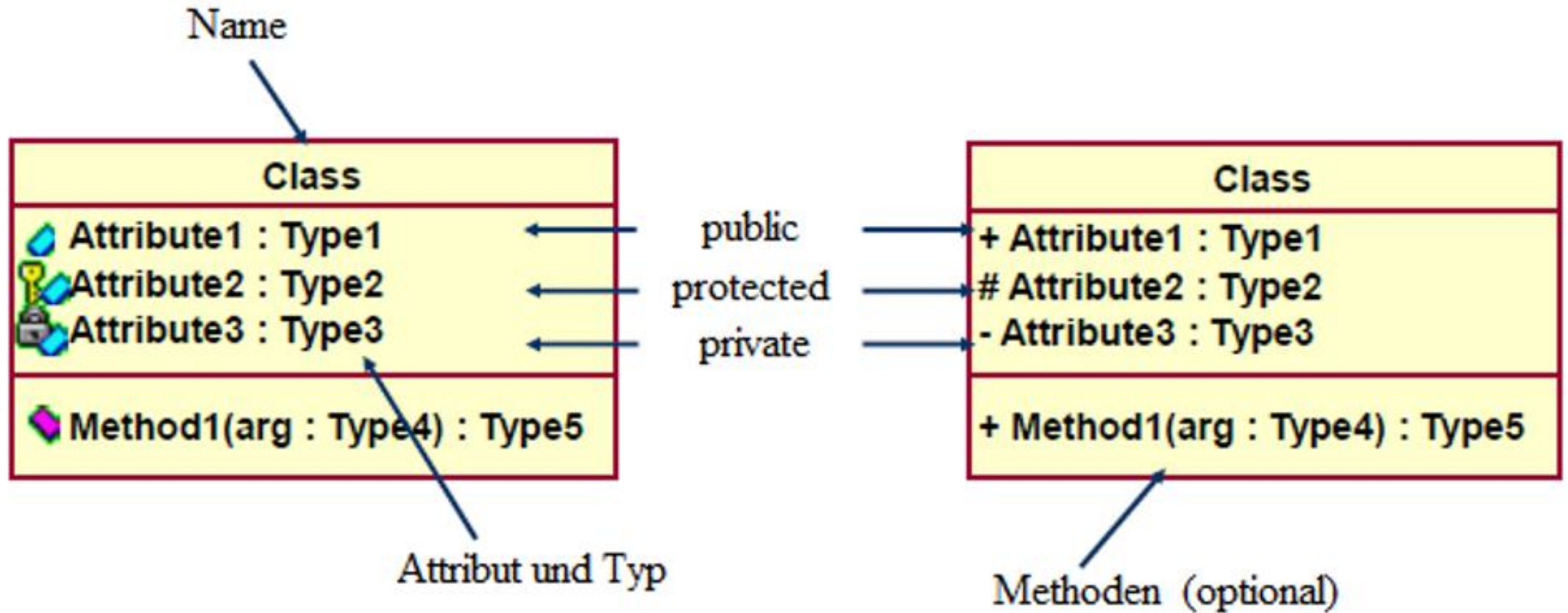


Konzeptueller Entwurf

UML Klassendiagramme



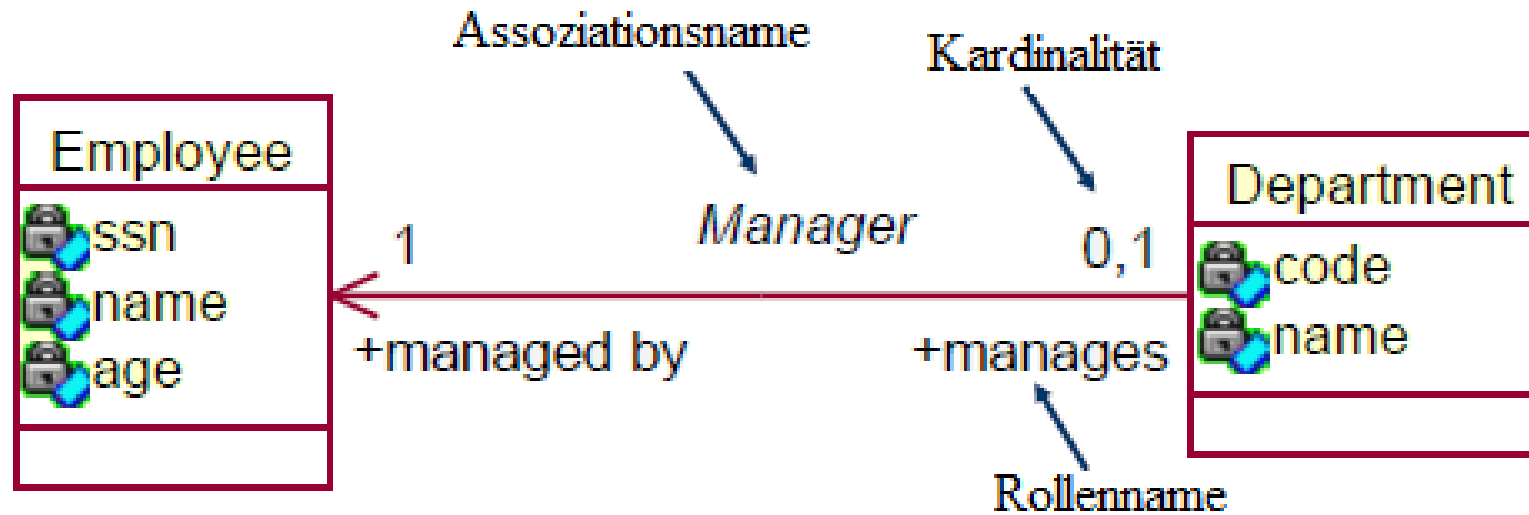
UML Assoziationen

- Entspricht Beziehungen
- Optional:
 - Assoziationsnamen
 - Leserichtung (\leftarrow oder \rightarrow), sonst bidirektional
 - Rollennamen
 - Kardinalitätsrestriktionen

UML Kardinalitätsrestriktionen

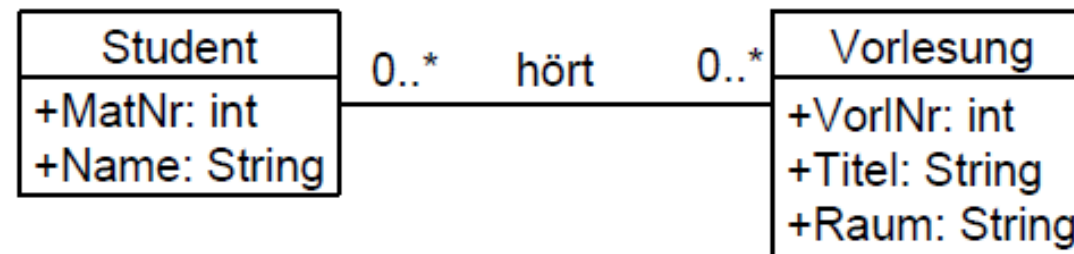
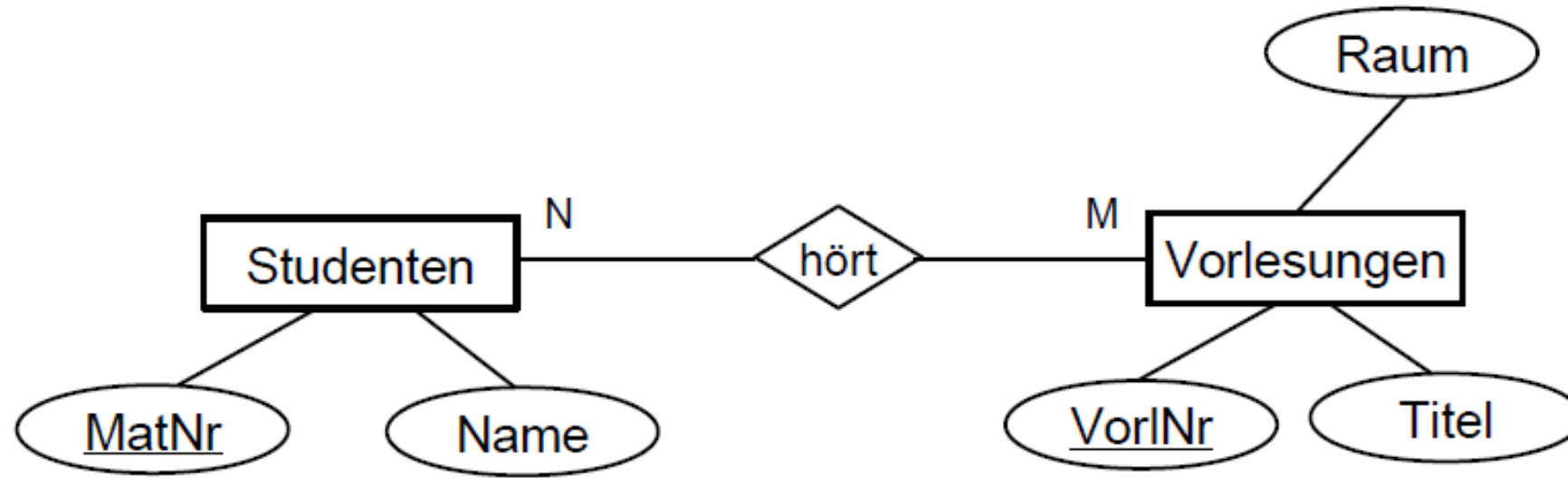
- $x \dots y$ mindestens x , maximal y Objekte nehmen an der Beziehung Teil
- $0 \dots *$ optionale Teilnahme an der Beziehung (0 oder mehrere)
- $1 \dots *$ obligatorische Teilnahme an der Beziehung (1 oder mehrere)
- $0 \dots 1$ es kann nur einen geben oder keinen
- 1 genau 1

UML Assoziationen



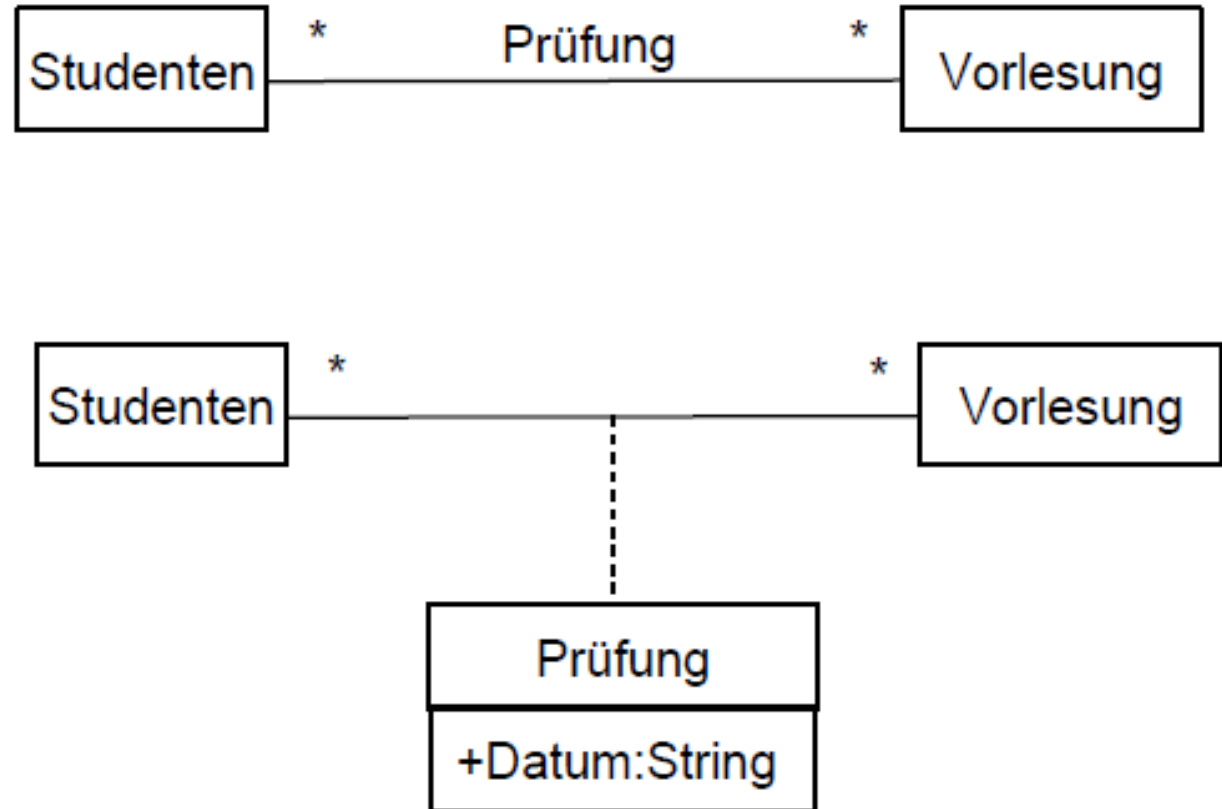
- An employee manages 0 or 1 departments
- A department is managed by 1 employee

ER vs. UML Beispiel



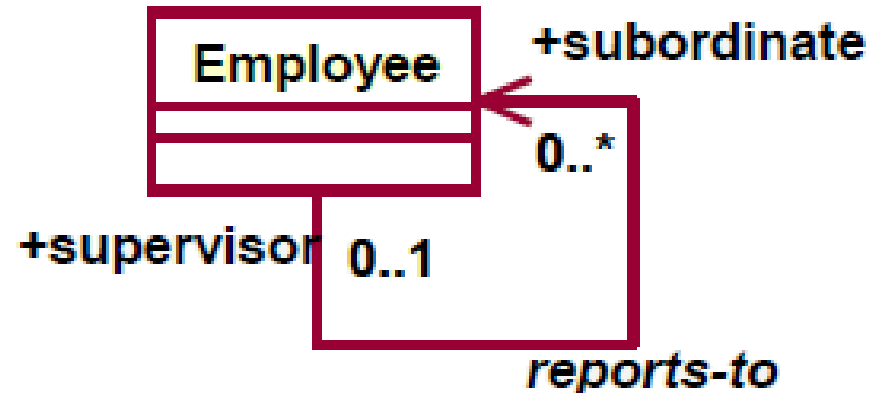
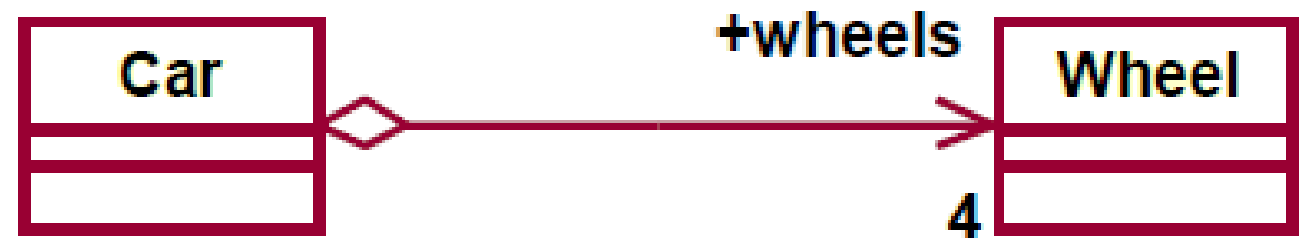
UML Assoziationsklassen

- Für Beziehungen mit eigenen Attributen ist eine Assoziationsklasse notwendig
 - Gestrichelte Linie
 - Name der Assoziationsklasse entspricht der Assoziation



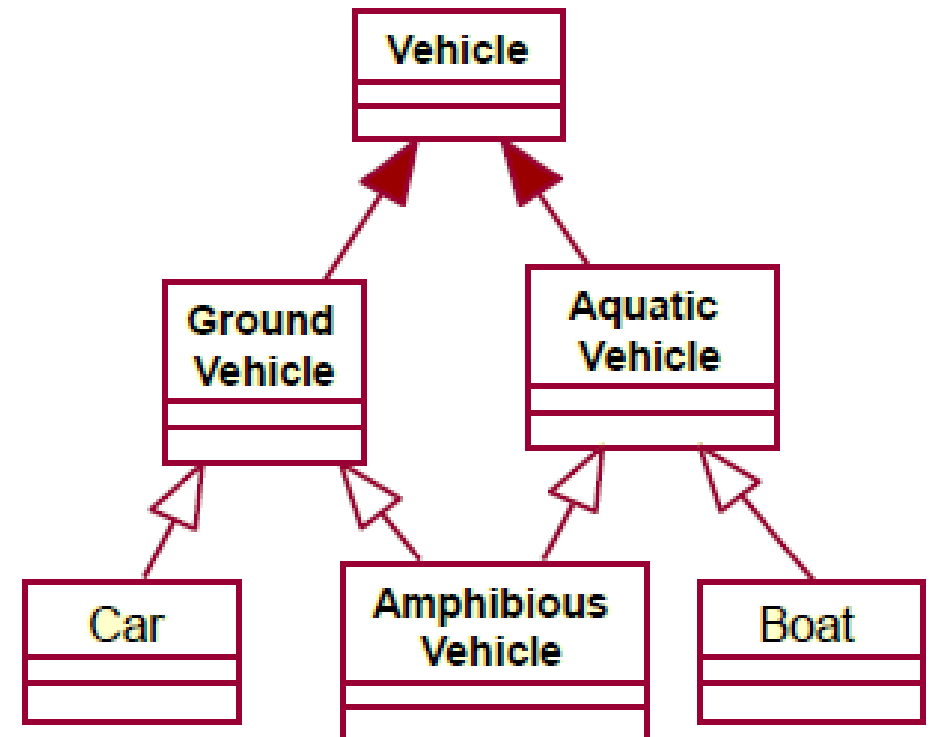
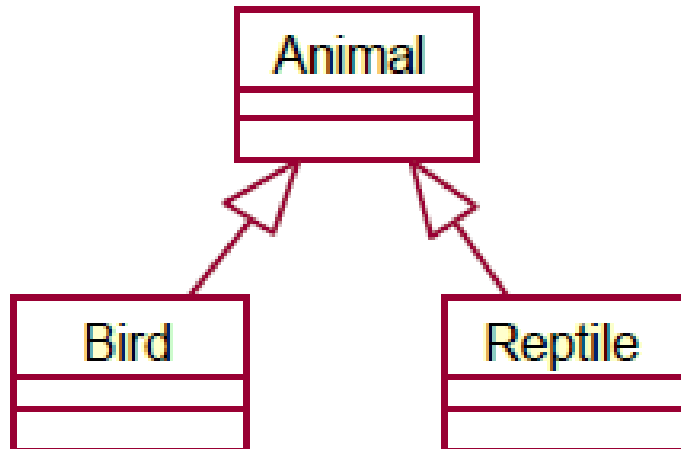
UML part-of (Teil-von) Beziehungen

- Aggregation
 - part-of Beziehung zwischen Subkomponente und Superkomponente
- Komposition
 - Teil-Objekt gehört genau zu einem Aggregatobjekt
- Reflexive Assoziation



UML is-a Beziehung - Vererbung

- Alle Instanzen der Subklasse sind auch Instanzen der Superklasse
- Vererbung von Eigenschaften (Attribute, Integritätsbedingungen, Methoden) der Superklasse an alle Subklassen
- Wiederverwendbarkeit, Erweiterbarkeit
- Keine Wiederholung (Redundanzen)

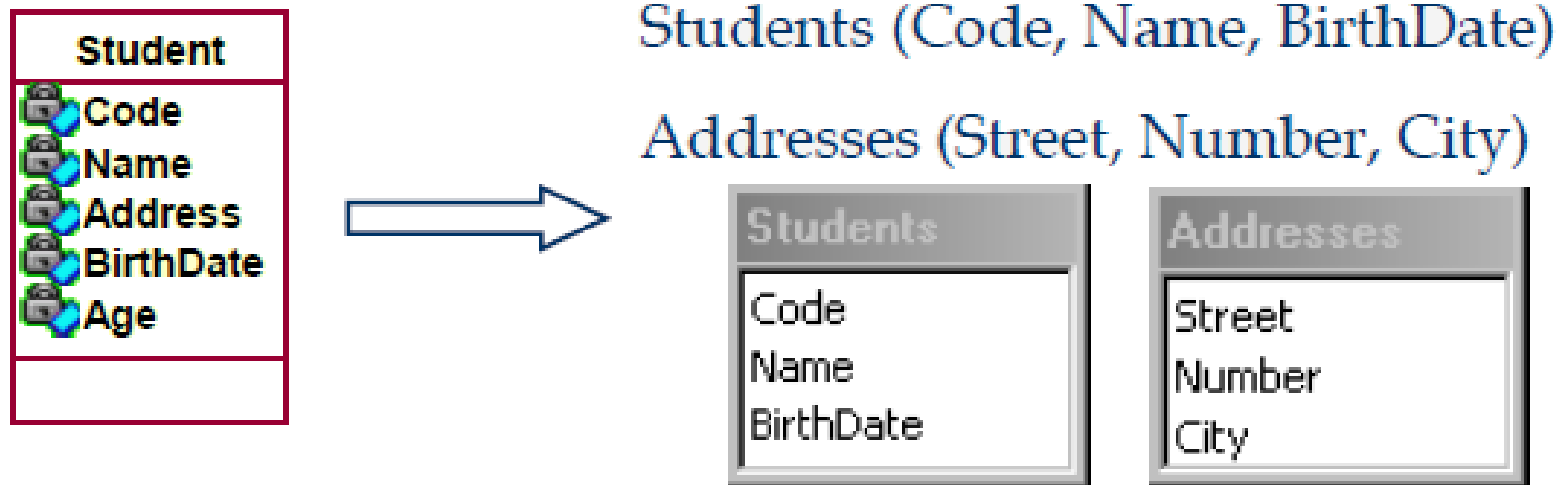


Konzeptueller Schema \Rightarrow Relationales Datenmodell

- Probleme für 1:1 Transformation der Klassen in Tabellen:
 - **Zu viele Tabellen** – es können mehr Tabellen als nötig erstellt werden
 - **Zu viele Joins** – zu viele Tabellen \Rightarrow zu viele Joins in den Abfragen
 - **Fehlende Tabellen** – M:M Beziehungen brauchen eine dritte Tabelle
 - **Falsche Modellierung der Vererbung**
 - **Denormalisierung der Daten** – manche Daten kommen in vielen Tabellen vor

Transformation der Klassen in Tabellen

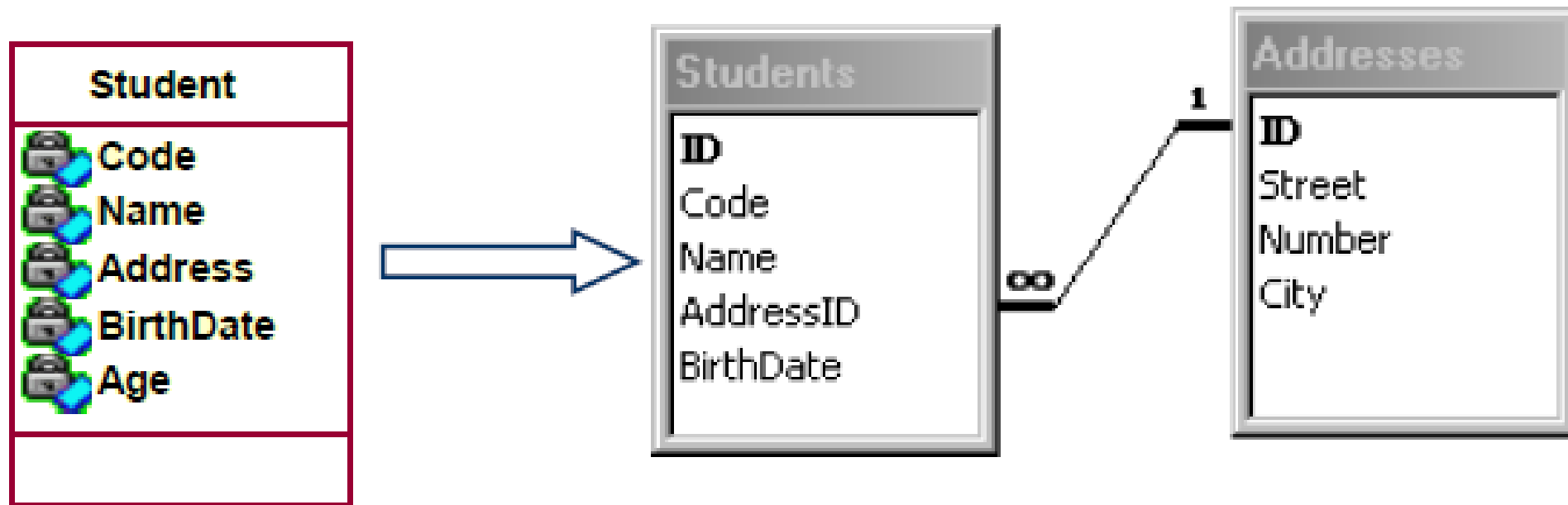
- Name der Tabelle = Plural von dem Klassennamen
- Alle einfache (atomare) Attribute werden Attribute in der Tabelle
- Zusammengesetzte Attribute werden als neue Tabelle modelliert
- Abgeleitete Attribute werden nicht in Tabellen gespeichert



Transformation der Klassen in Tabellen

- Ersatzschlüssel (surrogate keys) – Schlüssel, die nicht aus der Domäne des Problems kommen (z.B. IDs)
- In UML gibt es keine Schlüssel
- Eine gute Methode: wenn möglich kann man automatisch generierte Schlüssel (von DBMS) benutzen (auto-increment IDs)
 - Einfach zu verwalten (wird vom DBMS gemacht)
 - Effizient
 - Vereinfacht das Definieren von Fremdschlüssel
 - Primärschlüssel – ID
 - Fremdschlüssel - <TabelleName>ID

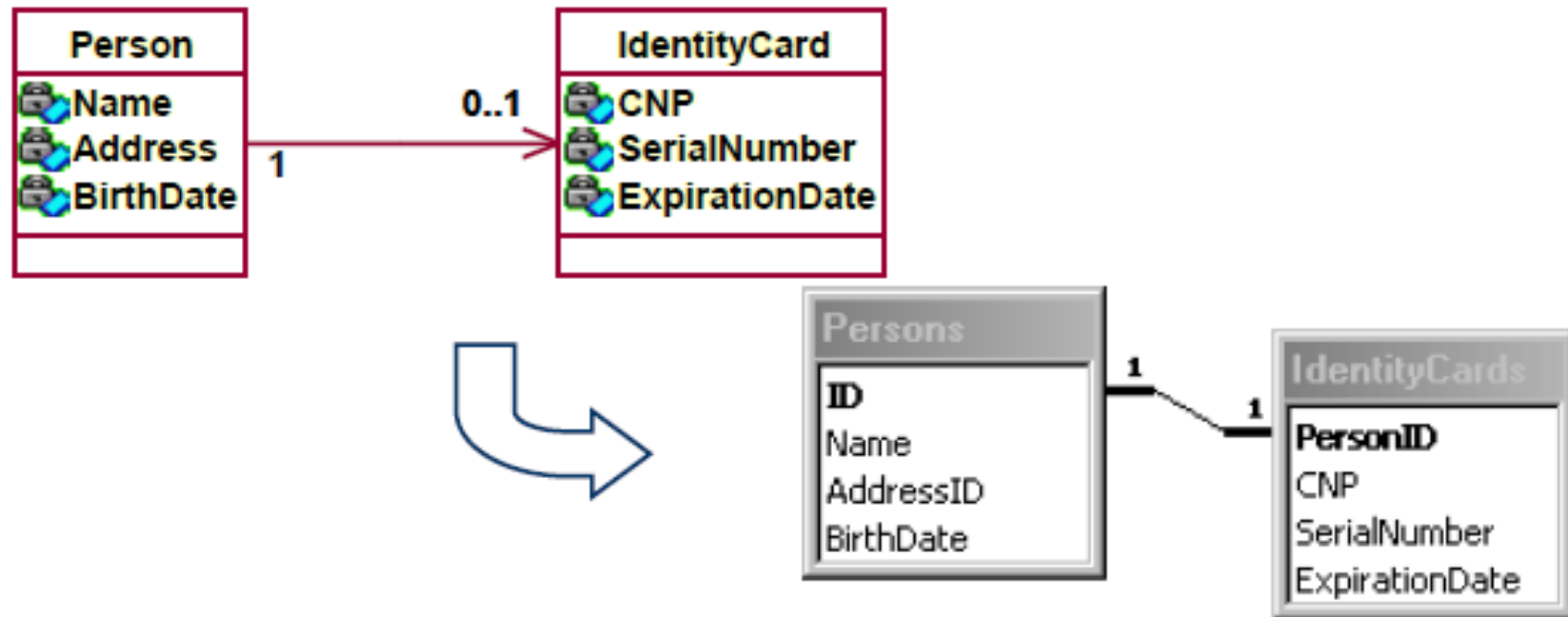
Transformation der Klassen in Tabellen



Transformation der Assoziation Beziehungen

- **1 : 0,1**

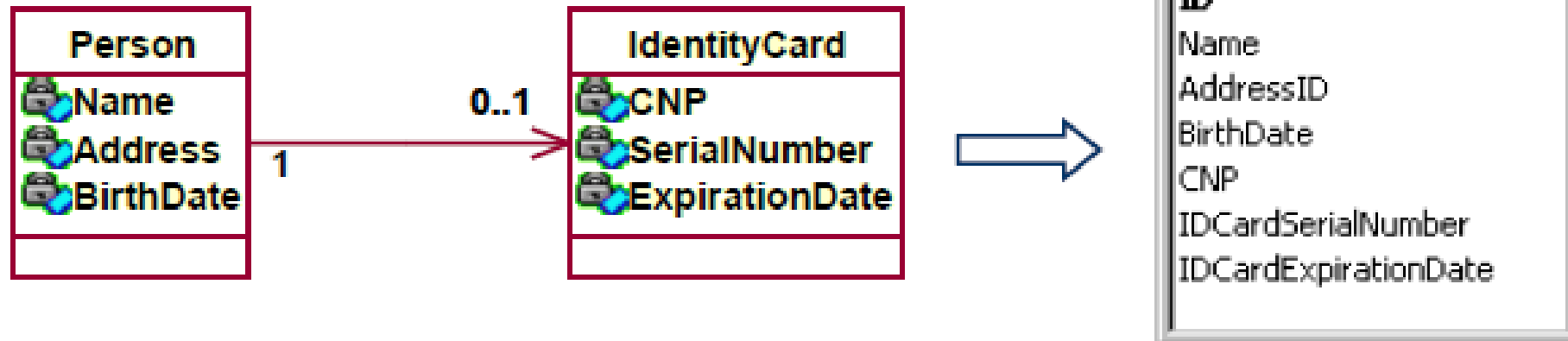
- Man erstellt eine Tabelle für jede Klasse aus der Assoziation
- der Primärschlüssel der „0..1“ Tabelle verweist auf den Schlüssel der „1“ Tabelle



Transformation der Assoziation Beziehungen

- **1 : 1**

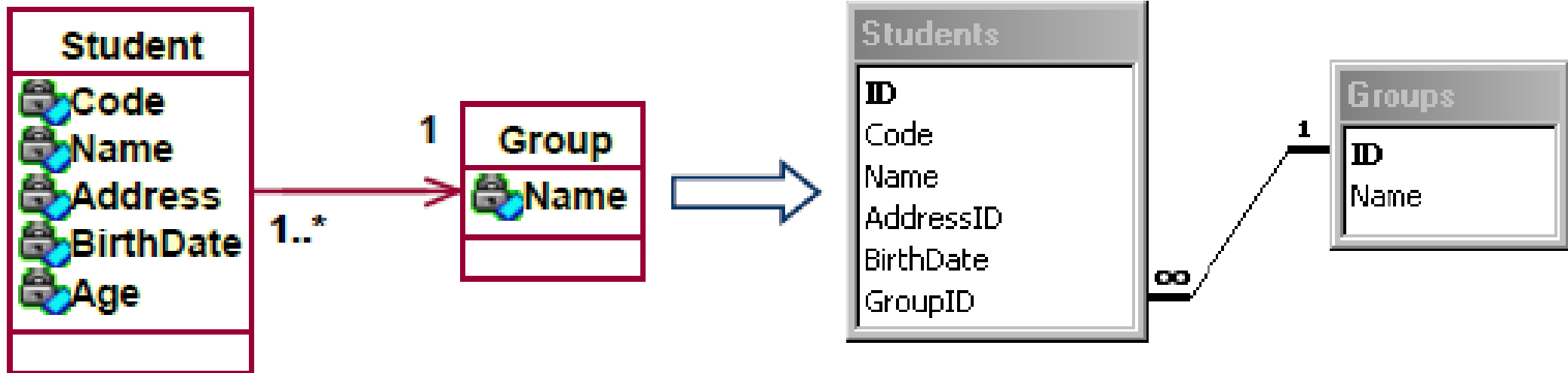
- Meistens erstellt man eine einzige Tabelle, welche die Attribute beider Klassen enthält
- Diese Lösung kann man auch für 1 : 0..1 benutzen, wenn es wenige Fälle gibt, in denen Objekte der ersten Klasse mit keinem Objekt der zweiten Klasse in Beziehung sind (wenige Null-Werte)



Transformation der Assoziation Beziehungen

- **1 : M (1 : 1..*)**

- Erstelle eine Tabelle für jede Klasse aus der Assoziation
- Der Schlüssel der „1“ Tabelle ist ein Fremdschlüssel in der „M“ Tabelle
(anders gesagt: ein Fremdschlüssel in der „M“ Tabelle verweist auf den Primärschlüssel der „1“ Tabelle)

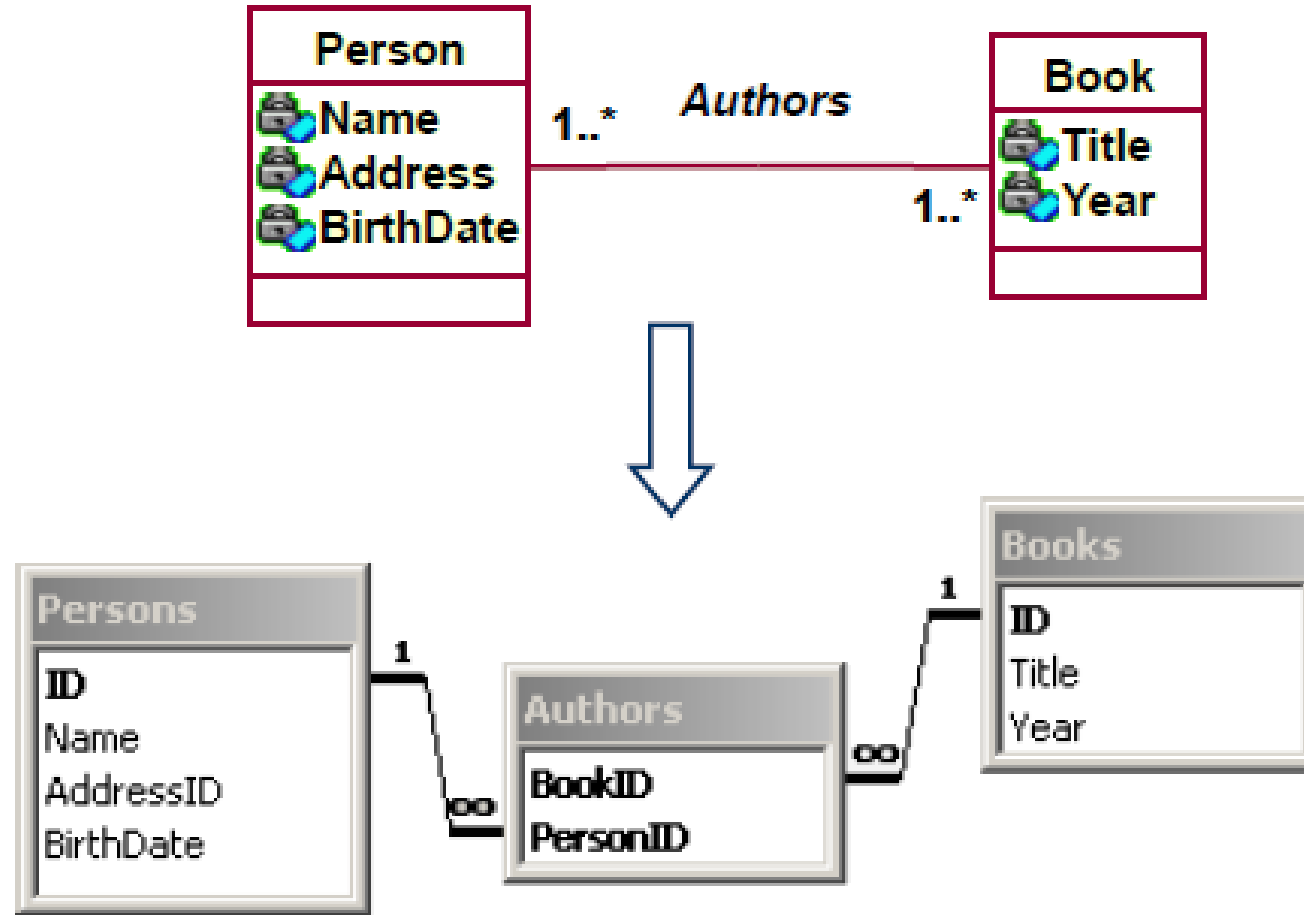


Transformation der Assoziation Beziehungen

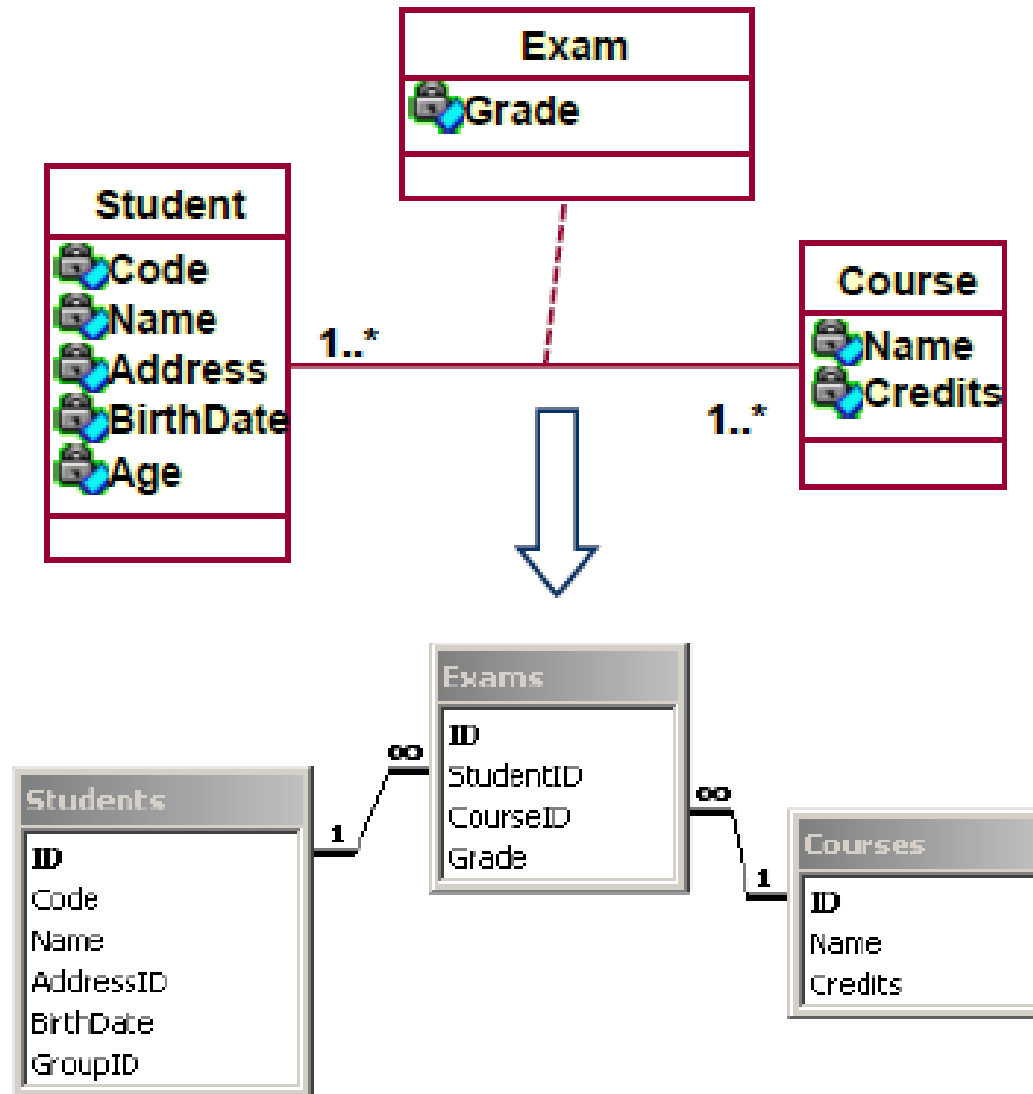
- **M : N (1 .. * : 1 .. *)**

- Man erstellt eine Tabelle für jede Klasse aus der Assoziation
- Man erstellt eine zusätzliche Tabelle (cross table/ Durchschnittstabelle)
- Die Primärschlüssel der ursprünglichen Tabellen werden als Fremdschlüssel in dem Cross Table definiert
- Der Primärschlüssel in dem Cross Table wird meistens als Zusammensetzung der Fremdschlüssel definiert (manchmal braucht man ein zusätzliches Attribut, z.B. ein Datum, um die Eindeutigkeit zu erhalten)
- Manchmal benutzt man in dem Cross Table einen Ersatzschlüssel (ID) als Primärschlüssel
- Wenn die Assoziation eine Assoziationsklasse hat, werden alle Attribute der Assoziationsklasse in dieser Cross Table eingefügt
- Meistens besteht der Name der Cross Table aus den Namen der Tabellen deren Beziehung dieser modelliert

Transformation der Assoziation Beziehungen



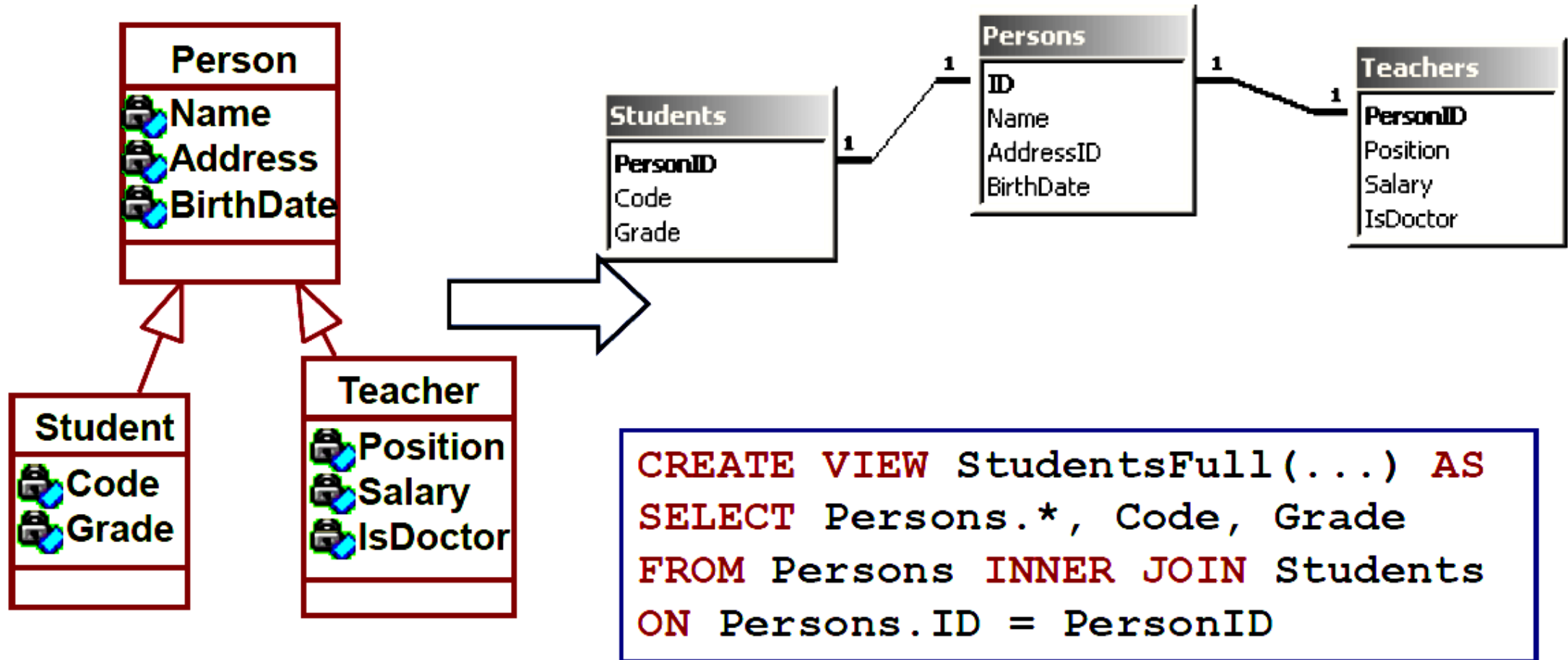
Transformation der Assoziation Beziehungen



Transformation der Vererbung

- 1. Methode
 - Man erstellt eine Tabelle für jede Klasse, ein Sicht (View) für jedes Paar Superklasse-Subklasse
 - Flexibilität – erlaubt neue Subklassen ohne Auswirkungen auf die existierenden Tabellen und Sichten
 - Diese Methode erstellt die meisten Tabellen und Sichten
 - Effizienzprobleme – jede Abfrage braucht einen Join

Transformation der Vererbung

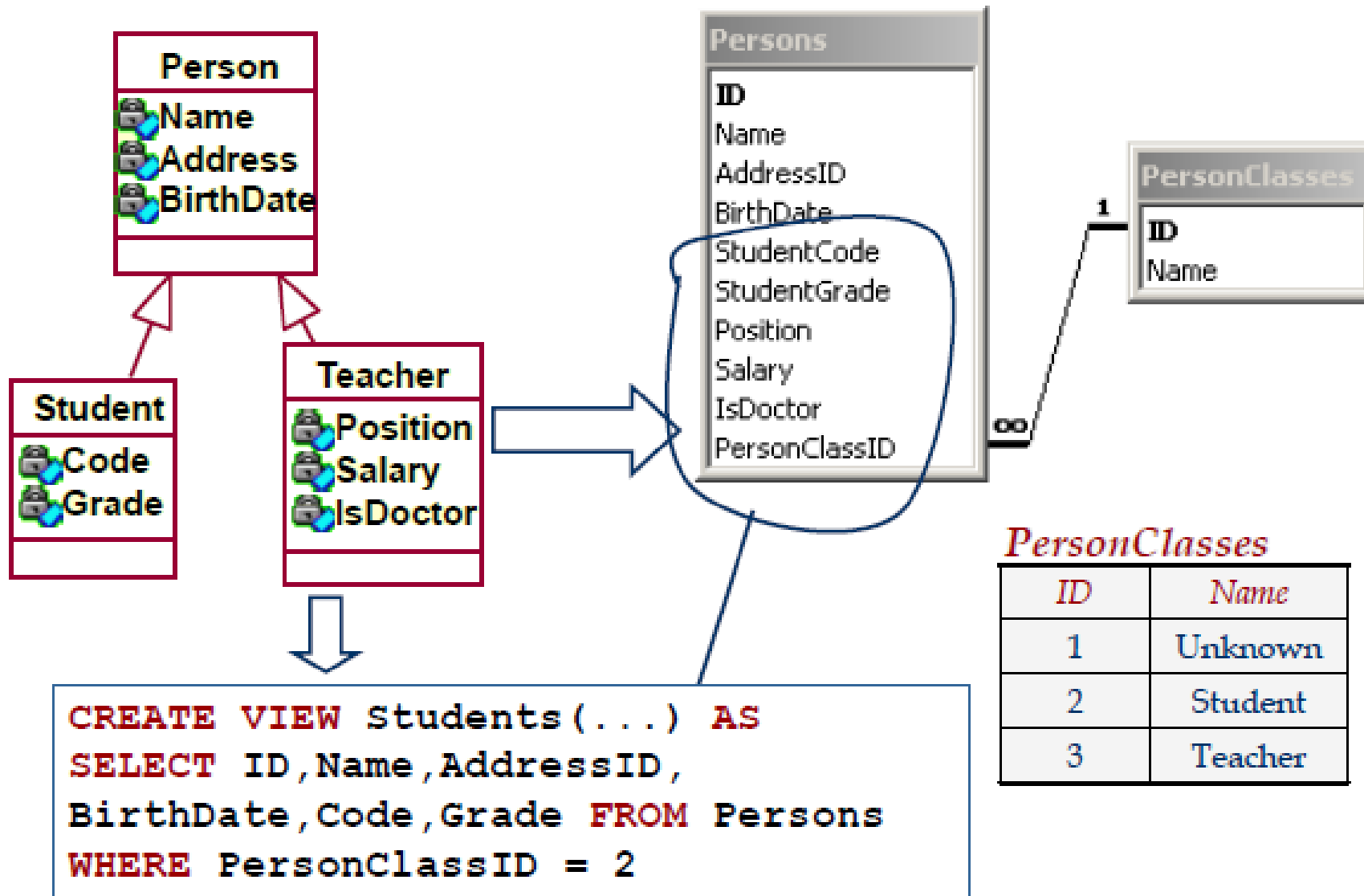


Transformation der Vererbung

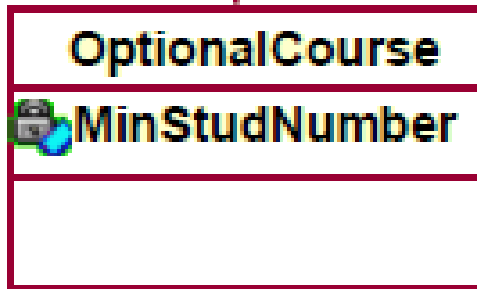
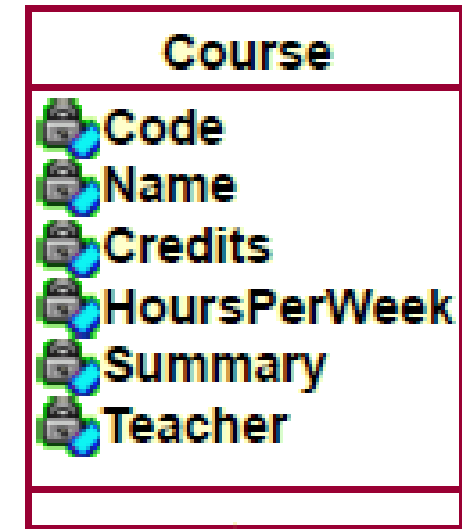
- 2. Methode

- Man erstellt eine einzige Tabelle (für die Superklasse) und man erstellt zusätzliche Attribute für die Subklassen
- Optional, kann man eine Tabelle von Subklassen definieren und Sichten für jede Subklasse
- Diese Methode erstellt am wenigsten Tabellen
- Gute Effizienz
- Eine neue Subklasse \Rightarrow man muss die Struktur der Datenbank ändern (neue Attribute)
- Ein Tupel hat eine größere „Länge“ (es können Nullwerte vorkommen) \Rightarrow kann die Effizienz beeinflussen

Transformation der Vererbung



Transformation der Vererbung



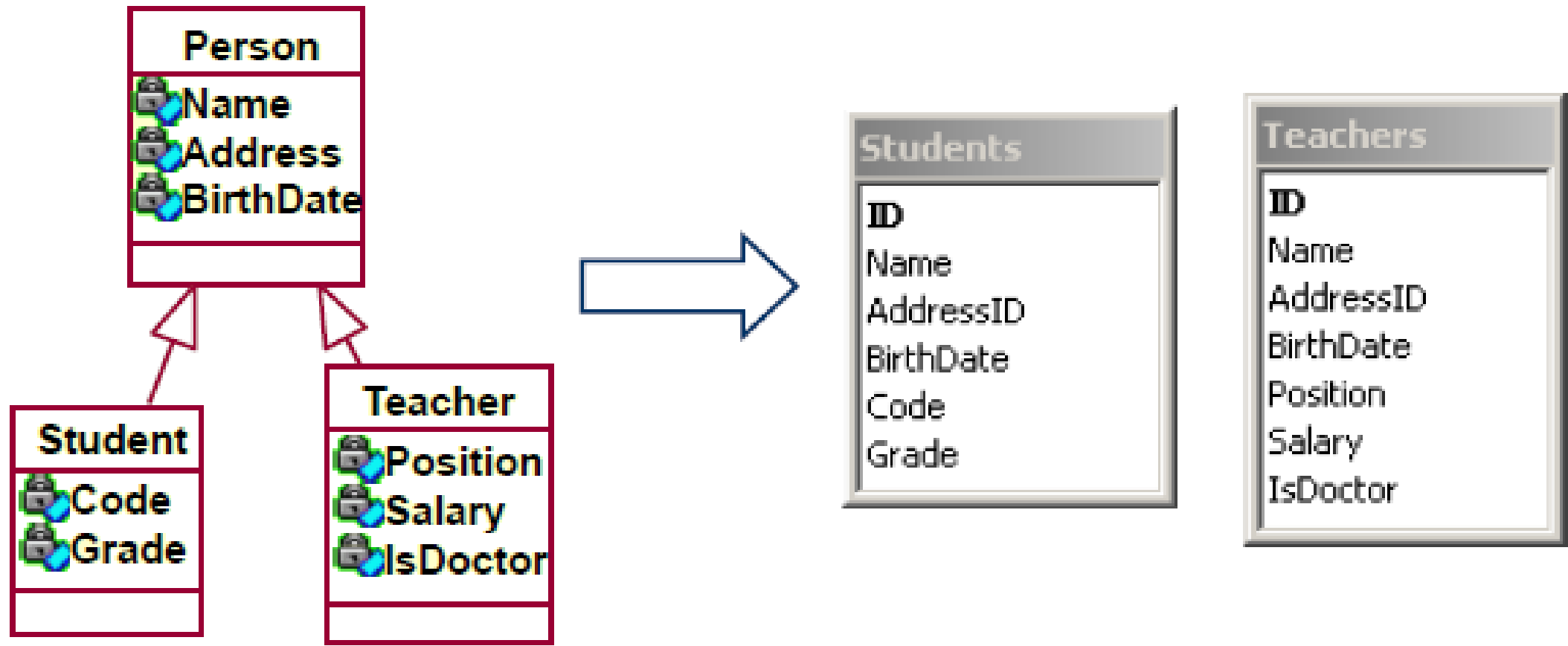
allow NULL

Transformation der Vererbung

- 3. Methode

- Man erstellt eine Tabelle für jede Subklasse und die Attribute der Superklasse werden in jeder Subklasse–Tabelle eingefügt
- Ziemlich gute Effizienz
- Eine neue Subklasse verursacht keine Strukturänderungen
- Änderungen in der Struktur der Superklasse verursachen Änderungen in der Struktur aller Tabellen

Transformation der Vererbung



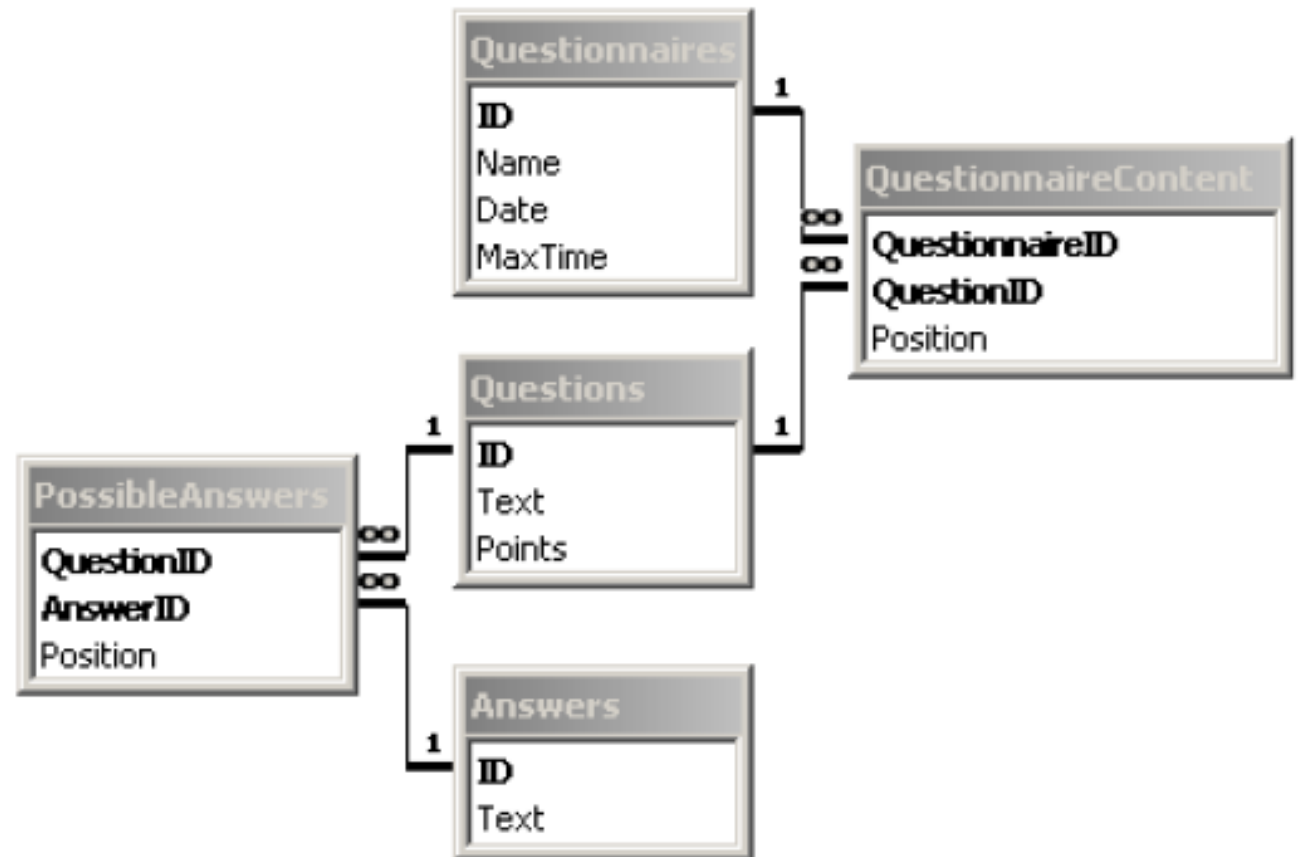
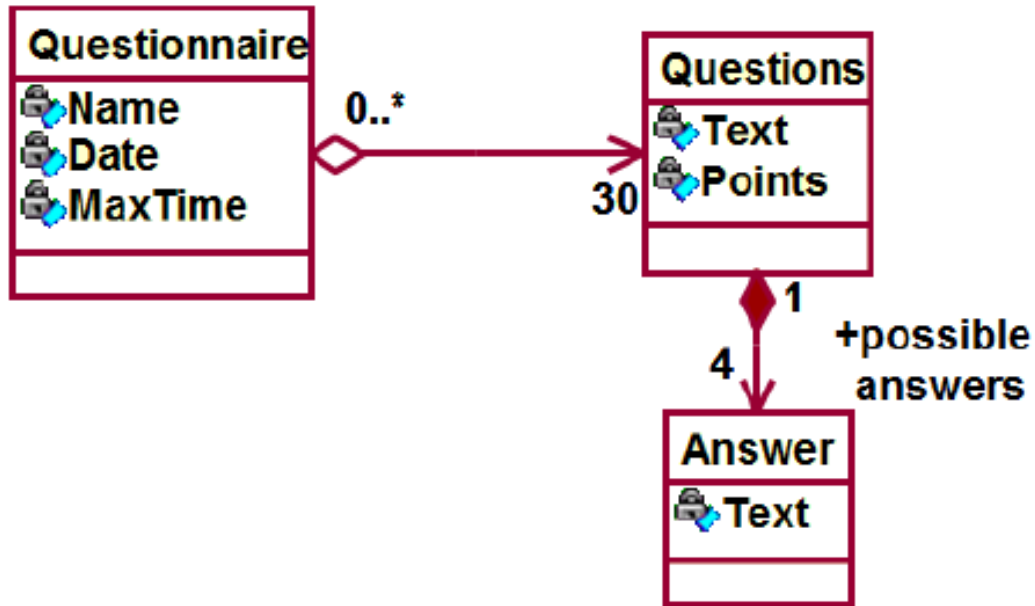
Transformation der Vererbung

- **Welche Methode ist besser?**
 - Wenn die Anzahl der Tupel in den Tabellen klein ist (also Effizienz ist kein Problem), dann kann man die flexible Methode wählen, also die **1. Methode**
 - Wenn die Superklasse wenige Attribute hat im Vergleich zu den Subklassen, dann wählt man am besten die **3. Methode**
 - Wenn die Subklassen wenige Tupeln haben, dann wählt man am besten die **2. Methode**

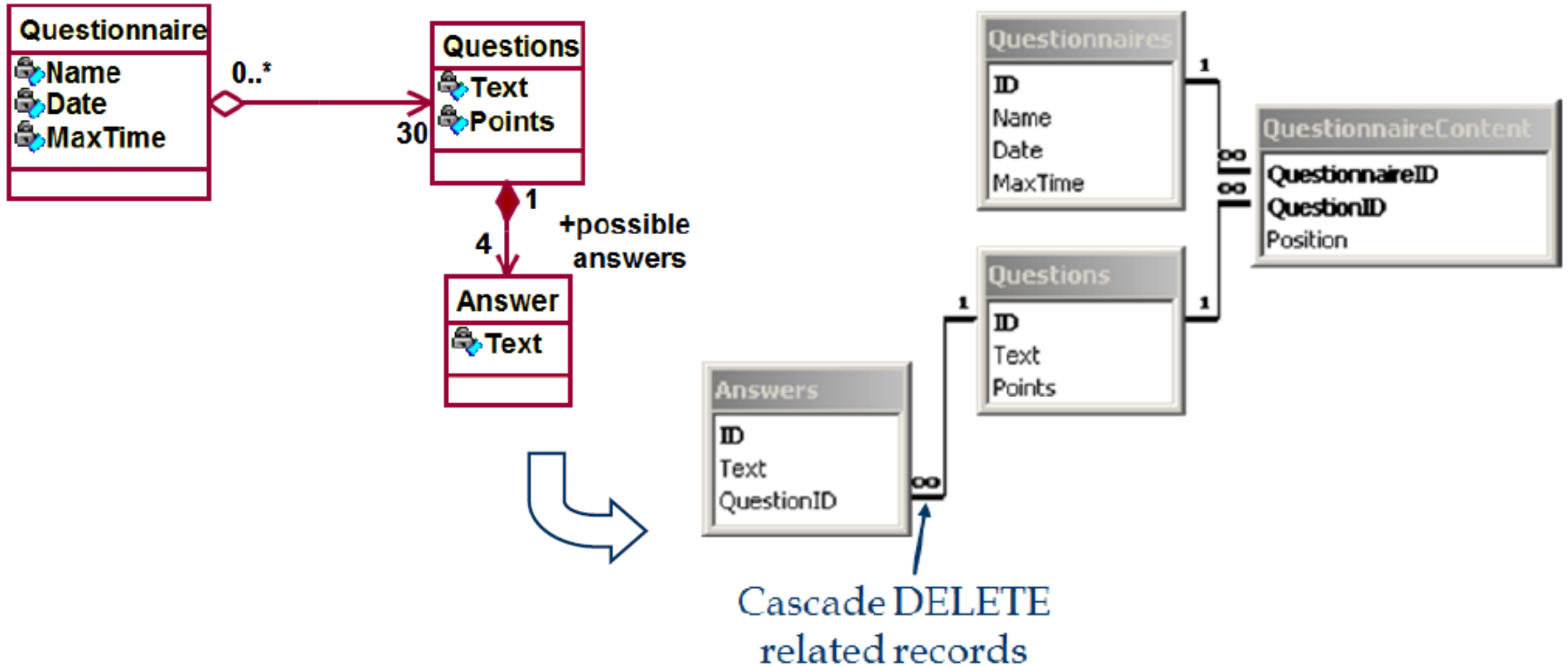
Transformation der Aggregation und Komposition

- Aggregationen und Kompositionen werden ähnlich wie Assoziationen modelliert
- Kompositionen werden oft in derselben Tabelle modelliert, da diese 1:1 Beziehungen haben
- Wenn Komposition in mehreren Tabellen modelliert wird, dann muss unbedingt ON DELETE CASCADE implementiert werden (für Aggregation ist das nicht notwendig)
- eine feste Anzahl von „Teile“ aus einem „Ganzen“ \Rightarrow genau dieselbe Anzahl von Fremdschlüssel in der Tabelle „Ganzen“

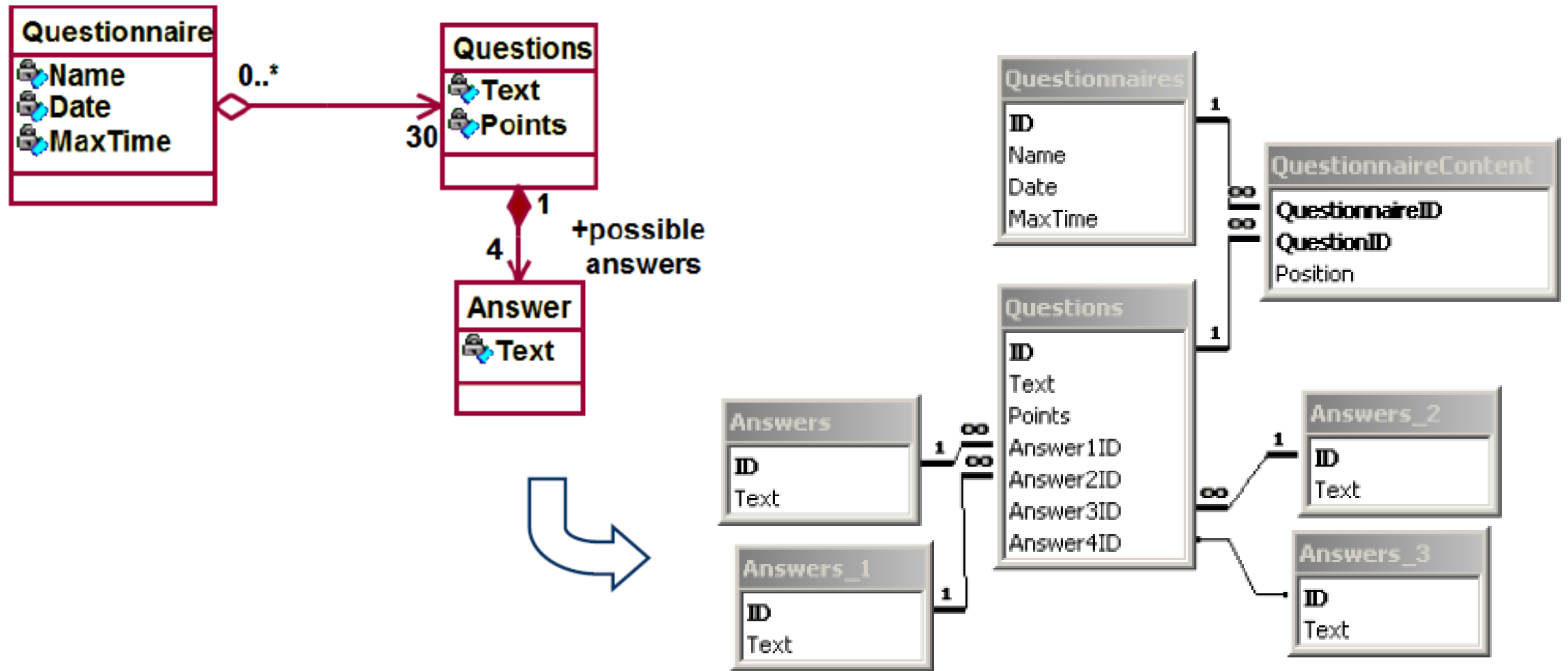
Transformation der Aggregation und Komposition



Transformation der Aggregation und Komposition

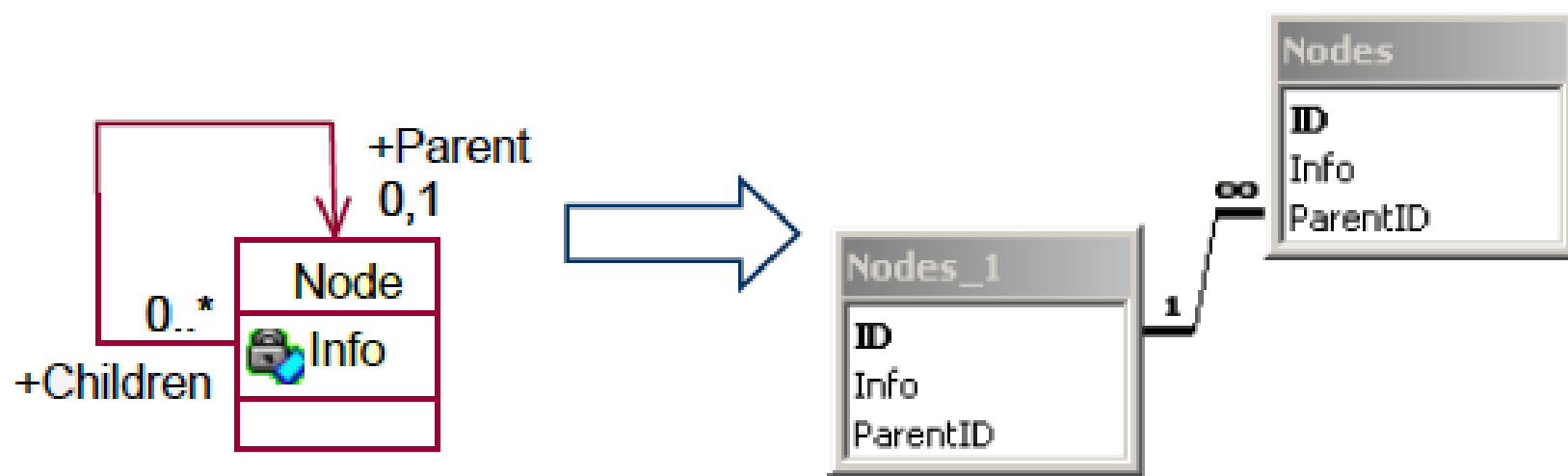


Transformation der Aggregation und Komposition



Transformation der Reflexiven Assoziation

- Man fügt ein Fremdschlüssel ein, der auf dieselbe Tabelle verweist (rekursive Relation)
- Wenn die Eigenschaft ON DELETE CASCADE benutzt wird, und zwei Objekte verweisen beide aufeinander, dann wird das Löschen eines der Objekte eine Fehlermeldung erzeugen



Transformation der Reflexiven Assoziation

- ON DELETE CASCADE gibt eine Fehlermeldung aus auch wenn zwei Tabellen gegenseitig auf die andere Tabelle verweisen

