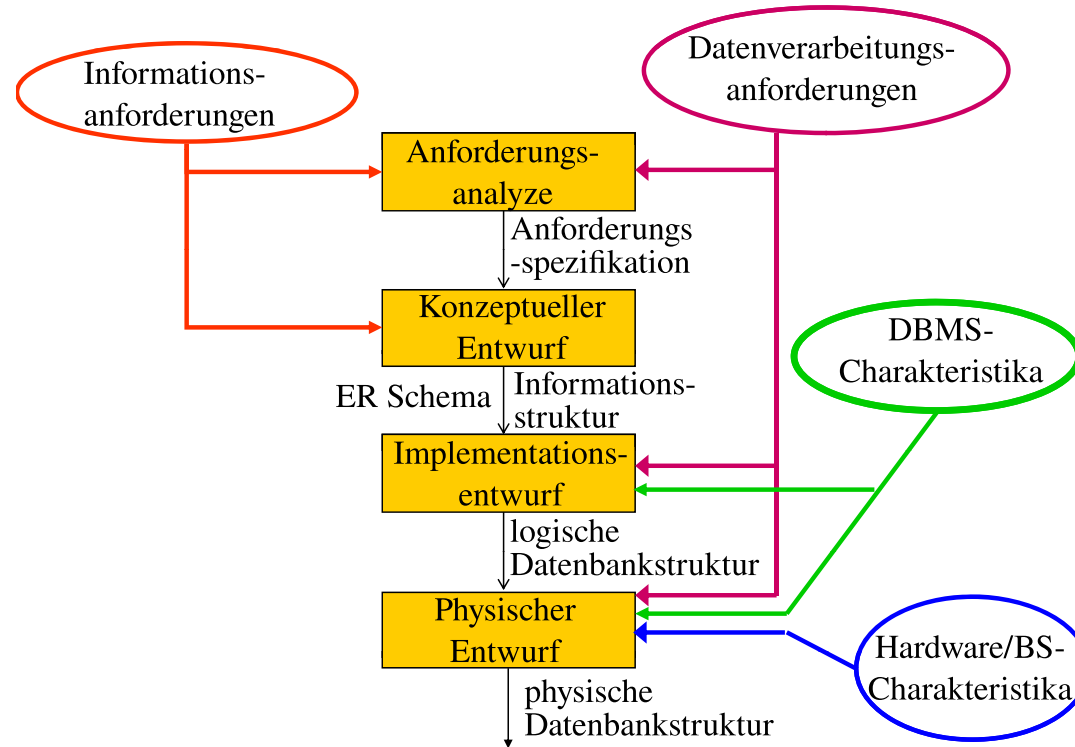


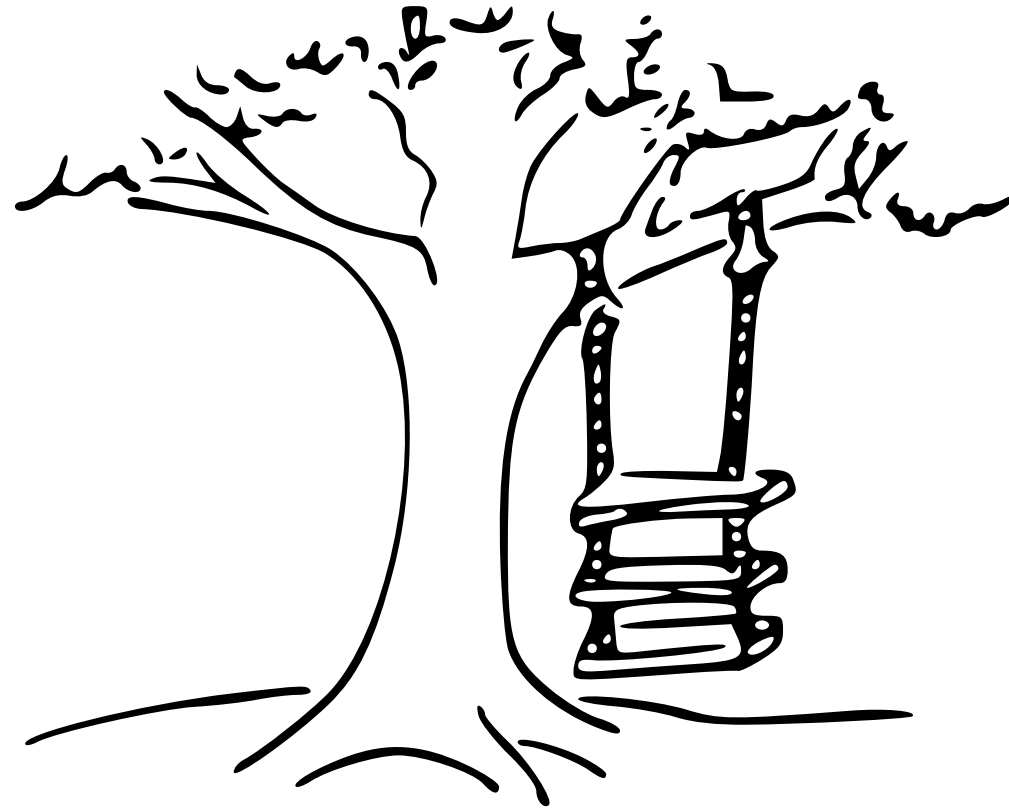
Datenbankentwurf

Datenbankentwurf

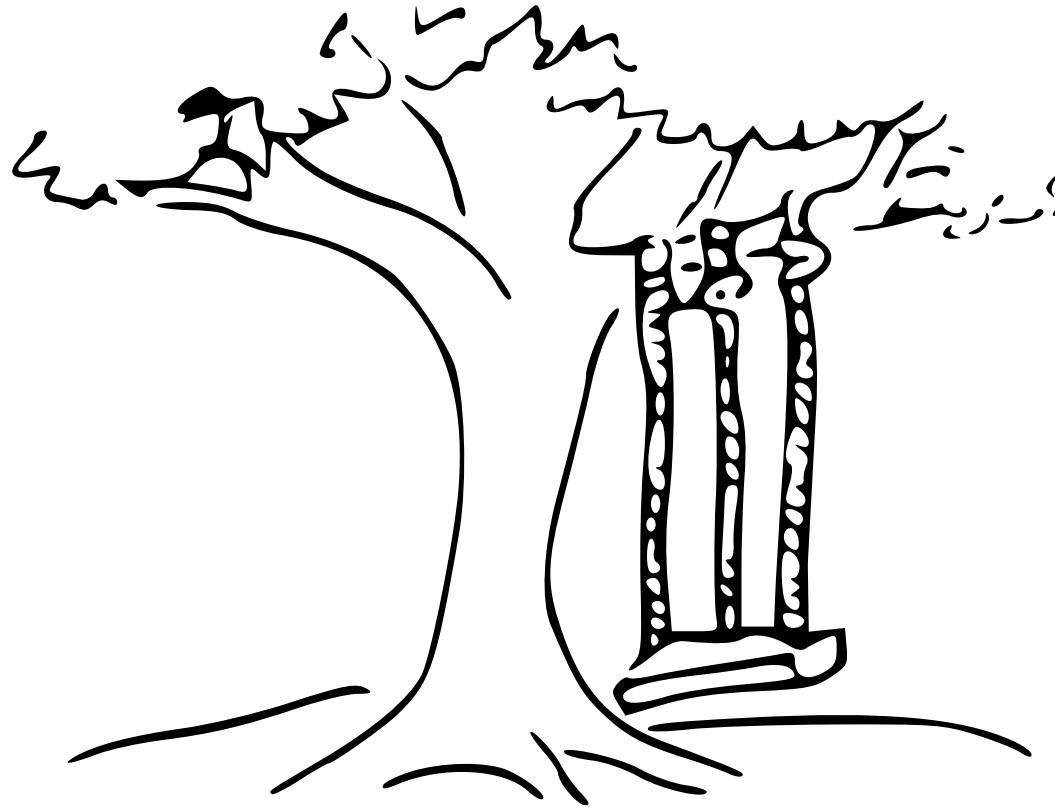


- Wird detaillierter in Softwaretechnikvorlesungen besprochen
- Wir streifen hier nur kurz dieses Gebiet
- Sehr wichtiger Punkt bei Projekten: Anwender und Entwickler verstehen sich gegenseitig

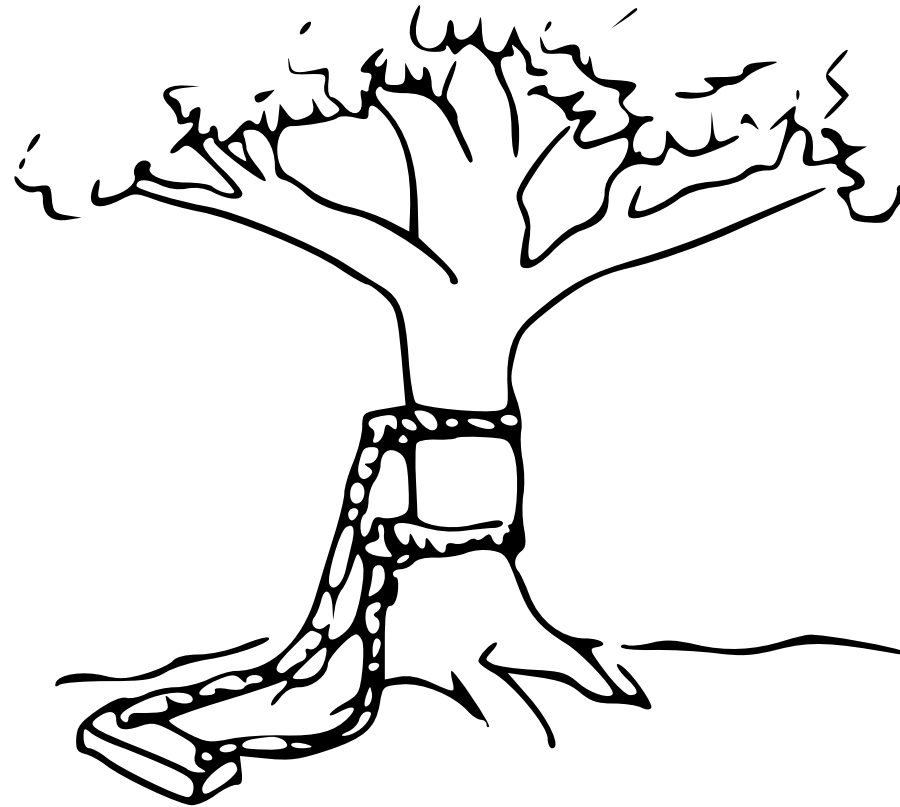
Wie es vom Geldgeber vorgeschlagen wurde:



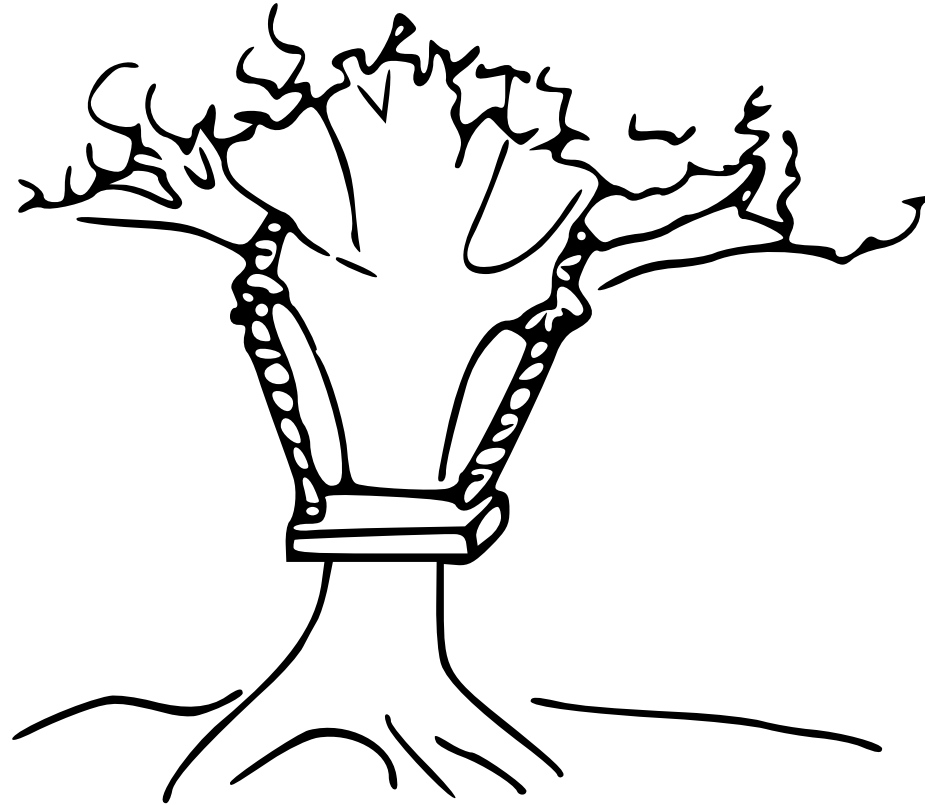
Wie es in der Spezifikation beschrieben wurde:



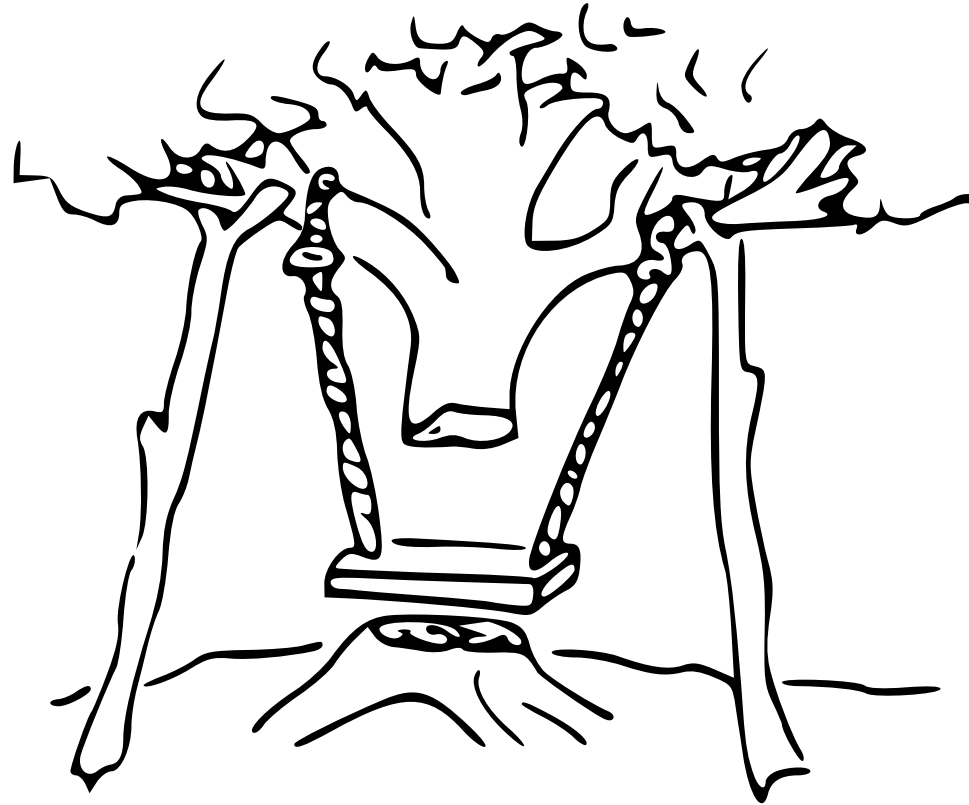
Wie es vom Chefdesigner entworfen wurde:



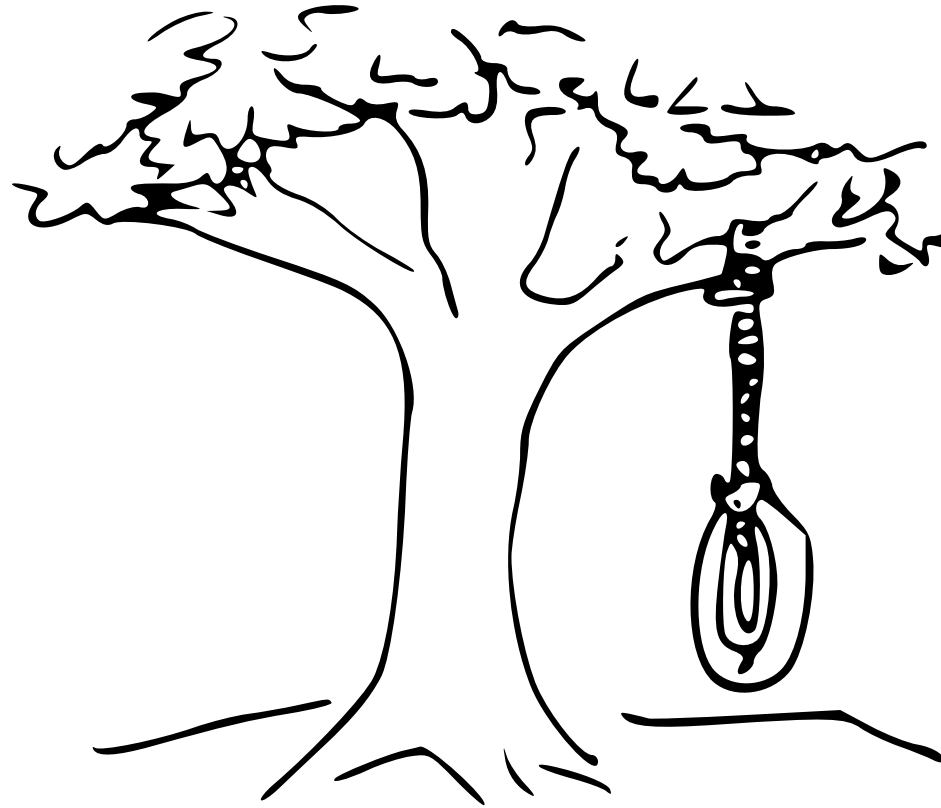
Wie es vom Programmierer implementiert wurde:



Wie es schließlich in Betrieb genommen wurde:



Was der Kunde eigentlich wollte:



Grobes Vorgehen:

1. Identifikation von Organisationseinheiten
2. Identifikation der zu unterstützenden Aufgaben
3. Anforderungs-Sammelplan
4. Anforderungs-Sammlung
5. Filterung
6. Satzklassifikationen
7. Formalisierung

Uni-Angestellte

Anzahl: 1000

Attribute

- **PersonalNummer**

- Typ: char
- Länge: 9
- Wertebereich: 0...999.999.99
- Anzahl Wiederholungen: 0
- Definiertheit: 100
- Identifizierend: ja

- **Gehalt**

- Typ: dezimal
- Länge: (8,2)
- Anzahl Wiederholung: 0
- Definiertheit: 90
- Identifizierend: nein

- **Rang**

- Typ: text
- Länge: 4
- Anzahl Wiederholung: 0
- Definiertheit: 100
- Identifizierend: nein

- Beteiligte Objekte:
 - Professoren als Prüfer
 - Studenten als Prüfling
 - Vorlesungen als Prüfungsstoff
- Attribute der Beziehung:
 - Datum
 - Uhrzeit
 - Note
- Anzahl: 100 000 pro Jahr

- Häufigkeit: halbjährlich
- benötigte Daten
 - Prüfungen
 - Studienordnungen
 - Studenteninformation
 - ...
- Priorität: hoch
- Zu verarbeitende Datenmenge
 - 500 Studenten
 - 3000 Prüfungen
 - 10 Studienordnungen

- Hier wird das *was* festgelegt (nicht das *wie*)
- Bestandteile der Spezifikation sind:
 - Prozeßbeschreibungen
 - Use cases
 - Benutzerschnittstelle
 - sonstige Schnittstellen (zu anderen Systemen)

- Die ideale Spezifikation ist
 - eindeutig
 - vollständig
 - verständlich (für alle Leser)
 - redundanzfrei
 - ...und in der Realität nicht zu erreichen

- Es gibt eine große Auswahl an Methoden:
 - Strukturierte Analyse
 - Sachbearbeiter-Methode
 - Objektorientierte Methoden
 - ...
- Aber nicht jede Methode ist immer sinnvoll

- Die eigentliche Analyse ist ein iterativer (oft auch sehr politischer) Prozeß
 - Anwender erzählt Entwickler was er gern hätte
 - Entwickler schreibt alles (was er verstanden hat) in seiner “Sprache” auf ...
 - ...und übersetzt es in die “Sprache” des Anwenders
 - Dies wird dem Anwender gezeigt, mit dem Ergebnis, daß vieles noch nicht stimmt
 - Änderungswünsche werden aufgenommen
 - Zurück zum zweiten Schritt

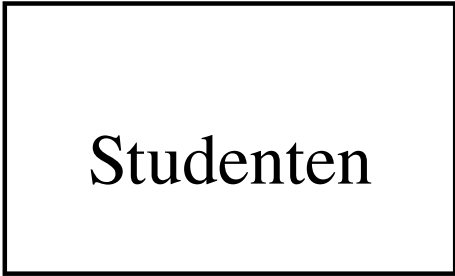
- Besonders wichtig für Datenbanksysteme ist die Modellierung der Daten, die in der Datenbank gespeichert werden sollen
- Dies wird im konzeptuellen Entwurf getrennt behandelt

- Entity-Relationship-Modell (ER-Modell) wird häufig für konzeptuellen Entwurf eingesetzt
- Ein ER-Schema ist eine graphische Repräsentation der konzeptuellen Modellierung der Daten
- Identifiziert *Entitäten* (Entities) und *Beziehungen* (Relationships) zwischen Entitäten einer Anwendungsdomäne

- Ein ER-Schema
 - enthält alle Informationseinheiten, die im zu implementierenden DBS enthalten sein sollen
 - dient auch der Kommunikation mit den späteren Benutzern (um Vollständigkeit und Korrektheit sicherzustellen)
 - kann (relativ) einfach in ein logisches Schema transformiert werden

- Entitätentyp
- Attribut
- Schlüssel
- Relationship (Beziehungstyp)
- Rolle

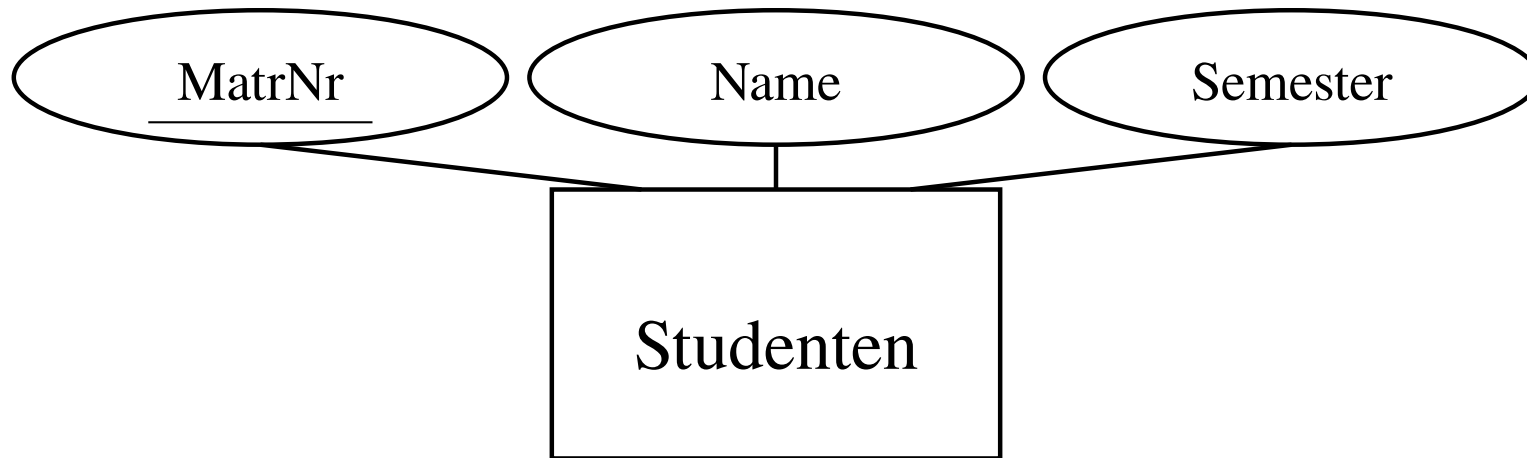
- Entität: ein Objekt aus der realen Miniwelt, das sich von anderen Objekten unterscheidet (z.B. Dinge, Personen, Konzepte)
- Entitätentypen werden durch Rechtecke repräsentiert:



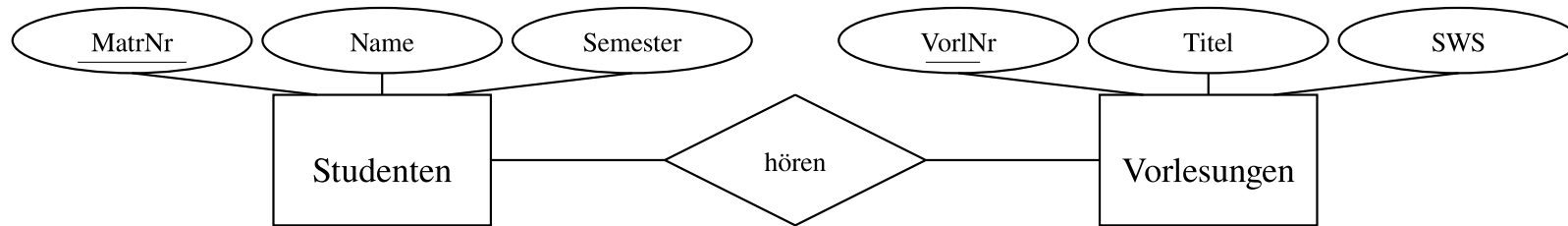
Studenten

A diagram showing a rectangular box with a black border. Inside the box, the word 'Studenten' is written in a black serif font, centered horizontally and vertically.

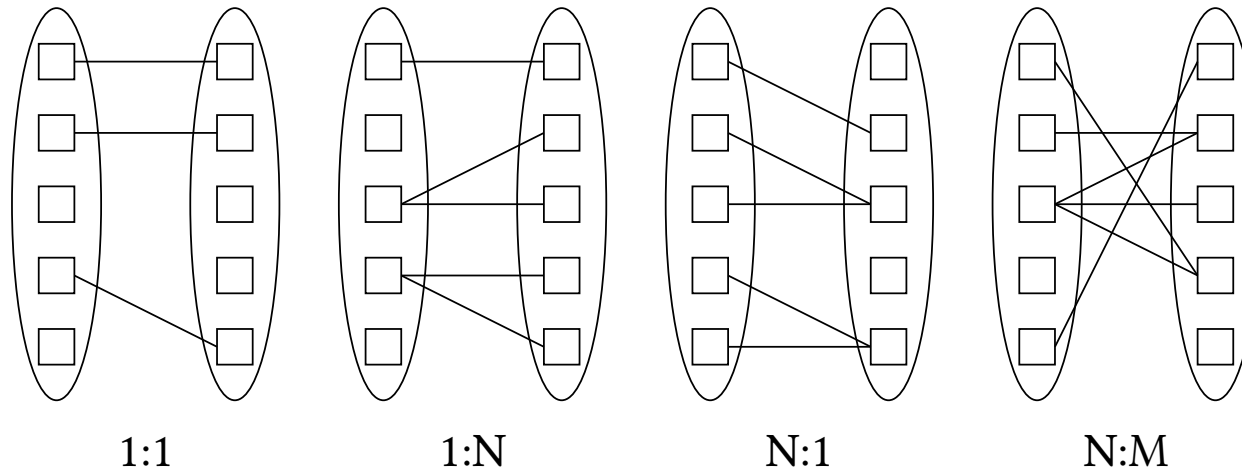
- Eine Entität wird durch Attribute beschrieben
- Zu jedem Attribut gehört ein Wertebereich (z.B. Integer, text) der die möglichen Werte dieses Attributs beschreibt
- Ein Attribut das eine Entität eindeutig beschreibt wird *Schlüssel* genannt
- Attribute werden durch eine Ellipse repräsentiert:



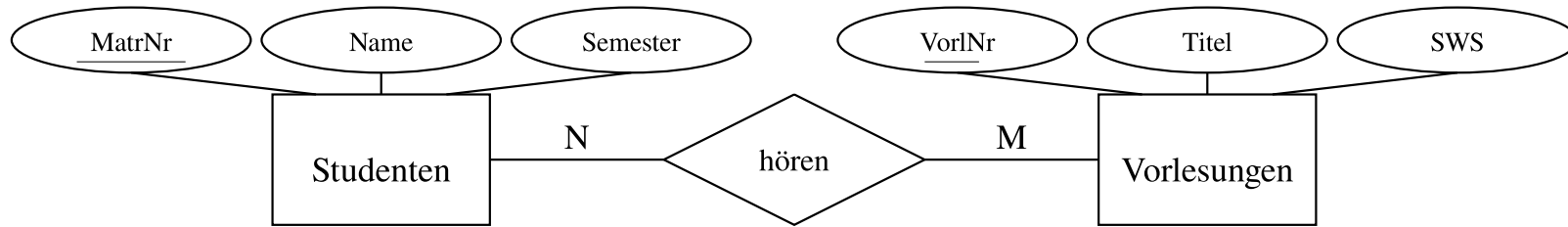
- Relationship: verbindet zwei oder mehr Entitätstypen miteinander
- Wird durch eine Raute repräsentiert:

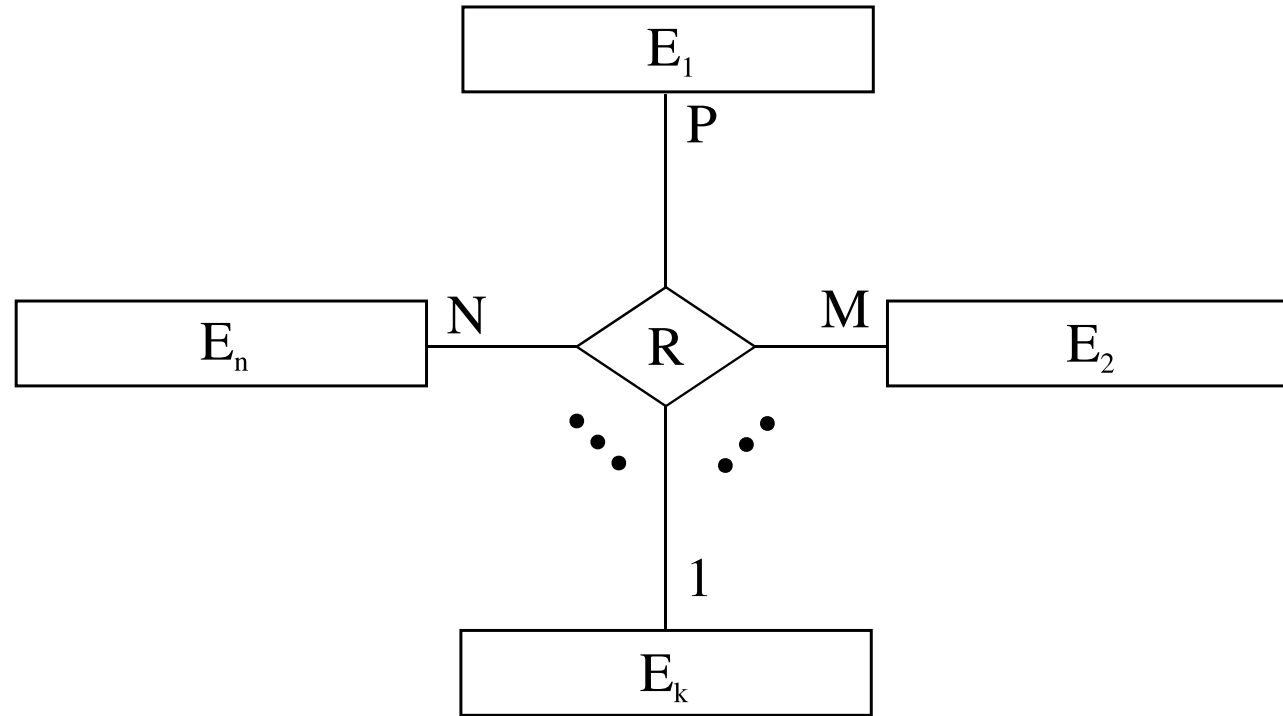


- Weitergehende Beschreibung von Beziehungen durch Funktionalitäten
- Funktionalitäten drücken aus mit wieviel anderen Entitäten eine Entität eine Beziehung eingeht



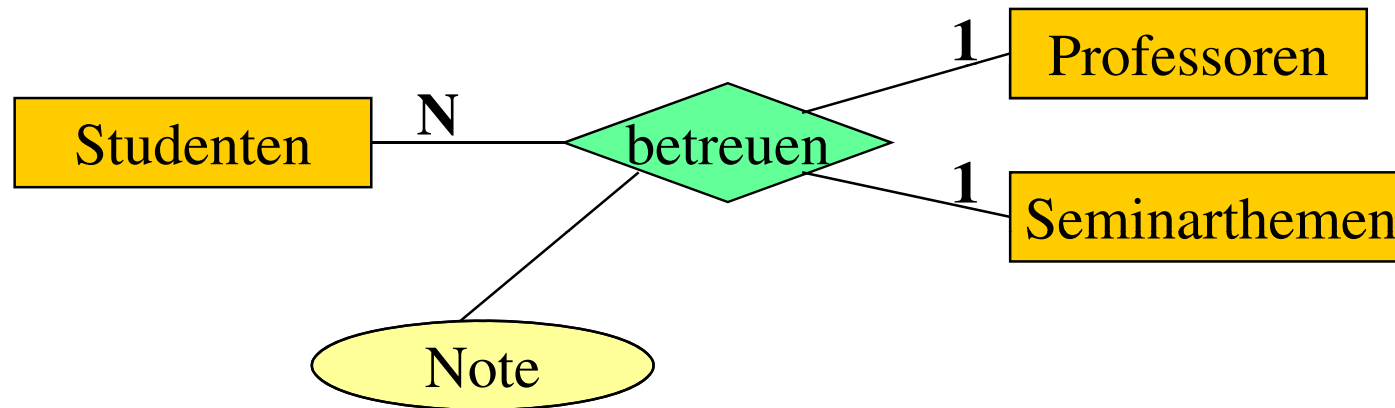
- hören ist eine N:M Beziehung:





$$R : E_1 \times \dots \times E_{k-1} \times E_{k+1} \times \dots \times E_n \rightarrow E_k$$

Beispielbeziehung: *betreuen*



$\textit{betreuen} : \textit{Professoren} \times \textit{Studenten} \rightarrow \textit{Seminarthemen}$

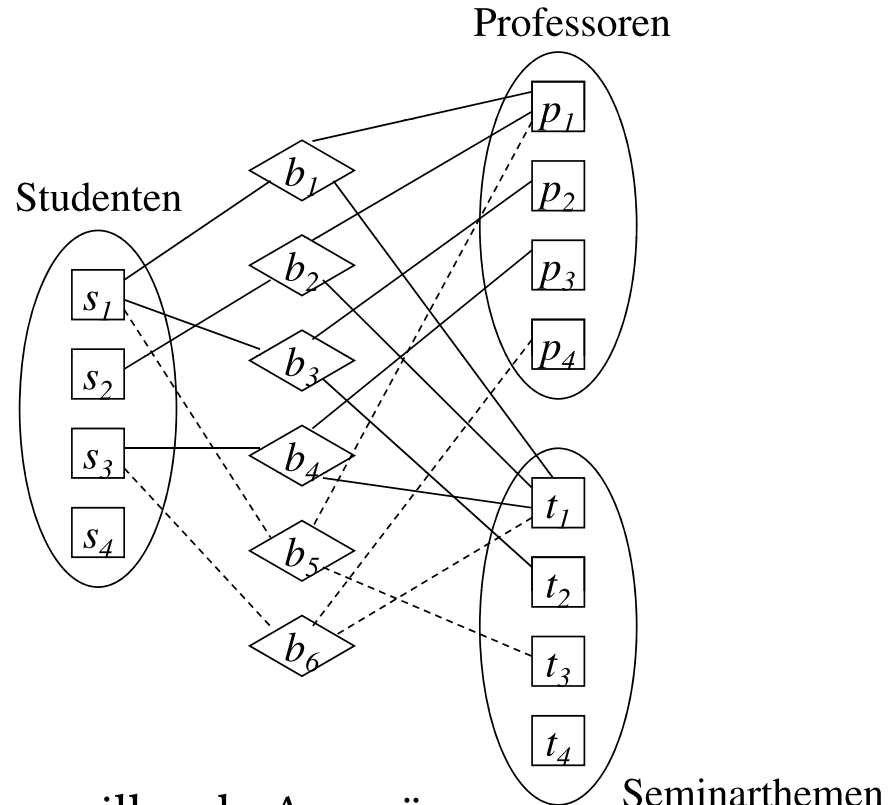
$\textit{betreuen} : \textit{Seminarthemen} \times \textit{Studenten} \rightarrow \textit{Professoren}$

Dadurch erzwungene Konsistenzbedingungen:

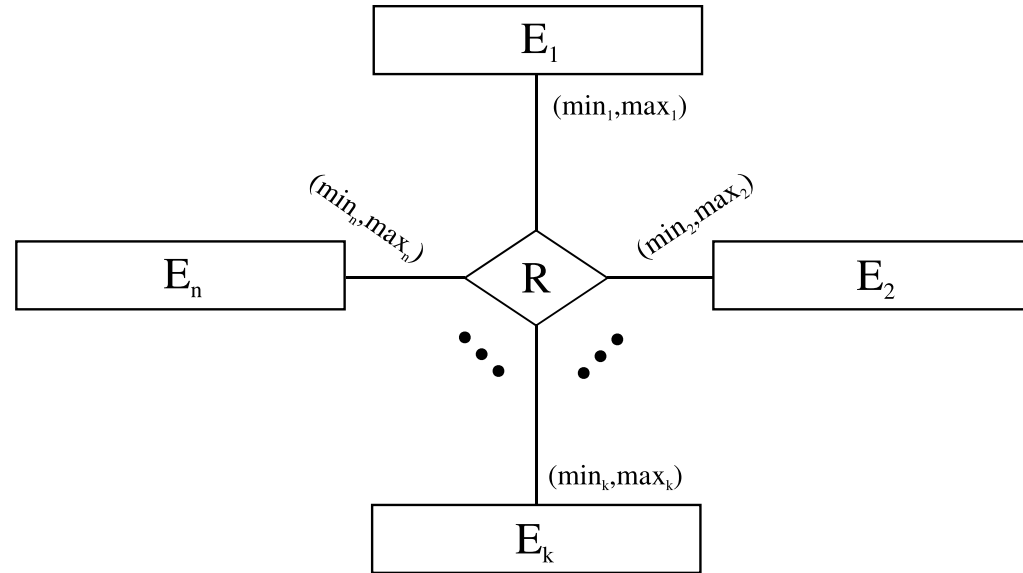
1. Studenten dürfen bei demselben Professor bzw. derselben Professorin nur ein Seminarthema “ableisten” (damit ein breites Spektrum abgedeckt wird).
2. Studenten dürfen dasselbe Seminarthema nur einmal bearbeiten – sie dürfen also nicht bei anderen Professoren ein schon einmal erteiltes Seminarthema nochmals bearbeiten.

Es sind aber folgende Datenbankzustände nach wie vor möglich:

- Professoren können dasselbe Seminarthema „wiederverwenden“ – also dasselbe Thema auch mehreren Studenten erteilen.
- Ein Thema kann von mehreren Professoren vergeben werden – aber an unterschiedliche Studenten.



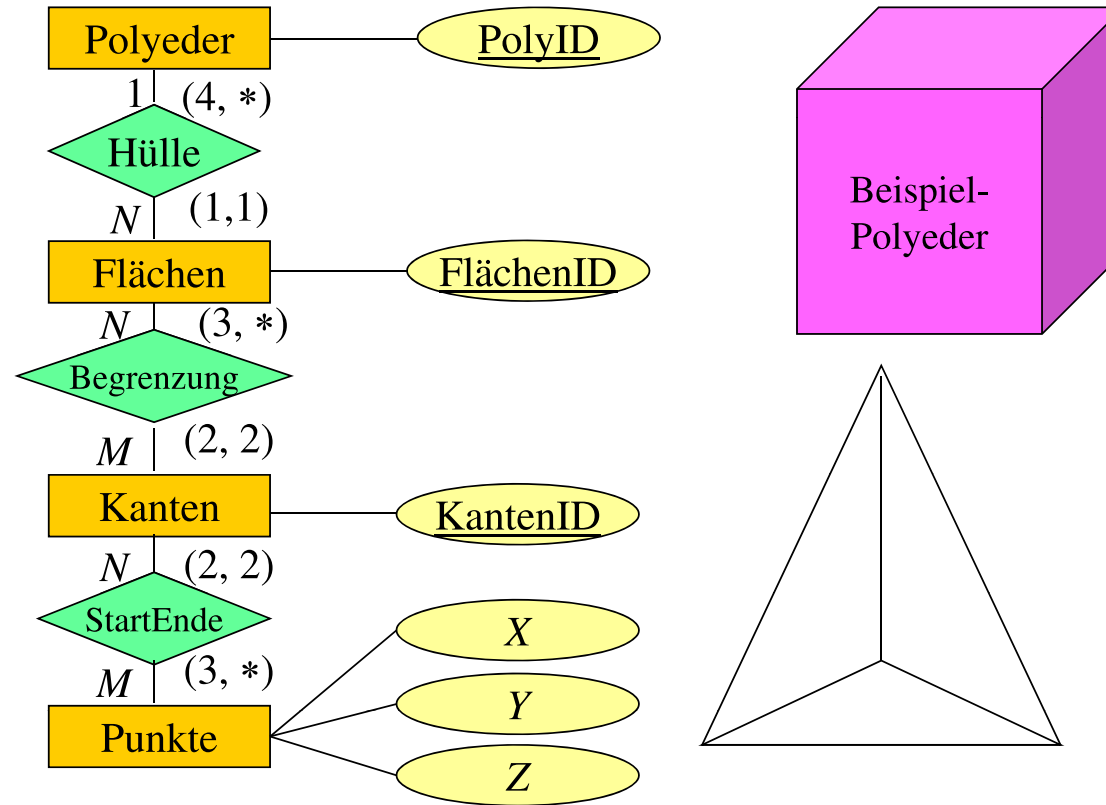
Gestrichelte Linien markieren illegale Ausprägungen.



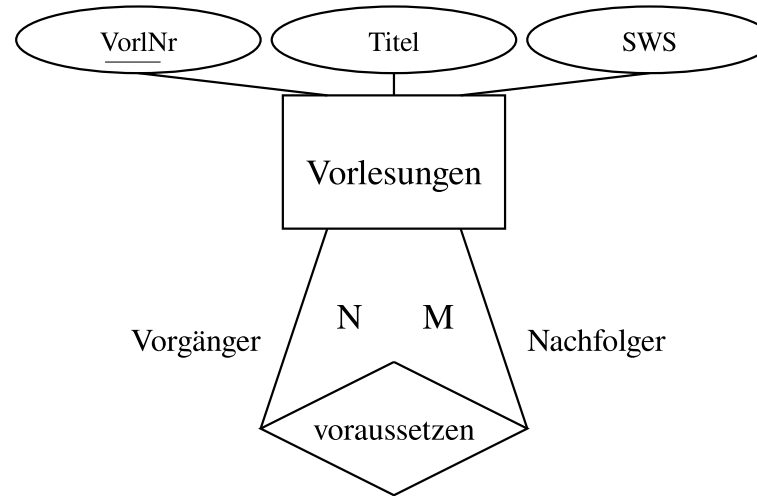
$$R \subseteq E_1 \times \dots \times E_i \times \dots \times E_n$$

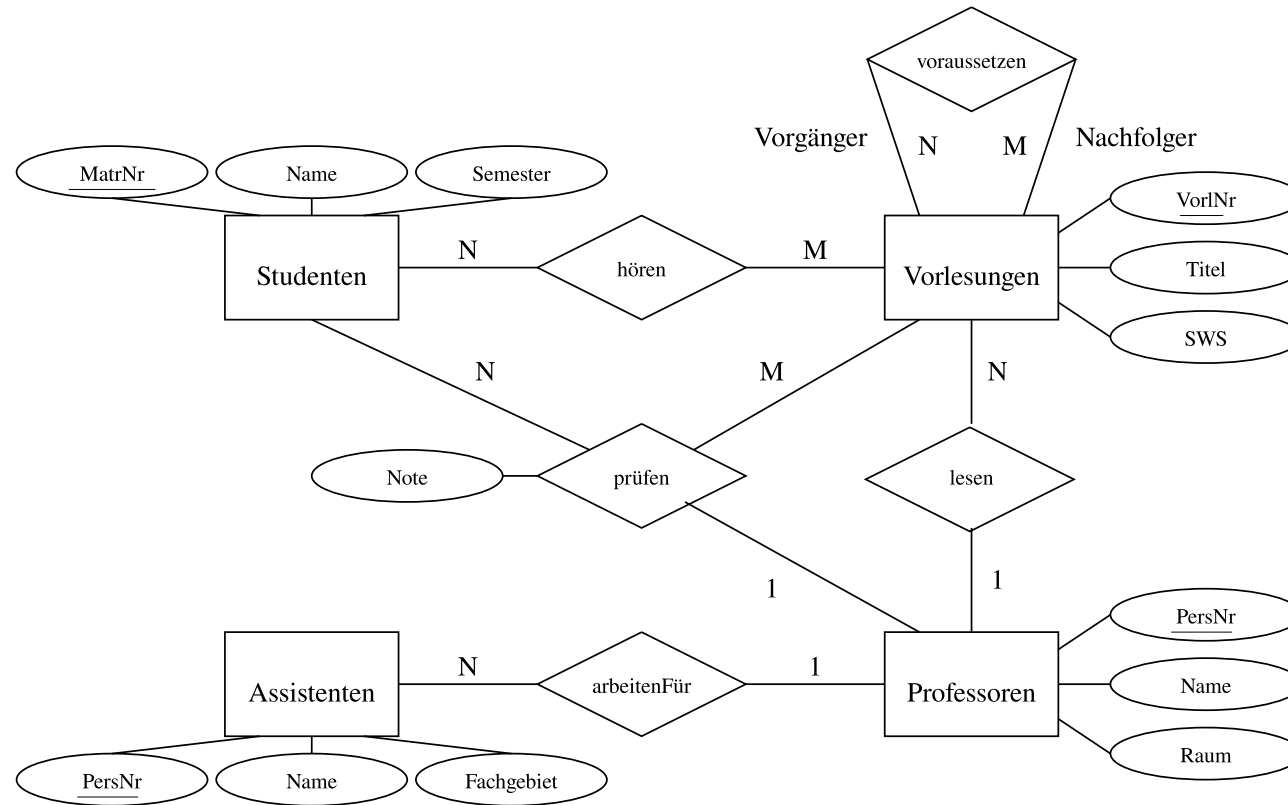
Für jedes $e_i \in E_i$ gibt es

- Mindestens $\min_i$ Tupel der Art $(\dots, e_i, \dots)$ und
- Höchstens $\max_i$ viele Tupel der Art $(\dots, e_i, \dots) \in R$



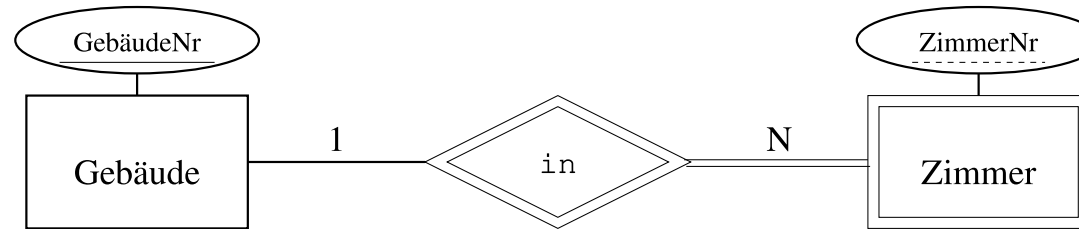
- Rollen beschreiben in welcher Weise Entitäten an einer Beziehung teilnehmen
- Besonders nützlich zur Beschreibung rekursiver Beziehungen



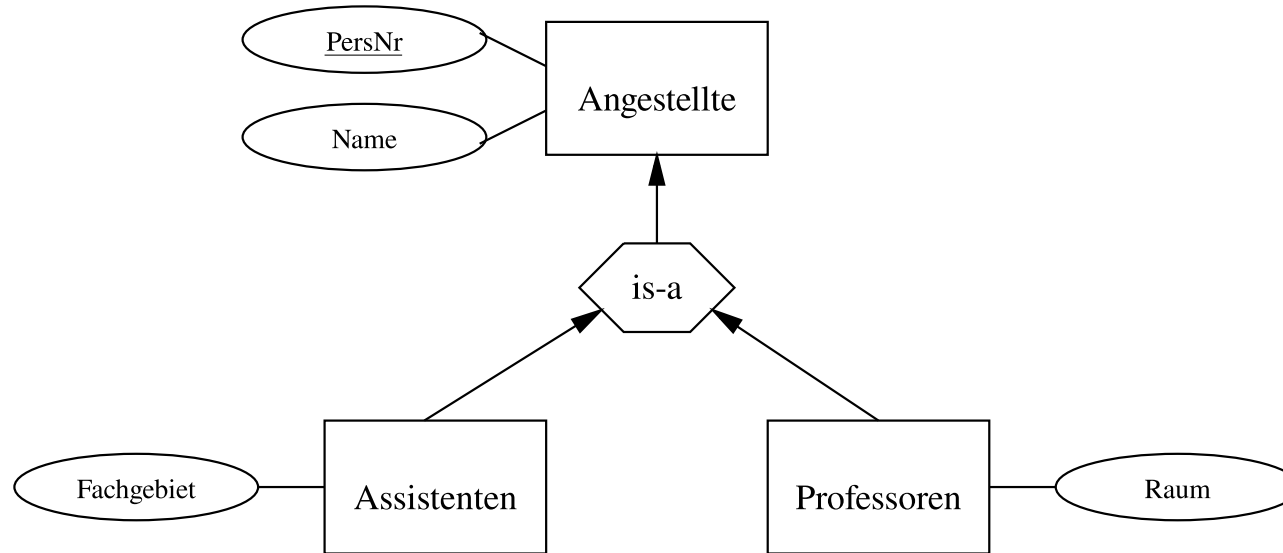


- Schwache Entitäten
- Generalisierungen
- Aggregationen

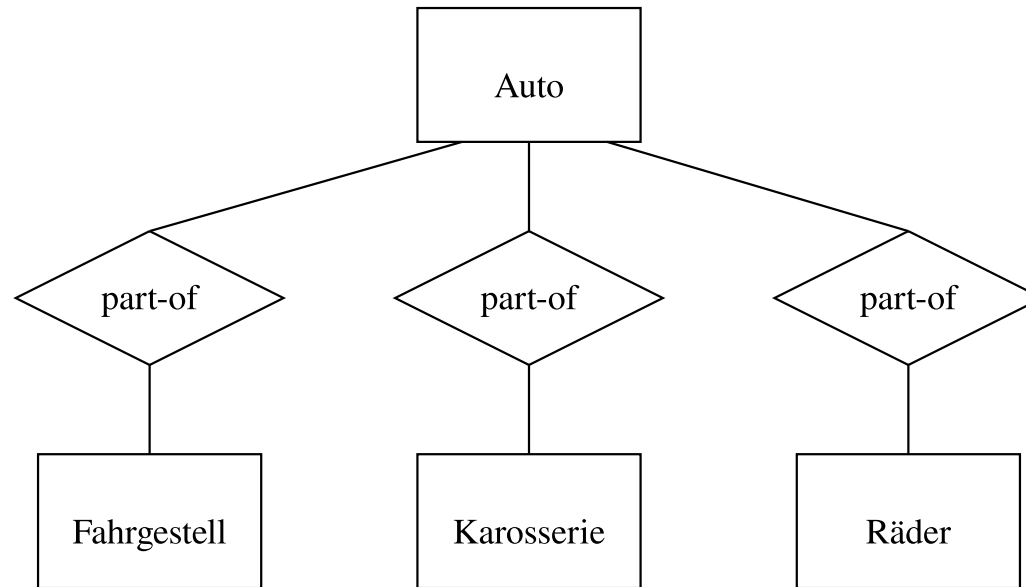
- Schwache Entität: kann nicht als eigenständiges Gebilde existieren
- Braucht eine “starke” Entität, um eindeutig identifiziert werden zu können
- Eine schwache Entität und zugehörige starke Entität gehen eine 1:N (oder seltener 1:1) Beziehung ein



- Wie in objekt-orientierten Programmiersprachen, können Attribute von einem Obertyp an einen Untertyp vererbt werden



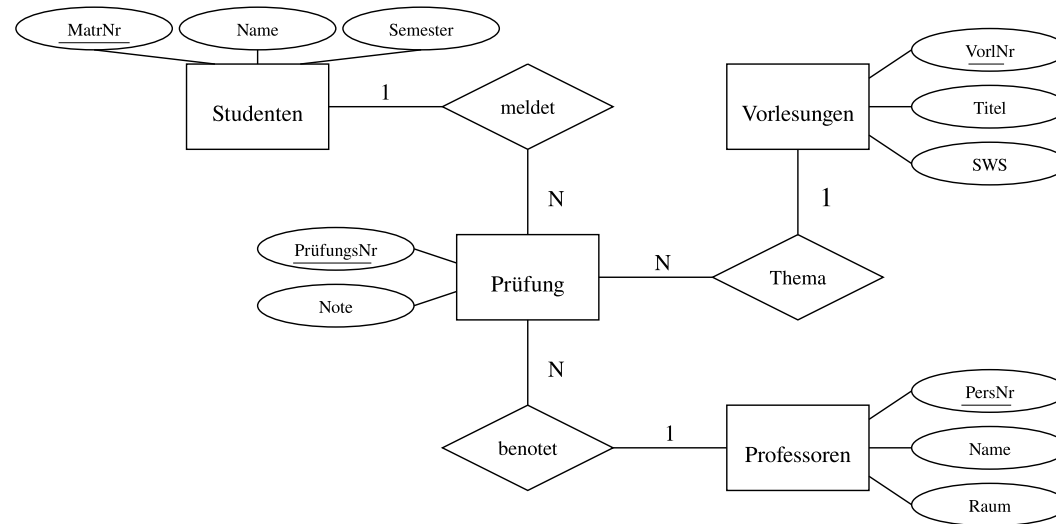
- Wird bei der Modellierung von ist-Teil-von Beziehungen benutzt



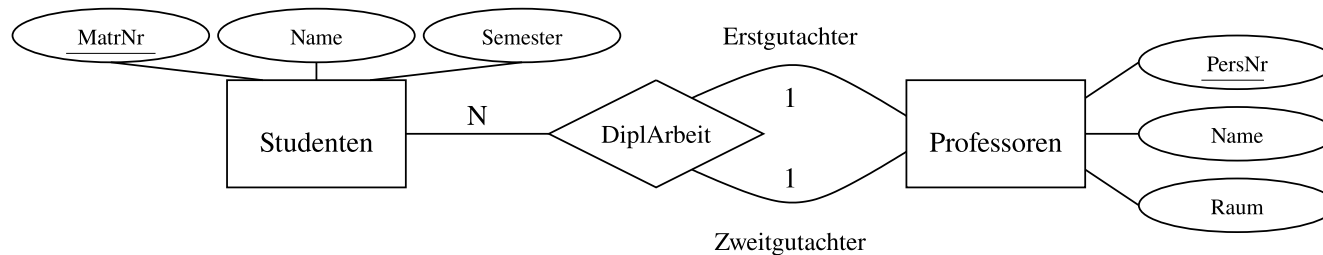
- Entität vs. Attribut
- Entität vs. Beziehung
- Binäre vs. ternäre Beziehungen
- Connection Trap

- Ein eigener Entitätstyp ist flexibler, ein Attribut einfacher
- Beispiel: wir wollen Adressinformationen von Kunden speichern
 - Falls es mehrere Adressen pro Kunde geben kann, modelliere Adresse als eigene Entität (da Attribute nicht mengenwertig sein dürfen)
 - Falls eine Adresse in einzelne Bestandteile zerlegt werden soll (Straße, Hausnummer, PLZ, Stadt, usw.), dann zerlege Adresse in einzelne Attribute (diese können dann eventuell in eigene Entität ausgliedert werden)

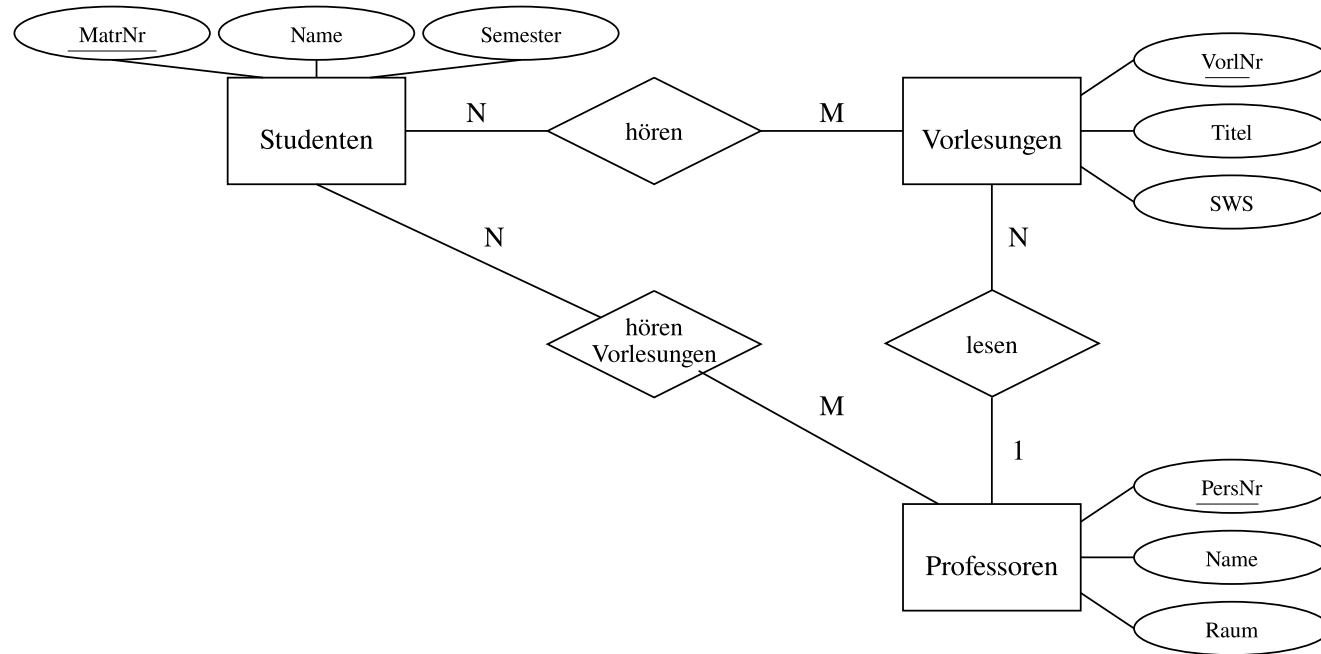
- Entität ist flexibler, Beziehung einfacher
- Beziehungen haben keine eigene Identität: dieselben Entitäten können nicht mehr als einmal eine Beziehung eingehen
- Beispiel: mehr als eine Note für die gleiche Vorlesung von der gleichen Professorin bzw. dem gleichen Professor:



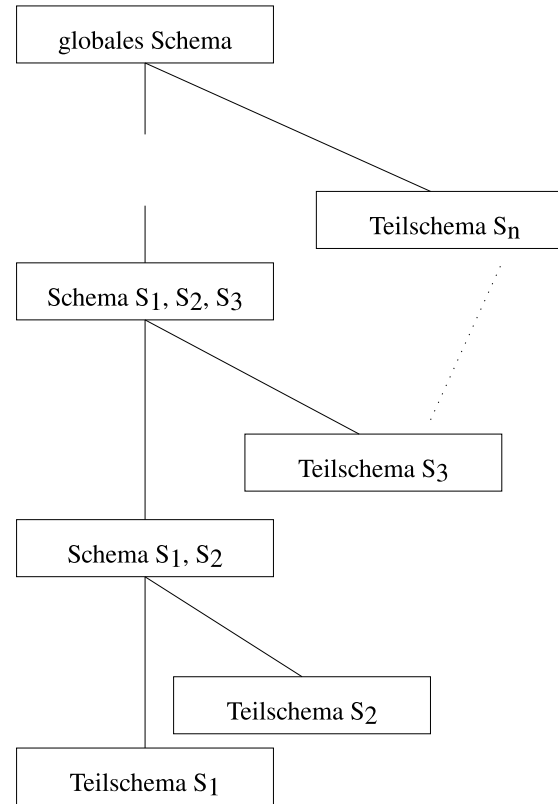
- Echte ternäre Beziehungen benötigen alle drei teilnehmenden Entitäten, um Beziehung zu beschreiben
- Beziehung prüfen im vollständigen Schema ist echt ternär
- Die folgende Beziehung ist es nicht:



- Eine Falle in die man leicht fällt ist die connection trap: redundante, zyklische Beziehungen
- hören Vorlesungen ist redundant, sie wird bereits durch hören und lesen ausgedrückt

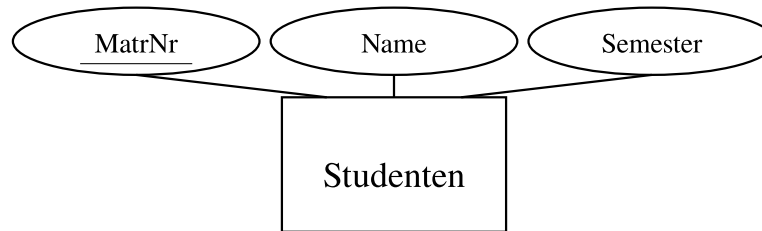


- Oft werden Anwendungen von verschiedenen Personen modelliert, die auch verschiedene Blickpunkte auf die Anwendung haben
- Es entstehen Teilschemata, die in ein Gesamtschema zusammengeführt werden müssen



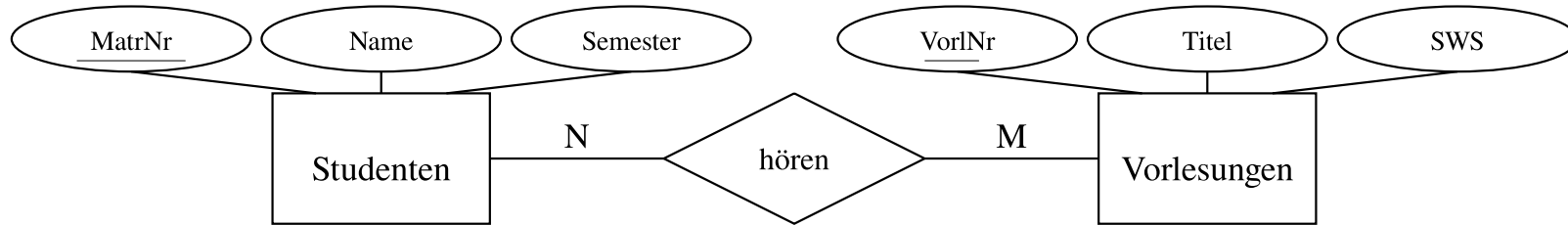
- Für ein relationales DBMS muß ein ER-Schema in ein relationales Schema übersetzt werden
- Dies geschieht in drei Schritten:
 - Übersetzung der Entitäten
 - Übersetzung der Beziehungen
 - Vereinfachung des Schemas

- Wandle jede Entität in eine Relation um
- Jedes Attribut der Entität wird zu einem Attribut der Relation (Schlüssel der Entität wird Primärschlüssel der Relation)

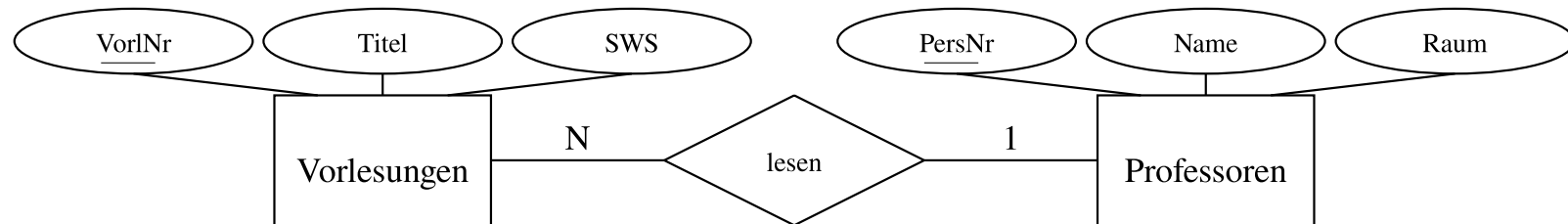


Studenten(MatrNr, Name, Semester)

- Wandle jede Beziehung in eine Relation um
- Attribute dieser Relation sind die Schlüssel der teilnehmenden Relationen (+ Attribute der Beziehung selbst)
- Der Schlüssel einer (binären) Beziehung hängt von der Art der Beziehung ab
 - N:M - Schlüssel setzt sich aus Kombination beider Entitätsschlüssel zusammen
 - 1:N, N:1 - Schlüssel der N-Seite wird Schlüssel
 - 1:1 - einer der beiden Entitätsschlüssel kann gewählt werden



$\text{hören}(\underline{\text{MatrNr}}, \underline{\text{VorlNr}})$



$\text{lesen}(\underline{\text{VorlNr}}, \text{PersNr})$

- Nach dem zweiten Schritt können Relationen mit dem gleichen Schlüssel existieren
- Diese Relationen werden in einer einzigen Relation zusammengefaßt
- Dies passiert bei 1:N, N:1 und 1:1 Beziehungen, da die Beziehungsrelation den gleichen Schlüssel wie eine der Entitätsrelationen hat

- Vorlesungen und lesen haben den gleichen Schlüssel:

Vorlesungen(VorlNr, Titel, SWS)

lesen(VorlNr, PersNr)

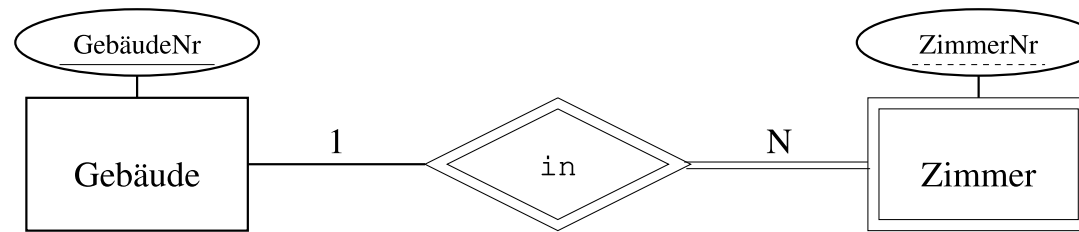


Vorlesungen(VorlNr, Titel, SWS, PersNr)

- PersNr wird *Fremdschlüssel* von Vorlesungen genannt, da er den Schlüssel in Professoren referenziert (und auch die gleichen Werte enthält)
- Häufig Umbenennung, um Referenzen klarzumachen

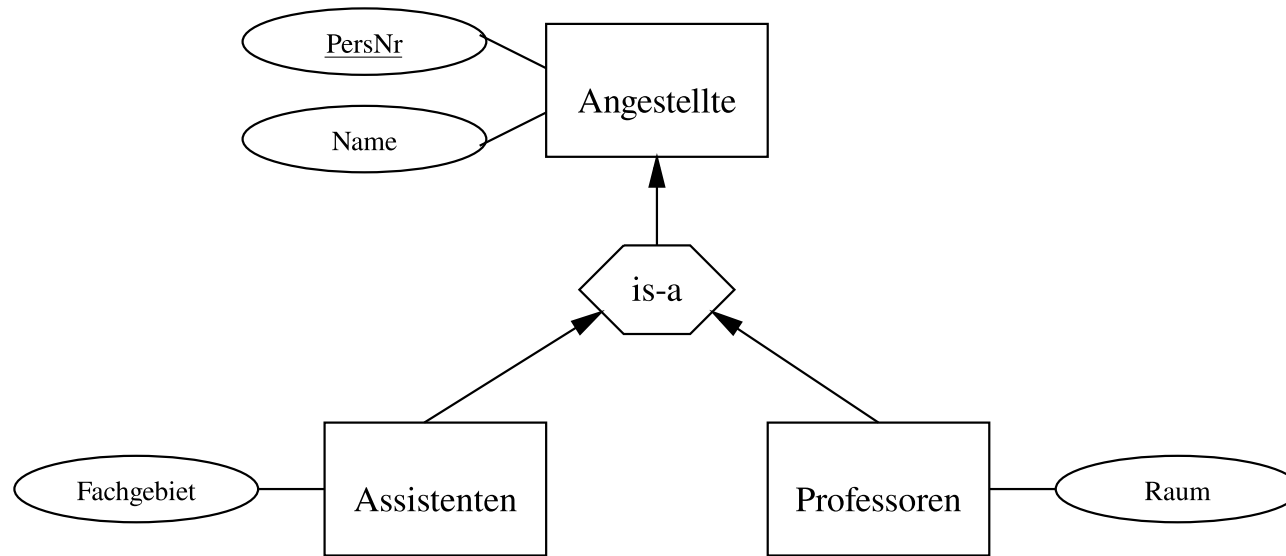
Vorlesungen(VorlNr, Titel, SWS, gelesenVon)

- Schwache Entitäten werden im Prinzip wie 1:N Beziehungen umgesetzt
- Einziger Unterschied: die schwache Entität hat einen zusammengesetzten Schlüssel



Gebäude(GebäudeNr, ...) Zimmer(GebäudeNr, ZimmerNr, ...)

- Generalisierungen können auf drei verschiedene Weisen umgesetzt werden (es gibt keine direkte äquivalente Übersetzung ins relationale Modell)



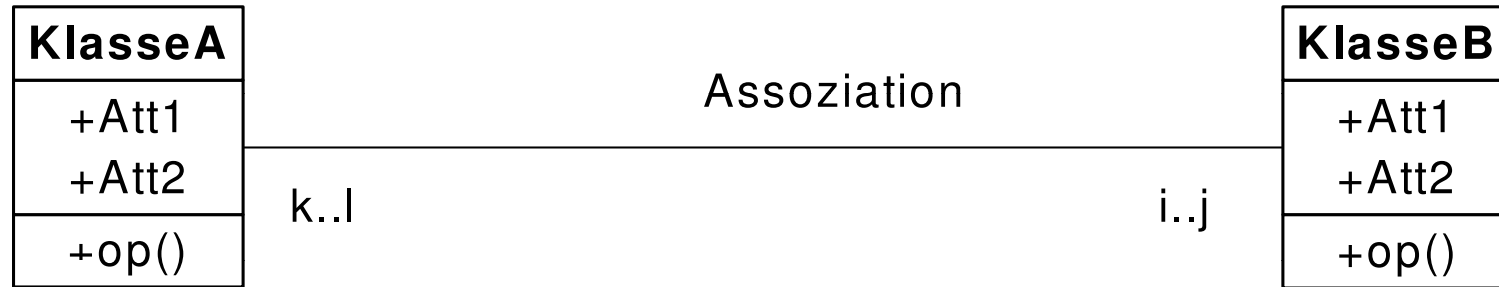
(1) Nur Obertyp, (2) nur Untertypen, (3) alle Typen

- Nur Obertyp:
Angestellte(PersNr, Name, Fachgebiet, Raum)
Problem: NULL-Werte
- Nur Untertypen:
Assistenten(PersNr, Name, Fachgebiet)
Professoren(PersNr, Name, Raum)
Problem: Angestellte die weder Assistenten noch Professoren sind
- Alle Typen:
Angestellte(PersNr, Name)
Assistenten(PersNr, Fachgebiet)
Professoren(PersNr, Raum)
Problem: Information ist verteilt auf mehrere Relationen

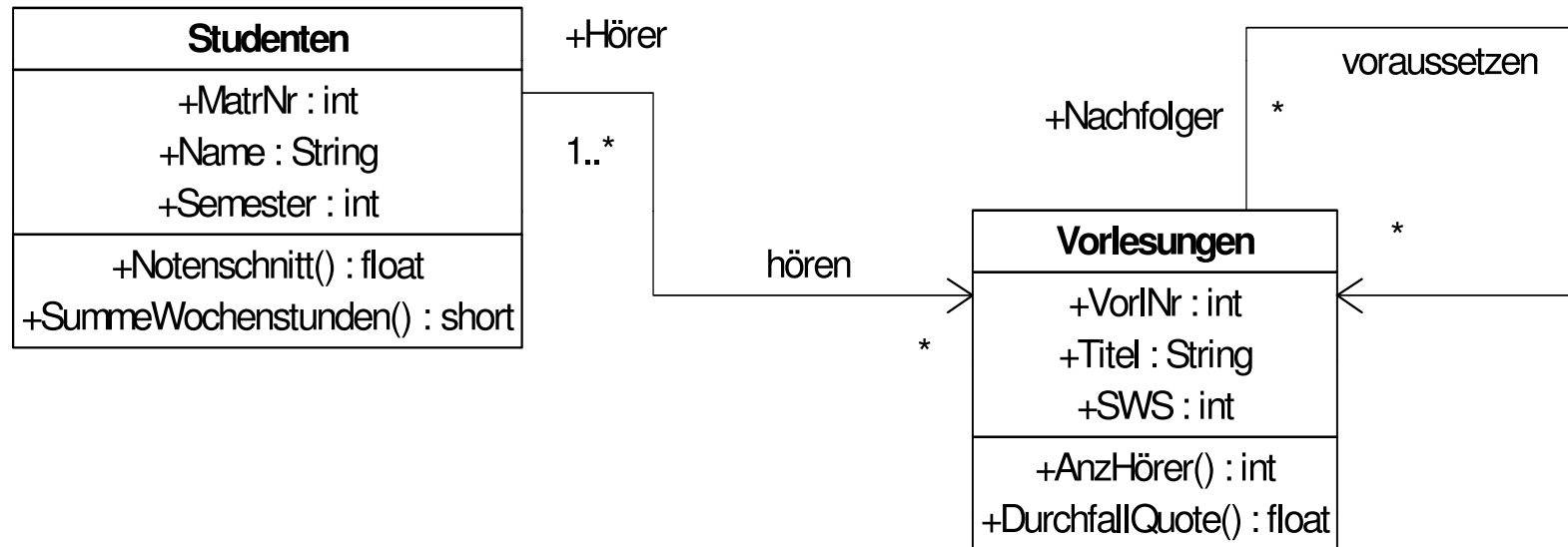
- De-facto Standard für den objekt-orientierten Software-Entwurf
- Zentrales Konstrukt ist die Klasse (class), mit der gleichartige Objekte hinsichtlich
 - Struktur (Attribute)
 - Verhalten (Operationen/Methoden)

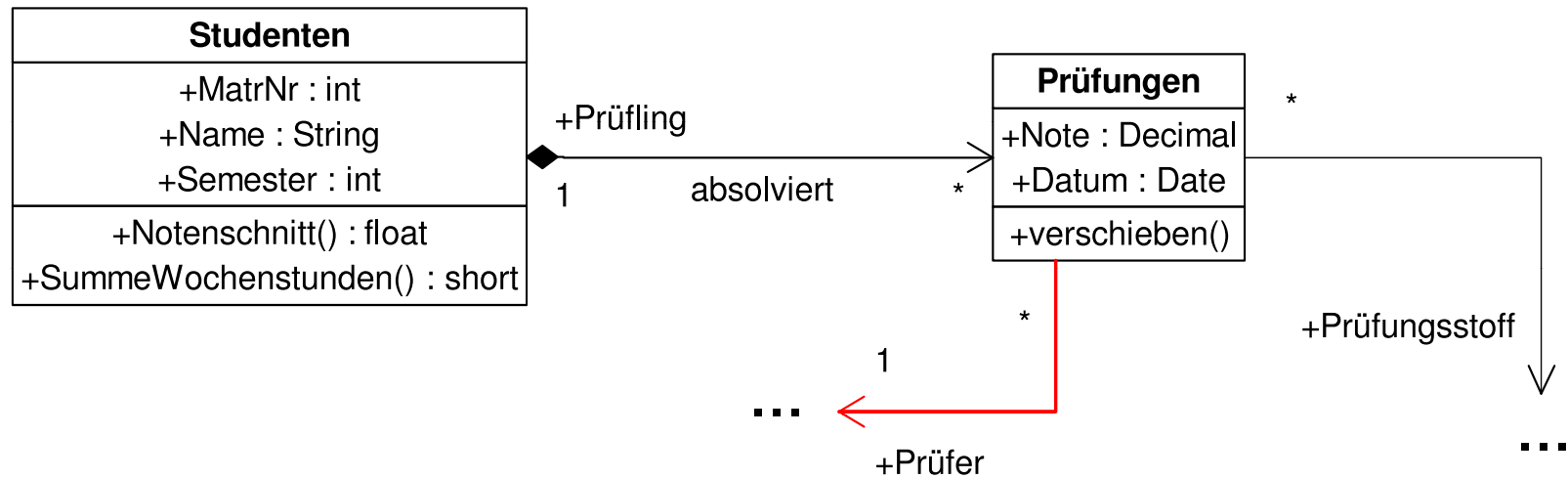
modelliert werden

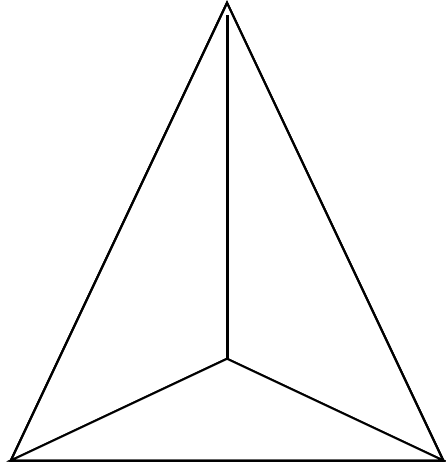
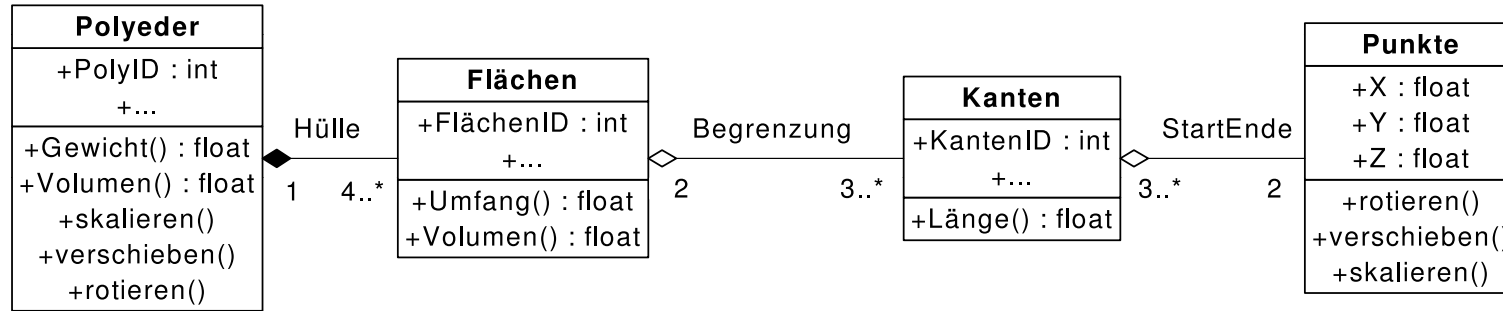
- Assoziationen zwischen Klassen entsprechen Beziehungstypen
 - Generalisierungshierarchien
 - Aggregation

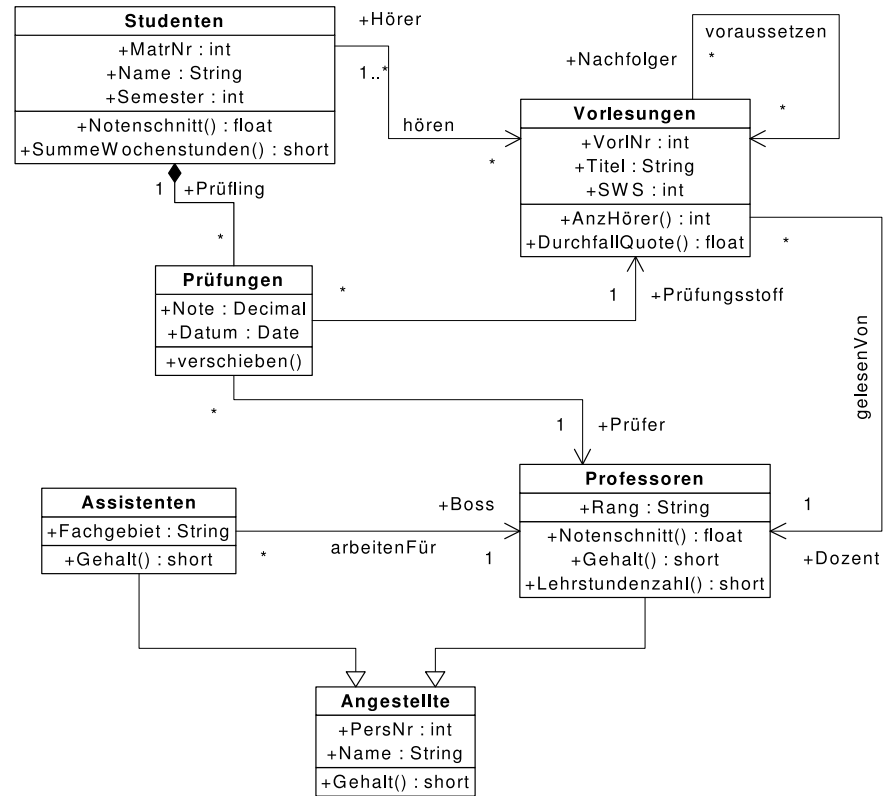


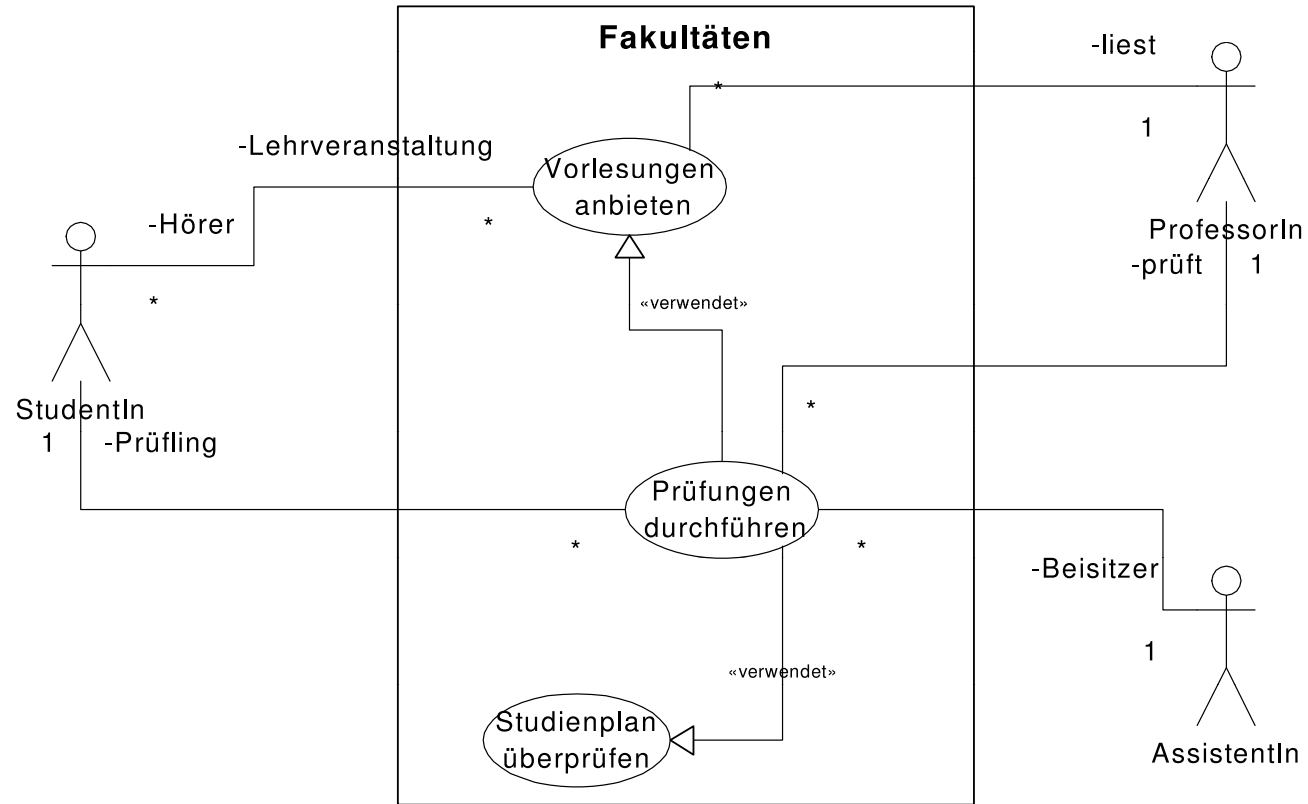
- Jedes Element von Klasse A steht mit mindestens i Elementen der KlasseB in Beziehung
- ... und mit maximal j vielen KlasseB-Elementen
- Analoges gilt für das Intervall $k..l$
- Multiplizitätsangabe ist analog zur Funktionalitätsangabe im ER-Modell
 - Nicht zur (min,max)-Angabe: **Vorsicht!**

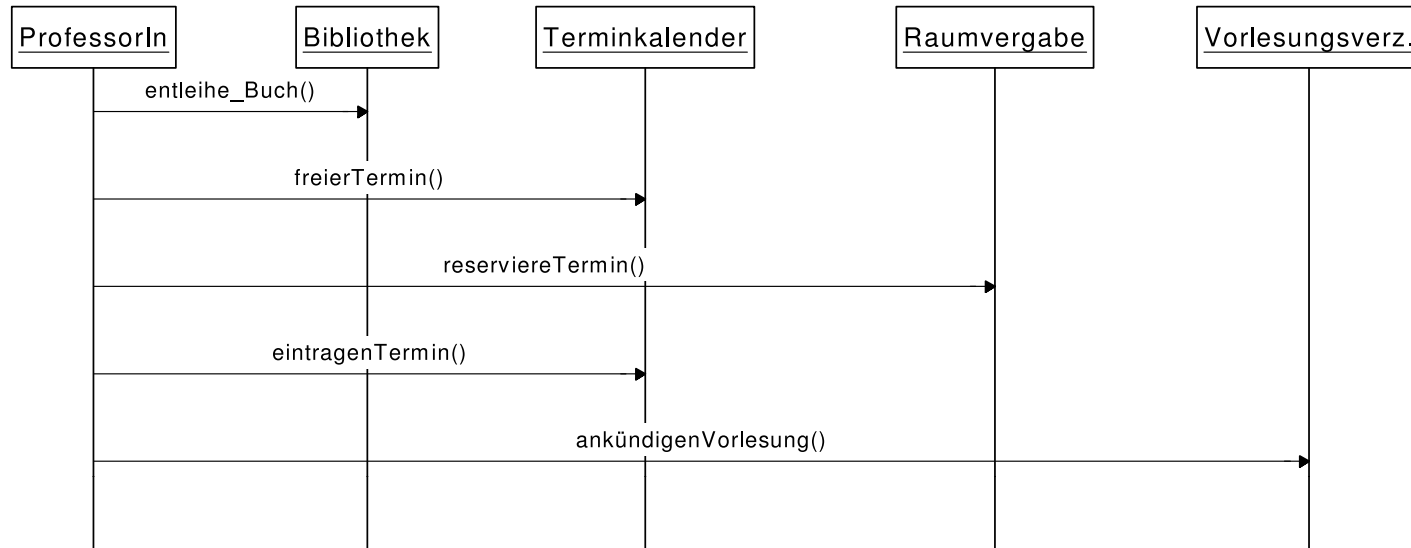


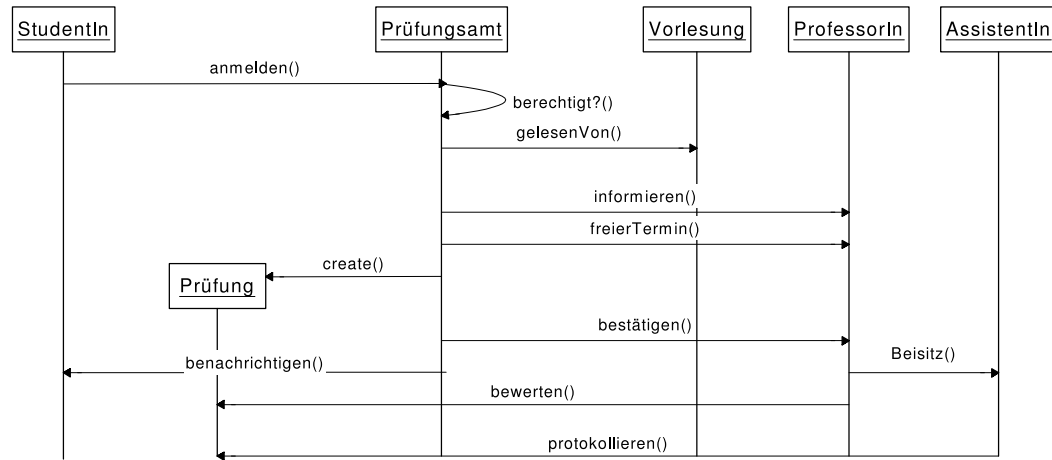












- ER und UML sind weit verbreitete Arten der konzeptuellen Modellierung
- Die Konzepte sind nicht allzu schwer zu begreifen, d.h. der Sachverhalt kann auch mit Informatiklaien (die den Fachbereich kennen) diskutiert werden
- Entwurf ist auch immer etwas subjektives, es gibt viele verschiedene Wege einen Sachverhalt zu modellieren (sich für den passendsten zu entscheiden ist eine Sache der Erfahrung)
- Die Qualität eines konzeptuellen Schemas kann auf der logischen Ebene noch einmal geprüft werden
- Das konzeptuelle Modell wird schließlich ins relationale Modell überführt