Datenbanken
Wintersemester 2018/19
Prof. Dr. W. May

1. Übungsblatt: ER-Modell und Relationales Modell

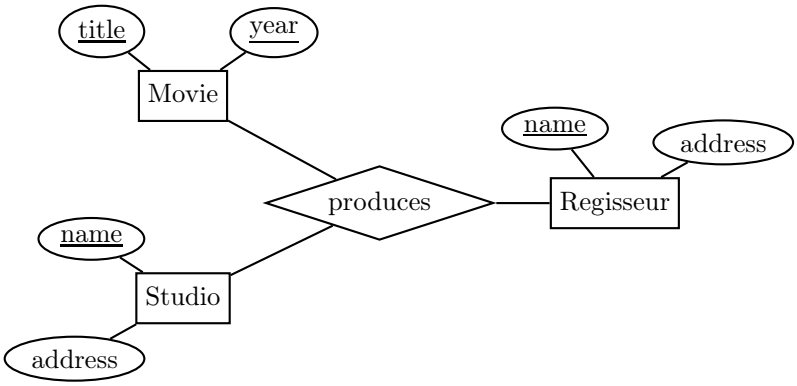
Besprechung voraussichtlich am 1./7./8.11.2018

Aufgabe 1 (ER-Modell: Film) Geben Sie ein ER-Modell für den folgenden Sachverhalt an: Filme werden in Filmstudios von Regisseuren gedreht. Filmstudios gehören einem Besitzer. In Filmen treten Schauspieler auf. Schauspieler erhalten eine Gage für jeden ihrer Verträge.

Entwickeln Sie zuerst ein einfaches Modell, und überlegen Sie dann, wie und ob Sie das Modell ergänzen könnten, um z.B. zu modellieren, dass sowohl Schauspieler als auch Regisseure und Besitzer von Filmstudios Personen sind, und manche Personen auch im selben Film oder in verschiedenen Filmen in mehreren dieser Rollen auftreten.

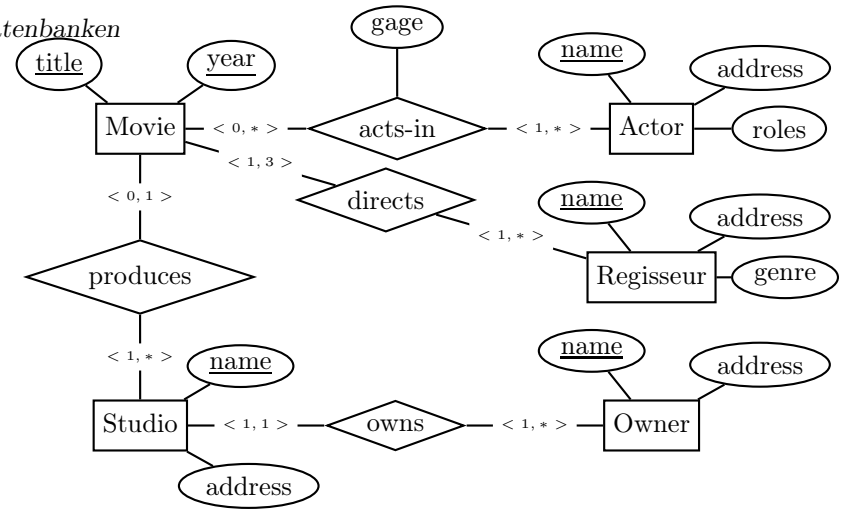
Nur die (bzw. eine) korrekte Lösung zu zeigen, wäre zu einfach und auch didaktisch nicht sinnvoll. Aus diesem Grund werden hier verschiedene Wege und Holzwege diskutiert.

Erster Ansatz: Man betrachtet den Satz "Filme werden in Filmstudios von Regisseuren gedreht." Naheliegend ist hier eine dreistellige Beziehung:



[Präsentation: Umsetzung in Relationen.] Man sieht an der Relation, bzw. an der anzustellenden Kardinalitätsbetrachtung, dass diese Modellierung nicht davor bewahrt, für einen Film, der von mehreren Regisseuren gemeinsam gedreht wird, auch unterschiedliche Studios zu speichern. Also sollte man diese Modellierung so nicht wählen.

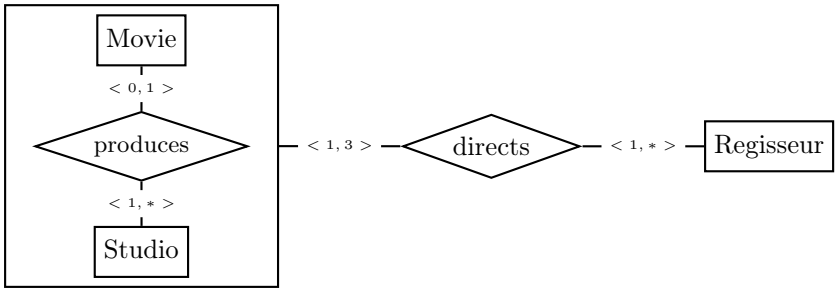
Naheliegend ist, die Beziehung in zwei zweispaltige Beziehungen (*Movie-Studio*) und (*Movie-Regisseur*) aufzulösen, womit man insgesamt die folgende "Musterlösung" erreicht:



Roles ist ein spezifisches Attribut von Schauspielern, das angibt, welche Arten von Rolle ein Schauspieler spielen kann (Erweiterung: die Beziehung *acts-in* ebenfalls *roles* zu erweitern, das angibt, welche Art von Rolle ein Schauspieler in diesem Film spielt). *Genre* ist ein spezifisches Attribute von Regisseuren, das angibt, welche Art von Filmen ein Regisseur dreht (auch hier kann man überlegen, *genre* als Attribut des Films zu nehmen).

Alternative Modellierung der linken Seite.

Eine mögliche Alternative ist, die dreistellige Beziehung *directs* aufzuspalten, indem man die Beziehung *produces* zwischen einem Studio und einem Film betrachtet, diese aggregiert (Produktion), und das Aggregat in eine *directs*-Beziehung mit Regisseuren stellt. *Owner* stehen weiterhin in Beziehung mit ihrem Studio, Schauspieler kann man wahlweise in Beziehung mit dem Film oder der Produktion stellen.



Hinweis: man erhält dasselbe relationale Modell, wenn man an der richtigen Stelle einbezieht, dass die 1:n-Beziehung zwischen Film und Studio dazu führt, dass der Schlüssel von *Production* nur (*title,year*) ist. (Es ist somit auch eigentlich keine echte Aggregation, da jede Instanz der Aggregation auch genau einer Instanz des Entitätstyps "Film" entspricht.)

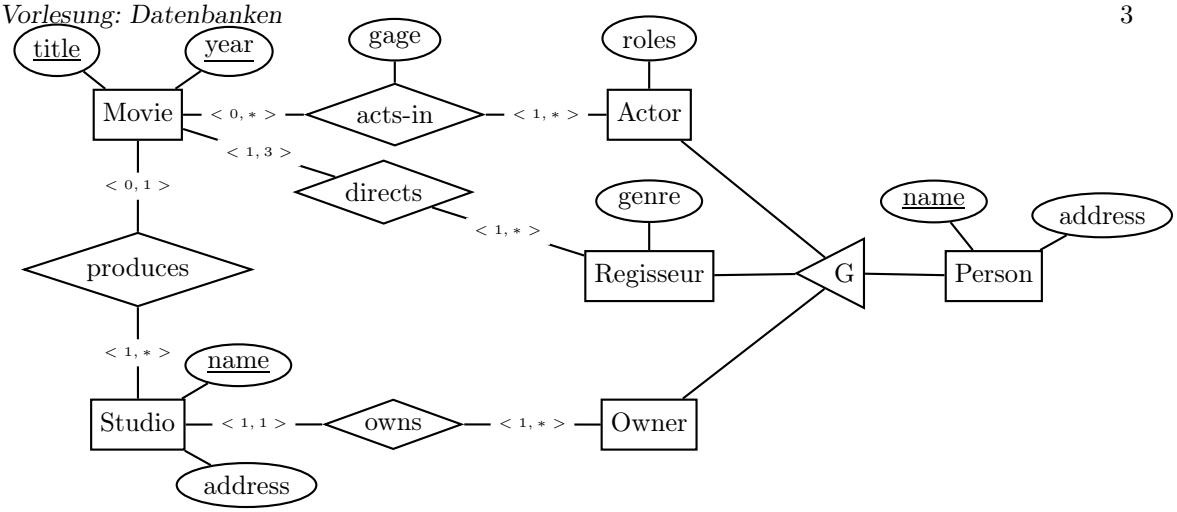
Hinweis: eine Aggregation (*Film, Regisseur*) und eine Beziehung des Aggregates zu *Studio* ist keine geeignete Modellierung, da dann für Filme, die von mehreren Regisseuren zusammen gedreht würden, auch verschiedene Studios angegeben werden könnten.

Modellierung mit Generalisierung "Person".

Betrachtet man bei der obigen Modellierung die Umsetzung ins relationale Modell, so stellt man fest, Information zu Personen, die z.B. sowohl als Regisseur als auch als Schauspieler gespeichert sind, *redundant* gehalten wird. Dies kostet nicht nur Platz, sondern kann auch zu inkonsistenten Datenbankzuständen führen:

Actor(AS, Hollywood Drive 42 - LosAngeles) und Regisseur(AS, Graben 1 - Wien).

Es ist daher sinnvoll, einen Entitätstyp *Person* als Generalisierung dieser Typen einzuführen:



Generalisierung bedeutet hier, dass

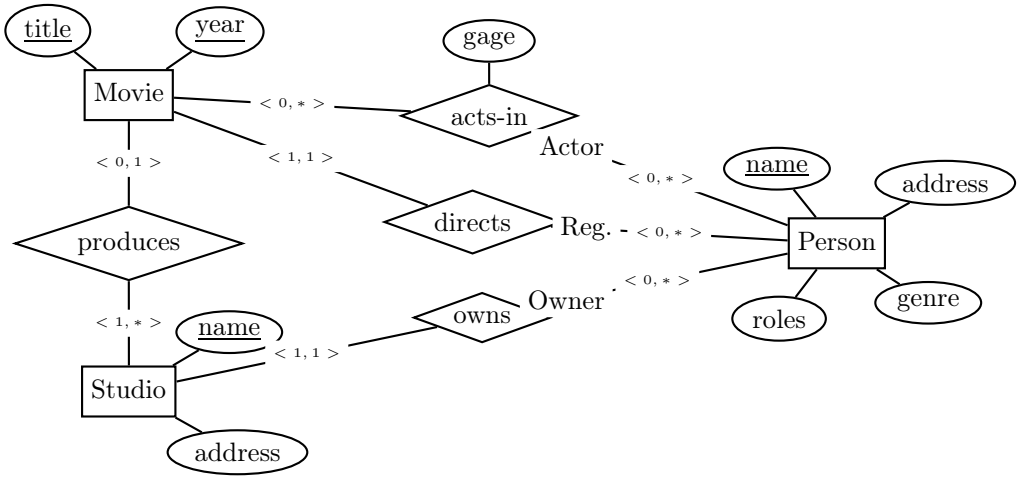
$$Persons = Actors \cup Regisseurs \cup Owners$$

ist. Würde man stattdessen eine Spezialisierung wählen, könnte man auch Personen haben, die nicht in eine dieser spezielleren Klassen fallen.

Bei einer Umsetzung ins relationale Modell erhält man je eine Tabelle für *Person(Name,Address)*, *Actor(Name,role)* (ggf. mehrwertig), *Regisseur(Name,genre)*, die jeweils den Primärschlüssel *Name* einer Person referenzieren.

Modellierung nur mit "Person" und Rollen.

Als dritte Möglichkeit könnte man nur Personen modellieren und ihre Rollen (in der Anwendungswelt) durch Rollen(bezeichner im ER-Modell) darstellen:



Die Attribute *roles* und *genre* müssen damit alle bei Person angesiedelt sein. Damit ist nicht klar, dass z.B. nur Schauspieler das Attribut *roles* besitzen. Außerdem sind potentiell viele Nullwerte in der Tabelle enthalten.

Aufgabe 2 (Umsetzung in das relationale Modell: Film) In Aufgabe 1 haben Sie ein ER-Modell für eine kleine Filmdatenbank erstellt. Transformieren Sie dieses in ein relationales Modell.

Einfachste Modellierung

Annahme: roles wird als String als Kommaliste abgelegt

Movie: (<u>title</u>, <u>year</u>)
Studio: (<u>name</u>, address)
Actor: (<u>name</u>, address, roles)
Regisseur: (<u>name</u>, address, genre)
Owner: (<u>name</u>, address)

produces: (studio, <u>title</u>, <u>year</u>)
acts-in: (<u>actor</u>, <u>title</u>, <u>year</u>, gage)
directs: (<u>regisseur</u>, <u>title</u>, <u>year</u>)
owns: (<u>owner</u>, <u>studio</u>)

Alternative, um zu einem Schauspieler mehrere Rollencharakterisierungen ablegen zu können:

Actor: (<u>name</u>, address)
ActorRoles: (<u>name</u>, <u>role</u>)

Alternativen

Da ein Film jeweils nur in einem Studio gedreht werden kann (siehe Beziehungskomplexitäten im ER-Diagramm), kann man die *produces*-Beziehung in *Movie* mit hineinnehmen:

Movie: (<u>title</u>, <u>year</u>, studio)

Modelliert man Filmproduktion als Aggregation aus *Movie, Studio*, erhält man dafür genau die Relation

production: (<u>title</u>, <u>year</u>, studio)

Die Produktion steht nun in Beziehung zu Regisseuren, womit man hierfür die Relation

directs: (regisseur, title, year)

erhält.

Generalisierung als Personen

... verschiedene Möglichkeiten.

- Personen mit Name und Adresse, roles und genre jeweils in Relationen Actor bzw. Regisseur die als Fremdschlüssel den Namen einer Person haben. Die Fremdschlüsselbeziehungen der Relationen zu den Beziehungstypen referenzieren die Relationen Actor, Regisseur, Owner:
  Movie: (<u>title</u>, <u>year</u>)
  Studio: (<u>name</u>, address)
  Person: (<u>name</u>, address)
  Actor: (<u>name</u>, roles)
  Regisseur: (<u>name</u>, genre)
  Owner: (<u>name</u>)

  produces: (studio, <u>title</u>, <u>year</u>)
  acts-in: (<u>actor</u>, <u>title</u>, <u>year</u>, gage)
  directs: (<u>regisseur</u>, <u>title</u>, <u>year</u>)
  owns: (<u>owner</u>, <u>studio</u>)

  $Actor.name \rightarrow Person.name$ (und analog)
  $acts\text{-}in.actor \rightarrow Actor.name$
  Aufgrund der detaillierten Modellierung sind die Konsistenzbedingungen an die Datenbank sehr scharf.

- Die vier Relationen *Person*, *Actor*, *Regisseur*, *Owner* kosten relativ viel Platz, weil die Namen mehrfach abgelegt sind (als *Person(AS, LosAngeles)*, *Actor(AS, hero)* und *Regisseur(AS, action)*. Man kann sie einsparen, wenn man stattdessen *roles* und *genre* als Attribute zu *Person* nimmt und mit Nullwerten auffüllt (z.B. *Person(AS, LosAngeles, hero, action, NULL)*). Dies entspricht auch dem ER-Diagramm, wo nur der Entitätstyp *Person* verwendet wird, und mit Rollenbezeichnungen gearbeitet wird:

  Movie: (<u>title</u>, <u>year</u>)
  Studio: (<u>name</u>, address)
  Person: (<u>name</u>, address, roles, genre)

  produces: (studio, title)
  acts-in: (person as actor, title, year, gage)
  directs: (person as regisseur, title, year)
  owns: (person as owner, studio)

  $acts.name \rightarrow Person.name$ etc.

  Man verliert die Integritätsbedingungen, dass nur Schauspieler das Attribut *roles* haben sowie dass die *acts*-Beziehung nur mit Schauspielern besteht (und analoges).
- Nun hat man ziemlich viele Nullwerte in *roles*, *genre*. Die meisten Personen werden Schauspieler sein. Man kann also z.B. das Attribut *roles* bei Person lassen, und *genre* wieder in separate Relationen für Regisseure (mit nur relativ wenigen Personen) legen:

  Movie: (<u>title</u>, <u>year</u>)
  Studio: (<u>name</u>, address)
  Person: (<u>name</u>, address, roles)
  Regisseur: (<u>name</u>, genre)
  Owner: (<u>name</u>)

  produces: (studio, title, year)
  acts-in: (person, title, year, gage)
  directs: (regisseur, title, year)
  owns: (owner, studio)

Aufgabe 3 (Lecturers, Courses, Students) Studenten hören Vorlesungen bei Dozenten. Zur Vereinfachung wird angenommen, dass jede Vorlesungen nur zu *einem wöchentlichen Termin* in einem bestimmten Raum stattfindet.

Modellieren Sie den Sachverhalt mit einem ER-Diagramm, transformieren Sie es in das relationale Modell und betrachten Sie geeignete Beispieldaten.

Betrachten Sie verschiedene Szenarien:

a) jede Vorlesung wird von *einem* Dozenten gehalten.

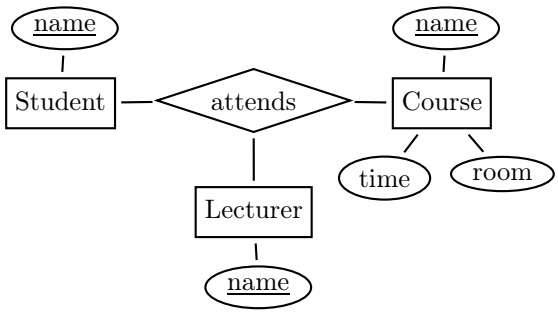
b) Vorlesungen können auch von mehreren Dozenten gemeinsam gehalten werden; z.B. *Informatik I* von *Müller* von Oktober bis Weihnachten, und von *Meier* den Rest bis zum Semesterende.

c) es gibt große (Anfänger)vorlesungen, die parallel von zwei oder mehr Dozenten in unterschiedlichen Hörsälen gehalten werden.

d) wie (a), aber jetzt kann jede Vorlesung an mehreren wöchentliche Termine in ggf. verschiedenen Räumen stattfinden (z.B. "Datenbanken" dienstags 14-16 Uhr im MN06 und mittwochs 10-12 im MN09).

---

a) Jede Vorlesung wird von *einem* Dozenten gehalten. Der erste Ansatz eines ER-Diagramms könnte folgendes ergeben:



attends		
Student	Course	Lecturer
S1	Databases	WM
S2	Databases	WM
S2	CompNetw	XF
S3	CompNetw	XF
S3	SQL	WM
S4	SQL	WM
S1	SQL	WM

Man kann bei dieser Modellierung nicht ausdrücken, dass ein Kurs von genau einem Dozenten gehalten wird. Bei der dreistelligen Relation wäre ein Datenbankzustand, der sowohl (S1, DB, WM) und (S3, DB, XF) enthält, erlaubt.

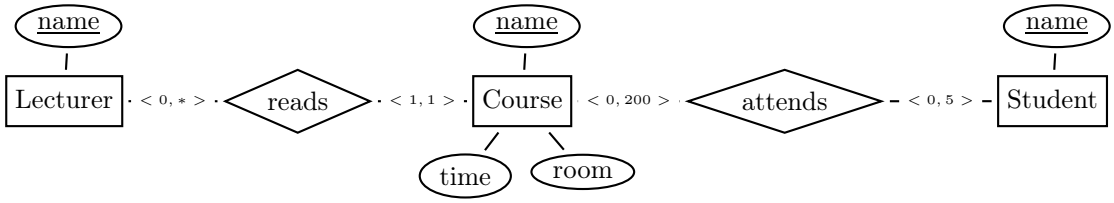
Das Schema dieser dreistelligen Relation ist

attends: (Student, Course, Lecturer)

wobei nicht wie zu erwarten alle drei Fremdschlüssel zusammen den neuen Key bilden. Dies ist grundsätzlich verdächtig.

Es gibt hier eine "funktionale Abhängigkeit" *Course* $\rightarrow$ *Lecturer* (ein Dozent kann trotzdem mehrere Vorlesungen halten), d.h. das Nichtschlüsselattribut "Lecturer" hängt nicht voll funktional vom Key ab, sondern eben nur von *Course*.

Man muss das Modell bzw. die Relation nach dieser funktionalen Abhängigkeit aufspalten: Logisch unabhängige Zusammenhänge bestehen zwischen Kurs und dem Vorlesenden,reads(Course, Lecturer), sowie zwischen Studenten und Kursen, attends(Student,Course):



Durch die Zerlegung in 2 zweistellige Beziehungen ergibt sich damit im ersten Schritt das folgende Schema:

Lecturer: (<u>name</u>, address)
Course: (<u>name</u>, ects, ...)
Student: (<u>name</u>, address, ...)
reads: (<u>Course</u>, Lecturer)
attends: (<u>Student</u>, <u>Course</u>)

Da in *reads* nur *Course* der Schlüssel ist (Kardinalität: jeder Kurs steht mit genau einem Lecturer in der *reads*-Beziehung), kann man diese auch nach *Course* reinziehen:

Lecturer: (<u>name</u>, address)
Course: (<u>name</u>, Lecturer, ects, ...)
Student: (<u>name</u>, address, ...)
attends: (<u>Student</u>, <u>Course</u>)

Fremdschlüssel-Integritätsbedingungen:

course.Lecturer $\subseteq$ Lecturer.name
attends.Student $\subseteq$ Student.name
attends.Course $\subseteq$ Course.name
(man schreibt häufig auch "course.Lecturer $\rightarrow$ Lecturer.name" für "... referenziert ...", was aber für Anfänger zu Verwechslungen mit der Notation von "$\rightarrow$" für funktionale Abhängigkeiten, die *innerhalb* einer Tabelle bestehen, führen kann)

$\Rightarrow$ Wenn man ein gutes ER-Modell hat, muss man sich nicht mit funktionalen Abhängigkeiten und Normalisierungstheorie beschäftigen. Das intuitive Wissen über diesen formalen Hintergrund hilft aber weiter, um das erhaltene Modell (mit Hilfe von Beispieldaten) zu validieren.

b) Wenn eine Vorlesung von mehreren Dozenten gemeinsam angeboten wird, kann die Modellierung dieselbe bleiben, nur die Kardinalität von "reads" bzgl. "Course" muss auf <1,*> angepasst werden. Dies hat allerdings Konsequenzen auf die Transformation in das relationale Modell:

Lecturer: (<u>name</u>, address)
Course: (<u>name</u>, ects, ...)
Student: (<u>name</u>, address, ...)
reads: (<u>Course</u>, <u>Lecturer</u>)
attends: (<u>Student</u>, <u>Course</u>)

Jetzt ist *reads* eine echte n:m-Beziehung, und kann nicht nach *Course* reingezogen werden.

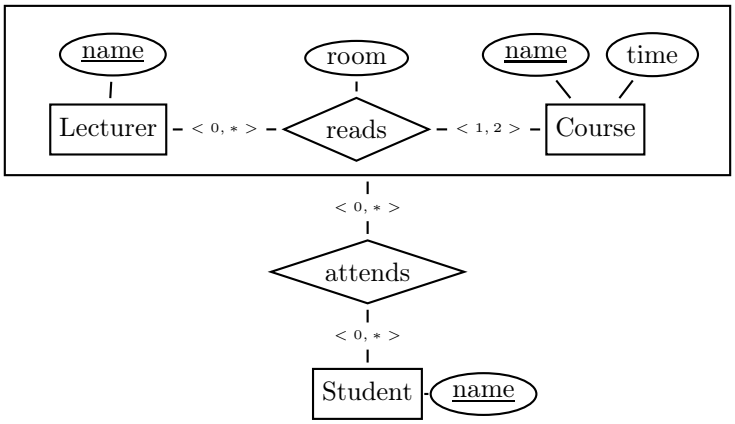
Fremdschlüssel-Integritätsbedingungen:

reads.Lecturer $\subseteq$ Lecturer.name
reads.Course $\subseteq$ Course.name
attends.Student $\subseteq$ Student.name
attends.Course $\subseteq$ Course.name

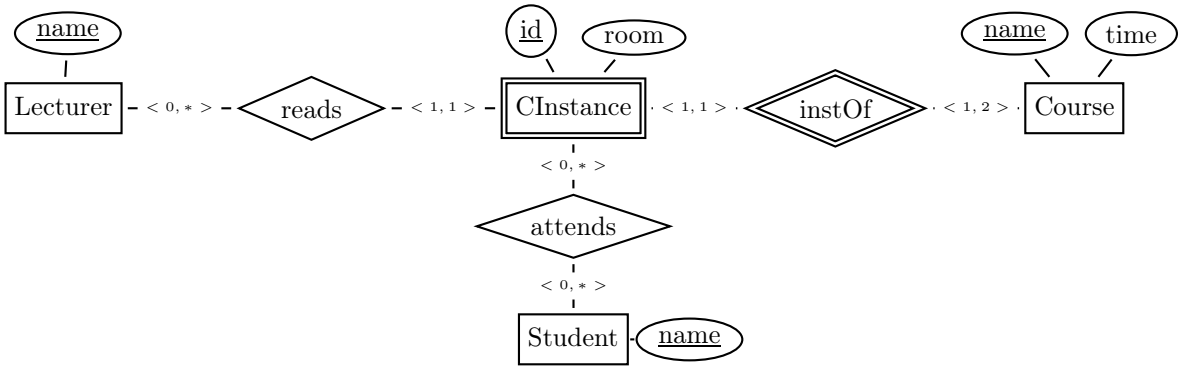
c) Die obige Modellierung funktioniert nicht mehr, wenn große Vorlesungen parallel bei zwei Dozenten (selbe Zeit, unterschiedliche Räume) angeboten werden *Informatik I* Dienstags 14-16 von *Schmidt* in HS1 und parallel von *Schulze* in HS3.

Möglichkeit der Modellierung: Aggregierung der *reads*-Beziehung zwischen Dozent und Vorle-

sung (entsprechend "Informatik I Kurs A/B"). Auch das Attribut "Raum" wird nun der Beziehung zugeordnet, während die (gemeinsame) Zeit beim Kurs verbleibt.



Andere Möglichkeit der Modellierung: Die "Instanzen" (*Informatik I-A* und *Informatik I-B*) einer Vorlesung werden als separater Entitätstyp *C(ourse)Instance* modelliert:



Beide Modellierungen sind insofern äquivalent, dass bei der Transformation in das relationale Modell ziemlich dasselbe rauskommt (die beiden $< 1, 1 >$-Kardinalitäten von *C(ourse)Instance* verkörpern die Aggregierung der Beziehung).

Bei der Variante mit Aggregierung erhält das Aggregat den Namen der Beziehung (oder man wählt einen anderen, z.B. *CourseInstance*, wobei jetzt die Schlüssel der *reads*-Beziehung hier genommen werden und man quasi die Instanz "Informatik I bei Müller" betrachtet:

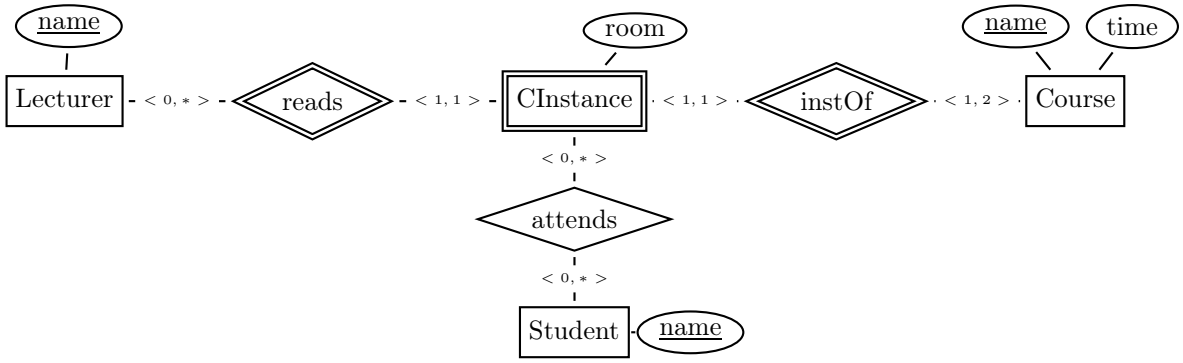
Lecturer: (name, address)
Course: (name, ects, ...)
Reads=CourseInstance: (course, Lecturer)
Student: (name, address, ...)
attends: (Student, Course, optional: id, wenn man eine feste Zuordnung haben will)
(*reads* wird wegen der $< 1, 1 >$-Kardinalität nach *CourseInstance* reingezogen)

Variante mit expliziten Instanzen:

Lecturer: (name, address)
Course: (name, ects, ...)
CourseInstance: (course, id, Lecturer)
Student: (name, address, ...)
attends: (Student, Course, optional: id, wenn man eine feste Zuordnung haben will)
(*reads* wird wegen der $< 1, 1 >$-Kardinalität nach *CourseInstance* reingezogen)

Fast dasselbe hätte man erhalten, wenn man im ER-Diagramm *CourseInstance* zu beiden Seiten

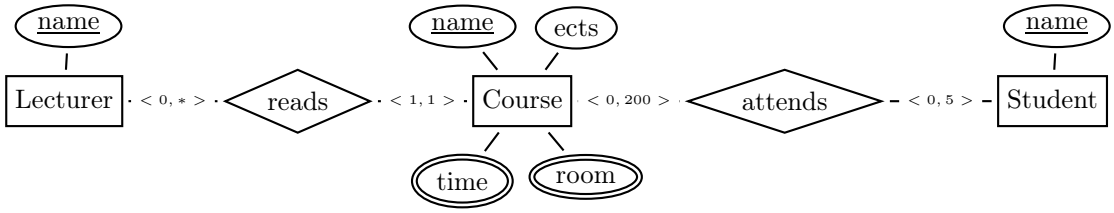
als *weak entity type* gemacht hätte, und somit die Keys *Course.Name* und *Lecturer.Name* geerbt hätte:



Lecturer: (<u>name</u>, address)
Course: (<u>name</u>, ects, ...)
CourseInstance: (<u>course</u>, <u>Lecturer</u>)
Student: (<u>name</u>, address, ...)
attends: (<u>Student</u>, <u>Course</u>, optional: <u>id</u>, wenn man eine feste Zuordnung haben will)
(*reads* wird wegen der $< 1, 1 >$-Kardinalität nach *CourseInstance* reingezogen)

- Bei attends: (<u>Student</u>, <u>Course</u>, <u>id</u> wird jeder Student fest zu einer der Instanzen (*Informatik I bei Müller* bzw. *Informatik I-A*) in Verbindung gesetzt. Dass er dabei eine bestimmte Vorlesung nur einmal hören kann, kann so nicht beschrieben werden. Dies muss zusätzlich durch Text angegeben werden.
- Üblicherweise ist es aber so, dass jeder Student an jedem Tag in die Vorlesungsinstanz gehen kann, die ihm besser gefällt (z.B. ein Kurs auf dem Zentralcampus, der andere auf dem Nordcampus) – die Klausur ist für beide Kurse gemeinsam.
  Daher kann man die Beziehung *attends* auch einfach als attends: (<u>Student</u>, <u>Course</u>) modellieren.

d) Jeder Kurs hat mehrere Zeiten, ggf in verschiedenen Räumen:
Der erste Versuch eines ER-Modells wäre einfach, *Course.Time* und *Course.Room* beide als mehrwertige Attribute zu modellieren:



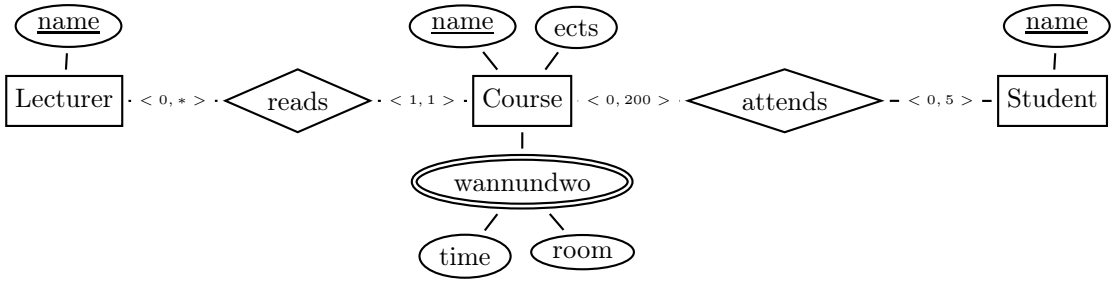
Bei der Umsetzung ins relationale Modell müssen die beiden mehrwertigen Attribute getrennt behandelt werden:

Course	
<u>name</u>	**ects**
Databases	5
CompNetw	5
:	:

CourseTime	
<u>name</u>	**<u>time</u>**
Databases	Tue 14-16
Databases	Wed 10-12
CompNetw	Mo 10-12
CompNetw	Thu 14-16
:	:

CourseRoom	
<u>name</u>	**<u>room</u>**
Databases	MN06
Databases	MN08
CompNetw	MN09
:	:

Bei dieser Modellierung wird nicht klar, dass gemeint ist "Datenbanken findet Dienstags 12-14 im MN06 und Mittwochs 10-12 im MN08" statt , sondern alle 4 Kombinationen wären die logische Semantik. Die Telematik-Vorlesung findet zu beiden Terminen im selben Hörsaal statt.
Es handelt sich also nicht um zwei *getrennte* Attribute, sondern um ein strukturiertes Attribut:

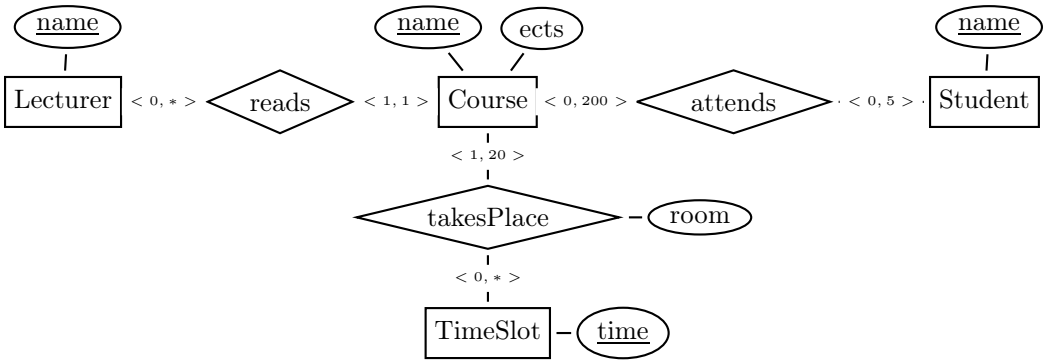


Course	
name	**ects**
Databases	5
CompNetw	5
:	:

CourseWhenWhere		
name	**time**	**room**
Databases	Tue 14-16	MN06
Databases	Wed 10-12	MN08
CompNetw	Mo 10-12	MN 08
CompNetw	Thu 14-16	MN08
:	:	:

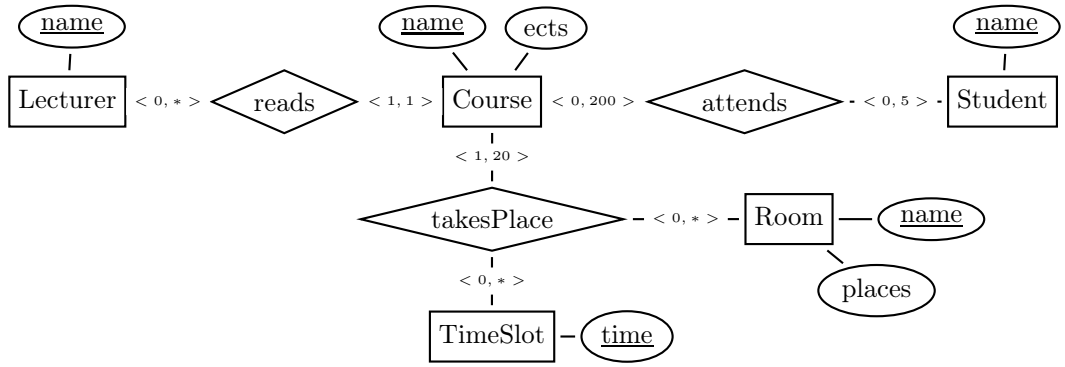
... mit der beachtenswerten Feinheit, dass in *CourseWhenWhere* nur *name* und *time* Schlüssel sind, wenn die Vorlesung nicht parallel in zwei Räumen gehalten werden soll.
Dies zeigt, dass das ER-Modell noch nicht ganz passend ist. Strukturierte Attribute sind auch nur im "erweiterten ER-Modell" enthalten.

Wieder andere Möglichkeit: Jede Vorlesung findet zu mehreren Timeslots statt, und dieses "stattfinden" geschieht jeweils in einem Raum:



Course: (name, ects, ...)
TimeSlot: (time) kann man auch weglassen (kann man aber auch benutzen, um die erlaubten Slots aufzuzählen – sie müssen ja als ForeignKey referenziert werden.
takesPlace: (Course, time, room)

Wenn man *Raum* auch als Entity Type sieht, erhält man wieder einmal eine dreistellige Beziehung:



Course: (name, ects, ...)
Room: (name, places)
TimeSlot: (time)
takesPlace: (Course, time, room)

Obwohl hier in *takesPlace* nur zwei der drei beteiligten Entitätstypen den Schlüssel bilden (ausser eine Vorlesung findet parallel in mehreren Räumen statt ...), kann man die Beziehung nicht in zweistellige Beziehungen aufspalten: das Nichtschlüsselattribut *room* hängt funktional nur von den beiden anderen *gemeinsam* ab; es gibt keine untergeordnete funktionalen Abhängigkeit. Für *takesPlace* wäre auch

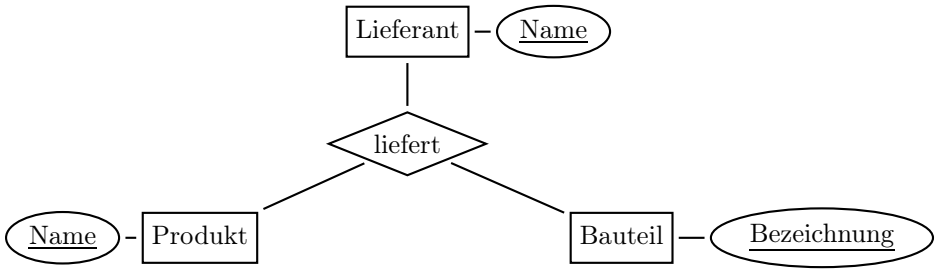
takesPlace: (Course, time, room)

korrekt und sinnvoll, um auszudrücken, dass zu jeder Zeit in einem Raum nur maximal *eine* Vorlesung stattfinden kann.

... man sieht, dass man hier mit mehreren Dozenten/parallelen Vorlesungen/mehreren Terminen in verschiedenen Räumen viele verschiedene Varianten (und ihre Integritätsbedingungen) modellieren kann.

Aufgabe 4 (Dreistellige Beziehungen (Lieferant, Produkt, Bauteil)) Es soll eine Datenbank für eine Marktübersicht über die Hersteller der Komponenten von Autos erstellt werden: Das Bauteil B des Modells M ist von Hersteller H; z.B. "das *Navigationssystem* im *VW Golf* ist von *Bosch*". Es sollen z.B. Anfragen der Art "Alle Modelle, deren Navigationssystem von Bosch ist, und die keine Bauteile von $derGrosseGammelProduzent" enthalten.

a) Geben Sie ein geeignetes ER-Modell an.



Eine kleine Beispieldatenbank:

liefert		
L	P	B
Bosch	Golf	Einspritzpumpe
Bosch	Golf	Navigationssystem
Bosch	Auris	Einspritzpumpe
Sony	Auris	Navigationssystem

Man sieht eigentlich schon dass es keinen untergeordneten Zusammenhang, (formal: keine nicht-vollständige funktionale Abhängigkeit) gibt. Alle drei Attribute bilden gemeinsam den Schlüssel.

b) Läßt sich dieser Sachverhalt mit ausschliesslich binären Beziehungen darstellen?

1) Zerlegung in 2 zweistellige Beziehungen:

"Lieferant L bietet Bauteil B an, und Produkt P benötigt Bauteil B" bedeutet aber nicht notwendigerweise, dass P dieses Bauteil auch von L geliefert bekommt:

Die beiden Tabellen der Beispieldatenbank sind dann

LB	
L	B
Bosch	Einspritzpumpe
Bosch	Navigationssystem
Sony	Navigationssystem

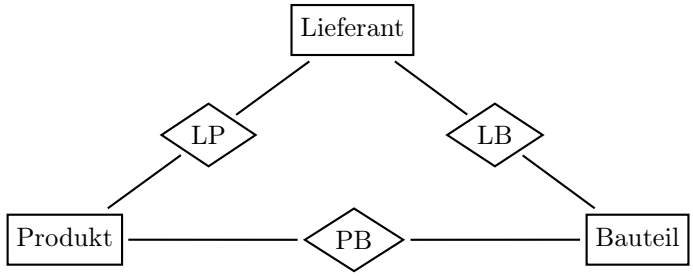
und

PB	
P	B
Golf	Einspritzpumpe
Golf	Navigationssystem
Auris	Einspritzpumpe
Auris	Navigationssystem

Bei Zusammenfassung erhält man auch die Tupel (Bosch, Navigationssystem, Auris) und (Sony, Navigationssystem, Golf), die nicht in der ursprünglichen Beziehung vorhanden waren.
Man hat offensichtlich den "dreistelligen" Zusammenhang "wer liefert wem was?" auseinandergerissen und dabei Informationen verloren ("nicht verlustfreie (=Verlust an Informationen) Zerlegung).

2) Zerlegung in 3 zweistellige Beziehungen:

Man könnte auch folgendes versuchen:



Zerlegung diesmal:

LB	
L	B
Bosch	Einspritzpumpe
Bosch	Navigationssystem
Sony	Navigationssystem

und

PB	
P	B
Golf	Einspritzpumpe
Golf	Navigationssystem
Auris	Einspritzpumpe
Auris	Navigationssystem

und

LP	
L	P
Siemens	Golf
Siemens	Auris
Sony	Auris

Kombiniert man diese Relationen durch ein Join, erhält man auch ein Tupel (Siemens, Auris,

Navigationssystem) das nicht in der ursprünglichen Beziehung vorhanden war.

Auch hier wurde der "dreistellige" Zusammenhang "wer liefert wem was?" auseinandergerissen und kann auch durch die dritte Relation nicht voll wiederhergestellt werden.

c) Betrachten Sie nun dreistellige Beziehungen wieder allgemein. Gibt es Situationen, in denen eine Darstellung durch zwei binäre Beziehungstypen möglich ist? Können diese Situationen exakt durch Kardinalitäten definiert werden?

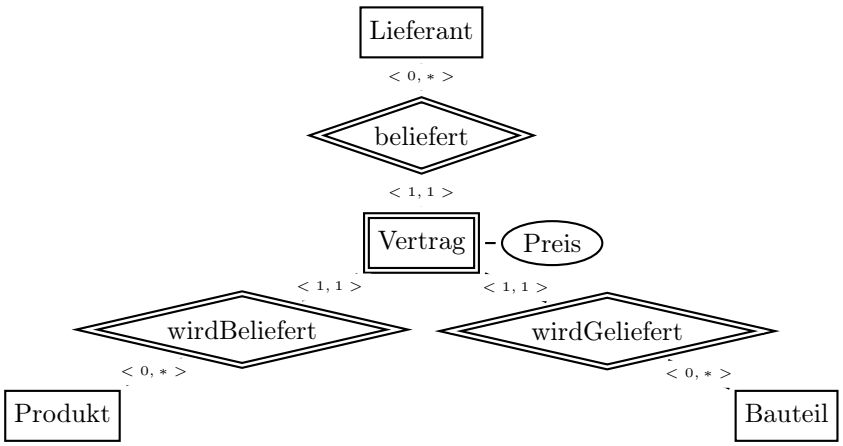
Beispiele siehe Aufgaben 1 und 1.

- wenn eine nicht volle funktionale Abhängigkeit innerhalb der dreistelligen Beziehung besteht.
- Das ist meistens dann der Fall, wenn man beim Versuch, der dreistelligen Beziehung hinreichend strenge Kardinalitäten hinzuzufügen scheitert. Eine Aufspaltung ist dann nicht nur möglich, sondern auch nötig.

d) Kann man dennoch dreistellige Beziehungen generell (unter Verwendung weiterer Hilfskonstrukte) durch zweistellige Beziehungen ersetzen?

Ja: durch "Reifikation" ("Versachlichung", "etwas zu einer Sache machen"). Jede Instanz der dreistelligen Beziehung wird als Entität betrachtet; inhaltlich nichts anderes als das Konzept der "Aggregation" im ER-Modell.

In diesem Beispiel ist das der "Vertrag", zwischen Lieferant und Abnehmer (repräsentiert durch das Zielprodukt) bezüglich dem Bauteil. Ihm wird –um damit weiter modellieren zu können– bei dieser Gelegenheit auch gleich der vereinbarte Lieferpreis hinzugefügt:



Man sieht, dass der reifizierte Entitätstyp zum einen ein *schwacher Entitätstyp* ist und zum anderen in allen Beziehungen eine Kardinalität $< 1, 1 >$ besitzt (jede Instanz repräsentiert ja genau *eine einzige* Instanz des ursprünglichen Beziehungstyps). Man kann (und muss, wegen dem Key) diese Beziehungen im relationalen Modell in den Beziehungstyp reinziehen:

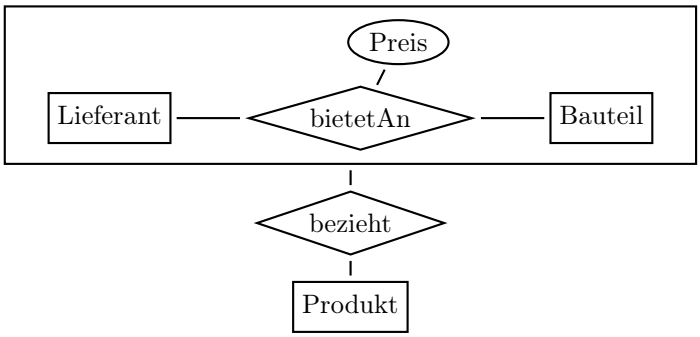
Lieferant: (Name, Adresse)
Bauteil: (Bezeichnung)
Produkt: (Name)
Vertrag: (Lieferant, Bauteil, Produkt, Preis)

Die Tabelle für *Vertrag* ist damit –bis auf die Spalte Preis– dieselbe, wie in Teil (a) für die dreistellige *liefert*-Beziehung (klar, die reifizierten Instanzen von *Vertrag* sind ja gerade die Instanzen des Beziehungstyps *liefert*).

Die obige Modellierung beinhaltet, dass der Preis bei jedem Vertrag einzneln zwischen dem Automobilproduzenten und dem Zulieferer ausgehandelt wird (d.h., der Preis, den VW für die Bosch-Einspritzpumpe bezahlt kann ein anderer sein, als der, den Toyota für dasselbe Bauteil bezahlt).

- (gehört nicht mehr direkt zu d) Eine andere Modellierung erhält man, wenn man eine Aggrega-

tion von Lieferant und Bauteil bildet, und dieser den Preis zuordnet – dann gilt *derselbe* Preis für jeden Abnehmer:



Lieferant: (Name, Adresse)
Bauteil: (Bezeichnung)
Produkt: (Name)
Angebot: (Lieferant, Bauteil, Preis)
bezieht: (Lieferant, Bauteil, Produkt)

---

e) Vergleichen Sie Vor- und Nachteile der verschiedenen Zerlegungen? Lassen sich die verschiedenen Integritätsbedingungen mittels Kardinalitäten ausdrücken?

Vorteil der möglichen Zerlegung in zwei zweistellige Beziehungen:

- Redundanzvermeidung und dadurch weniger Fehlermöglichkeiten im entsprechenden relationalen Modell.
  Eine Zerlegung ist genau dann möglich, wenn eine funktionale Abhängigkeit besteht (vgl. Film-Aufgabe). In diesem Fall sind bei der dreistelligen Beziehung entsprechende Relation (i) nicht alle Fremdschlüsselattribute Keys, (ii) das nicht-key-Attribut ist nicht vollständig funktional vom Key abhängig, sondern (ii) nur von einer Teilmenge des Keys (vgl. Courses-Lecturers-Students).
  In diesem Fall müsste die Relation zerlegt werden (was exakt der Modellierung durch zweistellige Beziehungen entspricht). Dieser Aspekt wird gegen Ende der Vorlesung (Entwurfstheorie, Normalformen) nochmal behandelt.
- In einer dreistelligen Beziehung lassen sich einige Integritätsbedingungen nicht mittels Kardinalitäten darstellen.

Zerlegung mit Hilfs-Entitätstyp: keine Vorteile in der Ausdruckskraft. Bei der Umsetzung in das relationale Modell sieht man auch, dass beide Varianten dasselbe relationale Modell erzeugen.

Man muss sehr genau die Semantik der Anwendung untersuchen, um zu entscheiden, mit welcher Modellierung man sie am genauesten trifft.

Aufgabe 5 (Umsetzung in das relationale Modell: Schlüsselbestimmung von Tabellen für Beziehungen)
In der Vorlesung wurde ein Kochrezept für die Umsetzung eines ER-Modells in ein relationales Modell angegeben. Dabei wurde für die Bestimmung der Schlüssel von Tabellen für Beziehungen auf die Übung verwiesen.

Analysieren Sie, welche Attribute einer solchen Tabelle Schlüssel sind. Beschränken Sie Ihre Betrachtung auf binäre Beziehungen. Welche unterschiedlichen Fälle müssen Sie dabei betrachten?

---

a) **Grundlegende Situation: R hat keine Attribute**:

A hat Schlüsselattribute $AK_1, \ldots, AK_i$, B hat Schlüsselattribute $BK_1, \ldots, BK_j$; beide Relationen können weitere Attribute haben. R hat keine Attribute.

Die Relation R hat damit die Attribute $AK_1, \ldots, AK_i$,$BK_1, \ldots, BK_j$.

(a) $bmax = 1$: Dies ist eine 1-zu-n-Beziehung (ein Land (A) hat n Provinzen (B), die nur zu ihm gehören).

Dann geht jedes b die Beziehung höchstens einmal ein. $BK_1, \ldots, BK_j$ sind Schlüssel der Tabelle $R(\underline{BK_1, \ldots, BK_j}, AK_1, \ldots, AK_i)$, auf die R abgebildet wird (im weiteren wird diese auch einfach als R bezeichnet).

Man kann R somit auch in die Tabelle für B mit aufnehmen (vgl. Province.inCountry zu Province). Ist auch $bmin = 1$, geht jedes b die Beziehung genau einmal ein. Ist $bmin = 0$, so gibt es Tupel, die nicht in einer Beziehung R zu einem a stehen, hier hätten diese Spalten den Wert *null*. Je nachdem, wie hoch der Anteil solcher Tupel ist, lohnt es sich oder auch nicht, die Tabellen zusammenzufassen (z.B. "Mountain located on Island" - jeder Berg liegt auf höchstens einer Insel, man könnte Mountain.locatedOnIsland als Spalte in Mountain aufnehmen, hätte aber viele Nullwerte - also bleibt mountainOnIsland(Mountain,Island) eine separate Tabelle).

(b) $amax = 1$ analog.

(c) $amax = bmax = 1$

Ist $bmax = 1$ und $amax = 1$ (vgl. Country und Capital in Mondial), hat man die Wahl, zu welcher Entity-Tabelle man die Beziehung dazunimmt. Normalerweise nimmt man diejenige, wo weniger Nullwerte auftreten (also Country.Capital und nicht City.isCountryCapitalOf).

Wenn $amin = 1$ ist, kann man sogar R und A in die Tabelle von B mit aufnehmen (vgl. in Mondial Estuary und Source zu River)! (man hätte also auch die Country-Tabelle via capital in City mit aufnehmen können, und mit der jeweiligen eindeutigen Hauptstadt genau einmal ablegen können, ohne "Probleme" zu bekommen – allerdings für den Preis vieler Nullwerte bei allen anderen Städten).

Falls $bmin = 0$, kann diese Tabelle dann b's enthalten, zu denen es keine Werte der von A beibetragenen Spalten enthält. Falls auch $bmin = 1$ hat man zu jedem A genau ein B und umgekehrt.

(d) $amax > 1, bmax > 1$. Dies ist eine n-zu-m-Beziehung. Diese kann man in keine der Tabellen mit hineinnehmen (weil man mehrere "Partner" ablegen muss). Alle AK und BK sind damit Schlüsselattribute von $R(\underline{BK_1, \ldots, BK_j, AK_1, \ldots, AK_i})$.

(Strenggenommen muss man noch fordern, dass es keine Mehrfachbeziehungen (a, b) gibt, deren Anzahl relevant ist.)

b) R **hat auch Attribute** $R_1, \ldots, R_k$:

(a) R hat nur skalare (d.h., nicht mengenwertige) Attribute (z.B. encompassed: country, continent, percent), und jedes a geht jede Beziehung mit einem bestimmten b maximal einmal ein.

Dann gilt dasselbe wie in Fall (1):

i) $bmax = 1$: $R(\underline{BK_1, \ldots, BK_j}, AK_1, \ldots, AK_i, R_1, \ldots, R_k)$.

ii) $amax = 1$: $R(\underline{BK_1, \ldots, BK_j}, \underline{AK_1, \ldots, AK_i}, R_1, \ldots, R_k)$.

i) $amax > 1$ und $bmax > 1$: $R(\underline{BK_1, \ldots, BK_j}, \underline{AK_1, \ldots, AK_i}, R_1, \ldots, R_k)$.

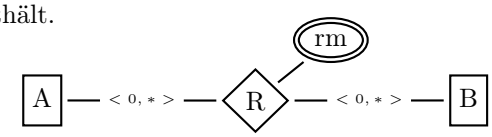
(b) R hat nur skalare (d.h., nicht mengenwertige) Attribute, aber jedes a kann jede Beziehung mit einem bestimmten b mehrmals eingehen. Dann muss man den Fall anwendungsabhängig tiefergehend analysieren (Theorie: funktionale Abhängigkeiten und nachfolgende Zerlegung), z.B.

- geschichtliche Daten zu Mitgliedschaften von Ländern in Organisationen, z.B. $(o, c, \mathsf{candidate}, von_1, bis_1)$ und $(o, c, \mathsf{member}, von_2, bis_2)$: hier wählt man am besten ismember(ccode, oabbrev, type, von, bis).
- historische Daten zu Mitgliedschaften von Personen in Gremien $(o, c, \text{"Mitglied"/"Vertreter"/...}, von, bis)$, wobei jeder Mitgliedschaftstyp in beliebigen Amts-

zeiten der Fall sein kann: dann wäre ismember(person, gremium, von, rolle, bis) wahrscheinlich die richtige Wahl.

- Telefongesprächsdaten: telefonierte_mit(A, B) mit skalaren Attributen von, bis. Dann wäre telefonierte_mit(anrufer, angerufener, von, bis) ein sinnvolles Schema (unter der realistischen Annahme, dass jeder nur ein Gespräch gleichzeitig führen kann, würde auch telefonierte_mit(anrufer, angerufener, von, bis) genügen – man sieht hier, dass durch die Attribute ein urpsrünglich für die Beziehung notwendiger Schlüssel auch wegfallen kann.

(c) R hat auch ein mengenwertiges Attribut RM: RM muss (wie bei mehrwertigen Attributen von Entitätstypen) rausgezogen werden in eine Tabelle, die alle Keys von R, und ausserdem RM als zusätzlichen Schlüssel enthält.
Beispiel:



Die Beziehung selbst wird durch eine Relation $R(\underline{AK_1, \ldots, AK_i, BK_1, \ldots, BK_j})$ repräsentiert (gegebenenfalls noch mit den funktionalen Attributen von R). Ausserdem hat man eine Relation $RRM(\underline{AK_1, \ldots, AK_i, BK_1, \ldots, BK_j, RM})$, in der zu jedem (A, B)-Paar alle RM-Werte abgelegt sind.

(d) R hat mehrere mengenwertige Attribute. In einer Anwendung in Social Web z.B. eine n-m-Beziehung telefonierte_mit(A, B) mit skalaren Attributen von, bis und mengenwertigen Attributen RM_1 = bespricht_Thema und RM_2 = lästert_über.
Dann benötigt man jeweils Tabellen $R_1(\underline{AK_1, \ldots, AK_i, BK_1, \ldots, BK_j, \text{von}, \text{bis}})$,
$R_2(\underline{AK_1, \ldots, AK_i, BK_1, \ldots, BK_j, \text{von}}, \text{bespricht\_Thema})$, und
$R_3(\underline{AK_1, \ldots, AK_i, BK_1, \ldots, BK_j, \text{von}}, \text{lästert\_über})$
(da das jeweilige Sachthema nicht an eine/mehrere belästerte Personen gebunden ist).
Dies führt auch schon unmittelbar zu 3-stelligen Beziehungen (die 3. Person als Lästerobjekt), in denen man auch die *funktionalen Abhängigkeiten* untersuchen muss, um festzustellen, ob sie in zwei Tabellen aufgespalten werden müsste.

Man kann diese Überlegungen immer auch direkt am ER-Diagramm durchführen, indem man die Beziehung als Entitätstyp aggregiert/reifiziert (z.B. einen schwachen Entitätstyp "Telefongespräch"), und dessen identifizierende Beziehungen (zum Anrufer und zum Angerufenen) und lokale Keys bestimmt.

Aufgabe 6 (Umsetzung in das relationale Modell: Mondial) Betrachten Sie die Umsetzung aller Entitäts- und Beziehungstypen in das relationale Modell von Mondial.

a) Entitätstypen,

b) Beziehungstypen,

c) ... und zur Kontrolle nochmal rückwärts: alle Relationen von Mondial.
Diese bekommen Sie z.B. mit der Anfrage

```
SELECT table_name FROM tabs;
```

Welche der Umsetzungen sind "einfach" dem Kochrezept entsprechend, welche enthalten Ausnahmen? Wie sind diese begründet?

Entitätstypen

- Continent: nach Kochrezept.
- Language, EthnicGrp, Religion: nach Kochrezept hätten die jeweiligen Relationen nur ein Attribut "Name". Man kann sie weglassen; alle Informationen sind in den Beziehungstabellen(!) "Language" (von "speak"), "EthnicGrp" (von "belong"), "Religion" (von "believe") enthalten.

- Organization: die 1:1-Beziehung "has_headquarter" wurde direkt mit hineingenommen (auf "City" mit dreistelligem Schlüssel (name, country, province)).
- Country: die 1:1-Beziehung "has_capital" wurde direkt mit hineingenommen (auf "City" mit dreistelligem Schlüssel (name, country, province)).
  Ausserdem wurden Daten in die Tabellen "politics", "economy", "population" (jeweils auch mit Schlüssel code→country) ausgelagert (dies wird als "vertikale Partitionierung") bezeichnet).
- City: weak entity type (name, country, province) nach Kochrezept.
- Province: weak entity type (name, country) nach Kochrezept.
  Ausserdem wurde die 1:1-Beziehung "has_capital" direkt mit hineingenommen (auf "City" mit dreistelligem Schlüssel (city(name), country, (province)name), wobei ((province)name, country) ja sowieso schon als Keys vorhanden sind.

- Lake: hier wurde zusätzlich die 1:1-Beziehung "to" als "river" mit aufgenommen.
- River: hier wurden zusätzlich die 1:1-Beziehungen "to" (zu einem Meer, See, oder einem anderen Fluss) und "has (Source)" und "(has) Estuary" mit aufgenommen (falls ein Fluss durch mehrere Seen fließt, ist nur der erste angegeben – hier ist die Modellierung nicht optimal gewählt).
  Auch die Entitätstypen "Source" und "Estuary" wurden mit aufgenommen, da *jede* Quelle und Mündung ja zu einem Fluss gehört, und somit alle Entitäten dieser Klasse erfasst werden.

- Sea: nach Kochrezept.
- Island: nach Kochrezept.
- Mountain: nach Kochrezept, wobei die funktionale Beziehung "mountainonisland" in eine separate Tabelle ausgelagert ist (auch hier wieder "vertikale Partitionierung", da diese Spalte sonst viele nullwerte enthalten würde).
- Desert: nach Kochrezept.
- Source, Estuary: beide in "River" einbezogen.

- Airport: nach Kochrezept (Beziehungen zu City, und damit auch zu Country mit einbezogen).

Beziehungstypen

- speak, belong, believe: s.o. unter "Country"
- encompasses: typische n:m-Beziehung mit einem Attribut nach Kochrezept in Tabelle "encompasses" umgesetzt.
- is_member: typische n:m-Beziehung mit einem Attribut nach Kochrezept in Tabelle "ismember" umgesetzt.
- dependent: 1:1-Beziehung in "politics" umgesetzt (und damit quasi in die Modellierung von "Country" integriert).
- wasdependent: 1:1-Beziehung in "politics" umgesetzt (und damit quasi in die Modellierung von "Country" integriert). Da einige Länder zuletzt von etwas nicht mehr existierendem (Sowjetunion, Jugoslawien, Osmanisches Reich) abhängig waren, ist dies kein Fremdschlüssel auf Country.code!
- has_headq_in: s.o. als 1:1-Beziehung in "Organization".
- is_capital: s.o. als 1:1-Beziehung in "Country" bzw. "Province".
- (Province) of (Country): als n:1-Beziehung in "Province" (wo es sowieso schon dabei ist, weil Province ein weak entity type ist).
- (City) in (Province): als n:1-Beziehung in "City" (wo es sowieso schon dabei ist, weil City ein

weak entity type ist).

- borders: rekursive (d.h. zwischen Country und Country) symmetrische n:m-Beziehung mit Attributen nach Kochrezept in Tabelle "borders" umgesetzt.
  Eine Besonderheit hier ist die Tatsache, dass die Relation symmetrisch ist. Jede Nachbarschaftsbeziehung wird nur einmal gespeichert (d.h., nur CH-D, nicht auch noch D-CH).
- (City) at (Lake/River/Sea): im Prinzip nach Kochrezept, alle in "located" gesammelt.
  Durch das Sammeln hat man allerdings das Problem, dass man keinen Schlüssel definieren kann, weil bei jedem Eintrag ein oder mehrere Gewässer-Einträge null sein können, und eine Stadt z.B. an mehreren Flüssen liegen kann ($\rightarrow$ mehrere Tupel für dieselbe Stadt).
- (City) on (Island): nach Kochrezept in "locatedon".
- ("geo-Objects" Lake/River/Sea/Island/Mountain/Desert/Source/Estuary) in (Province): n:m-Beziehungen, jede einzeln nach Kochrezept in "geo_lake", "geo_River", "geo_Sea", "geo_Island", "geo_Mountain", "geo_Desert" umgesetzt.
- merges (zwischen Meeren): rekursive symmetrische n:m-Beziehung ohne Attribute nach Kochrezept in Tabelle "mergeswith" umgesetzt.
  Auc hier ist die Relation symmetrisch, jede Nachbarschaftsbeziehung wird nur einmal gespeichert.
- (Island) in (Sea/Lake/River): n:m-Beziehung nach Kochrezept in "islandin" umgesetzt und gesammelt (wie "located").
- (Mountain) on (Island): n:1-Beziehung in "mountainonisland" umgesetzt. Hier hätte man sie auch als "island"-Attribut in "Mountain" mit aufnehmen können. Diese Spalte wäre aber bei vielen Bergen dann *null* gewesen.

- (River) has (Estuary) to (Gewässer): da zu jedem Fluss (maximal; manche versickern) eine (finale) Mündung gehört ("Zwischenmündungen" wenn ein Fluss in einem See fließt, aus dem er auch wieder hinausfließt, sind nicht hier abgelegt), wurden diese Beziehungen, sowie der Entitätstyp "Estuary" in die Relation "River" mit aufgenommen.
- (River) has (Source): wie bei "has (Estuary)".

Bemerkungen

Das Attribut "Mountains" ("Gebirge") tritt im ER-Modell mehrmals auf: in "Mountain" sowie in "Source". Man hätte einen separaten Entitätstyp "Mountains" mit Attributen "Name" und vielleicht "elevation" und "area" modellieren können, und dann "(mountains) in (Province)" wie für alle Geo-Objekte, auch "(City) in (Mountains)" und "(Mountains) on (Island)" aufnehmen können.

Mondial-Relationen

... alle oben beschrieben.