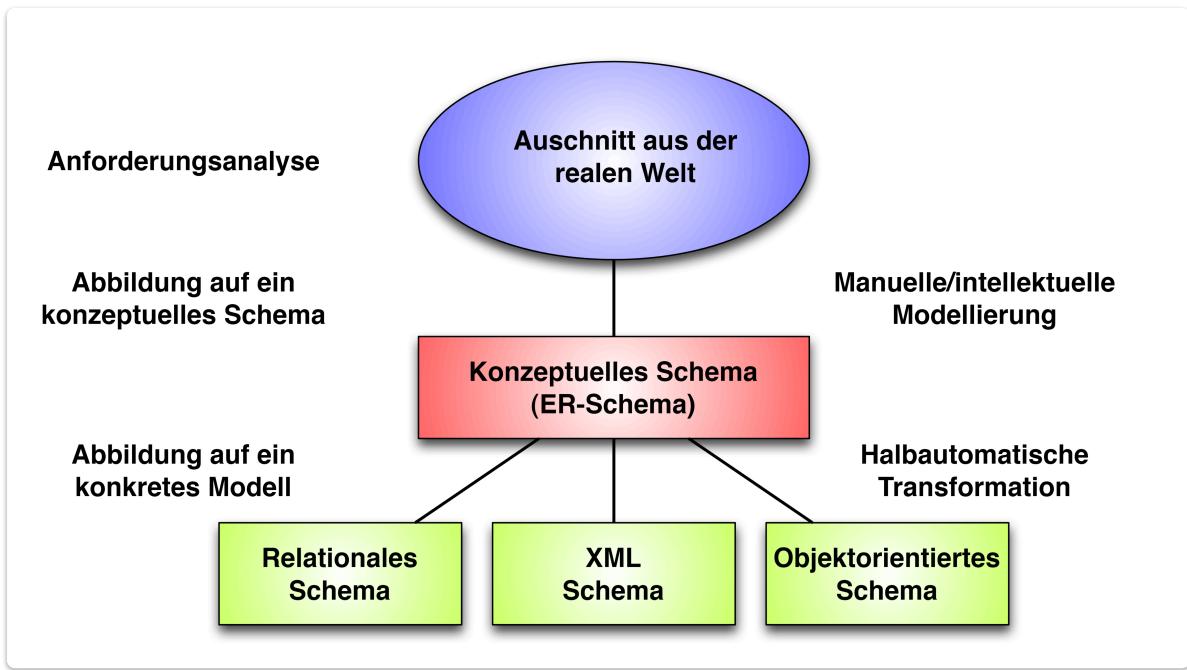


2. ER-Diagramme

Datenbankentwurf

Schritte des Datenbankentwurfs



1. Analyse des realen Welt-Ausschnitts:

- Startpunkt ist ein relevanter Ausschnitt aus der realen Welt, der in die Datenbank integriert werden soll.
- Dies beinhaltet eine **manuelle/intellektuelle Modellierung** durch den Datenbankdesigner.

2. Anforderungsanalyse:

- Erfassen aller relevanten Anforderungen an die Datenbank (Was soll die Datenbank können? Welche Daten müssen gespeichert werden?).

3. Abbildung auf ein Konzeptuelles Schema (ER-Schema):

- Das konzeptuelle Schema ist eine abstrakte Beschreibung der Daten und ihrer Beziehungen, unabhängig von einer spezifischen Datenbanktechnologie.
- Dies ist ein wichtiger Zwischenschritt, da es die Komplexität der realen Welt auf ein verständlicheres Modell reduziert.

4. Abbildung auf ein konkretes Modell:

- Das konzeptuelle Schema wird in ein spezifisches Datenmodell überführt. Dies kann sein:
 - **Relationales Schema** (für relationale Datenbanken)
 - **XML-Schema** (für XML-Datenbanken)
 - **Objektorientiertes Schema** (für objektorientierte Datenbanken)

- Dieser Schritt kann **halbautomatisch** erfolgen.
-

Schritt 1: Anforderungsanalyse

- **Definition:** Die Anforderungsanalyse ist der Prozess des Sammelns und Analysierens von Informationen über die Bedürfnisse und Erwartungen der Benutzer an das zukünftige Datenbanksystem.
- **Beispiele für Anforderungen:**
 - Studierende nehmen an Vorlesungen teil.
 - Professor:innen bieten Vorlesungen an.
 - Studierende werden durch die Matrikelnummer eindeutig identifiziert.

Objektbeschreibung

- **Ziel:** Identifizierung und Beschreibung der Objekte (Entitäten) und ihrer Attribute, die in der Datenbank gespeichert werden sollen.
- **Beispiel: Angestellte der Universität**
 - **PersonalNummer**
 - Typ: char (Zeichenkette)
 - Länge: 9
 - Wertebereich: 0.. .999.999.999
 - Verfügbarkeit: 100% (muss immer vorhanden sein)
 - Identifizierend: ja (kann einen Angestellten eindeutig identifizieren)
 - **Gehalt**
 - Typ: dezimal (Dezimalzahl)
 - Länge: (5, 2) (5 Ziffern insgesamt, davon 2 nach dem Komma, z.B. 123.45)
 - Verfügbarkeit: 10% (kann optional sein, nicht jeder Angestellte hat ein Gehalt im System hinterlegt)
 - Identifizierend: nein (ein Gehalt identifiziert keinen Angestellten eindeutig)
 - **Rang**
 - Typ: String (Text)
 - Länge: 50
 - Verfügbarkeit: 100%
 - Identifizierend: nein

Beziehungsbeschreibung

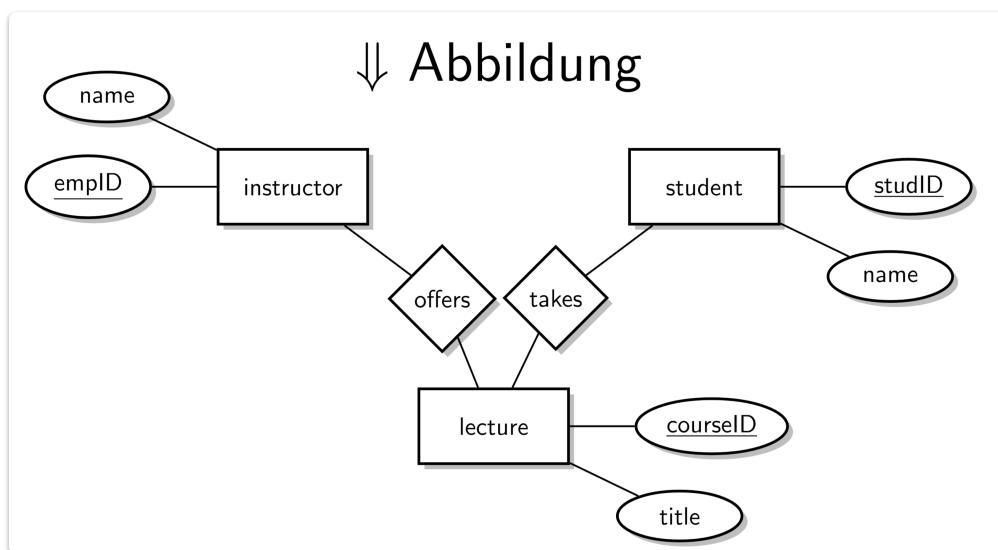
- **Ziel:** Identifizierung und Beschreibung der Beziehungen zwischen den identifizierten Objekten (Entitäten).
- **Beispiel: Beziehung „prüfen“**

- **Beteiligte Objekte:**

- Vortragende:r als Prüfer:in
- Studierende:r als Prüfling
- Vorlesung als Prüfungsstoff
- **Attribute der „prüfen“-Beziehung:**
 - Datum (Wann fand die Prüfung statt?)
 - Uhrzeit (Um welche Uhrzeit fand die Prüfung statt?)
 - Note (Welche Note wurde in der Prüfung erzielt?)

Schritt 2: Abbildung auf ein konzeptuelles Modell

- **Ziel:** Erstellung eines ER-Modells (Entity-Relationship-Modell) basierend auf den identifizierten Anforderungen.
- **Anforderungen (aus Schritt 1):**
 - Studierende nehmen an Vorlesungen teil.
 - Professor:innen bieten Vorlesungen an.
 - Studierende werden durch die Matrikelnummer eindeutig identifiziert.
- **Funktionale Anforderungen (Zusätzliche Anforderungen):**
 - Sekretär:in muss Noten eintragen können.



Schritt 3: Abbildung auf das relationale Modell

- **Ziel:** Überführung des ER-Modells in ein relationales Schema, das aus Tabellen (Relationen), Attributen und Schlüsseln besteht.
- **Resultierendes Relationales Modell (Beispiele):**
 - `student (studID: integer, name: string)`

- `studID` ist der Primärschlüssel (eindeutige Kennung für jeden Studenten).
 - `takes (studID: integer, courseID: integer)`
 - `studID` und `courseID` bilden zusammen den Primärschlüssel (ein Student kann ein spezifisches Fach nur einmal belegen).
 - `studID` ist ein Fremdschlüssel, der auf `student.studID` verweist.
 - `courseID` ist ein Fremdschlüssel, der auf `lecture.courseID` verweist.
 - `lecture (courseID: integer, title: string)`
 - `courseID` ist der Primärschlüssel (eindeutige Kennung für jede Vorlesung).
-

Schritt 4: Praktische Umsetzung und Implementierung

- **Ziel:** Die tatsächliche Erstellung der Datenbank und ihrer Tabellen in einem Datenbanksystem (DBMS) und das Einfügen von Daten.

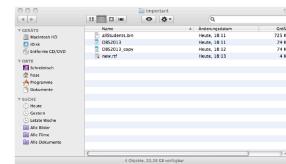
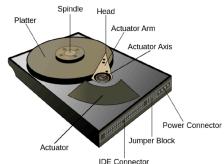
--hier Bild: Tabelle "student" mit Spalten "studID", "name" und Beispieldaten. Tabelle "takes" mit "studID", "courseID" und Beispieldaten. Tabelle "lecture" mit "courseID", "title" und Beispieldaten.--

- Tabellen einer DB (Beispieldaten):

student		takes		lecture	
studID	name	studID	courseID	courseID	title
26120	Pedersen	25403	5022	5001	DBS
25403	Hansen	26120	5001	5022	Belief and Knowledge
...

↓ Abbildung

Speicherseiten, Datenstrukturen, Indizes, Dateien, Geräte



- **Physische Speicherung:**

- Dies beinhaltet die Speicherung der Daten auf physischen Geräten.
- **Speicherseiten, Datenstrukturen, Indizes, Dateien, Geräte** spielen hier eine Rolle.

Zusammengefasst

1. Anforderungsanalyse

- *Mit was haben wir es zu tun?* (Was sind die Anforderungen und der Umfang des Datenbanksystems?)

2. Abbildung auf ein konzeptuelles Modell (konzeptueller Entwurf)

- *Welche Daten und Zusammenhänge müssen erfasst werden?* (Wie sollen die Daten und ihre Beziehungen abstrakt, d.h. unabhängig von einer spezifischen Datenbanktechnologie, dargestellt werden? Dies führt oft zu einem ER-Modell.)

3. Abbildung auf ein konkretes Modell (logischer Entwurf)

- *Wie müssen die Daten in einem bestimmten Modell strukturiert werden (hier: das relationale Modell)?* (Wie werden die Daten für ein spezifisches Datenbankmodell, z.B. das relationale Modell, organisiert und dargestellt? Dies beinhaltet die Definition von Tabellen, Attributen und Schlüsseln.)

4. Umsetzung und Implementierung (Physischer Entwurf)

- *Welche Anpassungen und Optimierungen sieht ein konkretes DBMS vor?* (Wie werden die logischen Schemata physisch auf Speichermedien abgebildet? Dies beinhaltet Aspekte wie Indizierung, Speicherorganisation und Dateiformate, um Performance und Speichereffizienz zu optimieren.)

Ein guter Entwurf vermeidet Redundanz und Unvollständigkeit. (Das bedeutet, dass Daten nicht unnötig mehrfach gespeichert werden sollten, um Inkonsistenzen zu vermeiden, und dass alle notwendigen Informationen vorhanden sind.)

Grundkonzepte des ER-Modells

Entität und Entitätstypen

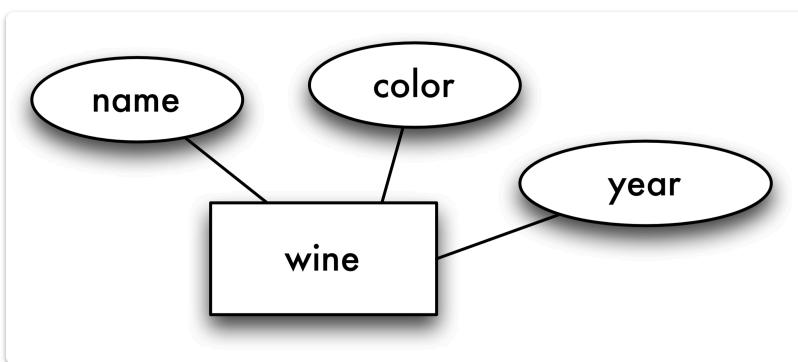
- **Entitäten** sind Objekte der realen Welt, über die wir Informationen abspeichern wollen.
 - Nur **Eigenschaften** der Entitäten können in einer Datenbank gespeichert werden (Beschreibung), nicht die Entitäten selbst.
- Entitäten werden in **Entitätstypen** eingeteilt.



- Eine **Entitymenge (entity set)** ist eine konkrete Menge von Entitäten des gleichen Entitätstyps.
 - Die zwei Begriffe Entitymenge (entity set) und Entitätstypen (entity type) werden oft als Synonyme verwendet.

Attribute

- **Attribute** modellieren Eigenschaften von Entitäten oder auch Beziehungen.
 - Alle Entitäten eines Entitätstyps haben dieselben Arten von Eigenschaften.
 - Attribute werden für Entitätstypen deklariert.
 - Attribute haben eine **Domäne** bzw. **Wertemenge** (eine definierte Menge von erlaubten Werten, z.B. für "Alter" nur positive ganze Zahlen).



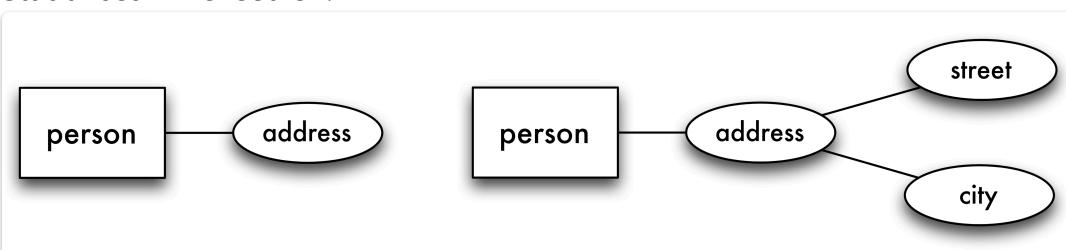
Single-valued (einwertige) vs. Multi-valued (mehrwertige) Attribute

- **Single-valued (einwertig):** Ein Attribut kann pro Entität nur einen einzigen Wert annehmen.
- **Multi-valued (mehrwertig):** Ein Attribut kann pro Entität mehrere Werte annehmen.
 - Beispiel: Eine Person kann mehrere Telefonnummern haben (oder eine einzige).



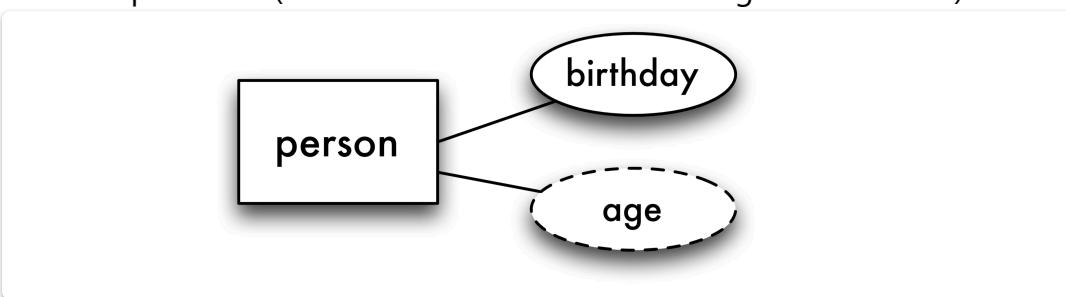
Simple (einfache) Attribute vs. Composite (zusammengesetzte) Attribute

- **Simple (einfache):** Ein Attribut, das nicht weiter unterteilt werden kann.
- **Composite (zusammengesetzte):** Ein Attribut, das aus mehreren einfacheren Attributen besteht.
 - Beispiel: Eine Adresse kann als String modelliert werden oder sich aus Straße und Stadt zusammensetzen.



Gespeicherte Attribute vs. derived (abgeleitete) Attribute

- **Gespeicherte Attribute:** Attribute, deren Werte direkt in der Datenbank abgelegt werden.
- **Derived (abgeleitete) Attribute:** Attribute, deren Werte aus anderen gespeicherten Attributen berechnet oder abgeleitet werden können. Sie werden nicht direkt gespeichert, sondern bei Bedarf ermittelt.
 - Zum Beispiel: Alter (kann aus dem Geburtsdatum abgeleitet werden).



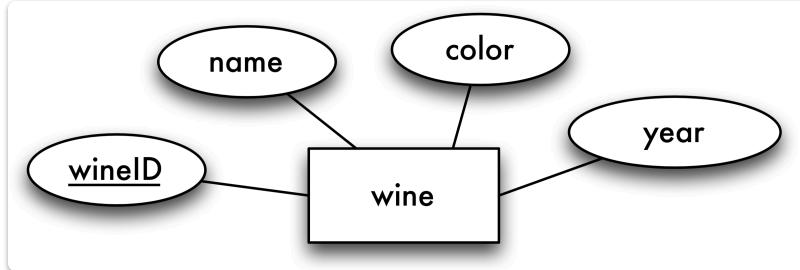
Schlüssel

- Ein **(Super-)Schlüssel** besteht aus einer Untermenge von Attributen eines Entitätstyps $E(A_1, \dots, A_m)$.
 - Die Menge der Schlüsselattribute ist $\{S_1, \dots, S_k\} \subseteq \{A_1, \dots, A_m\}$.
 - Die Attribute S_1, \dots, S_k eines Schlüssels werden **Schlüsselattribute** genannt.
- Die Werte der Schlüsselattribute identifizieren zusammen eindeutig ein bestimmtes Entität. (Das bedeutet, dass es keine zwei Entitäten geben kann, die die gleichen Werte für alle Schlüsselattribute haben.)
- Ein **Schlüsselkandidat** ist ein **minimaler Schlüssel**. (Minimal bedeutet, dass kein Attribut aus dem Schlüsselkandidaten entfernt werden kann, ohne dass die Eindeutigkeit verloren geht.)

- geht.)
- Gibt es mehrere Schlüsselkandidaten, so wird einer als **Primärschlüssel** ausgewählt.

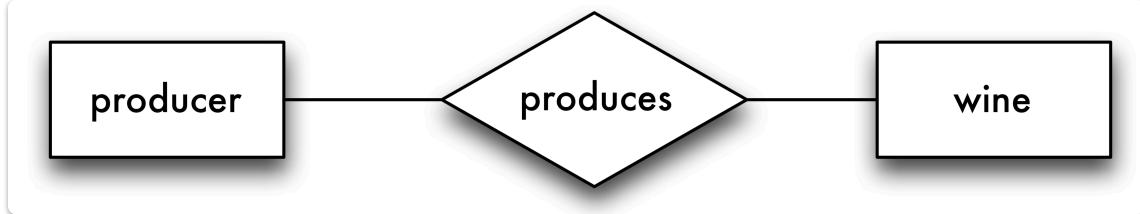
Primärschlüssel

- Attribute des Primärschlüssels werden durch Unterstrichen markiert.



Beziehung und Beziehungstyp

- Eine **Beziehung** beschreibt die Verbindung zwischen Entitäten.
- Beziehungen zwischen Entitäten werden zu **Beziehungstypen** zusammengefasst.



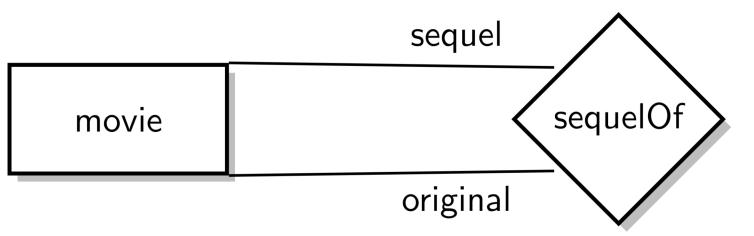
- Eine konkrete Verbindung zwischen zwei oder mehreren Entitäten wird **Beziehung (-instanz)** genannt.
- Eine **Beziehungsmenge** ist eine Menge von Beziehungsinstanzen.
- Die zwei Begriffe Beziehungsmenge (relationship set) und Beziehungstyp (relationship type) werden oft als Synonyme verwendet.

Mathematische Auffassung einer Beziehung

- Ein Beziehungstyp R zwischen den Entitätstypen E_1, E_2, \dots, E_n wird als **Relation** im mathematischen Sinne aufgefasst.
- Ausprägung** des Beziehungstyps R :
 - $R \subseteq E_1 \times E_2 \times \dots \times E_n$
- Ein Element $(e_1, e_2, \dots, e_n) \in R$ bezeichnet man als **Instanz** des Beziehungstyps.
 - Dabei ist $e_i \in E_i$ für alle $1 \leq i \leq n$.
- Anmerkung:** Diese Notation umfasst keine Attribute von Beziehungstypen.

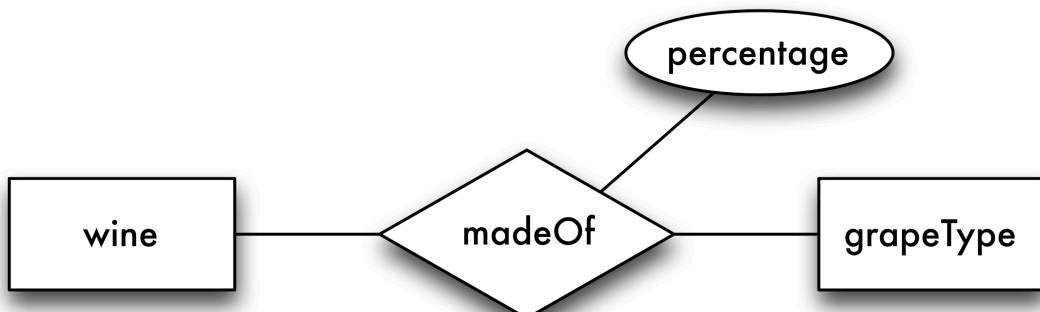
Rollennamen und Rekursive Beziehungstypen

- Rollennamen** sind optional und werden zur weiteren Charakterisierung einer Beziehung verwendet.
- Besonders sinnvoll bei einem **rekursiven Beziehungstyp**, bei dem ein Entitätstyp mehrfach an einem Beziehungstyp beteiligt ist.



Beziehungsattribute

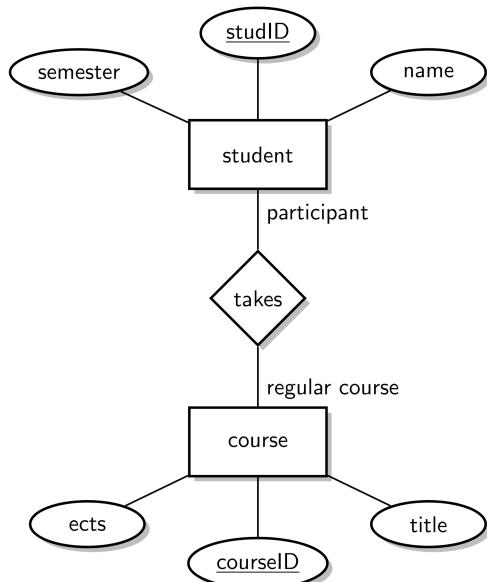
- Beziehungstypen können ebenfalls Attribute besitzen.



Beispiel mit Ablauf zum erstellen von einem ERD

Studierende nehmen an Vorlesungen teil

- Entity → Entitytyp



- Beziehung → Beziehungstyp
- Attribute (Eigenschaften)
- Primärschlüssel
- Rollen

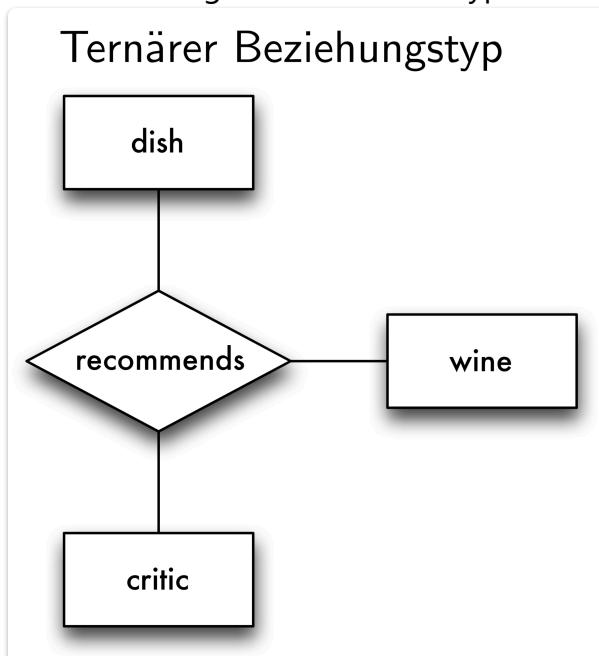
Eigenschaften von Beziehungstypen

Merkmale von Beziehungstypen

- **Stelligkeit bzw. Grad**
 - Beschreibt die Anzahl der beteiligten Entitätstypen an einer Beziehung.
 - Häufig: **binär** (zwei Entitätstypen beteiligt)
 - Weniger häufig: **ternär** (drei Entitätstypen beteiligt)
 - Allgemein: **n-stellig** (n Entitätstypen beteiligt)
- **Funktionalität / Kardinalität / Participation Constraints**
 - Beschreibt die Anzahl von Entitäten, die an einer Beziehung teilnehmen.
 - **Funktionalität (Chen-Notation):**
 - 1 : 1 (Eins-zu-Eins)
 - 1 : N (Eins-zu-Viele)
 - N : M (Viele-zu-Viele)
 - **Participation Constraints:**
 - **partiell:** Eine Entität muss nicht an der Beziehung teilnehmen.
 - **total:** Eine Entität muss an der Beziehung teilnehmen.
 - **Kardinalität ([min,max]-Notation):** Gibt die minimale und maximale Anzahl der Teilnahmen an einer Beziehung an.
 - [min, max]

Zwei- vs. mehrstellige Beziehungen

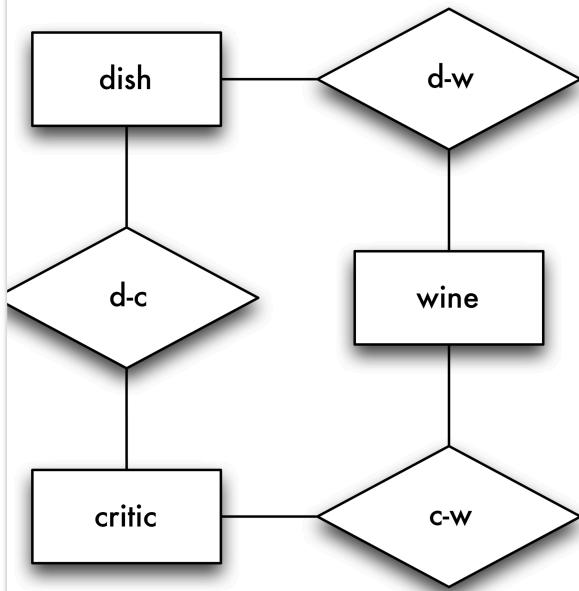
- **Ternärer Beziehungstyp:**
 - Eine Beziehung, die drei Entitätstypen miteinander verbindet.



- **Drei binäre Beziehungstypen:**

- Eine ternäre Beziehung kann auch durch drei binäre Beziehungen dargestellt werden, aber dies kann zu Informationsverlust führen oder komplexer sein.

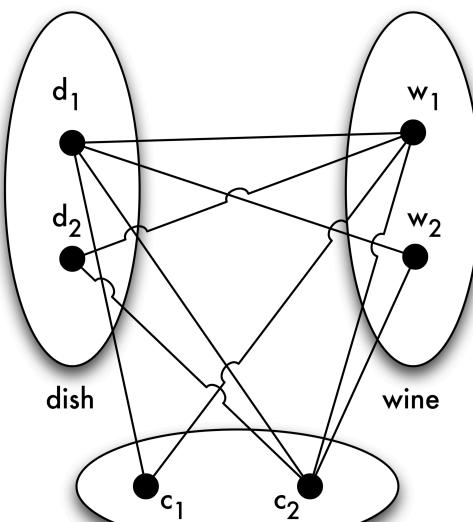
Drei binäre Beziehungstypen



Mehrstellige Beziehungstypen (Ausprägung)

- Zeigt die möglichen Instanzen bzw. Ausprägungen von mehrstelligen Beziehungen.

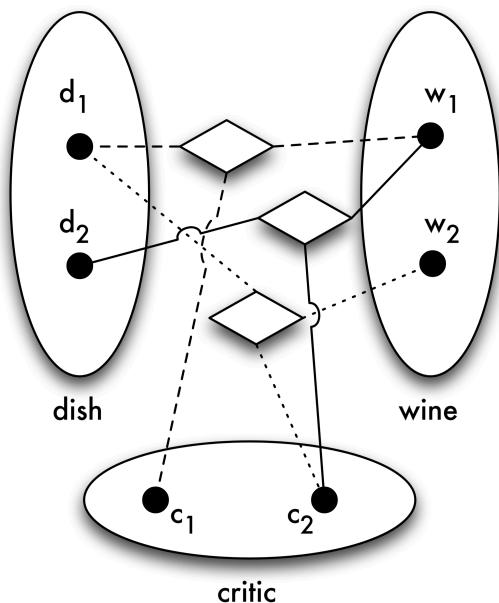
Binäre Beziehungstypen



Rekonstruierbare Beziehungen

- $d_1 - c_1 - w_1$
- $d_1 - c_2 - w_2$
- $d_2 - c_2 - w_1$
- aber auch: $d_1 - c_2 - w_1$

Ternärer Beziehungstyp



Mit binären Beziehungstypen können wir die folgende Beziehung rekonstruieren
 $d_1 - c_2 - w_1$
nicht jedoch mit einem ternären Beziehungstypen!

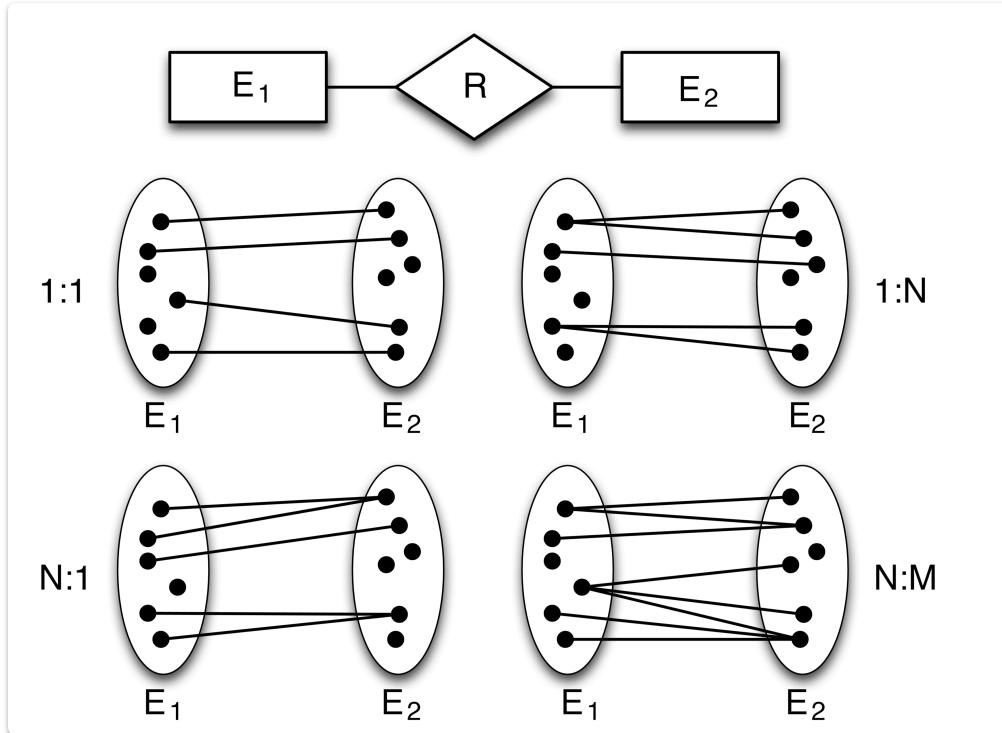
Merkmale von Beziehungstypen (Wiederholung und Ergänzung)

- **Stelligkeit bzw. Grad:**
 - Anzahl der beteiligten Entitätstypen.
 - Häufig: binär
 - Weniger häufig: ternär
 - Allgemein: n-stellig
- **Funktionalität / Kardinalität / Participation Constraints:**
 - Anzahl von Entitäten, die an einer Beziehung teilnehmen.
 - **Funktionalität (Chen-Notation):** Definiert, wie viele Instanzen eines Entitätstyps mit Instanzen eines anderen Entitätstyps in Beziehung stehen können.
 - 1 : 1 (Eins-zu-Eins)
 - 1 : N (Eins-zu-Viele)
 - N : M (Viele-zu-Viele)
 - **Participation Constraints:**
 - **partiell:** Eine Entität muss nicht an der Beziehung teilnehmen.
 - **total:** Eine Entität muss an der Beziehung teilnehmen.
 - **Kardinalität ([min,max]-Notation):** Gibt die minimale und maximale Anzahl der Teilnahmen an einer Beziehung an.
 - $[min, max]$

Chen-Notation (Funktionalität)

- Eine Beziehung R zwischen zwei Entitätstypen E_1 und E_2 wird in der Chen-Notation dargestellt.

- $R \subseteq E_1 \times E_2$

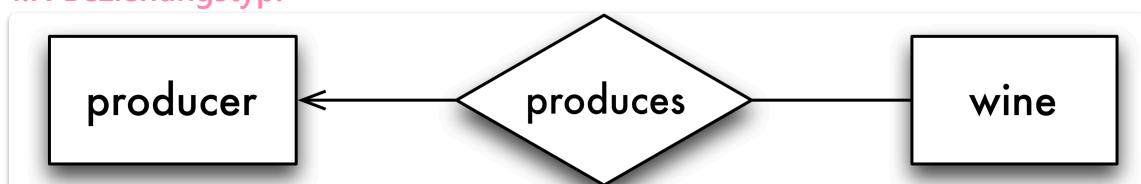


Funktionale Beziehungstypen

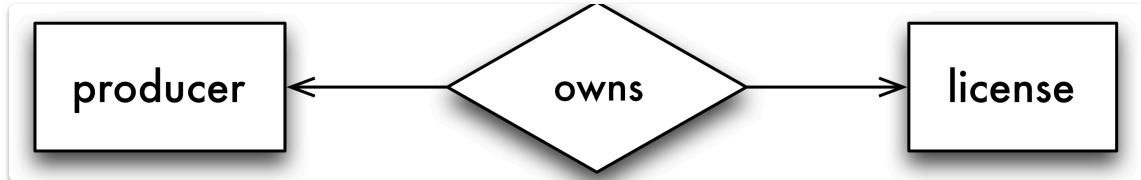
- $1 : 1$, $1 : N$ und $N : 1$ Beziehungen können als **partielle Funktionen** angesehen werden (oft auch als totalen Funktionen).
 - **1:1 Beziehungstypen:** $R : E_1 \rightarrow E_2$ und $R^{-1} : E_2 \rightarrow E_1$ (jeder e_1 ist zu höchstens einem e_2 zugeordnet und umgekehrt)
 - **1:N Beziehungstypen:** $R^{-1} : E_2 \rightarrow E_1$ (für jedes e_2 gibt es höchstens ein e_1 , mit dem es in Beziehung steht)
 - **N:1 Beziehungstypen:** $R : E_1 \rightarrow E_2$ (für jedes e_1 gibt es höchstens ein e_2 , mit dem es in Beziehung steht)
 - Dies wird auch als **funktionale Beziehung** genannt.
- Die "Richtung" ist entscheidend!
 - Die Funktion geht immer vom "N"-Entitätstyp zum "1"-Entitätstyp, d.h., die Seite mit der "vielen" Kardinalität bildet auf die Seite mit der "einen" Kardinalität ab.
- In dieser Vorlesung wird zwischen partiellen Funktionen (\rightarrow) und totalen Funktionen (\rightarrow) unterschieden. Es wird jedoch vereinfacht geschrieben.

Notation funktionaler Beziehungstypen

- **1:N Beziehungstyp:**



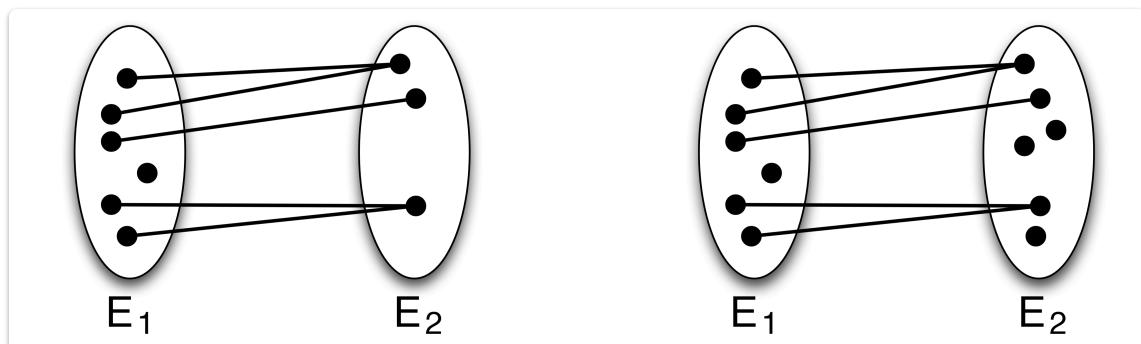
- Hier kann ein Produzent viele Weine produzieren, aber ein Wein wird nur von einem Produzenten produziert.
- **1:1 Beziehungstyp:**



- Hier besitzt ein Produzent genau eine Lizenz und eine Lizenz gehört genau einem Produzenten.

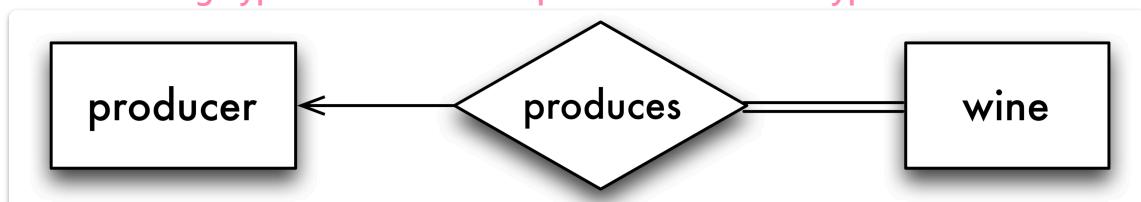
Participation Constraints

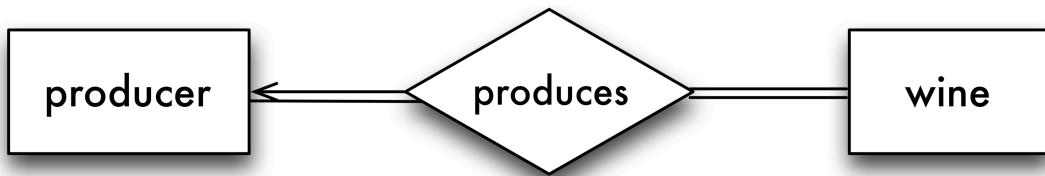
- **Total (totale Partizipation):**
 - Jedes Entity eines Entitätstyps **muss** an einer Beziehung teilnehmen. Es kann nicht existieren, ohne zu partizipieren.
 - Dies wird oft durch eine **Doppellinie** dargestellt.
- **Partiell (partielle Partizipation):**
 - Jedes Entity eines Entitätstyps **kann** an einer Beziehung teilnehmen. Es kann existieren, ohne zu partizipieren.
 - Dies wird oft durch eine **Einzellinie** dargestellt.



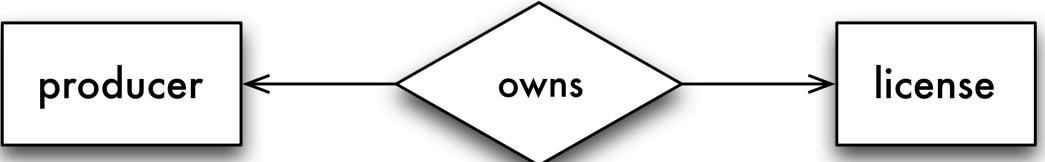
Graphische Darstellung von Participation Constraints

- **1:N Beziehungstyp mit totaler Partizipation vom Entitätstyp "wine":**
- Das bedeutet, jeder Wein muss von einem Produzenten produziert werden.
- **1:N Beziehungstyp mit totaler Partizipation von beiden Entitätstypen ("producer" und "wine"):**





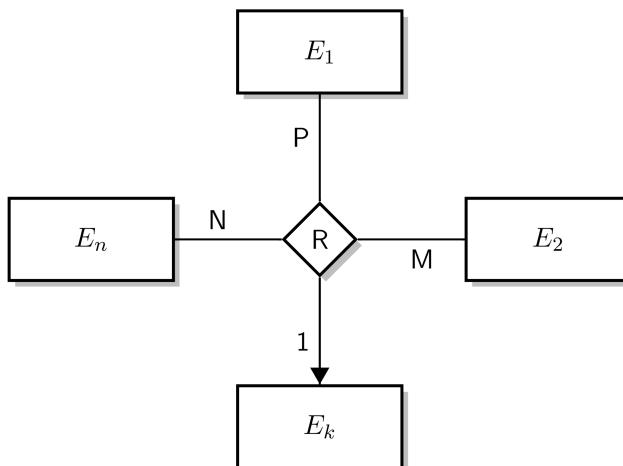
- Das bedeutet, jeder Produzent muss Weine produzieren und jeder Wein muss von einem Produzenten produziert werden.
- **1:1 Beziehungstyp mit partieller Partizipation:**



- Das bedeutet, ein Produzent kann eine Lizenz besitzen (muss aber nicht) und eine Lizenz kann von einem Produzenten besessen werden (muss aber nicht).

Funktionalitäten bei n-stelligen Beziehungstypen

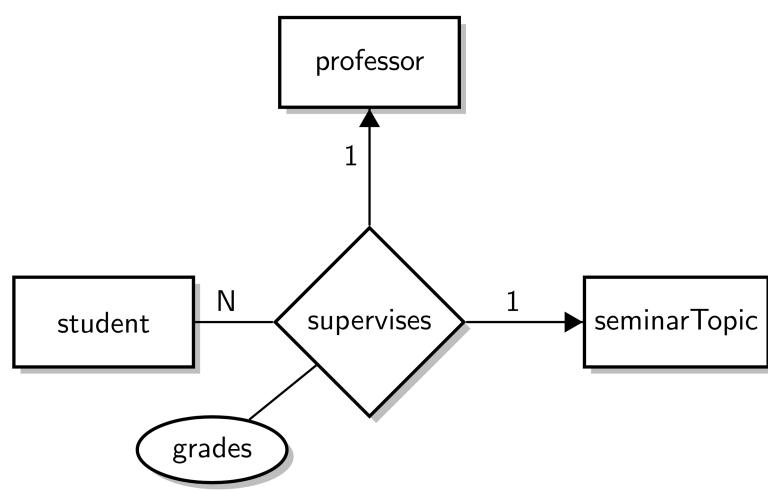
- Bei n-stelligen Beziehungstypen gibt es nicht nur die einfachen 1:1, 1:N oder N:M Funktionalitäten, sondern komplexere Zuordnungen.
- Eine n-stellige Beziehung R ist ein Teilprodukt der Entitätstypen:
 - $R \subseteq E_1 \times E_2 \times \dots \times E_k \times E_{k+1} \times \dots \times E_n$



$$R : E_1 \times E_2 \times \dots \times E_{k-1} \times E_{k+1} \times \dots \times E_n \rightarrow E_k$$

- **Anmerkung zur Notation im Allgemeinen:**
 - Die Verwendung von Pfeilen und 1, N, M etc. sind äquivalent.
 - Beide Darstellungen sind richtig, wobei eine eben manchmal hilfreicher sein kann, um die Funktionalität intuitiver zu erfassen.

Beispielbeziehungstyp: "supervises" (Betreut)



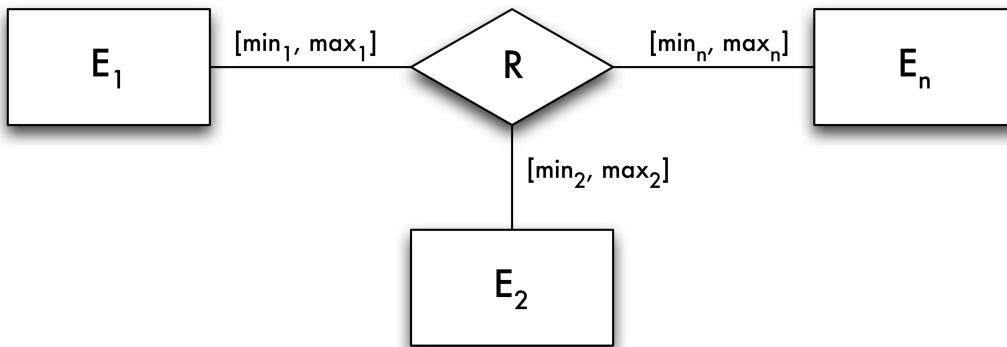
- Dieser ternäre Beziehungstyp "supervises" verbindet:
 - professor
 - student
 - seminarTopic
- Zusätzlich gibt es ein Beziehungsattribut `grades` (Noten).
- Die Funktionalitäten sind hier komplexer als bei binären Beziehungen.
- Der Beziehungstyp `supervises` kann als Abbildung dargestellt werden:
 - `supervises: professor × student → seminarTopic`
 - Dies bedeutet, dass eine bestimmte Kombination aus Professor und Student **maximal einem** Seminar-Thema zugeordnet ist.
 - `supervises: professor × seminarTopic → student`
 - Dies bedeutet, dass eine bestimmte Kombination aus Professor und Seminar-Thema **maximal einem** Studenten zugeordnet ist.

Merkmale von Beziehungstypen (Wiederholung)

- **Stelligkeit bzw. Grad:**
 - Anzahl der beteiligten Entitätstypen.
 - Häufig: binär
 - Weniger häufig: ternär
 - Allgemein: n-stellig
- **Funktionalität / Kardinalität / Participation Constraints:**
 - Anzahl von Entitäten, die an einer Beziehung teilnehmen.
 - Funktionalität (Chen-Notation): $1 : 1$, $1 : N$, $N : M$.
 - Participation Constraints: partiell oder total.
 - **Kardinalität ([min,max]-Notation):** $[min, max]$

$[min, max]$ -Notation (Kardinalität)

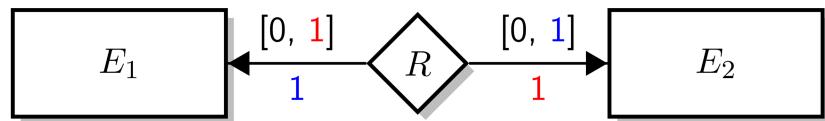
- Die $[min, max]$ -Notation schränkt ein, wie oft ein Entity eines Entitätstyps an einer Beziehung teilnehmen kann.



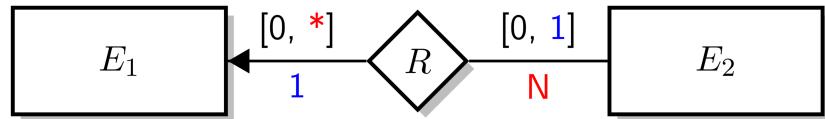
- Für jedes $e_i \in E_i$ gibt es:
 - mindestens** min_i Instanzen des Beziehungstyps der Art (\dots, e_i, \dots) und
 - höchstens** max_i viele Instanzen des Beziehungstyps der Art (\dots, e_i, \dots).
- Die Kardinalitätsbedingung ist also: $min_i \leq |\{r \in R \mid r \text{ enthält } e_i\}| \leq max_i$.
 - Das bedeutet, die Anzahl der Beziehungen, an denen ein bestimmtes Entity e_i beteiligt ist, muss zwischen min_i und max_i liegen.
- Spezielle Wertangabe für min_i : \emptyset
 - Bedeutet, dass ein Entity dieses Typs nicht an der Beziehung teilnehmen muss (partielle Partizipation).
- Spezielle Wertangabe für max_i : * (oder N)
 - Bedeutet, dass ein Entity dieses Typs an beliebig vielen Beziehungen teilnehmen kann.
- Wenn $min_i = 1$, bedeutet das **totale Partizipation**.
- Wenn $max_i = 1$, bedeutet das eine "1"-Seite in der Chen-Notation.

Chen vs min, max

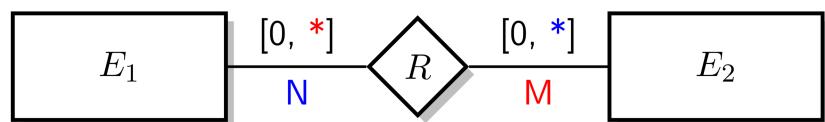
1:1 Beziehungstyp



1:N Beziehungstyp



N:M Beziehungstyp

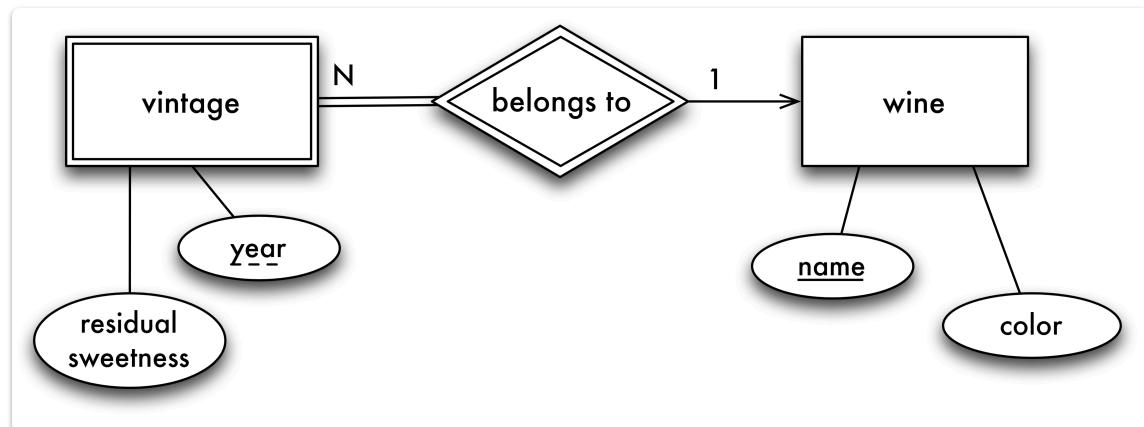


Abgrenzung von Funktionalität und Kardinalität

- In der Literatur werden die Begriffe **Funktionalität** und **Kardinalität** oft synonym verwendet.
- In dieser Vorlesung wird unterschieden:
 - **Funktionalität:**
 - Wird im Zusammenhang mit der **Chen-Notation** verwendet (z.B. $1 : 1$, $1 : N$, $N : M$).
 - **Kardinalität:**
 - Wird im Zusammenhang mit der **[min,max]-Notation** verwendet (z.B. $[0, 1]$, $[1, *]$).

Zusätzliche Konzepte

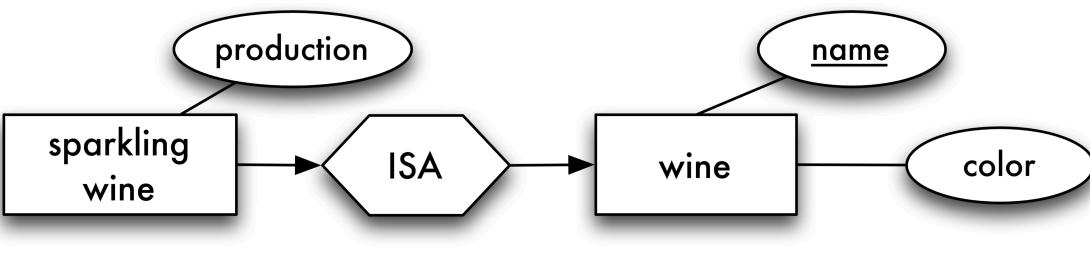
Schwache Entitätstypen



- Hier hat "vintage" die Attribute "year" und "residual sweetness".
- "wine" hat die Attribute "name" und "color".
- Die Funktionalität ist $N : 1$ von "vintage" zu "wine".
- Die Existenz eines **schwachen Entitys** (z.B. `vintage`) hängt von der Existenz eines **starken Entitys** (identifying/owning entity, z.B. `wine`) ab.
 - Sie sind durch eine **identifizierende Beziehung** verbunden.
- **Eigenschaften von schwachen Entitätstypen:**
 - **Totale Partizipation** des schwachen Entitätstyps: Ein schwaches Entity kann ohne die Beziehung zu seinem starken Entity nicht existieren. (Im Beispiel: Eine "vintage" (Jahrgang) existiert nur im Kontext eines "wine" (Weins)).
 - Nur in Kombination mit $1 : N$ ($N : 1$) oder selten auch $1 : 1$ Beziehungstypen.
 - Der starke Entitätstyp ist immer auf der "1"-Seite der Beziehung (also der Entitätstyp, von dem die schwache Entität abhängt).
- **Schlüsselattribute bei schwachen Entitätstypen:**
 - Schwache Entitäten sind oft nur mit dem **Schlüssel des entsprechenden starken Entitys eindeutig identifizierbar**.
 - Die Schlüsselattribute des schwachen Entitytyps werden gestrichen unterstrichen (sogenannte **Teilschlüssel**).
 - Im Beispiel wäre der "year" des "vintage" ein Teilschlüssel, da ein Jahrgang nur in Verbindung mit einem bestimmten Wein eindeutig ist (z.B. "Chardonnay 2020" vs. "Merlot 2020").

Der ISA-Beziehungstyp (Spezialisierung/Generalisierung)

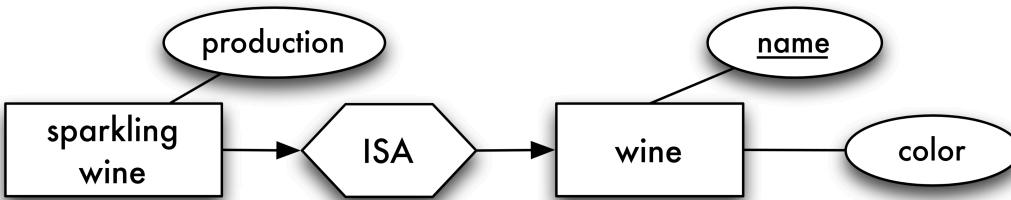
- **Spezialisierung und Generalisierung** (Vererbung) wird durch den **ISA-Beziehungstyp** ausgedrückt.



- Hier ist "sparkling wine" eine Spezialisierung von "wine" (d.h. Sekt ist eine Art Wein).
- Attribute von "wine" werden an "sparkling wine" vererbt.
- "sparkling wine" kann zusätzliche, spezifische Attribute haben ("production").

Eigenschaften von ISA-Beziehung

Grundlagen

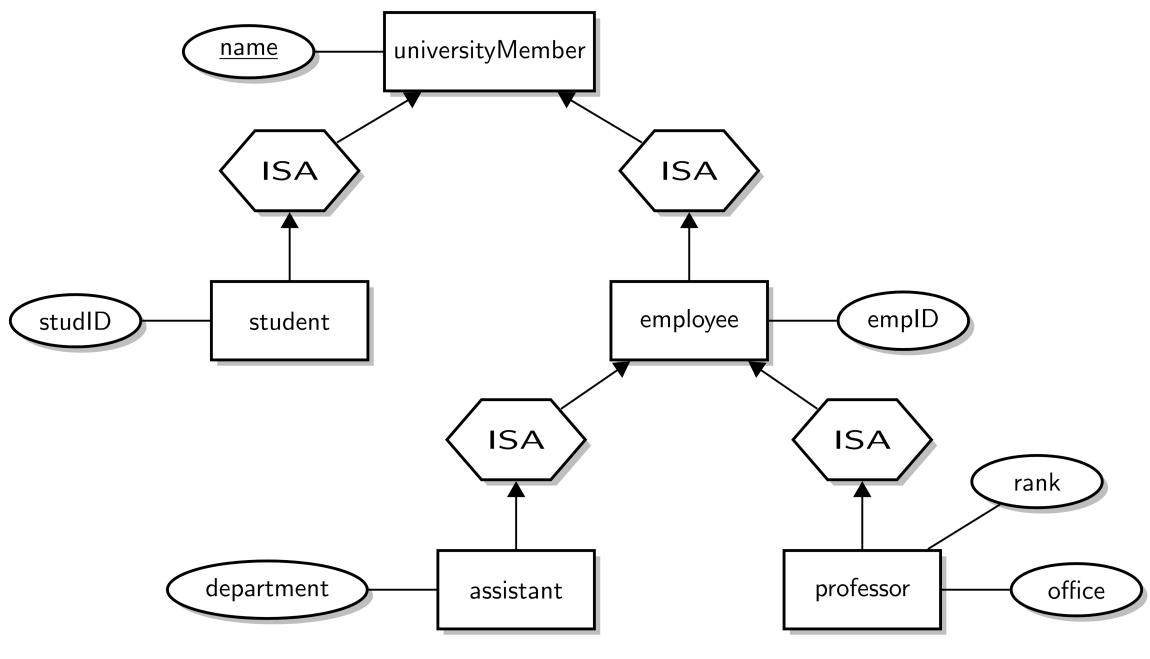


- Jeder Schaumwein ist genau einem Wein zugeordnet.
- Schaumweine werden durch die funktionale ISA-Beziehung identifiziert. (Eine Spezialisierung, bei der die Unterklasse eine Teilmenge der Oberklasse ist und alle Instanzen der Unterklasse auch Instanzen der Oberklasse sind.)
- Nicht jeder Wein ist zugleich ein Schaumwein. (Die Beziehung ist nicht umkehrbar; es gibt Weine, die keine Schaumweine sind.)
- Attribute des Entitytyps **wine** werden an den Entitytyp **sparkling wine** vererbt. (Das bedeutet, *sparkling wine* erbt alle Attribute von *wine*, wie z.B. *name* und *color*).

Kardinalität bei ISA-Beziehungen

- Die Kardinalitäten sind $ISA(E_1)[1, 1], E_2[0, 1]$
 - Jede Instanz von E_1 (*sparkling wine*) nimmt genau einmal an der Beziehung teil. (Jeder Schaumwein *ist* genau ein Wein.)
 - Instanzen von E_2 (*wine*) nehmen maximal einmal an der Beziehung teil. (Ein Wein kann ein Schaumwein sein, muss aber nicht.)

Beispiel: Universitätsbeispiel (überlappende Spezialisierung)



Dieses Beispiel zeigt eine hierarchische Struktur, bei der:

- Ein *university member* kann entweder ein *student* oder ein *employee* sein.
- Ein *employee* kann weiter als *assistant* oder *professor* spezialisiert werden.
- Es handelt sich um eine **überlappende Spezialisierung**, d.h., eine Instanz der Oberklasse kann zu mehreren Unterklassen gehören. (Beispiel: Ein Mitarbeiter könnte gleichzeitig ein Student sein, z.B. ein wissenschaftlicher Mitarbeiter, der noch eingeschrieben ist.)

Spezielle Eigenschaften des ISA-Beziehungstyps (Zusätzliche Konzepte im ER-Modell)

Überlappende Spezialisierung

- **Definition:** Ein Entity (eine Instanz der Oberklasse) kann zu mehreren spezialisierten Entitymengen (Unterklassen) gehören.
 - **Beispiel:** Eine Person kann gleichzeitig *Student* und *Mitarbeiter* sein.
- **Darstellung im ER-Modell:** Separate ISA-Symbole werden verwendet, um die möglichen Mehrfachzugehörigkeiten zu kennzeichnen.

Disjunkte Spezialisierung

- **Definition:** Ein Entity (eine Instanz der Oberklasse) kann zu höchstens einer spezialisierten Entitymenge (Unterklasse) gehören.
 - **Beispiel:** Ein Fahrzeug kann entweder ein *Auto* oder ein *Motorrad* sein, aber nicht beides gleichzeitig.
- **Darstellung im ER-Modell:** Ein gemeinsames ISA-Symbol wird verwendet, um anzudeuten, dass sich die Unterklassen gegenseitig ausschließen.

Attribute und Beziehungstypen bei Spezialisierung/Generalisierung

Vererbung bei spezialisierten Entitytypen

Spezialisierte Entitytypen (Unterklassen) **erben**:

- **Attribute** von weniger spezialisierten Entitytypen (Oberklassen).
 - Beispiel: Ein *Student* erbt Attribute wie *Name* und *Adresse* von der Oberklasse *Person*.
- **Teilnahme an Beziehungstypen** von weniger spezialisierten Entitytypen.
 - Beispiel: Wenn *Person* an der Beziehung *wohnt in* (*Ort*) teilnimmt, dann nimmt auch *Student* an dieser Beziehung teil.

Zusätzliche Eigenschaften spezialisierter Entitytypen

Spezialisierte Entitytypen können:

- **Zusätzliche Attribute** haben, die nur für sie relevant sind.
 - Beispiel: *Student* hat das zusätzliche Attribut *Matrikelnummer*, das *Person* nicht hat.
- **An Beziehungstypen teilnehmen**, an denen die weniger spezialisierten Entitytypen nicht teilnehmen.
 - Beispiel: *Student* nimmt an der Beziehung *studiert* (*Studiengang*) teil, während *Person* nicht unbedingt an dieser Beziehung teilnehmen muss.

Participations-Constraints (Teilnahmebedingungen)

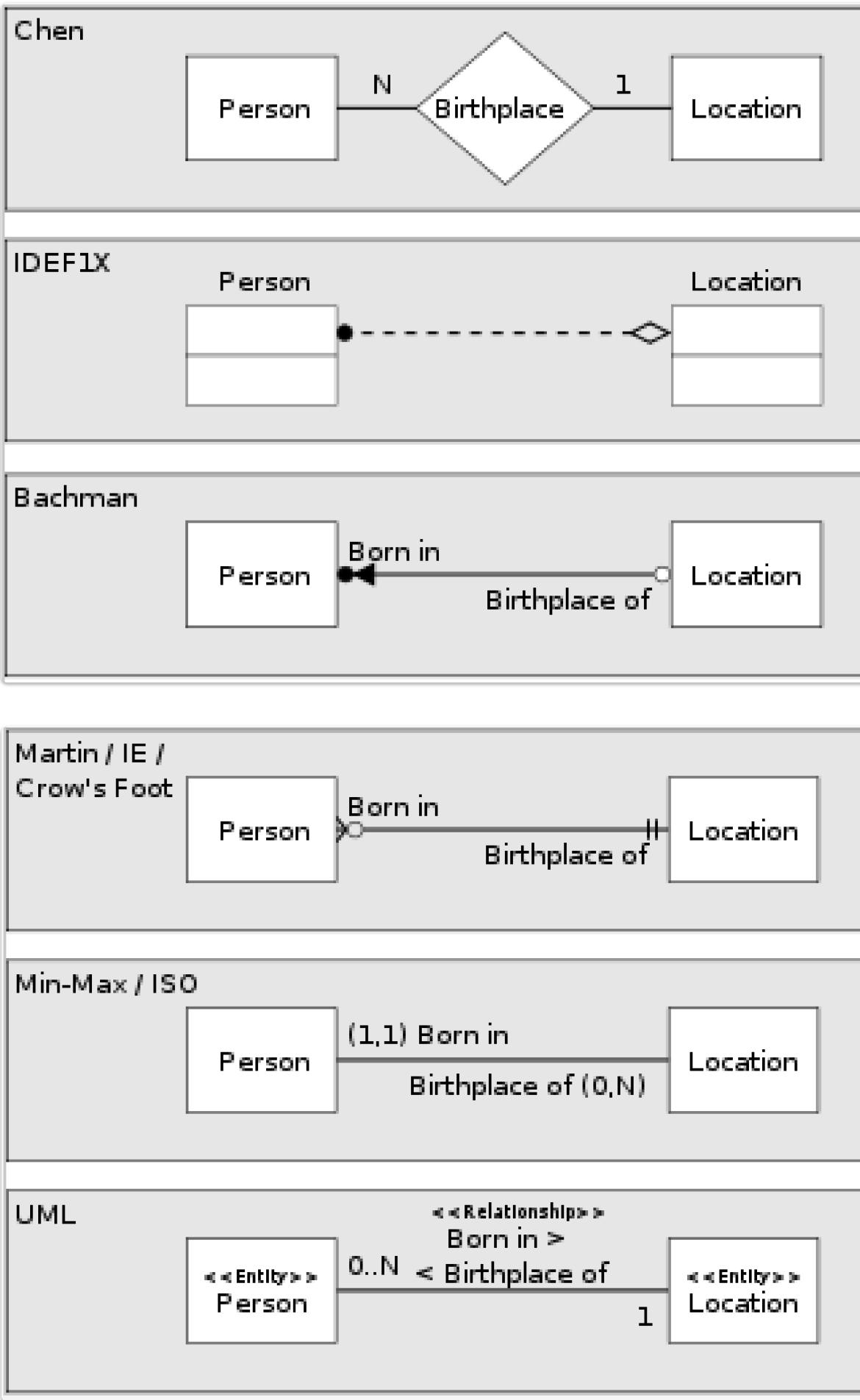
Totale Generalisierung/Spezialisierung

- **Definition:** Jeder weniger spezialisierte Entitytyp (Instanz der Oberklasse) **muss** zu einem spezialisierteren Entitytyp (mindestens einer Unterklasse) gehören.
 - Das bedeutet, jede Instanz der Oberklasse muss einer der definierten Spezialisierungen zugeordnet werden.
- **Notation im ER-Modell:** Eine **doppelte Linie** unter dem ISA-Symbol.

Partielle Generalisierung/Spezialisierung (Default)

- **Definition:** Jeder weniger spezialisierte Entitytyp (Instanz der Oberklasse) **kann** (aber muss nicht) zu einem spezialisierten Entitytyp (einer der Unterklassen) gehören.
 - Das bedeutet, es kann Instanzen der Oberklasse geben, die keiner der definierten Spezialisierungen zugeordnet werden.
- **Notation im ER-Modell:** Eine **einfache Linie** unter dem ISA-Symbol (dies ist der Standardfall, wenn nicht anders angegeben).

Alternative Notationen



Relationen aus Grundkonzepten ableiten

Entwurfsanmerkungen für ER-Modelle

- **Entitys** entsprechen Substantiven.
- **Beziehungen** entsprechen Verben.
- Jede Aussage in den Anforderungen sollte sich im ER-Schema widerspiegeln.
- Jedes ER-Diagramm (ERD) sollte sich in den Anforderungen wiederfinden.
- Ein konzeptueller Entwurf kann Inkonsistenzen und Mehrdeutigkeiten in den Anforderungen aufdecken, die zuerst geklärt werden müssen.

Relationale Modellierung aus ER-Modell

Entitätstypen

- **student**: `{} studID: integer, name: string, semester: integer {}`
- **course**: `{} courseID: integer, title: string, ects: integer {}`
- **professor**: `{} empID: integer, name: string, rank: string, office: integer {}`
- **assistant**: `{} empID: integer, name: string, department: string {}`

Grundsätzliches Vorgehen für Entitätstypen

Für jeden Entitätstyp wird eine Relation erstellt:

- Name des Entitätstyps → Name der Relation
- Attribute des Entitätstyps → Attribute der Relation
- Schlüssel des Entitätstyps → Schlüssel der Relation

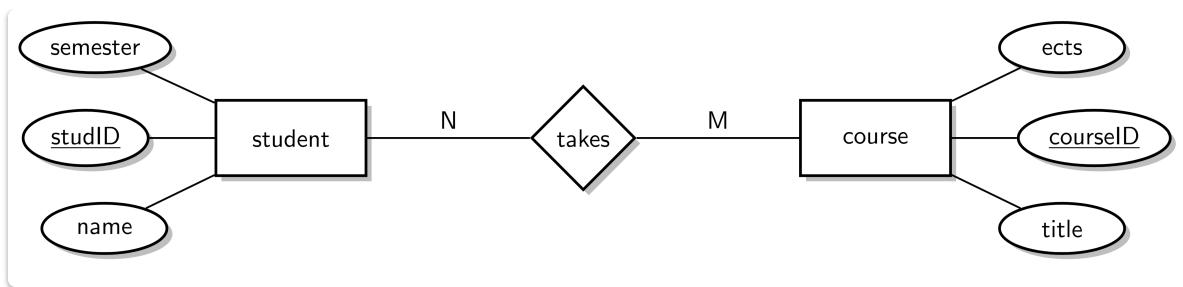
Notation von Relationenschemata

Beispiel:

- `student (studID, name, semester)`
- `student: {} studID, name, semester {}`

Die Reihenfolge der Attribute ist in diesem Kontext egal. Die Domäne der Attribute ist im Moment auch nicht wichtig.

Abbildung von N:M-Beziehungstypen



Grundsätzliches Vorgehen für N:M-Beziehungstypen

- Ein neues Relationenschema wird erstellt, das alle Attribute des Beziehungstyps enthält.
- Alle Primärschlüsselelemente der beteiligten Entitätstypen werden übernommen.
- Die "importierten" Schlüsselattribute der beteiligten Entitätstypen werden **Fremdschlüssel** genannt.

student		takes		course	
<u>studID</u>	...	<u>studID</u>	<u>coursID</u>	<u>coursID</u>	...
26120	...	26120	5001	5001	...
27550	...	27550	5001	4052	...
...	...	27550	4052	5041	...
		28106	5041	5052	...
		28106	5052	5216	...
		28106	5216	5259	...
		28106	5259	29120	...
		29120	5001	5001	...
		29120	5001	5041	...
		29120	5041	5049	...

Ein ovaler Kasten 'semester' ist über einen Pfeil mit dem Attribut studID des student-Typs verbunden. Ein ovaler Kasten 'ects' ist über einen Pfeil mit dem Attribut coursID des course-Typs verbunden.

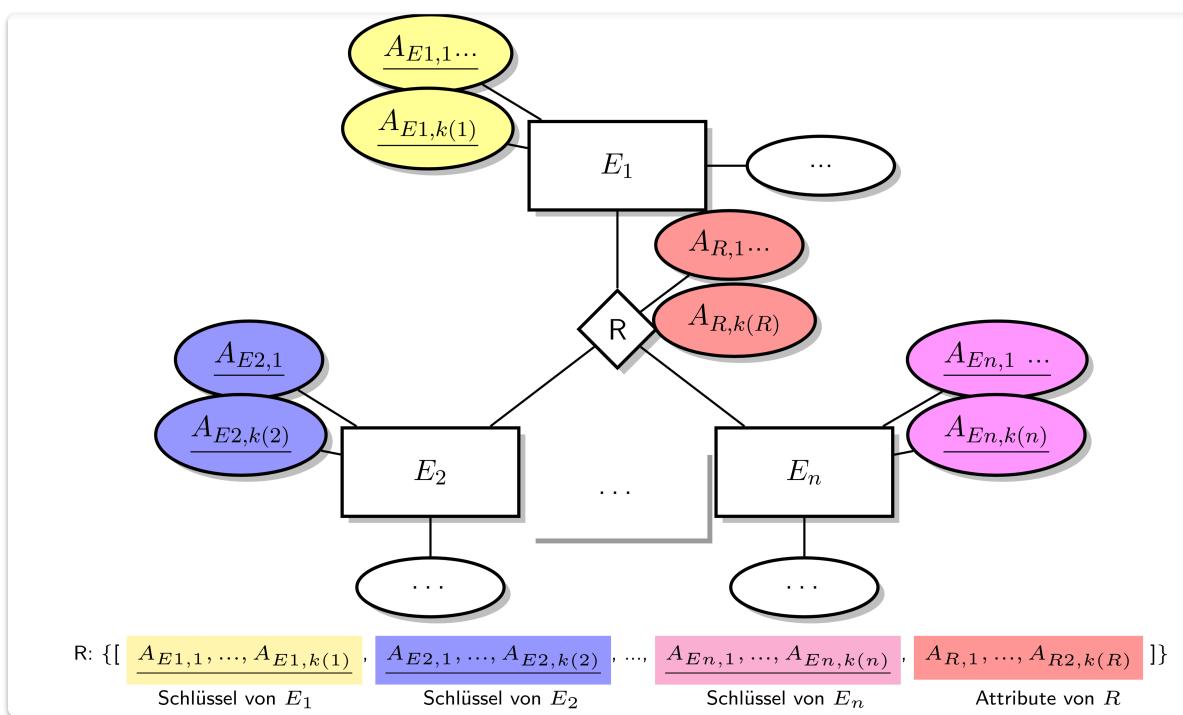
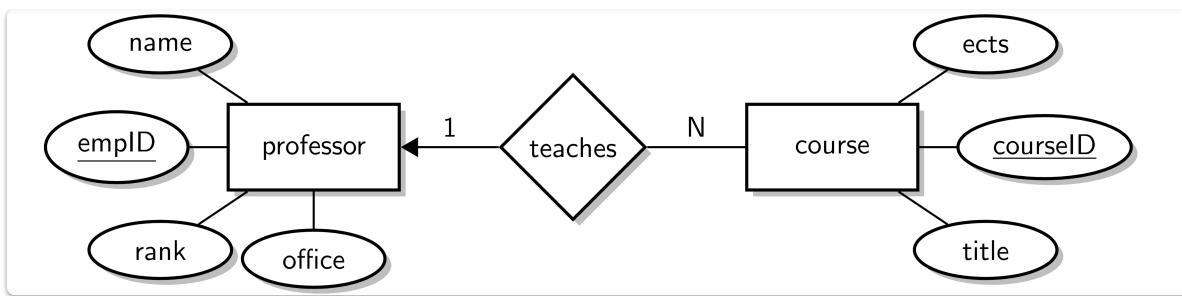


Abbildung von 1:N-Beziehungstypen



Grundsätzliches Vorgehen für 1:N-Beziehungstypen

- Ein neues Relationenschema wird mit allen Attributen des Beziehungstyps erstellt.
- Alle Primärschlüsselelemente der beteiligten Entitätstypen werden übernommen.
- Der Primärschlüssel der N-Seite (die Seite mit der Mehrfachbeziehung) wird zum Schlüssel im neuen Relationenschema.

Initialentwurf

- `course : {{ courseID, title, ects }}`
- `professor : {{ empID, name, rank, office }}`
- `teaches : {{ courseID \rightarrow course, empID \rightarrow professor }}`

Verbesserung durch Zusammenlegung (Finale Abbildung)

Bei 1:N-Beziehungen kann der Primärschlüssel der 1-Seite (in diesem Fall `professor`) direkt in die Relation der N-Seite (hier `course`) als Fremdschlüssel integriert werden. Dies vermeidet die Notwendigkeit einer separaten Relation für die Beziehung selbst.

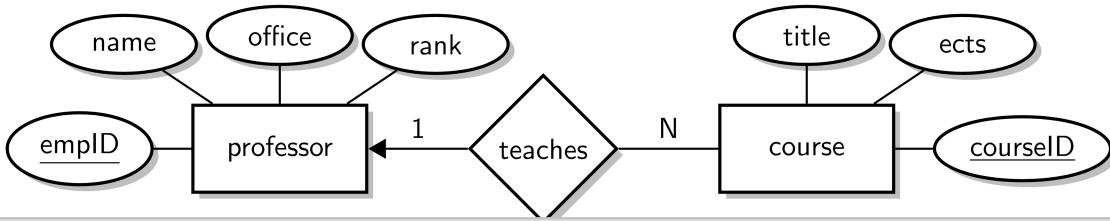
- `course : {{ courseID, title, ects, taughtBy \rightarrow professor }}`
- `professor : {{ empID, name, rank, office }}`

`taughtBy` ist ein **Fremdschlüssel** und referenziert den Primärschlüssel der Relation `professor`. Die Werte von `taughtBy` entsprechen den Werten von `empID` in der Relation `professor`.

Relationen mit denselben Schlüsseln können und sollten immer kombiniert werden, aber nur diese und keine anderen!

professor			
emplID	name	rank	office
2125	Socrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

course			
courselID	title	ects	taughtBy
5001	DBS	4	2137
5041	Robotics	4	2125
5043	Software Engineering	3	2126
5049	Ethics	2	2125
4052	Logic	4	2125
5052	Theory of Science	3	2126
5216	Bioethics	2	2126
5259	Chemistry	2	2133
5022	Belief and Knowledge	2	2134
4630	Physics	4	2137

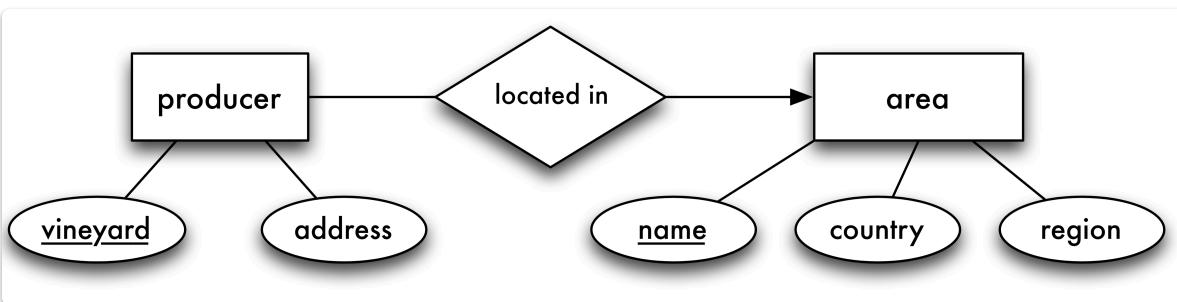


Achtung: Das funktioniert nicht

professor				
emplID	name	rank	office	teaches
2125	Socrates	C4	226	5041
2125	Socrates	C4	226	5049
2125	Socrates	C4	226	4052
...
2134	Augustinus	C3	309	5022
2136	Curie	C4	36	??
...

course		
courselID	title	ects
5001	DBS	4
5041	Robotics	4
5043	Software Engineering	3
5049	Ethics	2
4052	Logic	4
5052	Theory of Science	3
5216	Bioethics	2
5259	Chemistry	2
5022	Belief and Knowledge	2
4630	Physics	4

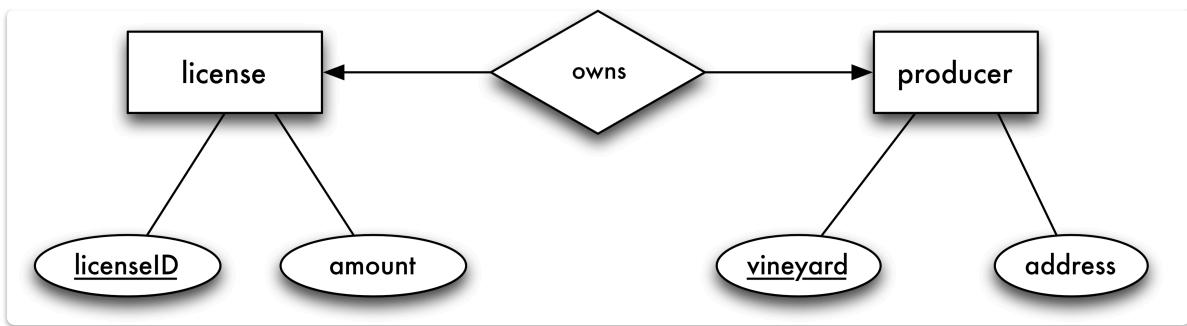
Zusammenfassung: N:1-Beziehungstypen



Wie bildet man dieses ER-Diagramm auf Relationen ab?

- producer : {{ vineyard, address, locatedIn \$\rightarrow\$ area }} (Der Fremdschlüssel zur 1-Seite (area) wird in die N-Seite (producer) integriert.)
- area : {{ name, country, region }}

1:1-Beziehungstypen



Unterscheidet sich die Herangehensweise hier zu einem 1:N-Beziehungstypen?

Ja, bei 1:1-Beziehungen gibt es mehr Flexibilität bei der Wahl, wo der Fremdschlüssel platziert wird.

Grundsätzliches Vorgehen für 1:1-Beziehungstypen

- Ein neues Relationenschema mit allen Attributen des Beziehungstyps wird erstellt (dies ist eine Option, aber nicht zwingend notwendig, wie weiter unten gezeigt wird).
- Alle Primärschlüsselelemente der beteiligten Entitätstypen werden übernommen.
- Irgendein Primärschlüssel des involvierten Entitätstyps wird der Schlüssel im neuen Relationenschema (wenn eine separate Relation für die Beziehung erstellt wird).

Initialentwurf

- **license:** {[licenselID, amount]}
- **producer:** {[vineyard, address]}
- **owns:** {[licenselID → license, vineyard → producer]} oder
owns: {[licenselID → license, vineyard → producer]}

Verbesserung durch Zusammenfassung

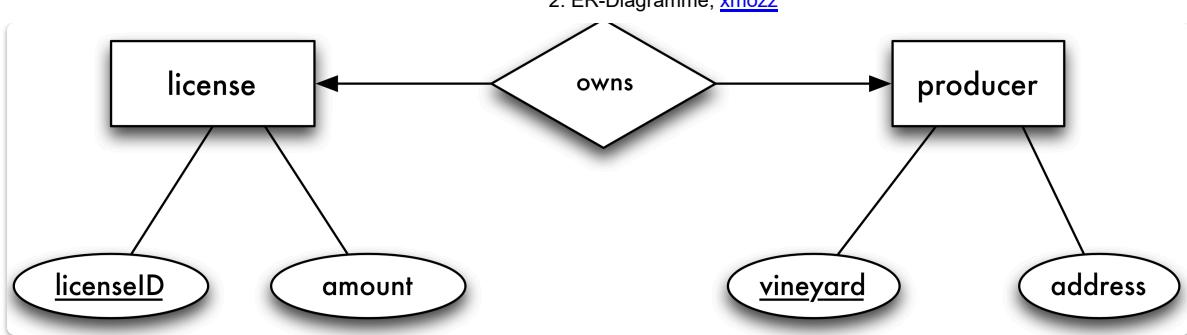
- **license:** {[licenselID, amount, ownedBy → producer]}
- **producer:** {[vineyard, address]}

oder

- **license:** {[licenselID, amount]}
- **producer:** {[vineyard, address, ownsLicense → license]}

Am besten erweitert man eine Relation mit **totaler Partizipation**.

Warum nicht eine einzige Relation für 1:1-Beziehungen?



Problem bei der Zusammenführung in eine einzige Relation

Eine Zusammenführung von `license` und `producer` in eine einzige Relation (z.B. `producer` mit `licenseID` und `amount` als Attribut) wäre nur korrekt bei **totaler Partizipation** ((1, 1)) beider beteiligter Entitätstypen.

- **Totale Partizipation** bedeutet: Jede Instanz des einen Entitätstyps muss mit genau einer Instanz des anderen Entitätstyps in Beziehung stehen.
 - Beispiel: Jeder `producer` besitzt genau eine `license` UND jede `license` gehört genau einem `producer`.

producer	vineyard	address	licenseID	amount
	Rotkäppchen Weingut Müller \perp	Freiberg Dagstuhl \perp	42-007 \perp 42-003	10.000 \perp 100.000

Was passiert bei partieller Partizipation?

- **Partielle Partizipation** beider Entitätstypen (z.B. ein `producer` hat keine `license` oder eine `license` gehört keinem `producer`):
 - Führt zu **Null-Werten** in allen Attributen, die den nicht beteiligten Entitätstyp repräsentieren würden (z.B. `licenseID` und `amount` wären `NULL` für einen `producer` ohne Lizenz).
 - Die Bestimmung eines Primärschlüssels ist nicht mehr eindeutig oder schwierig, da Nullwerte auftreten können und die Identität nicht immer gegeben ist.
 - Führt zu **Speicherplatzverschwendungen** (unnötige `NULL`-Werte belegen Speicherplatz).

Zusammenfassend: Eine einzige Relation für 1:1-Beziehungen ist nur effizient und korrekt, wenn eine totale Partizipation von beiden Seiten der Beziehung vorliegt. Andernfalls sollte man die Entitäten getrennt halten und den Fremdschlüssel in eine der beiden Relationen platzieren, um Nullwerte zu minimieren.

Zusammenfassung: Beziehungstypen auf Relationen abbilden

M:N-Beziehungstyp

- **Neue Relation** mit Attributen des Beziehungstyps erstellen.

- Attribute hinzufügen, die die Primärschlüssel der involvierten Entitätstypen referenzieren (*Fremdschlüssel*).
- **Primärschlüssel:** Menge der Fremdschlüsselelemente (Der Primärschlüssel dieser neuen Relation setzt sich aus den Primärschlüsseln der Entitäten zusammen, die an der M:N-Beziehung beteiligt sind. Dies stellt die Eindeutigkeit jeder Beziehung sicher).

1:N-Beziehungstyp

- Attribut zur Relation des Entitätstyps auf der „N“-Seite hinzufügen:
 - *Fremdschlüssel*, der den Primärschlüssel des Entitätstyps auf der „1“-Seite referenziert (Der Fremdschlüssel auf der "N"-Seite verweist auf den Primärschlüssel der "1"-Seite, um die Beziehung zwischen den Entitäten herzustellen).
 - Attribute des Beziehungstyps hinzufügen (falls vorhanden).

1:1-Beziehungstyp

- Attribut zur Relation eines der involvierten Entitätstypen hinzufügen:
 - *Fremdschlüssel*, der den Primärschlüssel des Entitätstyps der anderen Seite referenziert (Der Fremdschlüssel kann zu einer der beiden beteiligten Entitäten hinzugefügt werden, da die Beziehung in beide Richtungen eindeutig ist. Es ist oft sinnvoll, den Fremdschlüssel zu der Entität hinzuzufügen, die seltener an der Beziehung beteiligt ist oder aus logischen Gründen besser passt).
 - Attribute des Beziehungstyps hinzufügen (falls vorhanden).

Fremdschlüssel

Ein **Fremdschlüssel** ist ein Attribut (oder eine Kombination von Attributen) einer Relation, das auf den **Primärschlüssel** (oder Schlüsselkandidaten) einer anderen Relation verweist.

Notation

- course: {{ courseID, title, ects, taughtBy -> professor }}
- professor: {{ empID, name, rank, office }}

Alternative Notation

- course: {{ courseID, title, ects, taughtBy }}
- professor: {{ empID, name, rank, office }}

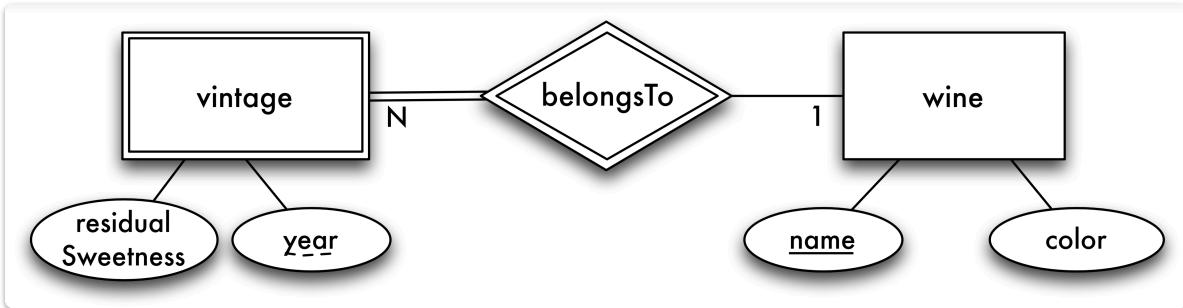
Foreign key: course.taughtBy → professor.empID (Dies bedeutet, dass das Attribut `taughtBy` in der Relation `course` ein Fremdschlüssel ist, der auf das Attribut `empID` in der Relation `professor` verweist.)

Notation für zusammengesetzte Schlüssel

$\{R.A_1, R.A_2\} \rightarrow \{S.B_1, S.B_2\}$ (Hierbei verweisen die Attribute A_1, A_2 der Relation R auf die Attribute B_1, B_2 der Relation S . Dies wird verwendet, wenn ein Fremdschlüssel aus mehreren Attributen besteht, die zusammen einen Primärschlüssel in einer anderen Relation bilden.)

Relation aus zusätzlichen Konzepten ableiten

Schwache bzw. (existenz-)abhängige Entitätstypen



Entität eines schwachen Entitätstyps sind:

- In ihrer Existenz von einem anderen, *übergeordneten (starken) Entitätstypen* abhängig. (Ein schwacher Entitätstyp kann nicht ohne die Existenz des starken Entitätstyps existieren.)
- Normalerweise nur in Kombination mit dem Schlüssel des übergeordneten Entitätstyps eindeutig identifizierbar. (Der eigene "Teilschlüssel" des schwachen Entitätstyps ist nur in Verbindung mit dem Primärschlüssel des starken Entitätstyps eindeutig.)

Wie kann man schwache Entitätstypen als Relationen abbilden?

Schwache Entitätstypen und deren identifizierende Beziehungstypen können **immer zusammengeführt** werden.

- `wine: {{ color, name }}`
- `vintage: {{ [[name]] -> wine, year, residualSweetness }}` (Der doppelt umklammerte Name `[[name]]` in `vintage` deutet auf den **partiellen Schlüssel** des schwachen Entitätstyps hin, der zusammen mit dem Primärschlüssel der starken Entität (`wine`) den vollständigen Primärschlüssel bildet. Der Pfeil `->` `wine` zeigt an, dass `name` aus `wine` als Fremdschlüssel integriert wird.)

Rekursive Beziehungstypen

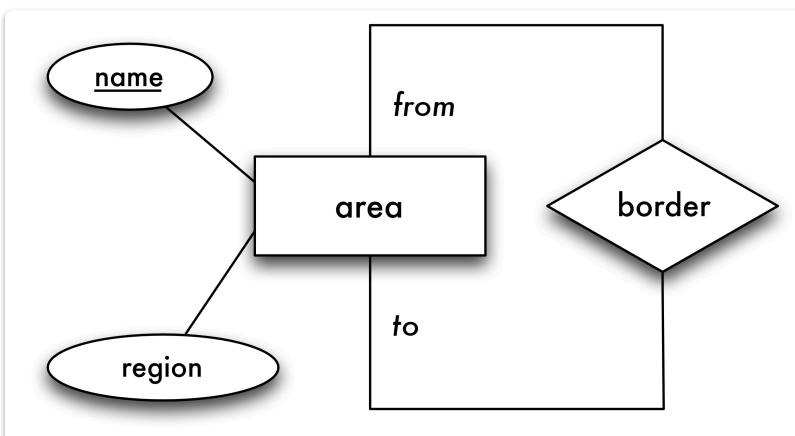


Abbildung wie normale M:N-Beziehungstypen mit Umbenennung der Fremdschlüssel.

- area: {{ name, region }} (Hier wird angenommen, dass `region` ein weiteres Attribut ist, das `area` eindeutig identifiziert oder beschreibt.)
- border: {{ [[from]] -> area, [[to]] -> area }} (Der Primärschlüssel der Relation `border` besteht aus zwei Fremdschlüsseln, die jeweils auf den Primärschlüssel von `area` verweisen. `from` und `to` sind dabei die Rollen, die die Beziehung innerhalb der `area`-Entität beschreiben.)

Rekursive funktionale Beziehungstypen

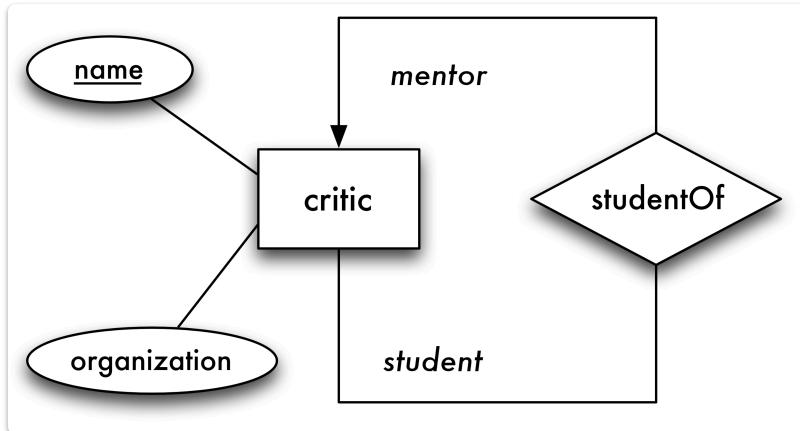
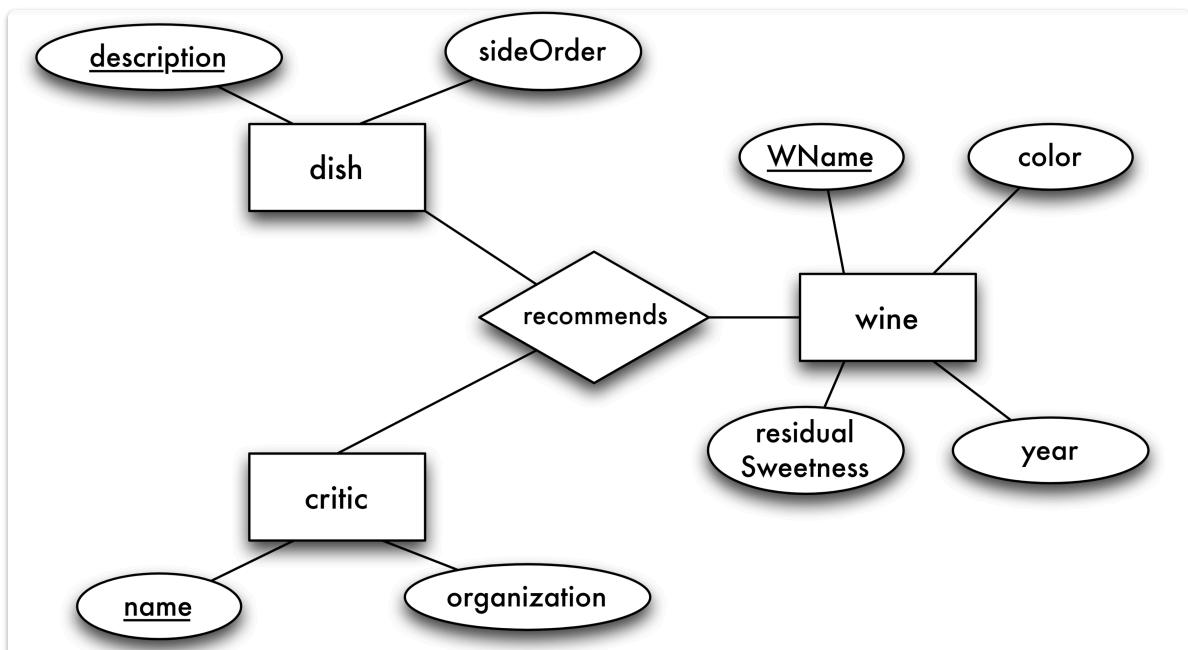


Abbildung wie normale 1:N-Beziehungstypen inklusive Zusammenführung.

- `critic: {{ [[name]], organization, mentor -> critic }}` (Hier ist `mentor` ein Fremdschlüssel, der auf den Primärschlüssel von `critic` selbst verweist, um die Mentor-Beziehung abzubilden. Da die Beziehung funktional ist (1:N), wird der Fremdschlüssel (`mentor`) direkt in die Relation des Entitätstyps auf der "N"-Seite (`critic`) integriert.)

N-äre Beziehungstypen



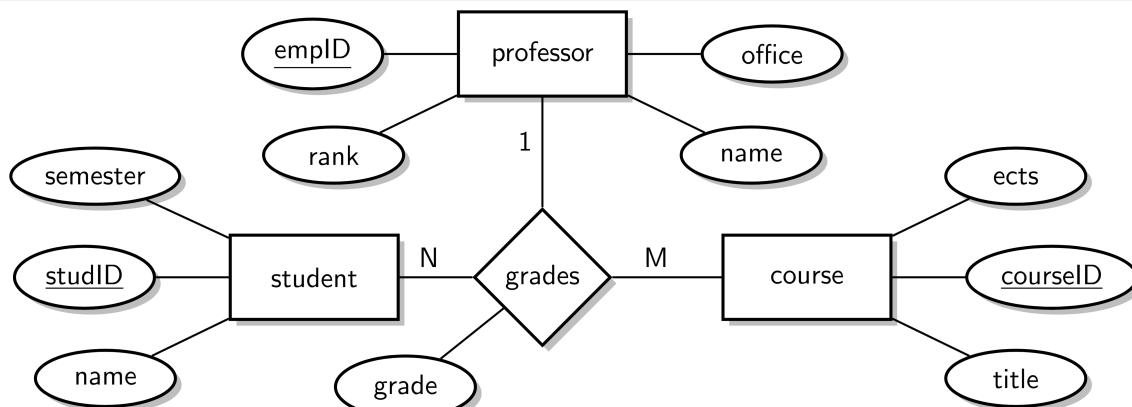
Entitätstypen wurden zunächst nach den Standardregeln abgebildet.

- critic: {{ [[name]], organization }}
- dish: {{ [[description]], sideOrder }}
- wine: {{ [[WName]], year, residualSweetness }}

N-äre Beziehungstypen (N:M:P)

- recommends: {{ WName -> wine, description -> dish, name -> critic }} (Für einen n-ären Beziehungstyp wird eine eigene Relation erstellt. Der Primärschlüssel dieser Relation besteht aus den Fremdschlüsseln der beteiligten Entitätstypen. Hier sind WName, description und name Fremdschlüssel, die jeweils auf die Primärschlüssel von wine, dish und critic verweisen.)

N:M:1-Beziehungstyp

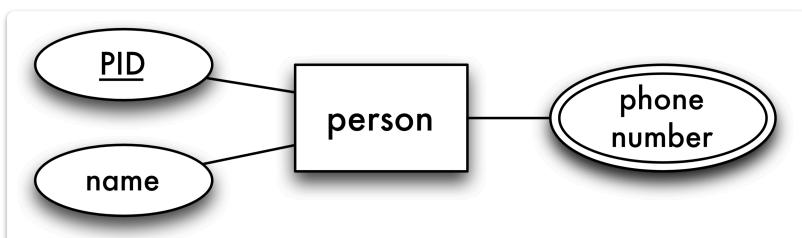


Relationen

- student: {[studID, name, semester]}
- course: {[courselID, title, ects]}
- professor: {[emplID, name, rank, office]}
- grades: {[studID → student, courselID → course, emplID → professor, grade]}

Spezielle Attribute

Mehrwertige Attribute

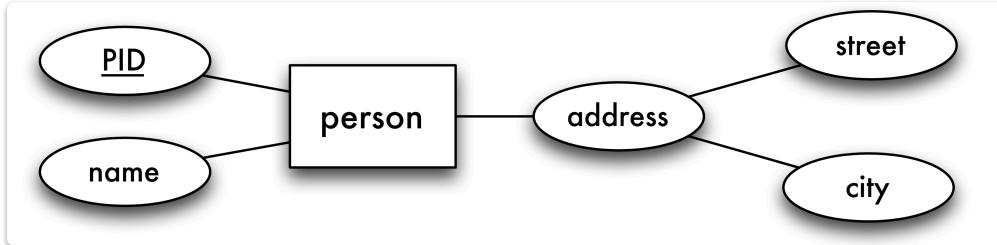


Es wird eine separate Relation für jedes mehrwertige Attribut erstellt.

Relationen:

- person: {{ PID, name }} (Der Entitätstyp person wird als normale Relation abgebildet.)
- phoneNumber: {{ [[PID]] -> person, number }} (Eine neue Relation phoneNumber wird für das mehrwertige Attribut erstellt. Ihr Primärschlüssel besteht aus dem Primärschlüssel des ursprünglichen Entitätstyps (PID , hier als Fremdschlüssel auf person verweisend) und dem Attribut des mehrwertigen Attributs (number). Dies ermöglicht mehrere Telefonnummern pro Person.)

Zusammengesetzte Attribute

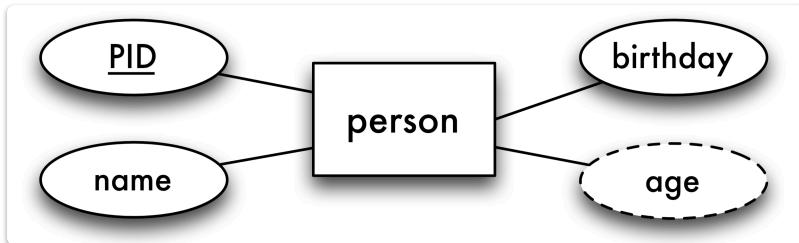


Zusammengesetzte Attribute werden der Relation des zugehörigen Entitätstyps hinzugefügt.

Relation:

- person: {{ PID, name, street, city }} (Die Komponenten des zusammengesetzten Attributs (street , city) werden direkt als einzelne Attribute in die Relation des Entitätstyps (person) aufgenommen. Das zusammengesetzte Attribut address selbst wird nicht als separate Relation abgebildet.)

Abgeleitete Attribute



Diese werden bei der Abbildung in Relationen **ignoriert** und später mittels **Views** hinzugefügt. (Abgeleitete Attribute wie age können zur Laufzeit aus anderen Attributen (birthday) berechnet werden und müssen daher nicht physisch in der Datenbank gespeichert werden. Views sind virtuelle Tabellen, die das Ergebnis einer Abfrage sind und zur Darstellung abgeleiteter Daten verwendet werden können.)

Überblick der Schritte

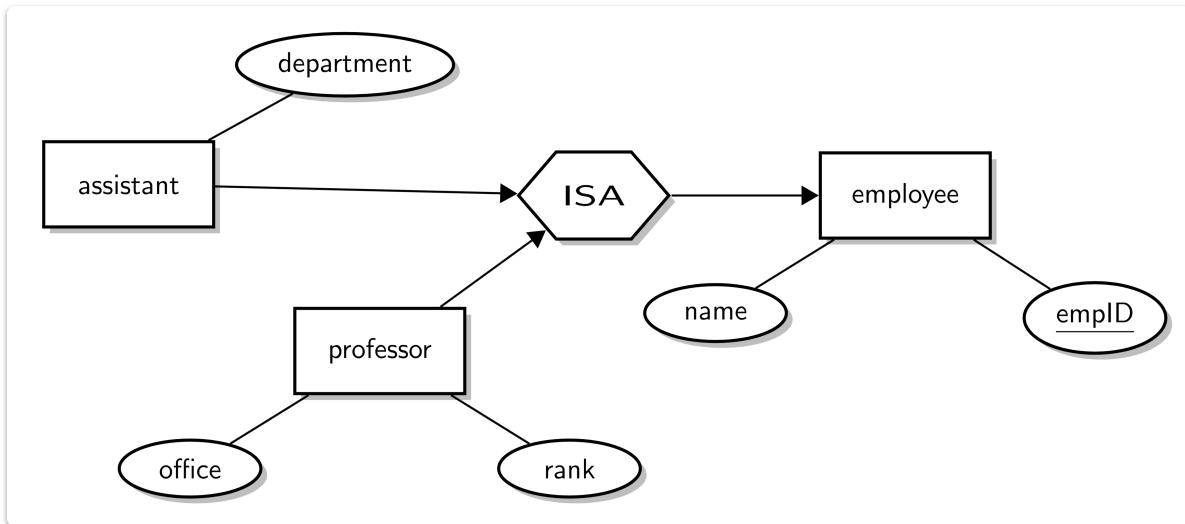
- **Regulärer Entitätstyp:** Relation erstellen, spezielle Attributtypen beachten.
- **Schwacher Entitätstyp:** Relation erstellen, Primärschlüssel aus partiellem Schlüssel und Fremdschlüssel des starken Entitätstyps bilden.

- **1:1 binärer Beziehungstyp:** Erweitern einer Relation mit Fremdschlüssel.
- **1:N binärer Beziehungstyp:** Erweitern einer Relation mit Fremdschlüssel.
- **M:N Beziehungstyp:** Erstellen einer neuen Relation.
- **N-ärer Beziehungstyp:** Erstellen einer neuen Relation.

Relationale Modellierung der Generalisierung

Das relationale Modell unterstützt keine Generalisierung und kann daher keine Vererbung ausdrücken.

→ Generalisierung wird **simuliert**.



Arten um dieses ER-Diagramm auf Relationen abzubilden

Alternative 1: Hauptklassen

Ein bestimmtes Entity wird abgebildet als *ein Tupel* in einer einzigen Relation (zur zugehörigen Hauptklasse). (Das bedeutet, alle Attribute der Subklassen werden in die Relation der Superklasse integriert. Nicht zutreffende Attribute sind dann NULL.)

- `employee: {{ empID, name, rank, office, department }}` (Die `employee`-Tabelle enthält alle Attribute der Subklassen. Wenn ein Mitarbeiter z.B. kein Professor ist, wären `rank` und `office` NULL.)
- `professor: {{ empID, name, rank, office }}` (Alternativ könnte auch eine separate Tabelle für `professor` erstellt werden, die den Primärschlüssel der `employee`-Tabelle enthält und die spezifischen Attribute von `professor`.)
- `assistant: {{ empID, name, department }}` (Ähnlich wie bei `professor`, eine separate Tabelle für `assistant` mit dem Primärschlüssel der `employee`-Tabelle und den spezifischen Attributen von `assistant`.)

employee	
empID	name
2123	P. Müller
2124	A. Schmidt

employee:
{{ empID, name }}

professor			
empID	name	rank	office
2125	Socrates	C4	226
2126	Russel	C3	232
2127	Kopernikus	C3	310
2128	Curie	C4	36

professor:
{{ empID, name, rank, office }}

assistant		
empID	name	department
2150	C. Meyer	DBS
2151	B. Fischer	Physics

assistant:
{{ empID, name, department }}

Alternative 2: Partitionierung

Teile eines bestimmten Entitys werden in *mehreren Relationen* abgebildet, der Schlüssel wird dupliziert. (Das bedeutet, die Superklasse hat eine Relation, und jede Subklasse hat eine eigene Relation, die jeweils den Primärschlüssel der Superklasse als Fremdschlüssel enthält.)

- employee: {{ empID, name }} (Diese Tabelle enthält die gemeinsamen Attribute aller Mitarbeiter.)
- professor: {{ empID -> employee, rank, office }} (Die professor -Tabelle enthält den Fremdschlüssel empID (der auf employee verweist) und die spezifischen Attribute rank und office . empID ist hier auch der Primärschlüssel.)
- assistant: {{ empID -> employee, department }} (Die assistant -Tabelle enthält den Fremdschlüssel empID (der auf employee verweist) und das spezifische Attribut department . empID ist hier auch der Primärschlüssel.)

employee	
empID	name
2123	P. Müller
2124	A. Schmidt
2125	Socrates
...	...
2150	C. Meyer
2151	B. Fischer

employee:
 $\{[\underline{\text{empID}}, \text{name}]\}$

professor		
empID	rank	office
2125	C4	226
...

professor:
 $\{[\underline{\text{empID}} \rightarrow \text{employee}, \text{rank}, \text{office}]\}$

assistant	
empID	department
2150	DBS
2151	Physics

assistant:
 $\{[\underline{\text{empID}} \rightarrow \text{employee}, \text{department}]\}$

Alternative 3: Vollständige Redundanz

Ein bestimmtes Entity wird *redundant* in mehreren Relationen gespeichert inklusive aller geerbten Attribute. (Jede Subklasse und die Superklasse erhalten eigene Relationen, die alle relevanten Attribute, einschließlich der geerbten, enthalten. Dies führt zu Redundanz, da dieselben Informationen mehrfach gespeichert werden.)

- employee: $\{[\underline{\text{empID}}, \text{name}]\}$
- professor: $\{[\underline{\text{empID}}, \text{name}, \text{rank}, \text{office}]\}$ (Die professor -Tabelle enthält alle Attribute, die für einen Professor relevant sind, inklusive der geerbten von employee .)
- assistant: $\{[\underline{\text{empID}}, \text{name}, \text{department}]\}$ (Die assistant -Tabelle enthält alle Attribute, die für einen Assistenten relevant sind, inklusive der geerbten von employee .)

employee	
empID	name
2123	P. Müller
2124	A. Schmidt
2125	Socrates
...	...
2150	C. Meyer
2151	B. Fischer

employee:
 $\{ [\underline{\text{empID}}, \text{name}] \}$

professor			
empID	name	rank	office
2125	Socrates	C4	226
...

professor:
 $\{ [\underline{\text{empID}}, \text{name}, \text{rank}, \text{office}] \}$

assistant		
empID	name	department
2150	C. Meyer	DBS
2151	B. Fischer	Physics

assistant:
 $\{ [\underline{\text{empID}}, \text{name}, \text{department}] \}$

Alternative 4: Eine einzige Relation

Alle Entitys werden in *einer einzigen Relation* gespeichert und ein besonderes Attribut hinzugefügt, welches die Zugehörigkeit zu einem bestimmten Subtyp angibt. (Dies ist eine gängige Methode, bei der eine einzige breite Tabelle für die Superklasse erstellt wird, die alle Attribute aller Subklassen enthält. Ein zusätzliches "Typ"-Attribut gibt an, zu welcher spezifischen Subklasse das jeweilige Tupel gehört.)

- employee: $\{ [\underline{\text{empID}}, \text{name}, \text{type}, \text{rank}, \text{office}, \text{department}] \}$ (Das `type`-Attribut könnte Werte wie 'employee', 'professor' oder 'assistant' enthalten. Attribute, die für einen bestimmten Typ nicht zutreffen, sind NULL.)

employee: $\{ [\underline{\text{empID}}, \text{name}, \text{type}, \text{rank}, \text{office}, \text{department}] \}$

employee					
empID	name	type	rank	office	department
2123	P. Müller	employee	⊥	⊥	⊥
2124	A. Schmidt	employee	⊥	⊥	⊥
2125	Socrates	professor	C4	226	⊥
2126	Russel	professor	C3	232	⊥
2127	Kopernikus	professor	C3	310	⊥
2128	Curie	professor	C4	36	⊥
2150	C. Meyer	assistant	⊥	⊥	DBS
2151	B. Fischer	assistant	⊥	⊥	Physics

Allgemeine Lösung

Im Allgemeinen ist **Alternative 2 (Partitionierung)** zu verwenden.

Teile eines bestimmten Entitys werden in mehreren Relationen abgebildet, der Schlüssel wird dupliziert. (Diese Alternative wird bevorzugt, da sie eine gute Balance zwischen Redundanz und Komplexität bietet und NULL-Werte minimiert.)

- `employee: {{ empID, name }}`
 - `professor: {{ empID -> employee, rank, office }}` (Der Primärschlüssel `empID` der `professor`-Relation ist gleichzeitig ein Fremdschlüssel, der auf die `employee`-Relation verweist.)
 - `assistant: {{ empID -> employee, department }}` (Ähnlich ist `empID` der `assistant`-Relation Primär- und Fremdschlüssel zu `employee`.)
-

Zusammenfassung

ER-Diagramme auf Relationen abbilden:

- **Entitätstypen:** Abbildung als Relation.
- **Binäre Beziehungstypen:**
 - 1:1-Beziehungstypen: Integration des Fremdschlüssels in eine der beteiligten Relationen.
 - 1:N-Beziehungstypen: Integration des Fremdschlüssels in die Relation auf der "N"-Seite.
 - M:N-Beziehungstypen: Erstellung einer *neuen Relation* für den Beziehungstyp, deren Primärschlüssel aus den Fremdschlüsseln der beteiligten Entitätstypen besteht.
- **N-stellige Beziehungstypen:** Erstellung einer *neuen Relation*, deren Primärschlüssel aus den Fremdschlüsseln der beteiligten Entitätstypen besteht.
- **Schwache Entitätstypen:** Integration in die Relation des starken Entitätstyps oder Erstellung einer eigenen Relation mit einem Primärschlüssel, der den partiellen Schlüssel und den Fremdschlüssel zum starken Entitätstyp kombiniert.
- **Rekursive Beziehungstypen:** Abbildung analog zu den entsprechenden binären oder n-stelligen Beziehungstypen, wobei die Fremdschlüssel auf dieselbe Relation verweisen.
- **Generalisation:**
 - Die „**Partitionierungsvariante**“ wird in den meisten Anwendungen bevorzugt. (Dies entspricht Alternative 2, bei der Super- und Subklassen eigene Relationen erhalten und über Fremdschlüssel verknüpft sind.)
- Die behandelten „**Abbildungsregeln**“ haben das Ziel, die **Anzahl von Relationen zu minimieren, nicht notwendigerweise die NULL-Werte**. (Dies ist ein wichtiger Design-Grundsatz, der die Effizienz der Datenbank beeinflusst. Die Minimierung von Relationen kann die Komplexität von Abfragen reduzieren, auch wenn dies in manchen Fällen zu NULL-Werten führen kann.)