Datenbanksysteme

Kap 8: Entwurfspraxis – ER-Modellierung

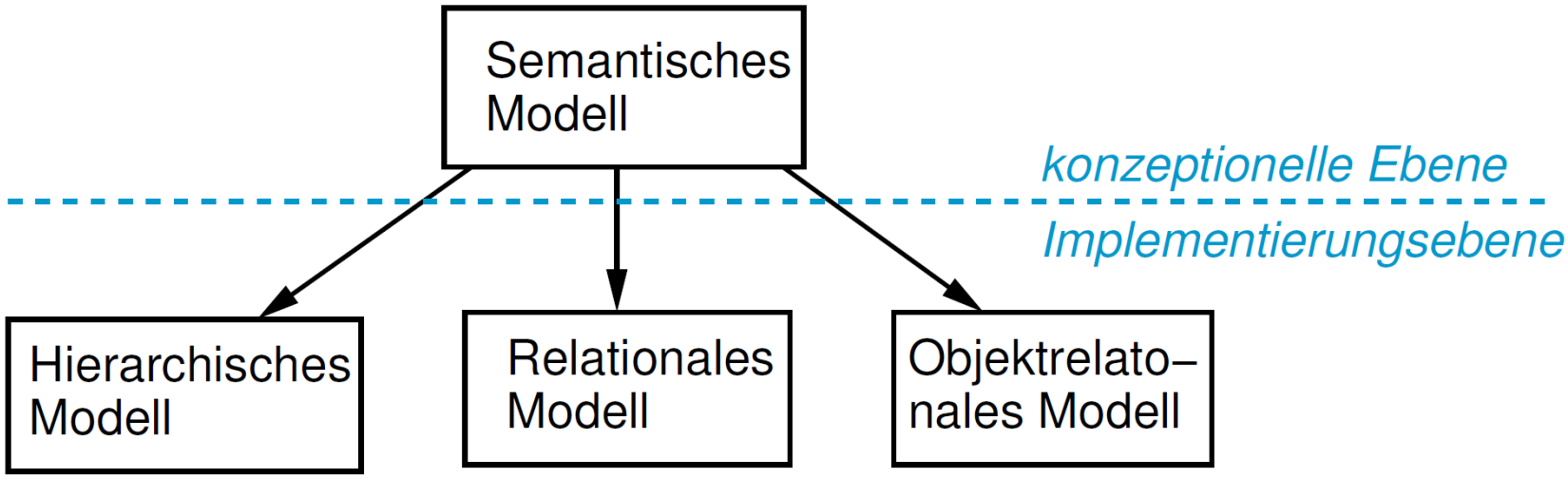
Überblick

- Letztes Kapitel
  - Formale Charakterisierung "schlechter" Datenbankschemata
  - Reparatur durch Normalisierung
    - Funktionale Abhängigkeiten
    - Normalformen
- Dieses Kapitel
  - Entwurfsmethodik auf semantischer Ebene
  - Führt (in den meisten Fällen) a priori zu einem "vernünftigen" Datenbankschema

Ebenen des Datenbankentwurfs (Wiederholung)

- Konzeptionelle Ebene
  - Logische Gesamtsicht des Anwenders auf die Daten
  - Unabhängig vom eingesetzten DBS-Typ
- Implementierungsebene
  - Konzeptionelle Datenstrukturen im Rahmen des eingesetzten DBS
  - Bei relationalem DBS z.B. Tabellen
- Physische Ebene
  - Konkrete Implementierung der Strukturen im Rahmen des eingesetzten DBS
  - Betrachtete Strukturen: Datenblöcke, Zeiger, Indexstrukturen

Semantischer Ansatz



- Versucht mehr von Daten-Bedeutung (Semantik) zu erfassen
  - Modelliert auf konzeptioneller Ebene
- Unabhängig vom eingesetzten Datenbank-System
  - Kann in verschiedenen DBS-Typen implementiert werden
- Bei Übergang zu Implementierungsebene geht Information verloren
  - Designschritt nicht reversibel

Basiselemente Semantischer Datenmodelle

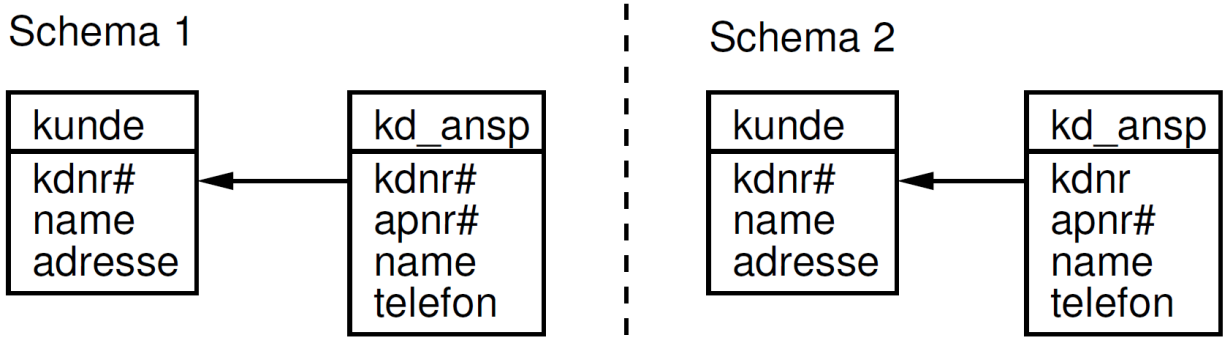
- Entity – unterscheidbares "Real World"-Objekt
- Property - Eigenschaft, die ein Objekt beschreibt
- Relationship - Zusammenhang zwischen Entitys

Semantische Modelle unterscheiden sich

- in der Art der unterstützten Relationships und den Abstraktionsmechanismen für Relationships (z.B. Abhängigkeit, Aggregation, Vererbung)
- in der Darstellung der Basiselemente und insbesondere der Relationships, z.B. durch
  - spezielle Diagrammsymbole und Linien (Entity-Relationship Modell)
  - Darstellung als Funktionen (Funktionales Datenmodell)

Basiselemente im relationalen Modell

- Auch Relationales Modell hat diese Basiselemente:
  - Entity: Relation
  - Property: Attribut, Primary Key Constraint
  - Relationship: Foreign Key Constraint
- Semantik aber unzureichend dargestellt



  - Tabelle *kd_ansp* ist abhängig von *kunde*, d.h. repräsentiert eigentlich eine Eigenschaft von Kunde
  - Zusammenhang in Schema 1 nur implizit repräsentiert (Wodurch?)
  - Diese Bedeutung ist in Schema 2 gar nicht repräsentiert

Vor- und Nachteile semantischer Modelle

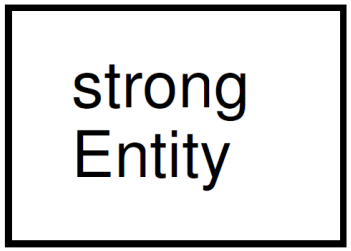
- Vorteile
  - Intuitiver verständlich (auch für Nichtexperten!)
  - Leichtere Modellierung durch größere Nähe zur "realen Welt"
  - Designer wird von Details der DBS entlastet (CASE-Tools)
  - Grafische Notation (Diagramme) anschaulich und (wenn grob vereinfacht) "pflichtenhefttauglich"
- Nachteile
  - Diagramme bei größeren Modellen nicht mehr praktikabel
  - Schritt zu SQL-DDL ist irreversibel
    - Kein "Reverse Engineering" möglich
    - Änderungen an implementiertem Modell schwierig
  - Geringe Unterstützung für Constraints und Tuning-Parameter
  - Unterscheidung Entity/Relationship oft künstlich
    - Direkte relationale Modellierung manchmal intuitiver

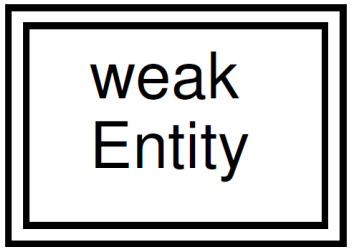
Entity-Relationship-Modell

- Verbreitetstes semantisches Modell
- 1976 von Chen vorgeschlagen
- Beinhaltet bestimmte Diagrammnotation (ER Diagramm)
- Später erweitert worden um Varianten der Vererbung (Spezialisierung, Generalisierung, Kategorien)
- Erweiterung des ER-Modells auf allgemeine Softwareentwicklung (nicht nur DB-Design) in Form von OMT und UML
  - UML (Unified Modelling "Language") benutzt aber andere Diagrammsymbole und Begriffe

Entity

- Unterscheidung von zwei Arten von Entitys
- Strong (regular) Entity
  - Eigenständiges Objekt
  - Kann unabhängig von anderen Objekten existieren
- Weak Entity
  - Abhängiges Objekt;
  - Kann nur existieren, wenn ein Objekt aus anderer Entity existiert
- Diagramm-Symbole





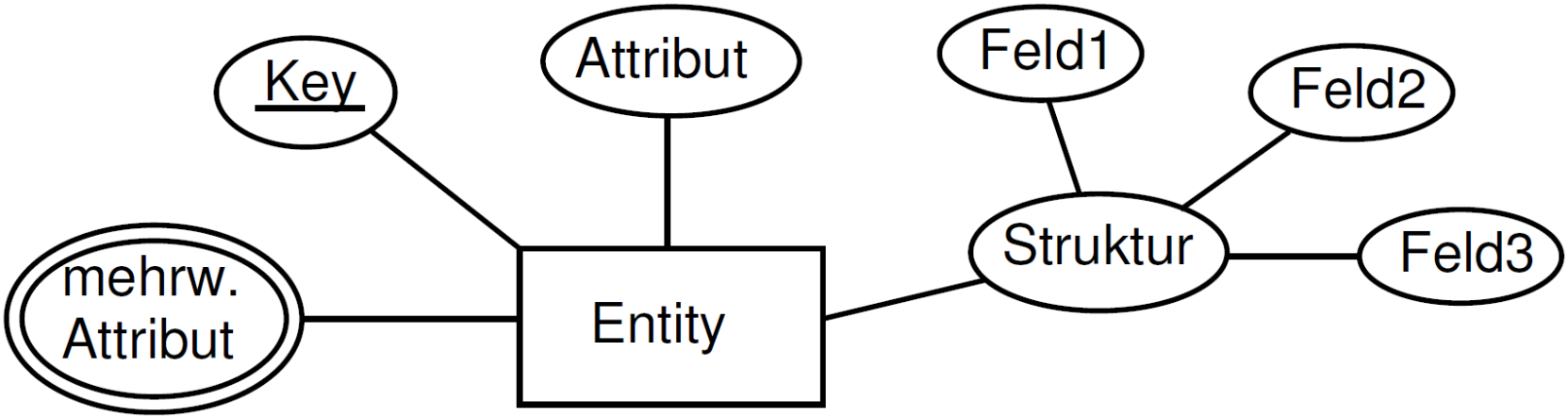
Entity

- Beispiel

  - Ansprechpartner ist kein unabhängiges Objekt
  - Existiert nur, wenn entsprechender Kunde auch existiert
- Bemerkungen
  - Es ist oft nicht offensichtlich, ob eine Entity "weak" ist
  - Z.B. kann im Hersteller/Produkt Beispiel die Entity *produkt* sowohl als strong, als auch als weak aufgefasst werden (Warum?)
  - Ob weak oder strong hängt von logischer Sicht auf die Daten ab
    → Semantik der Entitys wird modelliert

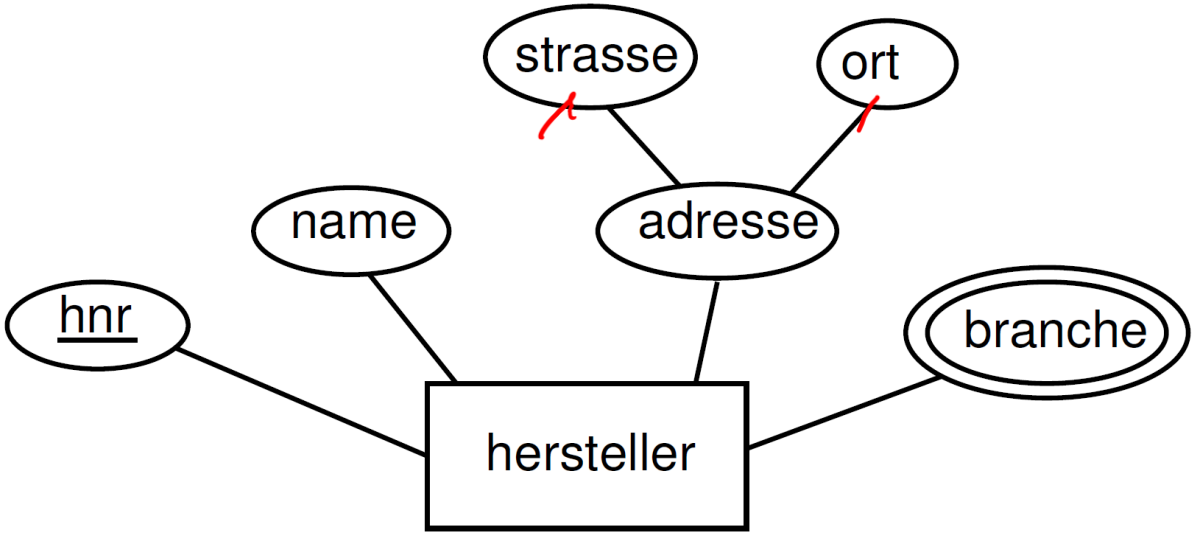
Property (Attribut)

- Attribute werden durch Blasen dargestellt
  - Zusammengesetzte Attribute (Strukturen) durch Zerlegung in weitere Blasen
  - Mehrwertige (mengenwertige) Attribute erhalten doppelten Rand
  - Schlüsselattribute werden unterstrichen



Property (Attribut)

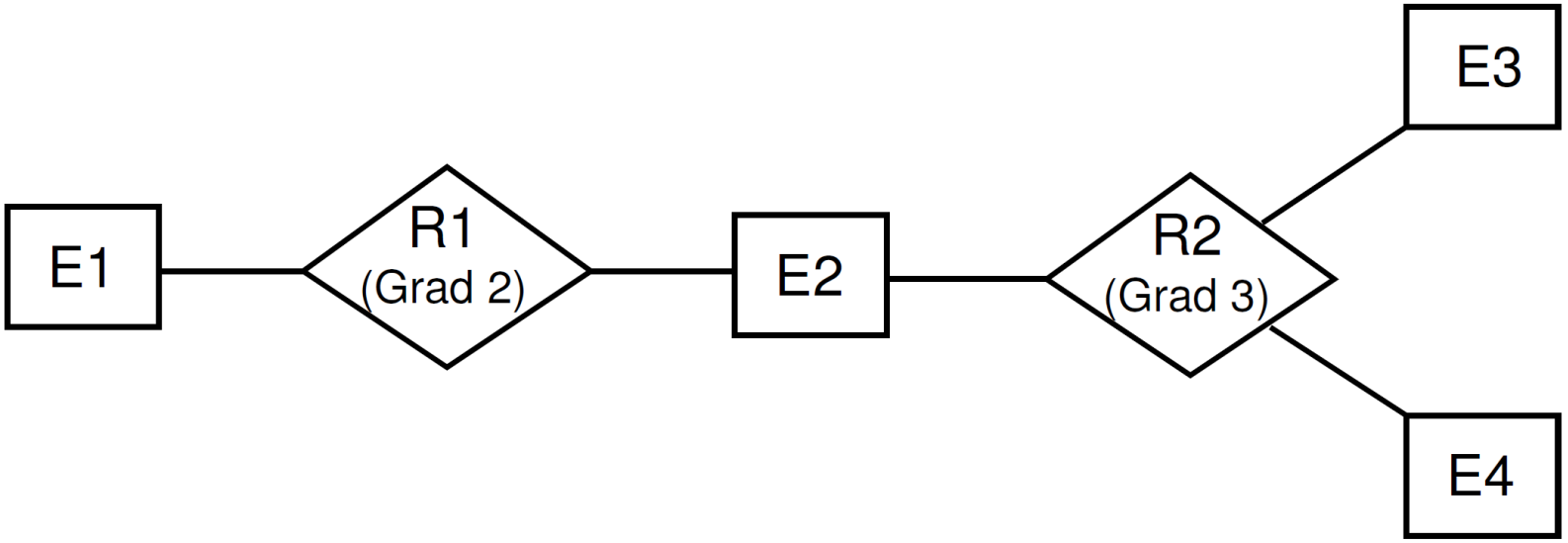
- Beispiel



- Bemerkungen
  - Zusammengesetzte und mehrwertige Attribute sind eigentlich überflüssig (siehe Diskussion zum Thema 1NF)
  - Mehrwertige Attribute machen aber Semantik klarer
  - Bei großer Zahl von Attributen sind Blasen nicht mehr darstellbar
    → als Spaltenvektor darstellen (vgl. UML-Klassendiagramm)

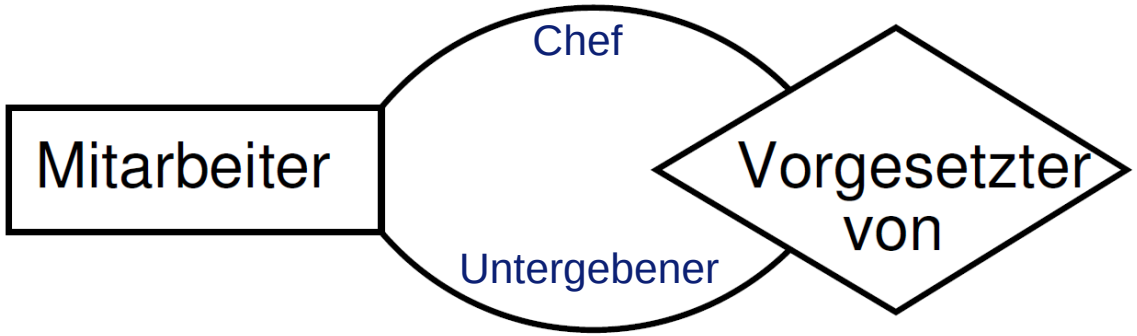
Relationship

- Stellt Zusammenhang zwischen Entitys her
- Darstellung durch Raute mit Linien zu beteiligten Entitys
- Anzahl beteiligter Entitys heißt Grad der Relationship

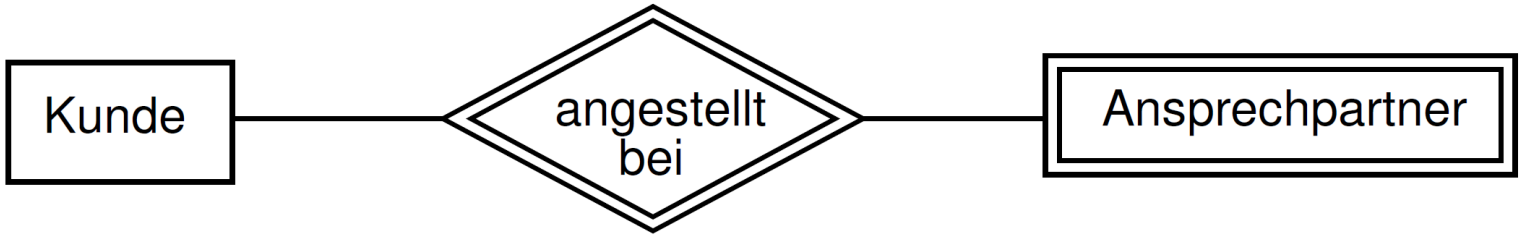


Relationship

- Reflexive Relationships
  - Relationship kann Objekte derselben Entity miteinander verknüpfen
  - Mit Rollennamen kann man die Rolle der beteiligten Entitys klärer

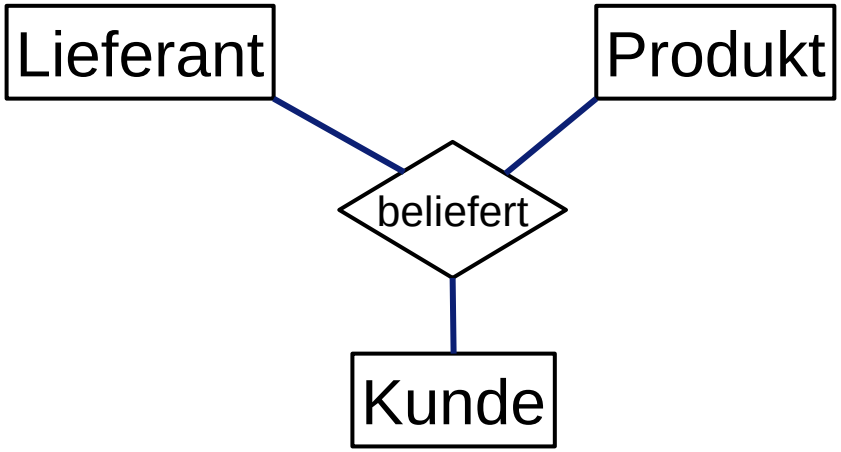


- Relationship zwischen strong und weak Entitys
  - Gekennzeichnet durch doppelten Rahmen



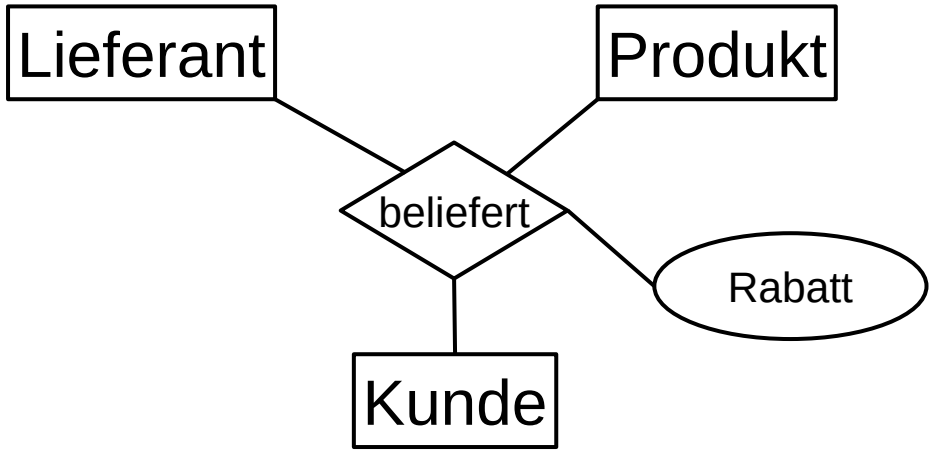
Relationship höheren Grads

- An einer Relationship können mehr als 2 Entitys beteiligt sein
- Beispiele:



Attribute von Relationships

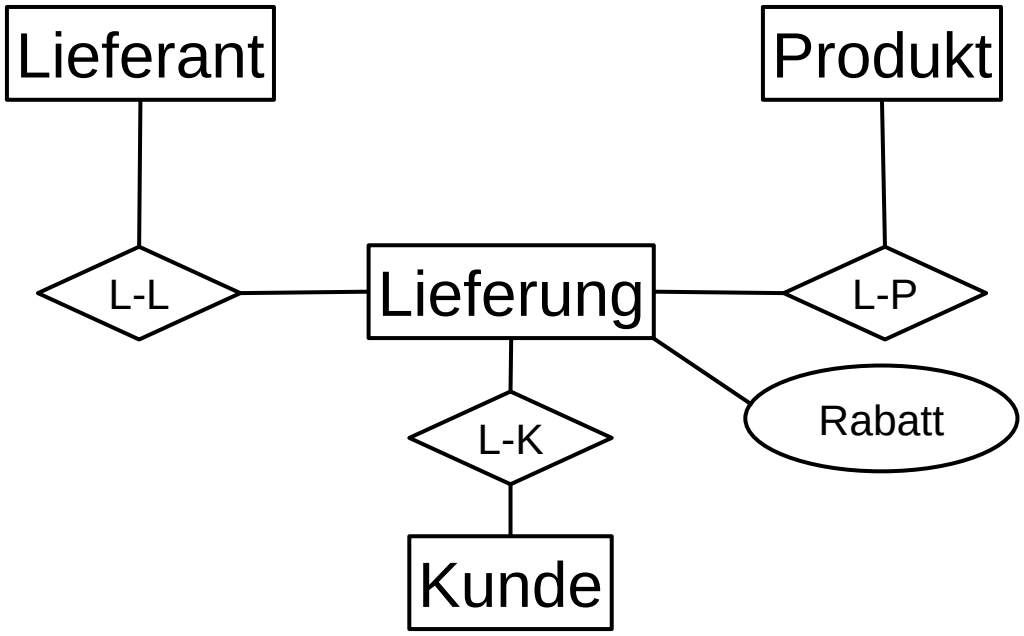
- Relationships können auch Attribute haben



- Rabatt ist keine allgemeine Eigenschaft von Lieferant, Produkt oder Kunde
  - Z.B. kann ein Kunde spezifische Rabatte je nach Lieferant und Produkt bekommen
- Verwischt Grenze zwischen Entitys und Relationships
  - Chen spricht von "Relationship Relation"

Alternative Modellierung

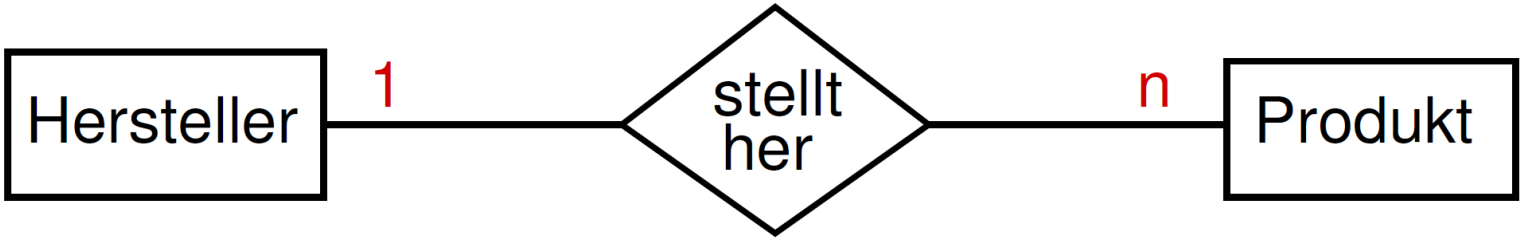
- Ersetze k-gradige Relationship durch neue Entity und k binäre Relationships
  - Hänge Relationship-Attribute an neue Entity



- Ergibt gleichwertiges, aber komplexeres Schema

Kardinalität von Relationships

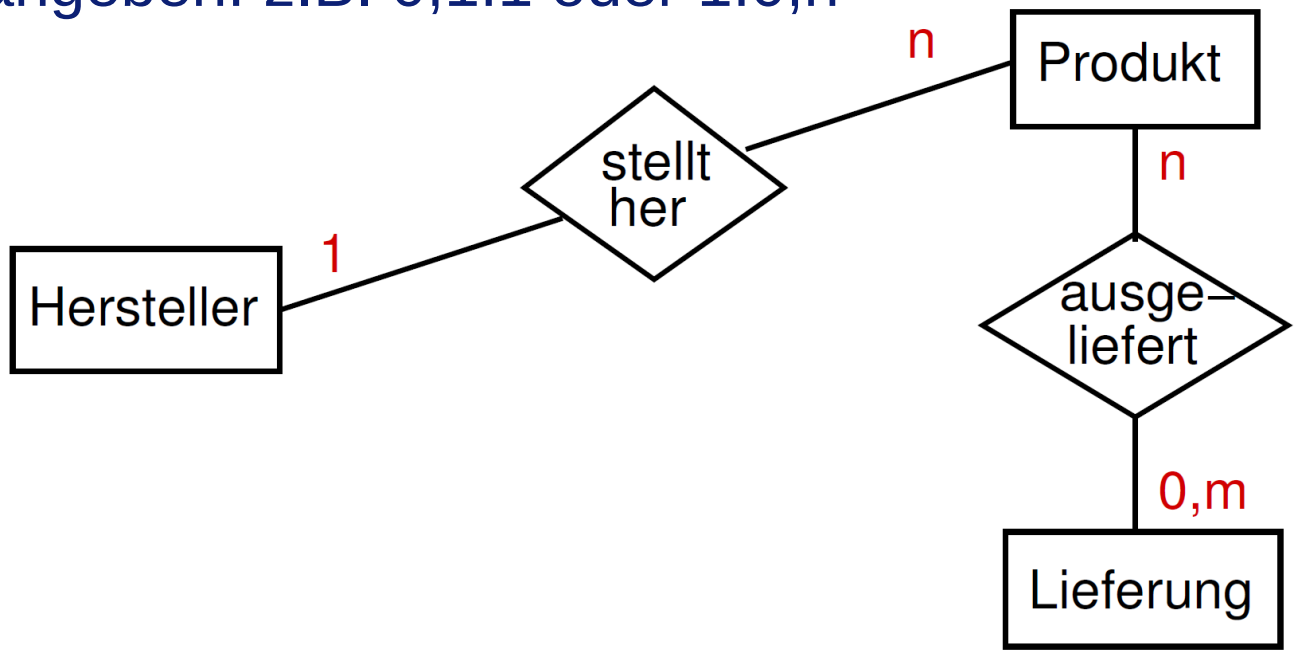
- Gibt an, mit wie vielen verschiedenen Elementen eine Entity-Instanz über die Relationship verknüpft sein kann
- Durch Zahlen an Verbindungslinien angegeben
- Beispiel:



  - 1:n-Beziehung zwischen Hersteller:Produkt
  - Ein Hersteller stellt mehrere (n > 0) Produkte her
  - Ein Produkt hat einen (1) Hersteller

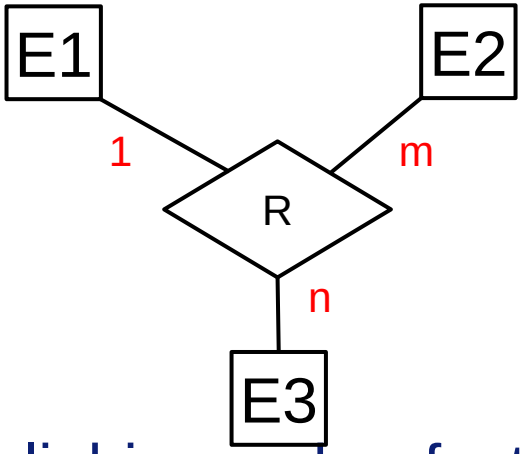
Kardinalitätstypen

- 1:1 (One-to-One)
- 1:n (One-to-Many)
- n:m (Many-to-Many)
- Für n,m werden Werte > 0 angenommen
  - Wenn auch kein Element zulässig sein soll, explizit Null mit angeben: z.B. 0,1:1 oder 1:0,n



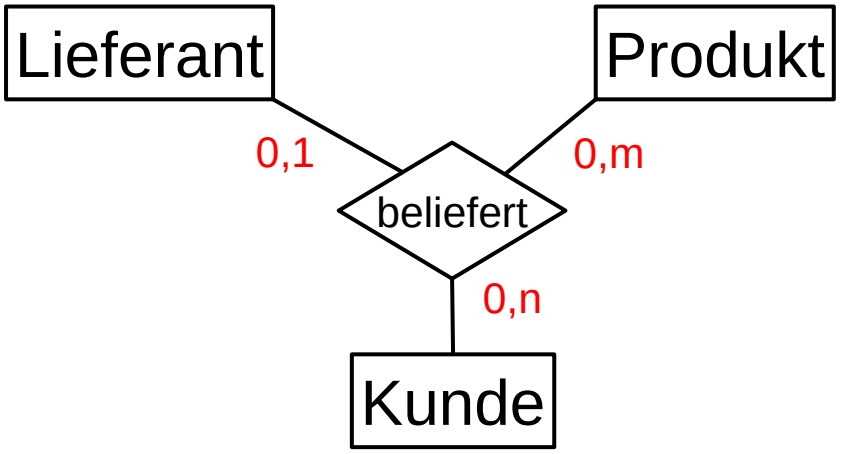
Kardinalitäten bei k-gradigen Relationships

- Welche Semantik haben Kardinalitäten hier?



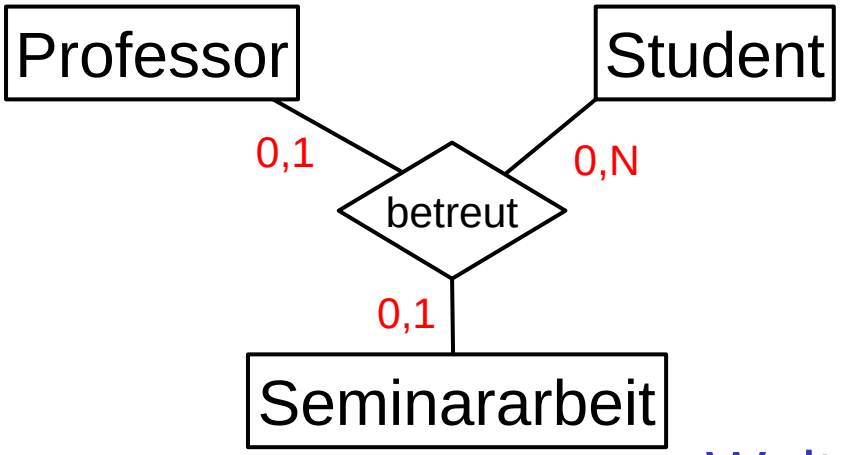
  - Betrachte ein beliebiges, aber festes Instanz-Tupel von jeweils k-1 Entitys: $(e_1, \ldots, e_{i-1}, e_{i+1}, \ldots, e_k)$
  - "Gegenüberliegende" Funktionalität
    - =1: wenn genau eine Beziehung zu einer Instanz von E_i erlaubt ist
    - = n: wenn Beziehungen zu mindestens einer Instanz von E_i erlaubt sind
    - 0,1 bzw. 0,n, wenn auch gar keine Beziehungen erlaubt sind
  - Kardinalitäten von 2-stelligen Relationships sind ein Spezialfall dieser Definition

Beispiel



- Ein bestimmter Lieferant hat für ein bestimmtes Produkt beliebig viele Kunden (0,n)
- Ein bestimmter Lieferant liefert einem bestimmten Kunden beliebig viele Produkte (0,m)
- Ein bestimmter Kunde erhält ein bestimmtes Produkt nur von maximal einem Lieferanten oder gar nicht (0,1)

Noch ein Beispiel



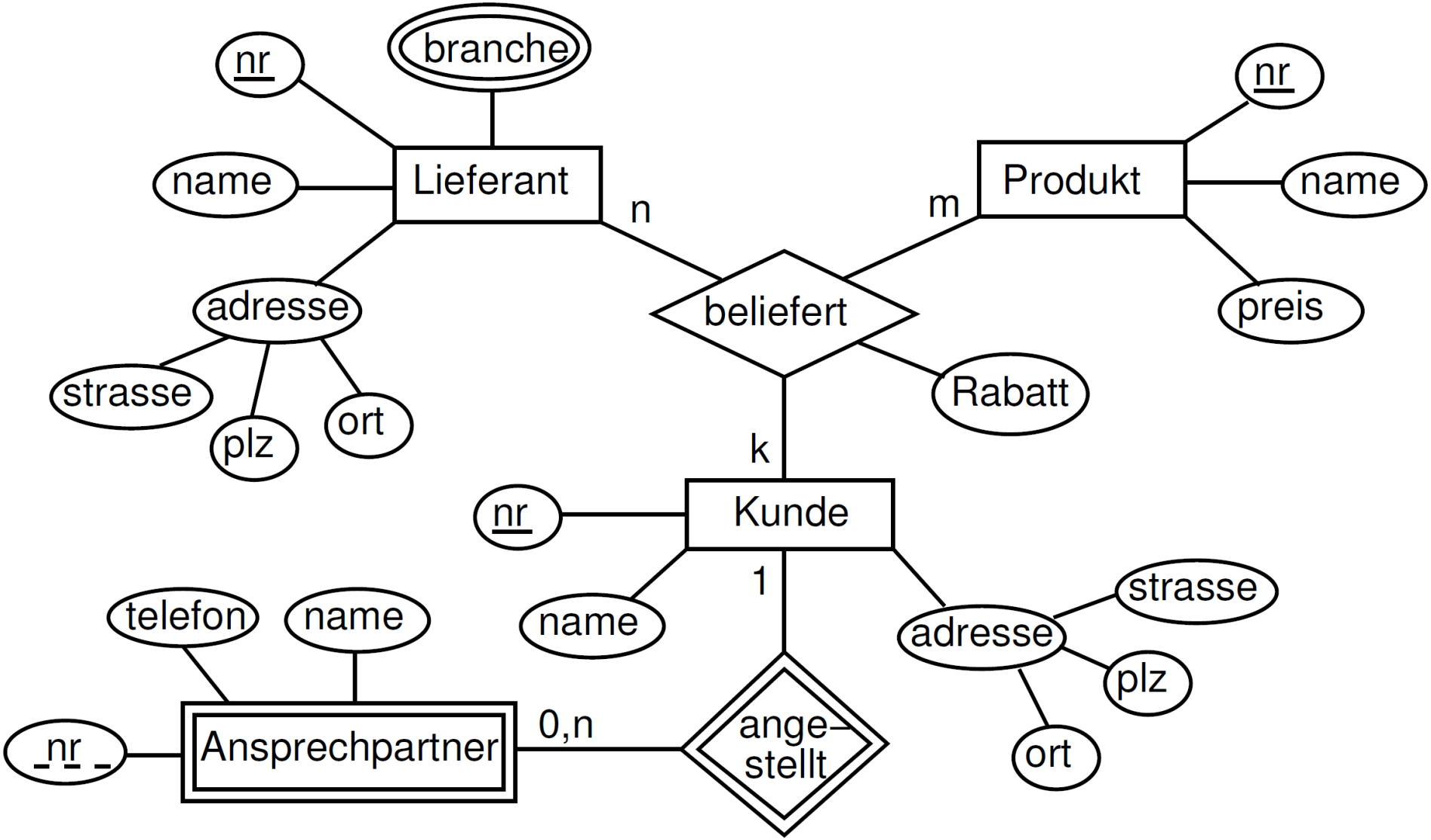
Einschränkungen:

- Keine "Groupies"
  - Studenten dürfen bei demselben Professor nur max. ein Seminarthema ableisten
- Kein "Reuse"
  - Studenten dürfen dasselbe Seminarthema nur einmal bearbeiten (d.h. nicht bei einem anderen Professor dasselbe Thema nochmal bearbeiten)

Weiterhin möglich:

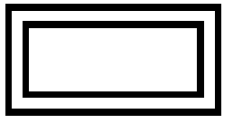
- Professoren können das gleiche Seminarthema wiederverwenden (d.h. an mehrere Studenten verteilen)
- Ein Thema kann von mehreren Professoren vergeben werden (aber nur an unterschiedliche Studenten!)

Komplettes Beispiel

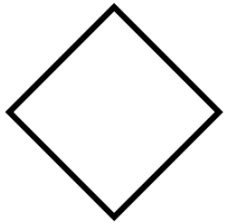


Zusammenfassung ER-Notation

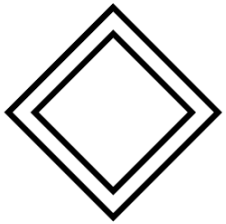
strong Entity



weak Entity



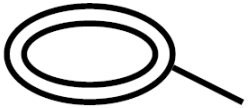
Relationship



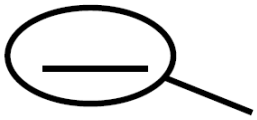
Zuordnung
strong/weak Entity

Attribut

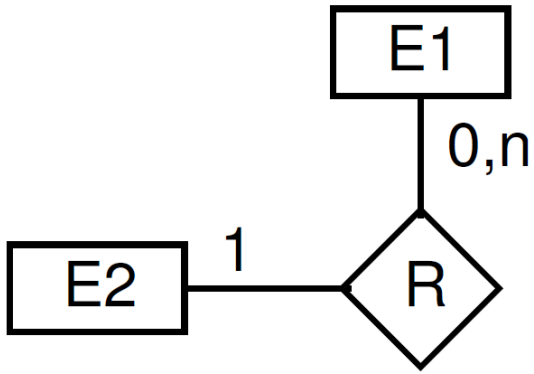
partieller Schlüssel
einer weak Entity



mehrwertiges
Attribut



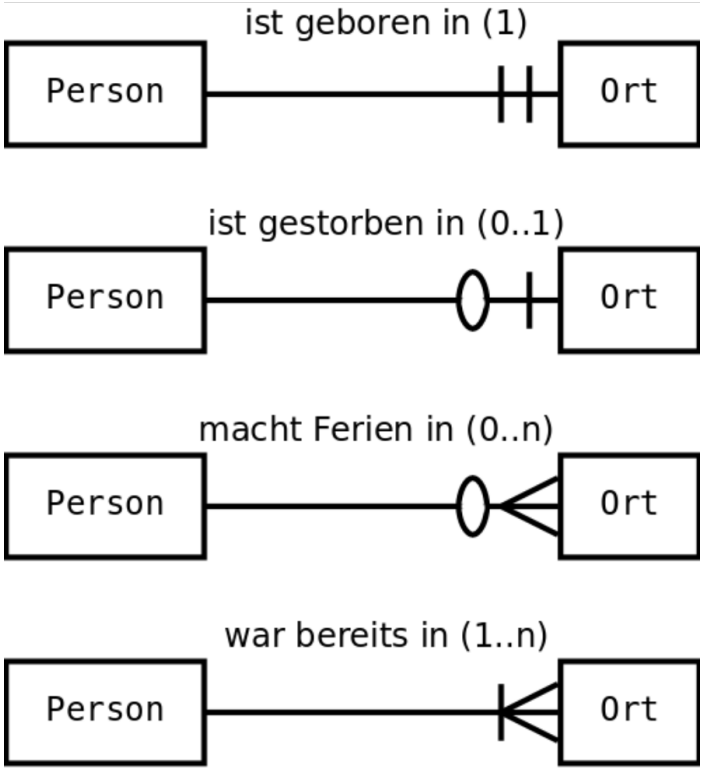
Schlüssel–
attribut



Relationship
mit Kardinalitäten

Alternative Notationen

- Martin-Notation (Krähenfuß-Notation)
  - Vereinfachte Darstellung von Relationships als Linien mit Beziehungsnamen
  - Spezielle Symbole für Kardinalitäten (jeweils min und max)
    - | (eins)
    - O (null)
    - ⋲ (n)

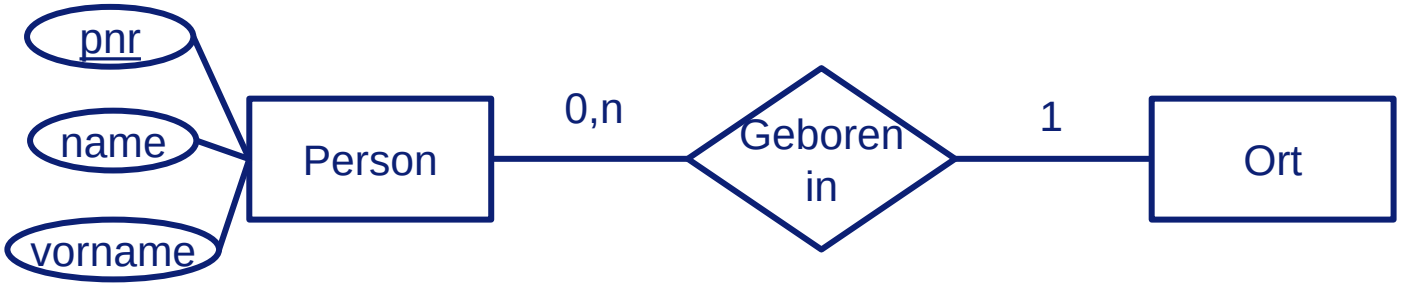


(Quelle: https://de.wikipedia.org/wiki/Martin-Notation)

Alternative Notationen

- UML-Klassendiagramme
  - Entity ⇔ Klasse
  - Relationship ⇔ Assoziation
  - Ökonomischere Darstellung von Attributen
  - UML ist mächtiger (z.B. Methoden, Sichtbarkeiten, spezielle Beziehungen wie Aggregation, IsA etc.)

ER:



UML:

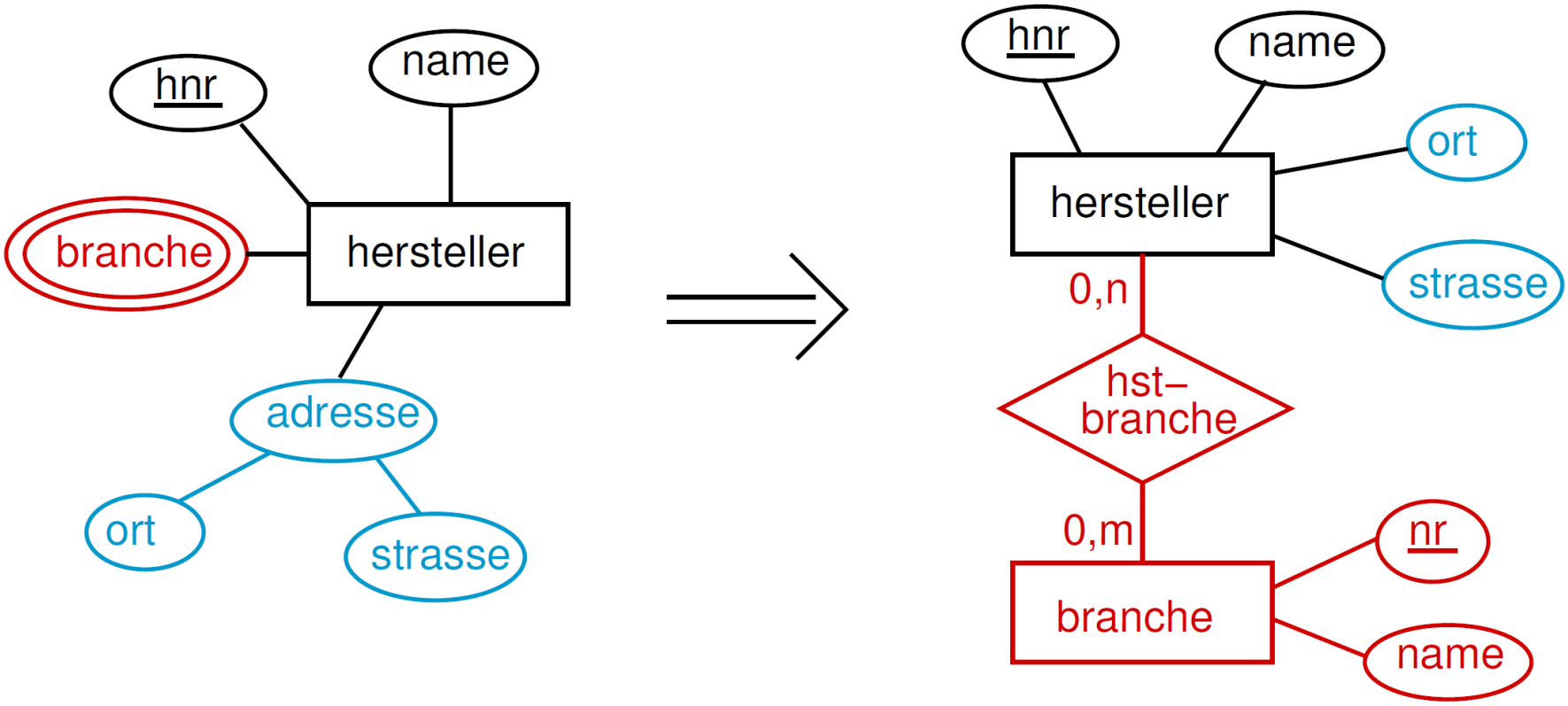
Umwandlung ER in Relationales Schema

- Allgemeines Vorgehen
  - Jede Entity wird Relation
    - Attribute und Primary Key (PK) werden übernommen
  - Jede Relationship wird Relation
    - Primary Key = alle PK's beteiligter Entitys

- Feinheiten
  - mehrwertige und zusammengesetzte Attribute
  - Behandlung von weak Entitys
  - Nicht alle Relationships brauchen eigene Relation
    - abhängig von Kardinalität der Relationship
  - Art der Foreign Key Constraints

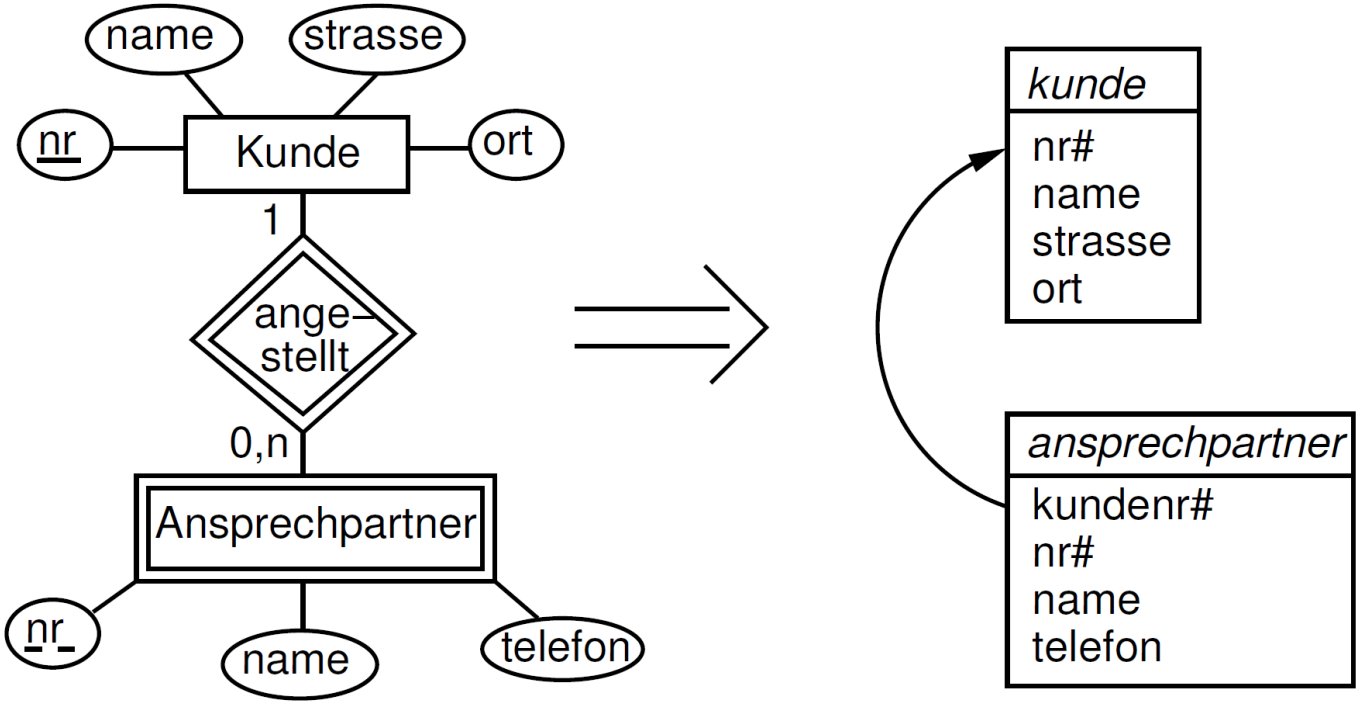
Strong Entitys

- Zusammengesetzte und mehrwertige Attribute können schon auf der ER-Ebene nach dem Muster der Normalisierung (siehe erste Normalform) umgeformt werden



Weak Entitys (1)

- Eigene Relation
  - Primary Key = eigene Key Attribute + PK strong Entity
  - Oder eigener künstlicher Schlüssel, wenn zusammengesetzte Keys unerwünscht (nicht empfohlen - Warum?)



Weak Entitys (2)

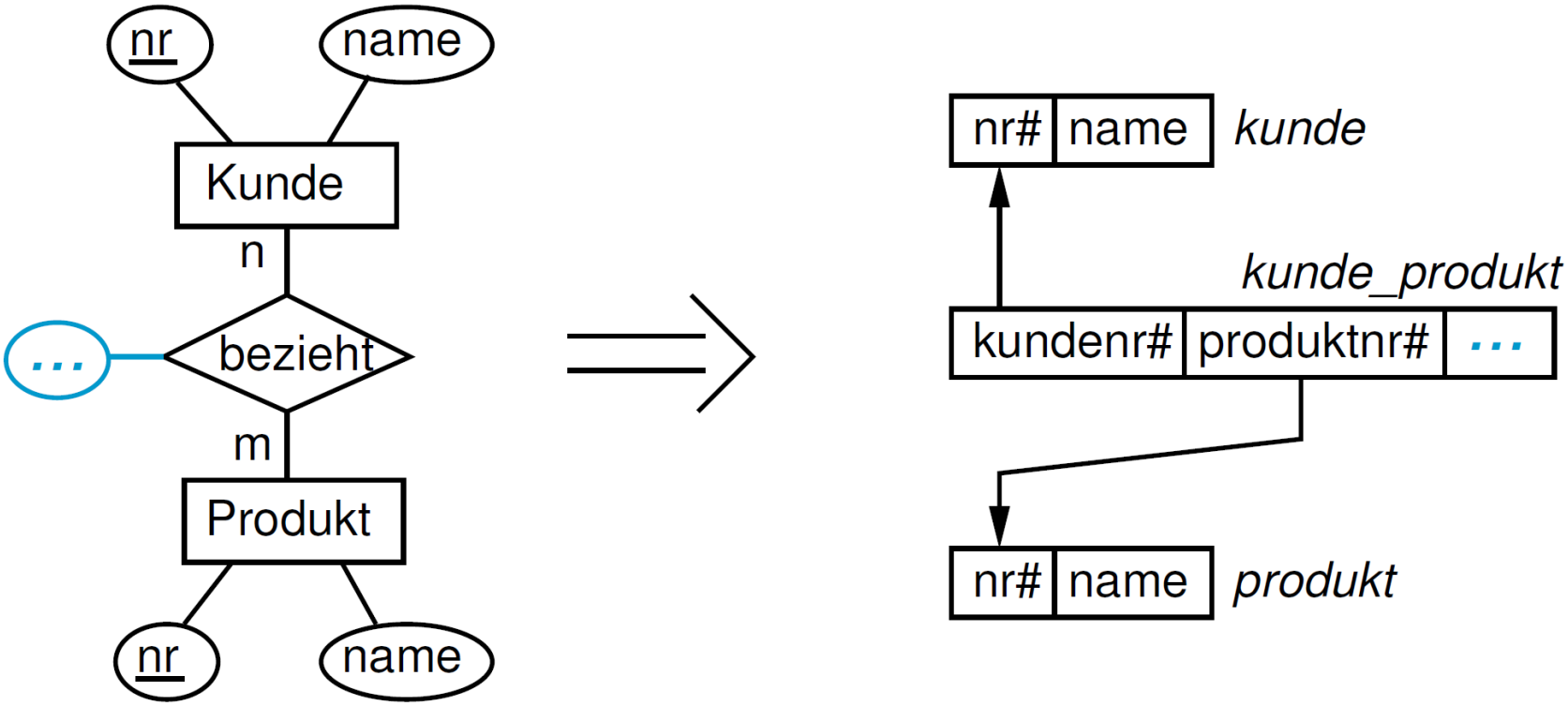
- Abhängige Entity ist an andere strong Entity gebunden
  - Bei Foreign Key Constraint folgende Optionen nötig:
  - `ON DELETE CASCADE`
    - Bewirkt Löschung von Ansprechpartner, wenn referenzierter Kunde gelöscht wird
  - `ON UPDATE CASCADE`
    - Ändert Fremdschlüssel in Ansprechpartner mit bei Schlüsseländerung des referenzierten Kunden

```
CREATE TABLE kunde (
 nr      INT8,
 name    VARCHAR(30),
 strasse VARCHAR(30),
 ort     VARCHAR(30),
 PRIMARY KEY (nr)
);
```

```
CREATE TABLE ansprechpartner (
 kundenr INT8  REFERENCES kunde(nr)
   ON DELETE CASCADE ON UPDATE CASCADE,
 nr      INT,
 name    VARCHAR(30),
 telefon VARCHAR(30),
 PRIMARY KEY (kundenr, nr)
);
```

Many-to-Many Relationship

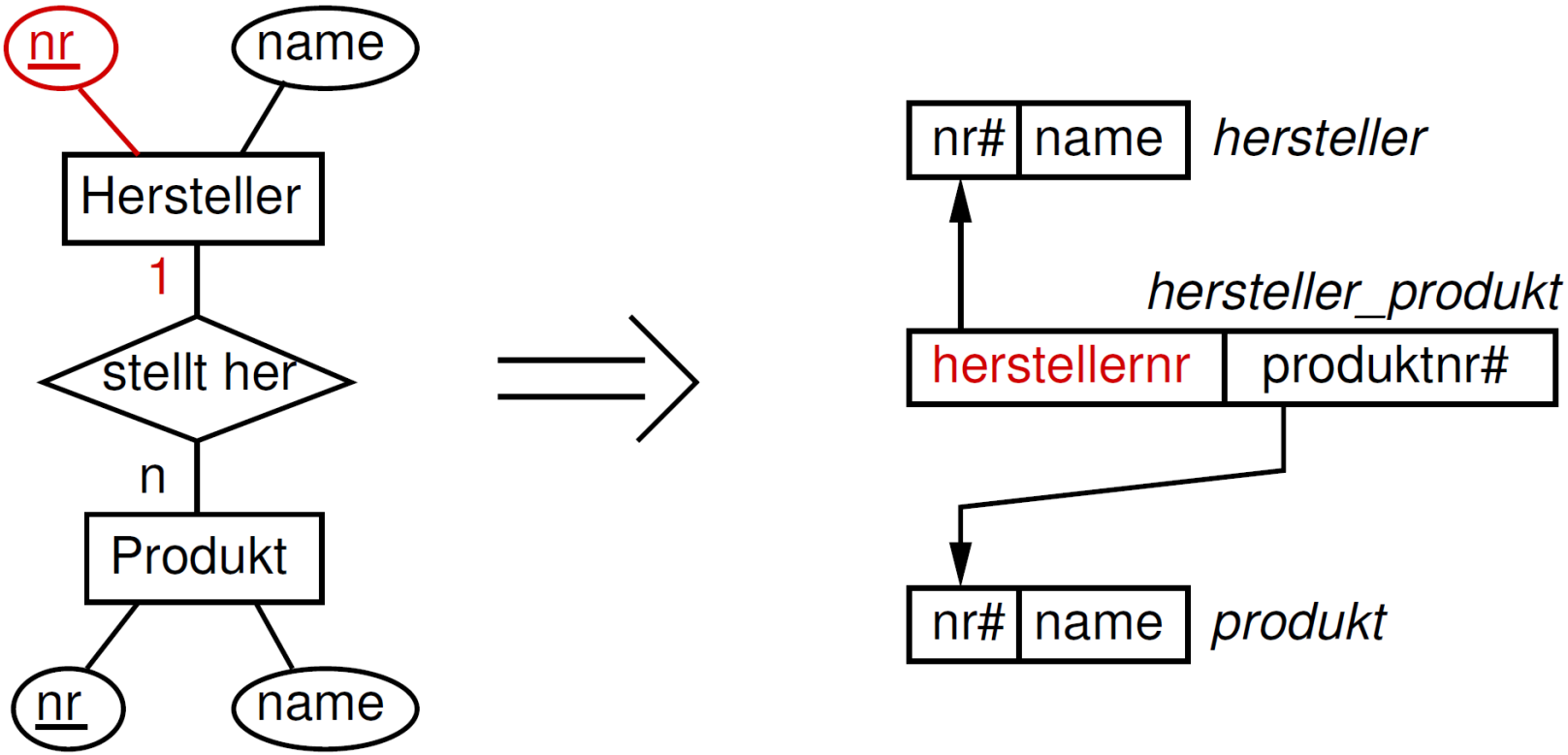
- n:m Relationships werden eine eigene Relation
- Primary Key = PK's aller beteiligten Entitys



- Foreign Key Constraints mit on `update cascade` Option

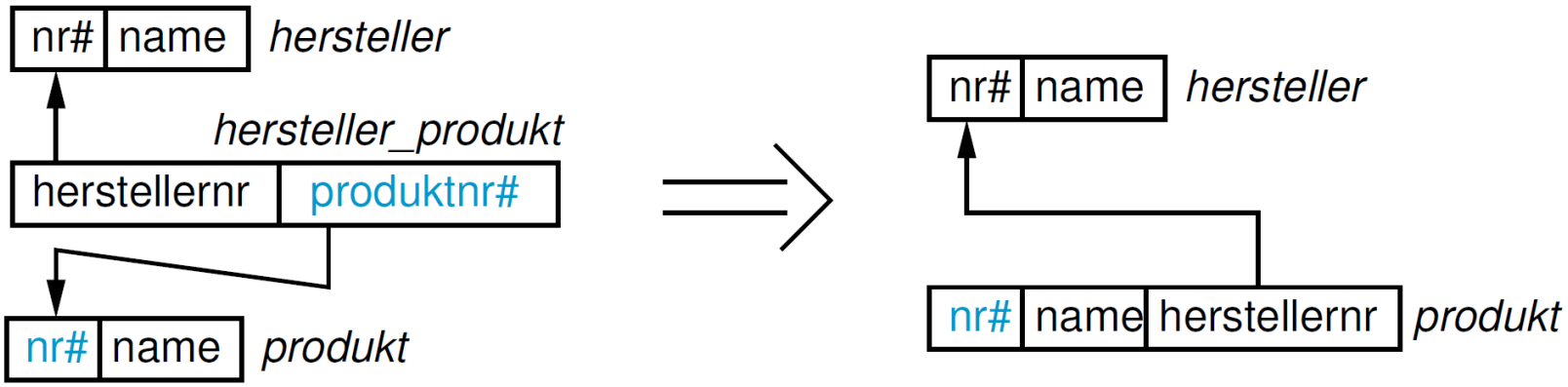
One-to-Many Relationship (1)

- 1:n Relationship kann wie n:m Relationship umgesetzt werden
  - Unterschied: PK der Entity am "1-Ende" nicht in PK der "Relationship-Relation" mit aufnehmen (Warum?)



One-to-Many Relationship (2)

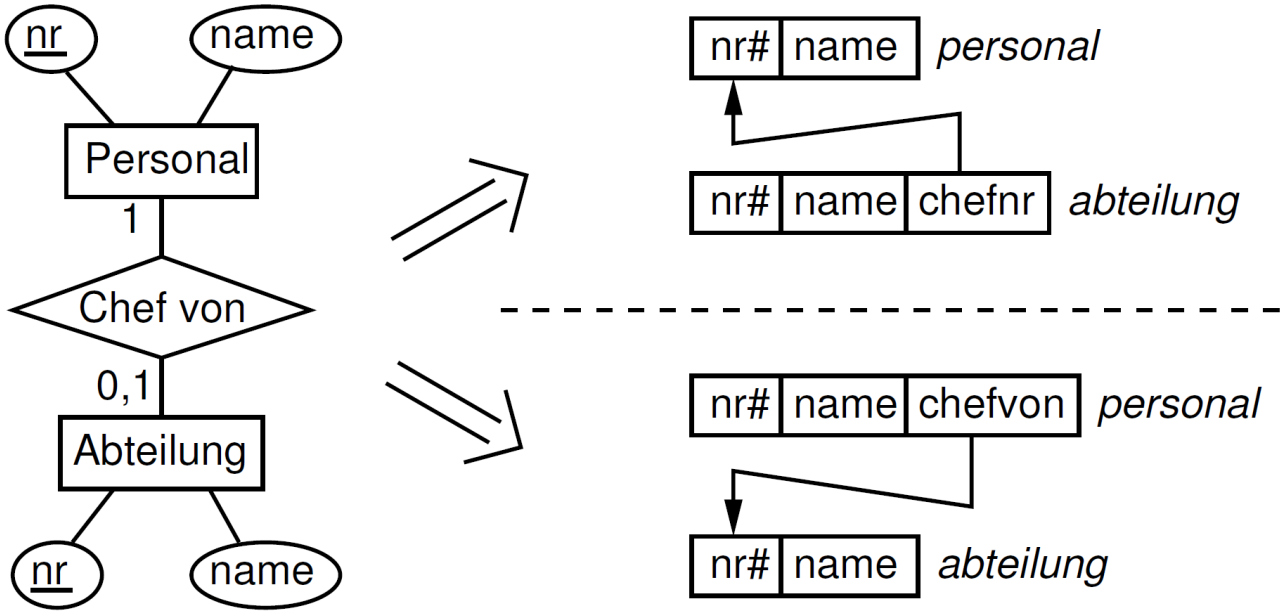
- Beobachtung: separate Relation *hersteller_produkt* unnötig
  - Zusammenlegung mit *produkt* ergibt:



- Vorteile
  - Weniger Relationen
  - Klarere Semantik: Hersteller Eigenschaft von Produkt
  - Aber: wenn oft kein Hersteller bekannt, vermeidet linke Lösung NULL-Werte in Foreign Key Feld

One-to-One Relationship

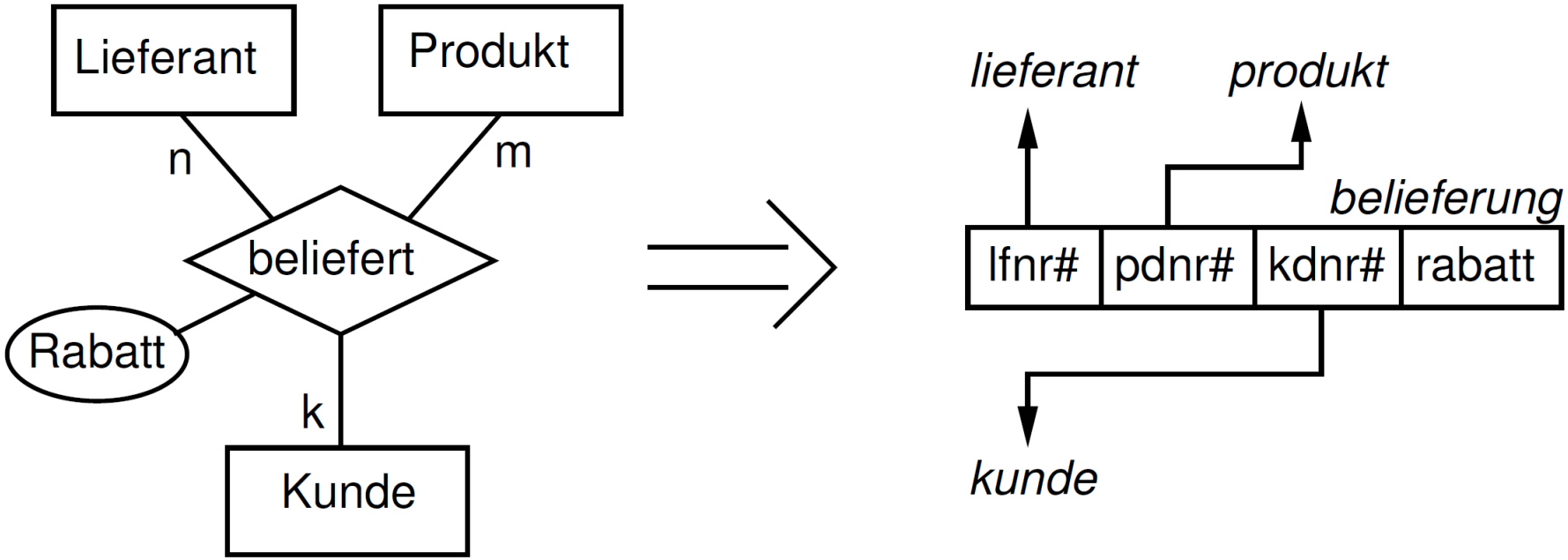
- Auch bei 1:1 Relationship keine eigene Relation nötig
  - PK einer Entity als Foreign Key in andere Entity aufnehmen



- Obere Lösung ist besser (Warum?)
  - Regel: erweitere Tabelle am "0-Ende"

Relationship höheren Grades

- Eigene Relation
  - PK beteiligter Relationen als Foreign Keys
  - PK = PKs beteiligter Relationen mit Kardinalität > 1



Zusammenfassung: ER→Relational

ER-Modell	Relationales Modell
Entity	Relation
Einfaches Attribut	Attribut
Zusammengesetztes Attribut	Mehrere Attribute
Schlüsselattribut	Primary Key
Mehrwertiges Attribut	Relation mit Fremdschlüssel
1:1 oder 1:N Relationship	Fremdschlüssel in Relation auf Seite der höheren Kardinalität (Alternative: separate Relation)
N:M Relationship	Relation mit zwei Fremdschlüsseln
Relationship n-ten Grades	Relation mit n Fremdschlüsseln

Entwurfsfragen

- Anforderungen an Datenmodell
  - Vollständig
  - Minimal bzw. redundanzfrei
  - Einfach und verständlich

- Damit zusammenhängende Aspekte
  - Namenskonventionen
  - Auswahl des Elements (Entity, Attribut, Relationship)
  - Mehrwert (?) von ER versus Relational
  - Allgemeines Vorgehen (Top-Down, Bottom-Up)

Namenskonventionen

- Namen von Objekten des Modells sollen Bedeutung entsprechen →leichter verständlich
- Konventionen erleichtern Verwendung (leichter merkbar, Fremdschlüssel erkennbar)

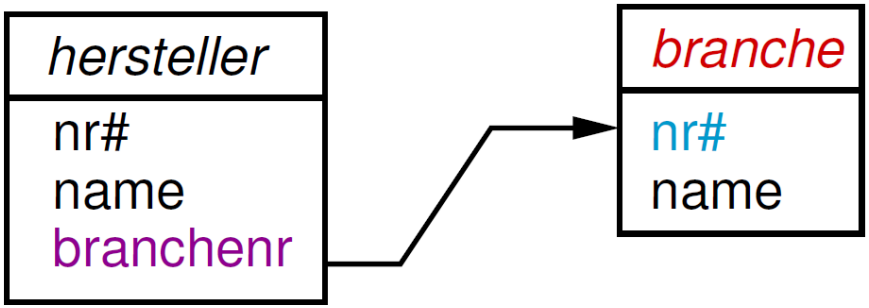
- Beipielkonventionen (1)
  - Entitys konsistent im Singular oder Plural.
  - Beides sinnvoll:

```
select * from kunden;
select kunde.nr,kunde.name from kunde ...;
```

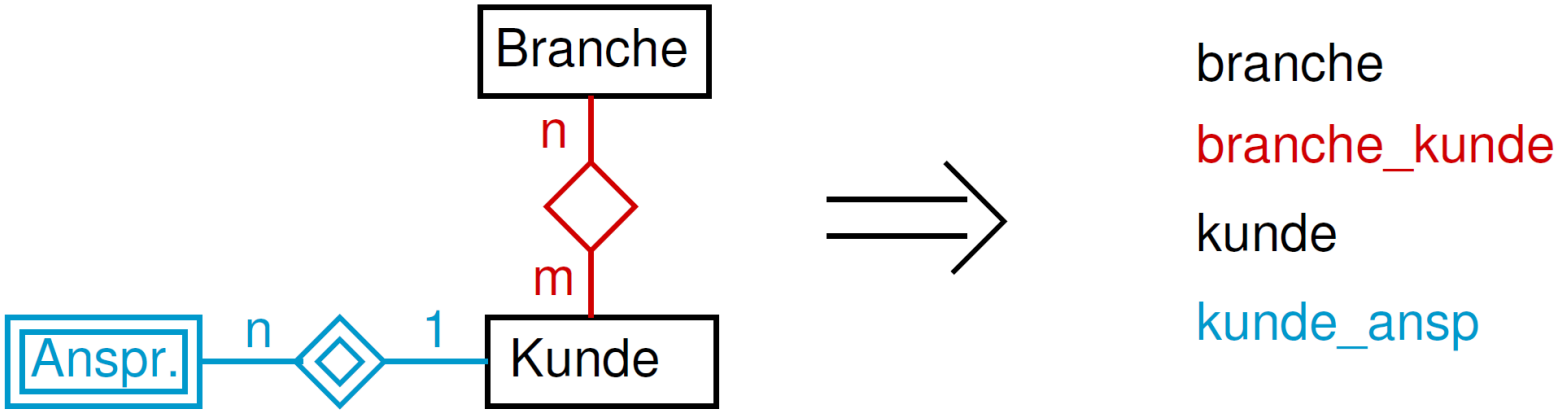
  - Übliche Konvention: Singular
  - Name von Key und Bedeutung einheitlich, z.B. `nr` und `name`

Namenskonventionen

- Beispielkonventionen (2)
  - Name Fremdschlüsselattribut = referenzierte Tabelle + PK



  - Name Relationship-Relation zusammengesetzt aus Namen der beteiligten Entitys; ebenso bei weak Entitys

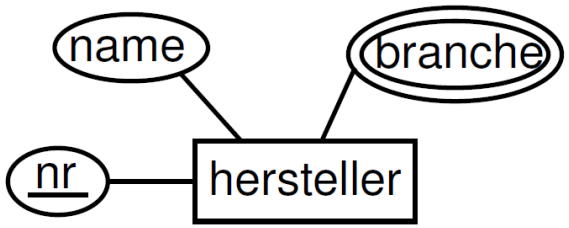


Wahl des Basiselements

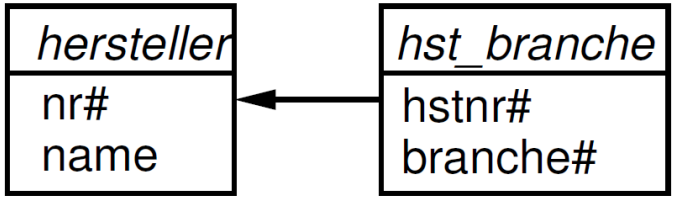
- Oft mehrere Möglichkeiten, einen Sachverhalt zu repräsentieren
  a) Attribut versus Entity und Relationship
  b) Relationship versus Entity
- Alternative a) ist echte Designfrage mit Auswirkungen auf
  – Dateneingabe und Datenkonsistenz.
- Alternative b) ist künstliches Problem im ER Modell
  – Im Relationalen Modell kein Unterschied
  – ER Modell unterstützt Foreign Keys nur implizit durch Relationships und weak Entitys
  – neuere Modelle und CASE-Tools erweitern ER um relationale Konzepte → Unterscheidung Entity/Relationship aufgehoben

Attribut versus Entity und Relationship (1)

- Betrachte Eigenschaft "Branche" eines Herstellers
- Lösung 1:



ER Modell

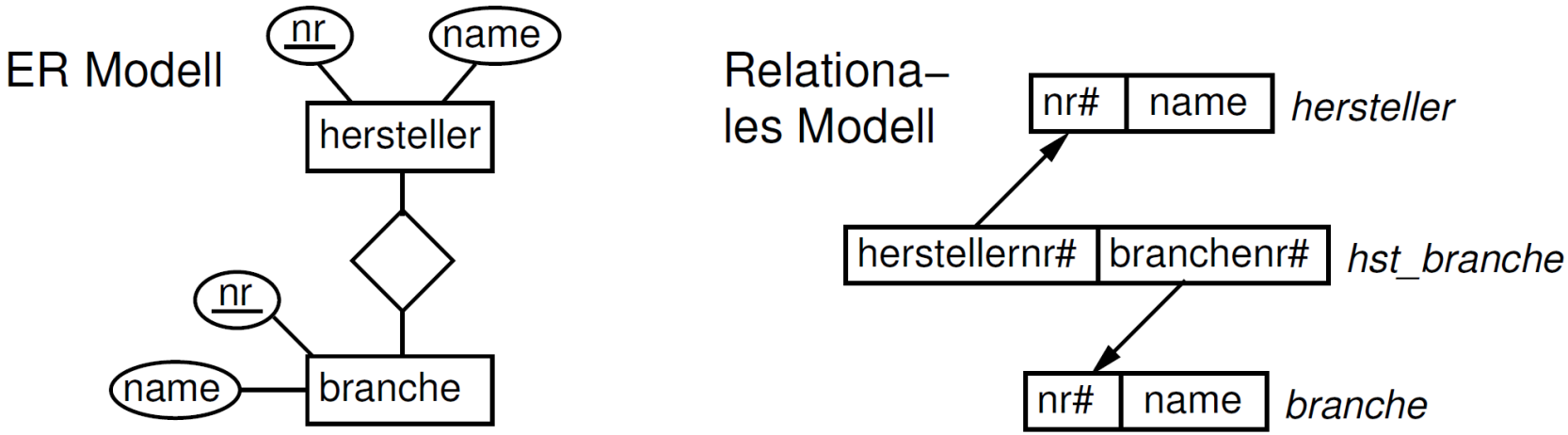


Relationales Modell

  - Branche als mehrwertiges Attribut modelliert
  - Freie Text-Eingabe für Branche möglich
    - Leichte Eingabe (keine Referenzdatenpflege)
    - Auswertung über Branche schwierig (z.B. Tippfehler)
  - Branchen nicht separat pflegbar sondern abhängig von Hersteller

Attribut versus Entity und Relationship (2)

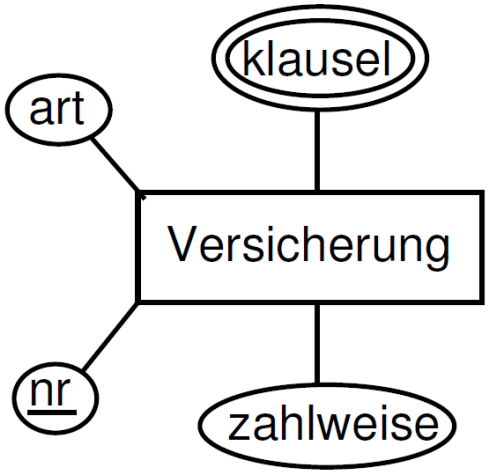
- Lösung 2:



- Branche als eigene, Hersteller-unabhängige Entity modelliert
- Keine freie Eingabe möglich, sondern Auswahlliste
  - Pflege separater Referenztabelle *branche* nötig
  - Auswertungen über Branchenschlüssel möglich

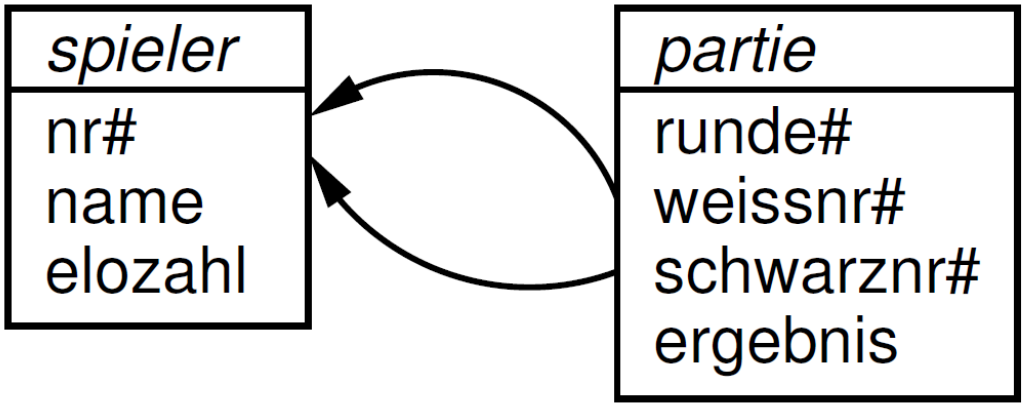
Attribut versus Entity und Relationship (3)

- Modellierungsalternative besteht nicht nur bei mehrwertigen, sondern bei allen Attributen
  - Verwende Referenztabelle, wenn Attributwerte nicht beliebig sind, sondern aus (konfigurierbarer!) Werteliste kommen sollen
  - Unterschied zu Domain-Constraint (Check-Constraint): Werteliste änderbar ohne Schemaänderung
- Beispiel
  - *art* und *klausel* sollten über Referenztabelle modelliert werden
  - *zahlweise* kann über Constraint modelliert werden:
    - Werte 1,2,3,6,12 sind fest
    - Werte tragen Bedeutung für Berechnungen



Entity versus Relationship (1)

- Betrachte Modellierung eines Schachturniers
- Relationales Modell ist offensichtlich

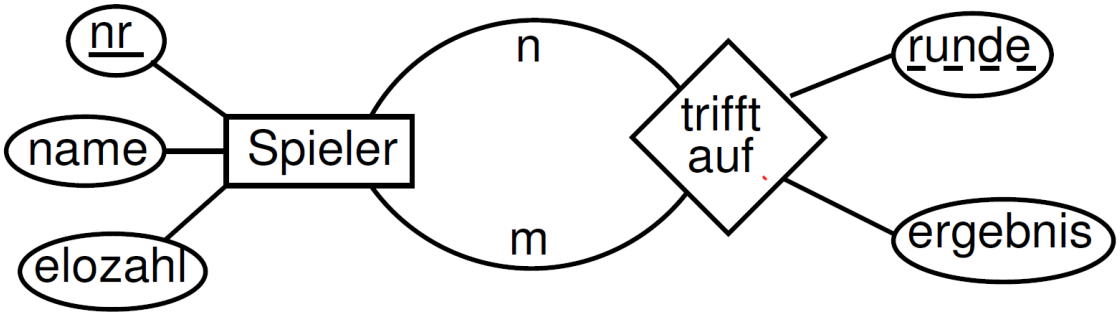


- ER-Modellierung weniger offensichtlich
- Ursache
  - Foreign Keys kennt das ER Modell nicht
  - Tauchen nur implizit auf bei Relationship oder weak Entity

  →zwei Modellierungsalternativen

Entity versus Relationship (2)

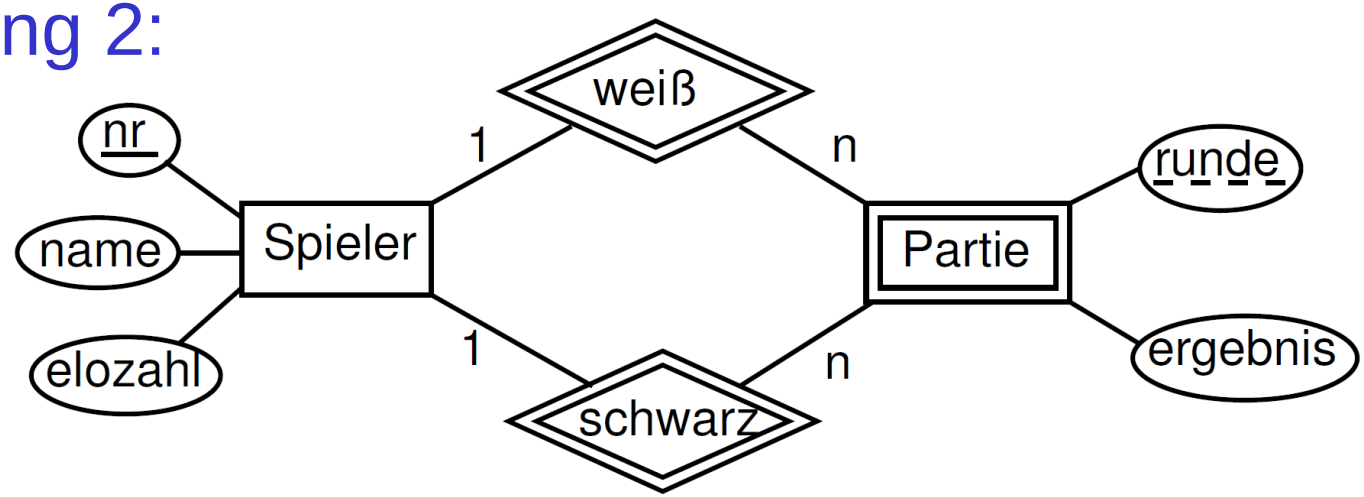
- Lösung 1:
  - Partien modelliert als reflexive Relationship zwischen zwei Spielern



- Probleme
  - Selbe Begegnung mehrmals möglich (gelöst über zusätzlichen Teilschlüssel *runde*)
  - Wer hat Weiß, wer Schwarz? (kann man mit Rollennamen lösen)
  - Eigentlich interessierendes Objekt "Partie" taucht gar nicht auf!

Entity versus Relationship (3)

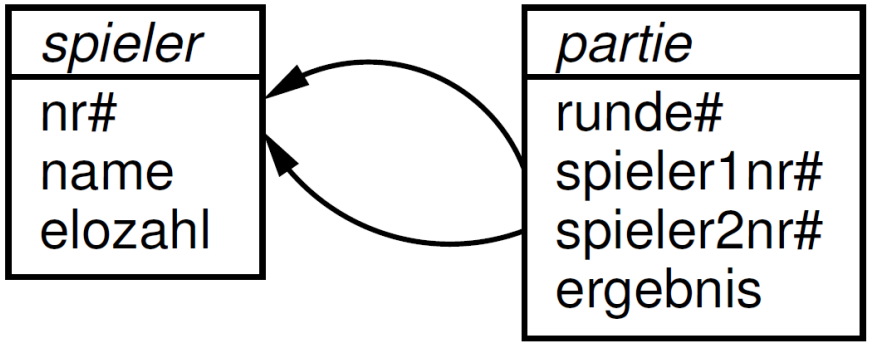
- Lösung 2:



- Partien modelliert als weak Entity, die von zwei strong Entitys abhängt → FK's gehen in PK ein
- Konstruktion etwas kurios, aber trotzdem angemessener
  - Information "weiß/schwarz" dargestellt"
  - "Partie" als Hauptgegenstand der Anforderung taucht explizit auf

Entity versus Relationship (4)

- Beide ER-Lösungen führen zum selben relationalen Modell



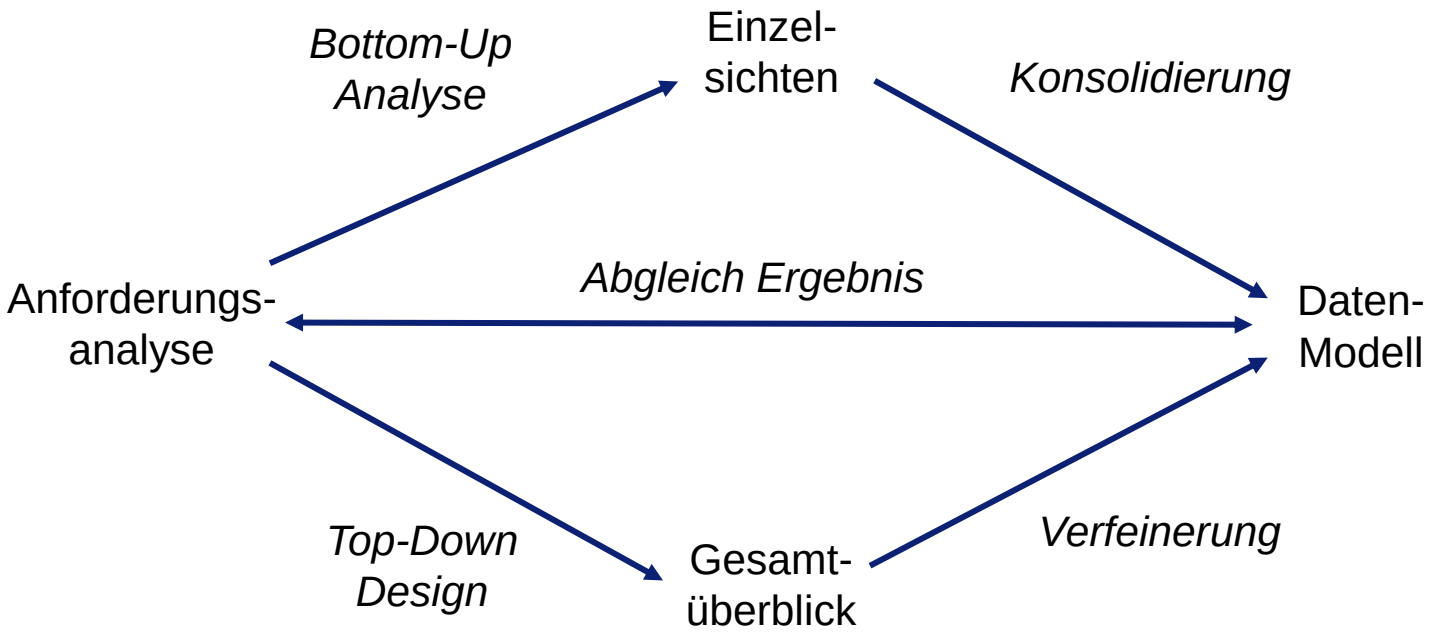
- Folgerungen
  - Übergang ER → Relational ist irreversibel: aus relationalem Schema lässt sich nicht mehr rekonstruieren, aus welchem ER-Schema es erzeugt wurde
  - Semantik von Lösung 1 (beide Spieler gleichwertig)
    - geht im Relationalen Modell verloren

Top-Down versus Bottom-Up (1)

- Top-Down
  - Starte mit grobem Entwurf, der zunehmend verfeinert wird (z.B. durch Einführung von Referenztabellen)
  - Erleichtert Verständnis des Systems, da zunächst aus der "Vogelperspektive" modelliert wird
- Bottom-Up
  - Arbeite Details aus und füge sie zu Gesamtsystem zusammen
  - Gesamtverständnis des Systems ist so schwerer zu gewinnen,
  - Gefahr des Verlierens im Detail wegen "Käferperspektive"

Top-Down versus Bottom-Up (1)

- In Praxis liefert Anforderungsanalyse meist vor allem Detailwissen
  - Zunächst Bottom-Up Analyse, bevor Einzelsichten konsolidiert werden



Mögliches Top-Down Vorgehen

- Modellieren der wesentlichen Entitys und Relationships
  - zwecks besseren Überblicks noch keine Attribute
- Ergänzen der Schlüsselattribute
- Modellieren aller Entity-Eigenschaften als (ggf. mehrwertige) Attribute
- Wo Auswahlliste für Attributwerte gewünscht, Attribute durch Relationships mit Referenztabellen ersetzen
- Wenn ein Attribut von mehreren Entitys verwendet wird, ebenfalls durch Referenzen auf neue Entity ersetzen (Warum?)