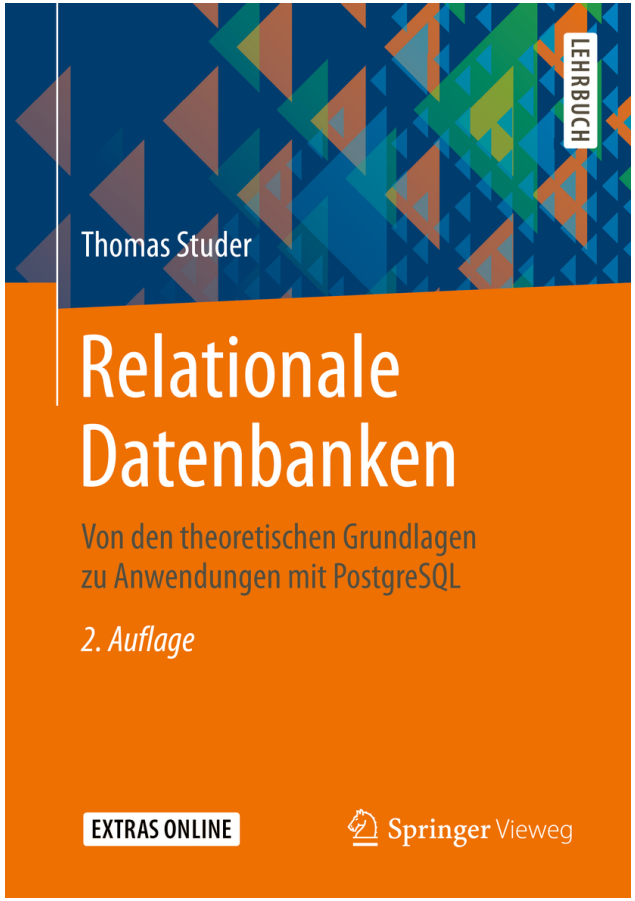


Thomas Studer
Relationale Datenbanken:
Von den theoretischen Grundlagen
zu Anwendungen mit PostgreSQL
2. Auflage
Springer Vieweg, 2019
ISBN 978-3-662-58975-5

Datenbanken

—

Diagramme und Modellierung

Thomas Studer

Institut für Informatik
Universität Bern

Datenmodellierung

Es geht darum, ein DB-Schema zu finden, so dass

1. alle benötigten Daten im DB-Schema abgespeichert werden können,
2. effizient auf die Daten zugegriffen werden kann und
3. die Datenkonsistenz gewährleistet ist.

Einfache Tabelle

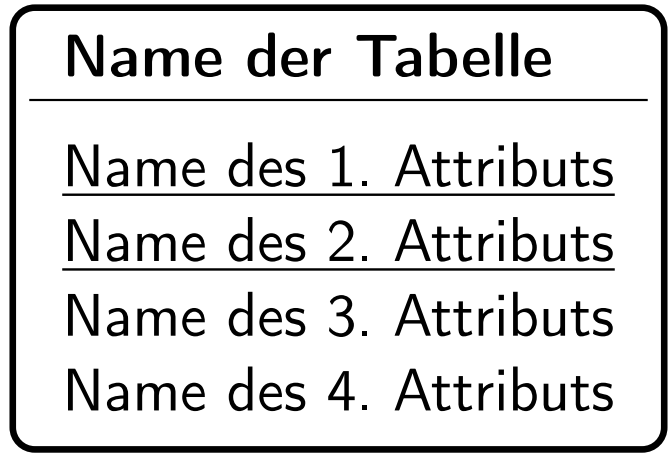


Tabelle mit vier Attributen, wobei die ersten beiden den Primärschlüssel bilden

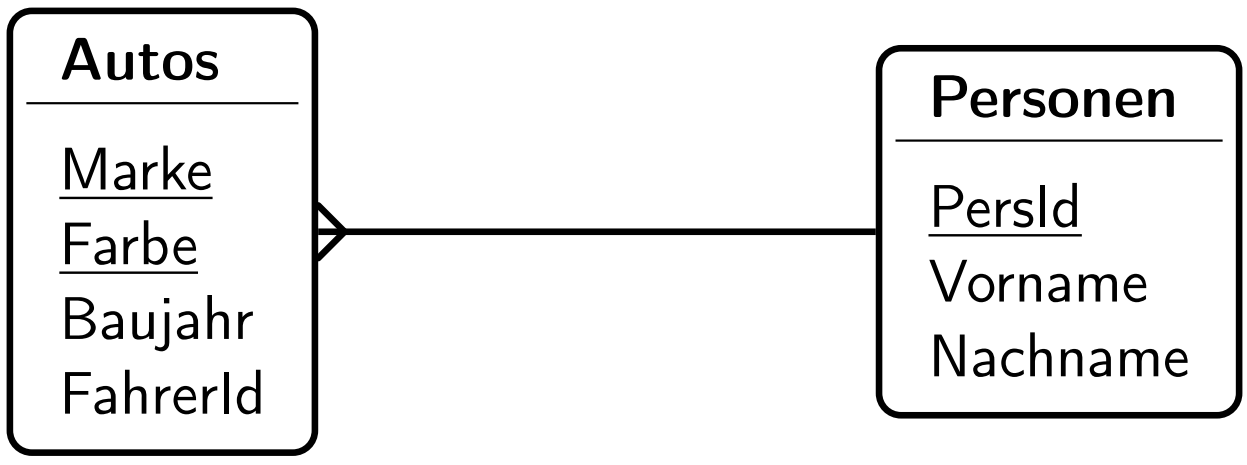
Vereinfachung

Name der Tabelle

Konkret


Autos
Marke
Farbe
Baujahr
FahrerId
Personen
PersId
Vorname
Nachname

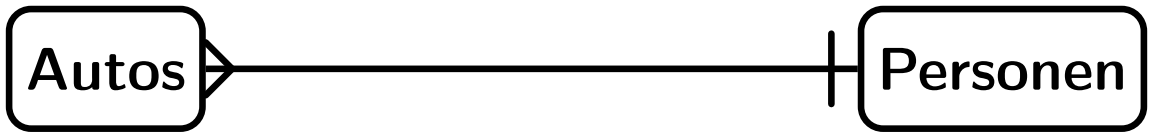
Fremdschlüssel-Beziehung



Eine solche Beziehung heisst *m:1-Beziehung*.

Existenzbedingung

Mit Hilfe eines *not null Constraints* auf dem Fremdschlüsselattribut `FahrerId` können wir verlangen, dass es zu jedem Auto *mindestens* einen Fahrer geben muss.

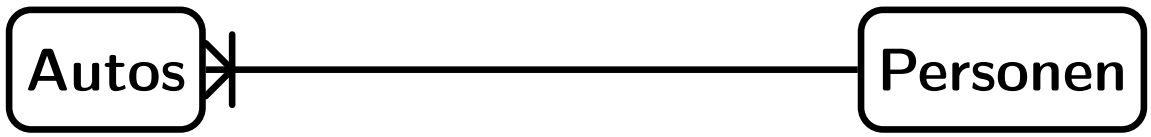


Diese Darstellung drückt zwei Sachverhalte aus:

1. Jedes Auto hat genau einen Fahrer, das heisst mindestens einen und auch höchstens einen Fahrer.
2. Jede Person kann kein, ein oder mehrere Autos fahren.

Existenzbedingung auf der anderen Seite

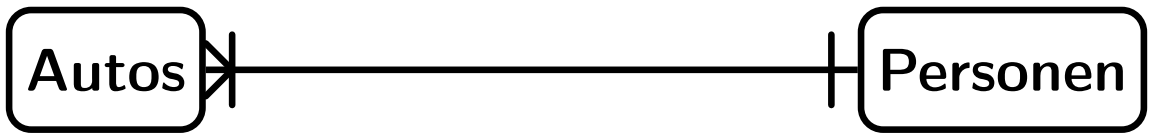
1. Jedes Auto hat keinen oder einen Fahrer.
2. Jede Person fährt ein oder mehrere Autos.



Diese Existenzbedingung können wir im relationalen Modell *nicht* durch einen not null Constraint auf einem Attribut ausdrücken.

Existenzbedingung auf beiden Seiten

1. Jedes Auto hat genau einen Fahrer.
2. Jede Person kann ein oder mehrere Autos fahren.



Zusammenfassung

Beschreibung	**Symbol**
Keine oder eine Beziehung	
Keine, eine oder mehrere Beziehungen	
Genau eine Beziehung	
Eine oder mehrere Beziehungen	

m:n-Beziehung

Betrachten wir ein DB-Schema für eine Bank, welche Kunden und ihre Konten verwalten muss. Dabei soll folgendes gelten:

1. Ein Kunde kann mehrere Konten haben.
2. Ein Konto kann mehreren Kunden gemeinsam gehören.

DB-Schema:

$$\mathcal{S}_{\text{Kunden}} := (\underline{\texttt{KundenNr}}, \texttt{Name})$$
$$\mathcal{S}_{\text{Konten}} := (\underline{\texttt{KontoNr}}, \texttt{Stand})$$
$$\mathcal{S}_{\text{KuKo}} := (\underline{\texttt{KundenNr}}, \underline{\texttt{KontoNr}})$$

Tabellen für m:n-Beziehungen

Kunden

KundenNr	Name
A	Ann
B	Tom
C	Eva
D	Bob

KuKo

KundenNr	KontoNr
A	1
A	2
B	2
C	3
D	3

Konten

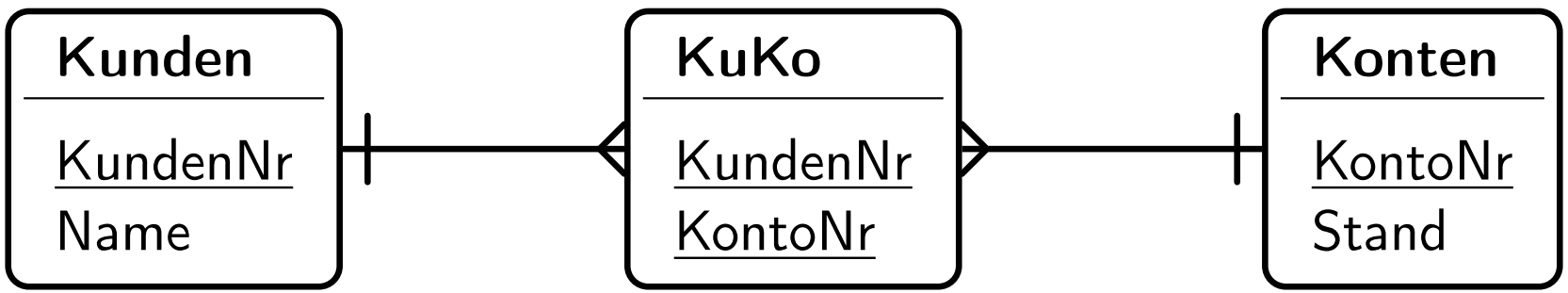
KontoNr	Stand
1	1000
2	5000
3	10

Diagramm

Zu jedem Eintrag in der `KuKo`-Tabelle müssen die entsprechenden Kunden und Konten existieren. Das heisst,

1. das Attribut `KundenNr` in `KuKo` ist ein Fremdschlüssel auf `Kunden`,
2. das Attribut `KontoNr` in `KuKo` ist ein Fremdschlüssel auf `Konten`.

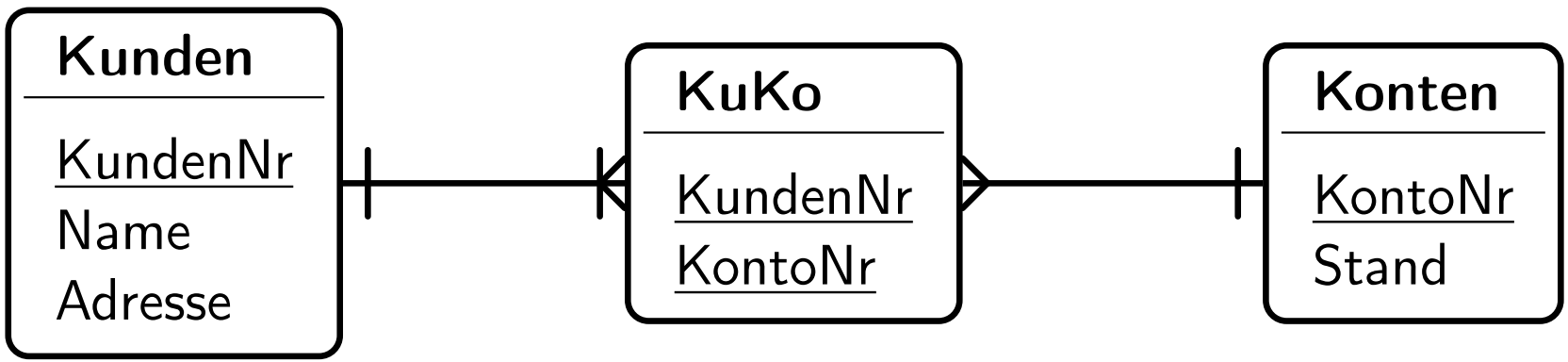
Diese Fremdschlüssel müssen einen not null Constraint erfüllen, da sie Teil des Primärschlüssels des `KuKo` Schemas sind.



Existenzbedingung

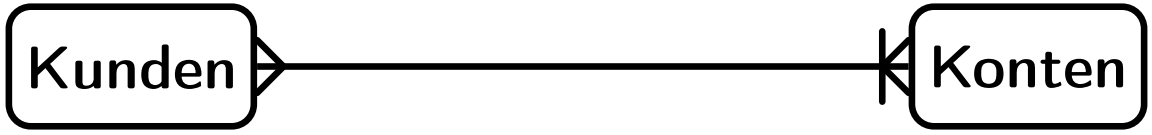
1. Hat jeder Kunde ein Konto?
2. Muss jedes Konto einem Kunden gehören?

Wir beantworten die erste Frage mit *ja*. Wer kein Konto hat, kann auch kein Kunde sein. Die zweite Frage verneinen wir. Unsere Bank ist etwas altmodisch und lässt nachrichtenlose Vermögen (d.h. Konten ohne bekannte Kundenbeziehung) zu.



Kurzform

In diesem Beispiel enthält die Tabelle `KuKo` keine weiteren Daten ausser den Primärschlüsseln für `Kunden` und `Konten`. Im Prinzip können wir somit die Box für dieses Schema bei der Diagrammdarstellung des DB-Schemas weglassen.



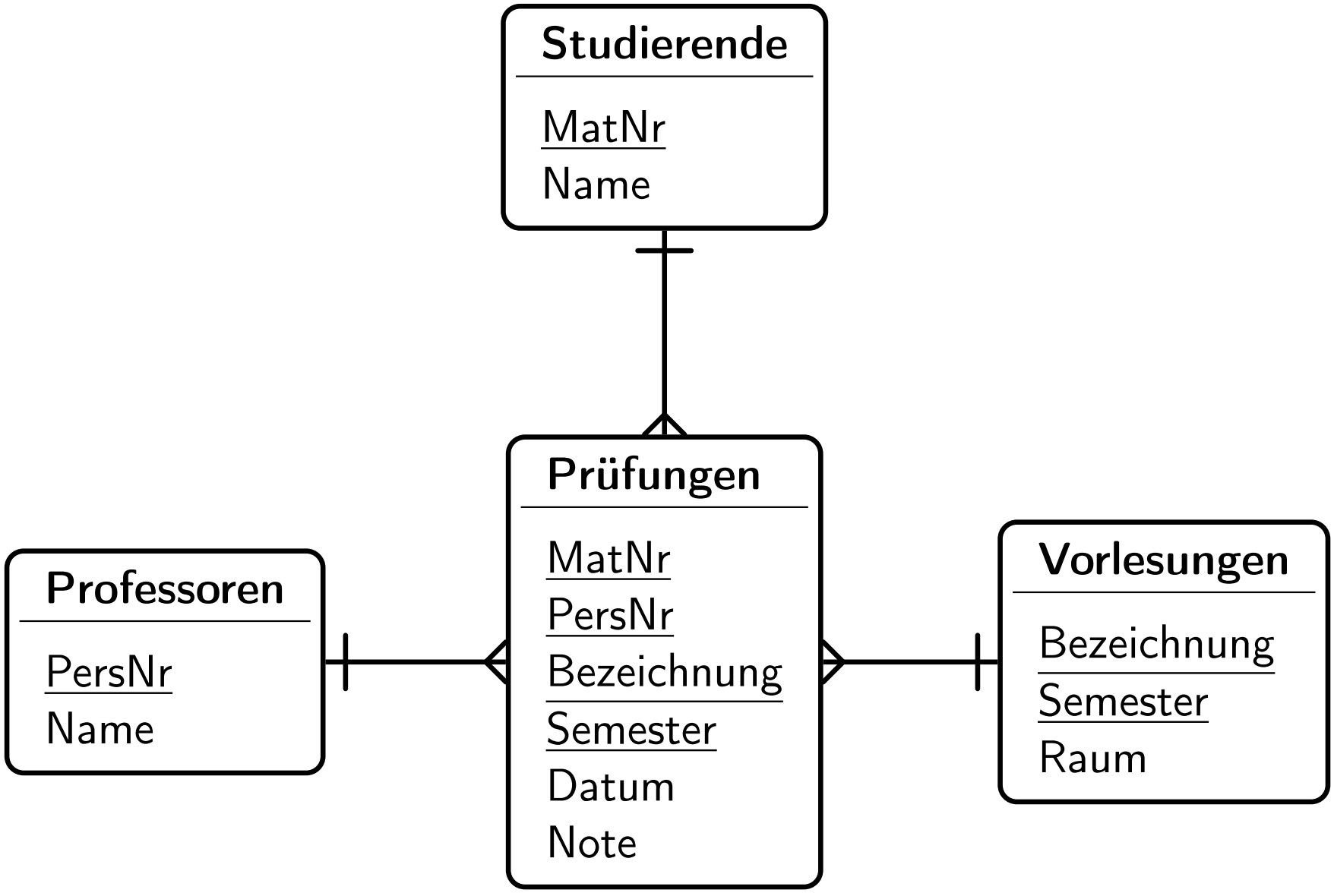
Ternäre Beziehungen

DB-Schema für Universität:

$$\mathcal{S}_{\text{Studierende}} := (\underline{\texttt{MatNr}}, \texttt{Name})$$
$$\mathcal{S}_{\text{Professoren}} := (\underline{\texttt{PersNr}}, \texttt{Name})$$
$$\mathcal{S}_{\text{Vorlesungen}} := (\underline{\texttt{Bezeichnung}}, \underline{\texttt{Semester}}, \texttt{Raum})$$
$$\mathcal{S}_{\text{Prüfungen}} := (\underline{\texttt{MatNr}}, \underline{\texttt{PersNr}}, \underline{\texttt{Bezeichnung}}, \underline{\texttt{Semester}}, \texttt{Datum}, \texttt{Note})$$

1. Beziehungen können zwischen mehr als zwei Konzepten bestehen.
2. Wenn ein Primärschlüssel aus mehreren Attributen besteht, so muss der ganze Primärschlüssel in der Beziehungstabelle vorkommen. Im Beispiel enthält $\mathcal{S}_{\text{Prüfungen}}$ die Attribute `Bezeichnung` und `Semester`, um eine Vorlesung zu identifizieren.
3. Beziehungen können zusätzliche Attribute haben.

Diagramm


Studierende
MatNr
Name
Prüfungen
MatNr
PersNr
Bezeichnung
Semester
Datum
Note
Professoren
PersNr
Name
Vorlesungen
Bezeichnung
Semester
Raum

1:1-Beziehungen, z.B. Bundesräte und Departemente

$$\mathcal{S}_{\text{Regierung}} := (\underline{\texttt{BrId}}, \texttt{DepId})$$
$$\mathcal{S}_{\text{Departemente}} := (\underline{\texttt{DepId}}, \texttt{BrId}) \ .$$

wobei `DepId` ein Fremdschlüssel auf $\mathcal{S}_{\text{Departemente}}$ und `BrId` ein Fremdschlüssel auf $\mathcal{S}_{\text{Regierung}}$ ist.

`Regierung`

BrId	DepId
A	Y
B	Y
C	Z

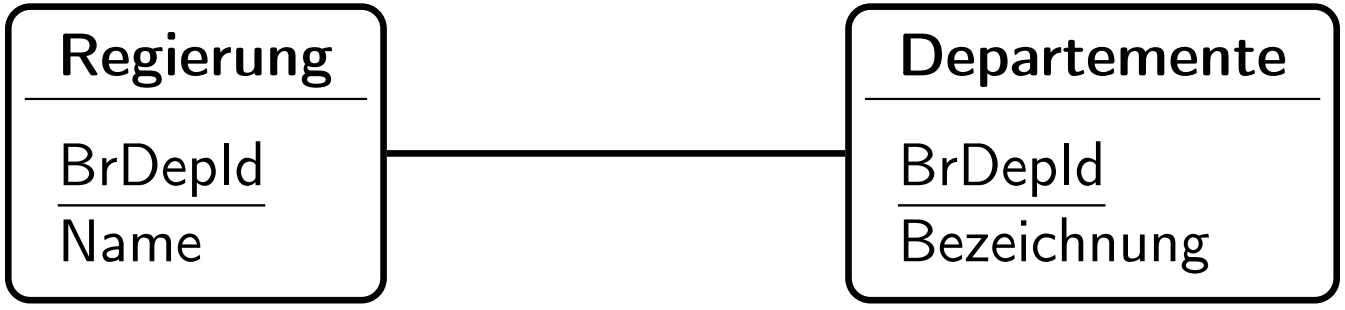
`Departemente`

DepId	BrId
Y	C
Z	A

Korrekte Modellierung

$$\mathcal{S}_{\text{Regierung}} := (\underline{\texttt{BrDepId}}, \texttt{Name})$$
$$\mathcal{S}_{\text{Departemente}} := (\underline{\texttt{BrDepId}}, \texttt{Bezeichnung}) \ .$$

Die graphische Darstellung dieses DB-Schemas ist nun



Instanz davon

Regierung

BrDepId	Name
1	Berset
2	Maurer

Departemente

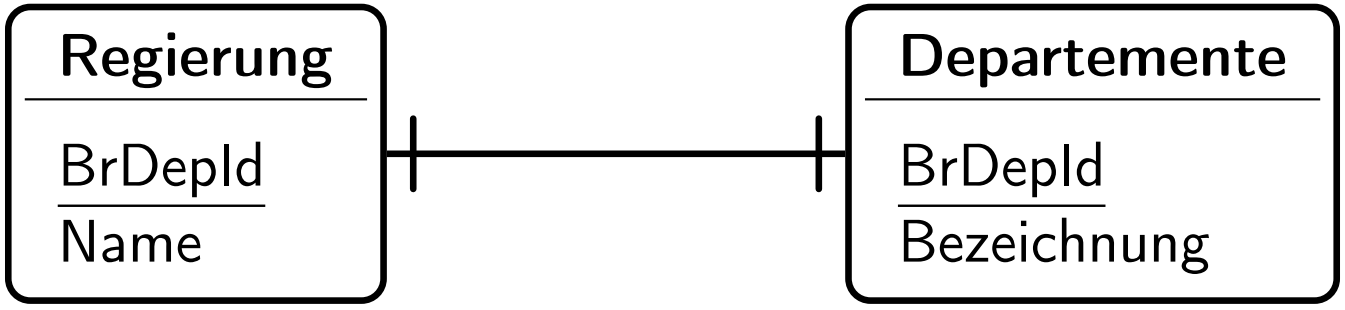
BrDepId	Bezeichnung
1	EDI
3	EJPD

Berset steht also dem EDI vor. Für Mauer gibt es jedoch keinen Eintrag in der Departemente-Tabelle, und auch wer das EJPD führt, ist in dieser Instanz nicht ersichtlich.

Um diese Situation zu verhindern, können wir im Diagramm noch Existenzbedingungen hinzufügen.

Mit Existenzbedingungen

1. Der Primärschlüssel `BrDepId` im Schema `Regierung` ist gleichzeitig ein Fremdschlüssel auf $\mathcal{S}_{\text{Departemente}}$.
2. Der Primärschlüssel `BrDepId` in `Akten` ist gleichzeitig ein Fremdschlüssel auf $\mathcal{S}_{\text{Angestellte}}$.



Vererbung

Schema für Personen

$$\mathcal{S}_{\text{Personen}} := (\underline{\texttt{PersId}}, \texttt{Name}) \ .$$

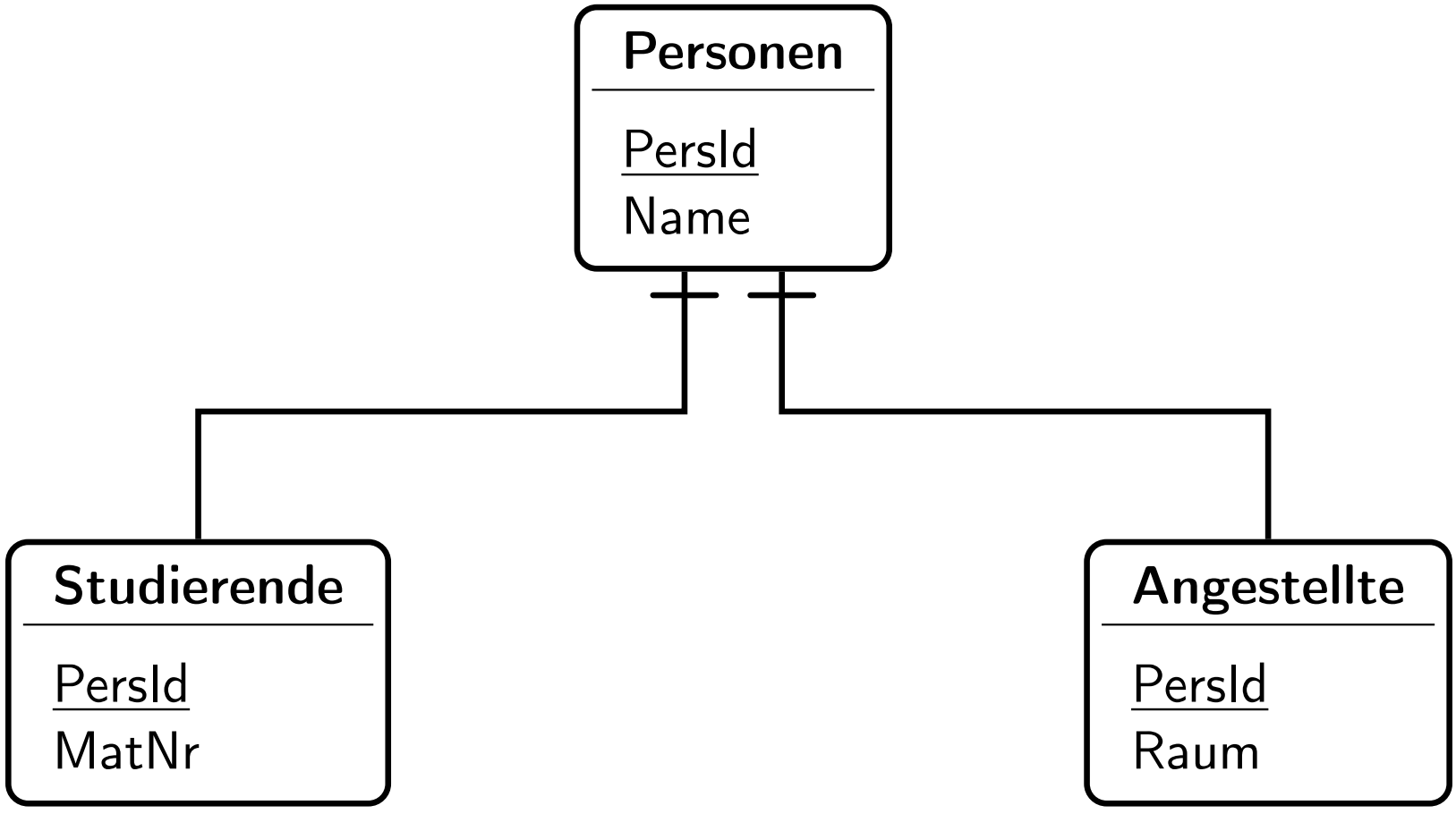
Spezialisierung zu *Angestellte* und *Studierende*

$$\mathcal{S}_{\text{Studierende}} := (\underline{\texttt{PersId}}, \texttt{MatNr})$$
$$\mathcal{S}_{\text{Angestellte}} := (\underline{\texttt{PersId}}, \texttt{Raum}) \ .$$

In beiden Schemata ist der Primärschlüssel `PersId` gleichzeitig Fremdschlüssel auf $\mathcal{S}_{\text{Personen}}$.

Die Relation `Person` heisst *Basisrelation*.
Die Relationen `Studierende` und `Angestellte` heissen *abgeleitete Relationen*.

Vererbung: Diagramm

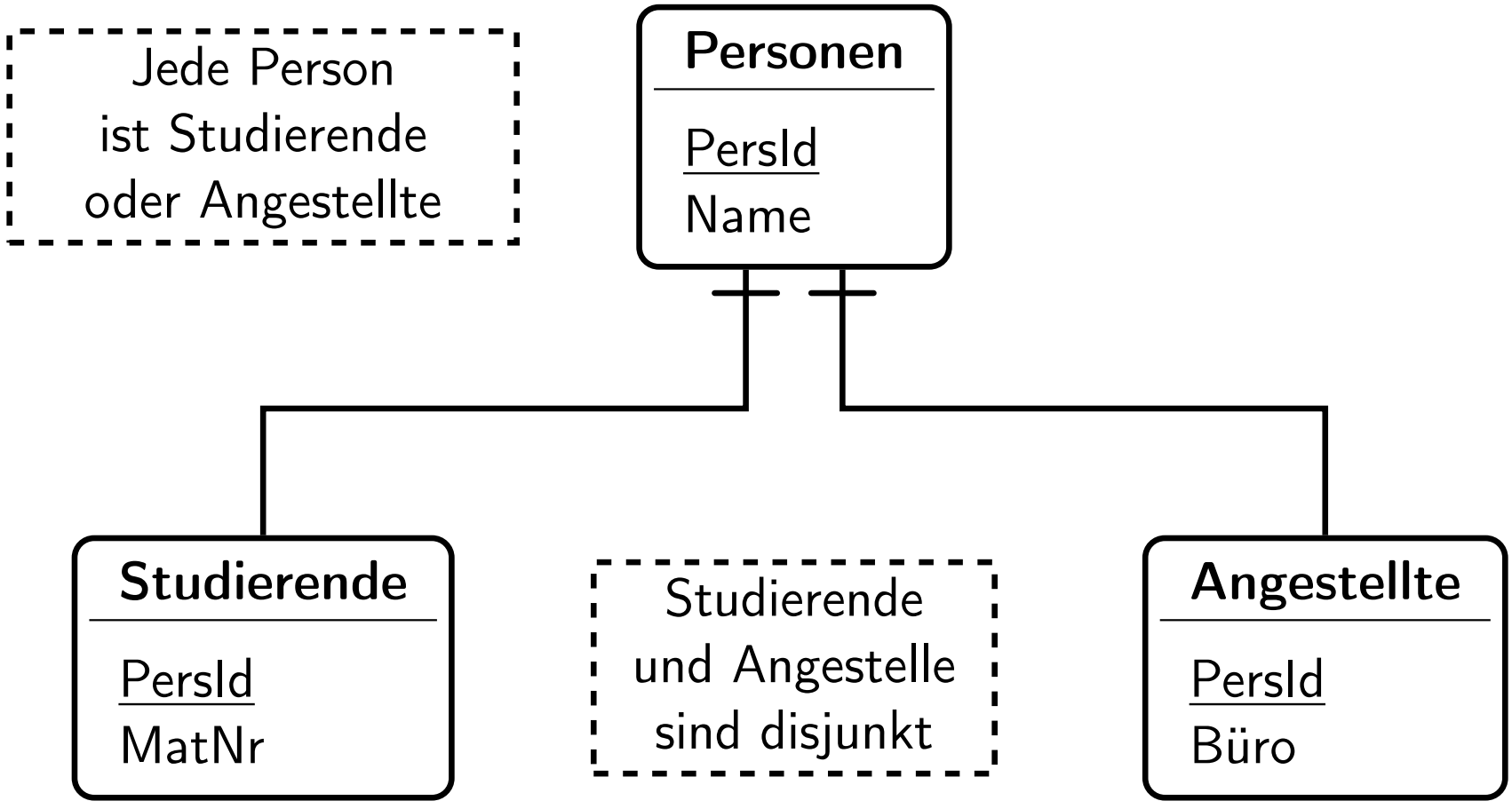


Bedingungen

Totalität Die Vererbungsrelation ist *total*, wenn es für jedes Tupel in der Basisrelation ein entsprechendes Tupel in mindestens einer der abgeleiteten Relationen gibt. In unserem Beispiel heisst das, es gibt keine Entitäten die nur Personen sind. Jede Person muss Studierender oder Angestellter sein.

Disjunktheit Die abgeleiteten Relationen heissen *disjunkt*, falls es kein Tupel in der Basisrelation gibt, zu welchem in mehr als einer abgeleiteten Relation ein entsprechendes Tupel existiert. In unserem Beispiel heisst das, es gibt keine Personen, die sowohl Studierende als auch Angestellte sind.

Diagramm


Jede Person
ist Studierende
oder Angestellte
Personen
PersId
Name
Studierende
PersId
MatNr
Studierende
und Angestelle
sind disjunkt
Angestellte
PersId
Büro

Vermeidung von Null

Personen mit zusätzlichem Attribut zur Korrektur der Brille:

$$\mathcal{S}_{\text{Personen}} := (\underline{\texttt{PersId}}, \texttt{Name}, \texttt{GebDatum}, \texttt{Brille}) \ .$$

Eine mögliche Instanz dieses Schemas ist:

Personen

PersId	Name	GebDatum	Brille
1	Eva	19710429	Null
2	Tom	19720404	Null
3	Eva	19680101	-3.5
4	Ann	19841214	Null
5	Bob	20140203	Null

Schema aufteilen

$$\mathcal{S}_{\text{AllePersonen}} := (\underline{\texttt{PersId}}, \texttt{Name}, \texttt{GebDatum})$$
$$\mathcal{S}_{\text{Brillenträger}} := (\underline{\texttt{PersId}}, \texttt{Brille}) \ ,$$

wobei `PersId` in $\mathcal{S}_{\text{Brillerträger}}$ ein Fremdschlüssel auf $\mathcal{S}_{\text{AllePersonen}}$ ist. Die obige Relation wird somit aufgeteilt in:

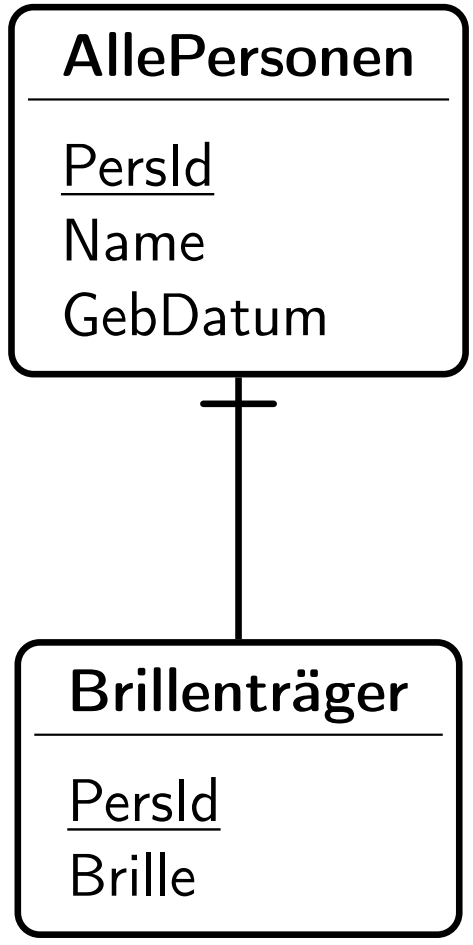
`AllePersonen`

PersId	Name	GebDatum
1	Eva	19710429
2	Tom	19720404
3	Eva	19680101
4	Ann	19841214
5	Bob	20140203

`Brillenträger`

PersId	Brille
3	-3.5

Diagramm


AllePersonen
PersId
Name
GebDatum
Brillenträger
PersId
Brille

Zwei Bedeutungen von `Null`

In der Tabelle `Personen` können wir nicht unterscheiden ob,

1. `Eva` keine Brille hat oder
2. die Korrektur von `Evas` Brille unbekannt ist.

In beiden Fällen lauter der Eintrag in `Personen`

$$(\ \texttt{1},\ \texttt{Eva},\ \texttt{19710429},\ \texttt{Null}\ )\ .$$

Wenn wir `Brillenträger` von `AllePersonen` ableiten, so können wir folgende Fälle unterscheiden:

1. `Eva` hat keine Brille. Dann gibt es in der Tabelle `Brillerträger` keinen Eintrag mit `PersId` 1.
2. Die Korrektur von `Evas` Brille ist unbekannt. Dann gibt es in `Brillerträger` einen Eintrag

$$(\ \texttt{1},\ \texttt{Null}\ )\ .$$

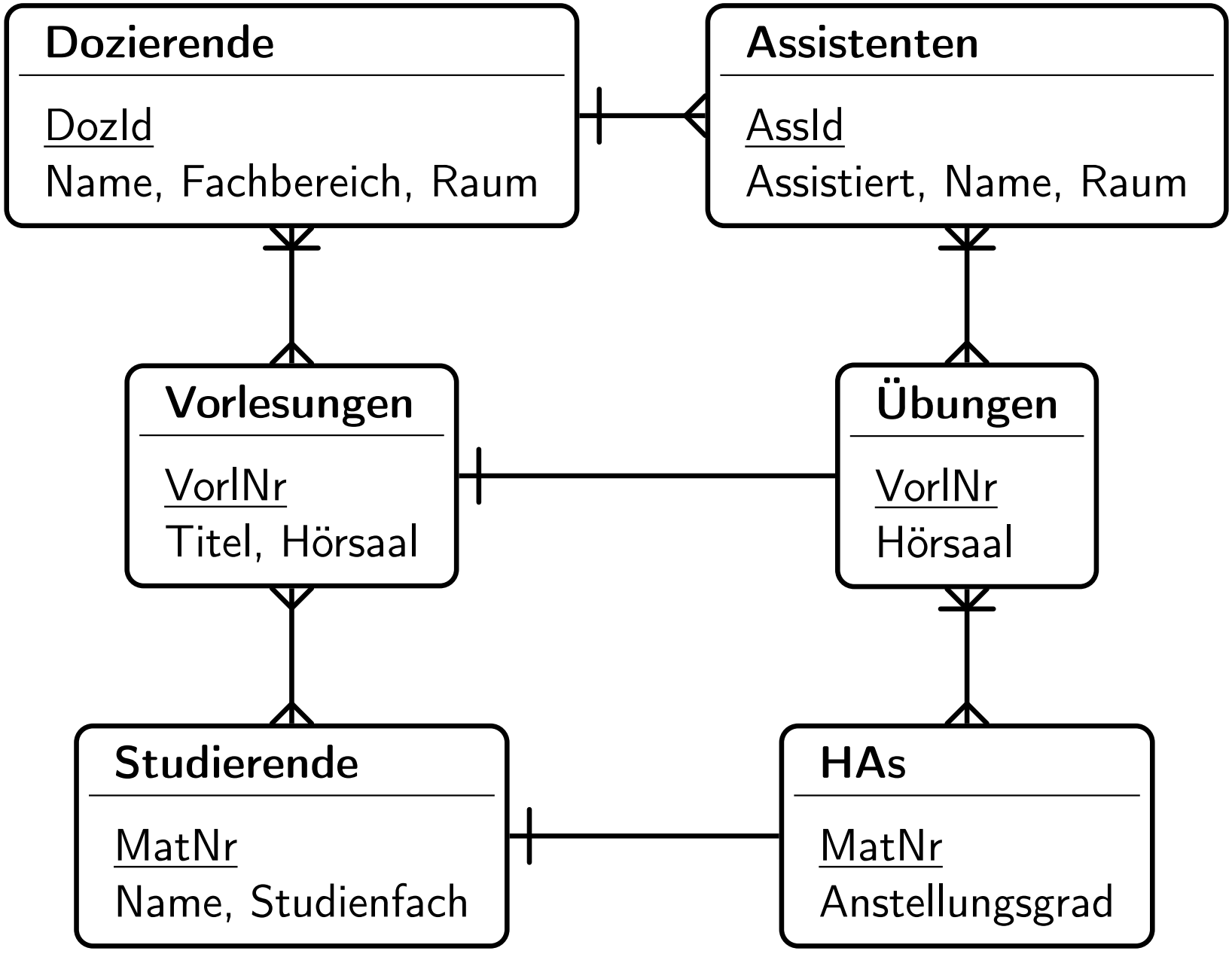
Hochschul Datenbank

Anforderungen. In einer Hochschul-Datenbank sind Daten über folgende Einzelheiten abzulegen:

1. Dozierende (Name, Fachbereich, Raum),
2. Assistenten (Name, Raum),
3. Vorlesungen (Nummer, Titel, Hörsaal),
4. Übungen zu Vorlesungen (Nummer, Hörsaal),
5. Studierende (Matrikel-Nummer, Name, Studienfach),
6. Hilfsassistenten (Matrikel-Nummer, Name, Anstellungsgrad).

Ferner ist zu beachten: Professoren haben Assistenten und halten Vorlesungen; Assistenten betreuen Übungen, welche wiederum nur in Verbindung mit einer Vorlesung stattfinden. Einer Übung können mehrere Hilfsassistenten zugeordnet werden (zur Korrektur von Übungsserien); ein Hilfsassistent ist jedoch insbesondere ein Studierender und hört als solcher Vorlesungen.

Diagramm (verkürzt)



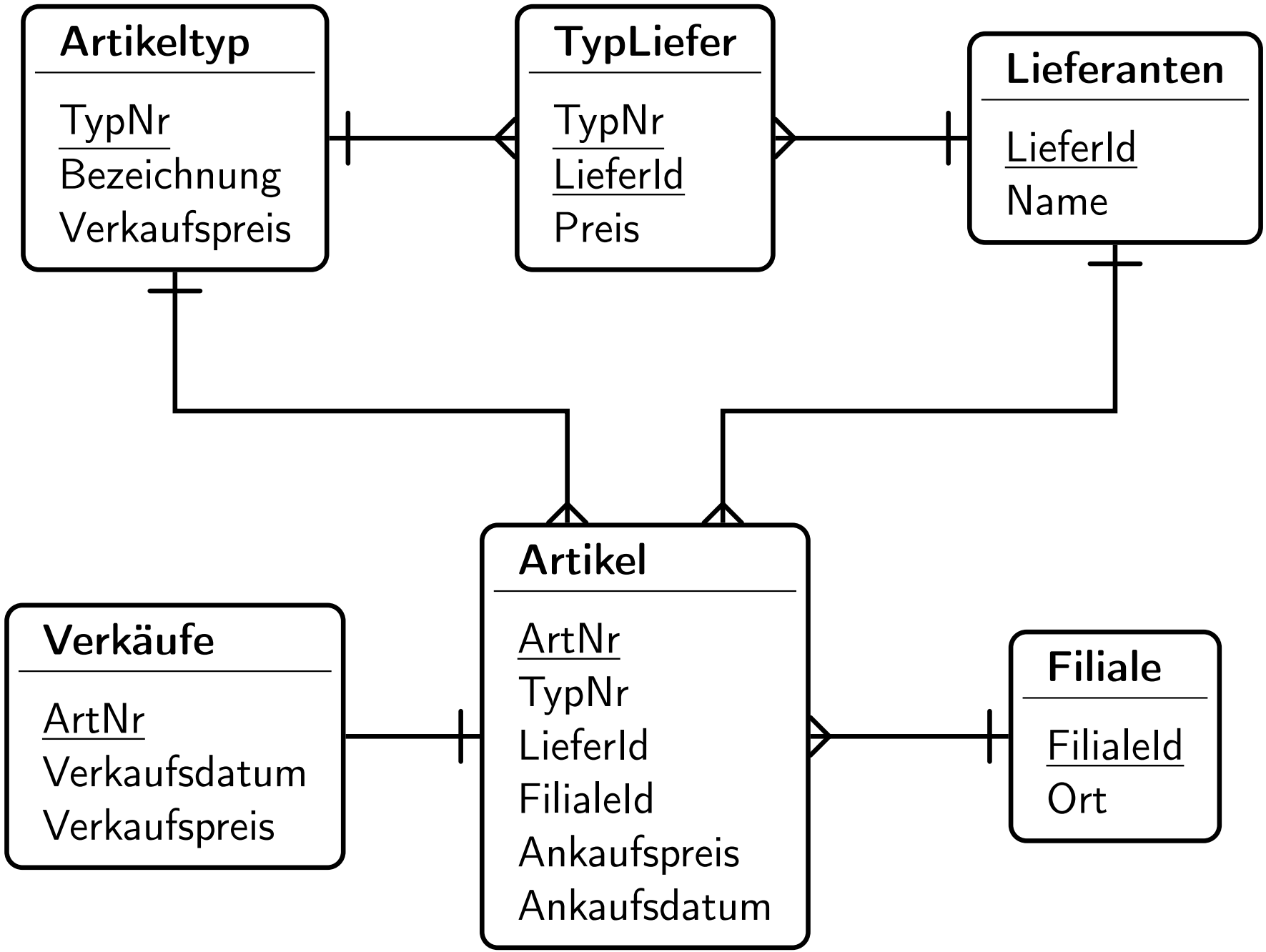
Überlegungen

1. `DozId` (in `Dozierende`) und `AssId` (in `Assistenten`) sind zusätzliche Attribute welche jeweils als Primärschlüssel dienen.
2. Ein Assistent ist bei *genau* einem Dozierenden angestellt. Um dies zu modellieren, ist in der Tabelle `Assistenten` das Attribut `Assistiert` ein Fremdschlüssel auf `Dozierende`.
3. Zu einer Vorlesung gibt es nur eine Übung und Übungen gibt es nicht ohne entsprechende Vorlesung.
4. `HAs` ist eine Spezialisierung von `Studierende`. Entsprechend wird das Attribut `Name` von `Studierende` geerbt und muss nicht extra in `HAs` erfasst werden.
5. Ein Hilfsassistent kann mehrere (muss jedoch mindestens eine) Übungen betreuen.
6. Ein Assistent kann mehrere (ev. auch keine) Übungen betreuen. Jede Übung hat mindestens einen Assistenten.
7. Das Schema garantiert nicht, dass ein Assistent, welche die Übungen zu einer Vorlesung betreut, auch bei einem Dozierenden dieser Vorlesung angestellt ist.

Warenhauskette

Anforderungen. Eine Datenbank für die Lagerverwaltung einer Warenhauskette soll Auskunft geben über das Sortiment jeder Filiale. Ausserdem soll für jeden Lieferanten ersichtlich sein, welchen Artikel er zu welchem Preis liefern kann. Für jeden Artikel sollen die Bezeichnung, für jeden verkauften Artikel der Verkaufspreis und das Verkaufsdatum, und für jeden gelieferten Artikel das Lieferdatum gespeichert werden.

Diagramm


Artikeltyp
TypNr
Bezeichnung
Verkaufspreis
TypLiefer
TypNr
LieferId
Preis
Lieferanten
LieferId
Name
Artikel
ArtNr
TypNr
LieferId
FilialeId
Ankaufspreis
Ankaufsdatum
Verkäufe
ArtNr
Verkaufsdatum
Verkaufspreis
Filiale
FilialeId
Ort

Überlegungen zum Schema

1. Wir haben allen Tabellen Ids hinzugefügt als Primärschlüssel.
2. Ein `Artikel` ist ein bestimmter Gegenstand, der an Lager sein kann oder bereits verkauft wurde. Falls z.B. die Warenhauskette Computer verkauft, so sind die einzelnen Computer `Artikel`. Ein bestimmtes Computermodell ist dann ein `Artikeltyp`.
3. Ein Artikel der ins Lager kommt, wird in der `Artikel`-Tabelle eingetragen. Wird er dann verkauft, so wird er auch noch in die Tabelle `Verkäufe` eingetragen (bleibt aber in `Artikel`).
4. `Preis` ist ein Attribut von `TypLiefer`, das angibt, welcher Lieferant diesen Artikeltyp gegenwärtig zu welchem Preis liefert.

Überlegungen zum Schema 2

1. Das Attribut `Verkaufspreis` in `Artikeltyp` gibt an, was der aktuelle Verkaufspreis für Artikel dieses Typs ist. So ist gewährleistet, dass alle gleichen Artikel auch gleich viel kosten. Bei Preisanpassungen (Aktionen) ist nur ein Update nötig und es muss nicht der Preis bei allen Artikeln einzeln angepasst werden.
2. Das Attribut `Ankaufspreis` in `Artikel` gibt an, zu welchem Preis dieser Artikel tatsächlich eingekauft wurde. Der aktuelle `Preis`, der in der Tabelle `TypLiefer` abgespeichert ist, kann davon abweichen.
3. Das Attribut `Verkaufspreis` in `Verkäufe` gibt an, zu welchem Preis dieser Artikel tatsächlich verkauft wurde. Der aktuelle Wert von `Verkaufspreis` in `Artikeltyp` kann davon abweichen.

Überlegungen zum Schema 3

Im DB-Schema für die Warenhauskette haben wir zur Vermeidung von `Null`-Werten das Muster aus dem Beispiel der Brillenträger angewendet. Das heisst, wir haben die Tabellen `Artikel` und `Verkäufe`. So gibt es keine `Null`-Werte in den Attributen `Verkaufsdatum` und `Verkaufspreis`. Gemäss Spezifikation soll unsere Datenbank zur Lagerverwaltung der Warenhauskette dienen. Jedoch ist es in unserem DB-Schema sehr aufwändig folgende Abfrage zu bearbeiten:

Welches ist der aktuelle Lagerbestand eines gegebenen Artikeltyps? (1)

Dazu müssen wir nämlich diejenigen Artikel des gegebenen Typs finden, welche *nicht* in der `Verkäufe`-Tabelle eingetragen sind.

Variante

Wir können diese Abfrage vereinfachen, indem wir die beiden Tabellen wieder zusammenfügen. Damit wird die Tabelle `Verkäufe` nicht mehr benötigt und sowohl `Verkaufspreis` als auch `Verkaufsdatum` sind Attribute von `Artikel`. Diese Attribute haben den Wert `Null`, falls der Artikel noch nicht verkauft wurde. Die Abfrage (1) wird nun bearbeiten indem alle Artikel des Typs gesucht werden, deren `Verkaufsdatum` den Wert `Null` hat.
Es ergibt sich nun ein neues Problem. Um die Abfrage

Finde alle verkauften Artikel eines gegebenen Typs (2)

zu beantworten, müssen alle Artikel des gegebenen Typs gefunden werden, deren Verkaufsdatum nicht `Null` ist. Allerdings kann es vorkommen, dass Abfragen, die auf nicht `Null` testen, nicht effizient durchführbar sind.

Variante 2

Um eine gute Performance zu erreichen, fügen wir der `Artikel`-Tabelle ein neues Attribut `Status` hinzu, welches zwei Werte annehmen kann: *an Lager* und *verkauft*. Die Abfrage (2) kann nun ganz einfach ausgeführt werden, indem auf Status *verkauft* getestet wird. In der endgültigen Version braucht es also die `Verkäufe`-Tabelle nicht mehr und die Tabelle `Artikel` hat die Attribute `ArtNr`, `TypNr`, `LieferId`, `FilialeId`, `Ankaufspreis`, `Ankaufsdatum`, `Verkaufspreis`, `Verkaufsdatum` und `Status`.