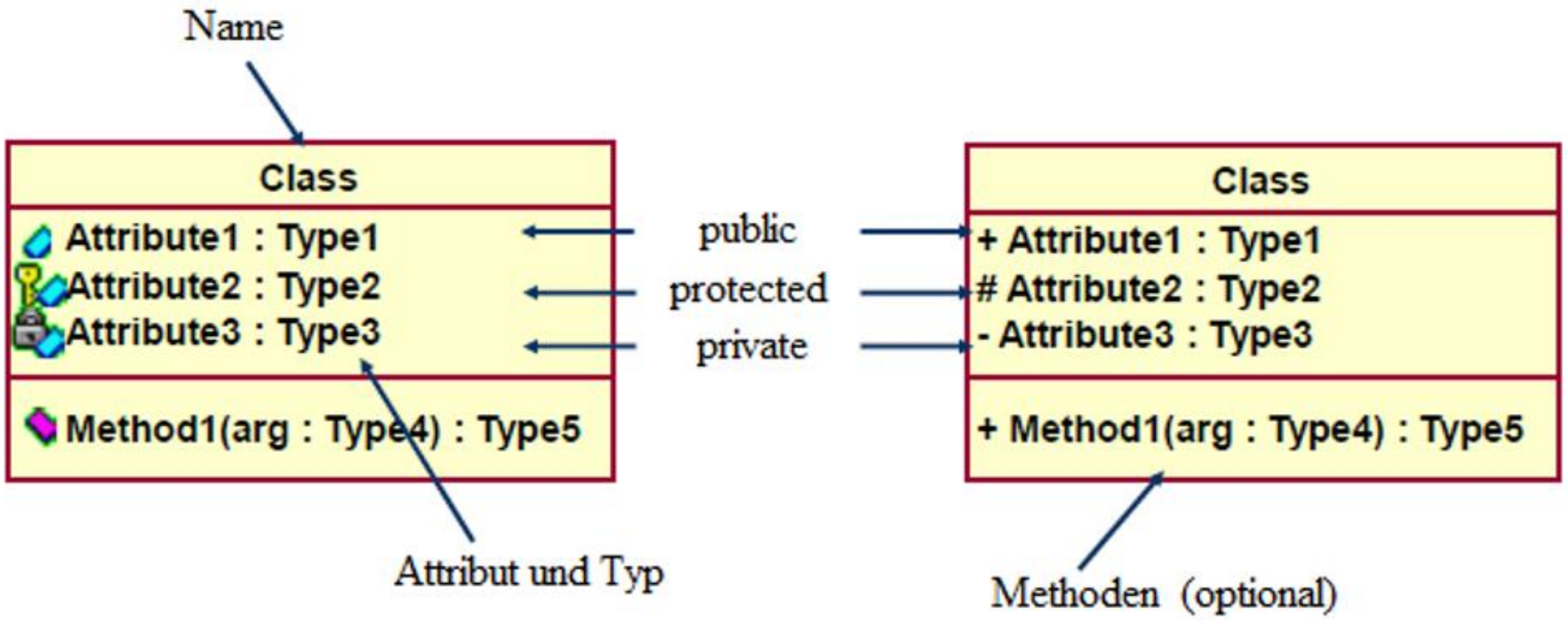
Konzeptueller Datenbankentwurf

UML Klassendiagrame



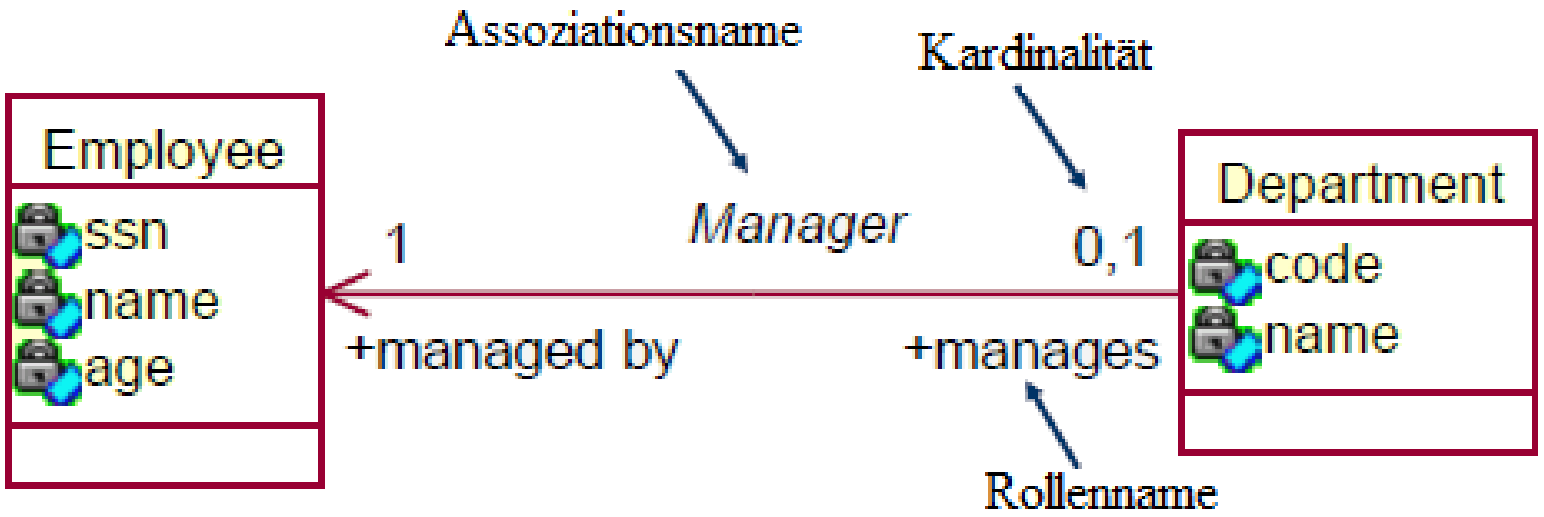
UML Assoziationen

- Entspricht Beziehungen
- Optional:
  - Assoziationsnamen
  - Leserichtung (← oder →), sonst bidirektional
  - Rollennamen
  - Kardinalitätsrestriktionen

UML Kardinalitätsrestriktionen

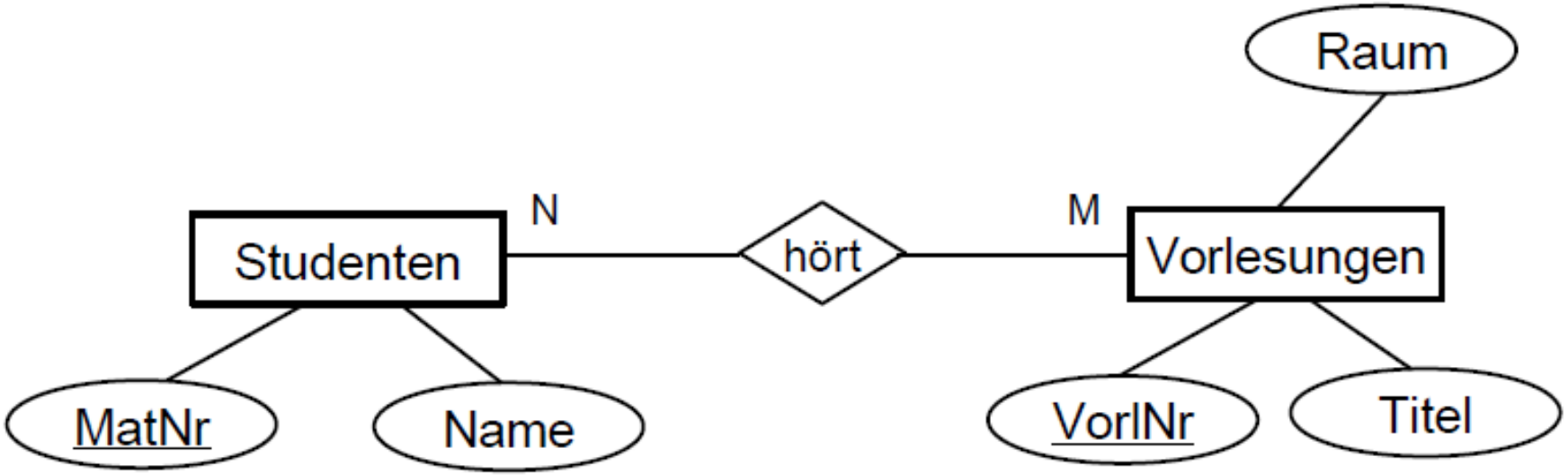
- x .. y mindestens x, maximal y Objekte nehmen an der Beziehung Teil
- 0 .. * optionale Teilnahme an der Beziehung (0 oder mehrere)
- 1 .. * obligatorische Teilnahme an der Beziehung (1 oder mehrere)
- 0 .. 1 es kann nur einen geben oder keinen
- 1 genau 1

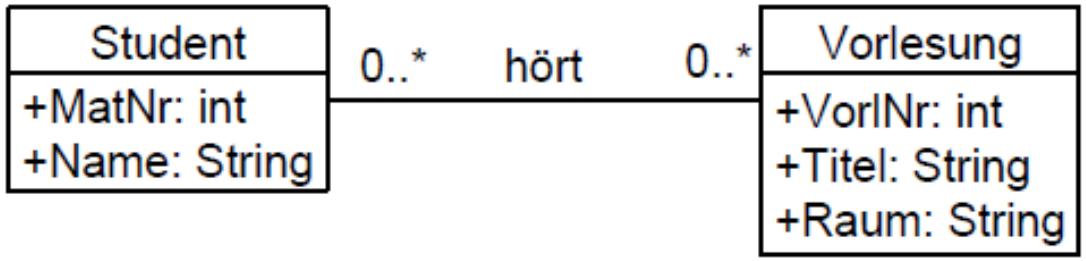
UML Assoziationen



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

ER vs. UML Beispiel

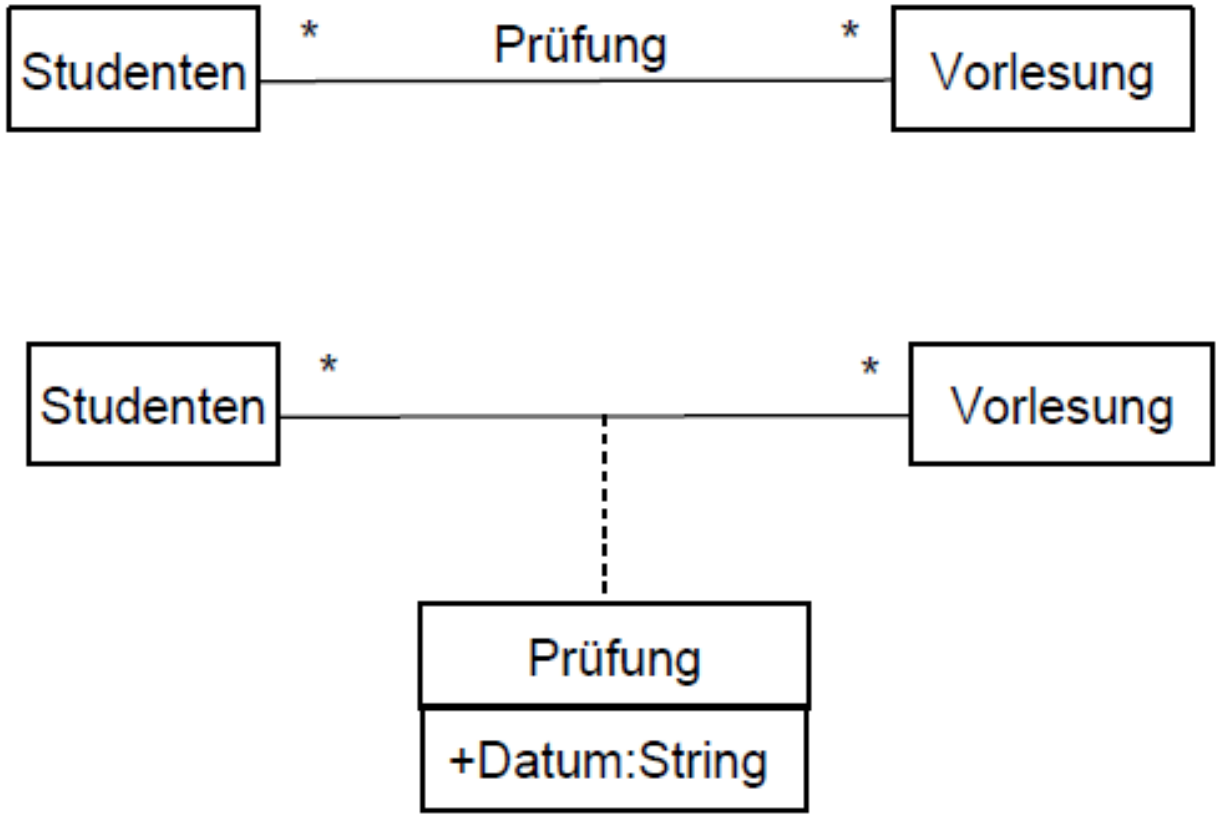




UML Assoziationsklassen

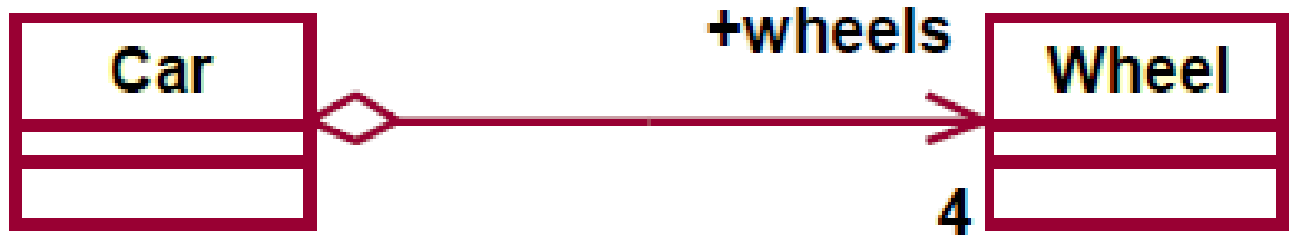
- Für Beziehungen mit eigenen Attributen ist eine Assoziationsklasse notwendig
  - Gestrichelte Linie
  - Name der Assoziationsklasse

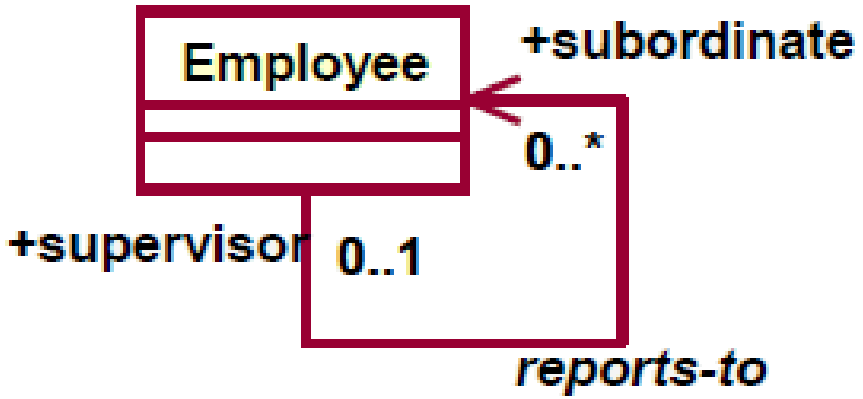
  entspricht der Assoziation



UML part-of (Teil-von) Beziehungen

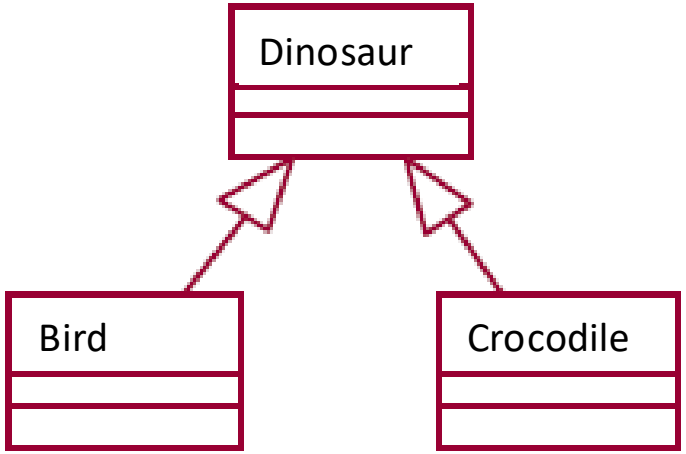
- Aggregation
  - part-of Beziehung zwischen Subkomponente und Superkomponente
- Komposition
  - Teil-Objekt gehört genau zu einem Aggregatobjekt
- Reflexive Assoziation

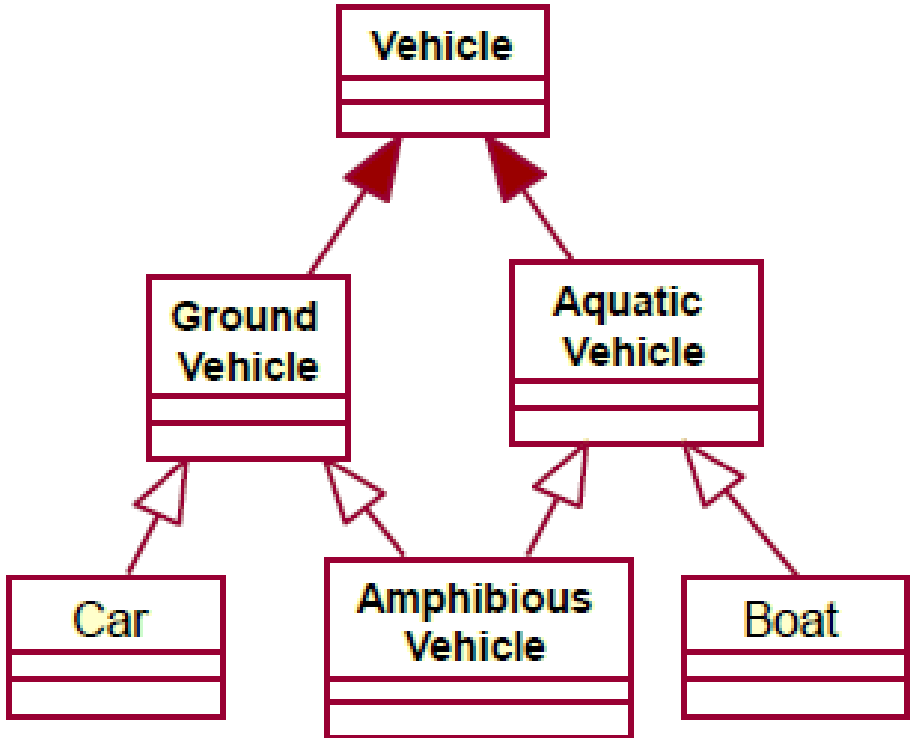




UML is-a Beziehung - Vererbung

- 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)



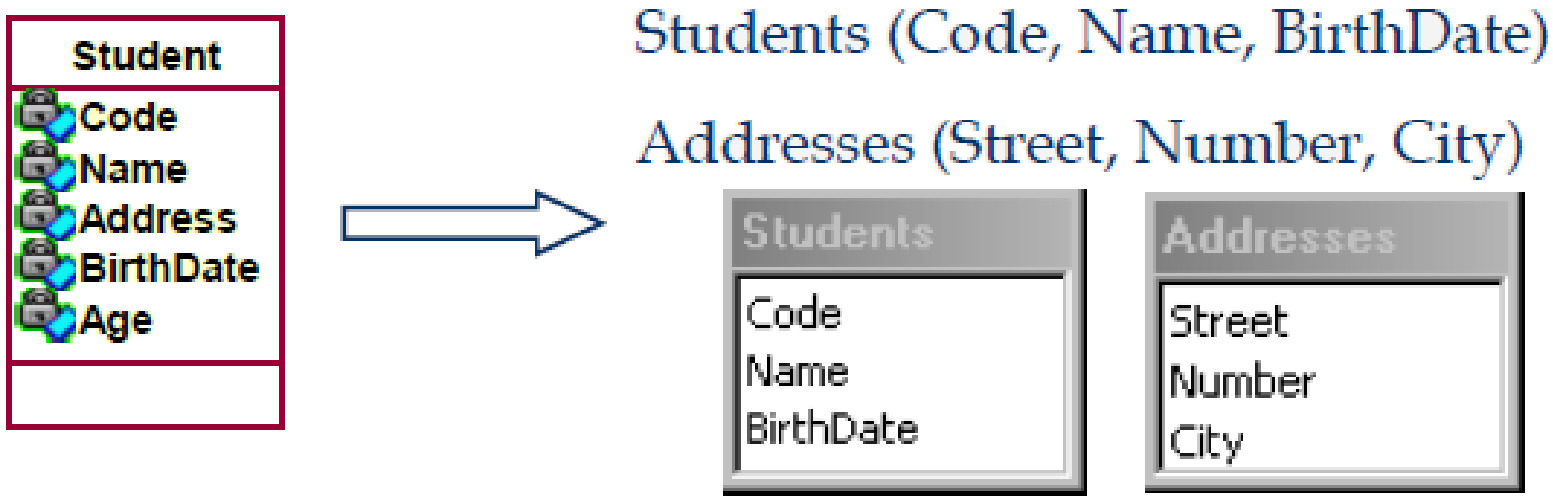


Konzeptuelles Schema – relationelles Datenmodell

- 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 ⇒ zu viele Joins in den Abfragen
  - **Fehlende Tabellen** – M:N Beziehungen brauchen eine dritte Tabelle
  - **Falsche Modellierung der Vererbung**
  - **Denormalisierung der Daten** – manche Daten kommen in vielen Tabellen vor

Transformation der Klassen in Tabellen

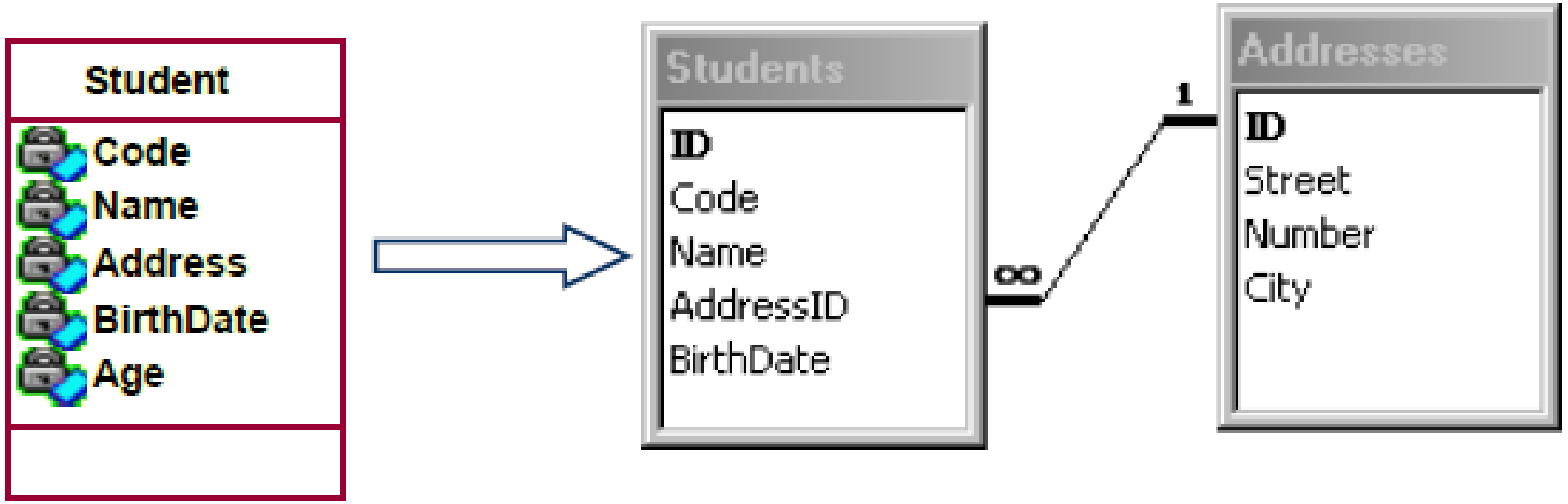
- Name der Tabelle = Plural von dem Klassennamen
- Alle einfache (atomare) Attribute werden Attribute in der Tabelle
- Zusammengestezte Attribute werden als neue Tabelle modelliert
- Abgeleitete Attribute werden nicht in Tabellen gespeichert



Transformation der Klassen in Tabellen

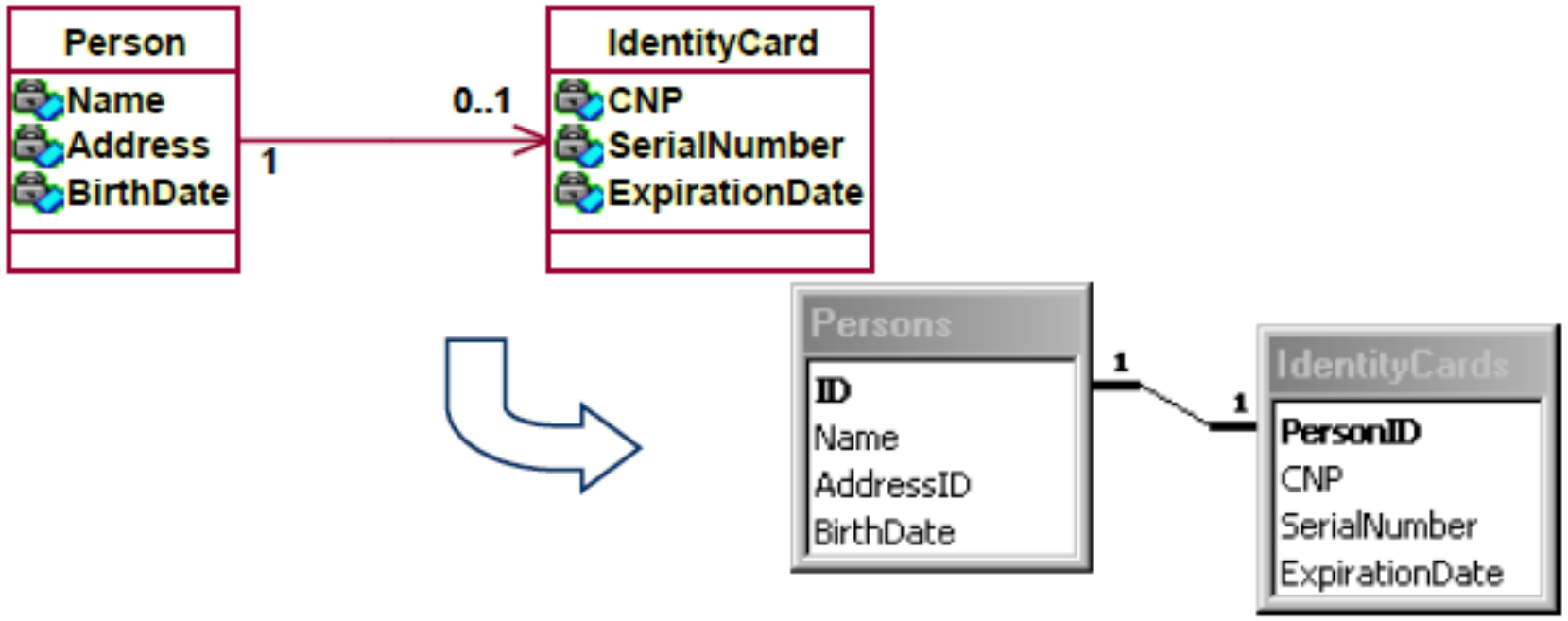
- Ersatzschlüssel (surrogate keys) – Schlüssel, die nicht aus der Domäne des Problems kommen (z.B. IDs)
- In UML gibt es keine Schlüssel
- Eine gute Methode: wenn möglich kann man automatisch generierte Schlüssel (von DBMS) benutzen (auto-increment IDs)
  - Einfach zu verwalten (wird vom DBMS gemacht)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

Transformation der Klassen in Tabellen



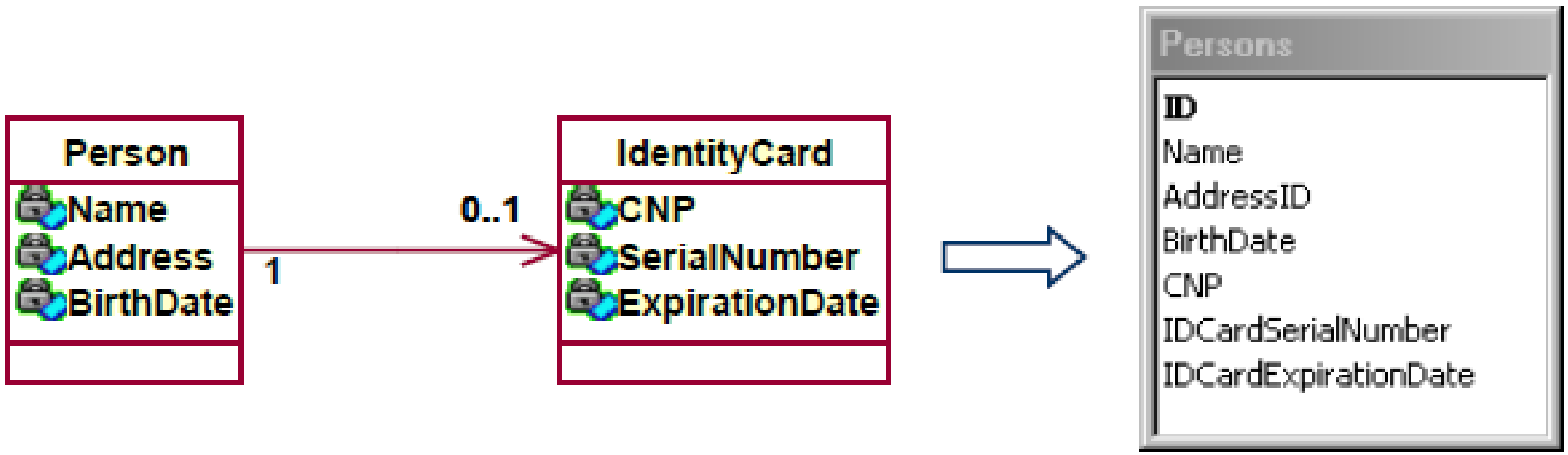
Transformation der Assoziationsbeziehungen

- **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 Assoziationsbeziehungen

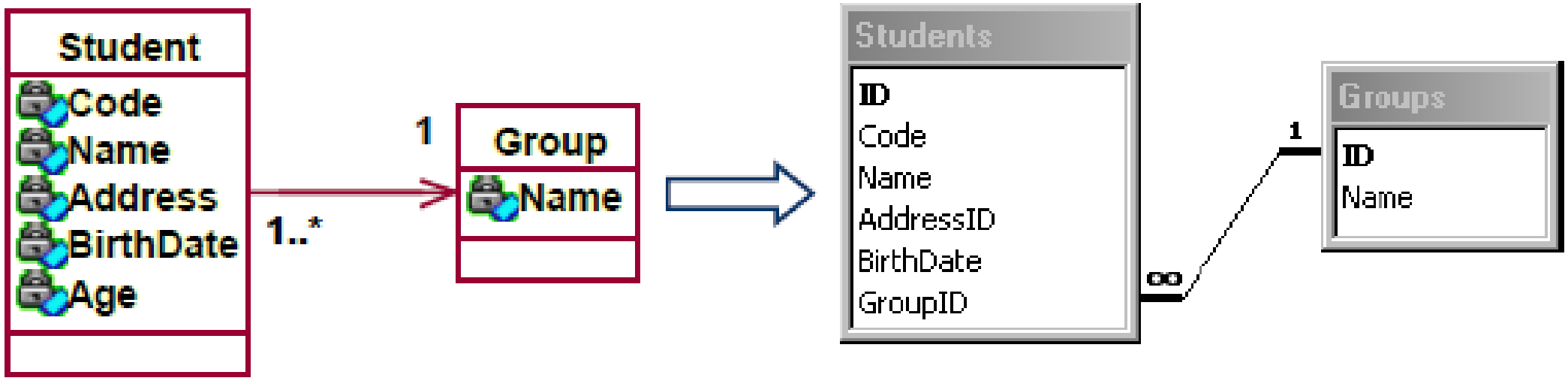
- **1 : 1**
  - Meistens erstellt man eine einzige Tabelle, welche die Attribute beider Klassen enthält
  - Diese Lösung kann man auch für 1 : 0..1 benutzen, wenn es wenige Fälle gibt, in denen Objekte der ersten Klasse mit keinem Objekt der zweiten Klasse in Beziehung sind (wenige Null-Werte)



Transformation der Assoziationsbeziehungen

- **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

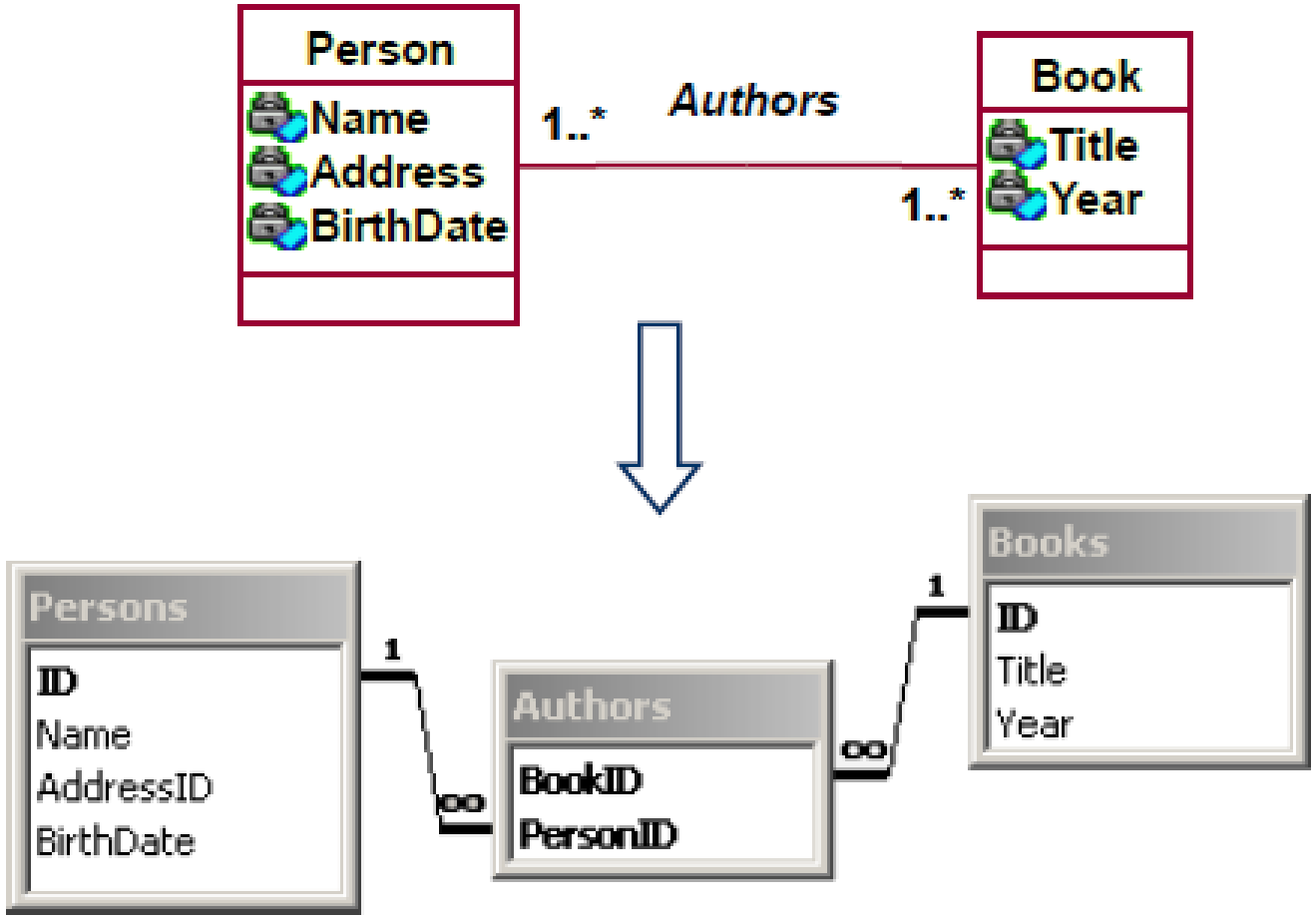
  (anders gesagt: ein Fremdschlüssel in der „M“ Tabelle verweist auf den Primärschlüssel der „1“ Tabelle)



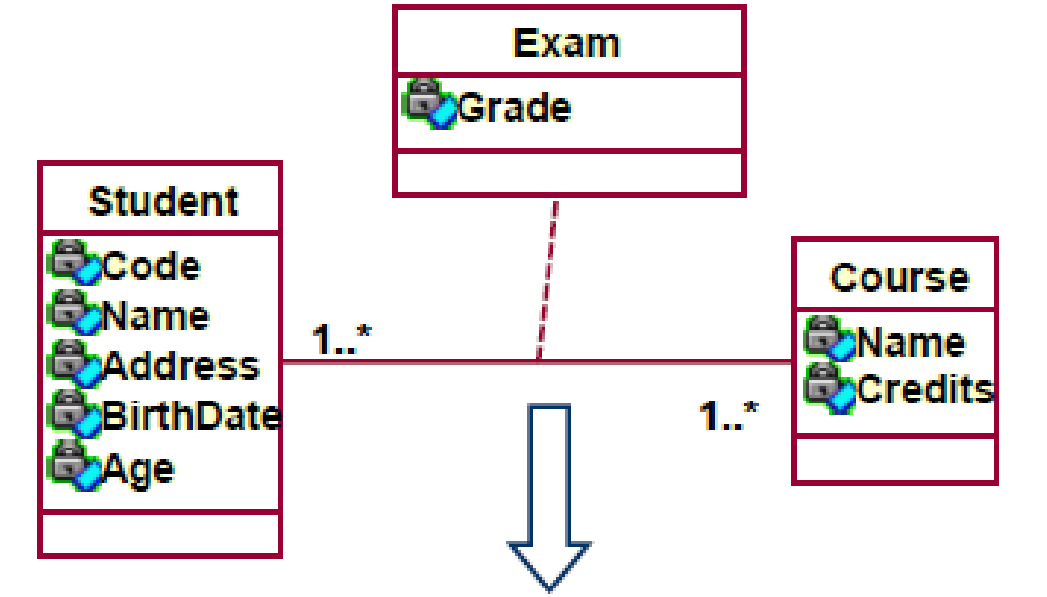
Transformation der Assoziation Beziehungen

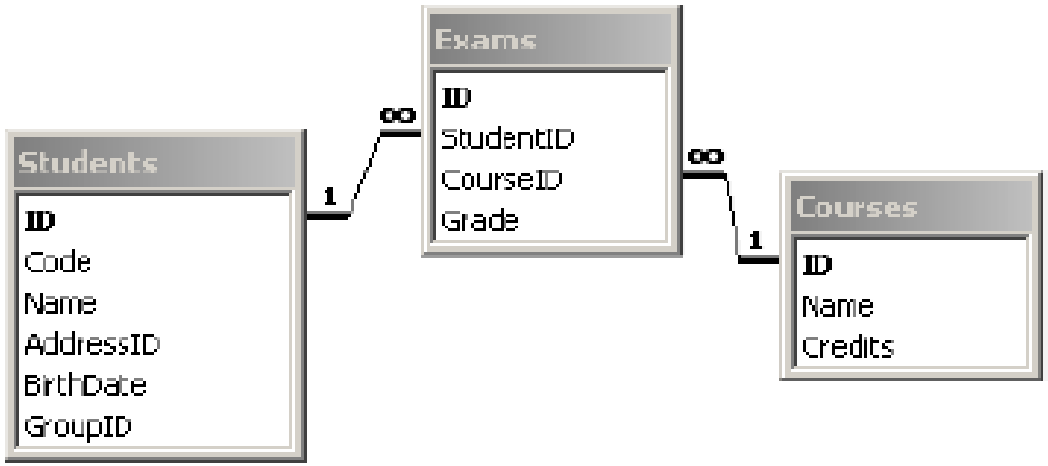
- **M : N (1 .. * : 1 .. *)**
  - Man erstellt eine Tabelle für jede Klasse aus der Assoziation
  - Man erstellt eine zusätzliche Tabelle (cross table / Durchschnittstabelle)
  - Die Primärschlüssel der ursprünglichen Tabellen werden als Fremdschlüssel in dem Cross Table definiert
  - Der Primärschlüssel 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

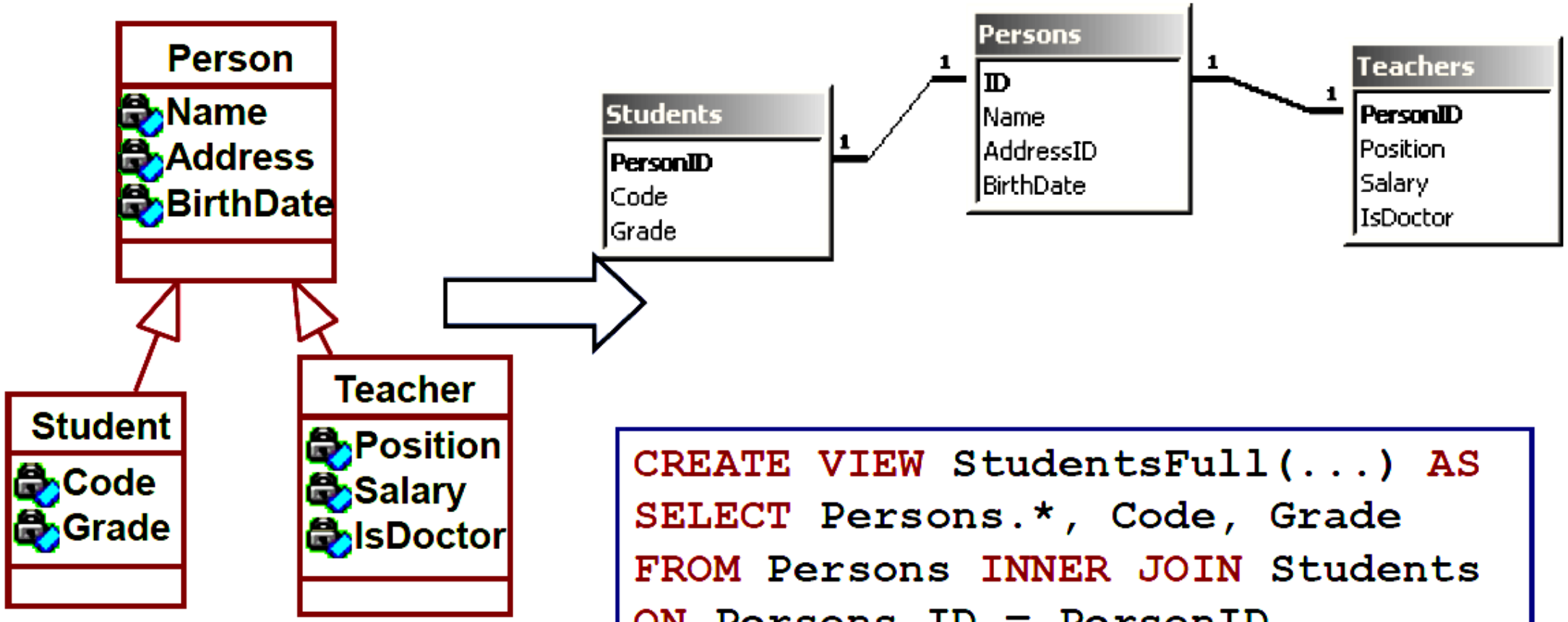




Transformation der Vererbung

- 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

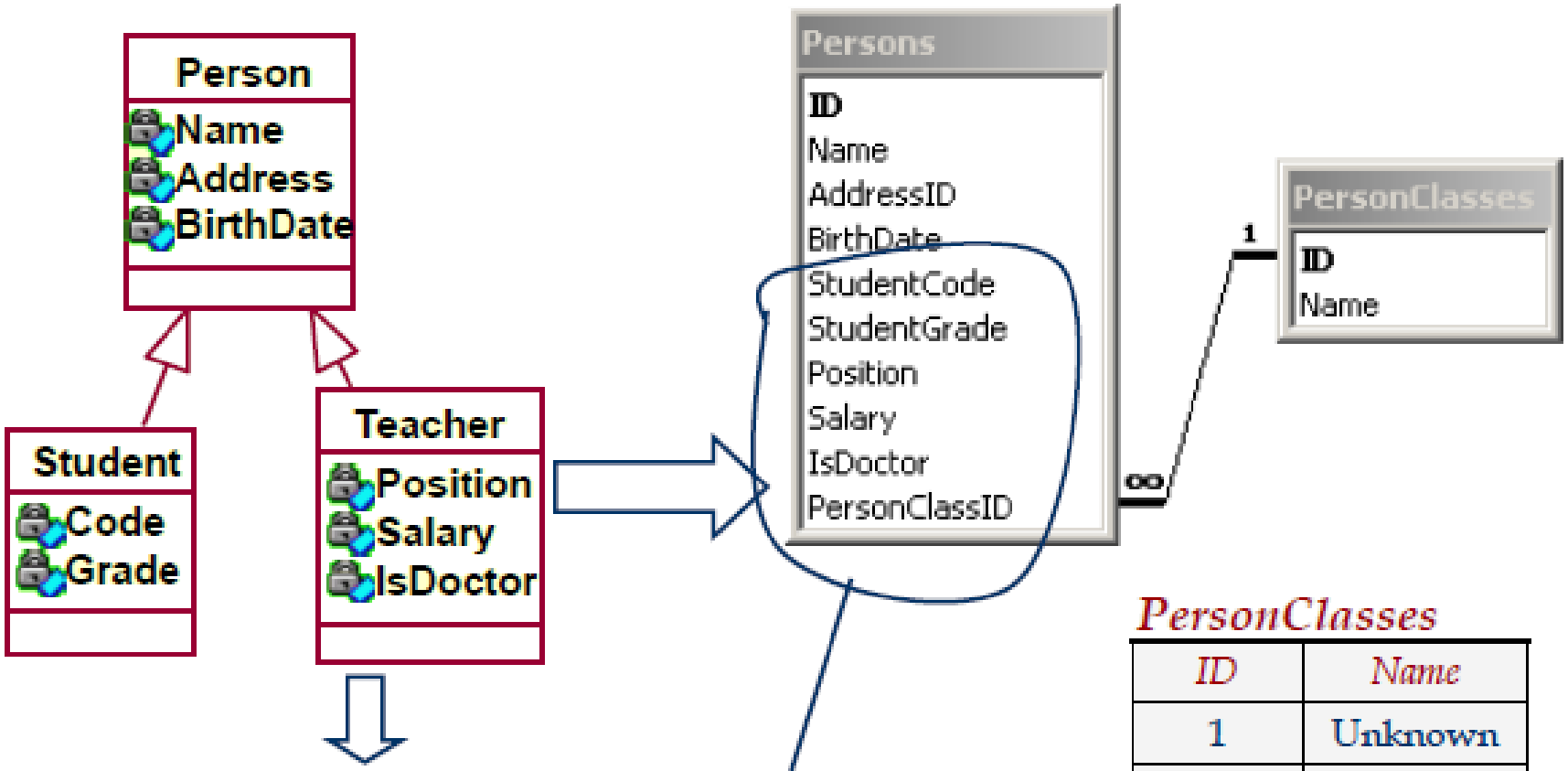



```
CREATE VIEW StudentsFull(...) AS
SELECT Persons.*, Code, Grade
FROM Persons INNER JOIN Students
ON Persons.ID = PersonID
```

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 ⇒ man muss die Struktur der Datenbank ändern (neue Attribute)
  - Ein Tupel hat eine größere „Länge“(es können Nullwerte vorkommen) ⇒ kann die Effizienz beeinflussen

Transformation der Vererbung

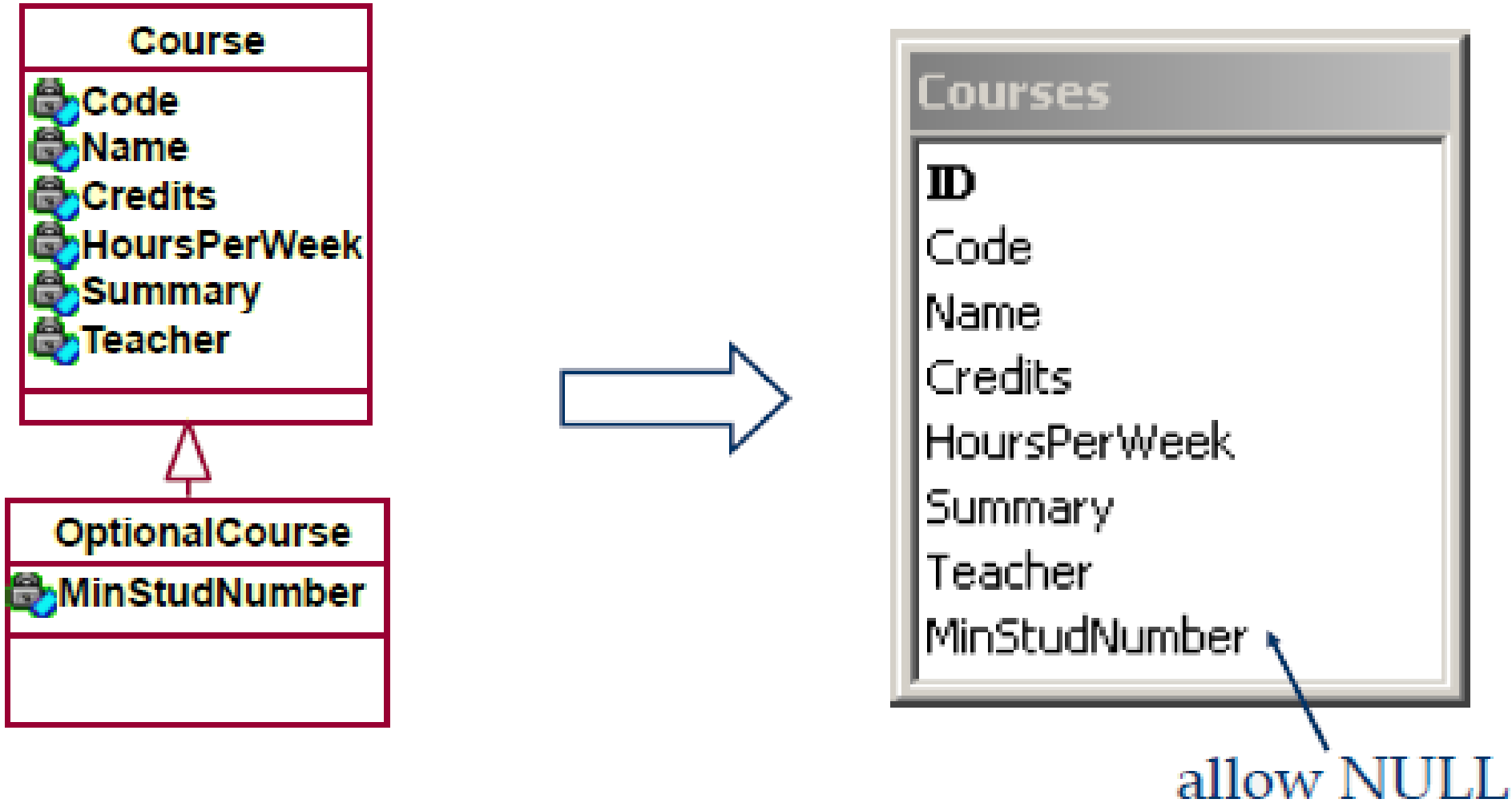



```
CREATE VIEW Students(...) AS
SELECT ID,Name,AddressID,
BirthDate,Code,Grade FROM Persons
WHERE PersonClassID = 2
```

PersonClasses

ID	*Name*
1	Unknown
2	Student
3	Teacher

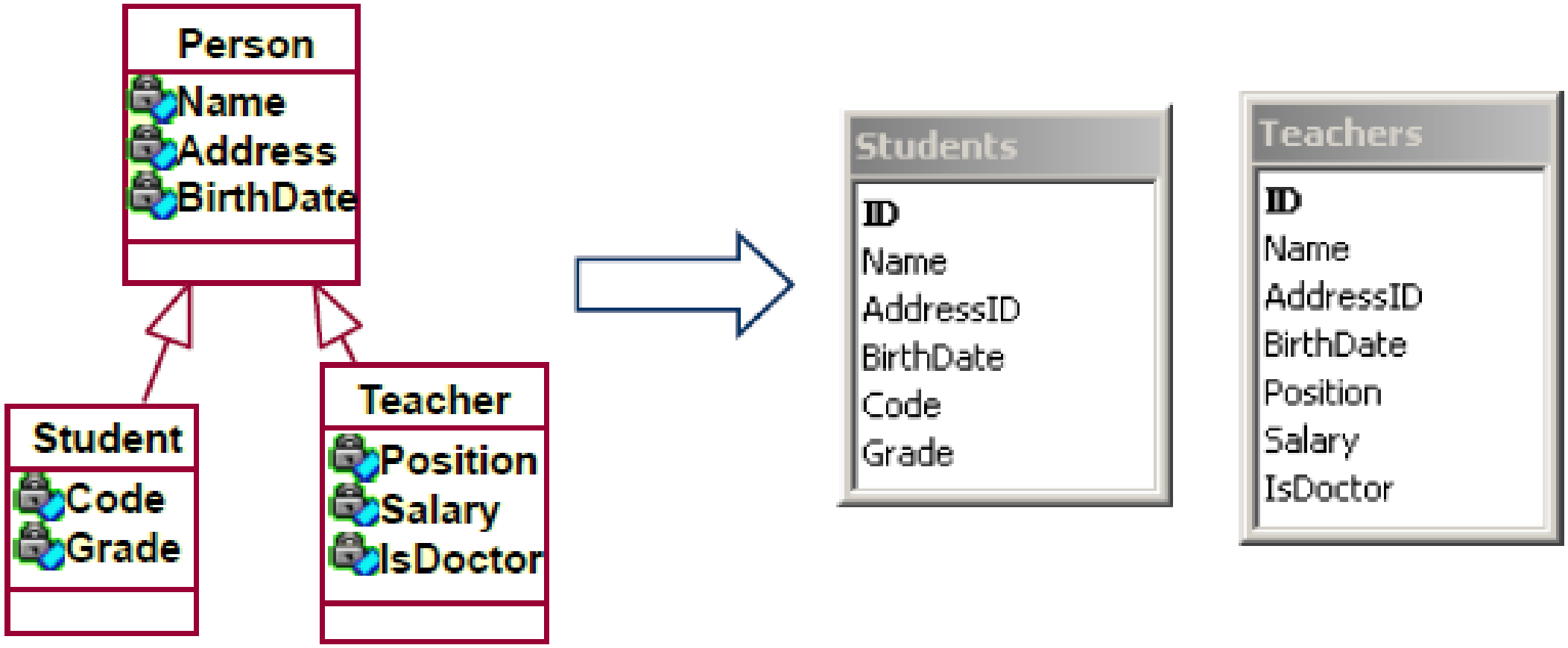
Transformation der Vererbung



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

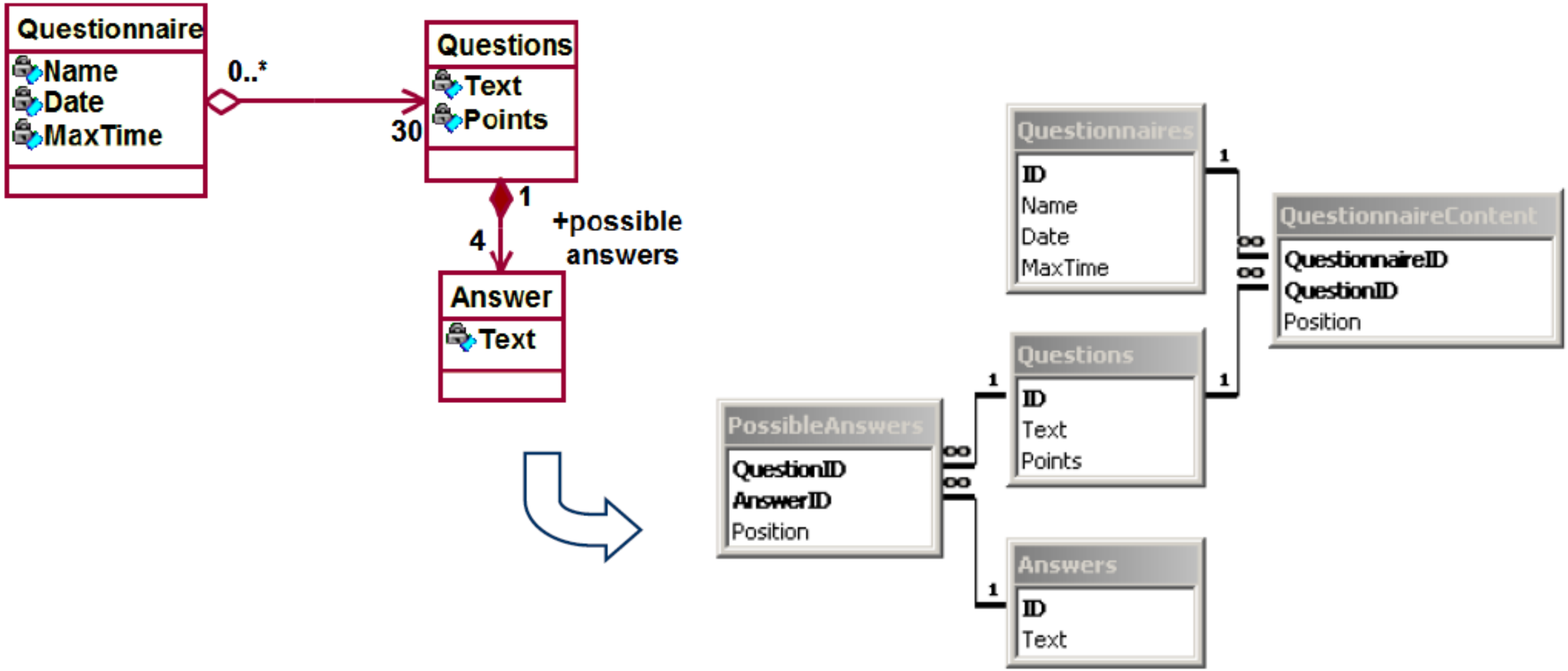
- **Welche Methode ist besser?**

  - 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 Unterklassen, dann dann wählt man am besten die **3. Methode**
  - Wenn die Unterklassenwenige Tupeln haben, dann wählt man am besten die **2. Methode**

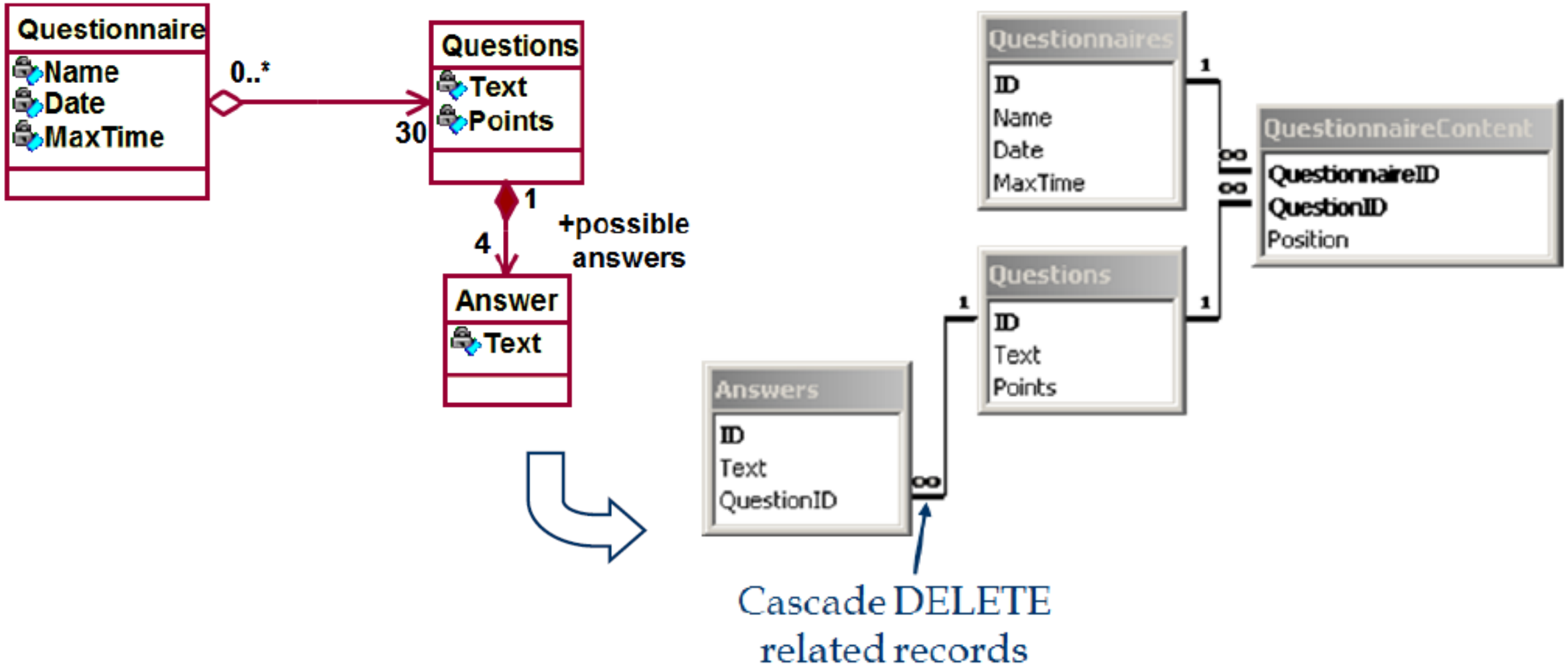
Transformation der Aggregation und Komposition

- 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 „Teilen“ aus einem „Ganzen“ ⇒ genau dieselbe Anzahl von Fremdschlüssel in der Tabelle „Ganzen“

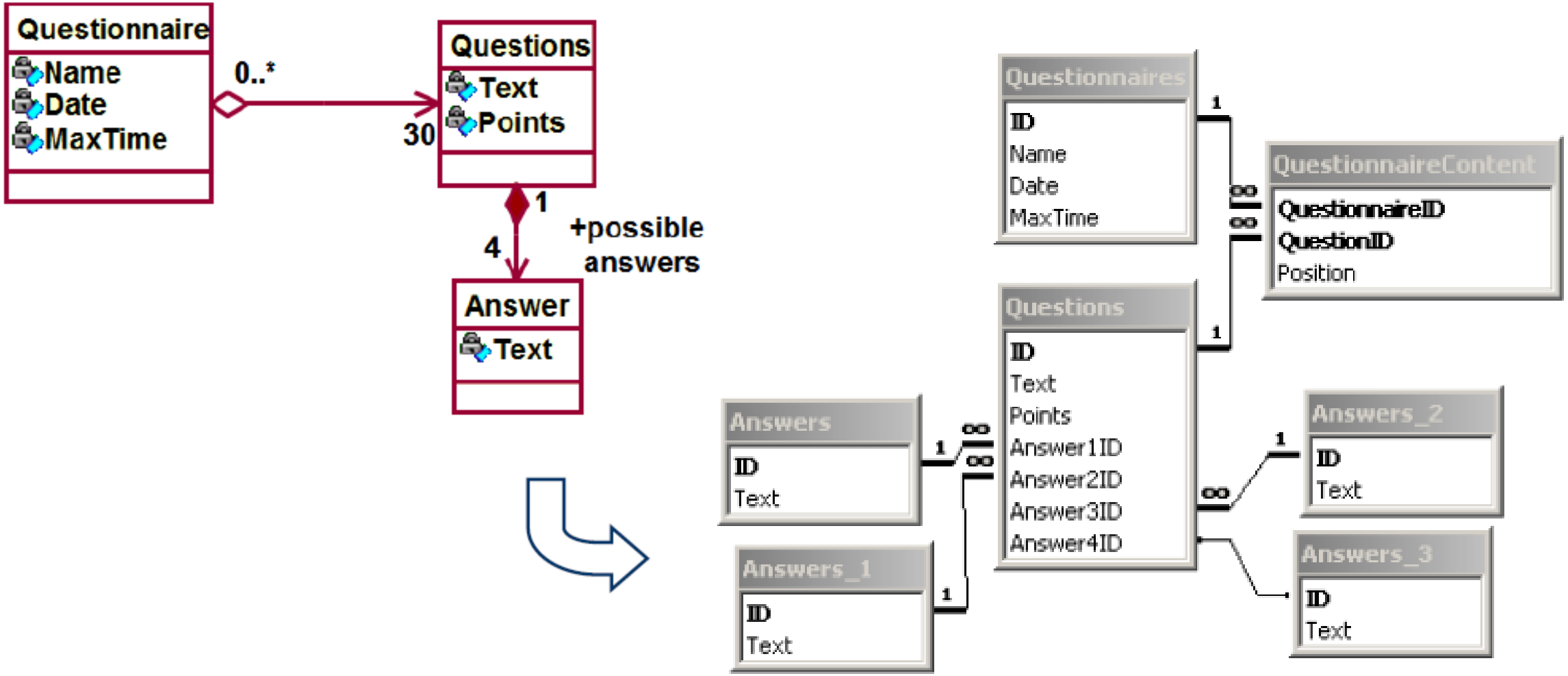
Transformation der Aggregation und Komposition



Transformation der Aggregation und Komposition

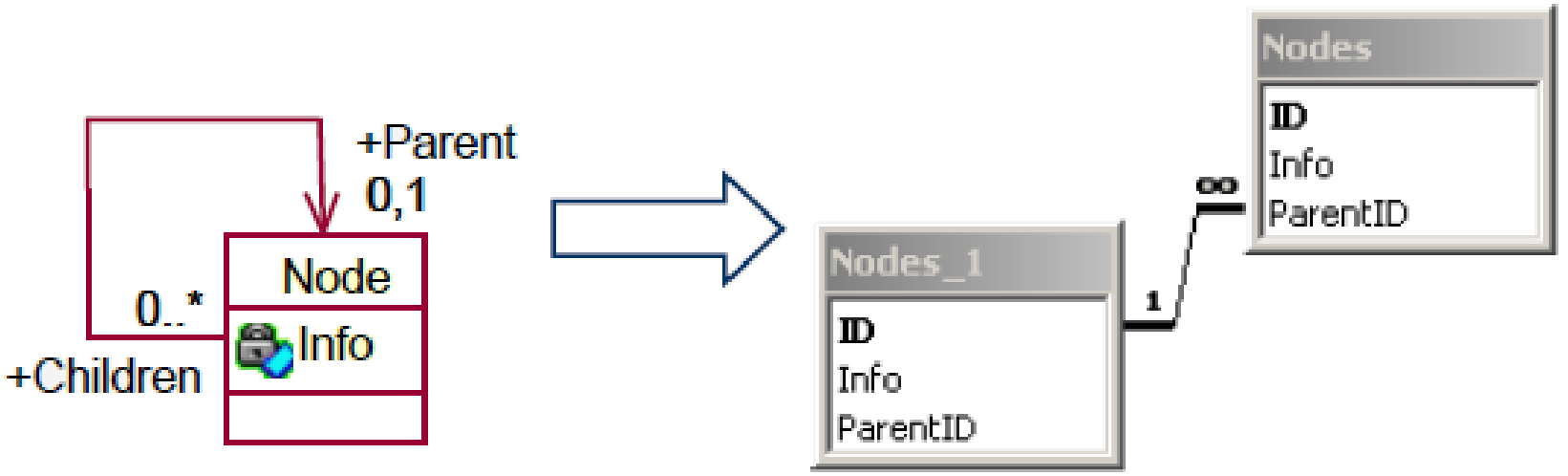


Transformation der Aggregation und Komposition



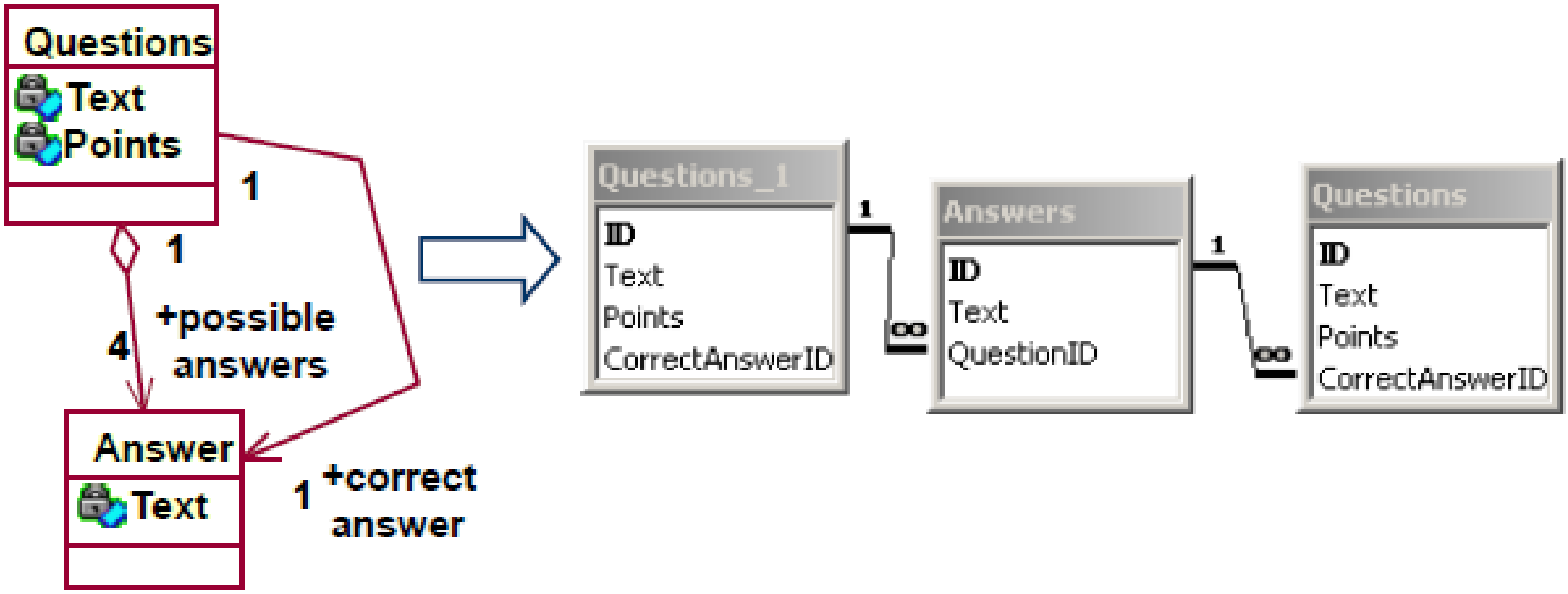
Transformation der Reflexiven Assoziation

- 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



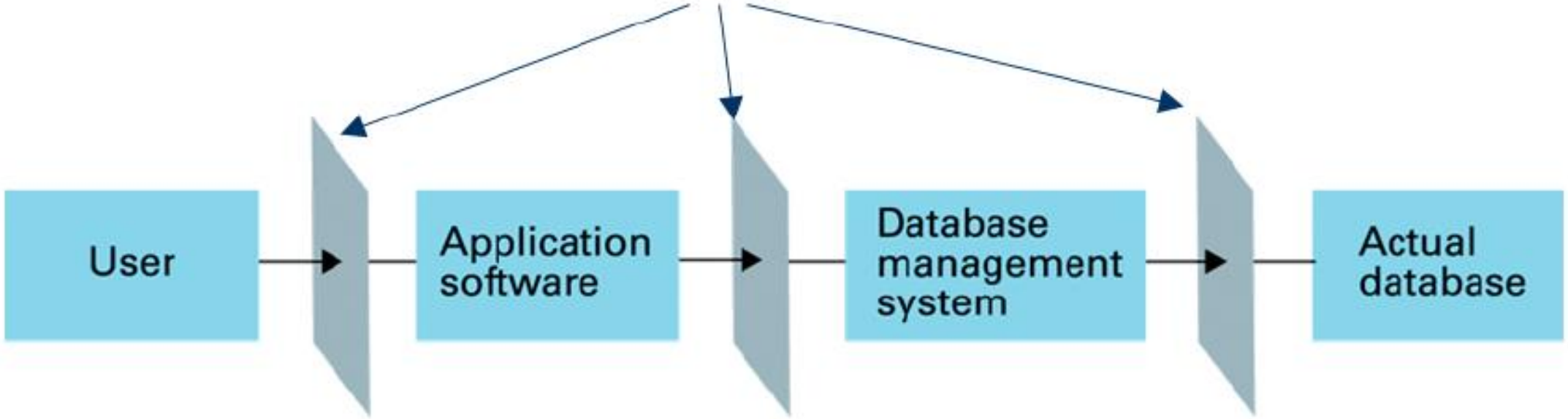
Transformation der Reflexiven Assoziation

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



Abstraktionsschichten.
Das Relationale Datenmodell

Verschiedene Abstraktionsebene



Data in Beziehung zur Application

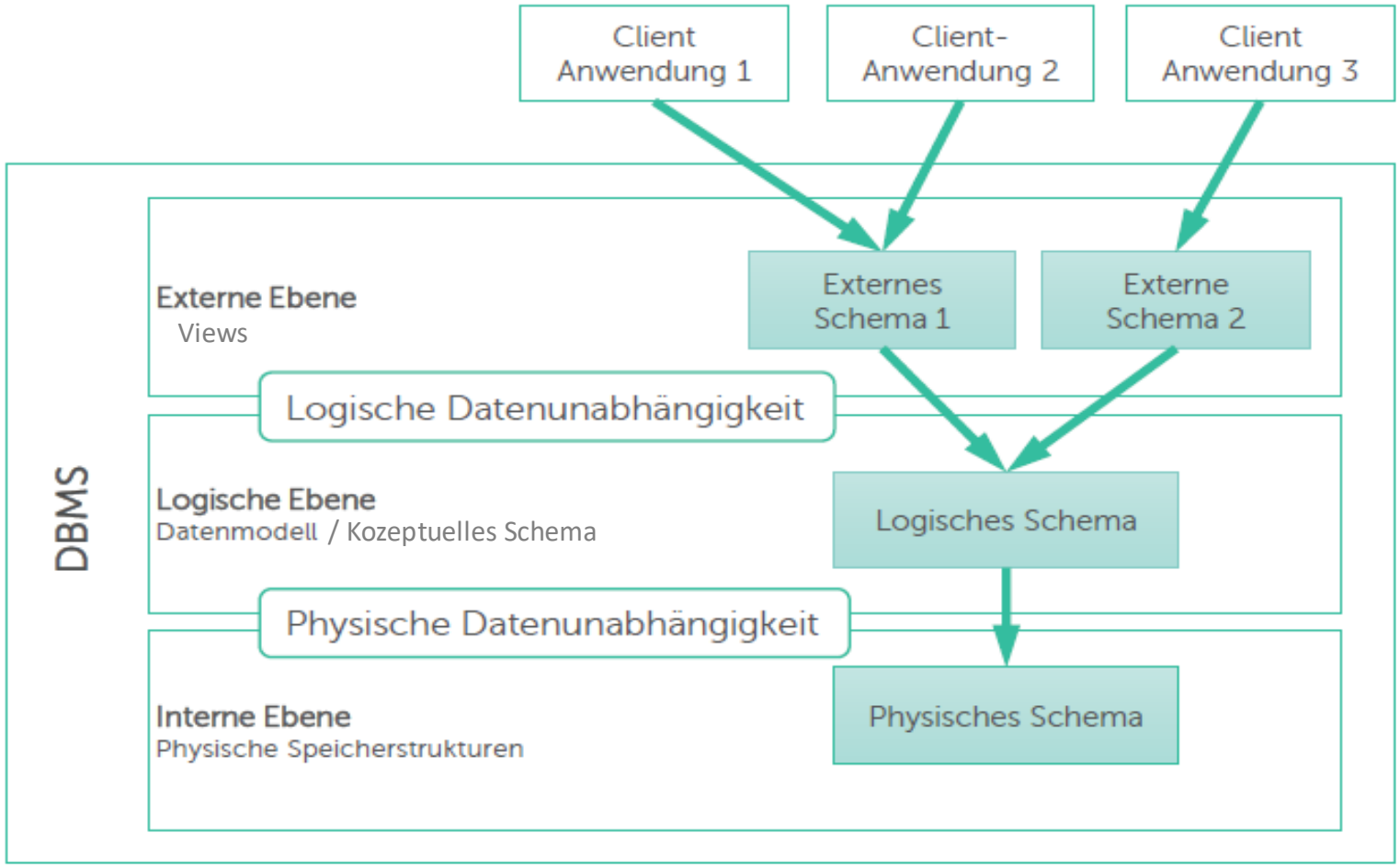
Data in Beziehung zur Datenmodell

Data in Beziehung zur physischen Darstellung

Datenunabhängigkeit

- Ziel: die Datendefinition in einer Schicht zu ändern, ohne dabei die Definition der Daten in der darüber liegenden Schicht zu beeinflussen
- Logische Datenstrukturen unabhängig von physischen Datenstrukturen
- Robustheit der Anwendungen gegenüber Änderungen

Abstraktionsebenen eines DBMS
ANSI-SPARC-Architektur



3-Schichten nach ANSI-SPARC

- Externe Ebene:
  - Anwendungs-spezifische Sichten/Views (Ausschnitte aus dem Datenmodell)
  - Ebene der Anwendungen und Verwendung der Daten
  - Beschreibt wie der Benutzer die Daten sieht
- Logische Ebene:
  - Definition der logischen Datenstrukturen: die Beziehungen zwischen den Daten unabhängig von der physischen Repräsentation
- Interne/Physische Ebene:
  - Definition des physischen Schemas (wie die Daten gespeichert werden)
  - Beschreibt die Dateien und Indexstrukturen die benutzt werden
  - Es geht um Leistungsfähigkeit der Datenbankanwendungen

Beispiel: **Universität** Datenbank

- Logische Ebene/Konzeptuelles Schema:
  - **Studenten**(sid:string, name:string, email:string, age:integer, gruppe:integer)
  - **Vorlesung**(vid:string, vname:string, ects:integer)
  - **Klausur**(sid:string,vid:string,note:integer)
- Physische Ebene:
  - Relationen werden als ungeordnete Dateien gespeichert
  - Indexstruktur auf die erste Spalte (sid) der Tabelle **Studenten**
- Externe Ebene/View:
  - Vorlesungsinfo(vid:string, enrollment:integer)

Physische Datenunabhängigkeit

- Änderungen an der Art der Datenspeicherung und den Zugriffstechniken haben keinen Einfluss auf Anwendungsprogramme
- Programme sind von interner Datenorganisation unabhängig
- Physische Datenunabhängigkeit wird durch relationale DBMS weitestgehend hergestellt
- Beispiele:
  - Verschiebung von Datenbank auf ein anderes Storage-System
  - Partitionierung
  - Indexstrukturen für performante Zugriffe

Logische Datenunabhängigkeit

- Ermöglicht durch externe Ebene
- Änderungen am logischen Schema haben nur geringe Auswirkungen auf die Anwendung
- Ist nur begrenzt möglich, denn Anwendungen basieren sich auf dem logischen Schema
- Sichtenkonzepz (Views) ermöglicht Verbergen von Details des logischen Datenmodells
- Beispiel: nach Umbenennung einer Tabelle oder eines Attributes kann die alte Datenstruktur virtuell über eine Sicht bereitgestellt werden

Queries (Abfragen) im DBMS

- Mögliche Fragen für die Universität Datenbank:
  - Welcher ist der Name des Studenten mit sid=2310?
  - Wie viele Studenten haben sich für die Vorlesung DB angemeldet?
  - Wie viele Studenten haben in der DB Klausur mehr als 9 gekriegt?
- Fragen, die sich auf Daten gespeichert im DBMS beziehen, sind Datenbank-Abfragen
- DBMS hat bestimmte Datenbanksprachen zum Abfragen und Manipulieren der Daten (Query Language)

Datenbanksprache

- Formale Sprache die folgende Komponenten hat:
  - Datenbeschreibungssprache / Data Definition Language (DDL)
    - Befehele zur Definition des Datenbankschemas (Anlegen, Ändern und Löschen von Datenstrukturen)
    - Constraint Definition Language (CDL) – beschreibt Integritätsregeln, die von den Instanzen der Datenbank erfüllt werden sollen
    - Storage Definition Language (SDL) – um das Layout des physischen Schema zu beeinflussen
  - Datenmanipulationssprache / Data Manipulation Language (DML)
    - Befehle zur Datenmanipulation (Abfragen, Einfügen, Ändern oder Löschen von Nutzdaten)
    - Procedural DML (wie) vs. Declarative DML (was)

Abfragesprachen für relationale Datenbanken

- SQL (Structured Query Language)
  - SELECT name FROM studenten WHERE age>20
- Relationale Algebra
  - $\pi_{name}(\sigma_{age>20}(Studenten))$
- Domänenkalkül (Domain Calculus)
  - {<X>|∃V ∃Y ∃Z ∃T: Studenten(V,X,Y,Z,T) ∧ Z>20}
- Tupelkalkül (Tuple Calculus)
  - {X| ∃Y : Y∈Studenten ∧ Y.age>20 ∧ X.name=Y.name}

Relationale Datenbankabfragesprache/ Relational Query Language

- Vorteil des relationalen Modells:
  - unterstützt einfache Datenbankabfragesprachen

- Abfragen können intuitiv formuliert werden und der DBMS ist zuständig für die optimale Abfrage Bearbeitung
- Aber: manchmal müssen wir die Abfrage optimal formulieren, um die gewünschte Zeitkomplexität zu erreichen

Structured Query Language (SQL)

- Ist die Sprache die von den meisten relationalen Datenbanken unterstützt wird
- Wurde 1970 von IBM entwickelt
- SQL Standard:
  - SQL 86
  - SQL 89
  - SQL 92 (SQL 2)
  - SQL 2003
  - SQL 2008
  - SQL 2011
  - SQL 2016
  - SQL 2019

SQL Befehle

- Kategorien von SQL Befehlen:
  - Datenbeschreibungssprache / Data Definition Language (DDL)
    - Anlegen, Ändern und Löschen von Relationen oder Views
    - Beschreiben von Integritätsregeln
  - Datenmanipulationssprache / Data Manipulation Language (DML)
    - Einfügen, Ändern oder Löschen von Daten in den Relationen (Tupeln)
    - Abfragen (Queries)
  - Datenüberwachungssprache / Data Control Language (DCL)
    - Befehle für Rechteverwaltung (auf Tabellen und Views) und Transaktionskontrolle