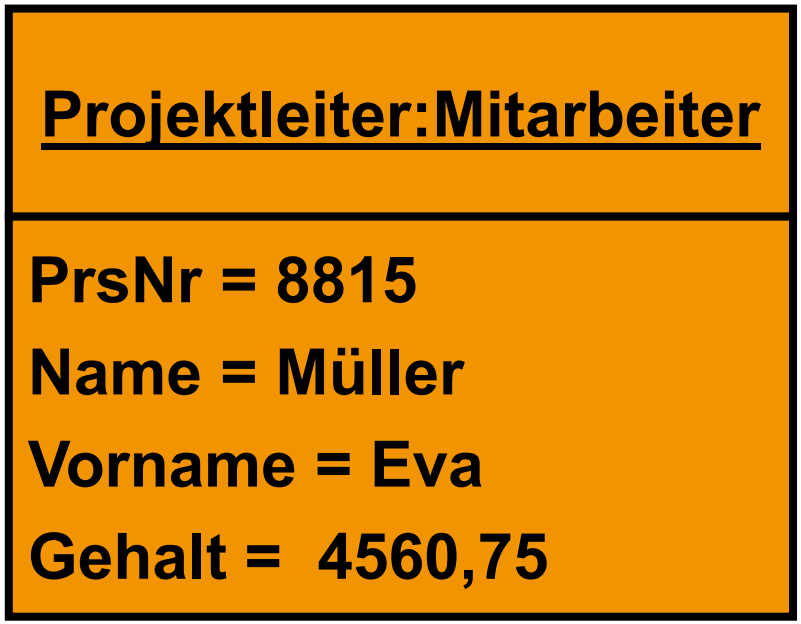


Verhalten eines Objekts: Methoden



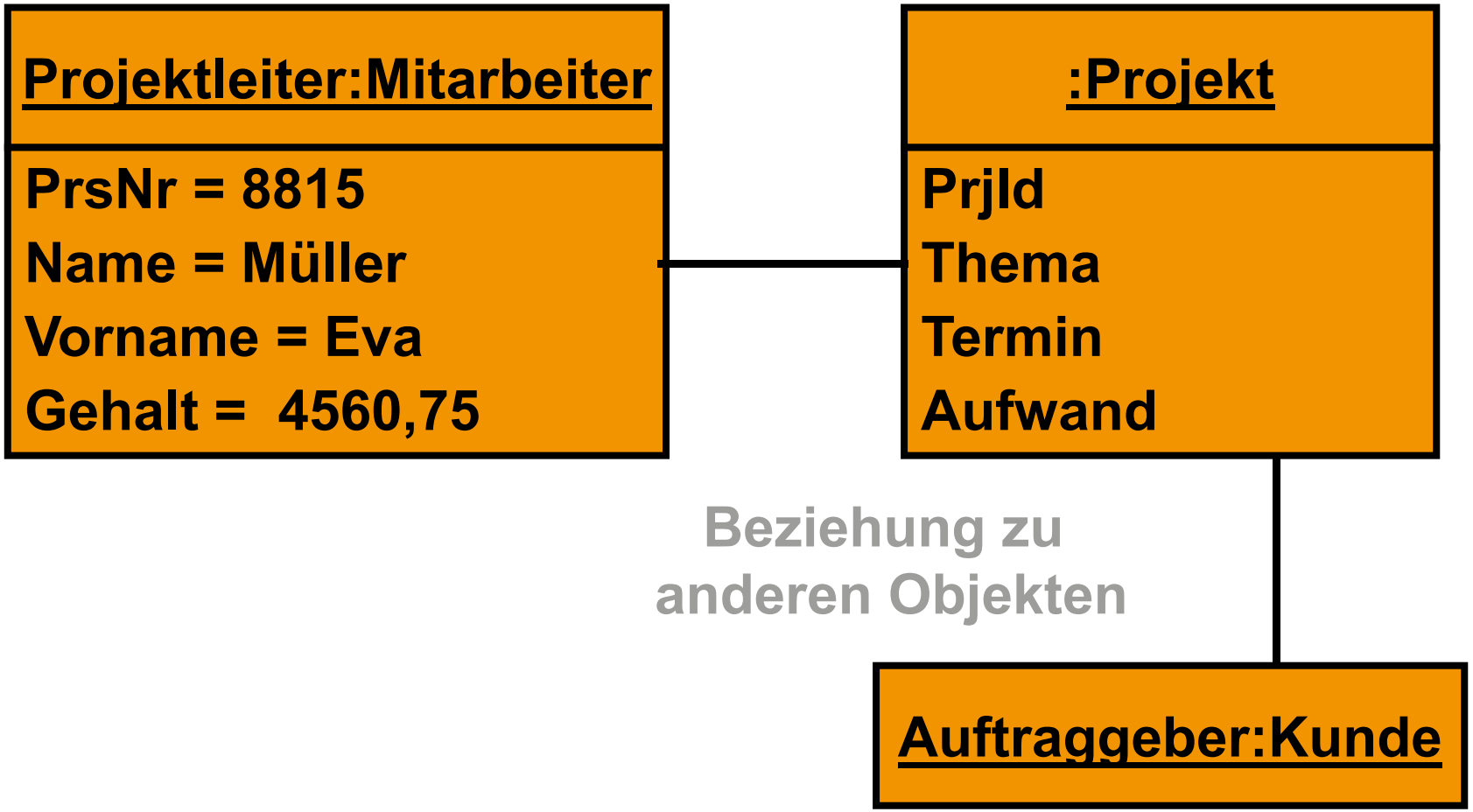
Objekt-Notation in der UML

- **konkretes Objekt**

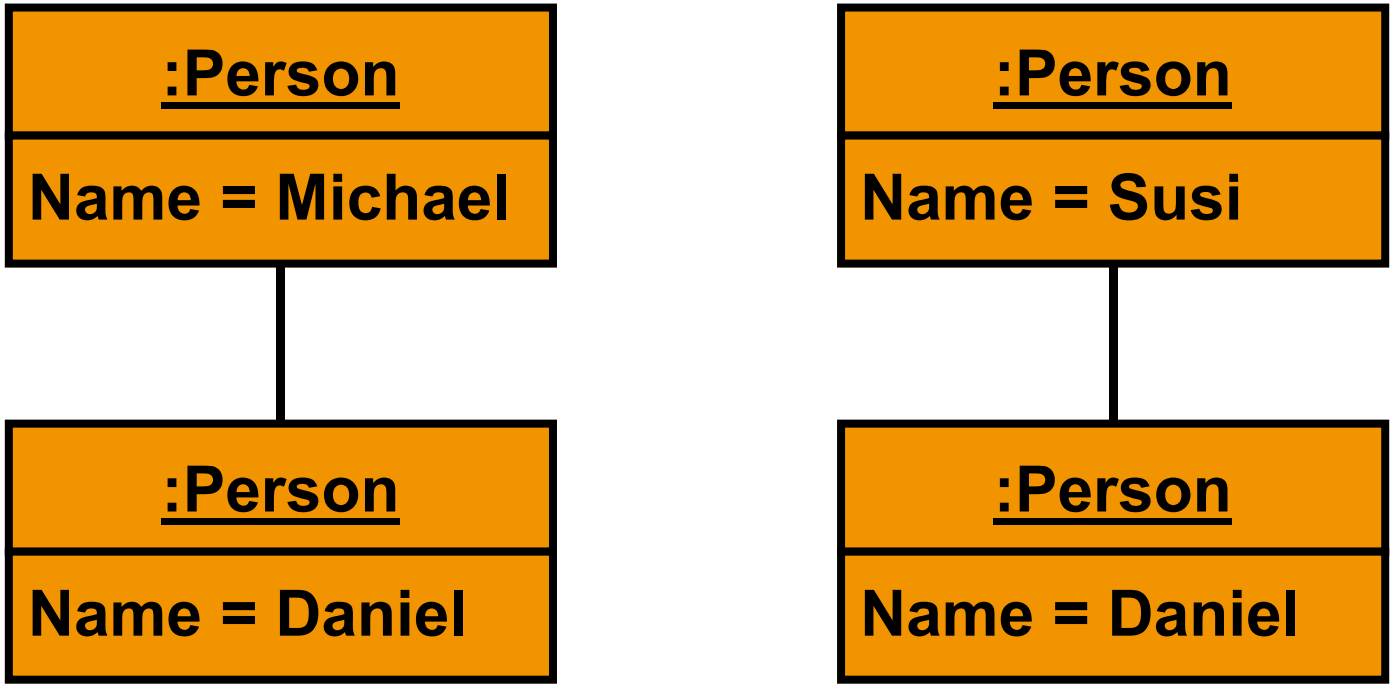


- **anonymes Objekt**

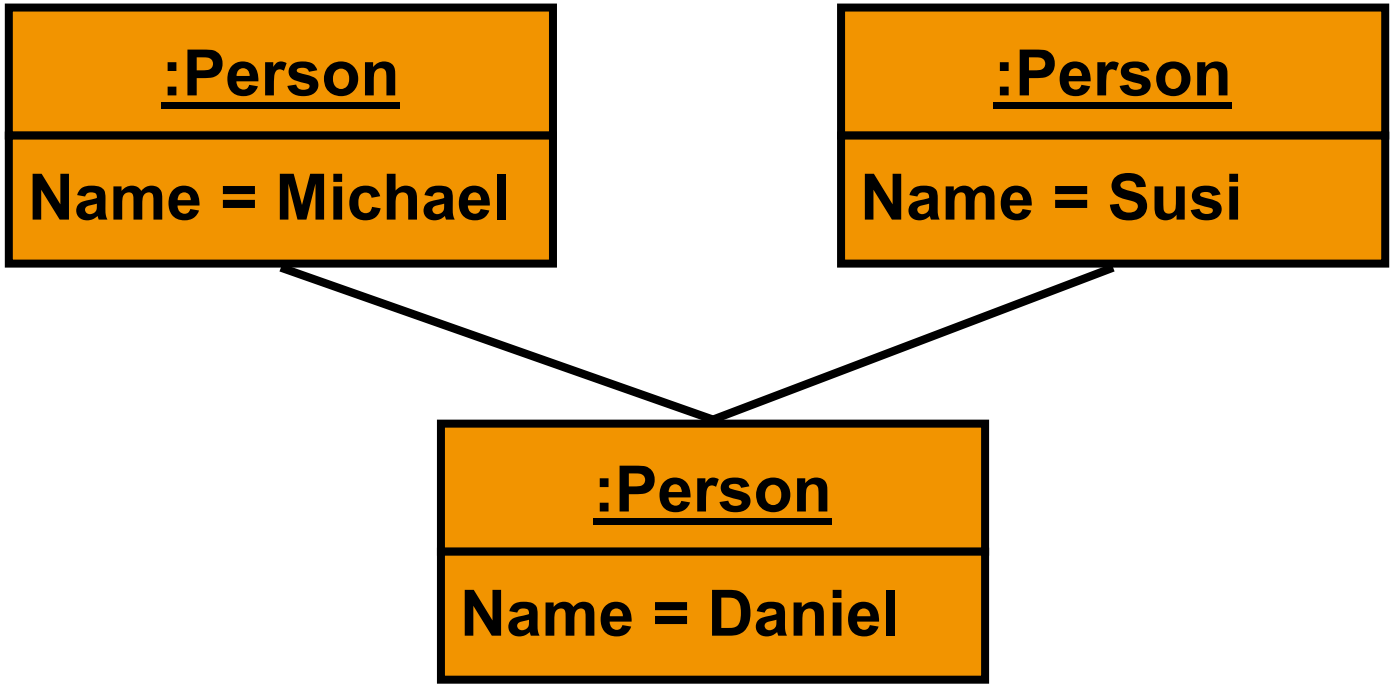
Objektdiagramm



Objektidentität (1) Gleichheit

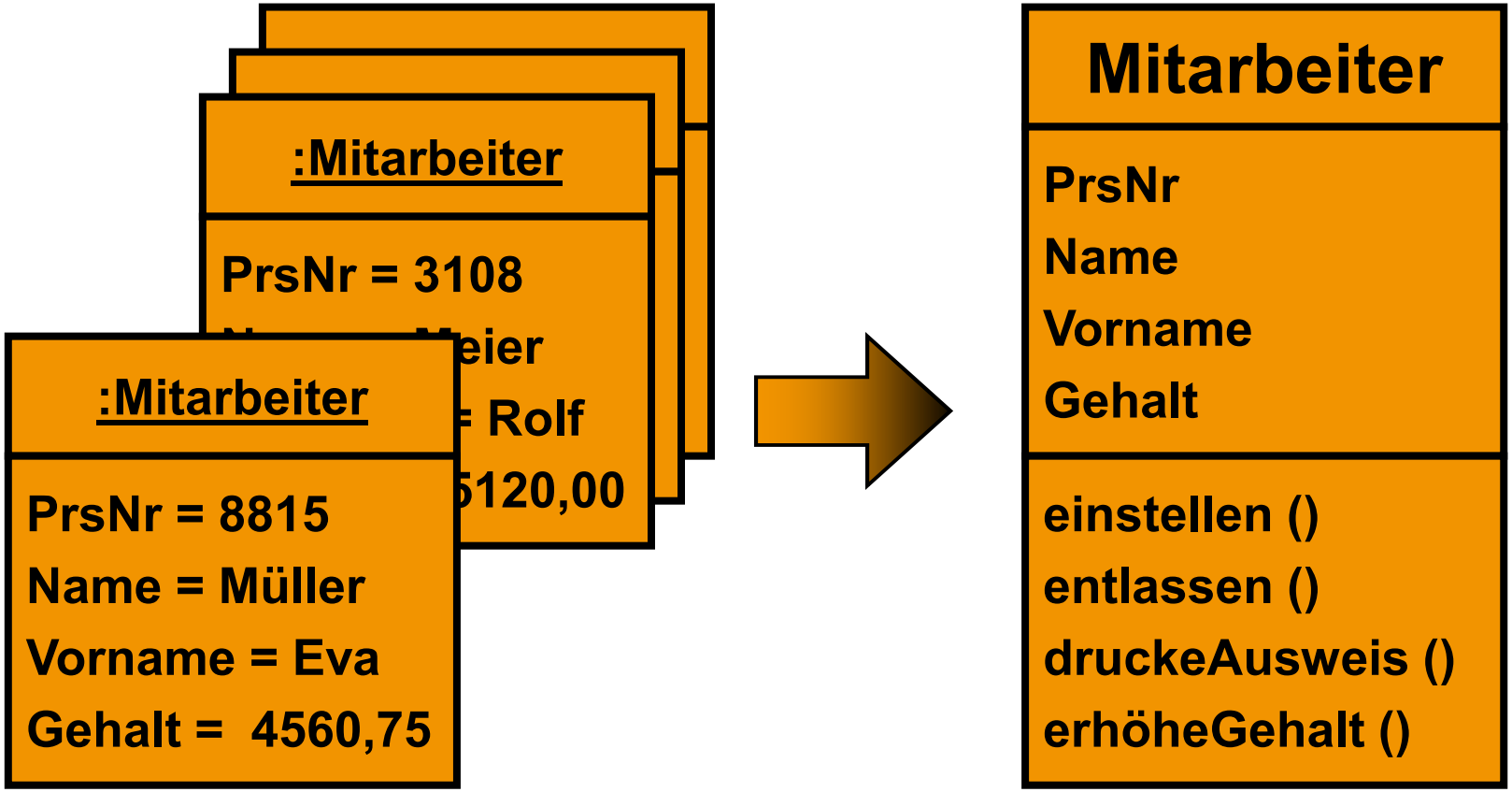


Objektidentität (2) Identität



Konzept der Klasse (1)

- **Objekte**
- **Klasse**

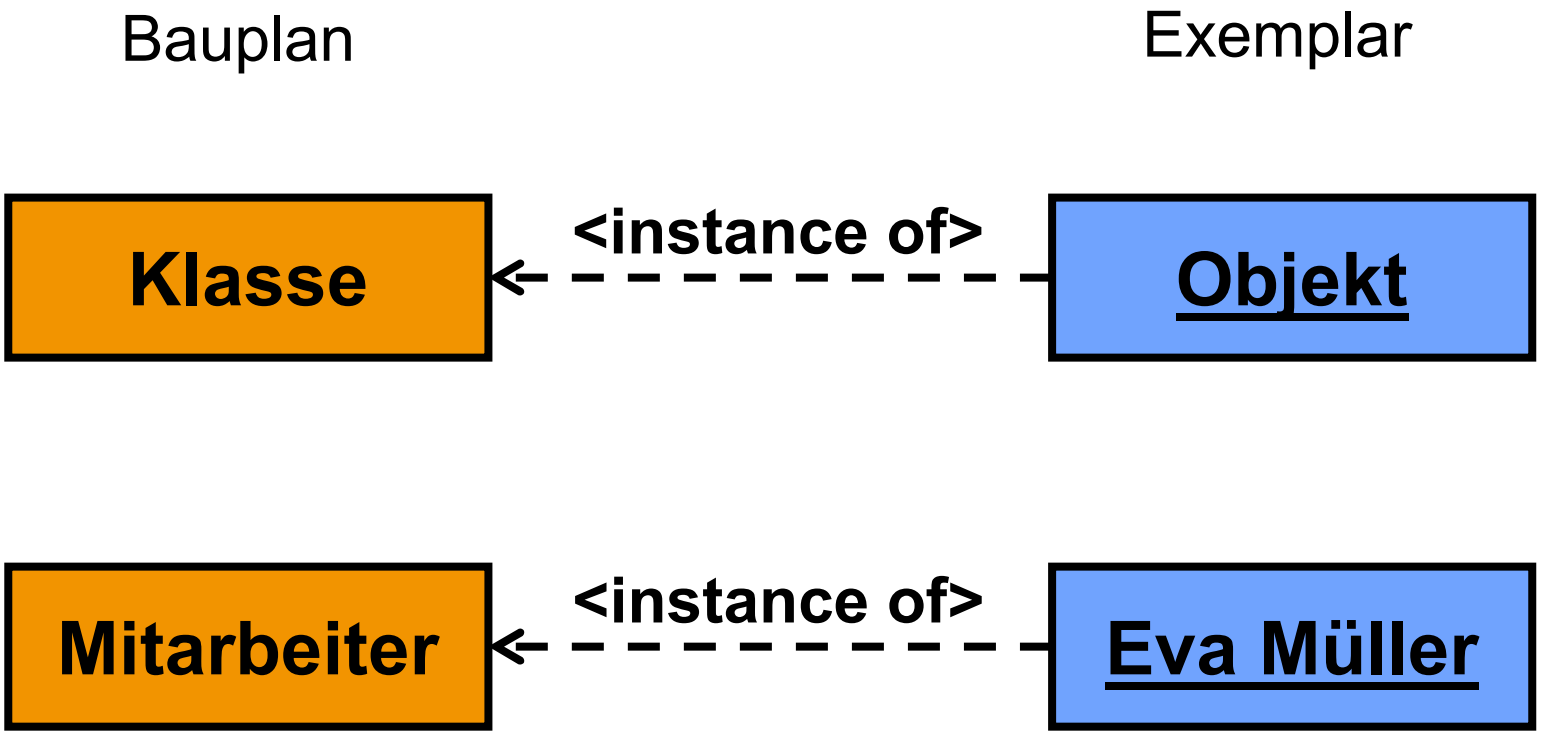


Konzept der Klasse (2)

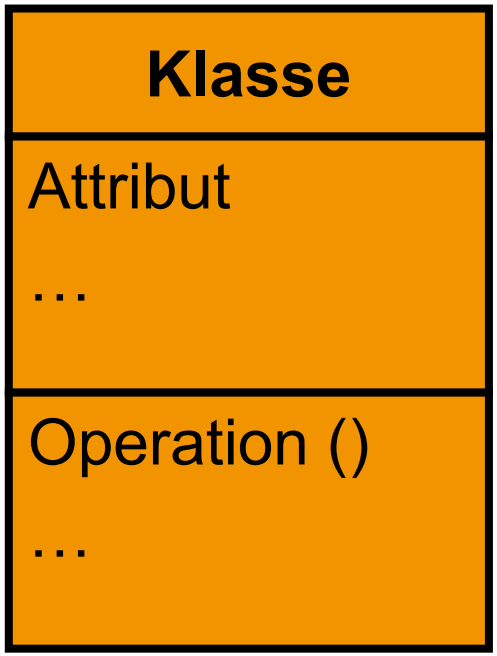
- **Klasse**
  beschreibt die Struktur und das Verhalten einer Menge gleichartiger Objekte. Sie ist demnach der **Bauplan** für eine ganze Menge von Objekten.

- **Objekt**
  ist eine zur Ausführungszeit vorhandene Instanz (Exemplar), die sich entsprechend der Definition – dem Bauplan – ihrer Klasse verhält. Durch Instanzierung (Konstruktion) entsteht aus dem Bauplan ein neues Objekt, welches sich von allen anderen Objekten dieser Klasse eindeutig identifizieren lässt.

Instanz, Objekt, Exemplar



Klassen-Notation in UML



Namensfeld

Attributliste

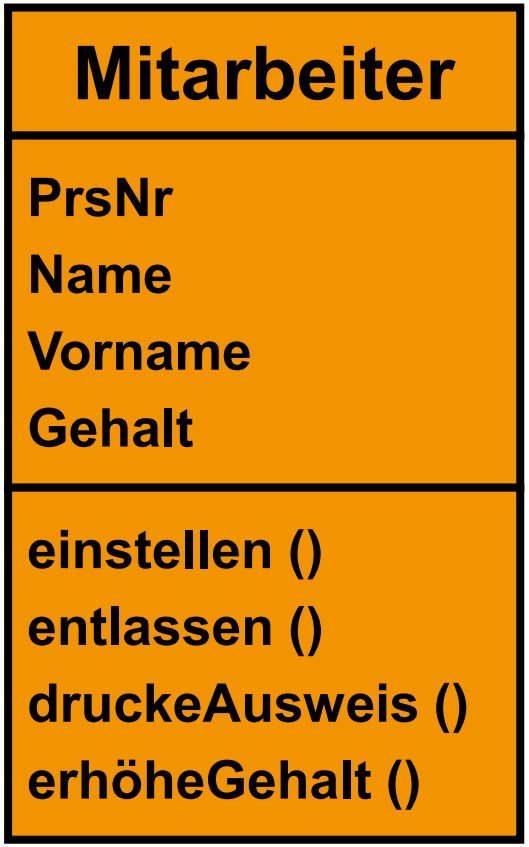
Methodenliste

Klassenbeschreibung

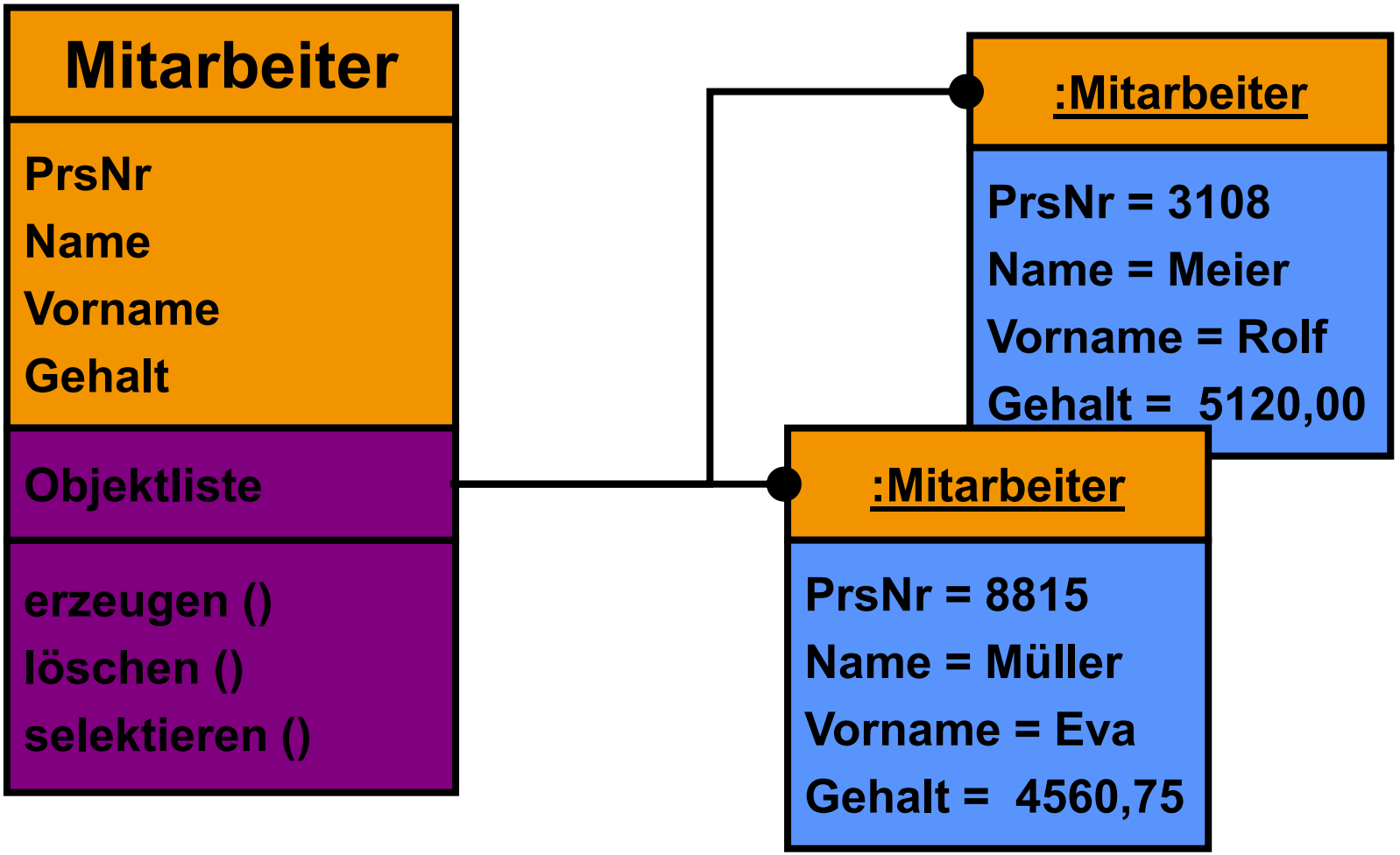
- **Klasse**

Klasse: Mitarbeiter

Jede Person, die in einem Angestelltenverhältnis oder als freier Mitarbeiter für das Unternehmen tätig ist und in Projekten mitarbeiten kann.



Objektverwaltung



Attributbeschreibung

- wird beschrieben durch Name und Typ:
  `Name: Typ [= Anfangswert]`
- ist "geheim"
- Optional:
  - Anfangswerte (initial values)
  - Liste von Merkmalen
  - Ist nicht veränderbar (konstant)
  - Mandatory/Optional
  - Kann berechnet werden
  - Ist ein Klassenattribut (im Gegensatz zu Instanzattribut)

Typ eines Attributs

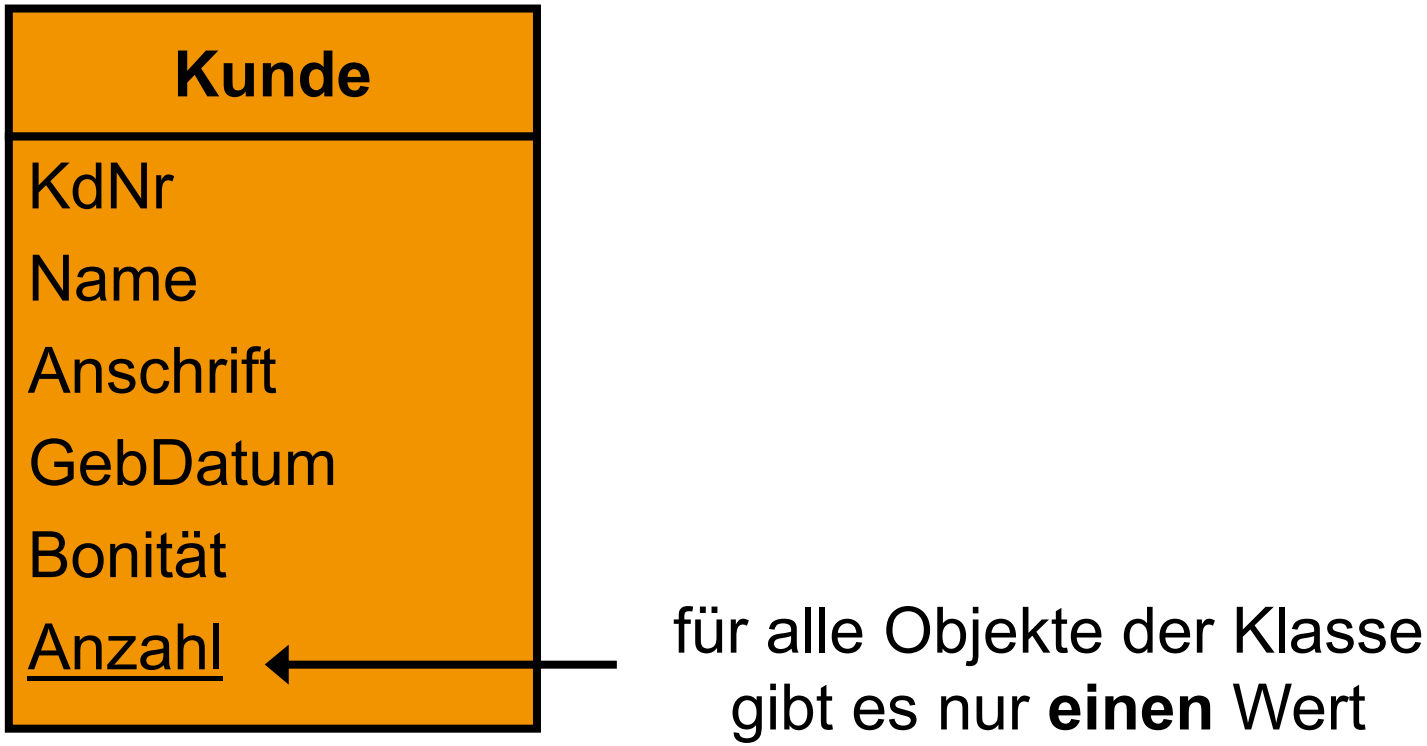
- **Standardtypen**
  String, Int, Float, Decimal(fixed), Boolean, Date, Time, Timestamp
- **Aufzählungstypen**
  enum
- (elementare) **Klassen**
  jede andere definierte Klasse als Datentyp eines Attributs
- **List<Typ>, Set<Typ>, Array<Typ>**
  Das Attribut besitzt mehrere (eine Liste von) Werte.
  Der Datentyp für jeden einzelnen Wert kann jede der genannten Möglichkeiten sein.

Klasse Student & Student-Objekt

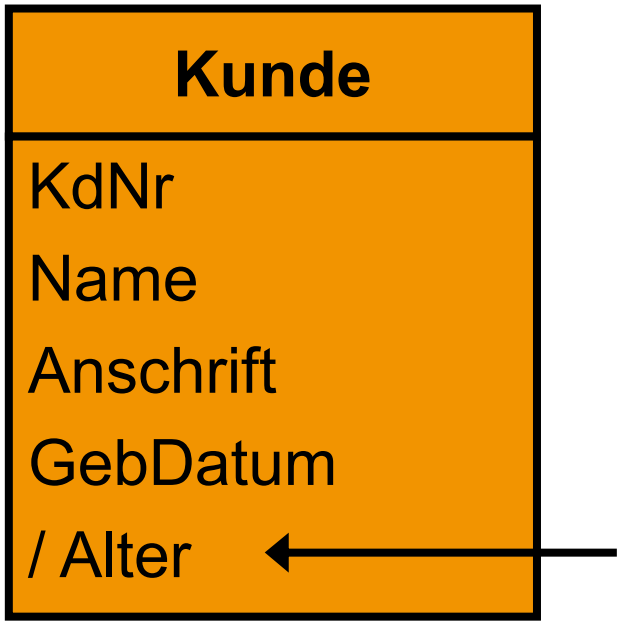
Student
MatrikelNr: integer
Name: String
Anschrift: Adresse
GebDatum: Date
Immatrikulation: Date
VordiplomNoten: List<Ergebnis>

:Student
MatrikelNr = 7003265
Name = (Hans, Meier)
Anschrift = (Parkstr. 7, 92224 Amberg)
GebDatum = 07.08.1976
Immatrikulation = 03.09.1998
Vordiplom
Noten = ((2.3, Mathematik), (1.7, Informatik))

Klassenattribut



Abgeleitetes Attribut

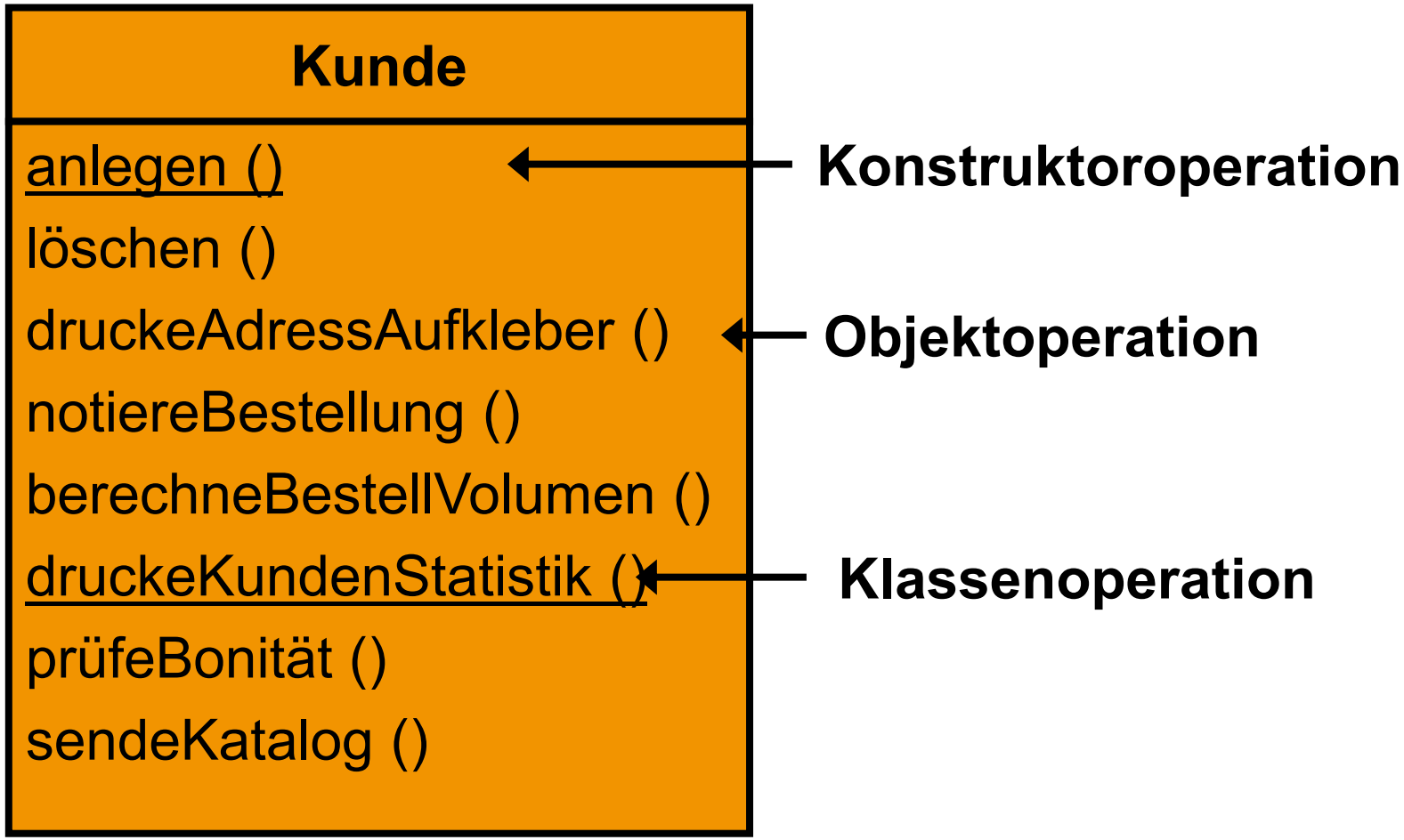


ergibt sich aus Geburtsdatum,
kann nicht geändert werden

Attributspezifikation (weitere Details)

- **Name**
- **Typ (data-typ)**
- **Anfangswert (default or initial value)**
- Muß/Kann Attribut (mandatory/optional)
- Schlüssel (key)
- nicht änderbar (frozen)
- Zulässige Wertebereiche
- Einheit (unit) oder Währung
- Beschreibung

Operationen (Methoden)



Operationsarten

- Operationen mit lesendem Zugriff (accessor)
- Operationen mit schreibendem Zugriff (update)
- Operationen zur Durchführung von Berechnungen
- Operationen zum Erzeugen (constructor) und Löschen (destructor) von Objekten
- Operationen die Objekte einer Klasse selektieren
- Operationen zum Herstellen von Verbindungen zwischen Objekten
- Operationen, die Operationen anderer Klassen aktivieren

Beschreibung von Operationen

Methode:

mit Objekt XY tueEtwas (plus Parameter) → Ergebnis

- Eingabe: Input Parameter
- Ausgabe: Output Parameter, Result
- Wirkung: Beschreibung der Wirkung aus Benutzersicht
- Ausnahmen: Verhalten im Fehlerfall

Teil 2: Konzeption und Gestaltung von Datenbanken

- Statische Konzepte -

2. Statische Konzepte und relationale Datenbanken

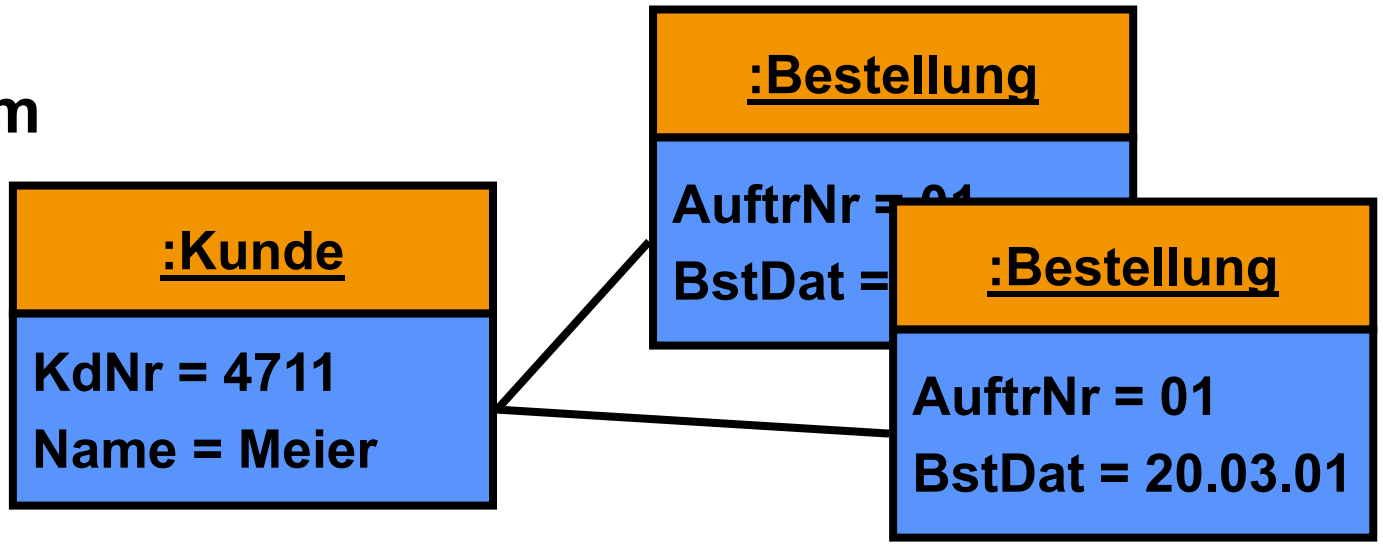
- Statische Konzepte
  - Assoziation
  - Aggregation
  - Komposition
  - Vererbung
  - Paket

- Entwurfsmuster

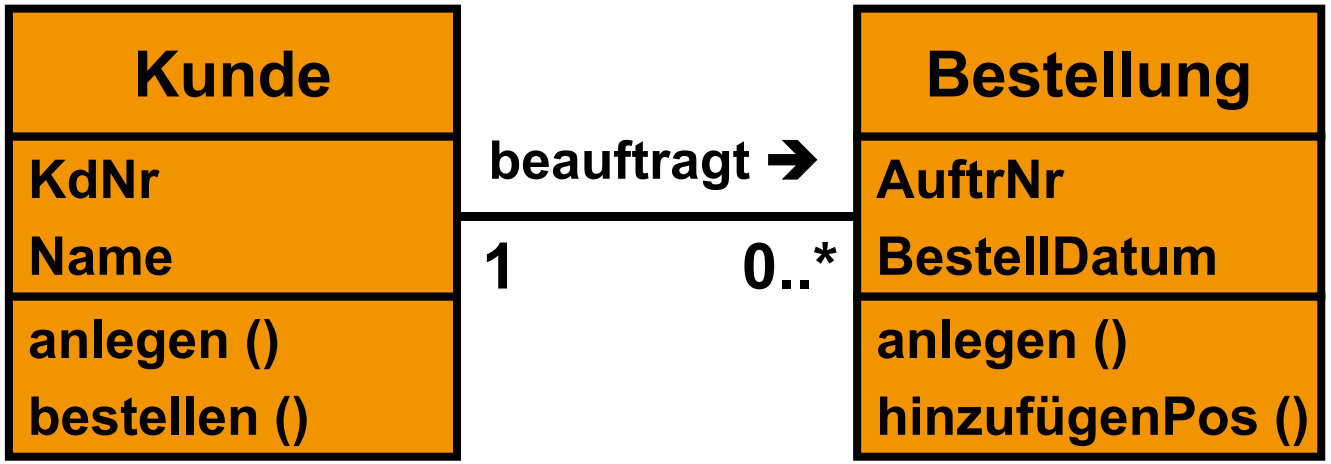
- Umfangreiches Beispiel
  - Kino Programminformation und Sitzplatzreservierung

Assoziation

Objekt-Diagramm



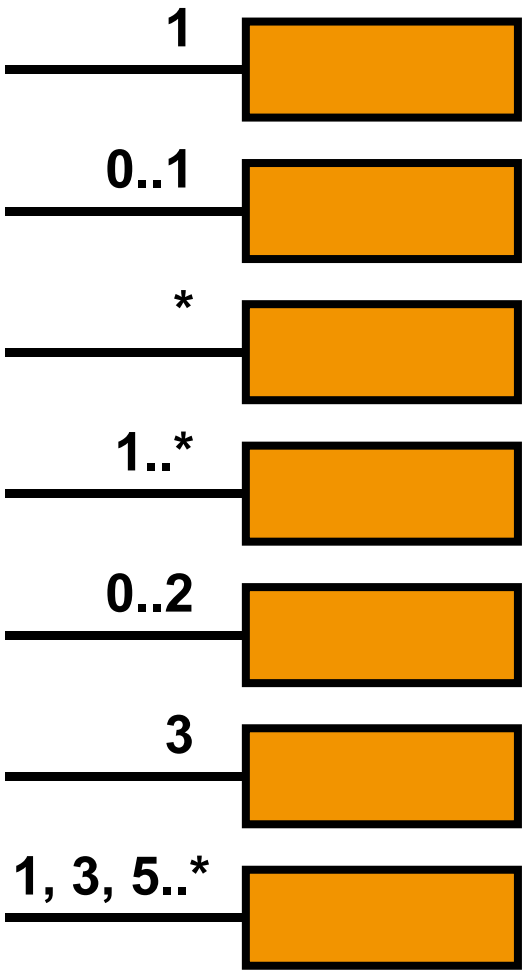
Klassen-Diagramm



Assoziation: Spezifikationen

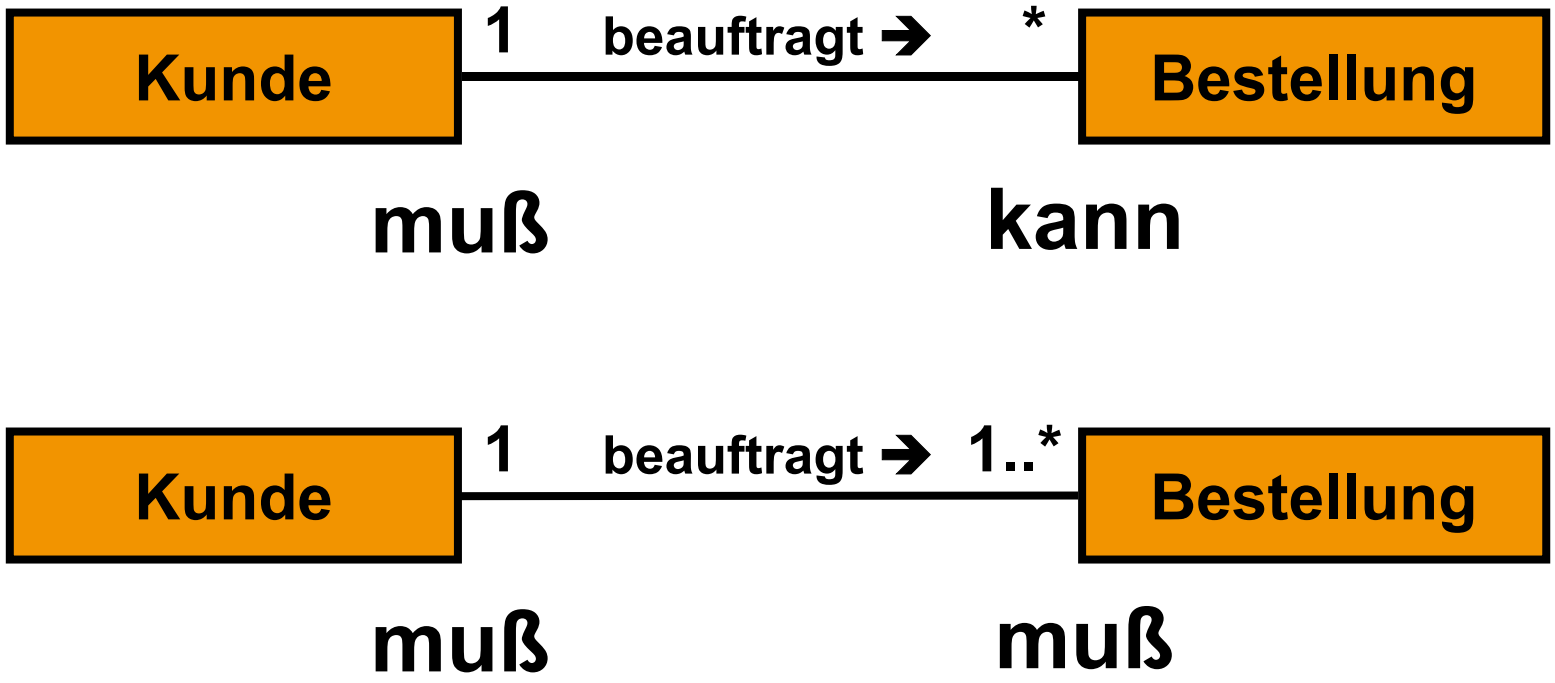
- Name der Assoziation (mit Richtung)
- Wertigkeit (meist binär)
- Richtung (meist bidirektional)
- Kardinalitäten
- Rollenbezeichnungen
- reflexive Assoziation

Assoziation: Kardinalitäten

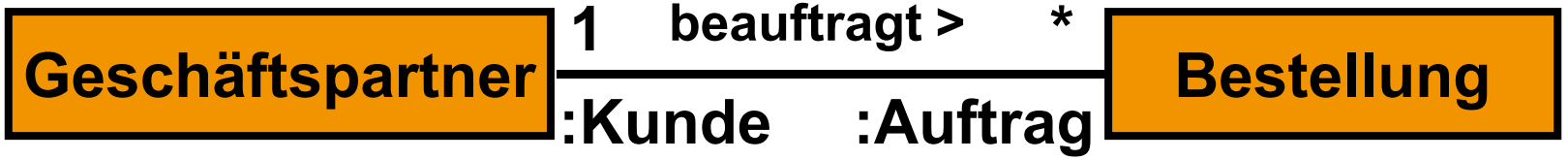


- **genau 1** — **muß**
- **0 oder 1** — **kann**
- **0 bis beliebig** — **kann**
- **1 bis beliebig** — **muß**
- **0, 1 oder 2** — **kann**
- **genau 3** — **muß**
- **nicht 2 oder 4** — **muß**

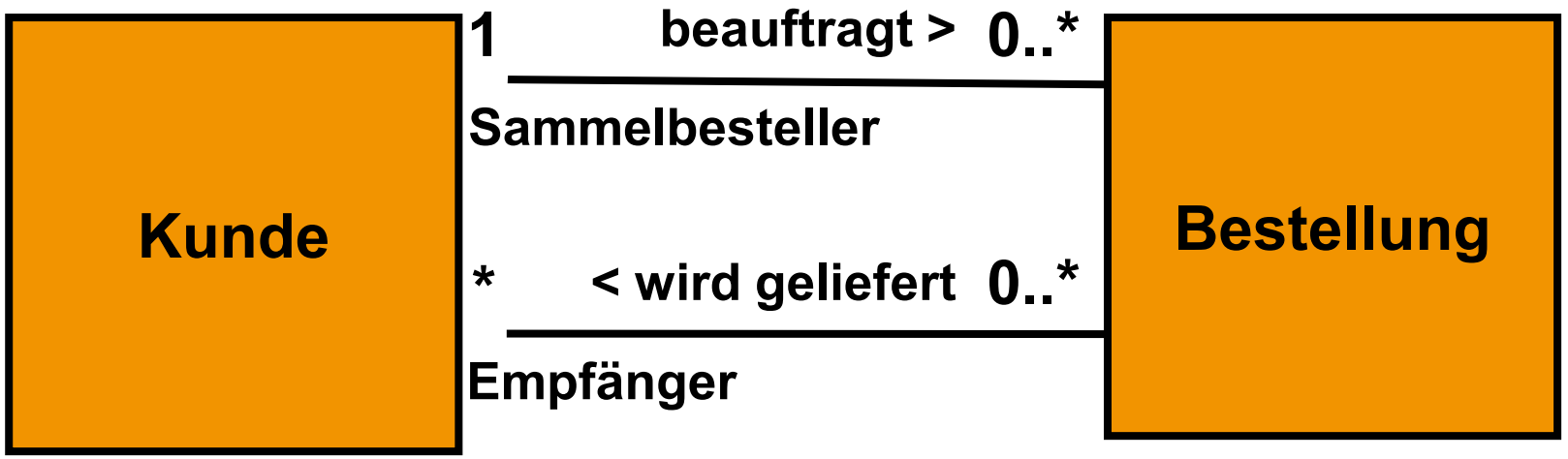
Assoziation: Muß / Kann



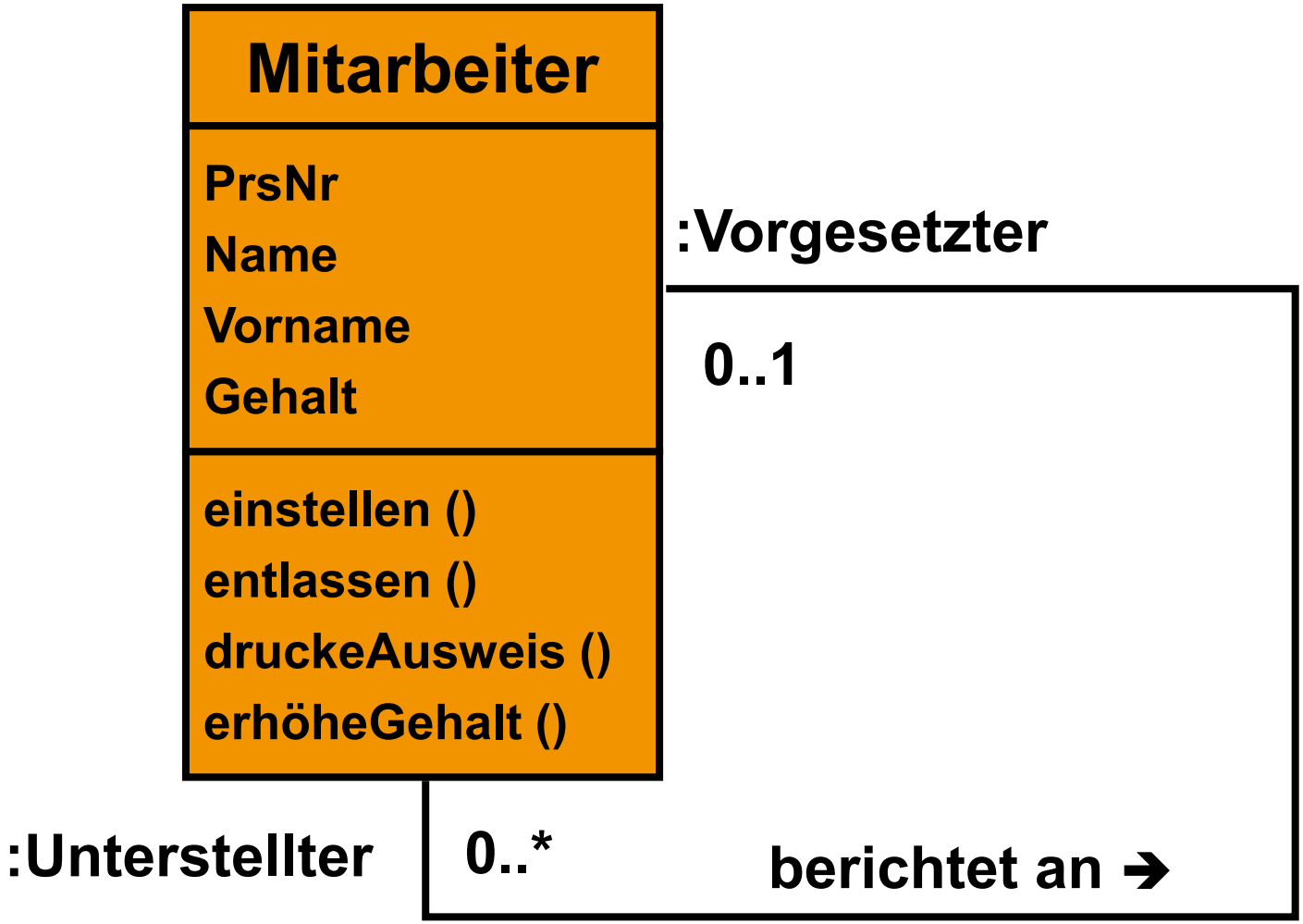
Assoziation: Rollen



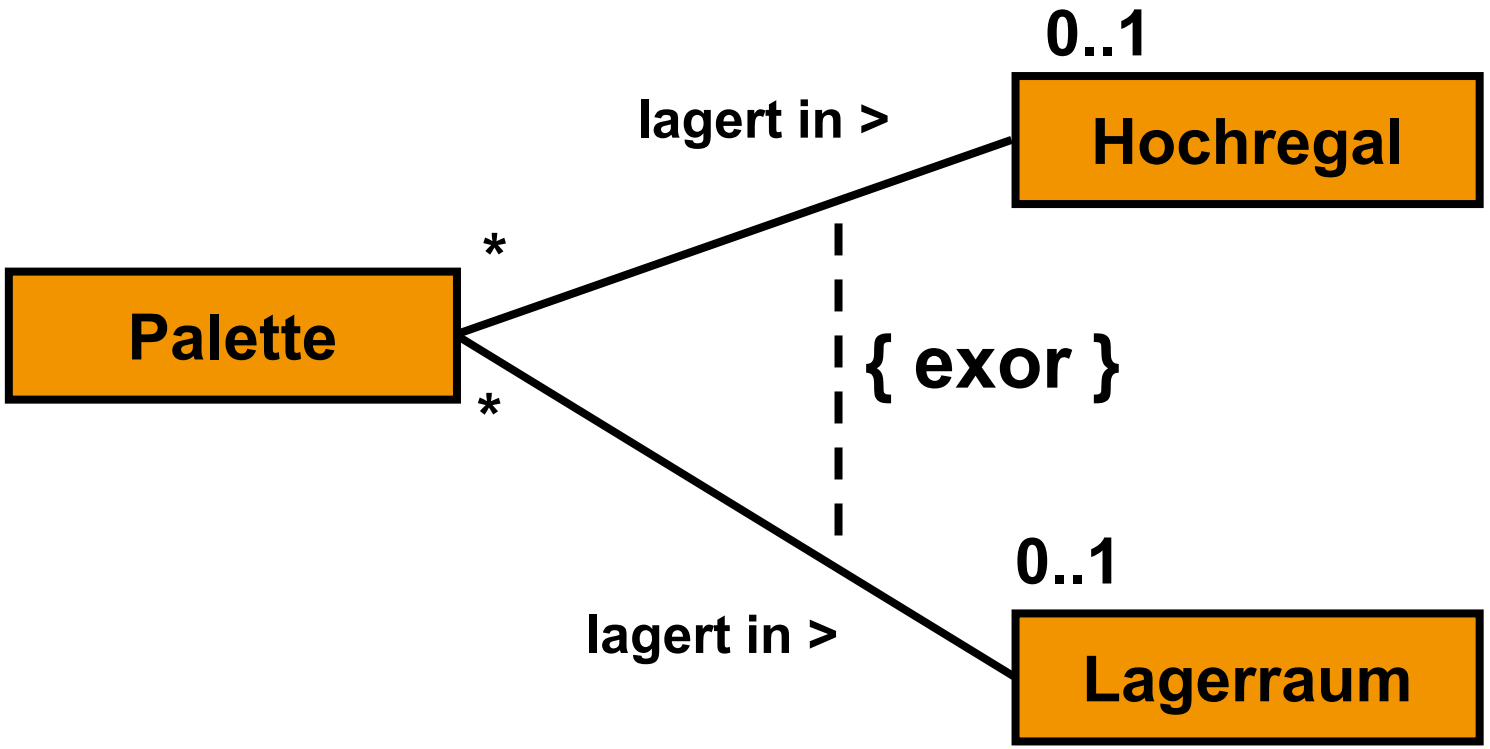
Assoziation: Rollen



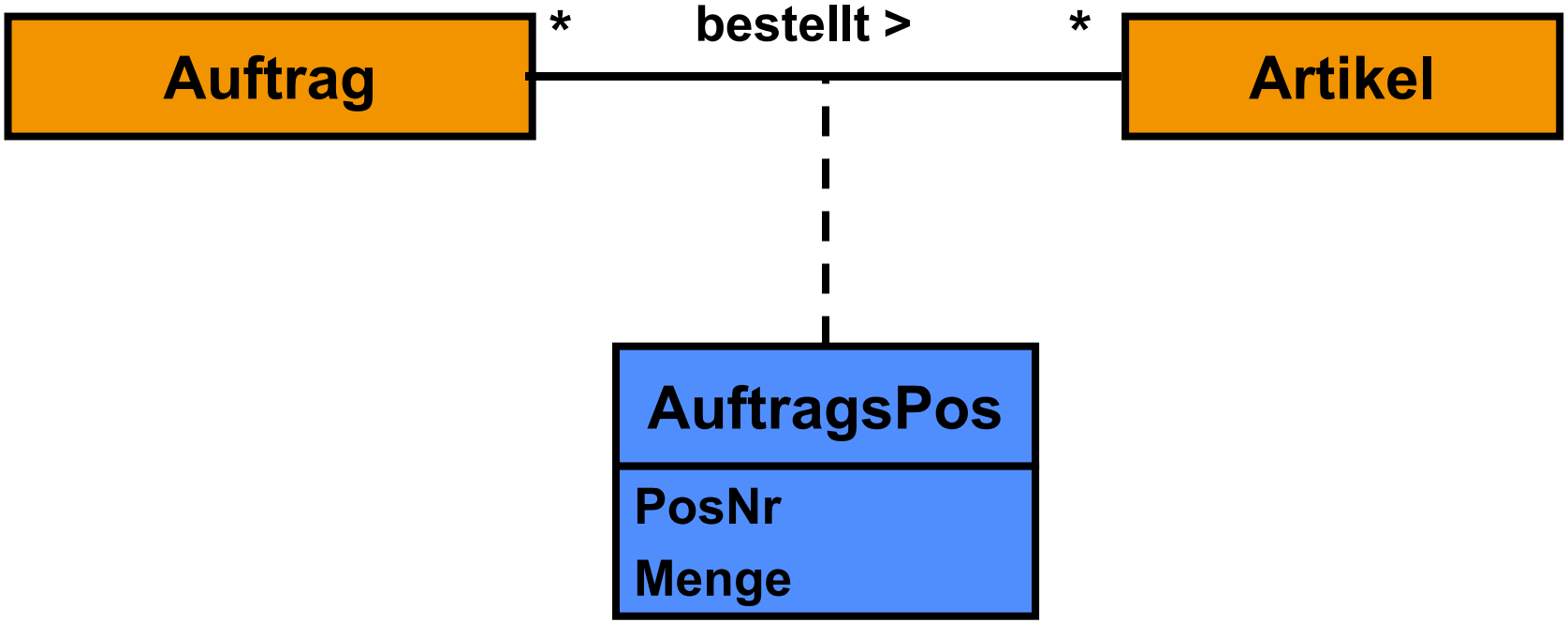
Assoziation: reflexiv



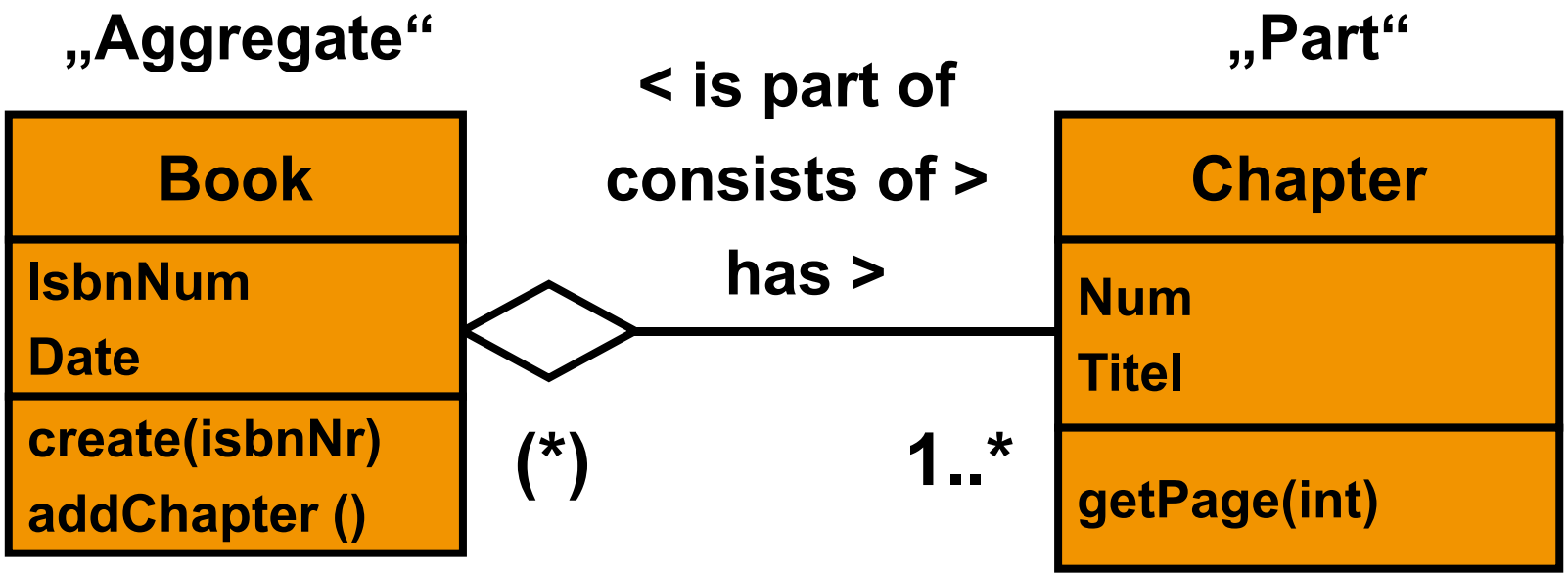
Assoziation: Restriktionen



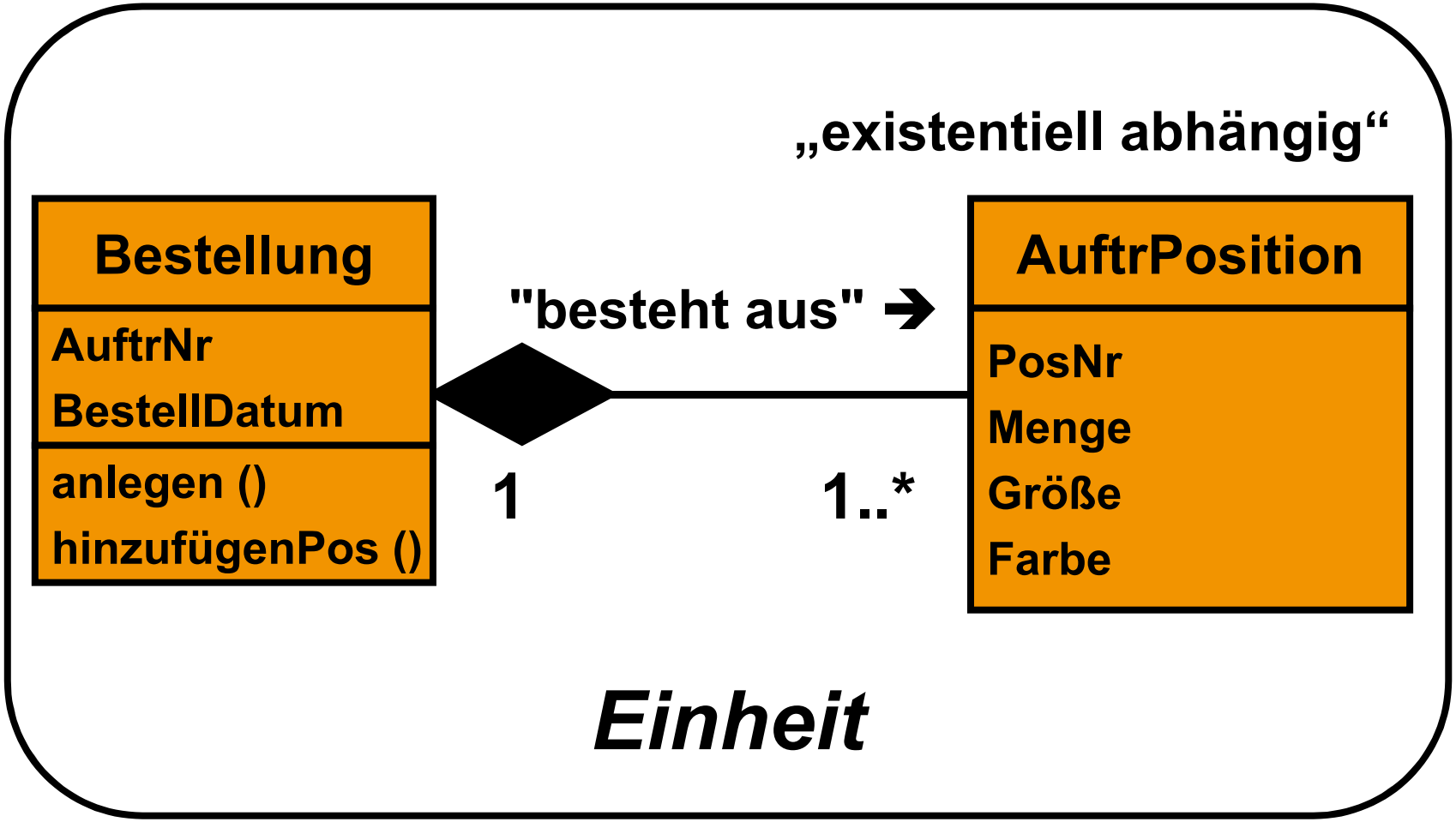
Assoziation: Assoziative Klasse



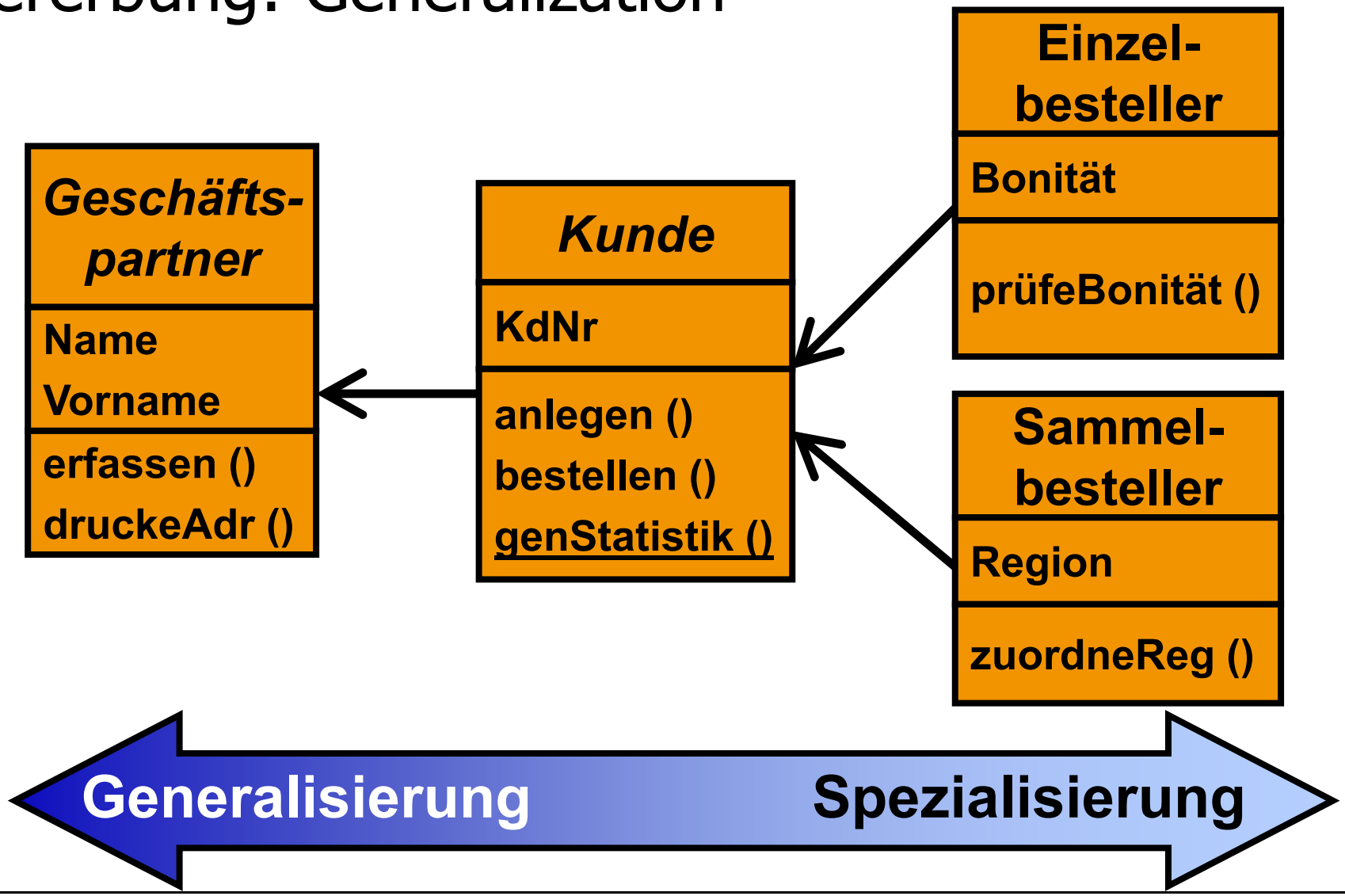
Aggregation



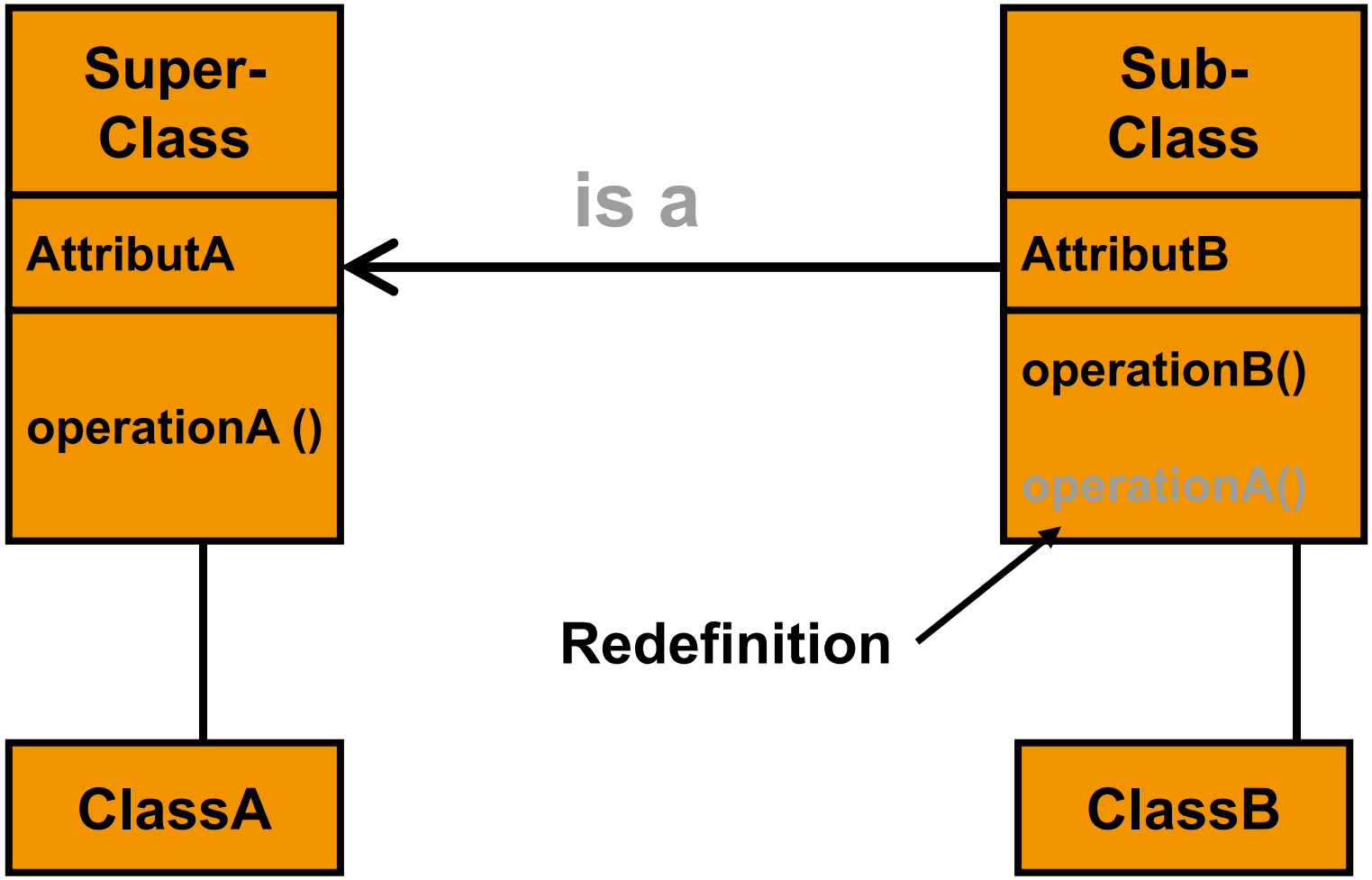
Komposition



Vererbung: Generalization



Vererbung: alg. Schema



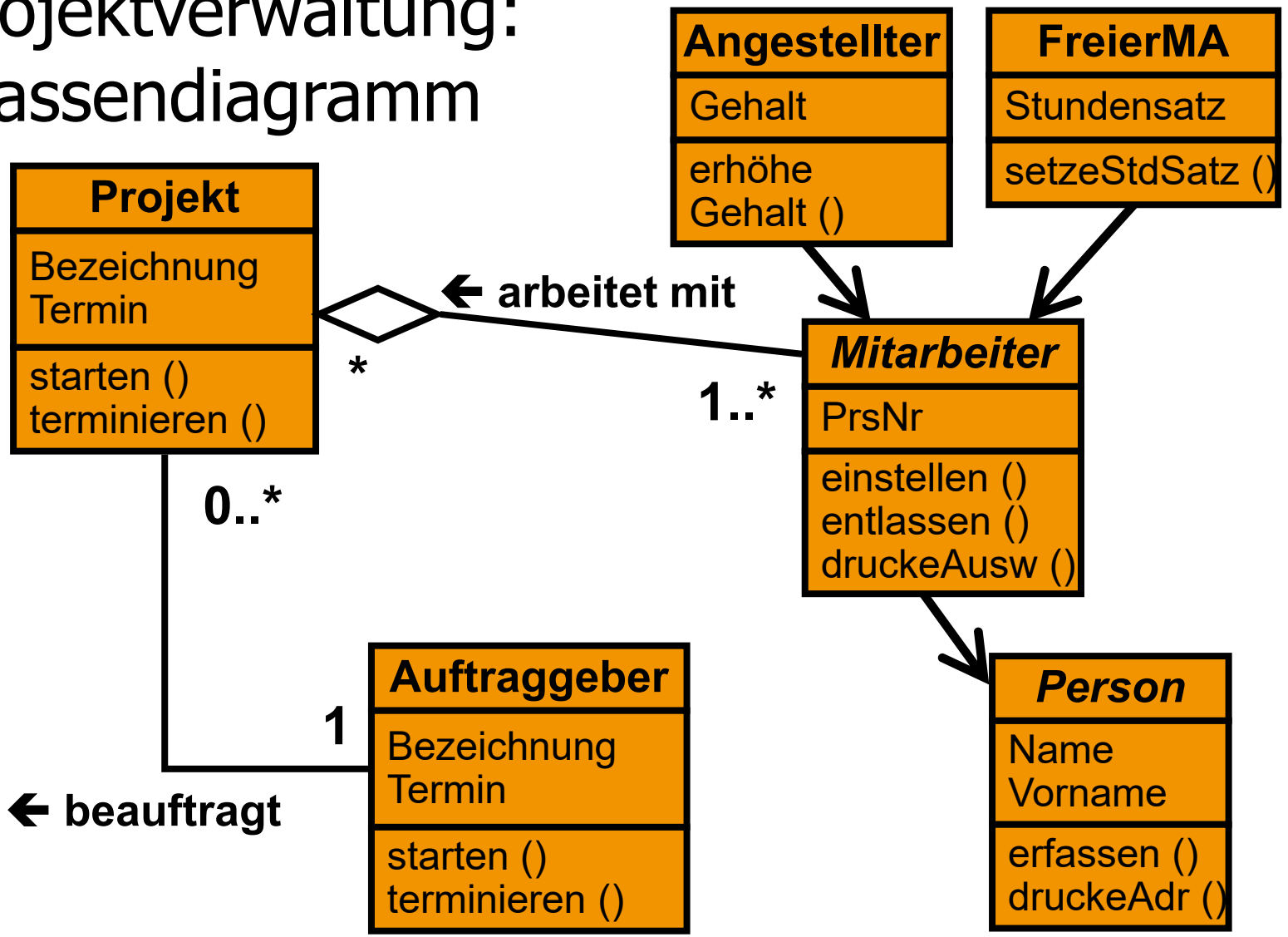
Was wird vererbt?

- Attribute
- Operationen
- Assoziationen

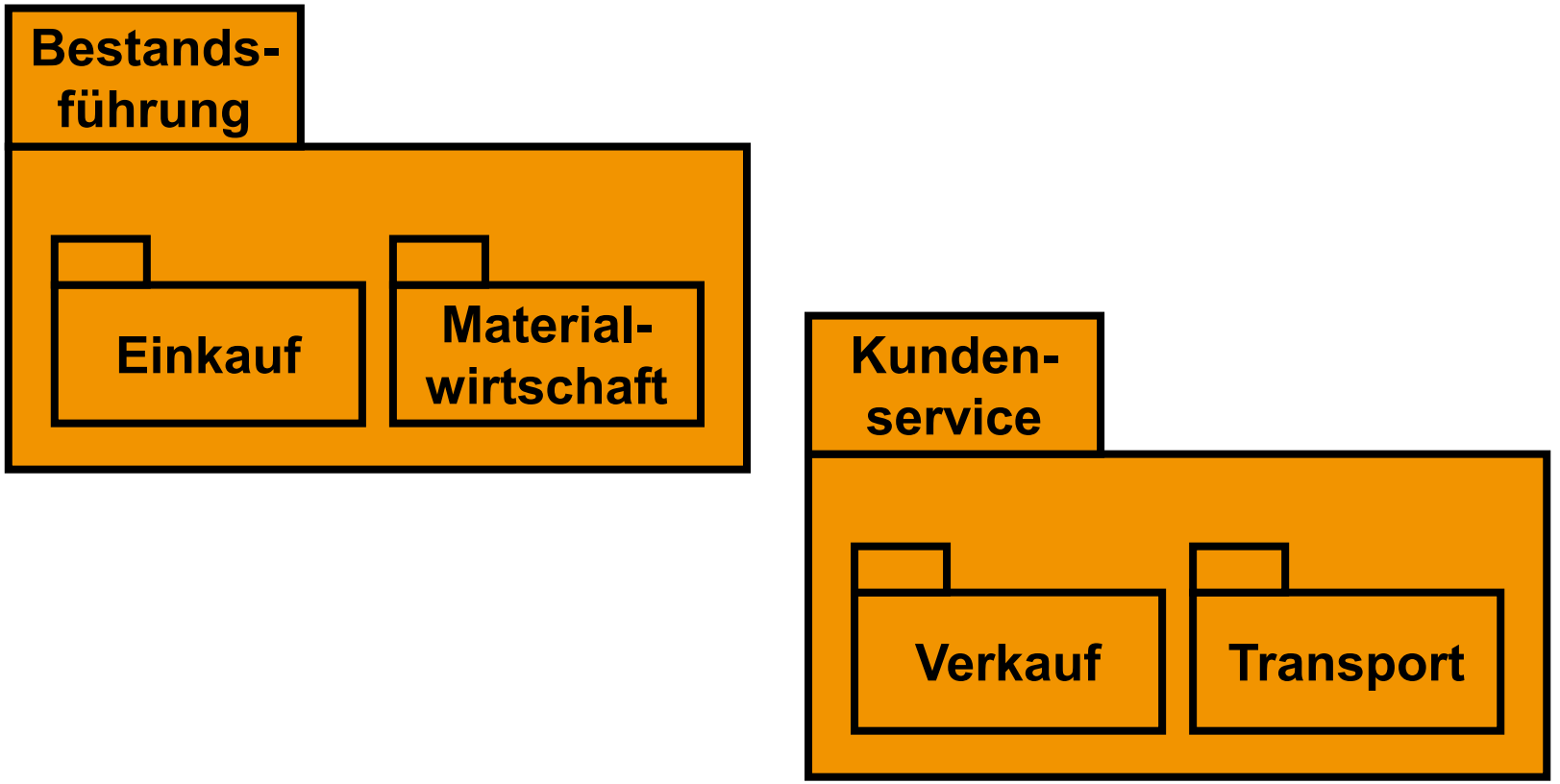
Aber:

- Redefinition von Operationen möglich

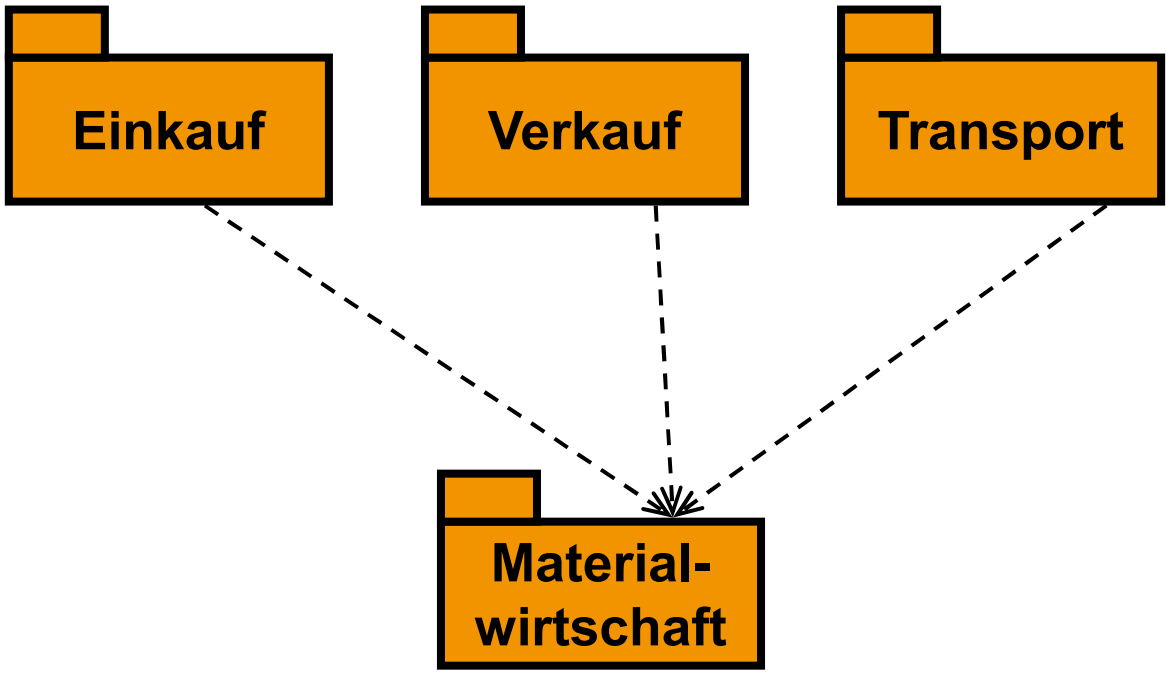
Projektverwaltung: Klassendiagramm



Paket


Bestands-
führung
Einkauf
Material-
wirtschaft
Kunden-
service
Verkauf
Transport

Paket: Abhängigkeit



OO Vorgehensmodell

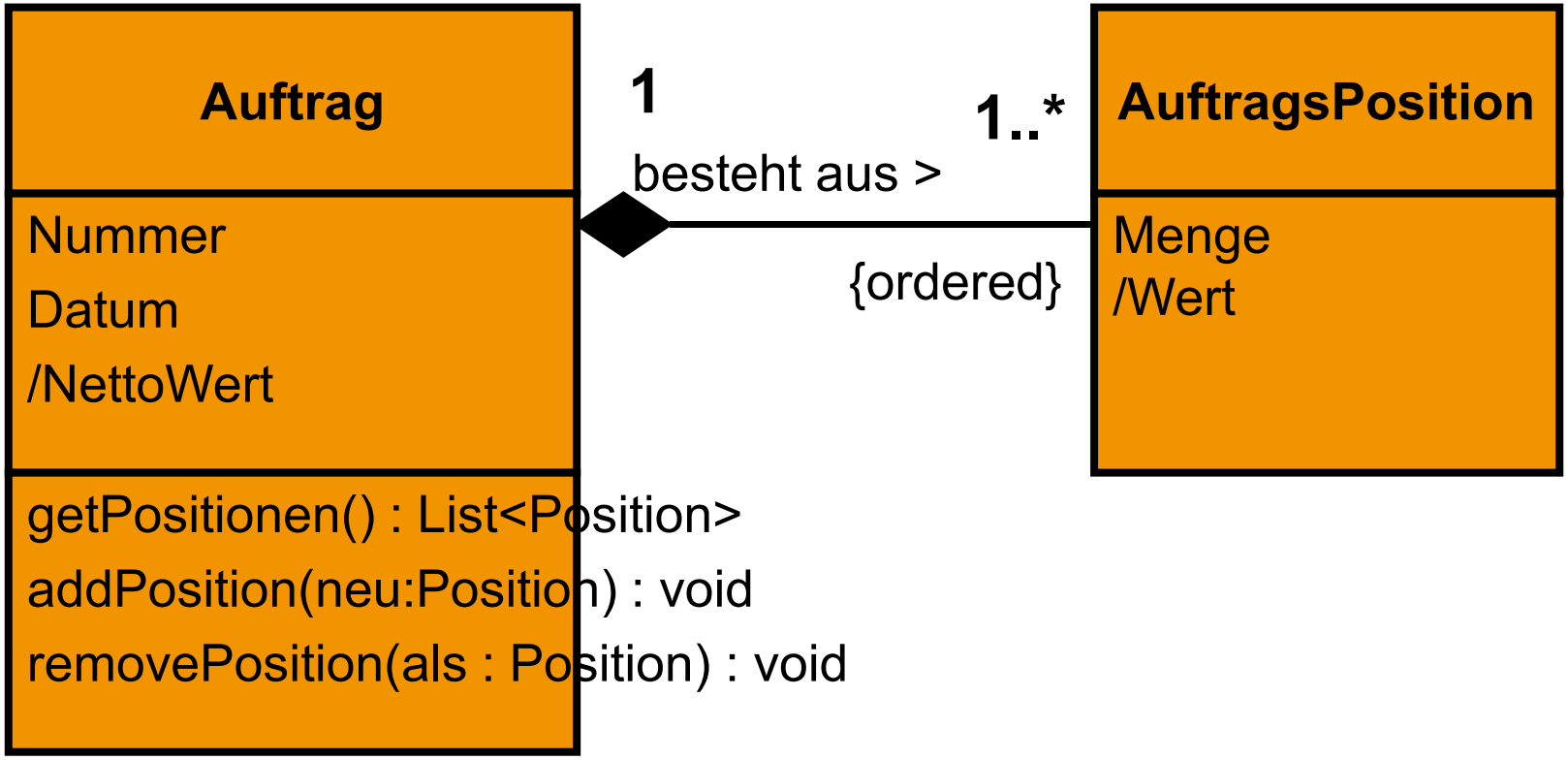
- OO Vorgehensmodell in der Analyse
  - Anforderungsanalyse
  - Problembereichsanalyse
- Dynamische Konzepte
  - Anwendungsfall (use case)
  - Aktivitätsmodell
  - Kollaborationsdiagramm
  - Sequenzdiagramm
  - Zustandsdiagramm

Entwurfsmuster

- Liste
- Exemplartyp
- Stückliste / Komponente / Kollektion
- Koordinator
- Rollen
- Historie
- Gruppe (mit Historie)

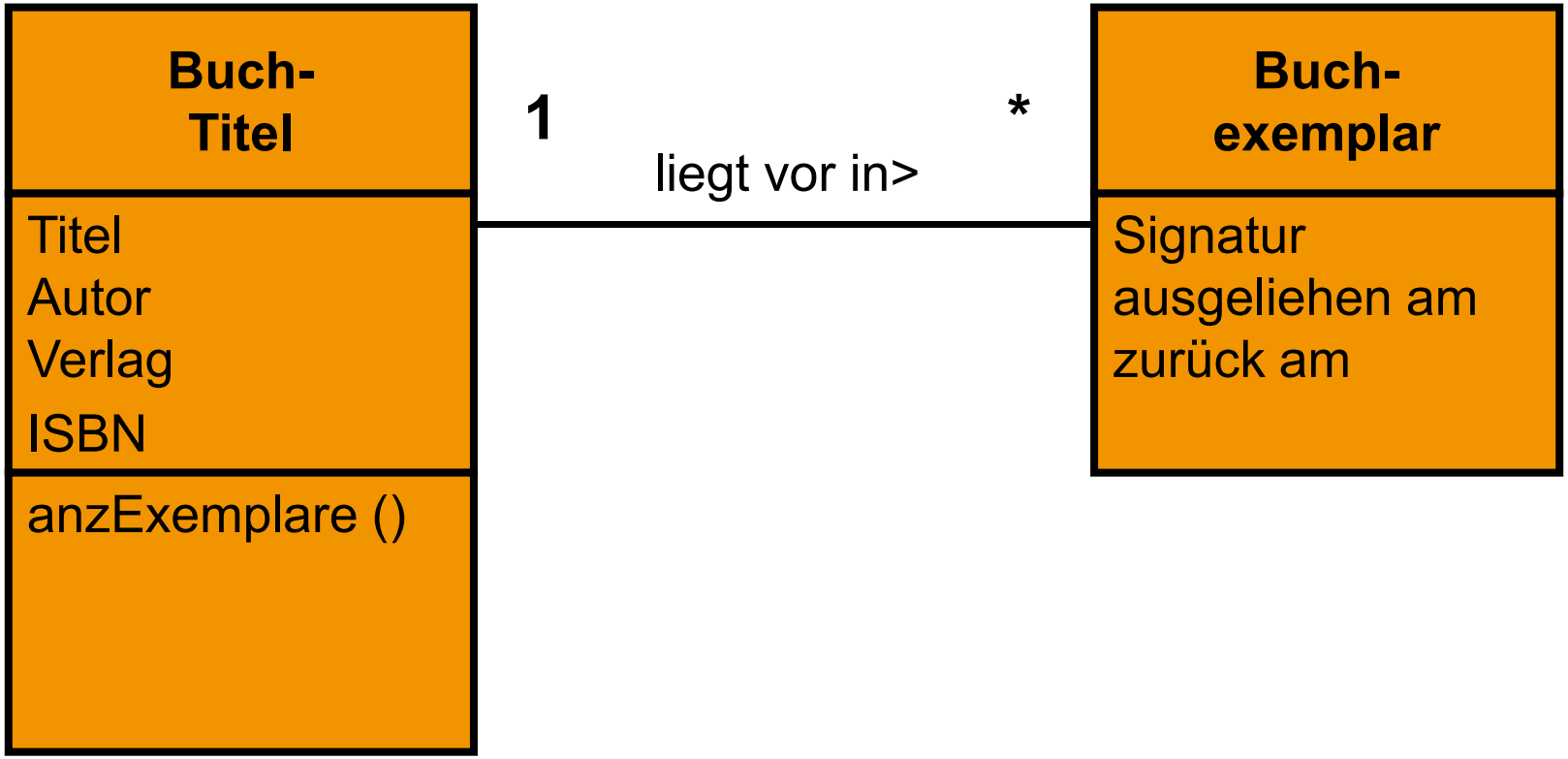
Beispiel für eine Liste

Bestellung



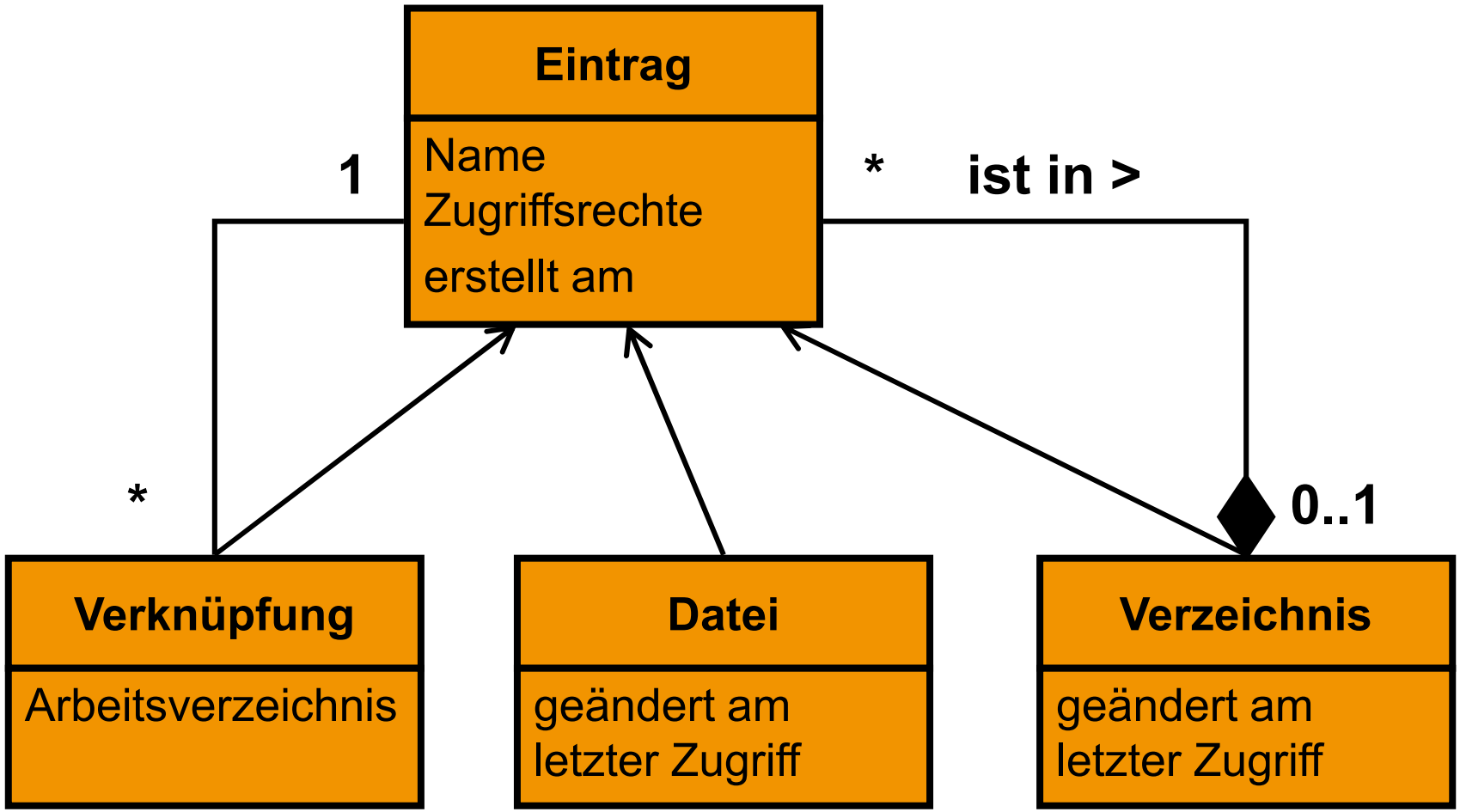
Beispiel für ein Exemplartyp

Bestellung



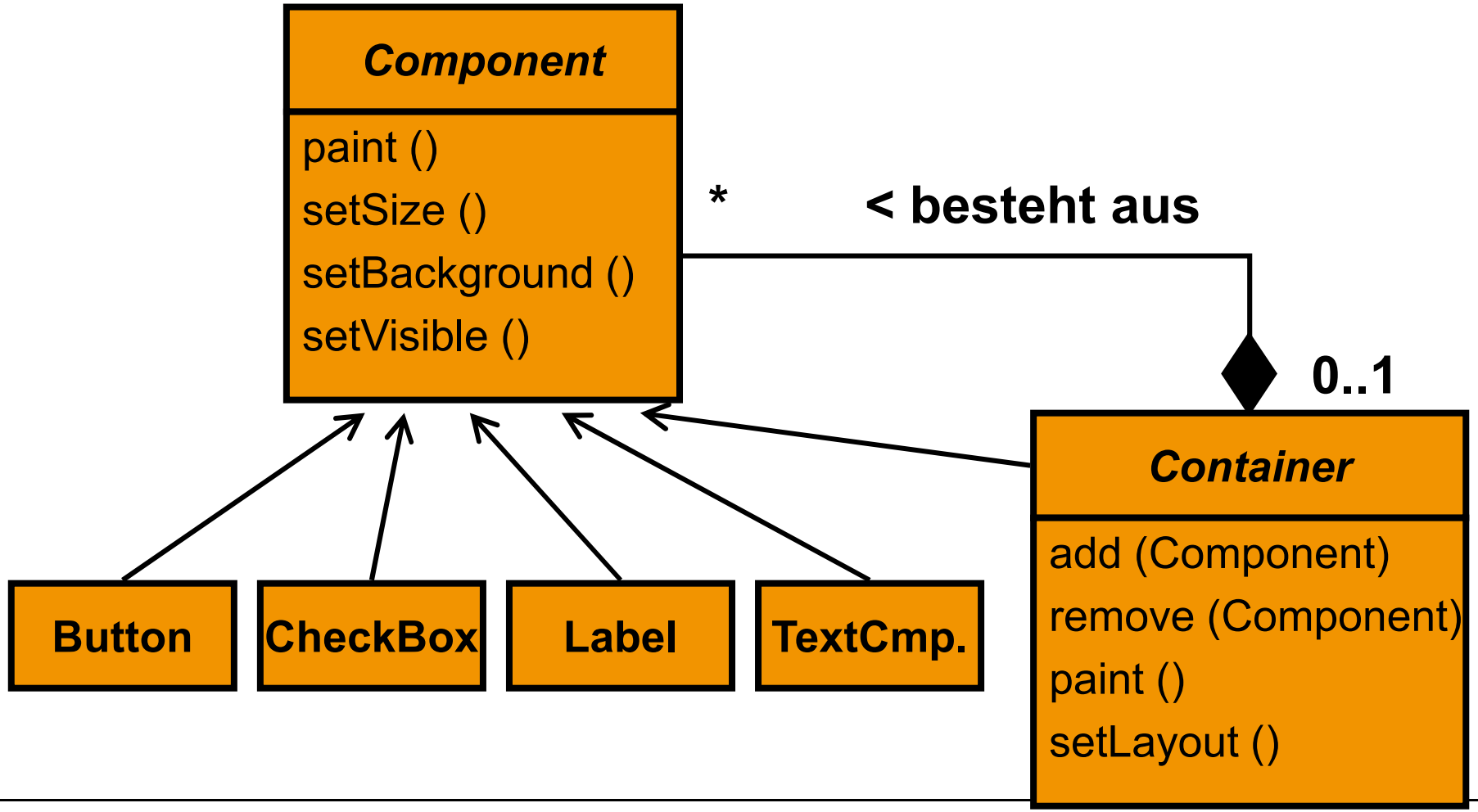
Beispiel für eine Stückliste

Dateisystem



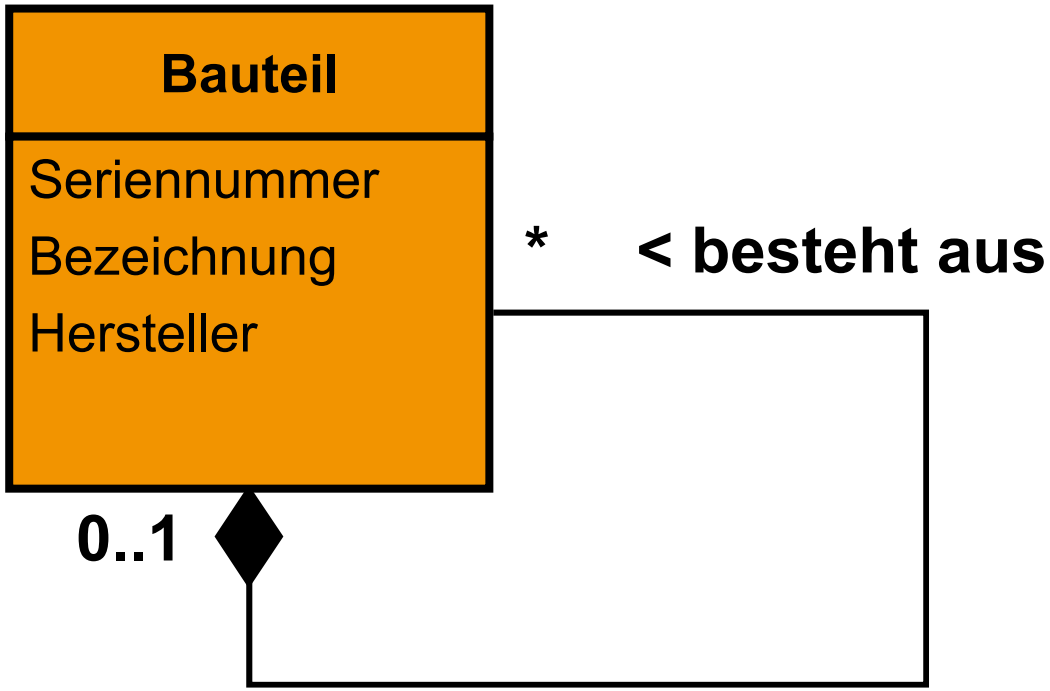
Beispiel für eine Stückliste

JAVA AWT-Komponenten



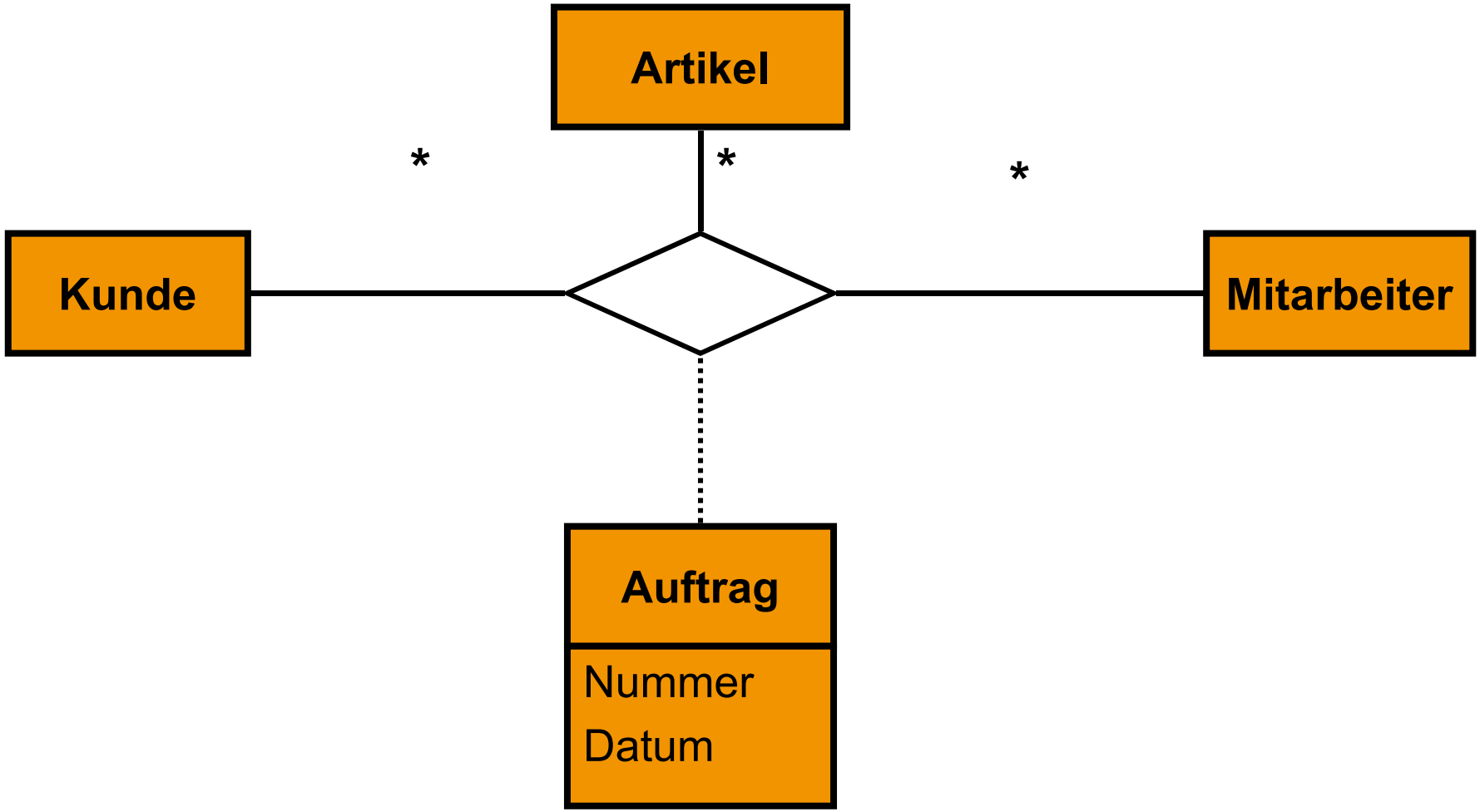
Beispiel für eine Stückliste

Gleichartige Bauteile



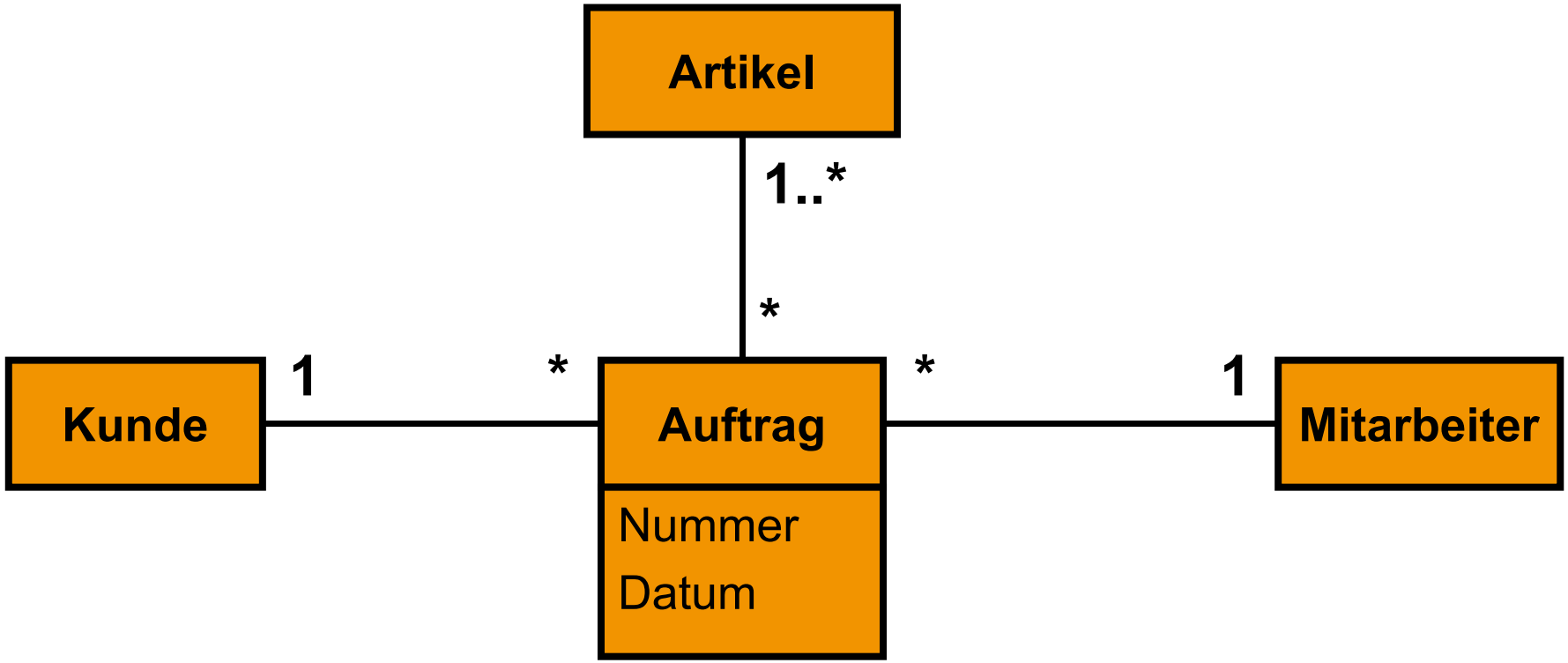
Beispiel für einen Koordinator

Bestellung (1)



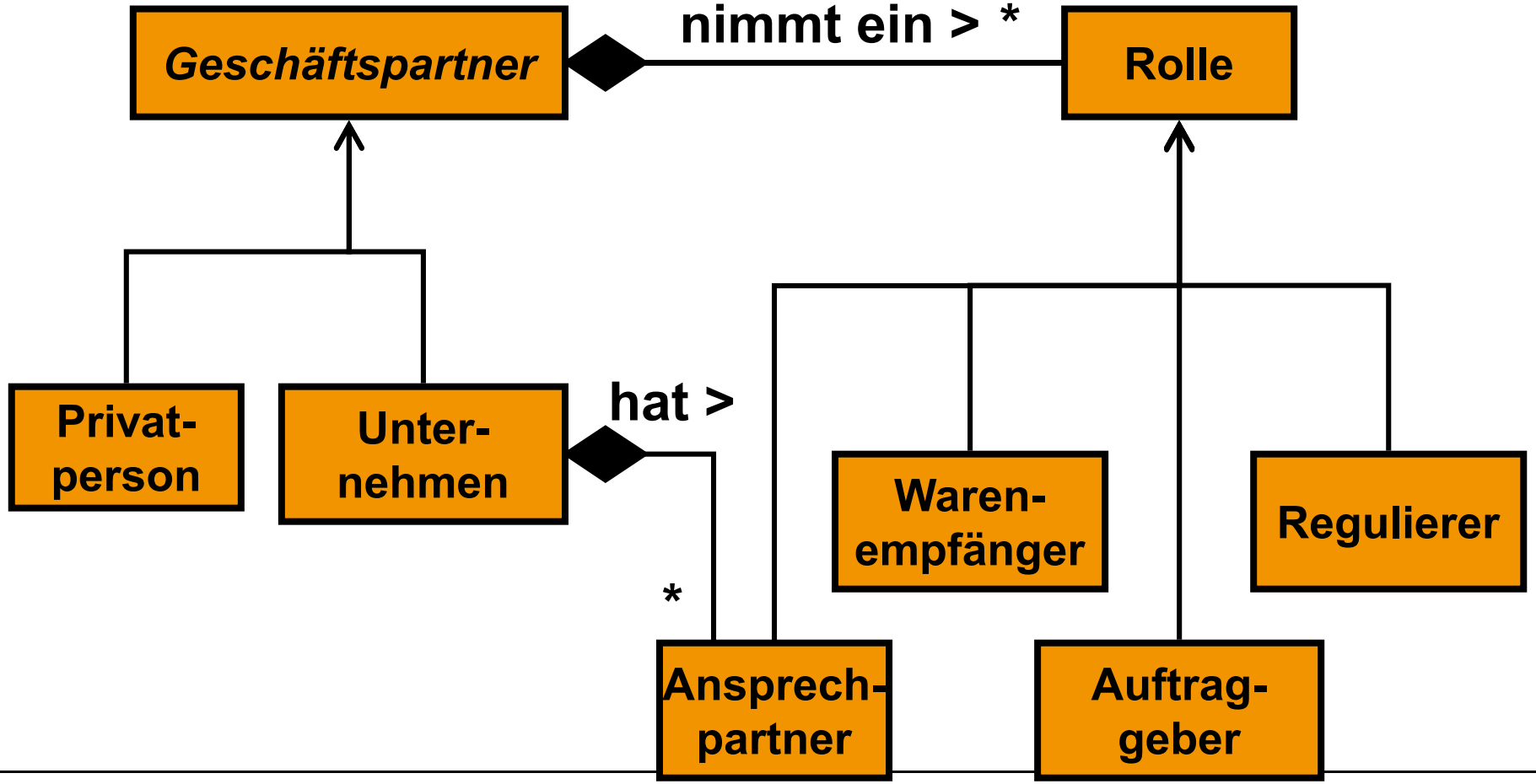
Beispiel für einen Koordinator

Bestellung (2)



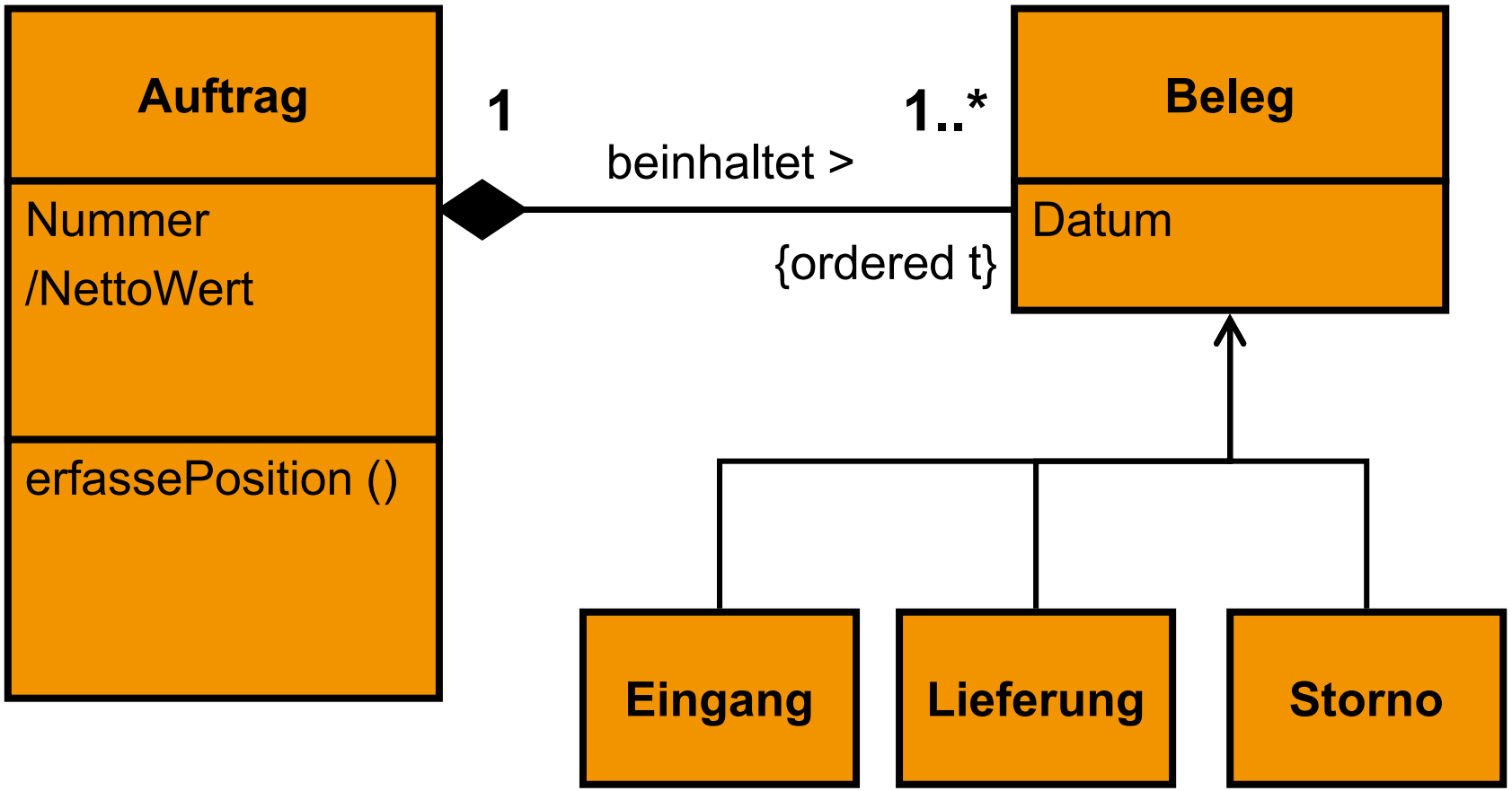
Beispiel für Akteur-Rollen

Geschäftspartner



Beispiel für Historie

Auftrag

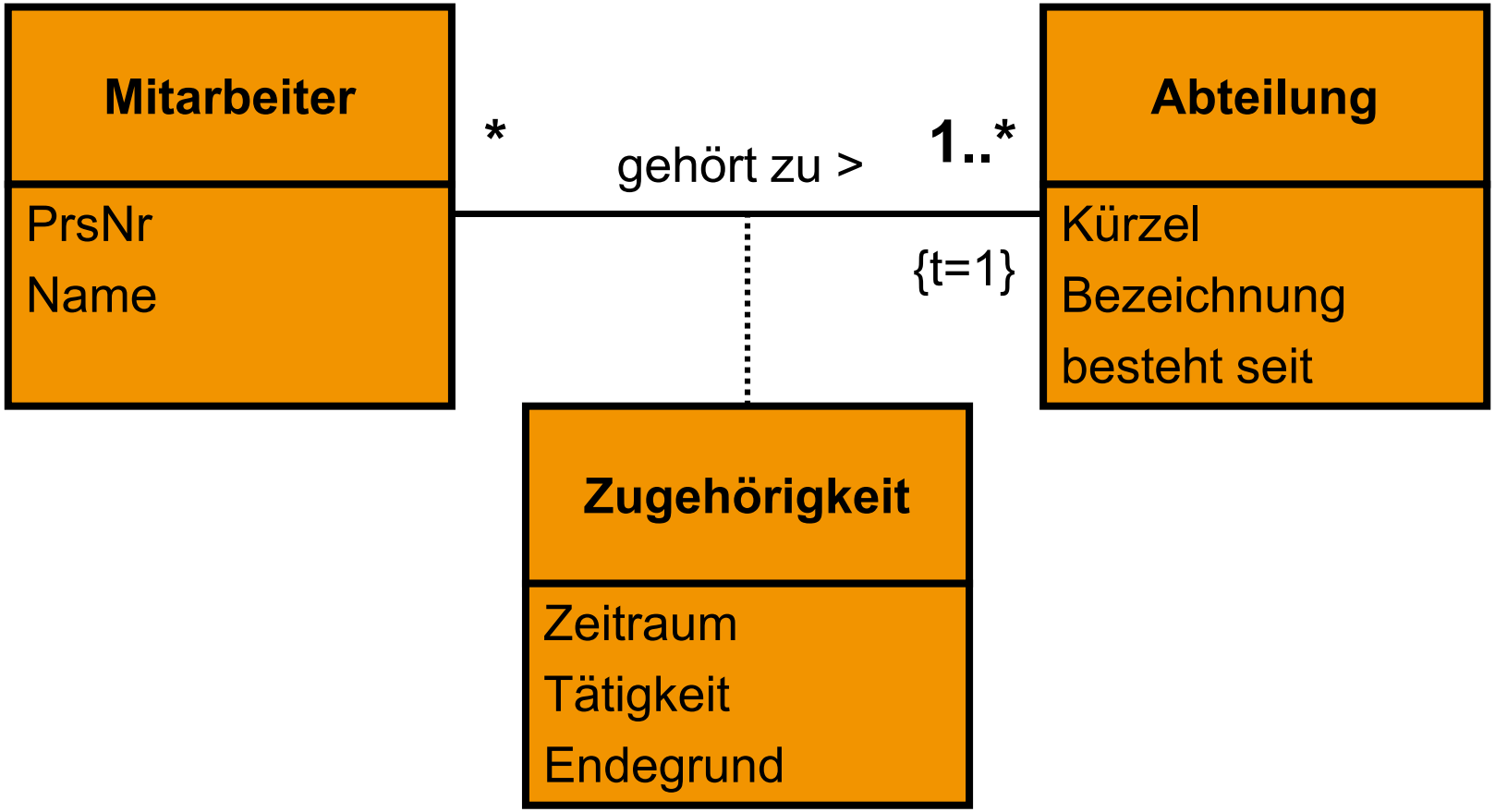


Beispiel für Gruppe

Mitarbeiter - Abteilung

Beispiel für Gruppe

Mitarbeiter - Abteilung mit Historie



Teil 3: Relationale Datenbanken und relationale Entwurfsmethodik

- Datenbank Implementierung -

Inhalt

Literatur

- Copyright © 1997, 2010, Oracle and/or its affiliates. All rights reserved.
  MySQL 5.1 Reference Manual
  http://dev.mysql.com/doc/refman/5.1/en/index.html

Skript und Übungen sind zu finden unter:
/NetStorage/DriveL/Skripten/Beham/WI-Informationssysteme/…

Was ist ein Datenbank-Managementsystem (DBMS)?

Ein **Datenbankverwaltungssystem** (DBMS, database management system) besteht aus einer Menge von gespeicherten Daten, deren Strukturbeschreibung und den zur Datenverarbeitung notwendigen Programmen.

- **Daten**
- **Schema**
- **Software**

The MySql Reference-Manual http://dev.mysql.com/doc

database management system

Ein „Datenbank-System" besteht aus:

- **Datenbasis** (Datenbankausprägung/-en)
  Die Gesamtheit aller gespeicherten Daten / Informationseinheiten

- **Datenbankschema**
  Beschreibung der Struktur der gespeicherten Daten

- **Datenbankverwaltungssystem**
  Alle Programme, die die Definition des Schemas ermöglichen oder den Zugriff und die Modifikation der Daten in der Datenbasis regeln

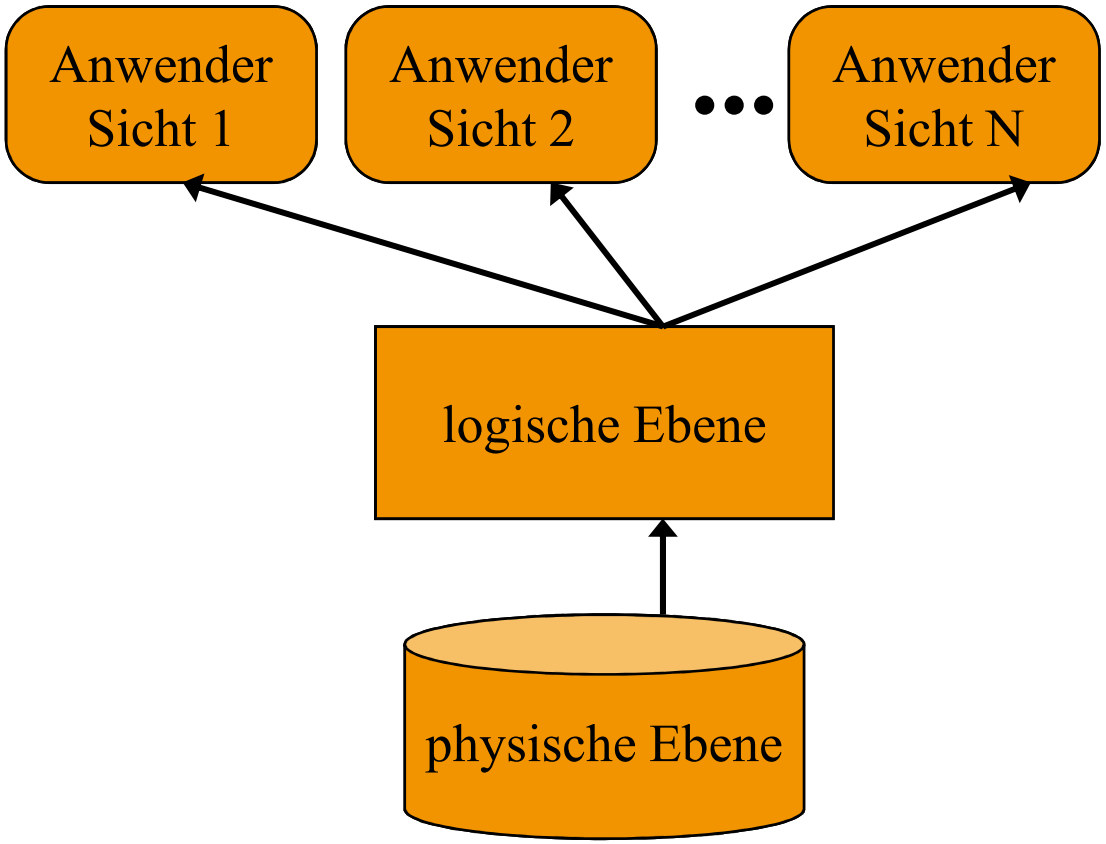
Motivation für den Einsatz eines DBMS

DBMS vermeiden folgende Probleme in der Datenhaltung:

- Redundanz und Inkonsistenz
- Beschränkte Zugriffsmöglichkeiten und fehlende Verknüpfungen
- Probleme des Mehrbenutzerbetriebs
- Verlust von Daten
- Integritätsverletzungen
- Unvollständige Transaktionen
- Sicherheitsprobleme
- Hohe Entwicklungskosten
- Verschiedenartige Schnittstellen im Datenzugriff

Abstraktion in verschiedenen Ebenen

- Unabhängigkeit der Anwendungen von der physikalischen Datenspeicherung



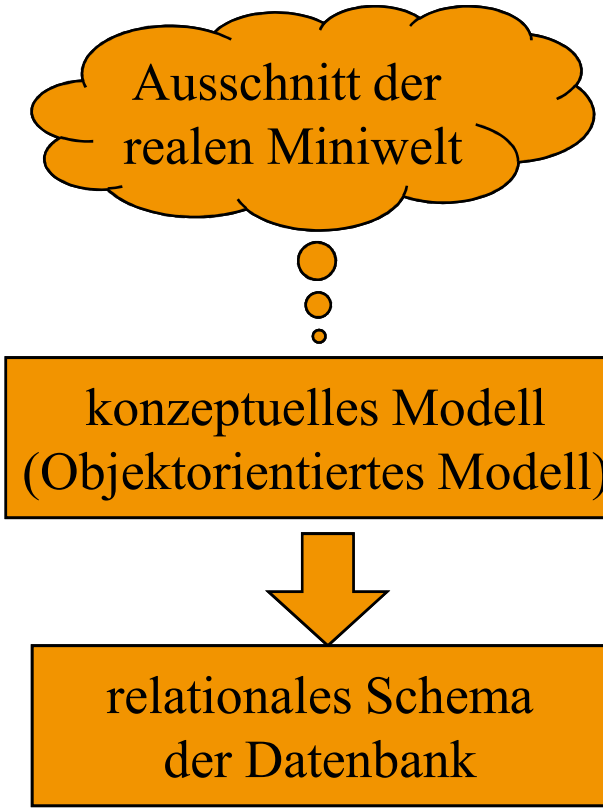
auf die speziellen Bedürfnisse der jeweiligen Anwender zugeschnitten

globales Schema (für alle Anwender gültig)

Speicherung der Daten auf entsprechenden Datenträgern (Festplatte)

Datenmodell
(Beschreibung der logischen Ebene)

- Das **Datenmodell** legt ein „computerisiertes" Informationsabbild der realen Welt (bzw. eines relevanten Ausschnitts) fest und beschreibt damit die „Struktur" der zu speichernden Daten.



manuelle / intellektuelle Leistung

Beschreibung z.B. in UML

halbautomatische Transformation

Beschreibung z.B. in SQL-DDL

- in dieser Ebene gibt es weitere Modelltypen: hierarchische, objektorientierte, deduktive, objekt-relationale

Das relationale Modell: Tabellen

- Anfang der siebziger Jahre wurde ein relationales Datenmodell konzipiert. Die Besonderheit dieses Datenmodells besteht in der mengenorientierten Verarbeitung. Es ist im Vergleich zu den bis dahin gebräuchlichen satzorientierten Modellen sehr einfach strukturiert. Es gibt im wesentlichen nur flache **Tabellen** (Relationen), deren Zeilen den Ausprägungen der Datenobjekte entsprechen. In dieser sehr einfachen - fast schon spartanischen - Struktur liegt aber wahrscheinlich der Erfolg der relationalen Datenbanktechnik begründet.
- Die in den Tabellen (Relationen) gespeicherten Daten werden durch entsprechende Operatoren ausschließlich **mengenorientiert** verknüpft und verarbeitet.

Mathematischer Formalismus (1)

- Domäne: Wertebereiche von atomaren Werten (Zahlen, Zeichenketten, Datum)
- Gegeben seien n nicht notwendigerweise unterschiedliche Domänen D1, D2, … , Dn dann ist eine Relation R definiert als:

$$R \subseteq D_1 \times D_2 \times \cdots \times D_n$$

- **Beispiel**:

  Telefonbuch $\subseteq$ string $\times$ string $\times$ integer

Telefonbuch		
Name	Straße	TelNr
Mickey Mouse	Main Str.	4711
Mini Mouse	Broadway	0815
Donald Duck	Highway	32168
…	…	…

Name der Relation

Attribute der Relation

Ausprägungen der Relation (Menge von Tupeln)

- **Schreibweise kurz**:

  Telefonbuch : { [Name : string, Straße : string, TelNr : integer] }

Mathematischer Formalismus (2)

- Achtung: Name der Relation (Tabelle) für **Ausprägung** und **Schema**
- **Schema** = Menge der Attribute (z.B. Name, Straße, TelNr), allgemein:

  $\mathbf{sch}(R) = \{A_1, A_2, \ldots, A_n\}$
- **Domäne** eines Attributes:

  $D_i = \mathbf{dom}(A_i), \quad i = 1, \ldots, n$
- **Ausprägung** R

  $R \subseteq \mathbf{dom}(A_1) \times \mathbf{dom}(A_2) \times \ldots \times \mathbf{dom}(A_n)$
- allgemeines Muster:

  R: { [$\underline{A_1} : D_1, \ \underline{A_2} : D_2, \ldots, A_n : D_n$] }
- Unterstreichung kennzeichnet den Primärschlüssel der Relation, z.B.:

  Telefonbuch : { [Name: string, Straße: string, <u>TelNr : integer</u>] }

Datenbankentwurf

Der **konzeptuell** „saubere“ Entwurf sollte Voraussetzung aller Datenbankanwendungen sein. An dieser Stelle sei eindringlich davor gewarnt, den Datenbankentwurf unvollständig oder nicht mit der notwendigen Systematik durchzuführen. Derartige Versäumnisse rächen sich in späteren Phasen des Datenbankeinsatzes und sind dann oftmals nicht mehr zu korrigieren, weil viele andere Entwurfsentscheidungen davon abhängig sind.

The MySql Reference-Manual http://dev.mysql.com/doc

Entity-Relationship-Modell = OO Modell

Bestandteile eines ER-Modells sind:

- **Objekte = Entität**,
  die zu abstrakten Objekttypen (Klassen) zusammengefaßt sind. Folgende Begriffe werden im folgenden nicht immer streng unterschieden:
  - Klasse = Objekttyp, ist die Beschreibung einer Menge von ähnlichen Objekten (z.B. Personen)
  - Objekt = Ausprägung, ist ein konkretes Element dieser Menge (z.B. Josef Maurer, geb. am 11.11.1970, Tel. 0987/654321)
- **Attribute = Tabellenspalte,**
  die diese Objekte beschreiben und identifizieren
- **Beziehungen = Relation** zwischen den Objekten,
  die ebenfalls zu Beziehungstypen abstrahiert werden
- evtl. **Rollen** von Objekten,
  die diese in den verschiedenen Beziehungen übernehmen

Beispiel: Projektverwaltung

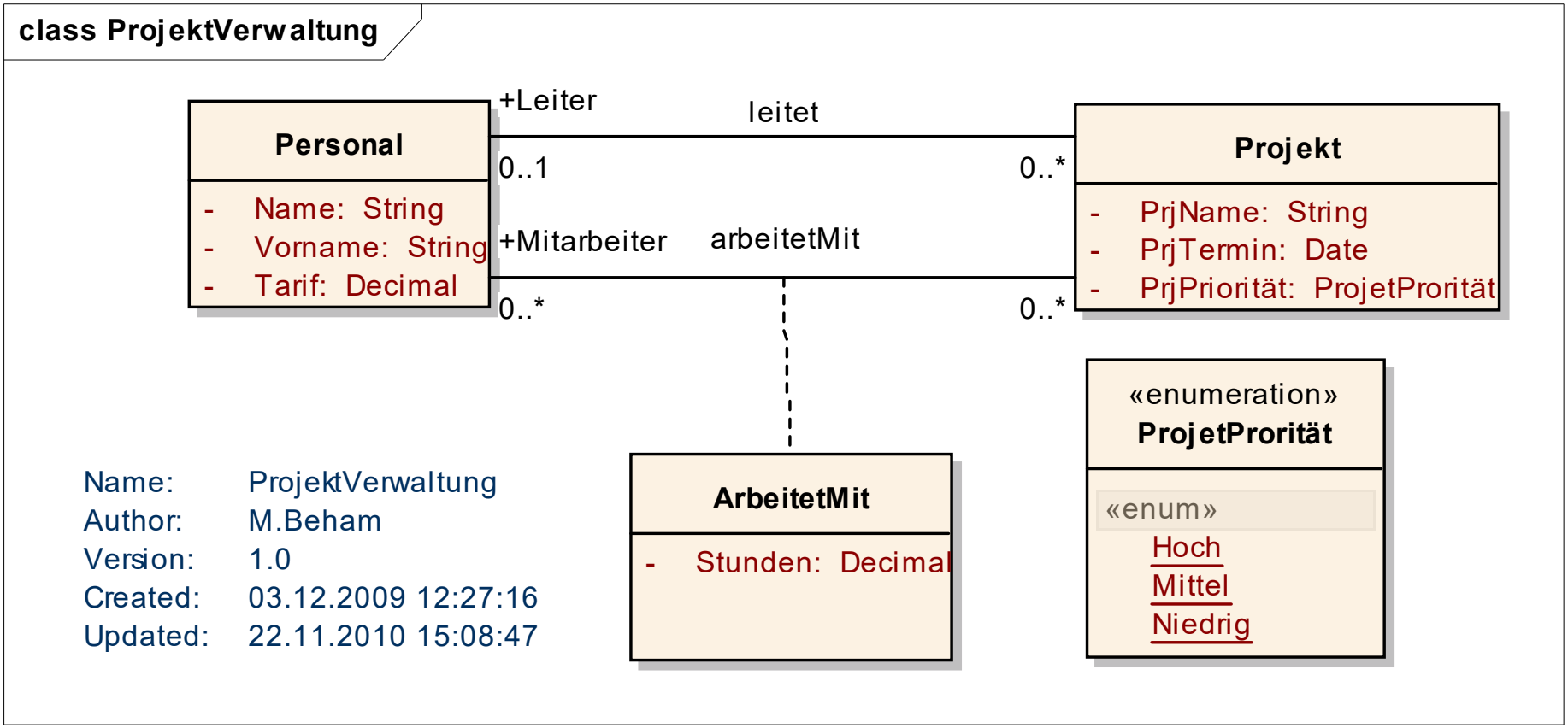
Eine (vereinfachtes) Projekt / Mitarbeiter Verwaltungssystem soll folgende Aufgaben übernehmen:

- **Personalverwaltung**
  Mitarbeiter anlegen unter Angabe von Namen, Stundenlohn, ...
- **Auftragsverwaltung**
  Anlegen von Aufträgen (Projekten) von Firmen unter Angabe von Terminen und möglichst mit der Benennung eines Projektleiters.
- **Zuordnung** von Mitarbeitern zu einem Projekt
  Der Projektleiter kann sein jeweiliges Projektteam zusammenstellen.
- **Kosten-**, Terminkontrolle
  Die Projekte sollen bezüglich Aufwand und Termintreue verfolgt werden können

Beispiel: Projektverwaltung
Konzeptueller OO Entwurf

- Objekt: **Mitarbeiter**
  Anzahl: ca. 500
  Eistenz: aktiv, mehrere Jahre
  - Attribut: **PrsId**
    Definiertheit: 100%
    Identifizierend: ja, eindeutig
    Datentyp: ganzzahlige Numerierung
    Range: ca. 100 - 1000
  - Attribut: **Name**
    …
  - Attribut: **Vorname**
  - …
- Objekt: **Projekt**
- Objekt: **Firma**
- Beziehung: <Mitarbeiter> **leitet** <Projekt>
  Rollen: Mitarbeiter als Leiter, Projekt als „sein Projekt"
  Anzahl: wie Projekte
  - Attribut: ?
- Beziehung: <Firma> **beauftragt** <Projekt>
- Beziehung: <Mitarbeiter> **arbeitet mit in** <Projekt>

Beispiel: Projektverwaltung
Konzeptuelles Klassendiagramm



Umsetzung des konzeptuellen Schemas (UML) in ein relationales Schema

- **Klasse** → Relation (Tabelle)
- **Objekt** → Entity (Zeile einer Tabelle)
- **Attribut** → Spalte einer Tabelle bzw. Domäne
- **Methode** → /
- **Assoziation** → Relation (Tabelle)
- **Objekt-Identität** → Primärschlüssel (unique constraints)
- **Cardinalitäten** → referential / unique constraints

Umsetzung flacher Klassen

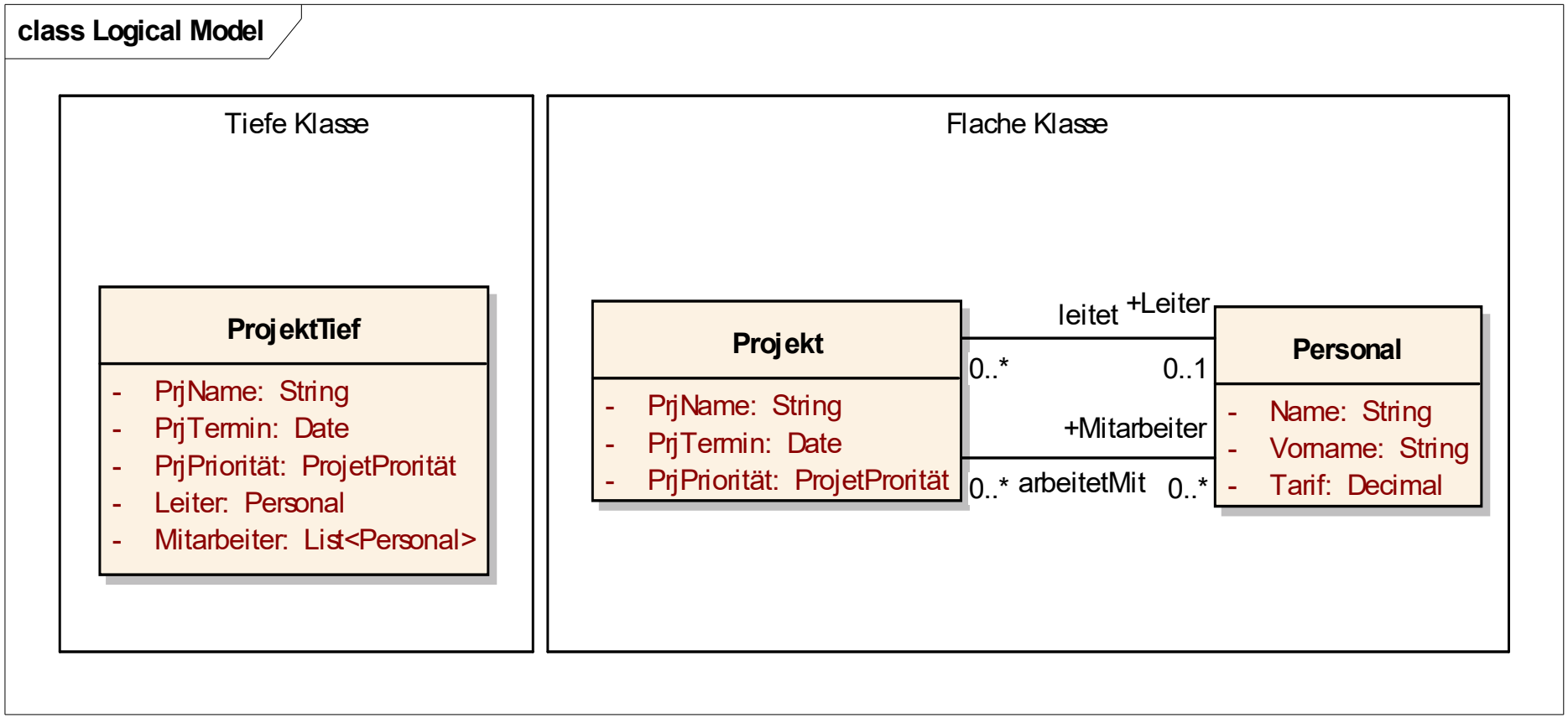
- „**Flache**" Klassen lassen sich direkt in Form von Relationen (Tabellen) darstellen.

  Flach: Alle Attribute sind von einfachen Datentypen (String, Date, Enum, Int, …)
  und nicht selbst wieder strukturiert (wie z.B. Klasse, List<?>, Set<?>)

- Den elementaren konzeptuellen Datentypen entsprechen physische Realisierungen des jeweiligen Datenbanksystems: SQL Typen
  Z. B.: `String` → `varchar(20)`

- Falls eine Klasse nicht flach ist, müssen alle strukturierten Attribute in Form von Assoziationen umgesetzt werden.
  Siehe nachfolgendes Beispiel.

Beispiel:
"Flachmachen" einer Klasse

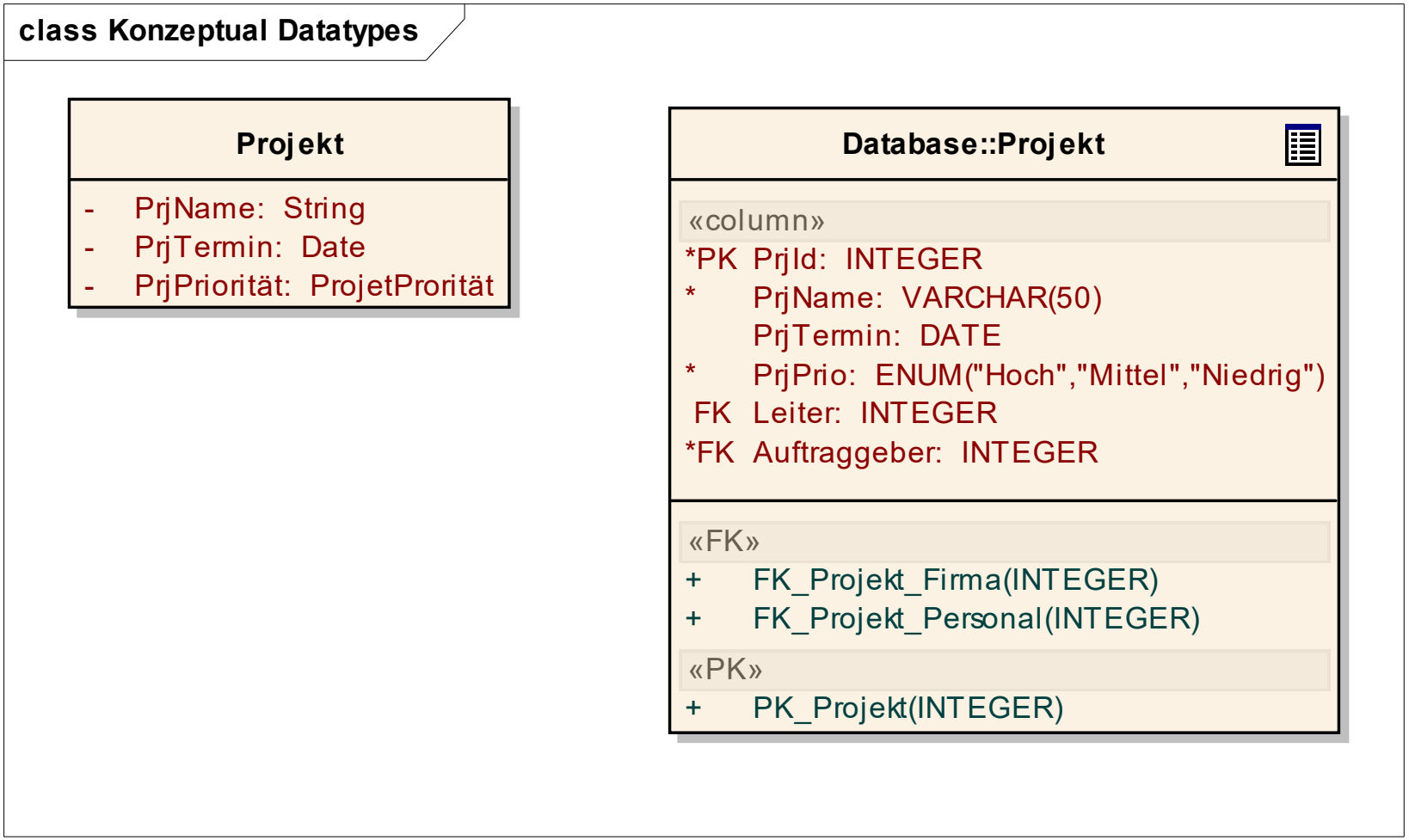


Umsetzung der elementaren Datentypen in SQL Datentypen

Die wichtigsten Datentypen am Beispiel von MySQL:

char(N)	Zeichenkette fester Länge, z.B. char (3)
varchar(N)	Zeichenkette variabler Länge mit Obergrenze, z.B. varchar (254)
decimal(N,M)	allg. Festkommazahl (auch **numeric** genannt) mit der Angabe: (ges. Stellenzahl, Nachkommastellen), z.B. decimal (5,2)
integer, ...	ganze Zahl und innerhalb des Wertebereichs exakt (allg. 32 Bit), auch mit **unsigned** Vorsatz
real, float, double	Fließkommadarstellung
date	Datum in der Form: 'YYYY-MM-DD' (Y2K: YYYY 0000 - 9999)
time	Zeit in der Form: 'hh.mm.ss'
timestamp, ...	Zeitpunkt in µs Auflösung; 'YYYY-MM-DD hh.mm.ss.uuuuuu'
bool	Boolean, true oder false
enum('a','b'...)	Enum Datentyp mit symbolischen Konstanten 'a', 'b', ...
clob	character large object, bis zu 2 GB Zeichenkette (Text)
blob	binary large object, bis zu 2 GB beliebige Daten

Beispiel:
Umsetzung der Datentypen

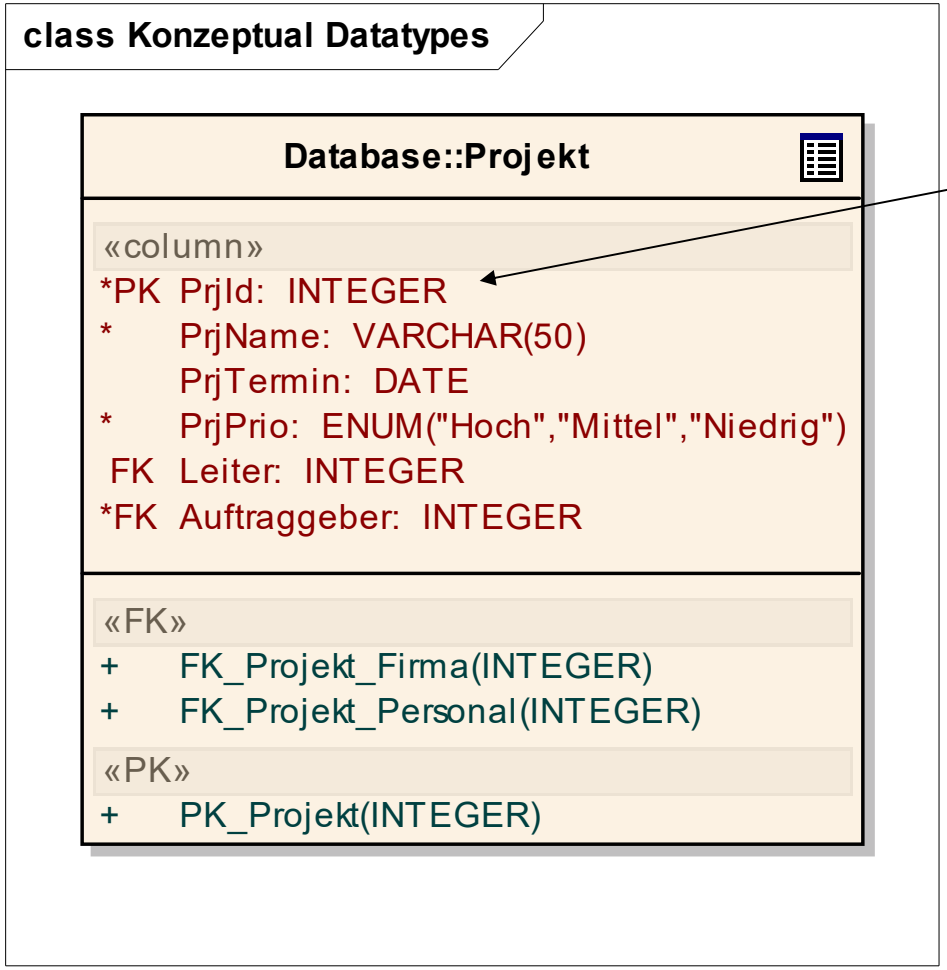


Identifikation von Objekten (primary key)

Innerhalb einer Tabelle (Relation) **müssen** alle Einträge anhand festgelegter Attribute eindeutig identifizierbar sein: **primary key**

- Eine minimale Menge von Attributen, deren Werte die Ausprägung einer Entität innerhalb der Menge aller Entitäten seines Typs eindeutig identifiziert, nennt man Schlüssel.
- Normalerweise darf es keine zwei identischen Ausprägungen eines Objekttyps innerhalb einer Menge geben. (Mengen von unterscheidbaren Elementen)
- Falls derartige Werte nicht existieren, werden oftmals künstliche Identifikatoren geschaffen
  (Personal-, Matrikel-, Konto-, Lieferschein-, Bestell-, … -Nummer)
- Gibt es mehrere Kandidaten von Attributen zur Bildung eines Schlüssels, wählt man einen als Primärschlüssel aus.

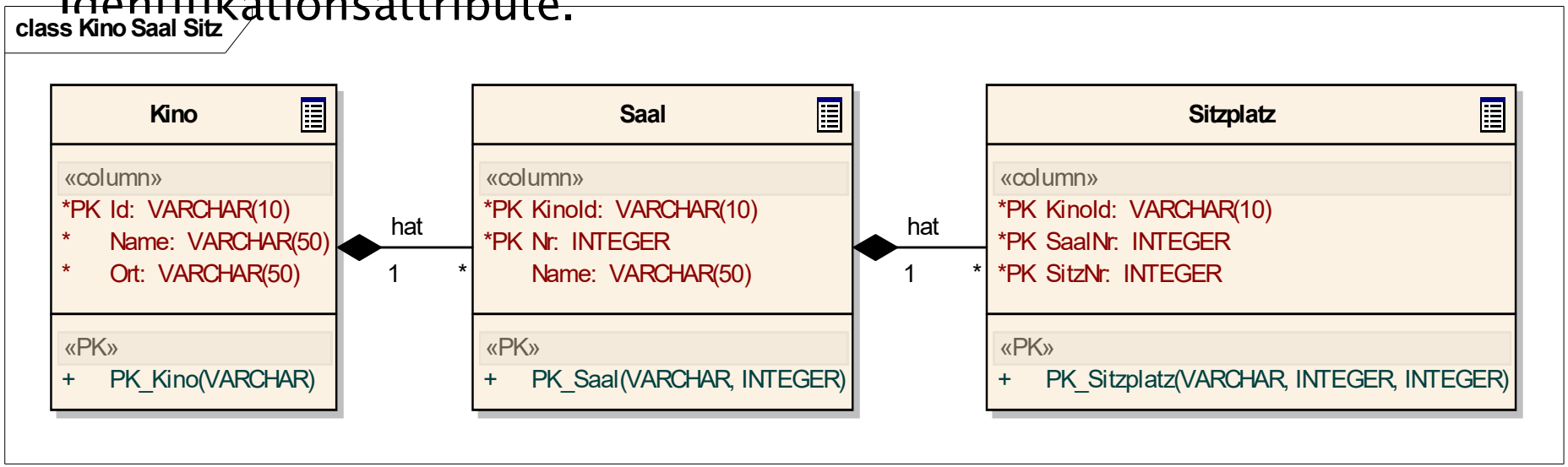
Beispiel:
primary key, Projektnummer



PK: primary key
* : mandatory
= auto_increment (optional)

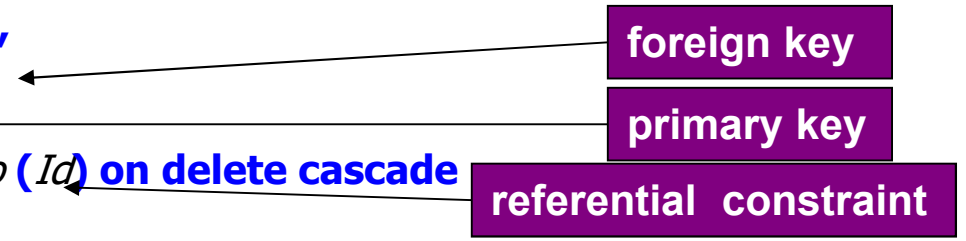
Komposition
Existenzabhängige Objekte

- sind in ihrer Existenz von einem anderen, übergeordneten Objekt abhängig und
- nur in Kombination mit dem Schlüssel des übergeordneten Objekts eindeutig identifizierbar.
- **Regel**: Der Primärschlüssel des abhängigen Objekts setzt sich zusammen aus allen Attributen des Primärschlüssels des übergeordneten Objekts plus zusätzlicher eigener Identifikationsattribute.



Komposition
Existenzabhängige Objekte
in SQL - DDL

```
create table Kino
  (         Id                  varchar(10) not null,
            Name                varchar(50) not null,
            Ort                 varchar(50) not null,
            primary key (Id),
  );
create table Saal
  (         KinoId              varchar(10) not null,
            Nr                  integer not null,
            Name                varchar(50),
            primary key (KinoId, Nr),
            foreign key (KinoId) references Kino (Id) on delete cascade
  );
create table Sitz
  (         KinoId              varchar(10) not null,
            SaalNr              integer not null,
            SitzNr              integer not null,
            primary key (KinoId, SaalNr, SitzNr),
            foreign key (KinoId, SaalNr) references Saal (KinoId, Nr) on delete cascade
  );
```



Umsetzung aller Assoziationen als eigenständige Relation mit Fremdschlüssel

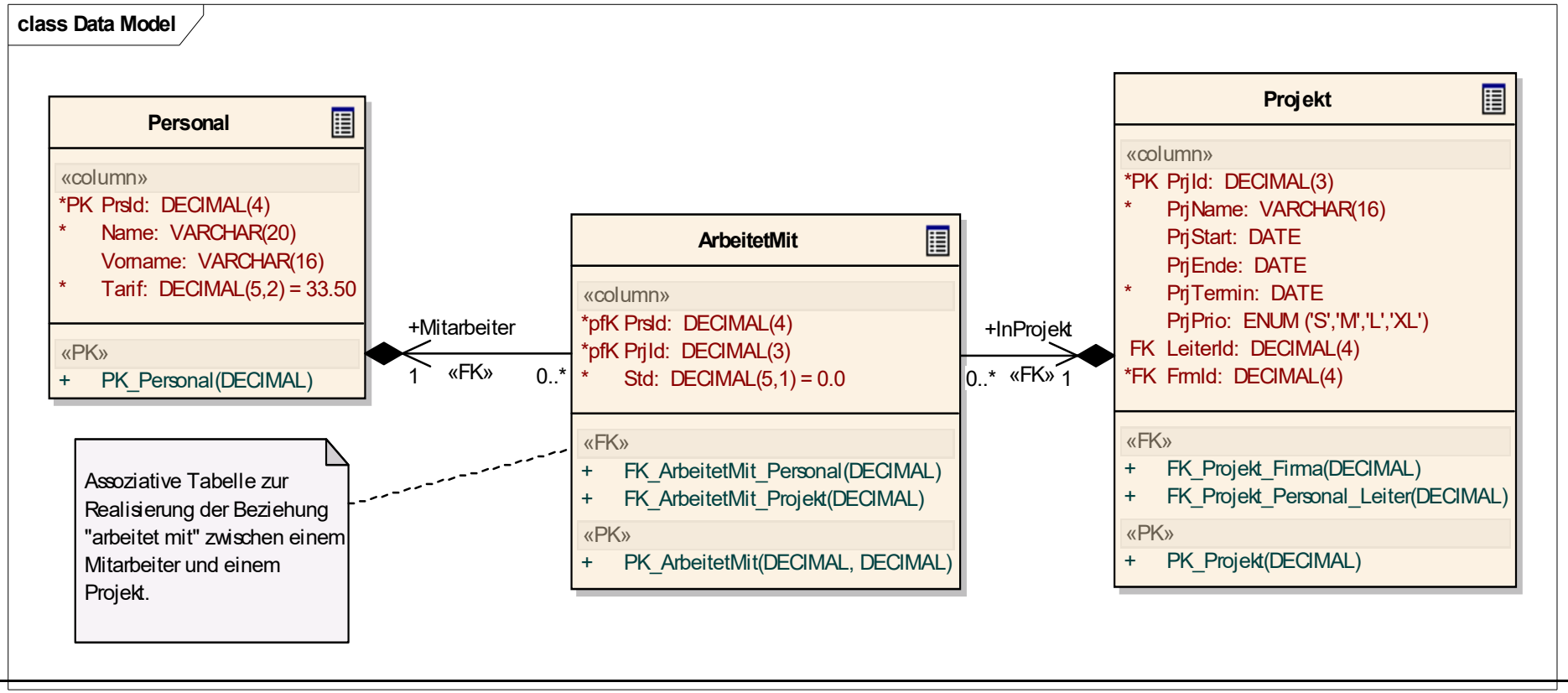
- **Start**: Jede einzelne Assoziation wird ebenfalls als Relation definiert. Einige dieser Relationen können dann später wieder eliminiert werden (Schemavereinfachung).
- **Grundprinzip**: Die n-stellige „Beziehungs-Relation" enthält alle Schlüsselattribute der beteiligten Objekte E1, E2, … En und zusätzlich die der Beziehung zugeordneten Attribute:

$$R \quad : \quad \{[\underbrace{A_{11},\ldots,A_{1k_1}}_{\text{Schlüssel von } E_1},\underbrace{A_{21},\ldots,A_{2k_2}}_{\text{Schlüssel von } E_2},\ldots,\underbrace{A_{n1},\ldots,A_{nk_n}}_{\text{Schlüssel von } E_n},\underbrace{A_{R1},\ldots,A_{Rk_R}}_{\text{Attribute von } R}]\}$$

- **Beispiel:** Projektverwaltung die Assoziation "arbeitet mit" → Tabelle: ArbeitetMit
  Personal: PrsId: integer → ArbeitetMit: PrsId:integer als Fremdschlüssel
  Projekt: PrjId: integer → ArbeitetMit: PrjId:integer als Fremdschlüssel arbeitet
  zusätzlich → ArbeitetMit: Stunden: decimal(5,2)
- **Frage:** Was sind die Primärschlüssel der Beziehungs-Relation ArbeitetMit
- **Rolle:** Neuer Name für ein Attribut in der Beziehungs-Relation möglich.

Beispiel: Grundprinzip – jede Assoziation wird durch eigene Tabelle realisiert

- Fremdschlüssel (foreign key) in **ArbeitetMit**:
  PrsId ist der PK aus Personal
  PrjId ist der PK aus Projekt



Umsetzung der Assoziationen Charakterisierung von Relationen als partielle Funktionen

Ein Beziehungstyp *R* zwischen den Entitytypen E_1, E_2, … E_n kann als Relation im mathematischen Sinn beschrieben werden:

$$R \subseteq E_1 \times E_2 \times \cdots \times E_n$$

$$\text{geordnetes Tupel}: (e_1, e_2, \cdots, e_n) \in R \text{ mit } e_i \in E_i$$

$$\text{Rolle als Name für die Elemente des Tupels}: (\text{Rolle}_1 : e_1, \cdots)$$

- *n* ist der Grad der Beziehung *R*. In der Praxis sind binäre (*n* = 2) am häufigsten
- Beispiel für eine binäre Beziehung:
  Beziehung zwischen Personen in Form eines Mietvertrags

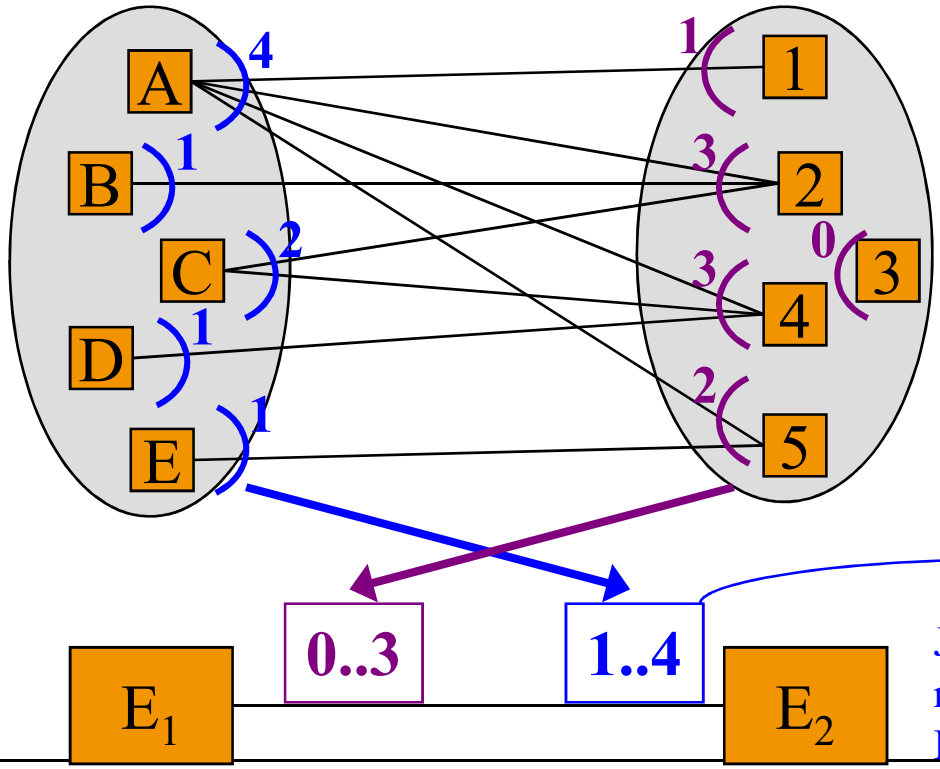
$$R<\text{mietet}> \subseteq E_1<\text{Person}> \times E_2<\text{Person}>$$

$$(\text{Mieter}: e_1, \text{Vermieter}: e_2) \in R<\text{mietet}>$$

Dadurch ist unmißverständlich festgelegt, daß das erste Element des Tupels der Mieter vom zweiten Element dem Vermieter ist.

Cardinalität (Funktionalität) der Assoziation

- Die (min, max)-Notation gibt für jeden an einer Beziehung beteiligten Objekttyp an, daß jede einzelne Entität dieses Typs mindestens min-mal und höchstens max-mal in der Beziehung steht (Beachte die Position der Cardinalität in UML!!!)



Jedem Element aus E_1 sind mindestens 1 und maximal 4 Element(e) aus E_2 zugeordnet

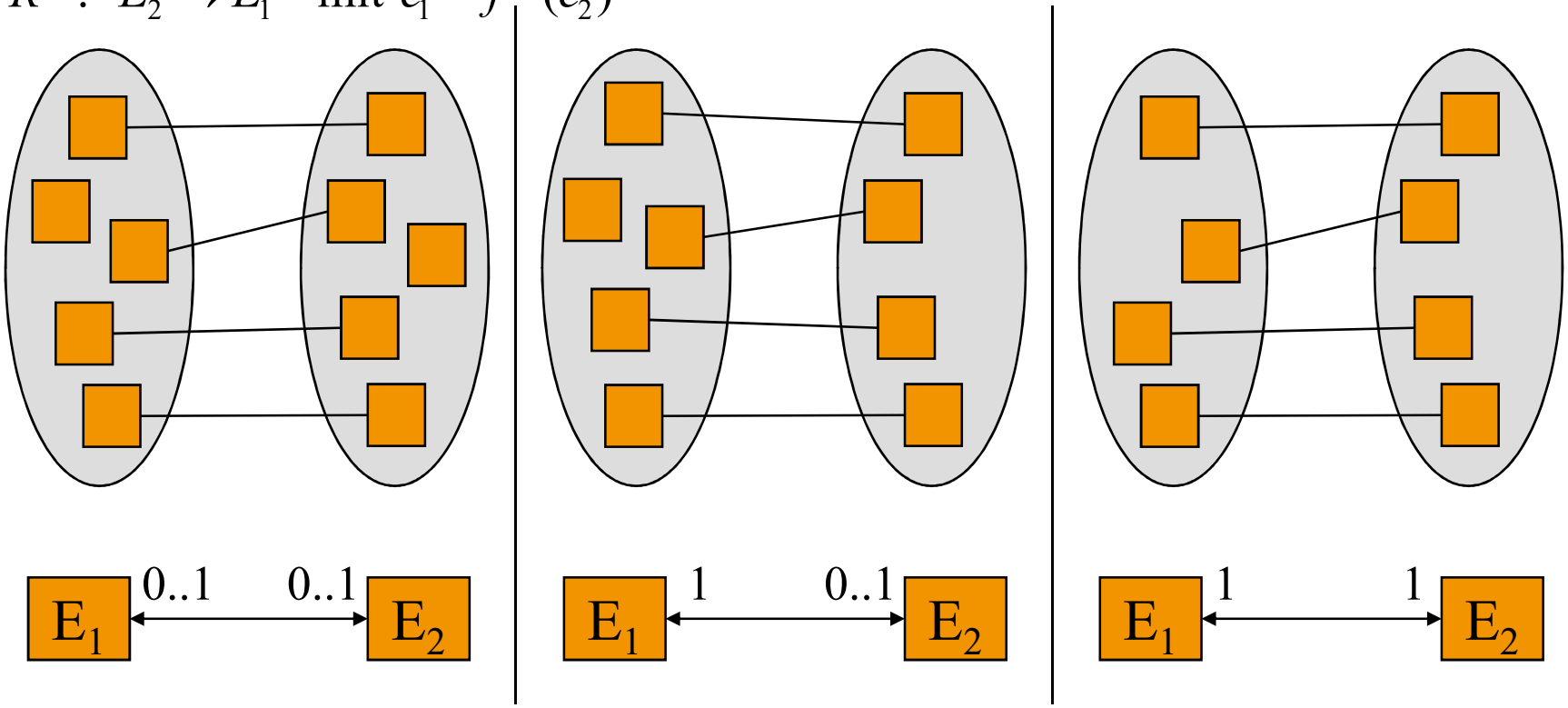
(e_1,	e_2)
A	1
A	2
A	4
A	5
B	2
C	2
C	4
D	4
E	5
	(3)
min 1	*min* 0
max 4	*max* 3

1:1 Assoziation

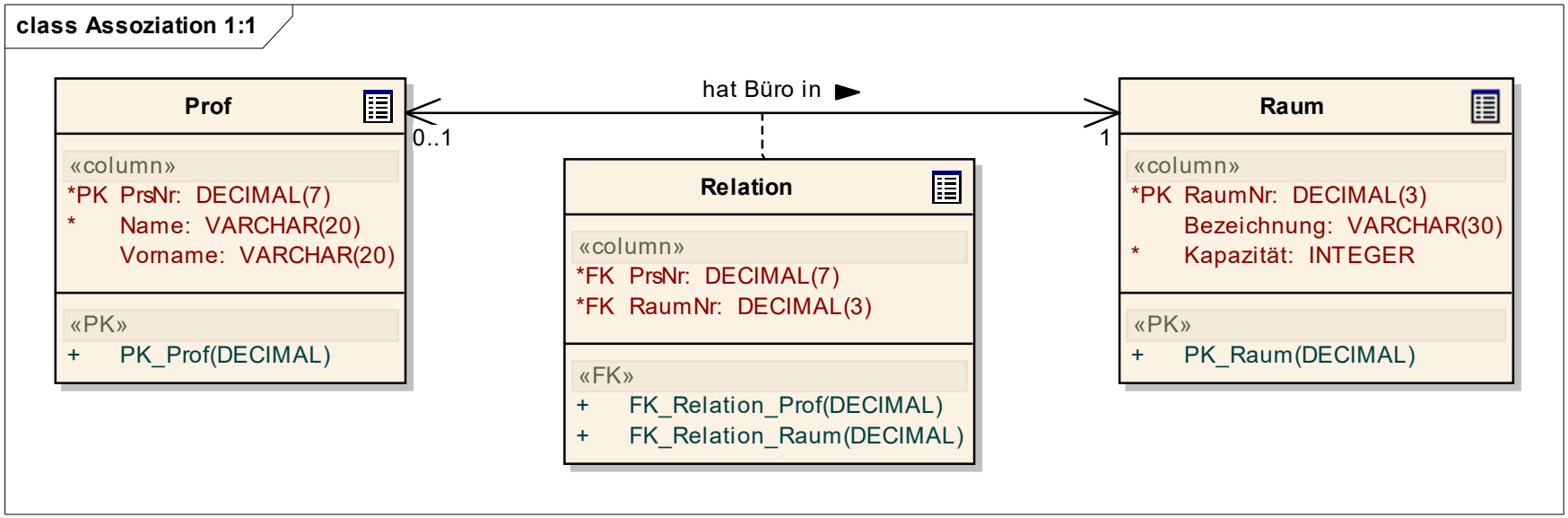
- Eine 1:1 Assoziation ist eine umkehrbare partielle (nicht alle Elemente des Definitionsbereichs werden zugeordnet) Funktion.

$$R: \quad E_1 \rightarrow E_2 \quad \text{mit } e_2 = f(e_1)$$

$$R^{-1}: \quad E_2 \rightarrow E_1 \quad \text{mit } e_1 = f^{-1}(e_2)$$



Beispiel:
Umsetzung 1:1 Assoziation



Prof		
PrsNr	Name	Vorname
111	'Beham'	'Manfred'
130	'Jaeger'	'Magnus'
177	'Magerl'	'Franz'
299	'Müller'	'Ulrich'

hatBüroIn	
PrsNr	RaumNr
111	142
130	045
177	105
299	043

unique unique

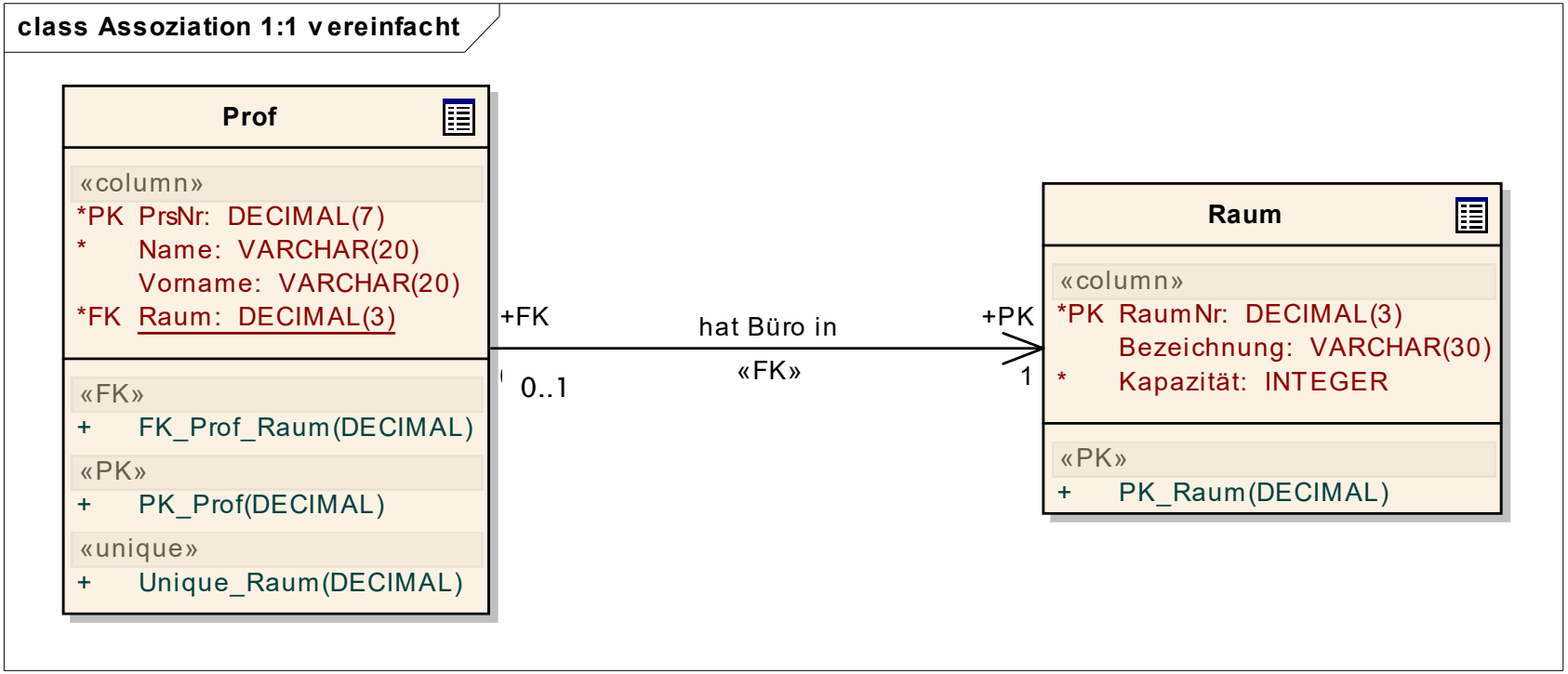
Raum		
RaumNr	Bez.	Kapazität
142	-	0
140	'ZF'	45
120	'Conrad'	60
045	-	0

Umsetzung von Assoziationen: Vereinfachung des Schemas durch Zusammenfassen von Tabellen

Regel: Relationen mit gleichem Primärschlüssel zusammenfassen!

- Diese Regel kann im allgemeinen nur auf 1:N oder 1:1 Beziehungen angewandt werden.
- Beispiel einer **1:1-Beziehung:** (1) - (0..1)
  Prof – hat Büro in – Raum mit zwei Möglichkeiten der Zusammenfassung:
  - Zusammenfassung: Prof – hat Büro in (gleicher PK), besser (!)
    Neuer 'muss' Fremdschlüssel (**not null**) in Prof verweist immer auf Raum und muss zusätzlich **unique** sein
  - Zusammenfassung: hat Büro in – Raum (gleicher PK)
    Neuer 'kann' Fremdschlüssel (**null**) in Raum verweist optional auf Prof und muss ebenfalls zusätzlich **unique** sein

Umsetzung 1:1 Assoziation vereinfacht



Prof			
PrsNr	Name	Vorname	Raum
111	'Beham'	'Manfred'	142
130	'Jaeger'	'Magnus'	045

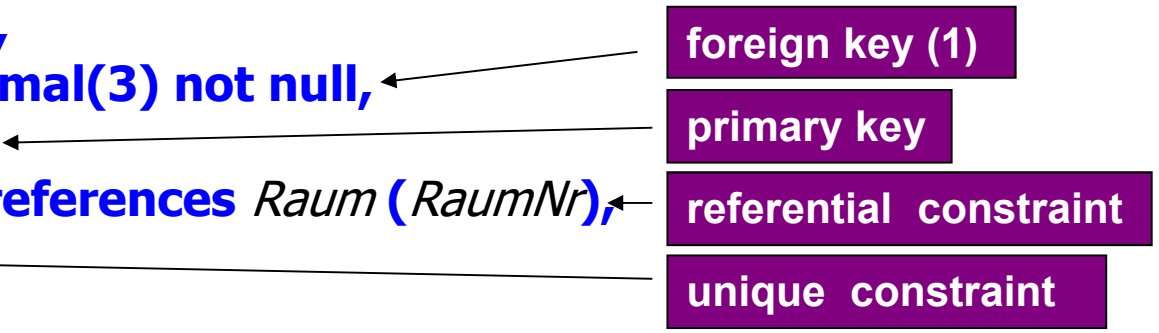
unique

Raum		
RaumNr	Bez.	Kapazität
142	-	0
140	'ZF'	45
120	'Conrad'	60
045	-	0

Beispiel in SQL-DDL: Umsetzung 1:1 Assoziation

- Definition der Tabellen in SQL-DDL (MySQL) für Prof – Raum

```
create table Raum
  (       RaumNr          decimal(3) not null,
          Bezeichnung     varchar(20),
          Kapazitaet      integer not null default 0,
          primary key (RaumNr)
  );
create table Prof
  (       PrsNr           decimal(7) not null auto_increment,
          Name            varchar(20) not null,
          Vornamevarchar(20),
          Buero           decimal(3) not null,
          primary key (PrsNr),
          foreign key (Buero) references Raum (RaumNr),
          unique (Buero),
  );
```

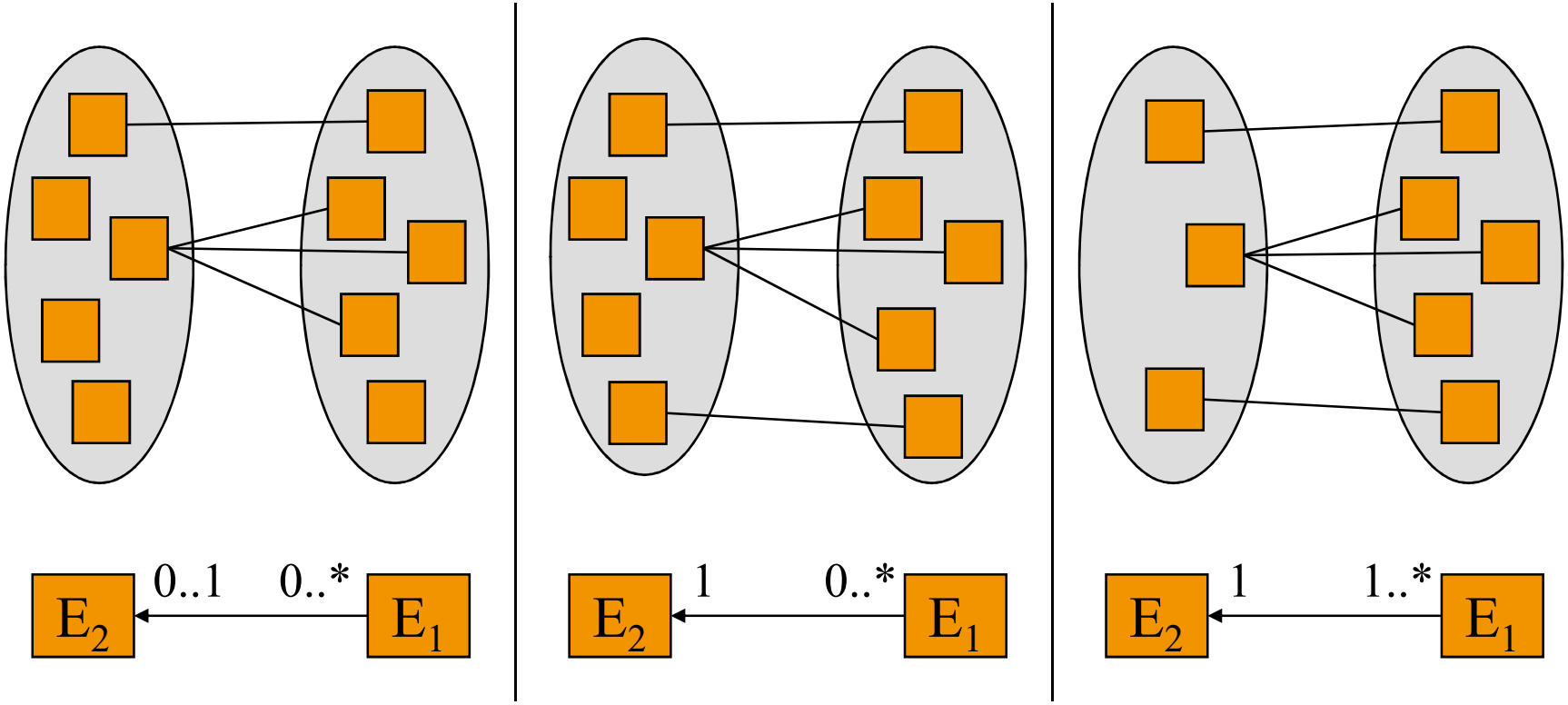


1:N Beziehungen

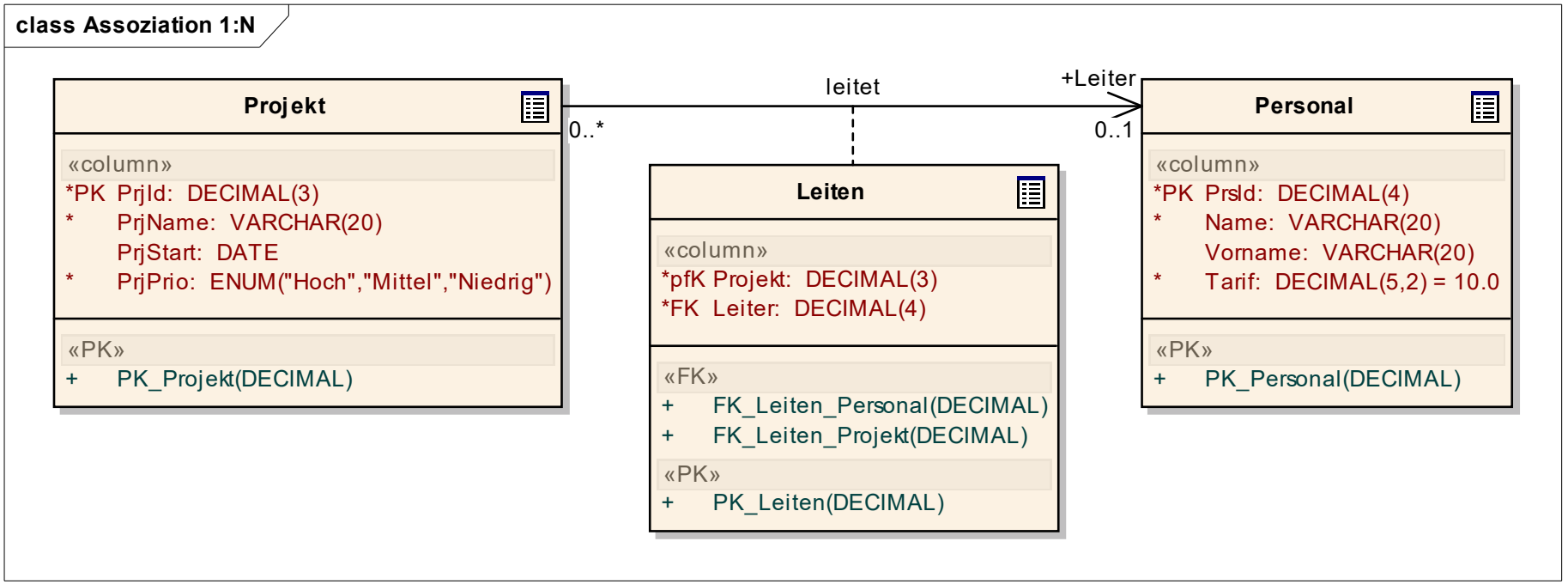
- Eine 1:N Beziehung ist eine nicht umkehrbare partielle Funktion.

$$R: \quad E_1 \rightarrow E_2 \quad \text{mit } e_2 = f(e_1)$$

Die Funktion f^{-1} existiert nicht



Umsetzung 1:N Assoziation



Projekt			
PrjId	PrjName	PrjStart	PrjPrio
111	'Datenbank'	2010.11.05	Hoch
130	'Hausnetz'	2011.01.20	Mittel
177	'Labor'	-	Hoch
299	'VirtualW'	-	Hoch

Leiten	
PrjId	PrsId
111	1324
130	1216
299	1216

unique

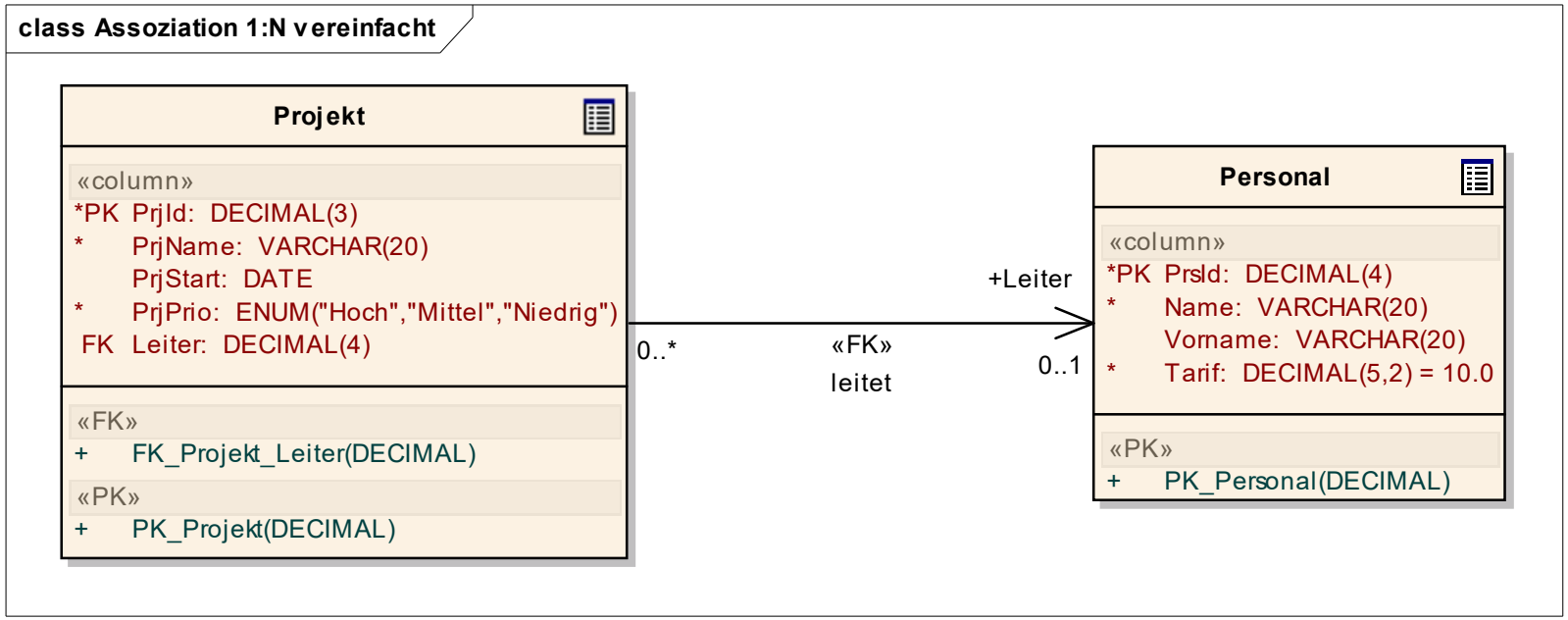
Personal			
PrsId	Name	Vorname	Tarif
1216	'Jelzin'	'Boris'	45.75
1324	'Fischer'	'Joschka'	68.99
1380	'Schmidt'	'Monika'	55.64
1413	'Ratlos'	'Rudi'	60.66
1530	'Camen'	'Bert'	120.00
1666	'Sorglos'	'Susi'	35.77

Umsetzung von Assoziationen: Vereinfachung des Schemas durch Zusammenfassen von Tabellen

Regel: Relationen mit gleichem Primärschlüssel zusammenfassen!

- Diese Regel kann bei einer 1:N nur auf die 0..* Seite angewandt werden.
- Beispiel einer **1:N-Beziehung:** (0..*) - (0..1)
  Projekt – Leiten – Personal hat nur eine Möglichkeit der Zusammenfassung:
  - Zusammenfassung: Projekt – Leiten (gleicher PK)
    Fremdschlüssel PrjId (**null**) in Projekt verweist optional auf einen Mitarbeiter aus Personal

Umsetzung 1:N Assoziation vereinfacht



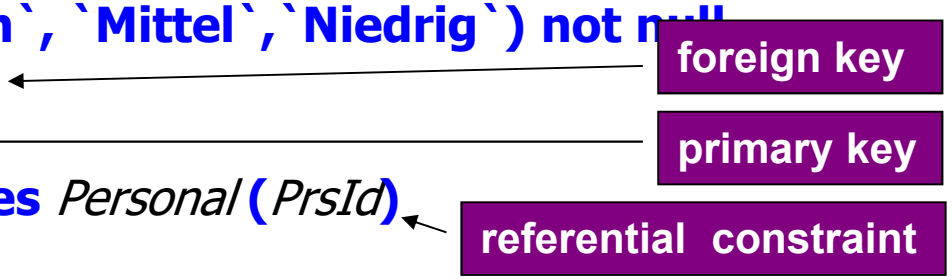
Projekt				
PrjId	PrjName	PrjStart	PrjPrio	***Leiter***
111	'Datenbank'	2010.11.05	Hoch	1324
130	'Hausnetz'	2011.01.20	Mittel	1216
177	'Labor'	-	Hoch	1216
299	'VirtualW'	-	Hoch	-

Personal			
PrsId	Name	Vorname	Tarif
1216	'Jelzin'	'Boris'	45.75
1324	'Fischer'	'Joschka'	68.99
1380	'Schmidt'	'Monika'	55.64
1413	'Ratlos'	'Rudi'	60.66
1530	'Camen'	'Bert'	120.00
1666	'Sorglos'	'Susi'	35.77

Beispiel in SQL-DDL: Umsetzung 1:N Assoziation

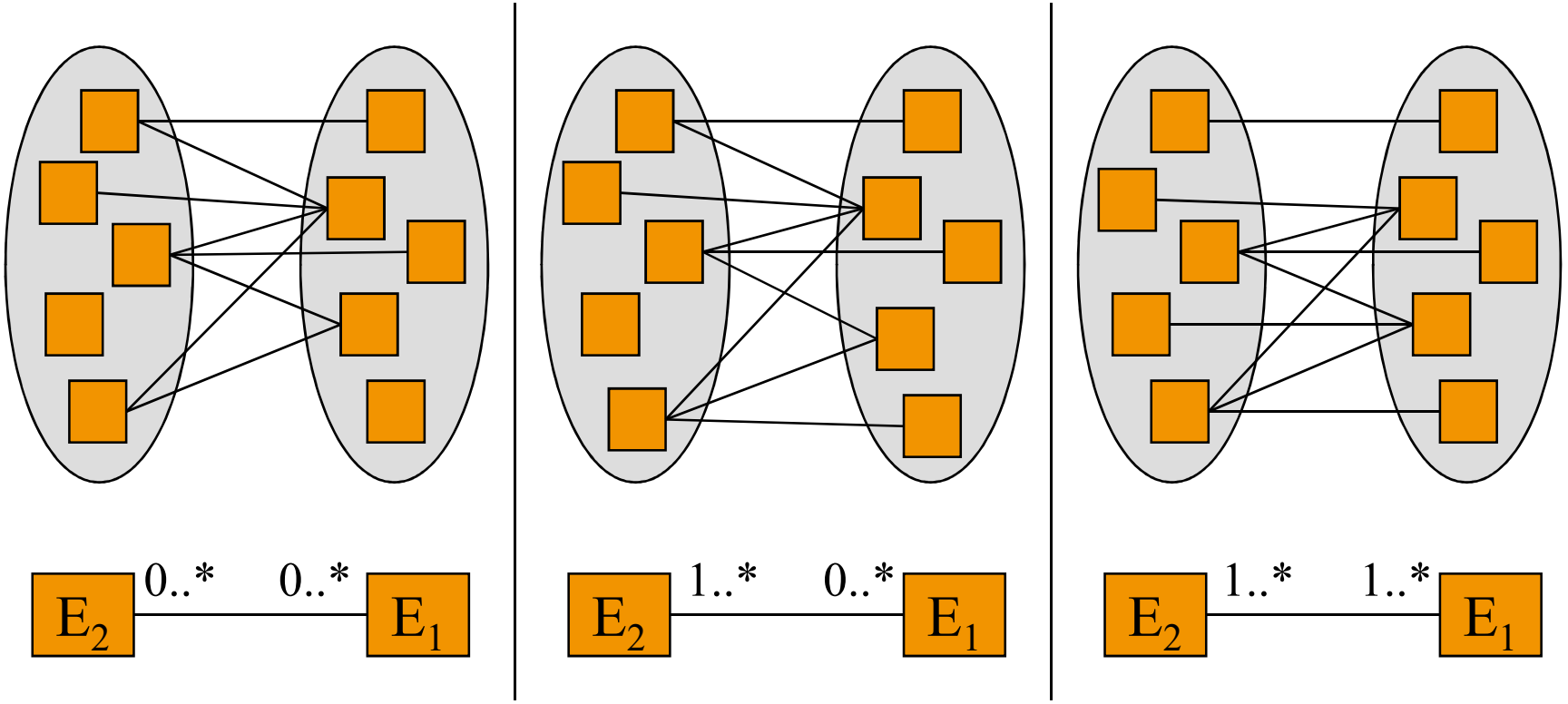
- Definition der Tabelle Projekt mit dem zusätzlichen Fremdschlüssel 'Leiter'

```
create table Projekt
  (       PrjId          decimal(3) not null,
          PrjName        varchar(20) not null,
          PrjStart       date,
          PrjPrio        enum(`Hoch`, `Mittel`,`Niedrig`) not null,
          Leiter         decimal(4),
          primary key (PrjId),
          foreign key (Leiter) references Personal (PrsId)
  );
```

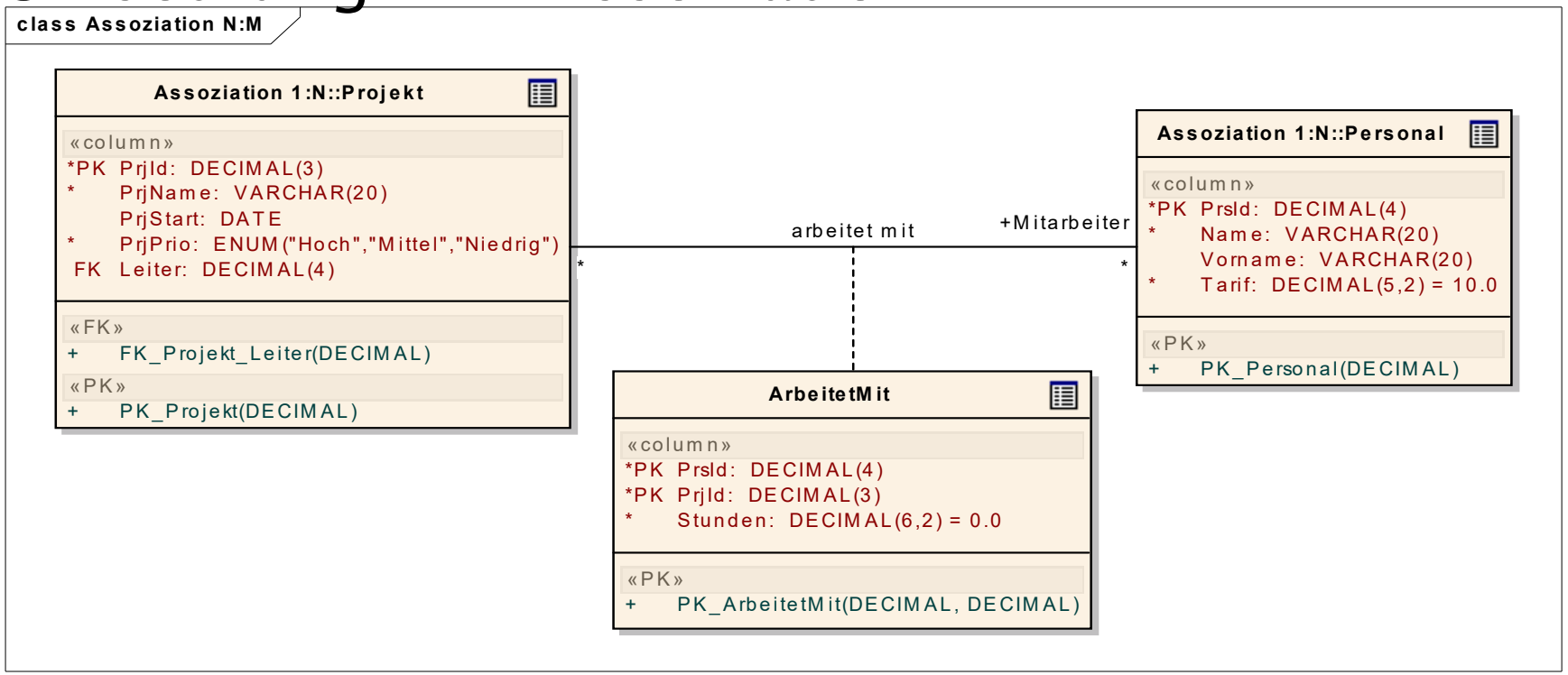


N:M Assoziation

- Eine N:M Beziehung läßt sich nicht direkt als Funktion darstellen
- Die N:M Beziehung muß als eigenständige Relation (Tabelle) mit jeweils einer funktionalen Beziehung zu den beiden Objekttypen in einer relationalen Datenbank realisiert werden (siehe folgende Seiten)



Umsetzung N:M Assoziation



Projekt

PrjId	PrjName	PrjStart	PrjPrio
111	'Datenbank'	2010.11.05	Hoch
130	'Hausnetz'	2011.01.20	Mittel
177	'Labor'	-	Hoch
299	'VirtualW'	-	Hoch

ArbeitetMit

PrjId	PrsId	Stunden
111	1324	2.15
111	1413	13.5
111	1530	27.2
177	1413	44.5
299	1413	16.5

Personal

PrsId	Name	Vorname	Tarif
1216	'Jelzin'	'Boris'	45.75
1324	'Fischer'	'Joschka'	68.99
1380	'Schmidt'	'Monika'	55.64
1413	'Ratlos'	'Rudi'	60.66
1530	'Camen'	'Bert'	120.00
1666	'Sorglos'	'Susi'	35.77

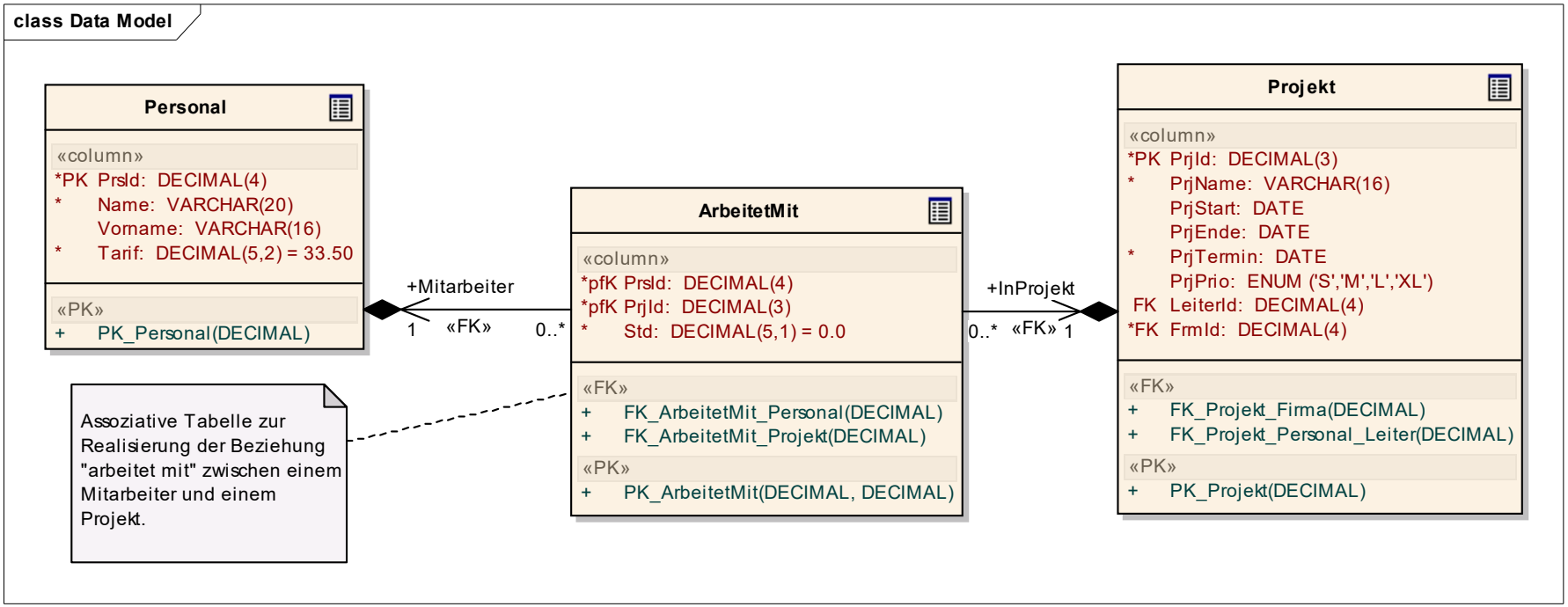
Umsetzung von N:M Assoziationen: Vereinfachung des Schemas durch Zusammenfassen von Tabellen **nicht** möglich

Regel: Keine Relation mit gleichem Primärschlüssel!

- Bei der 1:N Assoziation ist keine Vereinfachung möglich
- Beispiel einer **M:N-Beziehung:** (0..*) - (0..*)
  Projekt – ArbeitetMit – Personal hat keine Möglichkeit der Zusammenfassung:

  - Eigene Assoziative Tabelle: ArbeitetMit (neuer PK)
    Fremdschlüssel: PrjId (**not null**) verweist auf ein Projekt
    Fremdschlüssel: PrsId (**not null**) verweist auf einen Mitarbeiter aus Personal
    zusätzliches Attribut: Stunden

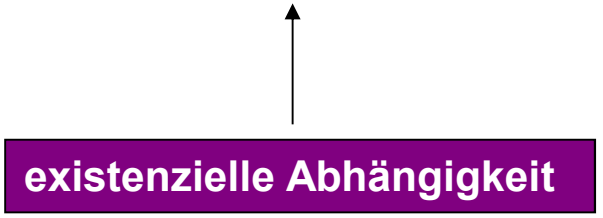
Umsetzung N:M Assoziation mit eigener Tabelle und zwei Fremdschlüsseln



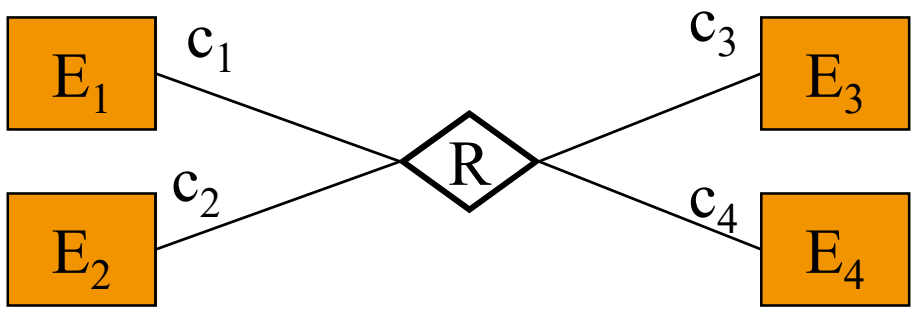
Beispiel in SQL-DDL: Umsetzung N:M Assoziation

- Definition der assoziativen Tabelle ArbeitetMit mit den zusätzlichen Fremdschlüsseln und einer `on delete` ... Regel

```
create table ArbeitetMit
  (       PrjId          decimal(3) not null,
          PrsId          decimal(4) not null,
          Stunden        decimal(5,1) not null default 0.0,
          primary key (PrjId, PrsId),
          foreign key (PrsId) references Personal (PrsId) on delete cascade,
          foreign key (PrjId) references Projekt (PrjId) on delete cascade
  );
```



Funktionalitäten von *n*-stelligen Assoziationen



Zur Bestimmung der *n* Cardinalitäten wird eine *n*-stellige Beziehung

$$R \subseteq E_1 \times E_2 \times \cdots \times E_n$$

in *n* binäre Beziehungen R_k überführt, indem jeweils eine Menge E_k (k = 1, … n) und das Kreuzprodukt der anderen Mengen ohne E_k in Beziehung gesetzt werden:

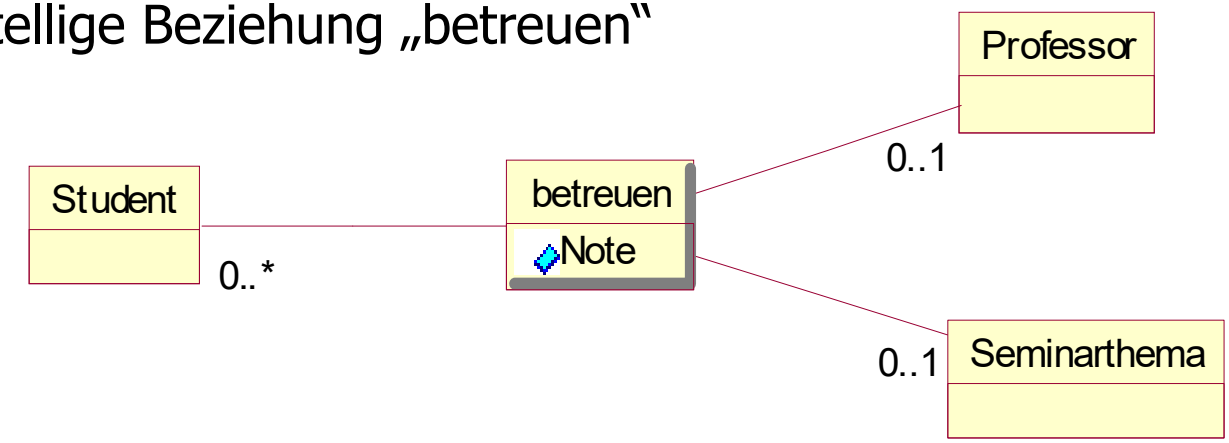
$$R_k : \left(E_1 \times \cdots \times E_{k-1} \times E_{k+1} \times \cdots \times E_n\right) \overset{R}{\longleftrightarrow} E_k$$

Die Cardinalität in Richtung E_k wird an dieser Entität wie gewohnt notiert. Von besonderem Interesse sind auch hier Beziehungen, die durch eine partielle Funktion beschrieben sind:

$$R_k : \left(E_1 \times \cdots \times E_{k-1} \times E_{k+1} \times \cdots \times E_n\right) \overset{R}{\longrightarrow} E_k \text{ mit } e_k = f(e_1 \times \cdots \times e_{k-1} \times e_{k+1} \times \cdots \times e_n)$$

Derartige partiell funktionale Beziehungen können dann mit einer :1-Angabe (im alg. 0..1) bei E_k gekennzeichnet werden.

Beispiel:
3-stellige Beziehung „betreuen“



Diese Beziehung zwischen Student, Professor und Seminarthema definiert zwei partielle Funktionen:

- betreuen 1.: Professor × Student → Seminarthema
- betreuen 2.: Seminarthema × Student → Professor

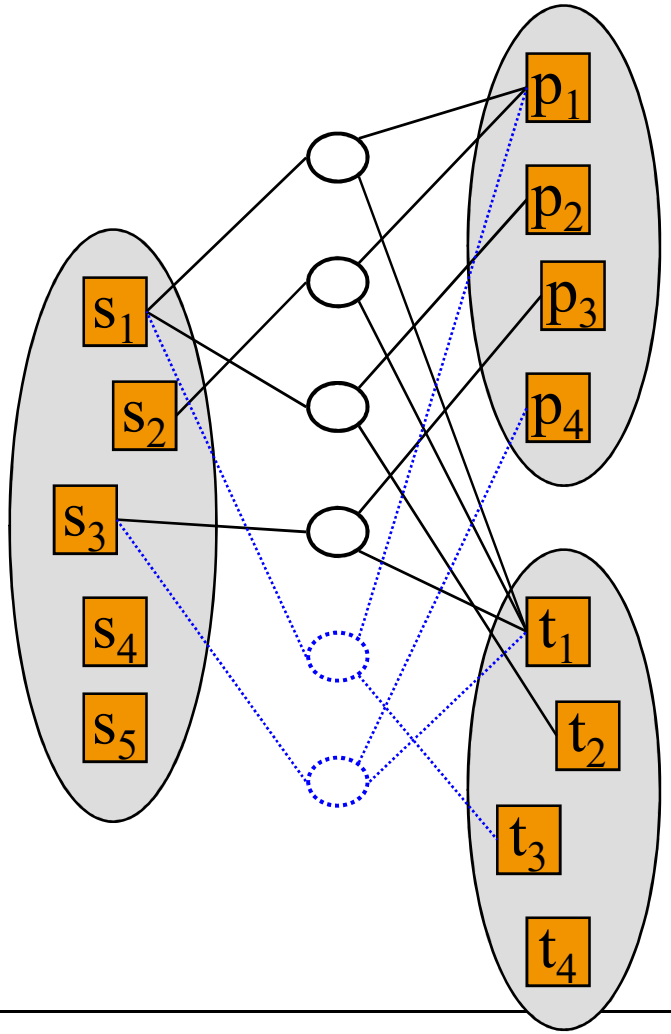
Diese 1:1:N Beziehung legt folgende Konsistenzbedingungen fest:

- Studenten dürfen bei einem Prof. nur ein Thema bearbeiten
- Studenten dürfen ein Thema nur einmalig bearbeiten

Aber:

- Professoren können ein Seminarthema „wiederverwenden“
- Ein Thema kann auch von verschiedenen Professoren vergeben werden

Beispiel:
3-stellige Beziehung „betreuen“



- Ausprägung der Beziehung „betreuen“: gestrichelte Linien markieren illegale Ausprägungen

S × T → P	S × P → T	P × T -- S
$(s_1, t_1) \rightarrow p_1$	$(s_1, p_1) \rightarrow t_1$	(p_1, t_1) -- s_1
$(s_2, t_1) \rightarrow p_1$	$(s_2, p_1) \rightarrow t_1$	(p_1, t_1) -- s_2
$(s_1, t_2) \rightarrow p_2$	$(s_1, p_2) \rightarrow t_2$	(p_2, t_2) -- s_1
$(s_3, t_1) \rightarrow p_3$	$(s_3, p_3) \rightarrow t_1$	(p_3, t_1) -- s_3
(s_3, t_1) -- p_4	(s_1, p_1) -- t_3	

Umsetzung: n-stellige Beziehungen

- Eine n-stellige Beziehung wird direkt in einer eigenen Relation realisiert.
- Die Schlüssel aller beteiligten Objekttypen werden Fremdschlüssel in der n-stelligen Relation.
- Der Schlüssel der n-stelligen Beziehung ergibt sich aus den Funktionalitäten der einzelnen Objekte, wobei jede Entität mit einer Kardinalität von 1 einen Schlüssel (unique key) als Kombination der Schlüssel der anderen Entitäten ergibt.
- Beispiel: 1:1:N-Beziehung „betreuen" (siehe S. 83)
- Professor : {[PrsId, Name, Rang]}
  Thema : {[Titel, Kurzbeschreibung, Umfang]}
  Student : {[MatNr, Name, Seit, Fach]}
  betreuen : {[PrsId, Titel, MatNr, Note]}
  unique keys : (PrsId, MatNr) und (Titel, MatNr)
- Bei einer allgemeinen Beziehung: L:M:N… wird die Kombination aller Schlüssel unique.

Beispiel Projekt-Management: Datenbankschema in SQL-DDL

```
CREATE TABLE Personal
(
  PrsId         INTEGER  NOT NULL,
  Name          VARCHAR(16)  NOT NULL,
  VorName       VARCHAR(16),
  Tarif         DECIMAL(5,2) NOT NULL,
  PRIMARY KEY (PrsId)
);

CREATE TABLE Firma
(
  FrmId         INTEGER  NOT NULL,
  FrmName       VARCHAR(16) NOT NULL,
  FrmOrt        VARCHAR(16),
  PRIMARY KEY (FrmId)
);

CREATE TABLE ArbeitetMit
(
  Std           DECIMAL(5,1) NOT NULL DEFAULT 0.0,
  PrjId         INTEGER NOT NULL,
  PrsId         INTEGER NOT NULL,
  PRIMARY KEY (PrjId, PrsId),
  FOREIGN KEY (PrjId) REFERENCES Projekt (PrjId),
  FOREIGN KEY (PrsId) REFERENCES Personal (PrsId)
);
```

```
CREATE TABLE Projekt
(
  PrjId         INTEGER NOT NULL,
  PrjName       VARCHAR(16) NOT NULL,
  PrjStart      DATE,
  PrjEnde       DATE,
  PrjTermin     DATE,
  PrjPrio       ENUM('S','M','L','XL') NOT NULL,
  FrmId         INTEGER NOT NULL,
  Leiter        INTEGER,
  PRIMARY KEY (PrjId),
  FOREIGN KEY (FrmId) REFERENCES Firma (FrmId),
  FOREIGN KEY (Leiter) REFERENCES Personal (PrsId)
);
```

Beispiel Projekt-Management: Ausprägung (Daten)

Personal			
PrsId	Name	Vorname	Tarif
1216	'Jelzin'	'Boris'	45.75
1324	'Fischer'	'Joschka'	68.99
1380	'Schmidt'	'Monika'	55.64
1413	'Ratlos'	'Rudi'	60.66
1530	'Camen'	'Bert'	120.00
1666	'Sorglos'	'Susi'	35.77
1670	'Mumm'	'Herbert'	115.00
1725	'Croft'	'Lara'	230.00

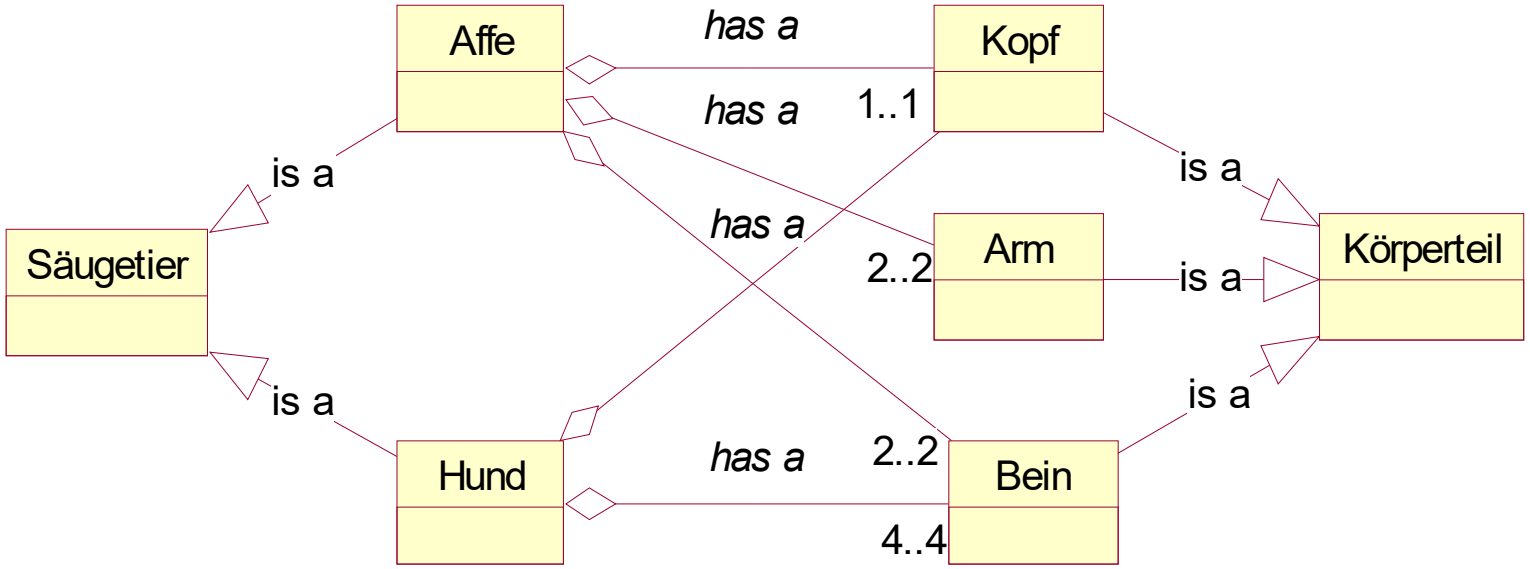
Firma		
FrmId	FrmName	FrmOrt
4000	'FH intern'	'Weiden'
6000	'MaschBau GmbH'	'München'
6500	'AutoHaus Nord'	'Bremen'
8000	'Bootswerft'	'Rostock'

ArbeitetMit		
PrjId	PrsId	Std
111	1413	20.0
111	1666	8.5
111	1380	10.7
111	1670	4.2
177	1380	205.7
177	1666	83.5
177	1216	55.6
130	1324	0.0
130	1670	0.0
130	1380	0.0

Projekt							
PrjId	PrjName	PrjTermin	PrjStart	PrjEnde	PrjPrio	FrmId	LeiterId
111	'Datenbank'	1999-11-30	1999-10-28	-	'X'	4000	-
130	'Hausnetz'	2000-03-15	2000-01-02	-	'M'	6500	1530
177	'Labor'	1999-09-30	1999-02-15	1999-11-02	'M'	6500	1216
299	'Virtual World'	2000-11-11	-	-	'S'	8000	1530

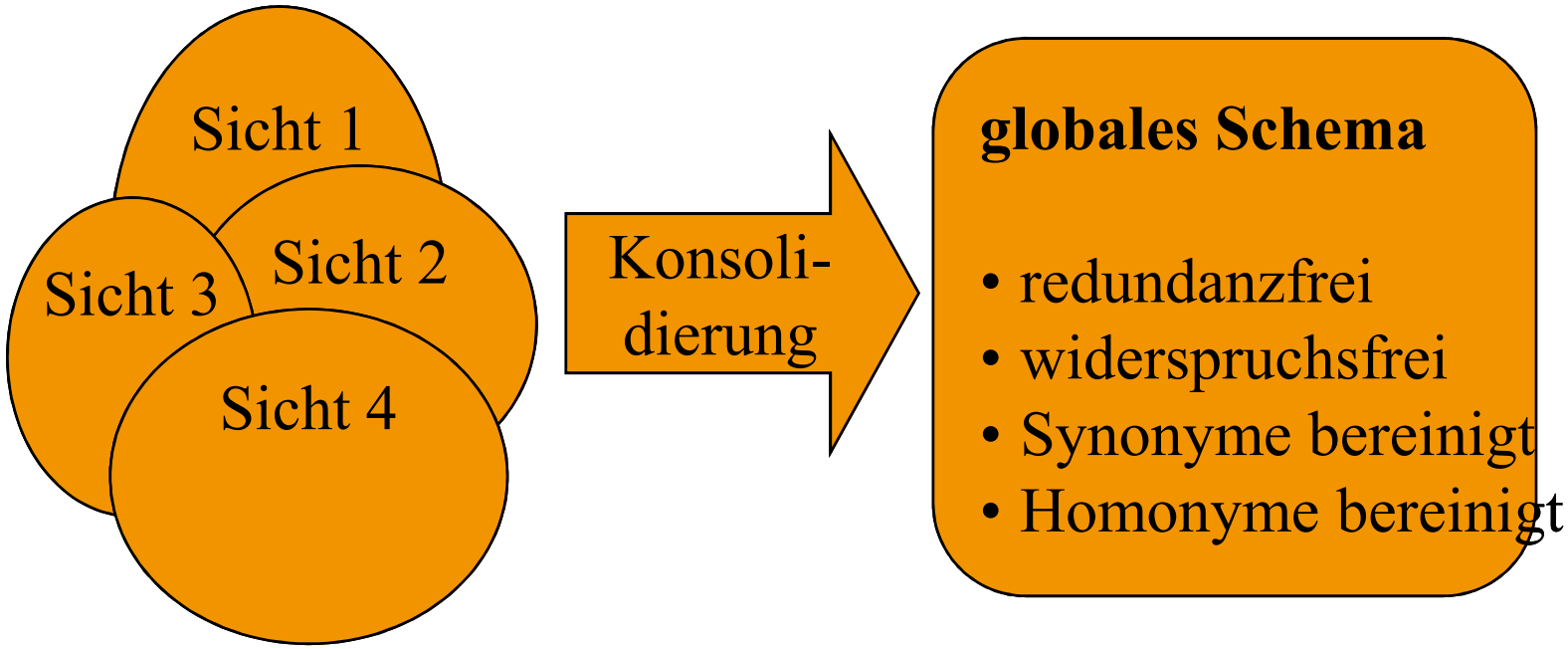
Weitere (objektorientierte) Modellelemente

- Generalisierung (is a)
  zur Modellierung hierarchischer Ähnlichkeiten
- Aggregation (has a)
  Objekte sind Bestandteil eines anderen Objekts
- Beispiel:



Integration verschiedener Sichten

- Verschiedene Anwender (Applikationen) eines Datenbankschemas haben oftmals unterschiedliche aber überlappende Anforderungen an die jeweilige Repräsentation der gespeicherten Daten.
  ⇒ Sichten



DDL: Schemadefinition

create table *Name*
(*Attribut1-Name data-type column-constraint …* ,
…
unique-constraint, referential-constraint,check-constraint
);

data-type:
siehe DDL Datentypen

column-constraint:
not null
with default *value*
primary key
references *Table-Name* **(** *Attribut-Name***)**
...

- Weitere Integritätsbedingungen (Constraints) werden im nächsten Kapitel behandelt (Beispiele: siehe Schema der Projektverwaltung).

DML: insert, update und delete

- Einfügen neuer Zeilen (konkreter Werte):

insert into *Table-Name* **(***Attribute1, Attribute2, …* **)**
values (*value1, value2, …* **), ... , (***value1, value2, …* **);**

- Allgemeine Form vom Einfügen:

insert into *Table-Name* **(***Attribute1, Attribute2, …* **)**
fullselect;

- Löschen von Einträgen (komplette Zeilen einer Tabelle):

delete from *Table-Name*
where *Condition*;

- Ändern von Einträgen in Zeilen

update *Table-Name* **set (***Attribute1, Attribute2, …* **) =**
(*value1, value2, …* **);**
update *Table-Name* **set (***Attribute1, Attribute2, …* **) =**
(*row-fullselect* **);**

3.5 Datenintegrität

Die Aufgabe eines DBMS ist nicht nur die Unterstützung bei der Speicherung und Verarbeitung von großen Datenmengen, sondern auch bei der Gewährleistung der Konsistenz der Daten. Diese Kapitel beschäftigt sich mit sogenannten *semantischen Integritätsbedingungen*, also solchen, die aus Eigenschaften der modellierten Welt abgeleitet werden können.

Die Erhaltung der Konsistenz der Daten bei Systemfehlern und unter Mehrbenutzerzugriff, sowie der Schutz vor unerlaubtem Zugriff wird später behandelt.

Referentielle Integrität in SQL: unique constraint

- *unique-constraint:*
  Jeder Schlüsselkandidat (unter Umständen aus mehreren Attributen) wird durch die Angabe von **unique** gekennzeichnet. Ein Schlüsselkandidat wird als Primärschlüssel gewählt und mit **primary key** (zusätzlich mit **not null**) markiert. Beispiele:

```
create table Personal1
(   PersNr      integer not null primary key ,
    Name        varchar (18) ,
    SozVerNr    char (10) not null unique
);
create table Personal2
(   Name        varchar (18) not null,
    Vorname     varchar (18) not null,
    SozVerNr    char (10) not null ,
    primary key (Name, Vorname) ,
    unique (SozVerNr)
);
```

Referentielle Integrität: referential constraint

- *referential-constraint:*
  Fremdschlüssel (Referenzen) werden mit **foreign key** gekennzeichnet. Sie können außerdem **not null** (1..*; sonst 0..*) und auch **unique** (1:1 Beziehung) sein.
  Beispiel:

```
create table Projekt
(   Name            varchar (18) not null primary key ,
    Auftraggeber    varchar (18) not null ,
    AuftragNr       integer not null ,
    Leiter          integer references  Personal (PersNr) ,
    foreign key (Auftraggeber, AuftragNr) references Auftrag (Firma, Nr),
    unique (Auftraggeber, AuftragNr)
);
```

Referentielle Integrität: update/delete rule

- *Rule-clause:*
  Das Verhalten bei Änderungen (update / delete) an Verweisen oder referenzierten Daten muß explizit angegeben werden. Folgende Änderungsklauseln können angegeben werden:

  [**on delete** | **on update**] [**no action** | **restrict** | **cascade** | **set null**]
- Beispiel:

```
create table Projekt
(   Name            varchar (18) not null primary key ,
    Auftraggeber    varchar (18) not null ,
    AuftragNr       integer not null ,
    Leiter          integer references  Personal (PersNr) on delete set null,
    foreign key (Auftraggeber, AuftragNr) references Auftrag (Firma, Nr)
          on delete cascade on update restrict,
    unique (Auftraggeber, AuftragNr)
);
```

Statische Integritätsbedingungen

- *check-constraint:*
  SQL ermöglicht das Einhalten von beliebigen weiteren Bedingungen, die mit Hilfe von **check** den Attributen zugeordnet werden.
  Beispiel:

```
create table Projekt
(   Name        varchar (18) not null primary key ,
    Priorität   char (1) check ( Priorität in ( 'X', 'L', 'M', 'S' )) ,
    Termin      date not null check ( Termin > current date ) ,
    ...
    check (     (Priorität =  'X'  and Termin < '2000-03-15') and
                (Priorität <> 'X' and Termin > '1999-12-15') ),
    check (     Name <> 'internal' or Priorität <> 'X'  )
);
```

Trigger

- Mit Hilfe eines Triggers können komplexe Verarbeitungen automatisch ausgelöst werden. Beispiel:

```
create trigger ProjektLeiter
after insert on Projekt
referencing new as NeuProjekt
when (NeuProjekt.LeiterId is not null)
insert into ArbeitetMit ( PrsId, PrjId, Std )
select
    LeiterId, PrjId, 0.0
from
    Projekt as p
where
    p.PrjId = NeuProjekt.PrjId
;
```

Übung Beispiel 2: Uni-Verwaltung

Entwickeln sie ein globales Schema, das die Organisation einer Universität beschreibt. Dabei sollen folgende Sichten berücksichtigt werden:

- Personalverwaltung:
  Professoren und nicht wissenschaftliche Angestellte sollen erfaßt werden mit einer Angabe der Titel, Besoldungsgruppe, Fachgebiet und Raumbelegung (bei Profs.)
- Angebot an Lehrveranstaltungen (Info. für Profs. und Stud.):
  Titel, SWS, Professor, Studenten (die diese Vorl. hören) und vorausgesetzte Lehrveranstaltungen, die für die Teilname nötig sind, sollen ersichtlich sein.
- Studentenkanzlei
  Name, MatNr, Zahl an Semestern, welche Vorlesungen werden besucht, …
- Prüfungsamt
  Verwaltung aller Prüfungen und zugehöriger Noten.

Erfinden sie bitte auch eine mögliche Ausprägung (mit sinnvollen Daten)!

Teil 4: Relationale Datenbanken und SQL

- Datenbank Anwendung -

Inhalt

4. Einführung in SQL

Einfache SQL Anfragen

- Allgemeine einfache Struktur: *subselect*

select [distinct] *Attribute1, Attribute2, …*
from *Table-Name*
where *Condition* **;**

- Sortieren der Ergebnismenge:

select-statement
(diese Form kann nur als Befehl ausgeführt werden, die Verwendung als *subselect* in anderen SQL Ausdrücken ist verboten)

select *Attribute1, Attribute2, …*
from *Table-Name*
where *Condition*
order by *Attribute1* **asc**, *Attribute2* **desc**, **…;**

- Beispiel: Die Mitarbeiter Projekt-Matrix in Tabellenform

```
select
    Name,Vorname, Tarif
from
    Personal
where
    Tarif < 200
order by
    Name, Vorname
```

Anfragen über mehrere Relationen

- „Alte Form" eines einfachen join:

select *Attribute1, Attribute2, …*
from *Table1, Table2*
where *Join-Condition*;

- „Neue Form" von joins in SQL-92

select *Attribute1, Attribute2, …*
from
Table1 Join-Operator Table2
on *Join-Condition* ;

Join-Operator:
inner join
left outer join
right outer join
full outer join

- Beispiel: Die Mitarbeiter Projekt-Matrix in Tabellenform

```
select
    Name, Vorname, PrjName, Std
from
    Personal
    left outer join
    PrsPrj on Personal.PrsId = PrsPrj.PrsId
    full outer join
    Projekt on Projekt.PrjId = PrsPrj.PrjId ;
```

Aggregatfunktionen und Gruppierung

- Aggregation und Gruppierung können Bestandteil eines *subselect* sein:

select *Attribute, ... ,Column-function,...*
from *Table,*
where *Condition*
group by *Attribute, ...*
having *Condition* **;**

Column-Function:
sum(*Expr***)** |
avg(*Expr***)** |
min(*Expr***)** |
max(*Expr***)** |
count(*) |
count(*Expr***)** | ...

- Beispiel: Die Arbeitsleistung aller Mitarbeiter

```
select
    Name,
    count(PrsPrj.Std),
    sum(PrsPrj.Std)
from
    Personal left join
    PrsPrj on Personal.PrsId = PrsPrj.PrsId
group by
    Name
```

Geschachtelte Anfragen

- In SQL können select-Anweisungen auf vielfältige Weisen verknüpft und geschachtelt werden. Jede select-Anweisung wird dabei wieder als Menge von Tupeln betrachtet, die wie eine elementare Tabelle verwendet werden kann und auch in den anderen Klauseln eines übergeordneten selects eingehen kann. Dabei müssen folgende Untermengen (*subselect*) unterschieden werden:
  - Anfragen, die höchstens ein Tupel zurückliefern (***scalar subselect***).
    Sie können wie ein elementarer Wert in einem Ausdruck (*expr*) eingesetzt werden.
  - Anfragen, die beliebig viele Tupel als Ergebnis liefern (***non scalar subselect***).
    Sie können nur dort verwendet werden, wo Mengen erwartet werden (z.B. **from**-clausel)

Scalar Subselect: korreliert / unkorreliert

- Beispiel eines unkorrelierten subselect:

```
select *
from  Personal
where Tarif =
      ( select  max(Tarif)
        from  Personal
      );
```

- Beispiel eines korrelierten subselect:

```
select
      PrsId,
      Name,
      ( select sum (Std) as Total
        from ArbeitetMit
        where Personal.PrsId = ArbeitetMit.PrsId
      )
from Personal;
```

```
select Name, Tarif -
   ( select avg (Tarif) from Personal )
from  Personal;
```

- Suche die Mitarbeiter, die mehr verdienen als ihr Projektleiter

```
select m.Name as Ma, l.Name as Ch
from Personal as m, Personal as l
where exists
(  select p.*
   from Projekt as p, PrsPrj as r
   where
      m.PrsId = r.PrsId and
      r.PrjId = p.PrjId and
      p.LeiterId = l.PrsId
) and
   m.Tarif > l.Tarif ;
```

Scalar Subselect: korreliert / unkorreliert (1)

- Ein weiteres Beispiel eines korrelierten / nicht korrelierten subselects von den Relationen Student und Professor, die beide ein Attribut GebDatum enthalten sollen. Diese selects liefern alle Studenten, die älter als der jüngste Prof. sind.

```
select s.*
from  Student as s
where exists   ( select p.*
                 from  Professor as p
                 where p.GebDatum > s.GebDatum
               );
```

- Die gleiche Anfrage unkorreliert:

```
select s.*
from  Student as s
where s.GebDatum <    ( select max (p.GebDatum)
                        from  Professor as p
                      );
```

- Übung: Mit der Beziehungsrelation „hören" können sie alle Studenten finden, die älter sind als ein Prof., dessen Vorlesung sie besuchen. Diese Anfrage kann nicht mehr so einfach in eine unkorrelierte überführt werden.

Non Scalar Subselect

- In der from-clausel eines selects können wieder beliebige subselects anstelle des Tabellennamens verwendet werden. Beispiel: Suche alle Projekte deren Gesamtaufwand (in Stunden) 100 übersteigt.

```
select q.Name, q.Aufwand
from   (  select PrjName as Name, sum(Std) as Aufwand
          from Projekt join ArbeitetMit   on Projekt.PrjId = ArbeitetMit.PrjId
          group by PrjName
        ) as q
where q.Aufwand > 100;
```

Mengenoperatoren: union, intersect, except

Mit Hilfe Vereinigung, Durchschnitt und Differenz können einzelne *subselects* verknüpft werden, die das gleiche Schema haben.

union [all]
bildet die Vereinigungsmenge zweier subselects. Mit **all** kann die Duplikatenelimination abgeschaltet werden.

intersect [all]
bildet den Durchschnitt mit / ohne Duplikatenelimination.

except [all]
bildet die Mengendifferenz mit / ohne Duplikatenelimination.

- Beispiel:

```
select Name from Personal
union
select Vorname as Name from Personal
union
select PrjName as Name from Projekt
```

Allg. Aufbau eines select-statement

select-statement:
[**with** *common-table-expression*] *fullselect* [**order by** *order-clause*]

fullselect:
subselect | **(** *fullselect* **)** | **values** *row-value* / *fullselect* *set-operator* *fullselect*

set-operator:
union [**all**] | **intersect** [**all**] / **except** [**all**]

subselect:
select *select-clause* **from** *table-reference* [**where** *search-condition*]
[**group by** *grouping-clause* [**having** *having-clause*]]

table-reference:
table-name | *view-name* | **(** *fullselect* **) as** *correlation-name* | *joined-table*

joined-table:
table-reference [**inner** / **left** / **right** / **full**] **join** *table-reference* **on** *join-condition*

Where-Clause:

- **and** | **or** | **not** können verwendet werden um einzelne logische Aussagen (Prädikate) zu verknüpfen. Dabei muß eine 3-wertige (true,false,unknown) Logik verwendet werden.
- Die elementaren Vergleichsoperatoren zur Bildung der Prädikate sind:
  = | **>** | **<** | **>=** | **<=** | **<>** | **between** *value1* **and** *value2*
- Der Operator **in** testet auf Mengenmitgliedschaft; **not in** auf nicht enthalten in. Beispiel:

  select Name **from** Personal
  where PrsId **not in (select** PrsId **from** ArbeitetMit **);**
- Eine *quantifizierende Bedingung* wird mit **any** oder **all** gebildet. Damit ist **in** identisch mit **= any**. Beispiele:

  select Name **from** Personal
  where Tarif <= **all (select** Tarif **from** Personal **);**

  select Name **from** Personal
  where Name > **any (select** Name **from** Personal **);**

Der Existenzquantor

- Operator: **exists (*subselect*)**
  wandelt einen non scalar subselect in einen skalaren Wert true / false, je nachdem ob die Ergebnismenge nicht leer / leer ist.
- Operator: **not exists (*subselect*)**
  wie oben, nur: true / false für leer / nicht leer
- Beispiel:

```
select Name
from Personal m
where
    not exists ( select * from ArbeitetMit a where a.PrsId = m.PrsId) and
    exists ( select * from Projekt p where p.LeiterId = m.PrsId) ;
```

Der Umgang mit NULL

- Ein arithmetischer Ausdruck wird **NULL** sobald einer der Operanden **NULL** ist.
- SQL verwendet in der where-clause eine dreiwertige logik, die **true**, **false** und **unknown** kennt (die Verknüpfung mit **and**, **or**, **not** ist intuitiv).
- Ins Ergebnis werden nur Tupel aufgenommen, für die die Bedingung **true** ist.
- Bei einem **group by** wird NULL als eigene Gruppe geführt.

- Ein zusätzliches Prädikat **is null** / **is not null** ermöglicht die explizite Abfrage auf NULL. Beispiel:

```
select PrjName
from Projekt
where LeiterId is null ;
```

Spezielle Sprachkonstrukte

- SQL-92 bietet ein spezielles **case when** Konstrukt, daß die Dekodierung von Attributwerten ermöglicht. Beispiel:

```
select
    PrjName,
    ( case
        when PrjPrio = 'X'     then 'sehr hoch'
        when PrjPrio = 'L'     then 'hoch'
        when PrjPrio = 'M'     then 'mittel'
        when PrjPrio = 'S'     then 'klein'
        else 'unbekannt'
    end ) as Priorität
from Projekt ;
```

Sichten

- Ein wichtiges Konzept, um das Datenbanksystem an die Bedürfnisse unterschiedlicher Benutzergruppen anzupassen, sind verschiedene Sichten (views). SQL bietet dafür:

  create view *view-name* **(** *attribute1, …* **)**
  as *fullselect* **;**

- Beispiel:

```
create view ProjekListe ( Name, Leiter, Prio, Termin, Start, Ende, Firma )
as
select
    p.PrjName, m.Name, p.PrjPrio, p.PrjTermin, p.PrjStart, p.PrjEnde, f.FrmName
from
    Projekt as p
    left join  Personal   as m on p.LeiterId = m.PrsId
    join       Firma      as f  on p.FrmId = f.FrmId
;
```

- Problem: insert bzw. update geht nicht bei allen views.

Sichten: Komplexes Beispiel

```
CREATE VIEW ProjektPrs (Vorname, Name, Rolle, Projekt, Kosten) AS
SELECT
  m.VorName, m.Name, 'Mitarb.', p.PrjName, CAST(pp.Std * m.Tarif AS DECIMAL(8,2))
FROM
  Personal AS m
  LEFT JOIN ArbeitetMit  pp ON m.PrsId = pp.PrsId
  LEFT JOIN Projekt p  ON p.PrjId = pp.PrjId
WHERE
  m.PrsId NOT IN (SELECT LeiterId FROM Projekt WHERE LeiterId is not null)
UNION ALL
SELECT
  m.VorName, m.Name, 'Leiter ', p.PrjName, CAST(NULL AS DECIMAL(8,2))
FROM
  Personal m
  JOIN Projekt p  ON p.LeiterId = m.PrsId
```

Impressum

Autor:	Prof. Dr.-Ing. Manfred Beham
Herausgegeben durch:	BMBF-Verbundprojekt „OTH mind“ der OTH Regensburg und der OTH Amberg-Weiden Wissenschaftliche Leitung Prof. Dr. habil. Clarissa Rudolph
Kontakt:	Prüfeninger Straße 58, 93049 Regensburg mind@oth-regensburg.de www.othmind-regensburg.de Hetzenrichter Weg 15, 92637 Weiden in der Oberpfalz othmind@oth-aw.de www.oth-aw.de/hochschule/oth_mind

Hinweis: Dieses Dokument wurde im Rahmen des vom Bundesministerium für Bildung und Forschung (BMBF) geförderten Bund-Länder-Wettbewerbs „Aufstieg durch Bildung: offene Hochschulen“ erstellt. Die in diesem Dokument enthaltenen Inhalte liegen in der alleinigen Verantwortung der Autor/innen.