Praktikum "Entwicklung einer Datenbankanwendung" Universitäts-Informationssystem

Als Ausgangspunkt dient die untenstehende Aufgabe aus der Vorlesung "Einführung in Datenbanken":

Aufgabe 1 (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.

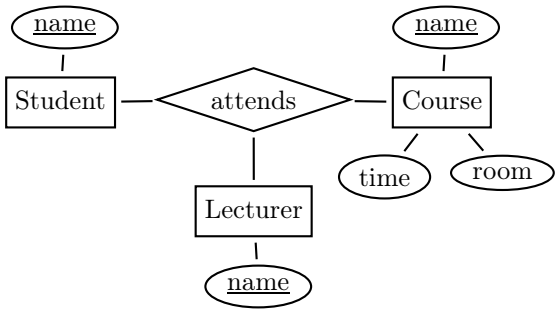
b) Vorlesungen können auch von mehreren Dozenten gemeinsam gehalten werden; z.B. *Informatik I* von *Müller* von Oktober bis Weihnachten, und von *Meier* den Rest bis zum Semesterende.

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.

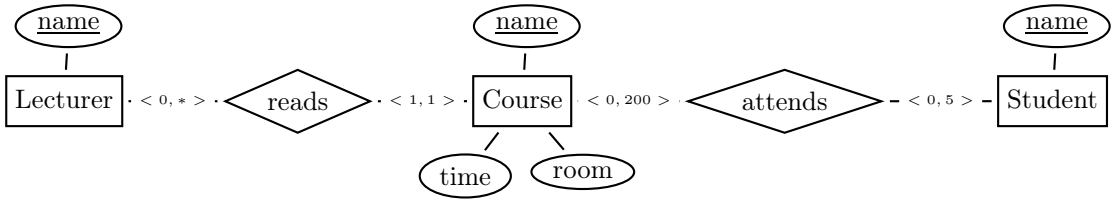
Das Schema dieser dreistelligen Relation ist

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* → *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: (name, address)
Course: (name, ects, ...)
Student: (name, address, ...)
reads: (Course, Lecturer)
attends: (Student, Course)

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: (name, address)
Course: (name, Lecturer, ects, ...)
Student: (name, address, ...)
attends: (Student, Course)

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: (name, address)
Course: (name, ects, ...)
Student: (name, address, ...)
reads: (Course, Lecturer)
attends: (Student, Course)

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

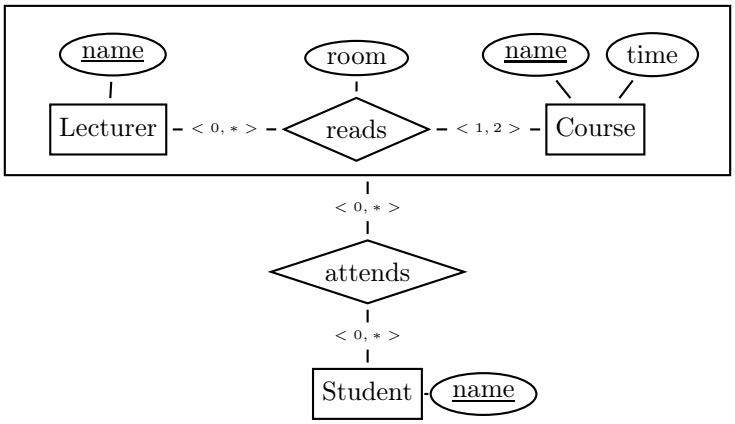
Fremdschlüssel-Integritätsbedingungen:

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

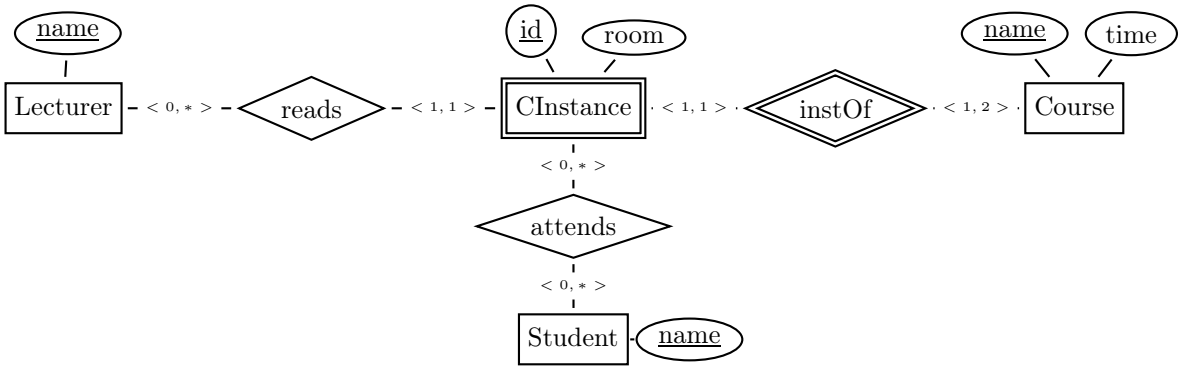
c) Die obige Modellierung funktioniert nicht mehr, wenn große Vorlesungen parallel bei zwei Dozenten (selbe Zeit, unterschiedliche Räume) angeboten werden *Informatik I* Dienstags 14-16 von *Schmidt* in HS1 und parallel von *Schulze* in HS3.

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:

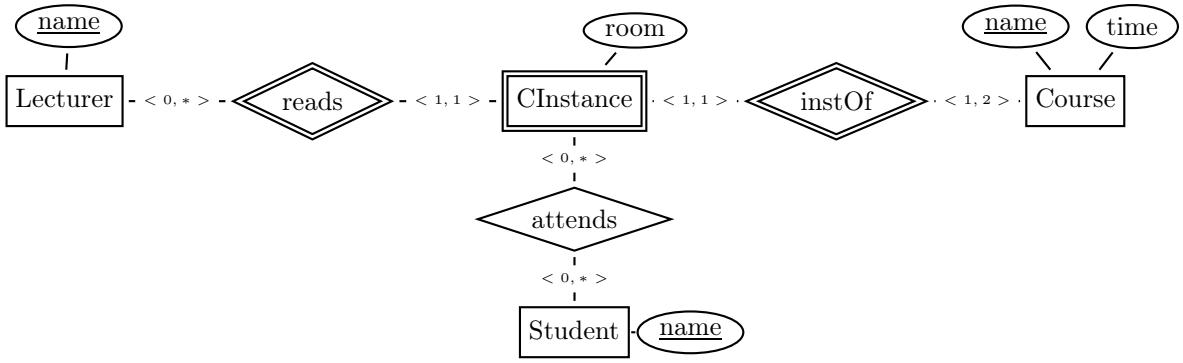
Lecturer: (name, address)
Course: (name, ects, ...)
Reads=CourseInstance: (course, Lecturer)
Student: (name, address, ...)
attends: (Student, Course, optional: id, wenn man eine feste Zuordnung haben will)
(*reads* wird wegen der $< 1, 1 >$-Kardinalität nach *CourseInstance* reingezogen)

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

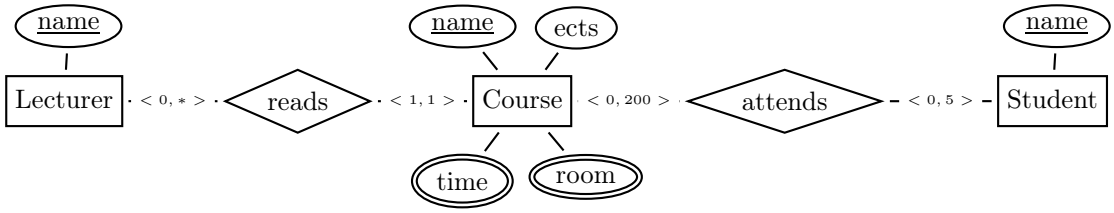
als *weak entity type* gemacht hätte, und somit die Keys *Course.Name* und *Lecturer.Name* geerbt hätte:



Lecturer: (name, address)
Course: (name, ects, ...)
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)

- Bei attends: (Student, Course, id 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: (Student, Course) 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:



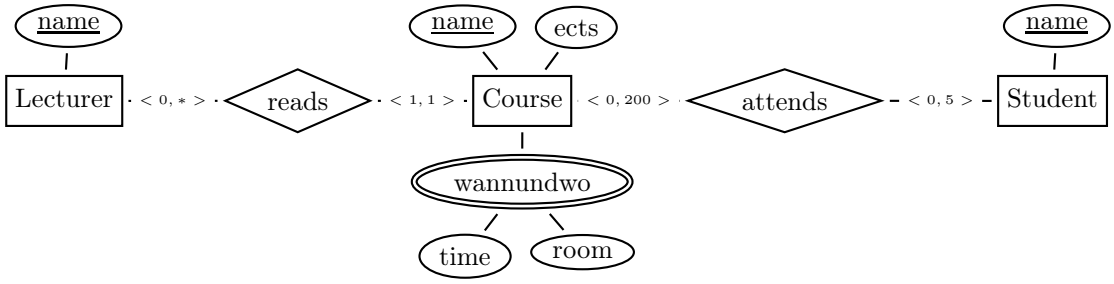
Bei der Umsetzung ins relationale Modell müssen die beiden mehrwertigen Attribute getrennt behandelt werden:

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

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

CourseRoom	
name	**room**
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:

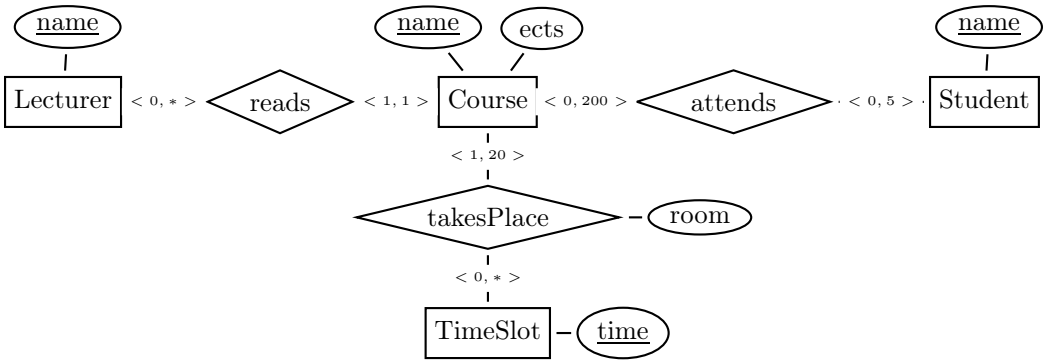


Course	
name	**ects**
Databases	5
CompNetw	5
:	:

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
:	:	:

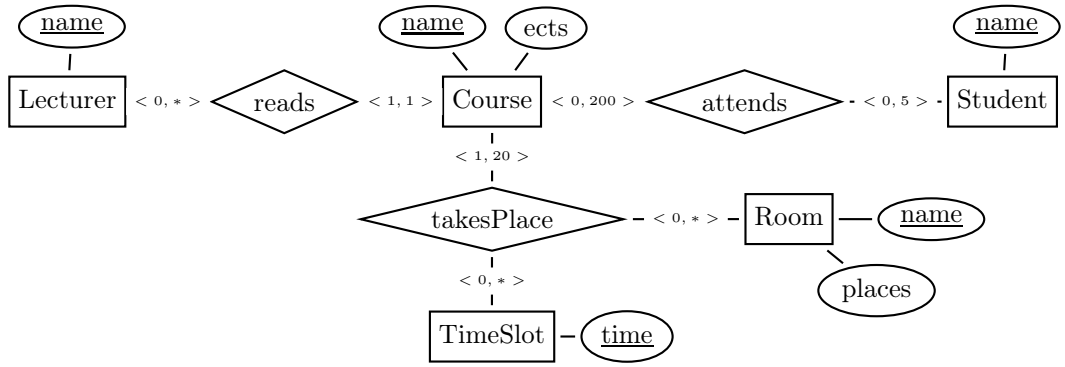
... mit der beachtenswerten Feinheit, dass in *CourseWhenWhere* nur *name* und *time* Schlüssel sind, wenn die Vorlesung nicht parallel in zwei Räumen gehalten werden soll.
Dies zeigt, dass das ER-Modell noch nicht ganz passend ist. Strukturierte Attribute sind auch nur im "erweiterten ER-Modell" enthalten.

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.