View Integration, Logical Design

Christian Hribernik, Anel Jusic,
Florian Kropfitsch, Daniel Smid

Aufgabe 1

View Integration

Angabe

Integration der Views

- Konflikte identifizieren (und deren Arten)
- Szenarien vorschlagen (and interschema properties)
- Schemas integrieren

View Integration

Mögliche Gründe:

- Erweiterungen
- Große, getrennt entwickelte Datenmodelle
- Organisationsfusionen
- usw.

Arten von Integration:

- One-Step
- Incremental
- Mixed

Für Integration besteht kein formalisiertes Konzept!

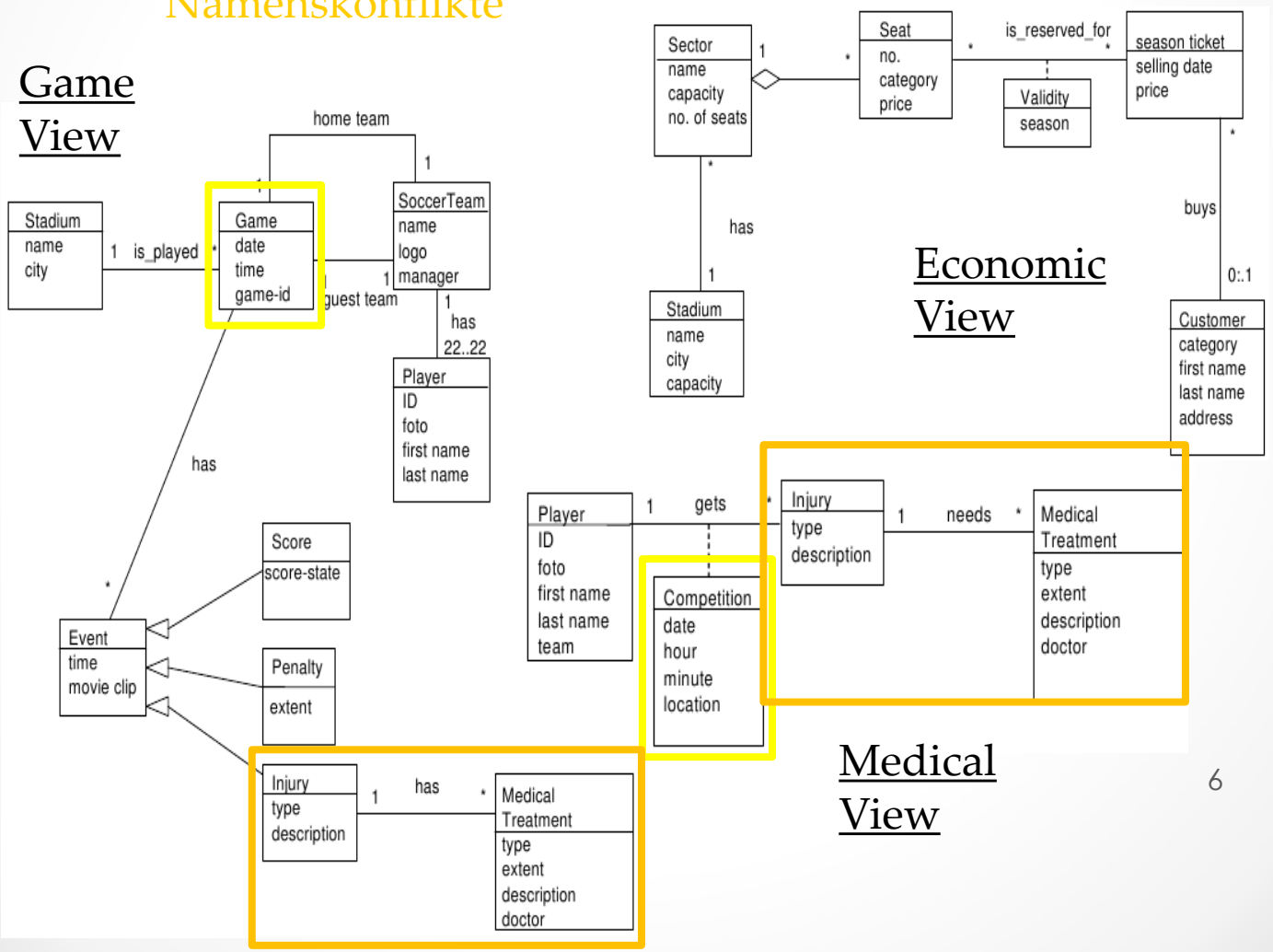
Konflikte

- Namenskonflikte
  - Homonym
  - Synonym
  - Ähnlichkeit / similarity
  - Unstimmigkeit / mismatch
- Strukturelle Konflikte
  - Identisches Konzept
  - Kompatibles Konzept
  - Inkompatibles Konzept

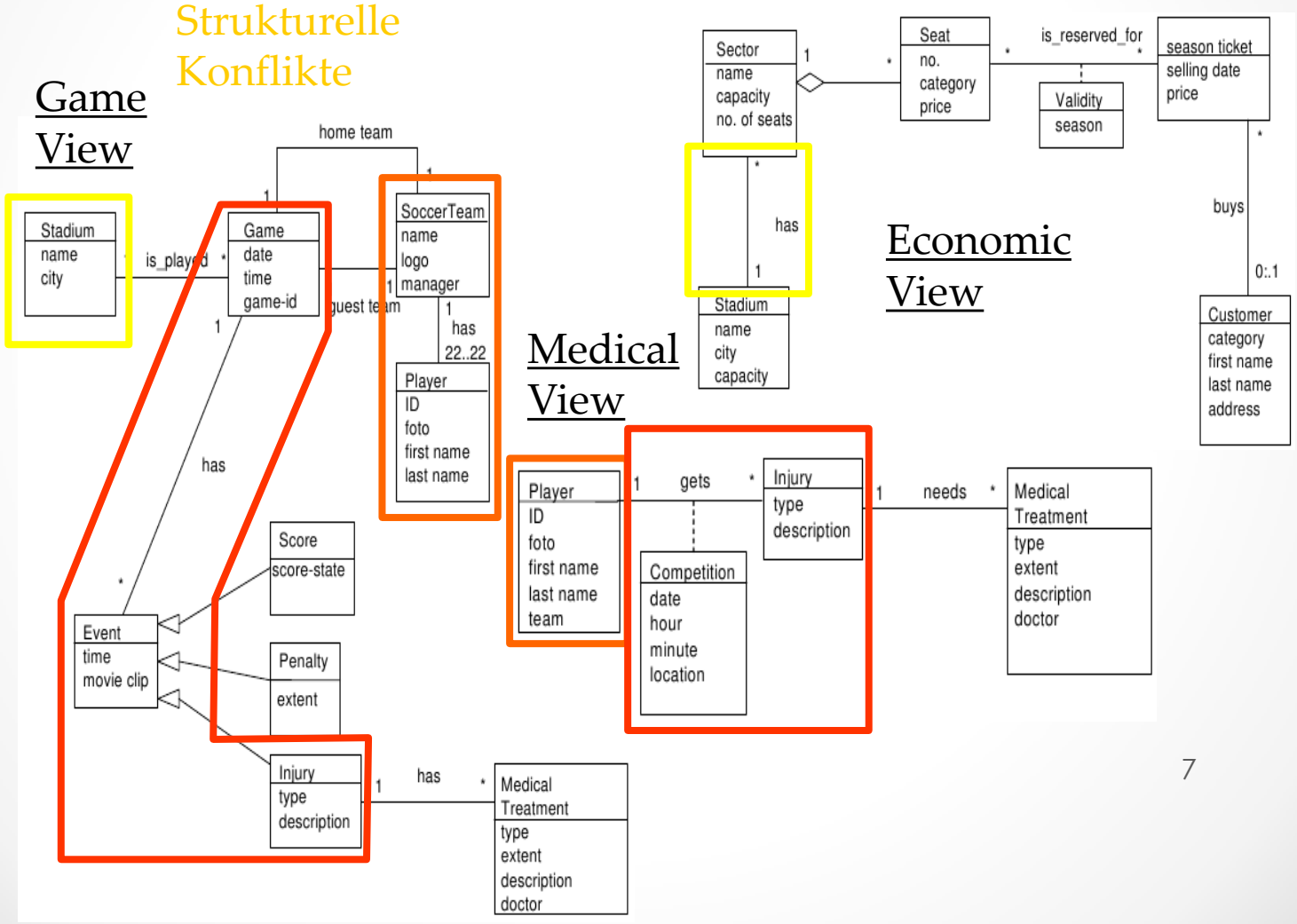
Analysieren & Lösen der Konflikte → Integration der Views

Identifikation von Konflikten I

Namenskonflikte



Identifikation von Konflikten II



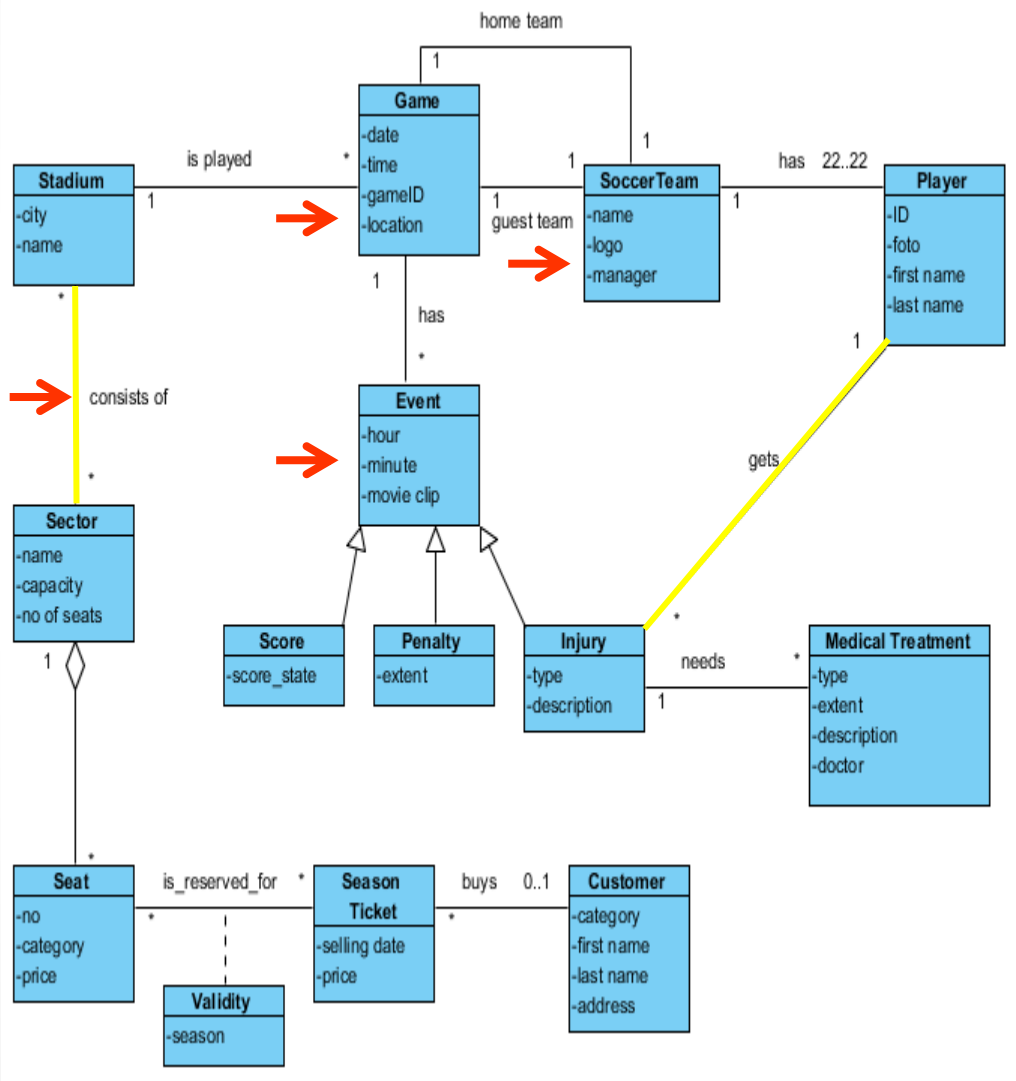
Identifizierte Konflikte

- Namenskonflikte:
  - Game – Competition (Synonym)
  - has – has (Homonym)

- Strukturelle Konflikte:
  - Stadium – Stadium (Mismatch, struktureller Konflikt)
  - Injury & Medical Treatment (Identisches Konzept)
  - Soccer Team & Player – Player (Kompatibles Konzept)
  - Player – Player (Mismatch, strukturell)
  - Game, Event & Injury – Injury & Competition (inkompatibles Konzept)

- Andere:
  - season ticket
  - game_id
  - 0:.1

Integration

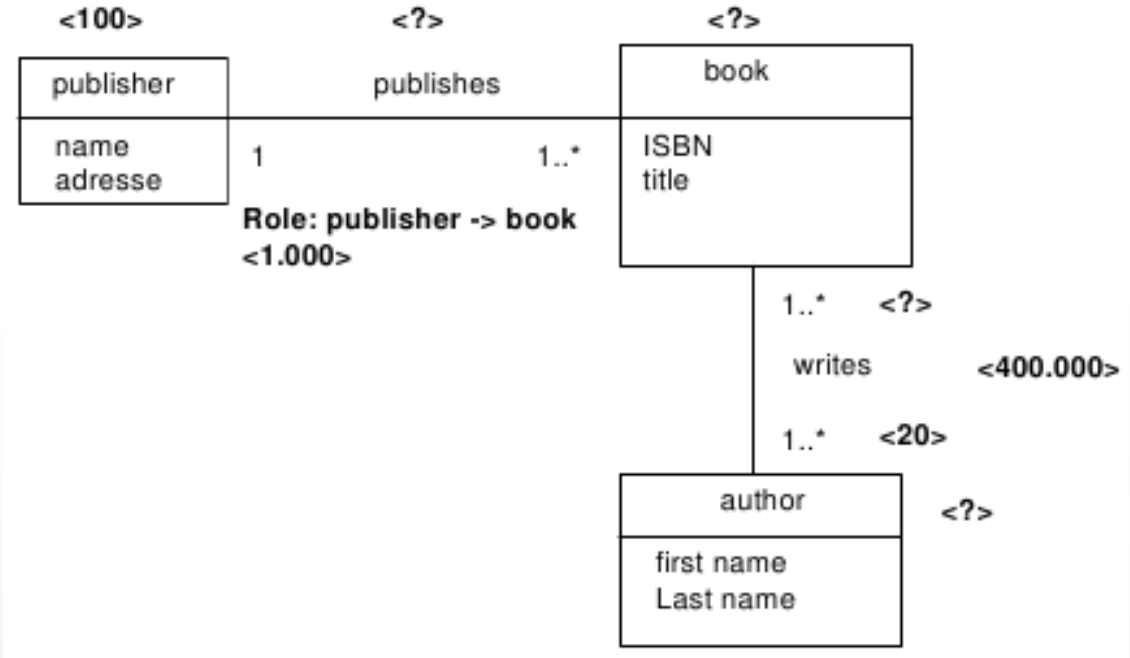


Aufgabe 2

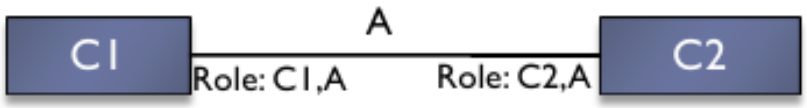
Logical Design (Determination of Quantities)

Angabe

- Fehlende Quantitäten ausrechnen
- Navigation

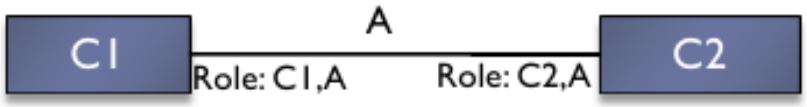


Data-Volume Information I



- N(C) = durchschnittliche Anzahl von Instanzen pro Klasse
- N(A) = durchschnittliche Anzahl von Assoziationen pro Klasse
- N(C,A) = durchschnittliche Kardinalität der Rollen (= durchschnittliche Anzahl der Klasseninstanzen pro Assoziation)

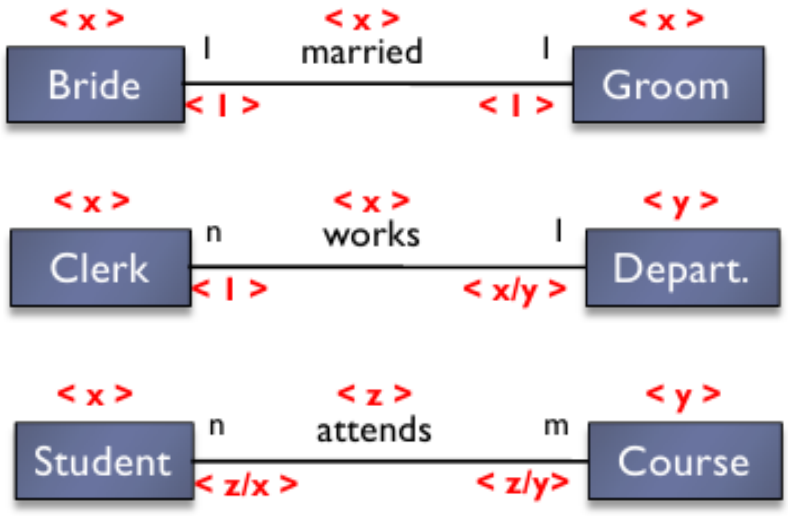
Data-Volume Information II



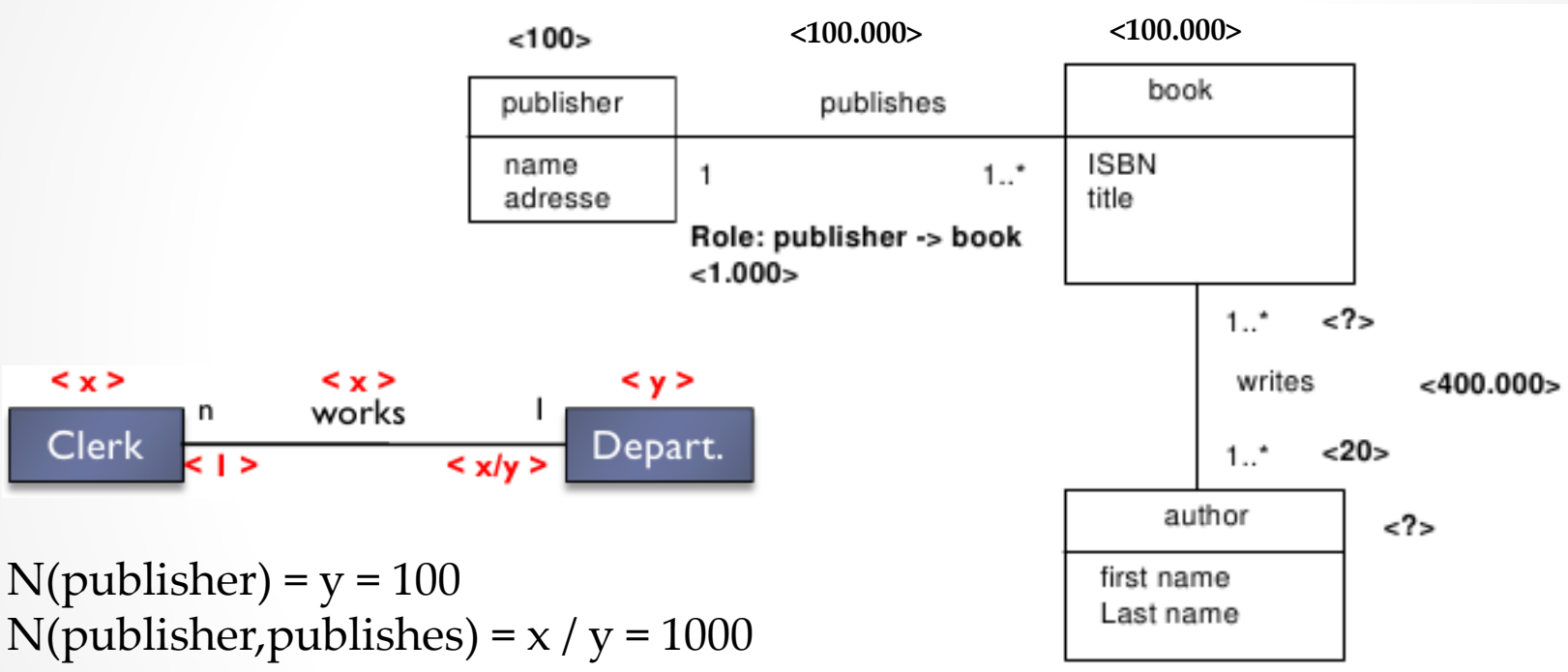
- N(C1) x N(C1,A) = N(A)
- N(C2) x N(C2,A) = N(A)

Data-Volume Information III

- Berechnung der fehlenden Werte
- 1:1 Beziehung
- 1:n Beziehung
- n:m Beziehung



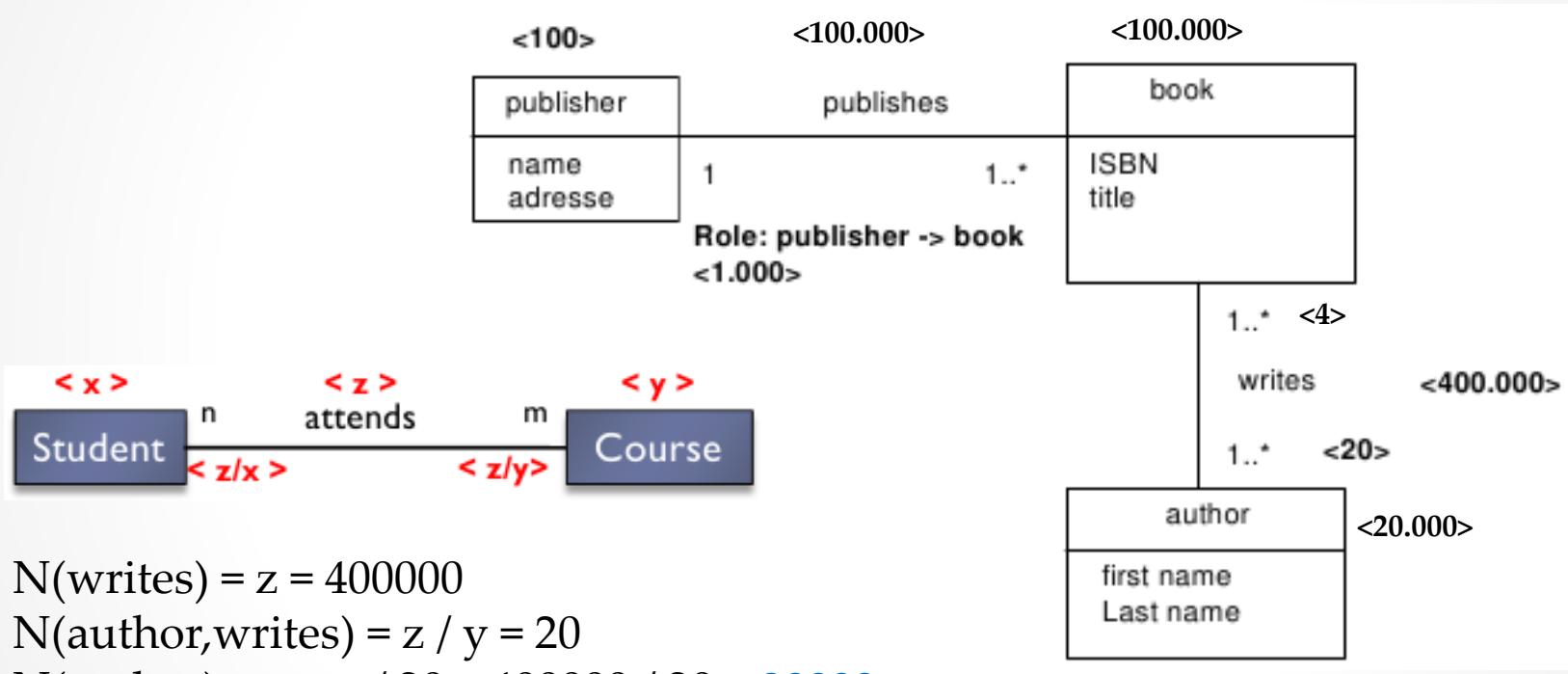
Beispiel



N(publisher) = y = 100
N(publisher,publishes) = x / y = 1000
N(publishes) = x = 1000 * y = 1000 * 100 = **100000**

N(book) = x = **100000**

Beispiel



N(writes) = z = 400000
N(author,writes) = z / y = 20
N(author) = y = z / 20 = 400000 / 20 = **20000**

N(book,writes) = z / x = 400000 / 100000 = **4**

Wie funktioniert Navigation?

- Um Query zu beantworten -> mehrere Tabellen wird zugegriffen -> WICHTIG: Weg durch die Datenbank
- Welche Klassen und Assoziationen sind betroffen
- Welche Attribute werden gelesen/geschrieben
- Welche Attribute sind Suchattribute
- Evaluierung der Kosten einer Datenbankoperation

Aufgabe 3

Logical Design (Derived Attributes)

Angabe a)

- Was ist Redundanz? Wieso sollte man Redundanzen in die Datenbank einfügen wollen?

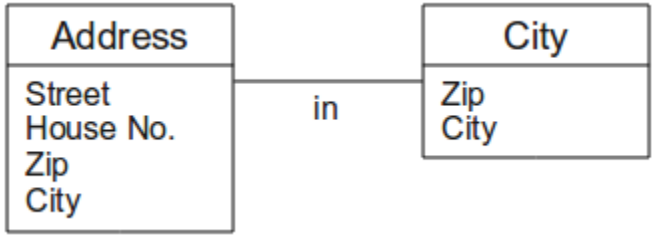
Was ist Redundanz?

- Überfluss von vorhandenen Daten
- Informationen, die mehr als einmal in der Datenbank vorkommen

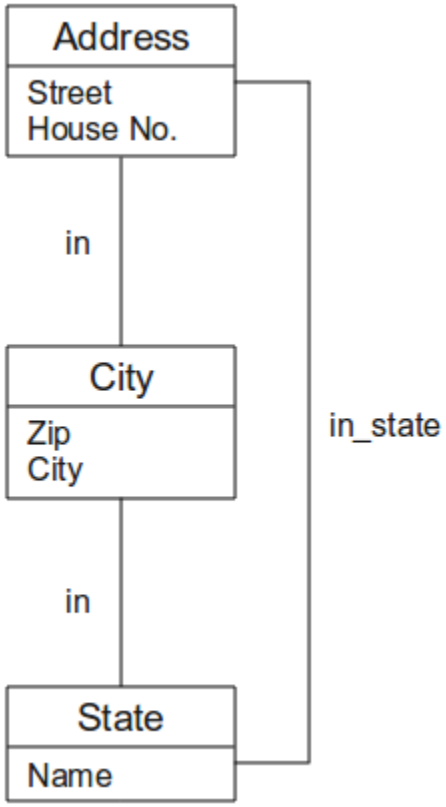
Warum Redundanz einführen? I

- Redundante Attribute
  - Können eingeführt werden, um die Zugriffszeit auf bestimmte Daten zu verkürzen
  - Zusätzliche Updates
  - Zusätzliche Überprüfungen auf Konsistenz
  - Zusätzlicher Speicherplatz wird benötigt

Warum Redundanz einführen? II



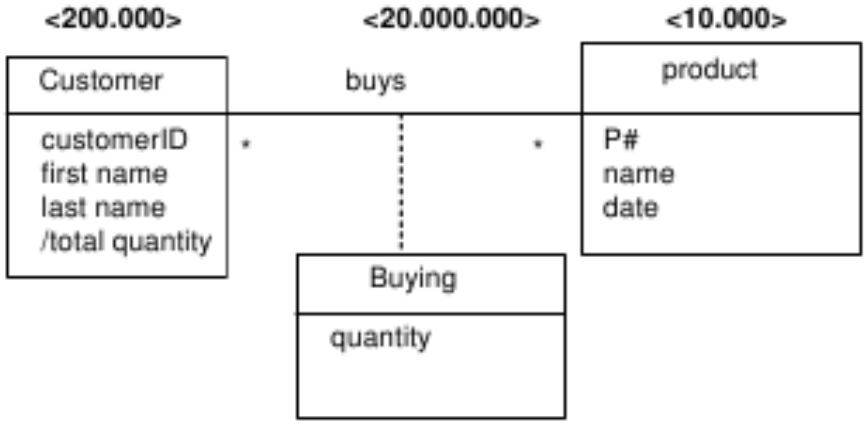
Warum Redundanz einführen? III



Entscheidung Redundanz

- Identifiziere Operationen
  - Operationen, die auf redundante Daten zugreifen
- Kalkuliere die Anzahl der Zugriffe für beide Varianten
  - Variante mit / ohne redundante Daten
- Entscheide
  - Vergleiche die Anzahl der Zugriffe
  - Variante mit der niedrigeren Anzahl der Zugriffe

Angabe b)


<200.000>
<20.000.000>
<10.000>
Customer
buys
product
customerID
first name
last name
/total quantity
*
*
P#
name
date
Buying
quantity

Angabe b)

- Durchschnittliche Quantitäten sind gegeben
- 20 Arbeitstage pro Monat
- Schreibzugriff -> Gewicht = 2
- Lesezugriff -> Gewicht = 1
- Update -> Gewicht = 3
- T1, T2, T3, T4 gegeben

Operation Frequency Table

Operation	Description	Frequency
T1	Insert new customer	50 per month / 20 = 2,5 per day
T2	Insert new product	5 per month / 20 = 0,25 per day
T3	Insert of a buying	50 per day
T4	Listing of the total quantity of each customer	2 per month / 20 = 0,1 per day

Data Access Table (ohne red. Attr.)

	Description	Concept		Avg. Accesses
T1	Insert new customer	Customer	W	2,5 * 2 = 5 per day
T2	Insert new product	Product	W	0,25 * 2 = 0,5 per day
T3	Insert of a buying	Buying	W	50* 2 = 100 per day
T4	Listing of the total quantity for each customer (ohne redundantem Attribut)	Customer Buying	R R	1 * 100 * 200 000*1 = 20.000.000 per m = 1.000.000 per day
		Summe		5 + 0,5 + 100 + + 2.000.000 = **1.000.105,5 per day** **= 20.002.110 per month**

Data Access Table (mit red. Attr.)

	Description	Concept		Avg. Accesses
T1	Insert new customer	Customer	W	2,5 * 2 = 5
T2	Insert new product	Product	W	0,25 * 2 = 0,5
T3	Insert of a buying Update Customer	Buying Customer	W U	2 * 1 = 2 3 * 1 = 3 50 * (2 + 3) = 250
T4	Listing of the total quantity for each customer (mit redundantem Attribut)	Customer	R	200.000 /20 = 10.000 per day
		Summe		5 + 0,5+ 250+ + 10.000 = **10.250,5 per day** **205.010 per month**

Aufgabe 4

Logical Design (Generalization Hierarchy)

Angabe

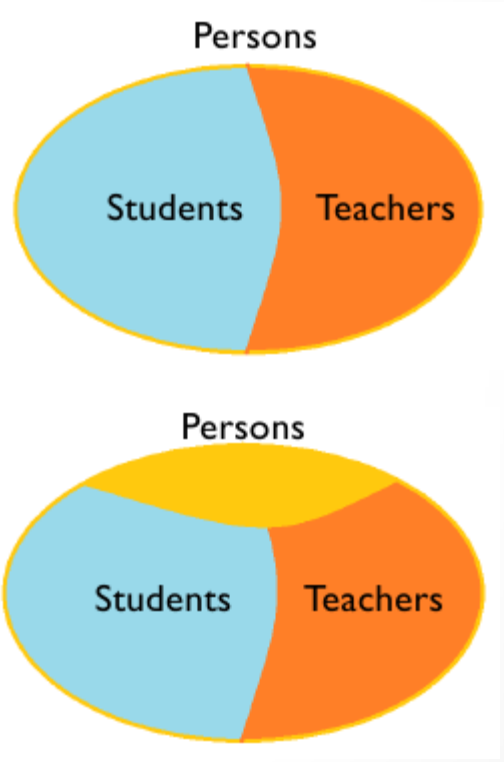
- Erkläre total, partial, exclusive und overlapping (Generalisierung)
- Welche Lösungsstrategien gibt es für Generalisierung?
- Wie wählt man solch eine Strategie aus?

Generalisierung

- Generalisierung -> Subklassenbeziehung zwischen zwei oder mehr Klassen
- Zum Beispiel Transformation eines UML Diagrammes in ein relationales Modell
- Um dieses Problem zu lösen -> verschiedene Generalisierungstypen

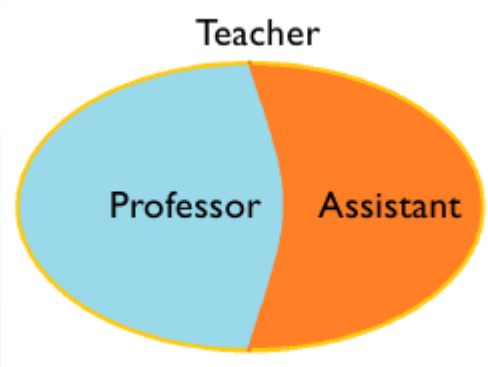
Generalisierung

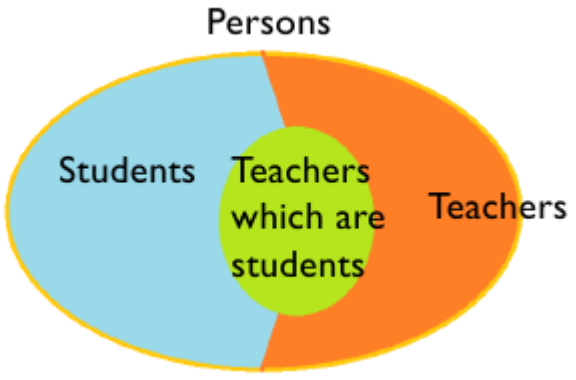
- Total vs. Partial
  - Total: jede Instanz einer Superklasse ist auch Instanz einer Subklasse
  - Partial: es muss nicht jede Instanz einer Superklasse Instanz einer Subklasse sein



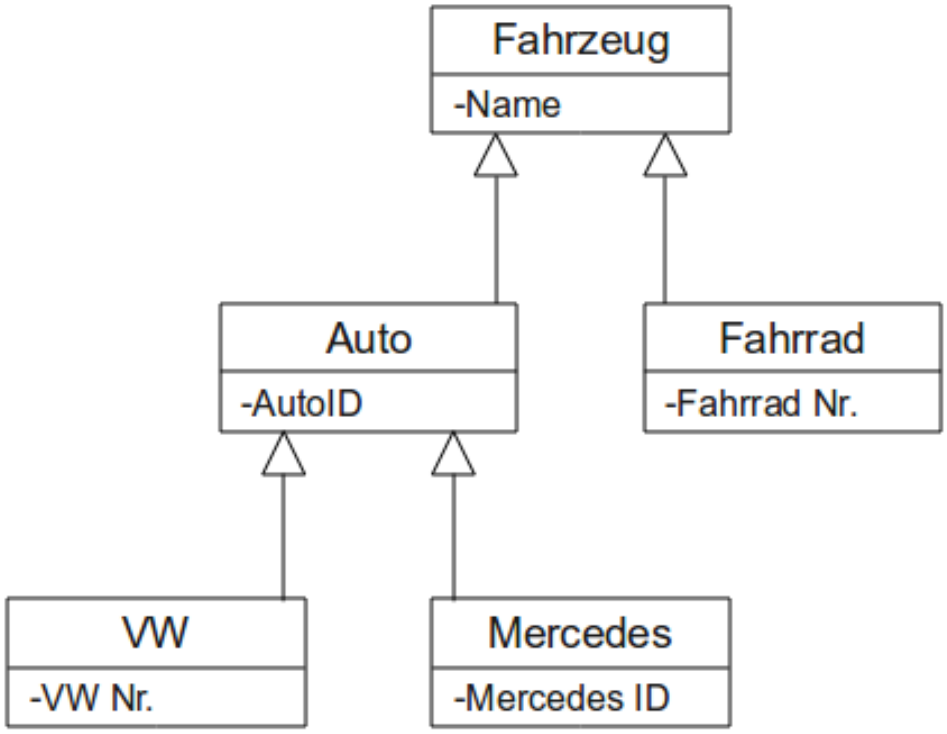
Generalisierung

- Exclusive vs. Overlapping
  - Exclusive: keine Instanz der Superklasse gehört zu mehr als einer Subklasse
  - Overlapping: Instanz einer Superklasse kann auch zu mehreren Subklassen gehören



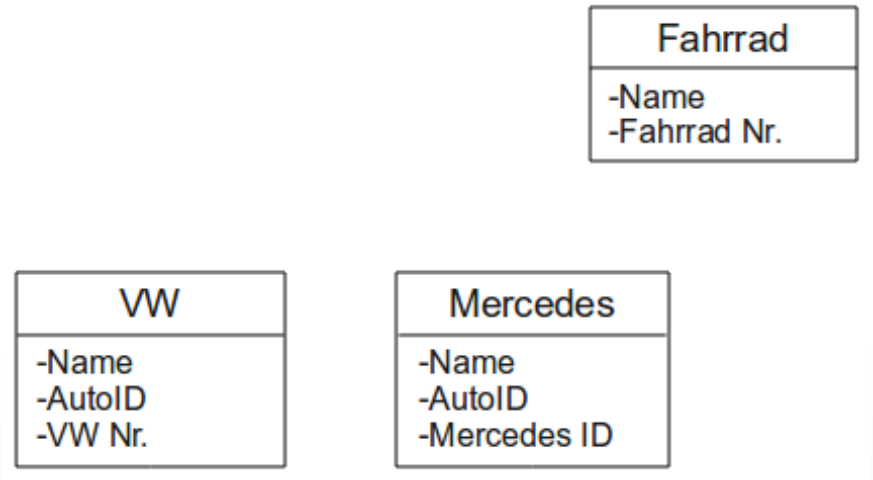


Beispiel


Fahrzeug
-Name
Auto
-AutoID
Fahrrad
-Fahrrad Nr.
VW
-VW Nr.
Mercedes
-Mercedes ID

Lösungsstrategie Floor

- Nur Klassen die im Baum Blätter darstellen bleiben übrig
- Superklassen werden entfernt
- Nur anwendbar bei totaler oder exklusiver Generalisierung!

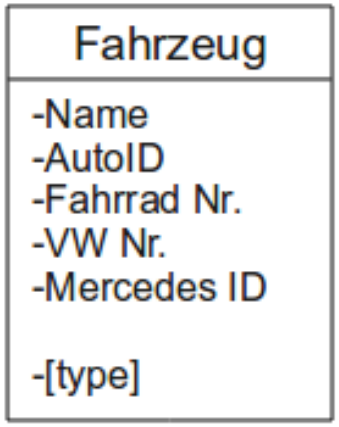


Lösungsstrategie Floor

- Vorteile
  - Es gibt eine Klasse pro Typ (total oder exclusive)
- Nachteile
  - Man benötigt Union Operator damit man alle Instanzen einer Superklasse erhält

Lösungsstrategie Ceiling

- Nur Superklasse mit allen Attributen aus den Subklassen
- Subklassen werden entfernt
- Zusätzliches Type Attribut wird hinzugefügt

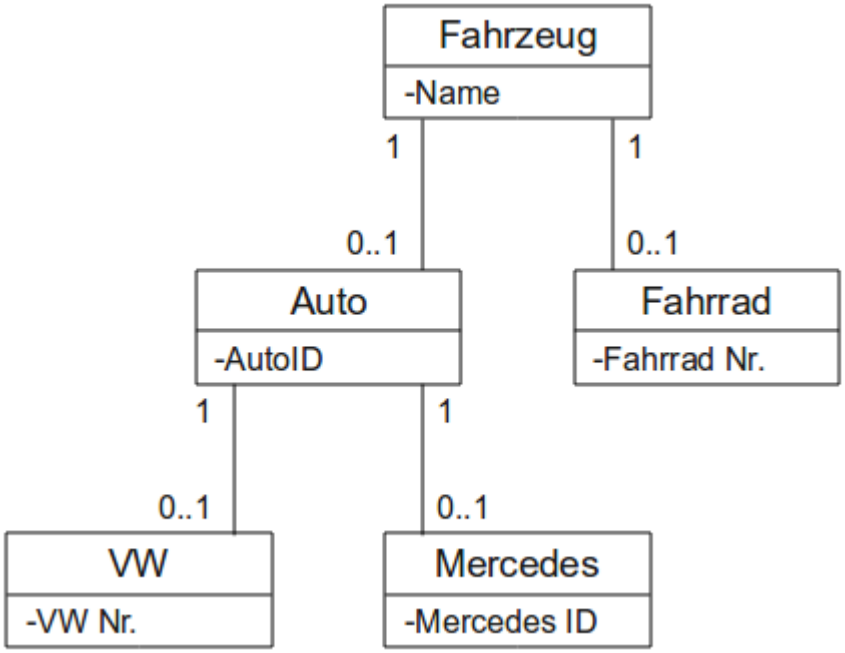


Lösungsstrategie Ceiling

- Vorteile
  - Eine Klasse beinhaltet alle Attribute
- Nachteile
  - Viele Attribute bleiben leer
  - Beziehungen gehen verloren

Lösungsstrategie Cohesion

- Jede Generalisierung wird durch eine 1:1 Beziehung dargestellt



Lösungsstrategie Cohesion

- Vorteile
  - Alle Klassen bleiben bestehen mit Assoziationen
- Nachteile
  - Benötigt Join Operation um alle Instanzen einer Superklasse zu bekommen
  - Semantik der Generalisierung geht verloren

Strategieauswahl

- Identifiziere zwei Mengen an Operationen
  - S1 (Menge an Operationen, die auf Attribute der Superklasse zugreifen)
  - S2 (Menge an Operationen, die auf Attribute der Superklasse und exakt einer Subklasse zugreifen)
- Berechne Anzahl der Zugriffe für S1 und S2
  - Unter Verwendung der Operation Frequency Table
- Wenn S2 dominiert -> verwende Floor Strategie
- Ansonsten analysiere die Update Operationen
  - Wenn Operationen dominieren, die auf Attribute der Superklasse und der Subklasse zugreifen -> Ceiling Strategie
  - Wenn Operationen dominieren, die auf Attribute der Superklasse oder der Subklasse zugreifen -> Cohesion Strategie

Strategieentscheidung

- Laufzeit/Kosten-Vergleich bei Zugriffen nur auf die Superklasse:
  - Floor
  - Ceiling

- Hohe Performance schwer erreichbar
- Für exklusive Generalisierung ist Ceiling am besten geeignet.

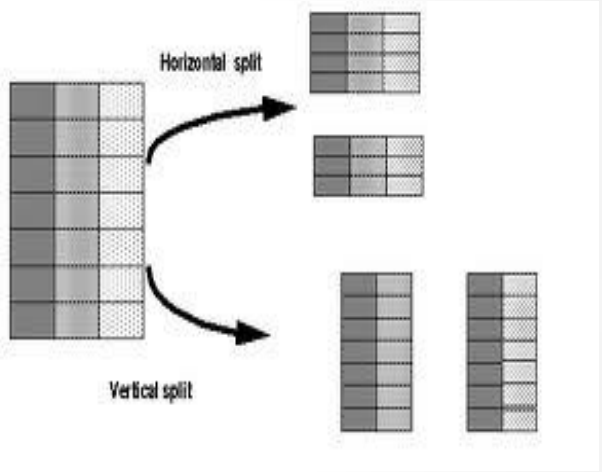
Aufgabe 5

Logical Design (Partitioning)

Angabe

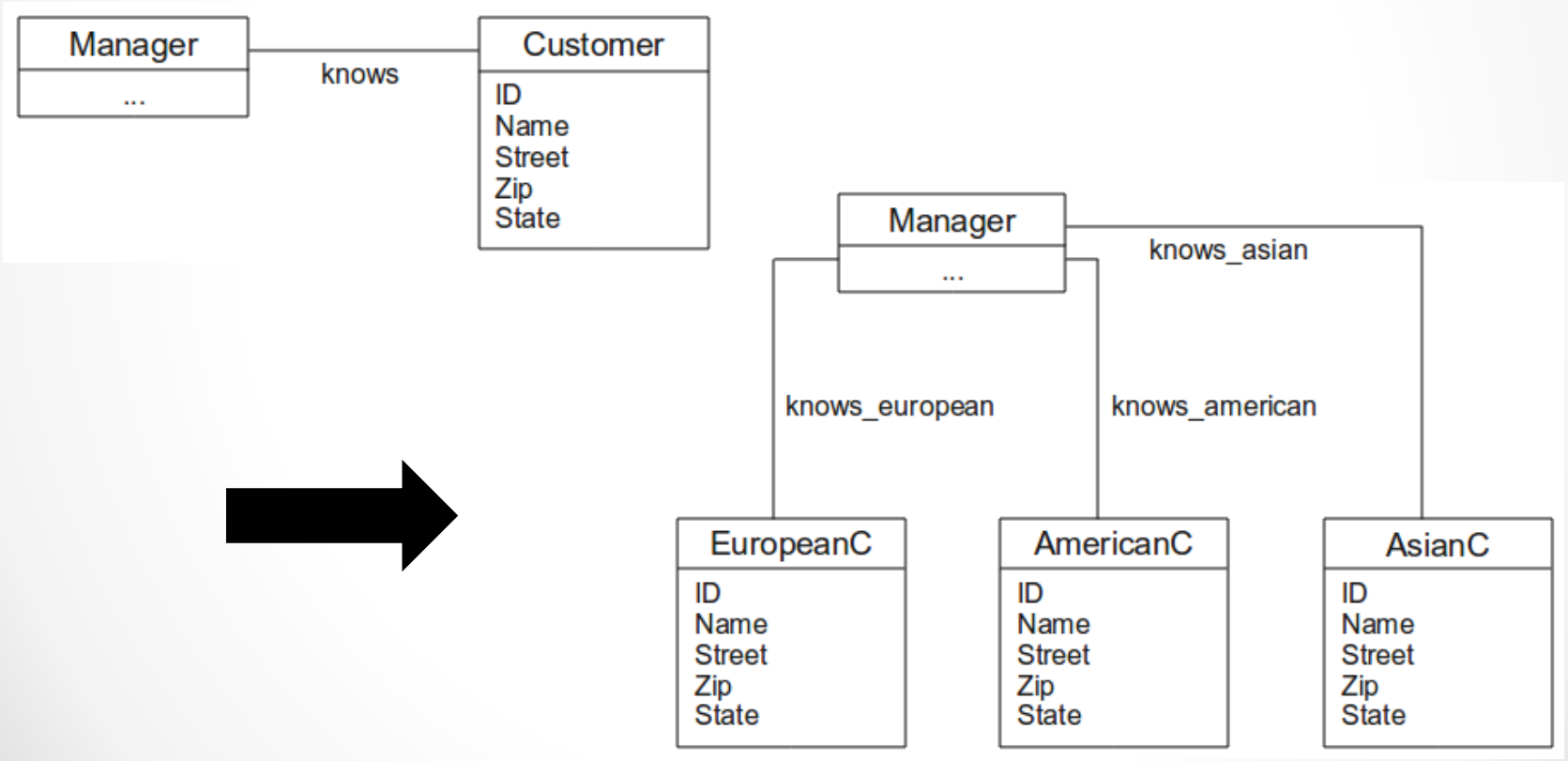
- Was ist vertikales / horizontales Partitionieren?
- Warum könnte Partitionieren notwendig sein?
- Auf was muss beim Partitionieren geachtet werden? Wie entscheidet man?

Horizontales Partitionieren I



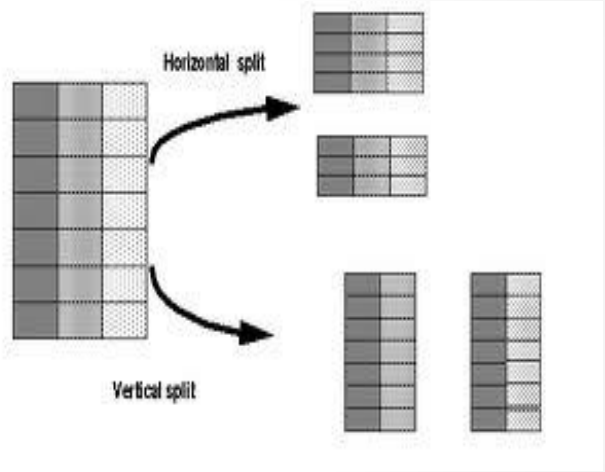
- Klasse wird horizontal aufgeteilt
- Neue Klassen haben die gleiche Menge an Attributen
- Multiplizieren der Assoziationen ist nötig
- Union Operation wird benötigt, um ursprüngliche Menge der Instanzen zu erhalten

Horizontales Partitionieren II

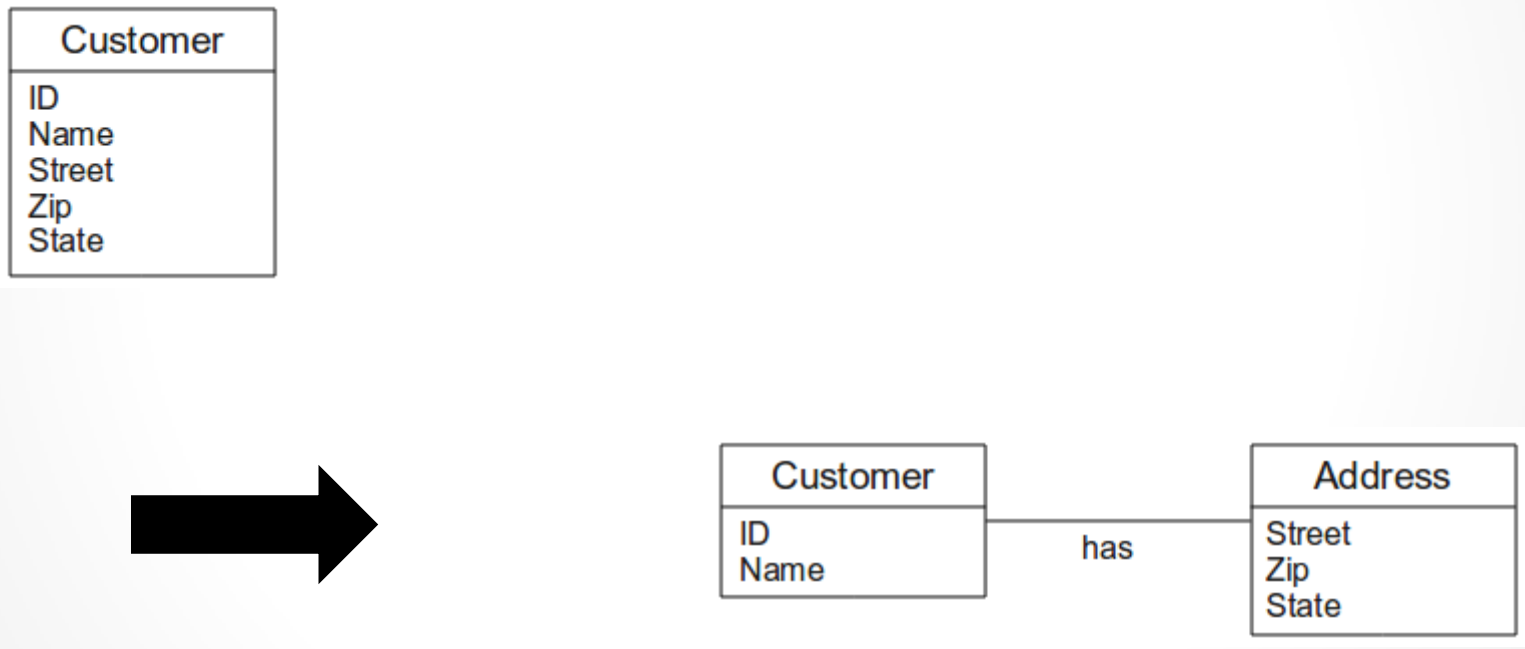


Vertikales Partitionieren I

- Klasse wird vertikal aufgespalten
- Neue Klassen haben unterschiedliche Attribute
- Join Operation wird benötigt, um die ursprünglichen Instanzen zu bekommen



Vertikales Partitionieren II



Gründe Partitionieren I

- Horizontales Partitionieren
  - Viele Operationen auf unterschiedlichen Mengen von Instanzen (z.B. AsianC, EuropeanC)
- Vertikales Partitionieren
  - Viele Operationen auf unterschiedlichen Mengen von Attributen (z.B. Address)

Gründe Partitionieren II

- Große Attribute aufspalten
- Sicherheitsaspekte
- Performance Gründe
- Traffic Reduktion in verteilten Datenbanken
- Schnellere Verfügbarkeit von Daten

Auf was muss geachtet werden?

- Welche Attribute werden oft abgefragt, aus Performance Gründen dann partitionieren

- Welche Partitionierung ist für das entsprechende Schema am besten
  (viele Attribute oder verschiedene Mengen in einer Klasse)