

Relationale Datenbanken

1. Aufbau und Organisation von Datenbanken und DBMS
2. Physische Architekturkonzepte
3. Statischer Entwurf mit dem Entity-Relationship-Modell
4. Das relationale Datenmodell
5. Integrität in relationalen Datenbanken
6. Normalisierung und algorithmischer Schema-Entwurf

Relationale Datenbanken (2)

7. Objektbasierte Modelle

8. Datenbanksprachen

9. Sprachen für Objektrelationale Datenbanken –
Beispiel PostgreSQL

10. Datenbanksystemtechnik

11. SQLite

1. Aufbau und Organisation von Datenbanken und DBMS

Zielstellung:

- Architekturvorschlag für Datenbanken
- Datenunabhängigkeit
- nach außen sichtbare Funktionen (nicht interne Realisierung)
- Schnittstellen zwischen den Bausteinen des Systems
- organisatorisches Zusammenspiel der Komponenten

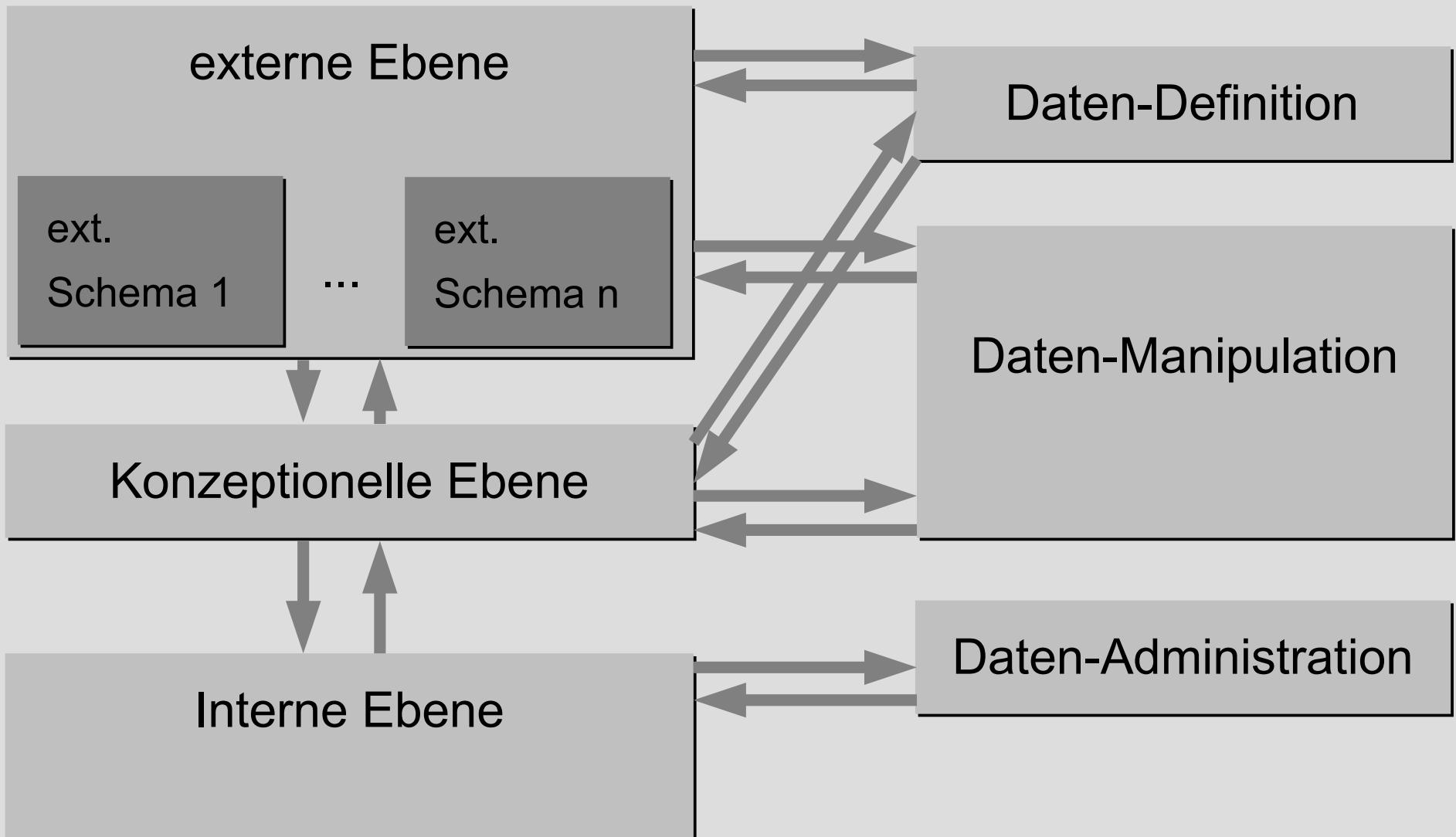
Datenunabhängigkeit

- Speicherung aller Daten einer Anwendung in integriertem Bestand (Datenbank):
 - dauerhaft
 - weitgehend frei von Redundanzen
 - Erhalt über Ausführungsdauer des zugreifenden Programms
- Verwaltung durch Kontrollinstanz:
 - kann Veränderungen durchführen (Updates)
 - vermeidet Inkonsistenzen (Überwachung)
- Zeitgleiche Nutzung des Datenbestandes durch alle Benutzer
 - einheitliche Integritätskontrollen
- Unterschiedliche logische Sicht möglich
 - anwendungsbezogene Strukturierung
 - Entwicklung neuer Anwendungen auf vorhandenem Datenbestand leicht möglich

ANSI / SPARC Schemaebenen

- DBMS basiert stets auf Datenmodell
 - Abstraktion eines gegebenen Ausschnittes der realen Welt
 - mit bestimmten Konzepten wird die Struktur der Datenbank beschrieben:
 - Datentypen
 - Beziehungen
 - Bedingungen
 - Operationen zur Beschreibung von Anfragen und Updates
- Kategorien von Datenmodellen:
 - High-Level oder konzeptionelle Modelle
 - Low-Level oder physische Modelle

3-Ebenen-Datenbankarchitektur



Sprachebenen und -klassen

- Ebenen kommunizieren untereinander bzw. mit der jeweiligen Außenwelt über Schnittstellen
- Abbildungen oder Transformationen zwischen:
 - externen Schemata und dem konzeptionellen
 - konzeptionellen und internen Schema
 - internen Schema und der Datenbank
- Zusätzlich stellt DBMS auch Sprachen bereit, welche mit den einzelnen Ebenen assoziiert werden können:
 - **Data Definition Language (DDL)**
 - **Data Manipulation Language (DML)**
 - **Data Administration Language (DAL)**

Einbettung DML in Wirtssprache

- Precompiler
 - akzeptiert Input mit DML-Statements
 - übersetzt in „reines“ Programm der Wirtssprache
- Um DML-Kommandos erweiterter Compiler
 - direkte Übersetzung des Programms in ausführbaren Code
- DML wird um Elemente einer höheren Programmiersprache erweitert
 - vollständige Programme lassen sich in dieser Sprache formulieren
 - eigenständiger DML-Compiler verarbeitet diese
 - strenggenommen keine Einbettung!

Beispiel einer relationalen Datenbank

Buch	InvNr	ErstAutor	WeitereAut	Titel	...
	123	Date	n	Intro DBS	
	234	Jones	y	Algorithms	
	345	King	n	Operating Syst.	

Ausleihe	InvNr	LeserNr	Rückdat
	123	225	22-04-1998
	234	347	31-07-1998

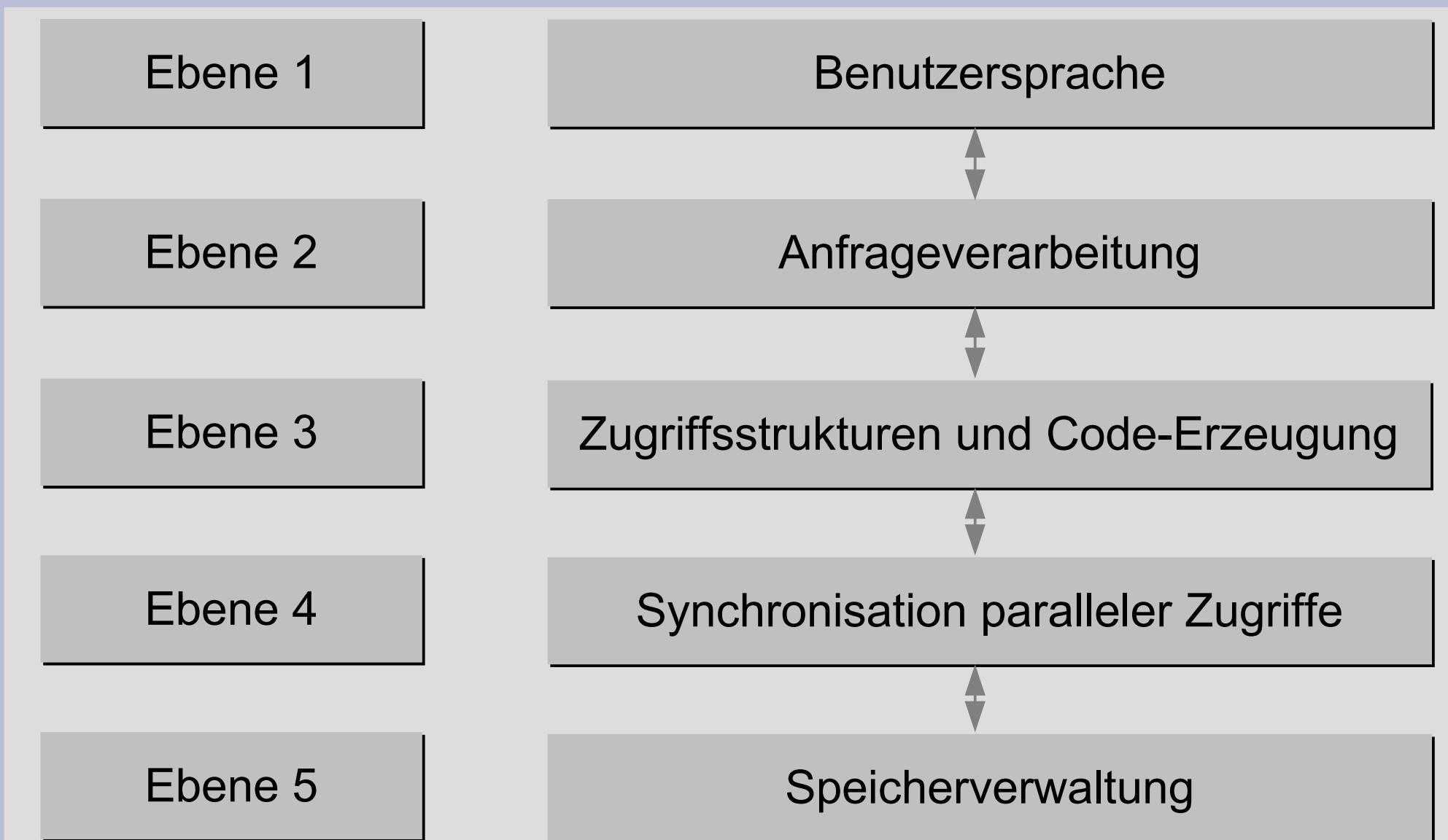
Leser	LeserNr	Name	...
	225	Peter	
	347	Laura	

Aufgaben des Datenbank-Administrators (DBA)

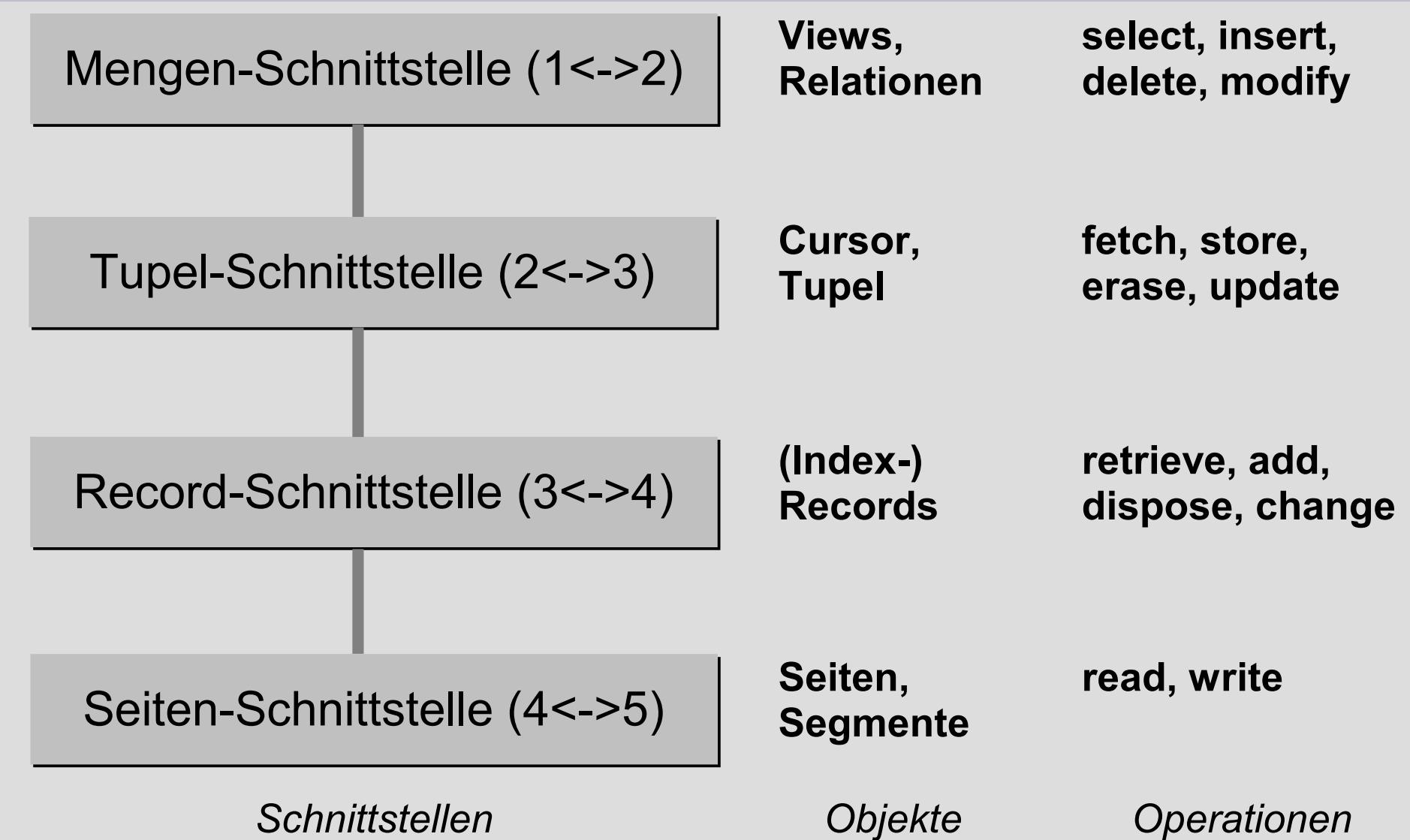
- Festlegung von Zugriffs- und Integritätskontrollen
- Sicherungs- und Wiederanlaufstrategien
- Überwachung Systemauslastung und Laufzeitverhalten des Systems (Performance)
- Tuning des Systems
 - Anpassung an neue und veränderte Aufgabenstellungen

Zunehmend verlagern Werkzeuge DBA-Aufgaben in die Kompetenz des Benutzers (Entwurfswerkzeuge für relationale Datenbanken)

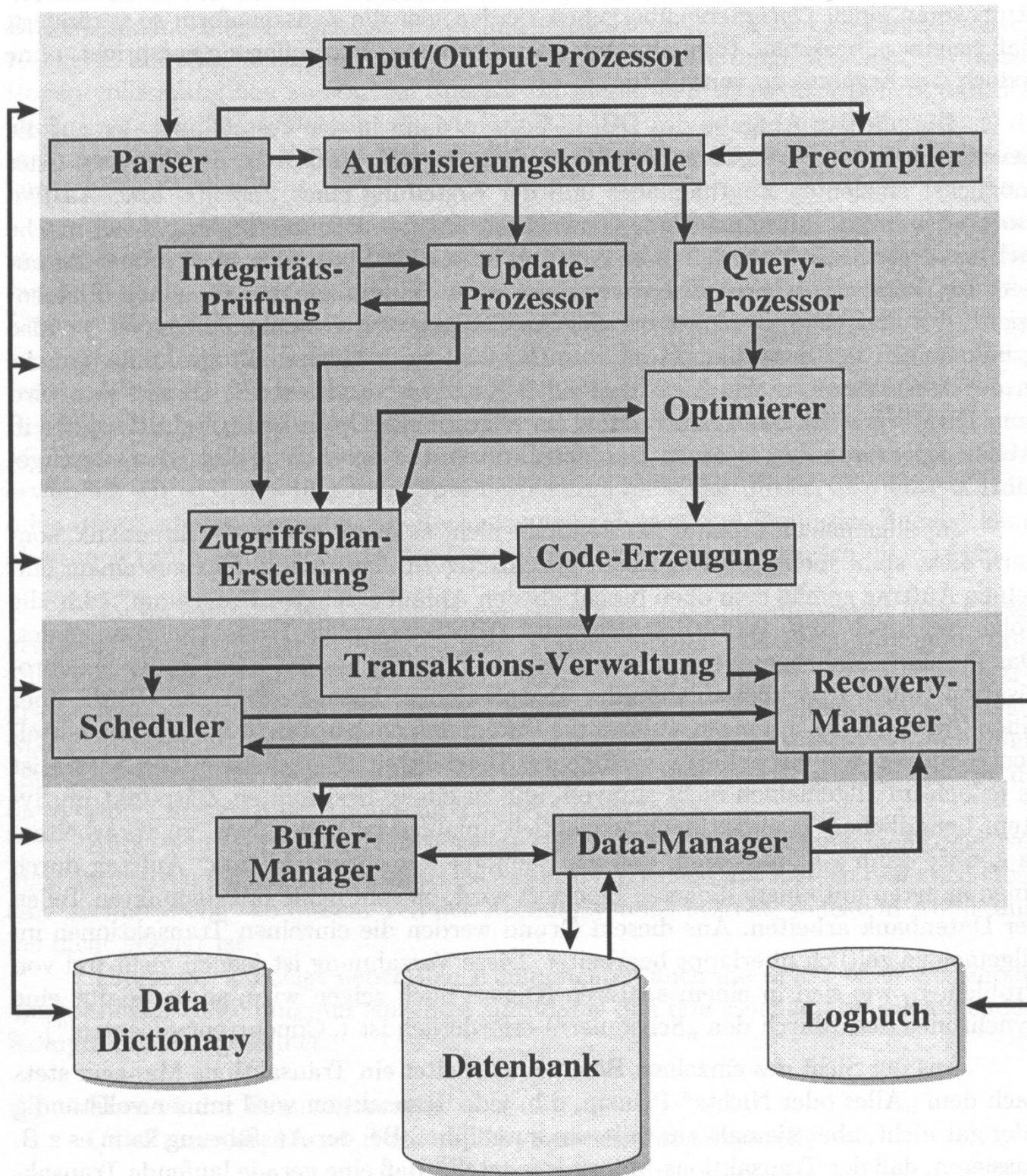
5-Schichten-Modell eines DBMS



DBMS-interne Schnittstellen



Komponenten eines DBMS



Ergänzungen

In der Informatik gibt es mindestens 3 zentrale Problemlösungs-Techniken:

- **Datenmodelle**
 - Abstraktionen zur Beschreibung von Problemen
 - Bäume, Listen, Mengen, Relationen, Graphen
 - Automaten, Grammatiken
- **Datenstrukturen**
 - Programmiersprachen-Konstrukte zur Repräsentation von Datenmodellen
 - Arrays, Records, Pointer
- **Algorithmen**
 - Techniken zur Herstellung von Problemlösungen durch Manipulation von Datenmodellen oder ihrer assoziierteren Datenstrukturen

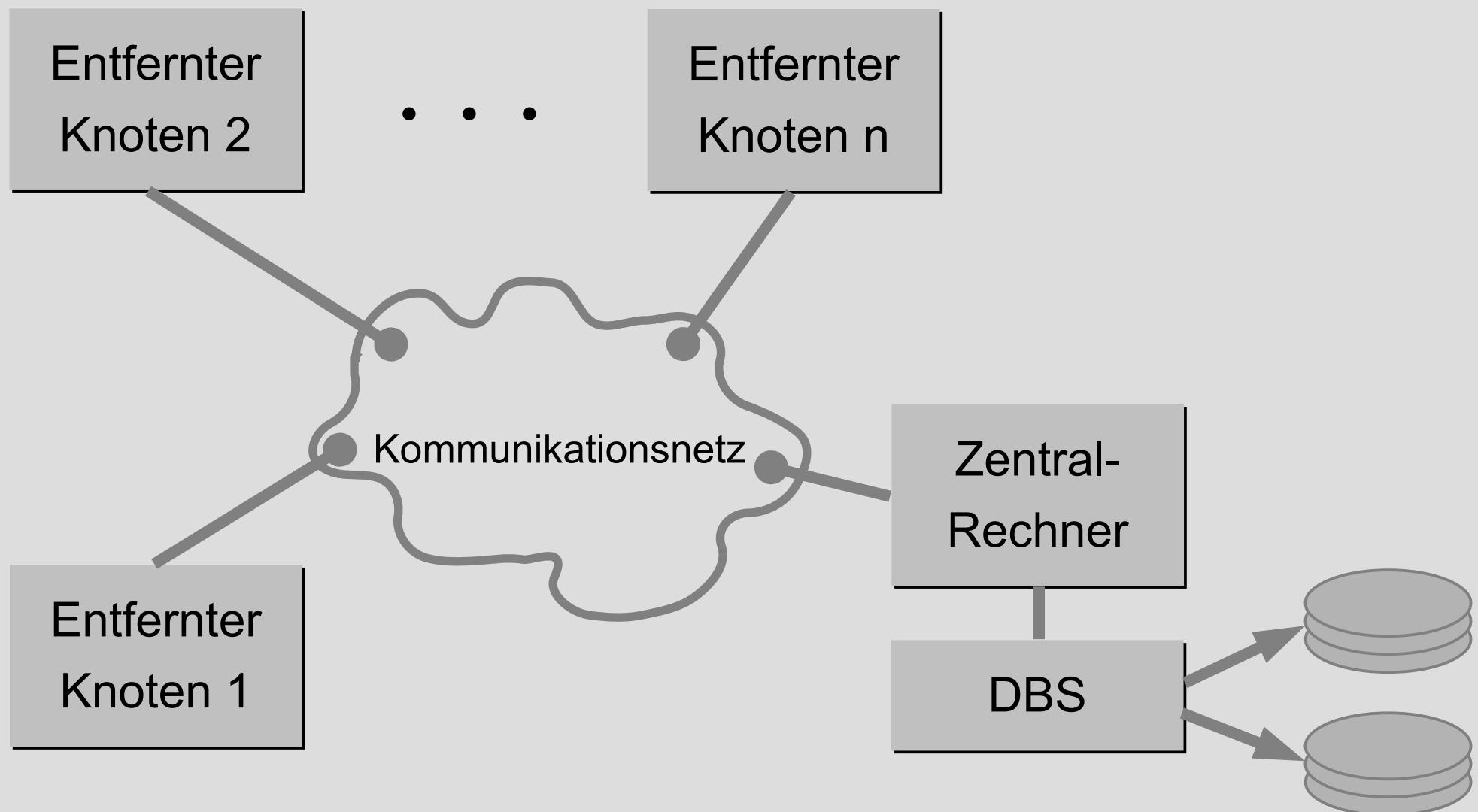
2. Physische Architekturkonzepte

- ANSI / SPARC Modell:
 - 3 Abstraktionsebenen
 - logisches Architekturkonzept
- (logische) Funktionsarchitektur nach Schema des letzten Abschnitts
- Kombination beider Konzepte mit der vorhandenen Rechnerumgebung ergibt:
physische Architekturkonzepte
 - Zentralisierte Systeme
 - Verteilte Systeme
 - Client-Server Konzept
 - Parallele Hardware

Zentralisierte Datenbanksysteme

- Ursprünglich: ein zentralisierter Rechtersystem
 - Zuständig für gesamte Funktionalität des DBMS
 - Auch logische Datensicht des ANSI/SPARC Modells wird hier abgebildet
- Aktuell: entfernte Knoten kommunizieren mit Zentralrechner
 - Räumliche Entfernung kaum wichtig
 - Jeder Nutzer von jedem Netzknoten hat die gleiche Sicht auf die Datenbank

Zentrale Datenbank

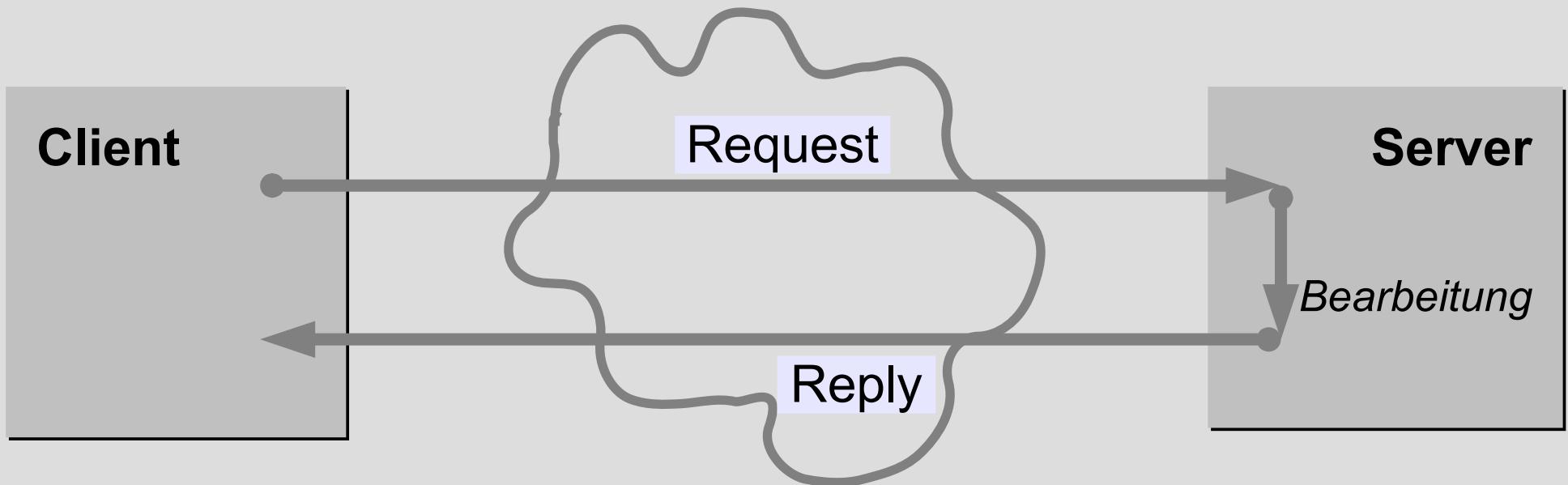


Datenbank-Serverarchitekturen

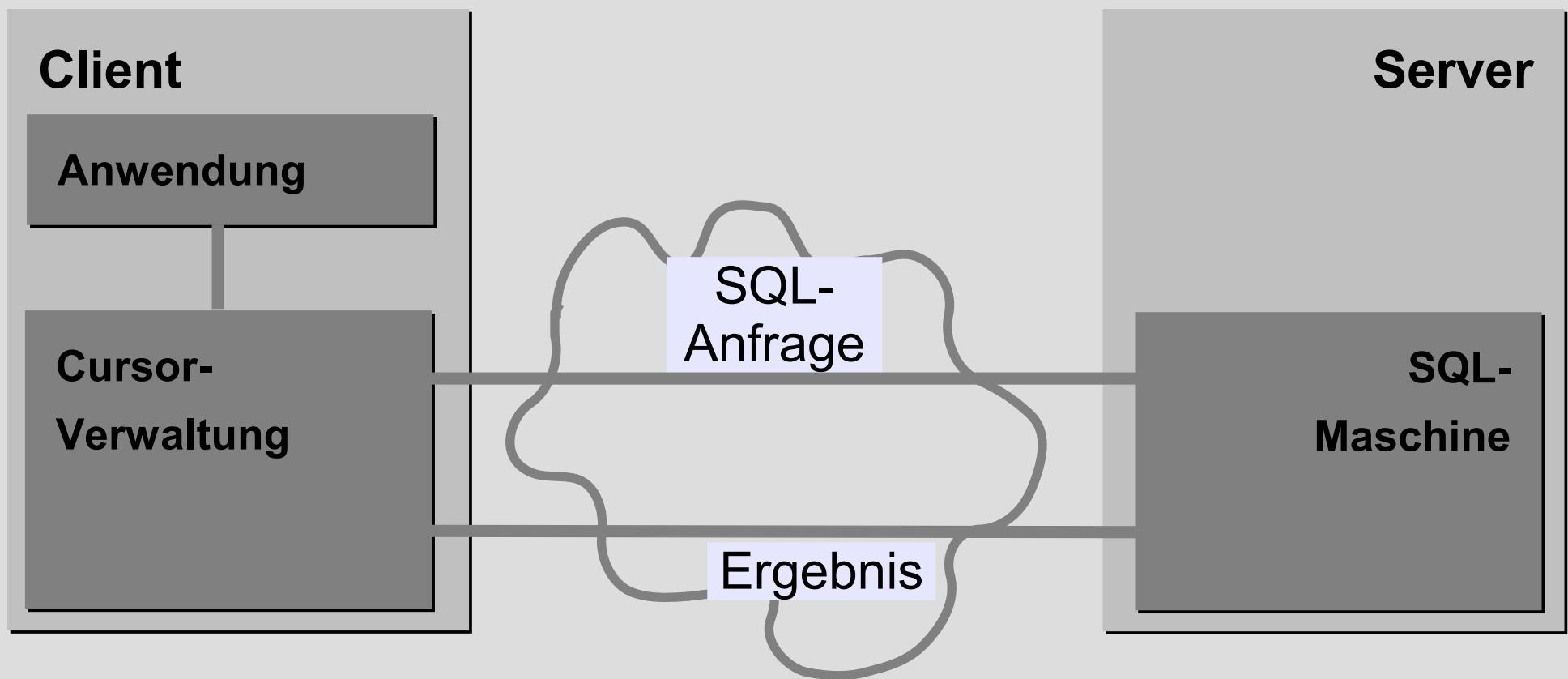
Wir unterscheiden u.a.:

- Client-Server-Konzept
- Anfrageserver
- Objekt- und Seitenserver

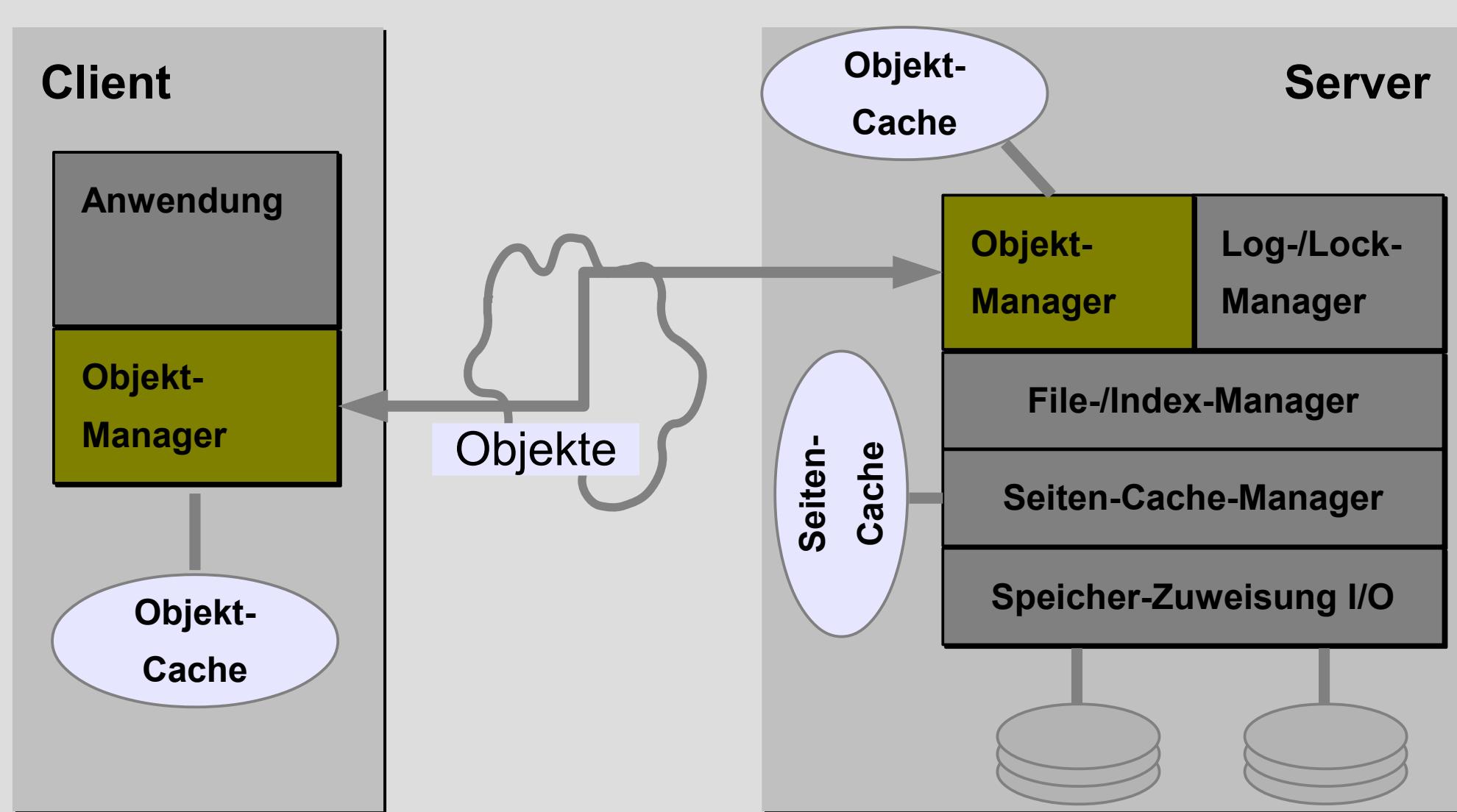
Client-Server Konzept



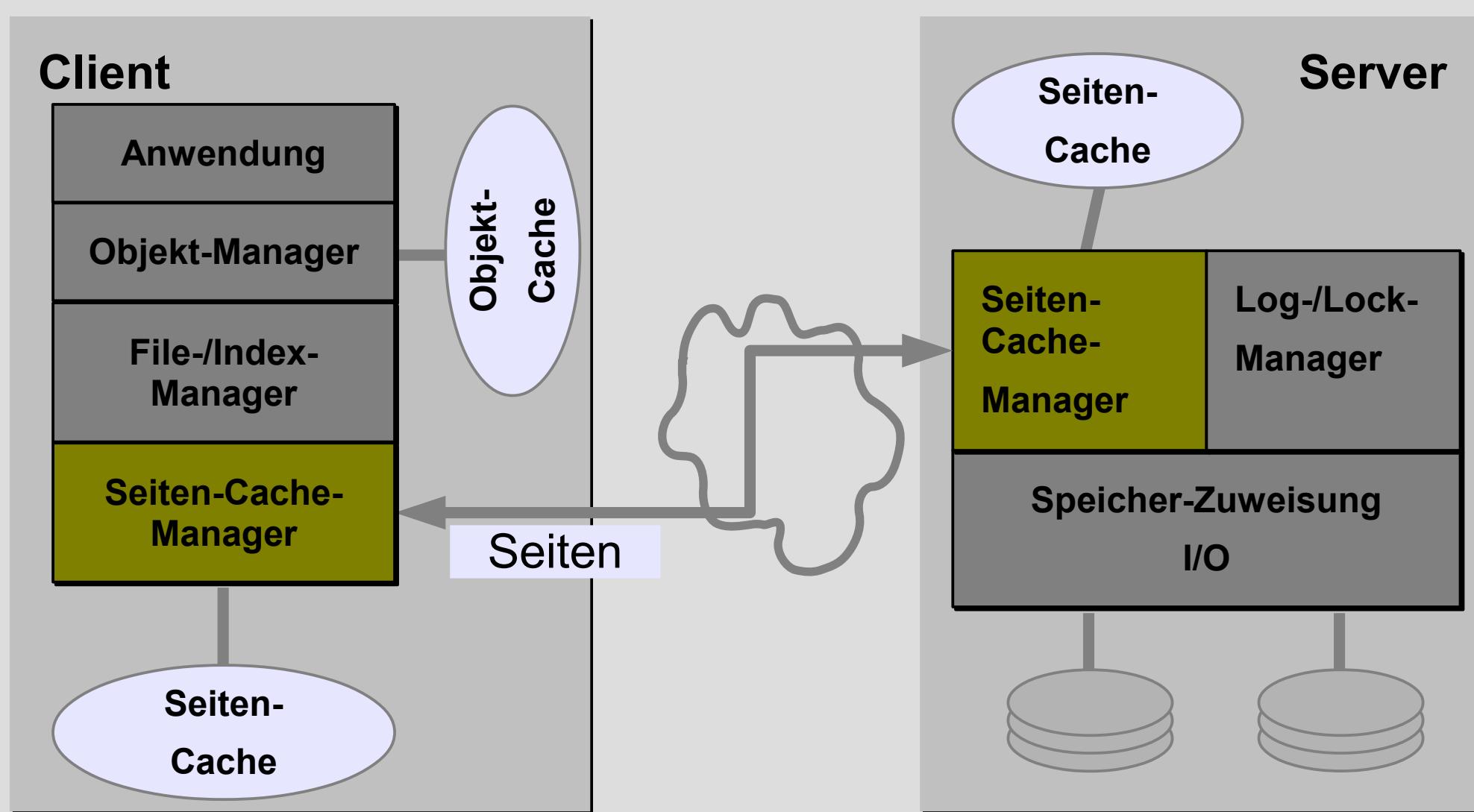
Relationaler Anfrageserver



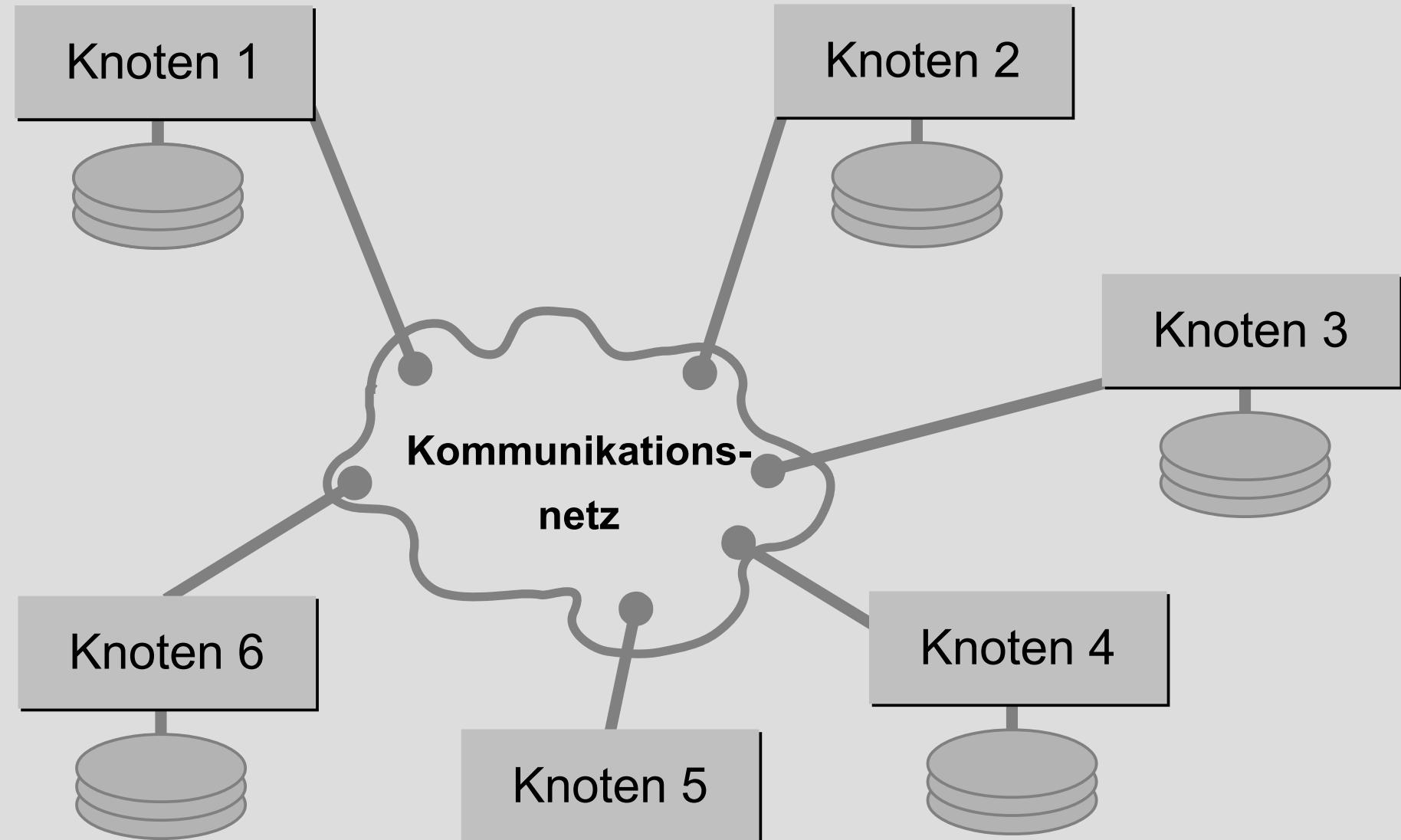
Objektserver-Architektur



Seitenserver-Architektur



Verteilte Datenbankumgebung



Verteilte Datenhaltung

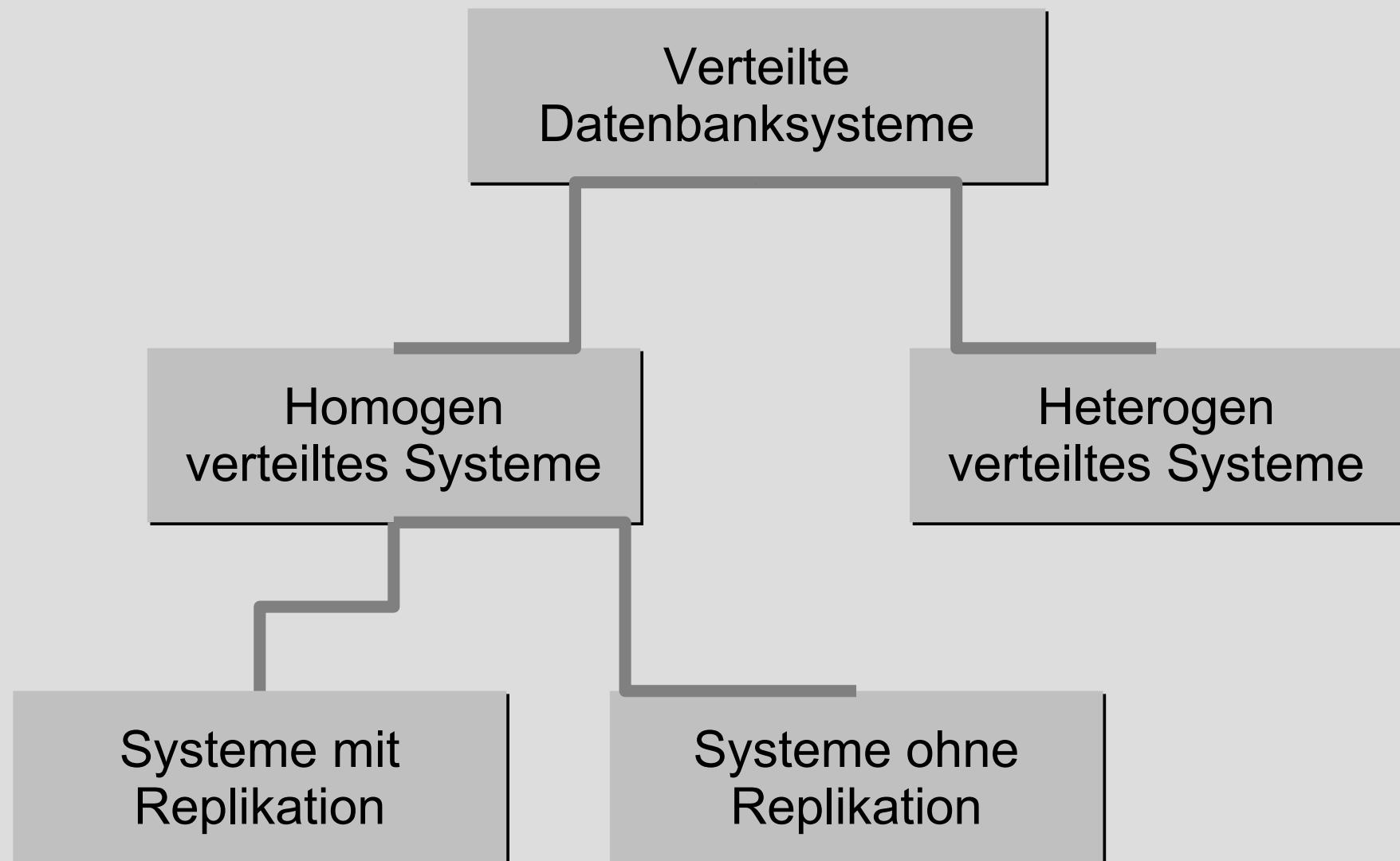
Vorteile

- Lokale Autonomie
- Leistungsverbesserung des Gesamtsystems
- Erhöhte
 - Zuverlässigkeit,
 - Verfügbarkeit und
 - Ausfallsicherheit
- Erweiterbarkeit
- Teilbarkeit

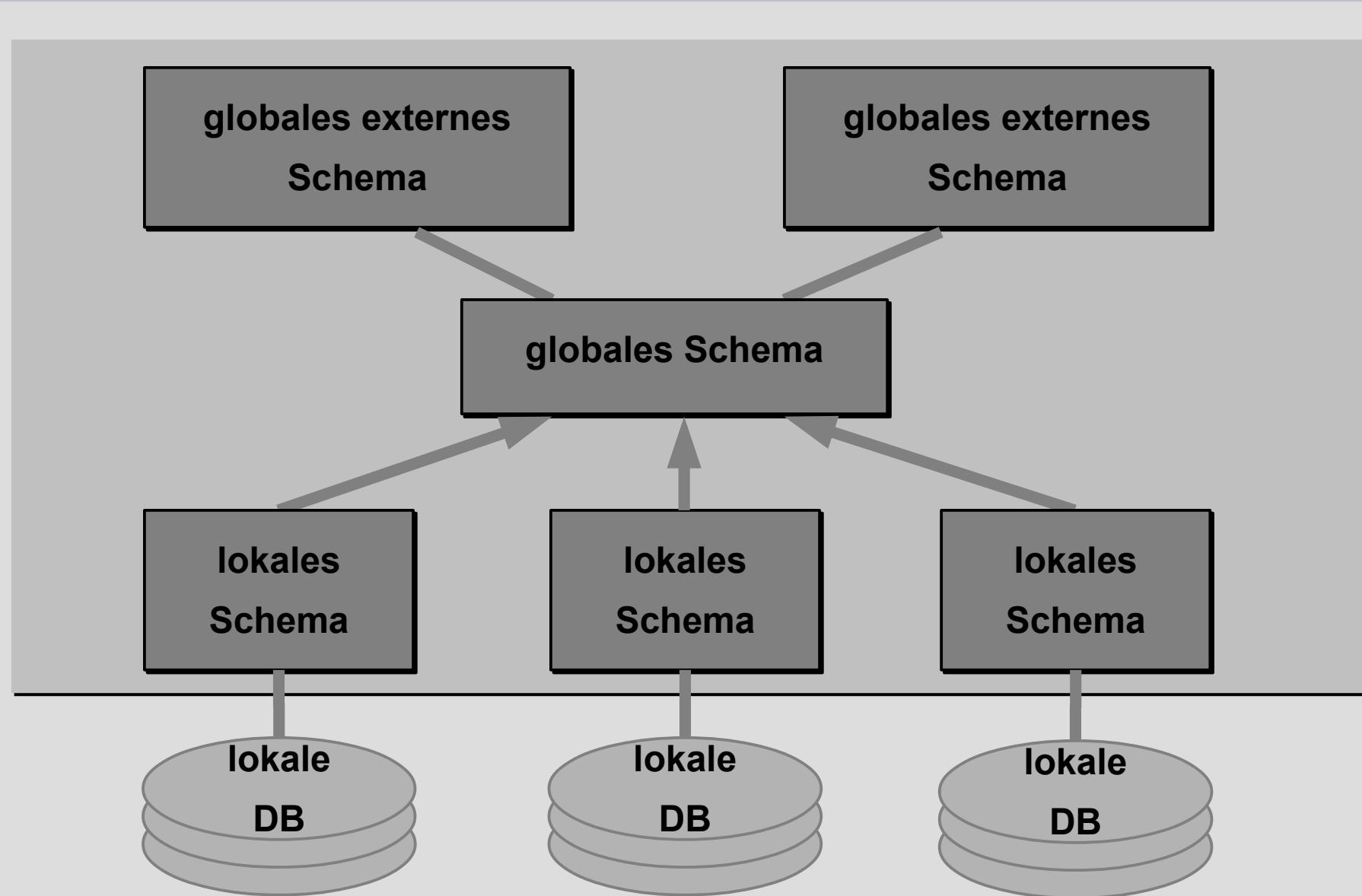
Nachteile

- Erfahrungsmangel
- Komplexität
- Kosten
- Verteilung von Kontrolle
- Sicherheit
- Schwierigkeit der Veränderung

Klassifikation verteilter Systeme



Schemata bei homogener Verteilung



Homogene Verteilung

Transparenz nach außen:

- Datenunabhängigkeit
 - Daten unabhängig vom zugreifenden Programm
 - Logische Sicht auf Daten wird von der physischen unterschieden
- Netzwerktransparenz
 - Einzelheiten des Netzwerks werden vor Nutzer verborgen
- Replikationstransparenz
 - Kopien werden vor Benutzer verborgen
- Fragmentierungstransparenz
 - Nutzer will nicht erfahren, wo die Daten liegen

Heterogene Verteilung

Sammlung i.a. Unterschiedlicher Datenbanksysteme

- Zusammenschluss von Unternehmen
- Verschiedene Anwendungen in Abteilungen

Hauptziel: Interoperabilität zwischen (ursprünglich) isolierten Systemen

Zentrale Anforderung: Autonomie

**2 Arten heterogen verteilter Datenbanksysteme
(auch Multidatenbanksysteme – MDBS - genannt):**

- MDBS mit globalem Schema

Teilnehmende Systeme exportieren Schema-Information an eine globale Ebene

- MDBS ohne globales Schema

Kein globales Schema

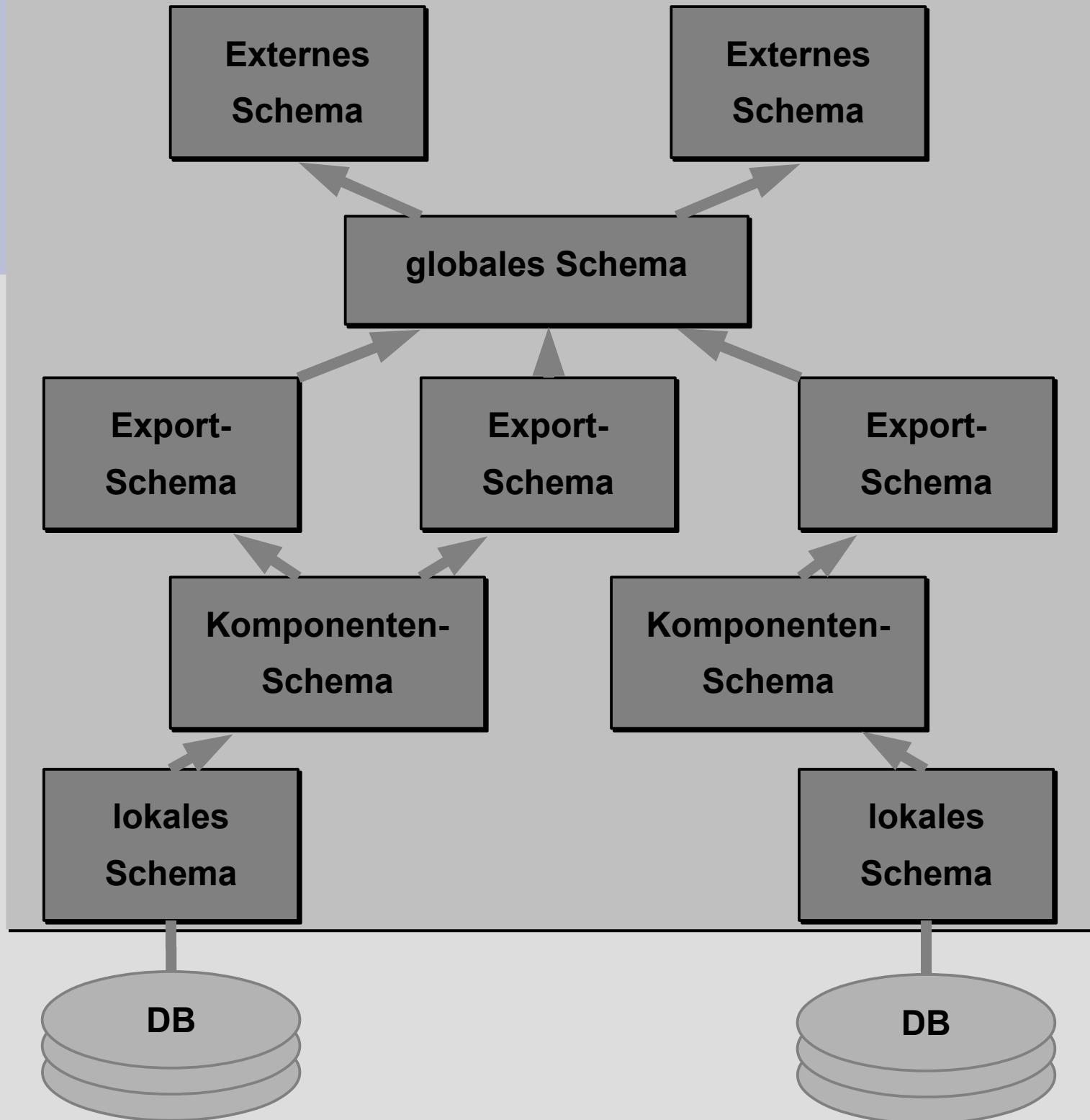
Schemaebenen bei heterogenen verteilten Systemen

Sichten-
Definition

Inte-
gration

Auf-
teilung

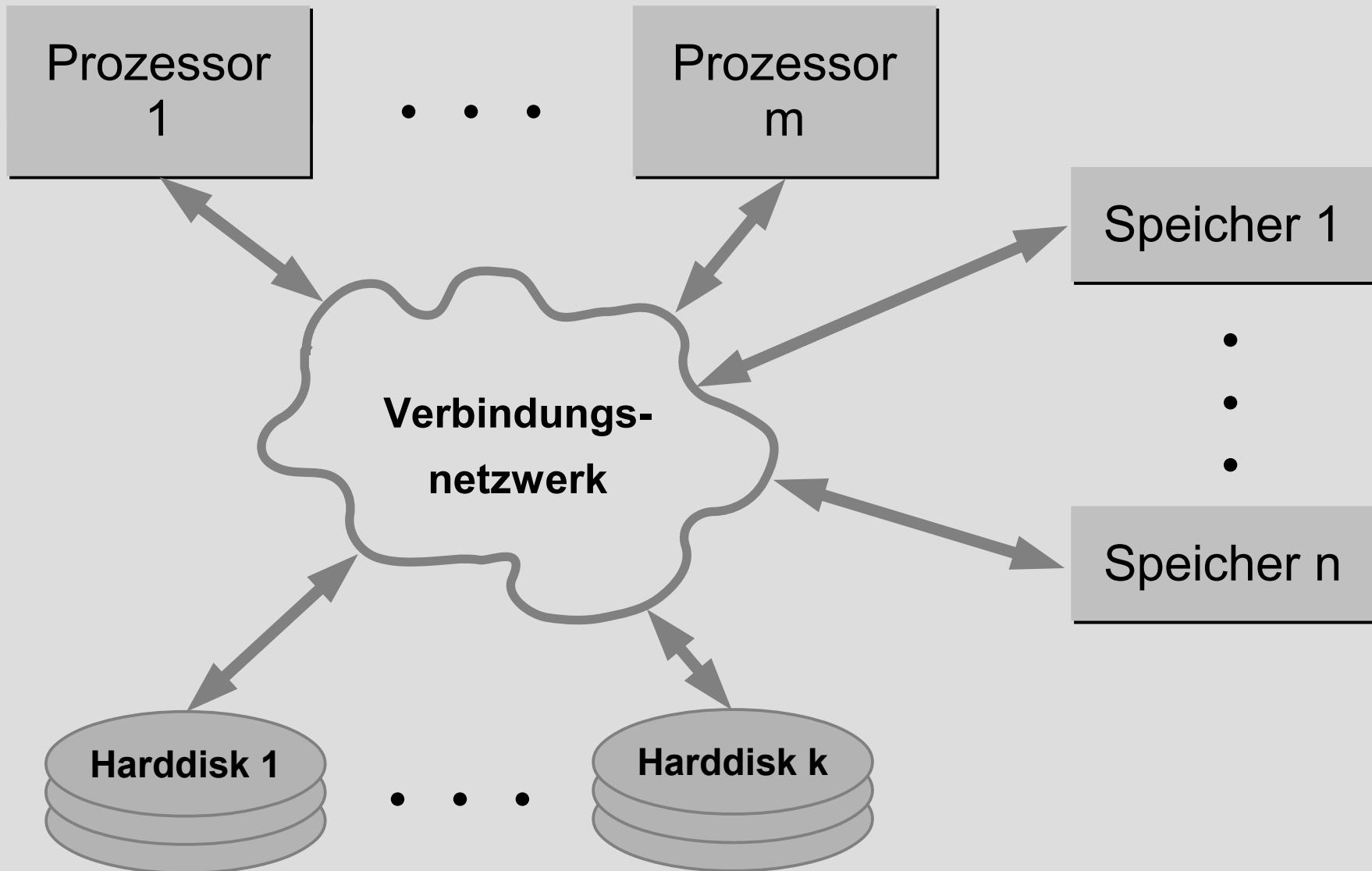
Über-
setzung



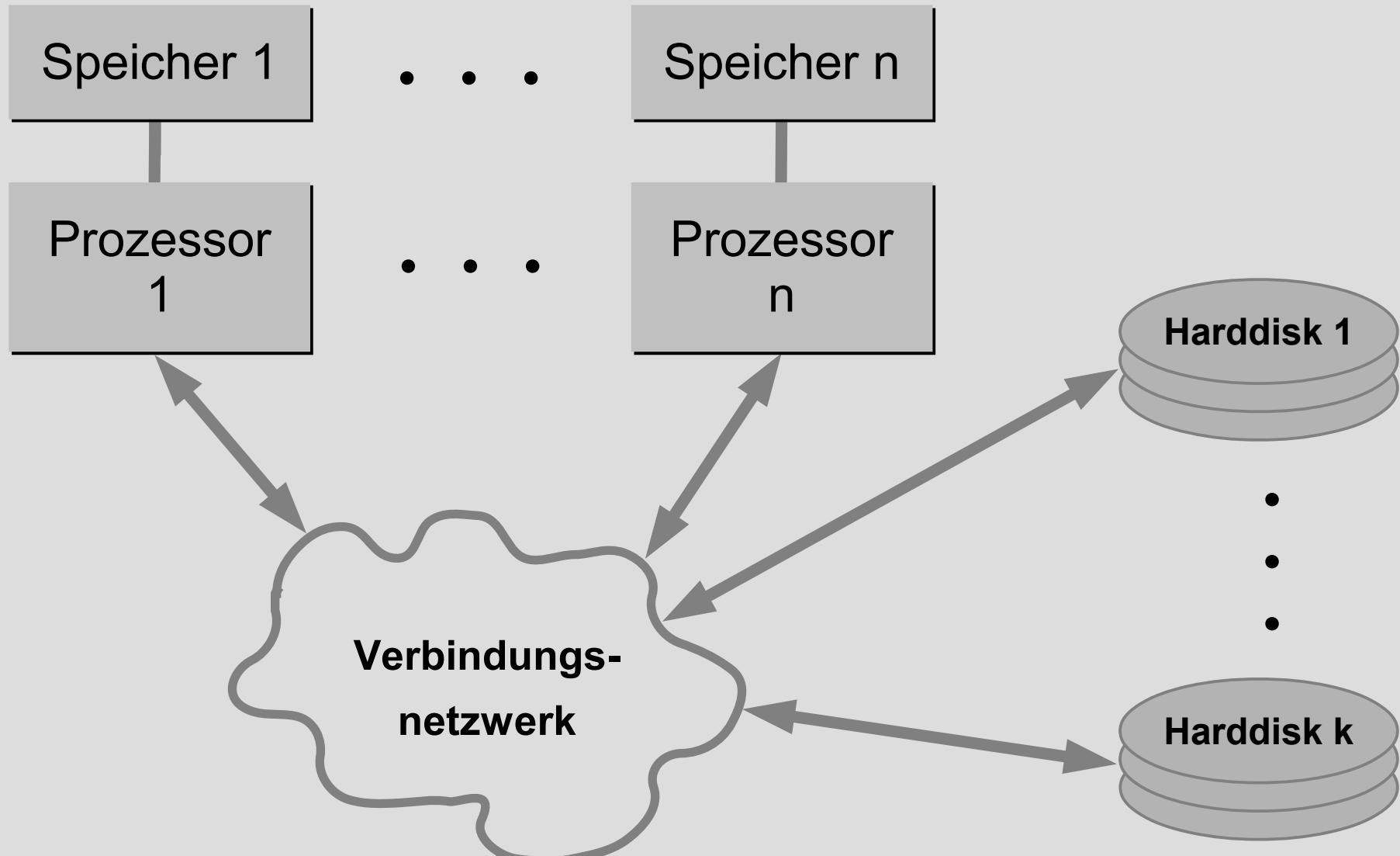
Parallele Datenbanksysteme

- Integration von Konzepten der Parallelverarbeitung in Datenbanksysteme
 - Funktionsparallelität (oberste Ebene DBS)
 - Datenparallelität (unterste Ebene DBS)
 - Abhängig von Architektur des Rechnersystems
- Parallelität in gewöhnlichem Mehrbenutzer-DBS:
 - Zeitliche verzahnter Zugriff von Transaktionen auf den gleichen Datenbestand
 - Scheinbar parallele Ausführung
 - Für Anwendungsprogrammierer transparent
- Paralleles System:
 - Mehrere Prozessoren
 - Lokaler / globaler Hauptspeicher
 - Sekundärspeicher
 - Verbindungsleitungen zwischen den Systembestandteilen

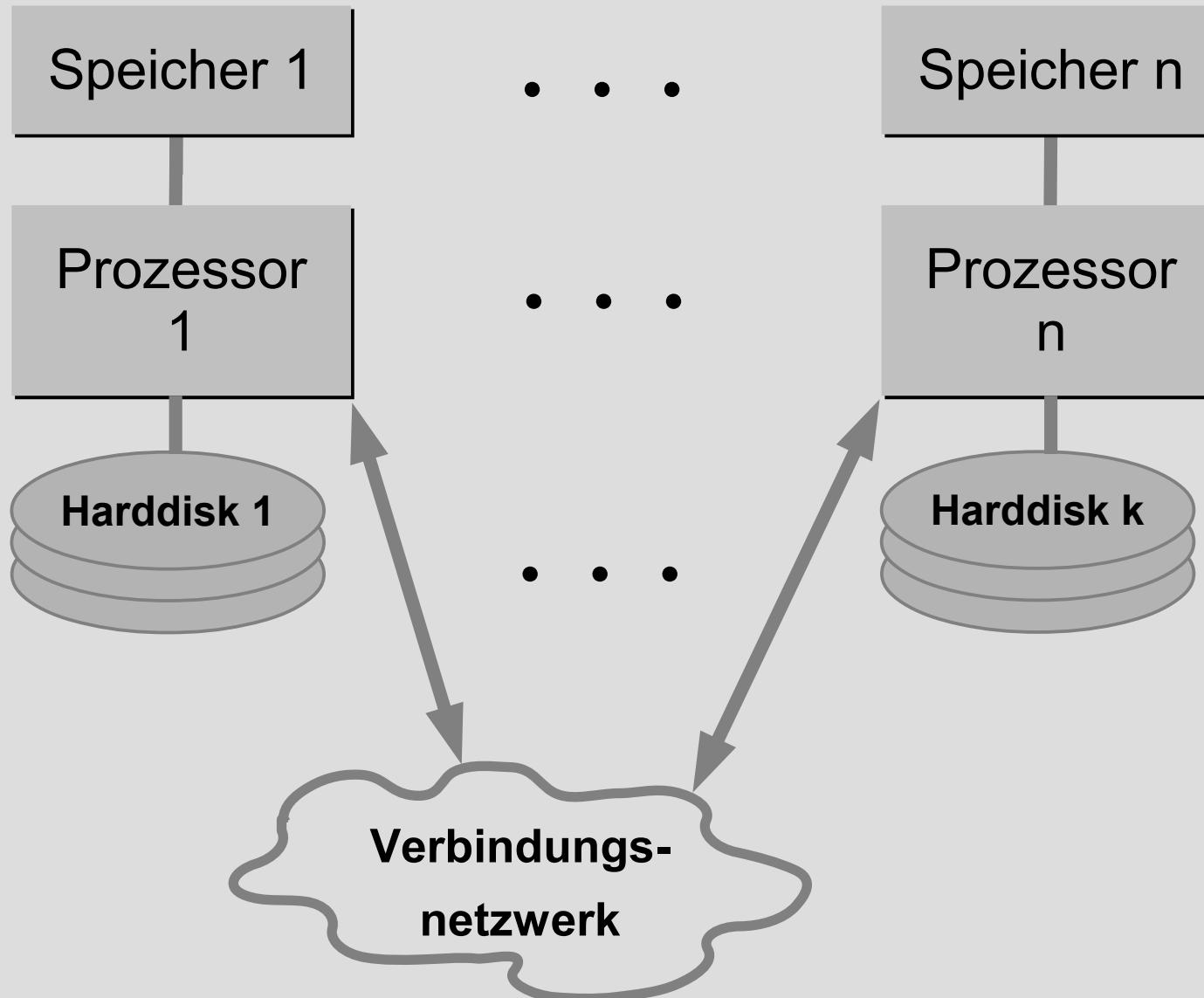
Shared-Memory-Architektur



Shared-Disk-Architektur



Shared-Nothing-Architektur



Parallelverarbeitung im DBS

- Parallelle Ausführung von Kommandos der Benutzersprache (Ebene 1)
 - z.B. mehrere Kommandos gehören zu Transaktion, Reihenfolge spielt keine Rolle
 - Parallelverarbeitung der Kommandos
- Parallelle Anfrageverarbeitung (Ebene 2)
 - z.B. Vereinigung der Ergebnisse zweier Teilausdrücke
 - Berechnung auf zwei Prozessoren möglich
- Parallelle Implementierung relevanter Operationen (Ebene 3)
 - Sortierung von Teilmengen auf verschiedenen Prozessoren
 - Ergebnisse mischen
- Paralleler Zugriff auf relevante Daten (Ebene 4)
 - z.B. Daten auf verschiedene Speichereinheiten verteilt

2. Statischer Datenbankentwurf

Datenbankentwurf:

Prozess der Bestimmung der Organisation und des inhaltlichen Aufbaus einer Datenbank.

Verwendetes Modell:

Entity Relationship Modell (ER)

Statischer Entwurf mit Entity-Relationship-Modell

Def. Datenbankentwurf:

- Die Aufgabe des Datenbankentwurfs ist der Entwurf der logischen und physischen Struktur einer Datenbank so, dass die Informationsbedürfnisse der Benutzer in einer Organisation für bestimmte Anwendungen adäquat befriedigt werden können.

Zu beachten sind insbesondere:

- Ermittlung logische Datenbankstruktur
- Entwurf der physischen Datenbankstruktur
- Nebenbedingungen, insbesondere Angemessenheit in Bezug auf die Anwendung durch den Nutzer

Lebenszyklus einer Datenbank

Informationssystem

- 1) Nutzen-Analyse
- 2) Anforderungsanalyse
- 3) Entwurf
- 4) Implementierung
- 5) Validation und Akzeptanz-Test
- 6) Betrieb

Datenbankanwendung

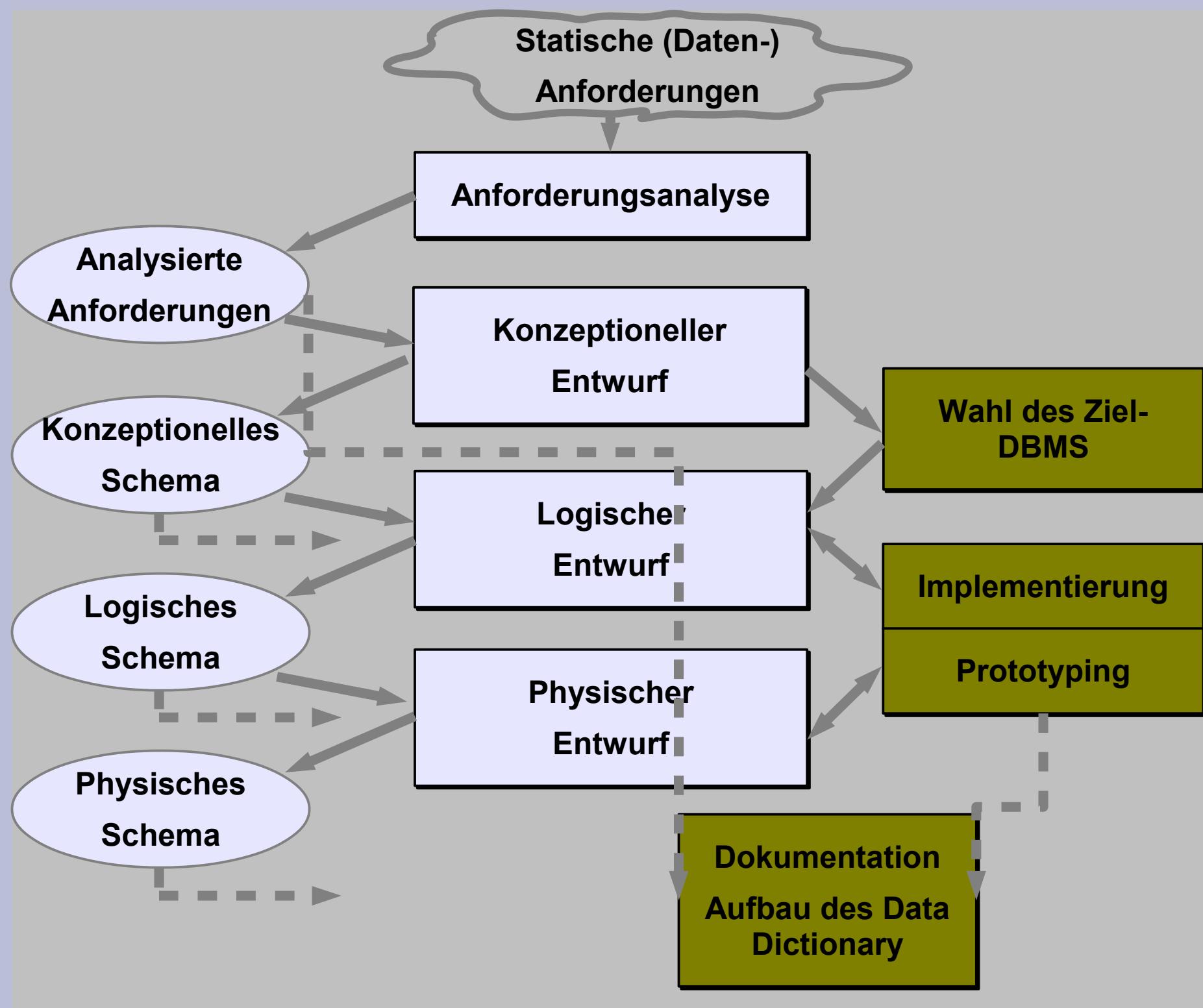
- 1) System-Definition
- 2) Entwurf
- 3) Implementierung
- 4) Laden und Daten-Konversion
- 5) Anwendungs-Konversion
- 6) Test und Validation
- 7) Betrieb
- 8) Überwachung und Wartung
- 9) Modifikation und Reorganisation

Qualitätssicherung

Setzt beim Schema an:

- Vollständigkeit
 - Erfasst alle relevanten Aspekte und Eigenschaften des Anwendungsbereichs
- Korrektheit
 - Verwendet die Konzepte des betreffenden Datenmodells in der richtigen Weise
- Minimalität
 - Jeder Aspekt der Anforderungen kommt nur einmal im Schema vor
- Lesbarkeit
 - Selbsterklärend
- Modifizierbarkeit
 - Modularer Aufbau, gut dokumentiert

Entwurfs-Phasen



Anforderungsanalyse

Requirements Analysis

- Informationsanforderungen
 - Welche statischen Informationen wird das DBS benutzen (Daten, Realwelt-Objekte, Typen, Attribute, Wertebereiche, Beziehungen ...)
 - Integritätsbedingungen
- Bearbeitungsanforderungen (Processing Requirements)
 - Spezifikation von dynamischen Aktivitäten und Prozessen
 - Abhängigkeiten zwischen Prozessen
 - Datenvolumen
- Typische Aktivitäten während der Anforderungsanalyse
 - Identifikation der wesentlichen Benutzergruppen und Anwendungsbereiche der zu entwerfenden Datenbank
 - Sichtung existierender Dokumentation
 - Fragebögen und Interviews

Konzeptioneller Entwurf

- 2 orthogonale Klassen von Strategien:
 - Top-down (schrittweise Verfeinerung)
 - Bottom-up (schrittweise Verallgemeinerung)
 - Zentralisiert (Globalsicht-orientiert)
 - Dezentralisiert (Einzelsicht-orientiert)
- Durch Kombination sind mindestens 4 Vorgehensweisen beim Entwurf möglich
- Beispiel:
Einzelsicht-orientiertes Bottom-up Vorgehen

Einzelsicht-orientiertes Bottom-up Vorgehen

Aufgabe des Designers:

- 1) Überführung der Beschreibungen der Benutzer in sogenannte Sichten (Views)
 - Für Modellierung der Sichten werden meist Datenmodelle verwendet, die von dem des Ziel-DBMS verschieden sind (wegen Einschränkungen DBMS)
 - für den konzeptionellen Entwurf üblicherweise: **Entity Relationship Modell (ER)**
- 2) Integration der Einzelsichten (View Integration) in eine konzeptionelle Globalsicht der Datenbank
 - z.B. Globalschema in ER-Terminologie oder
 - Konstruktion eines relationalen Schemas aus den initialen Schemata

Analyse Einzelsichten (Inkonsistenzen) und Misch-Vorgang

Auswahl des Zielsystems

Technische Einflussfaktoren:

- Benötigtes Datenmodell (Ausdruckskraft/Angemessenheit)
- Verfügbare Anfragesprachen
- Benötigte Programmiersprachen-Schnittstellen
- Leistung (Benchmark)

Ökonomische und organisatorische Aspekte:

- Herstellerbindung des betreffenden Unternehmens
- Vorhandene Hard- und Software (Kosten Neubeschaffung)
- Beschaffungs- und Wartungskosten des DBMS (TCO)
- Kosten einer Migration vom vorhandenen Altsystem
- Kosten für zusätzliches Personal (und Schulung auf neues System)

Politische Einflussfaktoren

Logischer Entwurf

Übersetzung konzeptionelles Schema in Datenmodell des verwendeten DBMS

- z.B. aus ER in relationales Modell übertragen
- Transformationsregeln sind wichtiges Hilfsmittel (falls diese existieren, bei Wahl Datenmodell konzeptioneller Entwurf darauf achten)
- Mehrere Teilschritte möglich, z.B.:
 - Transformation konzeptionelles Schema in Zielmodell
 - Optimierung des daraus resultierenden logischen Schemas (unter allgemeinen oder sich aus der Anforderungsanalyse ergebenden Kriterien)
 - Allgemeine Kriterien im Fall relationales Datenmodell: Normalisierung (Redundanzen vermeiden, folgt später)

Physischer Entwurf

Definition des internen Schemas sowie der damit zusammenhängenden Systemparameter

- Geeignete Speicherungsstrukturen für die einzelnen Elemente des konzeptionellen Schemas
- Zugriffsmechanismen
- Minimierung Zugriffszeiten (effiziente Zugriffspfade)
- Problem:
 - Datenmodell der konzeptionellen Ebene lässt sich meist nicht direkt in Speicherstrukturen umsetzen
 - Entscheidungen sind später während des Betriebs regelmäßig zu prüfen
 - Änderungen nennt man Tuning

Physischer Entwurf (2)

Aspekte des Physischen Entwurfs:

- Verwendbare Datei-Formate
- Block- bzw. Seitenzuweisung auf Platte
- Gruppierung von Blöcken zu Clustern und Verwaltung clusterübergreifender Records
- Indexauswahl
- Denormalisierung
(Relationsschema wurde zum Erreichen einer Normalform zerlegt, in Anfragen werden diese Schemas aber häufig verbunden -> Normalisierung eventuell rückgängig machen)

Weitere Entwurfsschritte

- Implementierung der verschiedenen Schemata
(unter Verwendung der DDL des Systems)
- Prototyping
(z.B. Laden einer Beispiel-Datenbank zum Zweck der Entwurfsverifikation, möglich sind u.a. Zufallsdaten)
- Dokumentation
(sollte den gesamten Prozess begleiten)

Die Phasen des Entwurfsprozesses sind für zentralisierte und verteilte Datenbanken nicht wesentlich voneinander verschieden.

Allgemeine Abstraktionskonzepte

Abstraktionsmechanismen im konzeptionellen Datenmodell:

- Klassifikation
 - Definition bestimmter Konzepte als Klassen von Dingen bzw. Objekten mit gemeinsamen Eigenschaften
- Aggregation (Zusammensetzung)
 - Definiert eine neue Klasse aus einer anderen, bereits existierenden oder setzt bereits bestehende Klassen zusammen
- Verallgemeinerung bzw. Spezialisierung
 - Definiert eine Teilmengenbeziehung zwischen den Elementen verschiedener Klassen

Entity-Relationship-Modell (ER)

- Chen P.P.-S., 1976. The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems*, 1, 9-36.

Verwendet für:

- Datenbankentwurf
- Modellierung von Realwelt-Zusammenhängen auf einer abstrakten Ebene

Hier:

- Einführung ER-Modell, soweit für Entwurf relationaler Datenbanken notwendig
- Viele inzwischen vorgenommene Erweiterungen werden nicht diskutiert

Vorteile des ER-Modells

- Unabhängig von einem bestimmten DBS
 - Keine Beschränkungen durch Systemimplementierung
 - z.B. speichert ein relationales DBS Relationen als Folgen von Tupeln, im Modell werden diese aber als Mengen betrachtet
- Grundkonstrukte: ***Entity*** und ***Relationship***



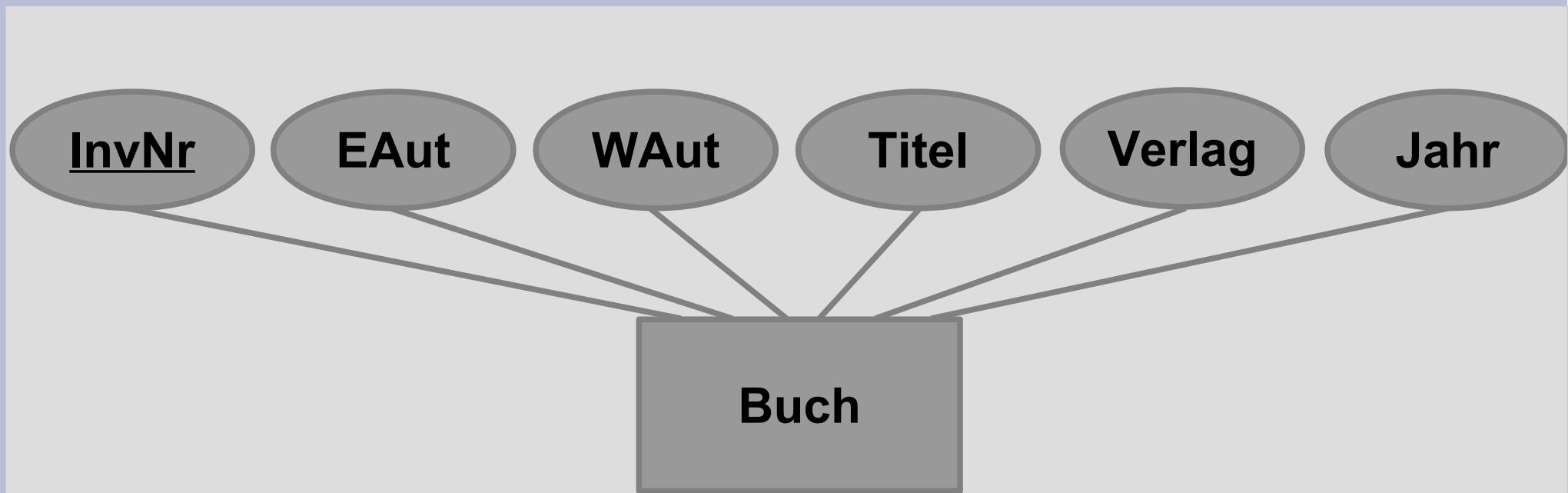
Als natürliches Ausdrucksmittel empfunden

In vielen Anwendungen bereits ausreichend betrachtet

Entities und Attribute

- ***Entities:***
 - Wohlunterscheidbare Dinge der realen Welt
 - Besitzen Eigenschaften, konkrete Ausprägung: Werte
 - Beispiele: Personen, Autos, Städte, Firmen
 - Bezeichnung: Kleinbuchstaben
- ***Entity-Set:***
 - Zusammenfassung einzelner Entities
 - Ähnlich, vergleichbar oder zusammengehörig
 - Attribute: die (bei allen Entities) auftretenden Werte
 - Beispiel: Alle Rechner eines Studenten
 - Bezeichnung: Großbuchstaben
- ***Wertebereich (Domain oder Value Set):***
 - Ausprägung aller möglichen bzw. zugelassenen Werte für eine Eigenschaft

Entity-Diagramm für Beispiel Buch



Attribut(name)

InvNr

Autor

Titel

Verlag

Domain

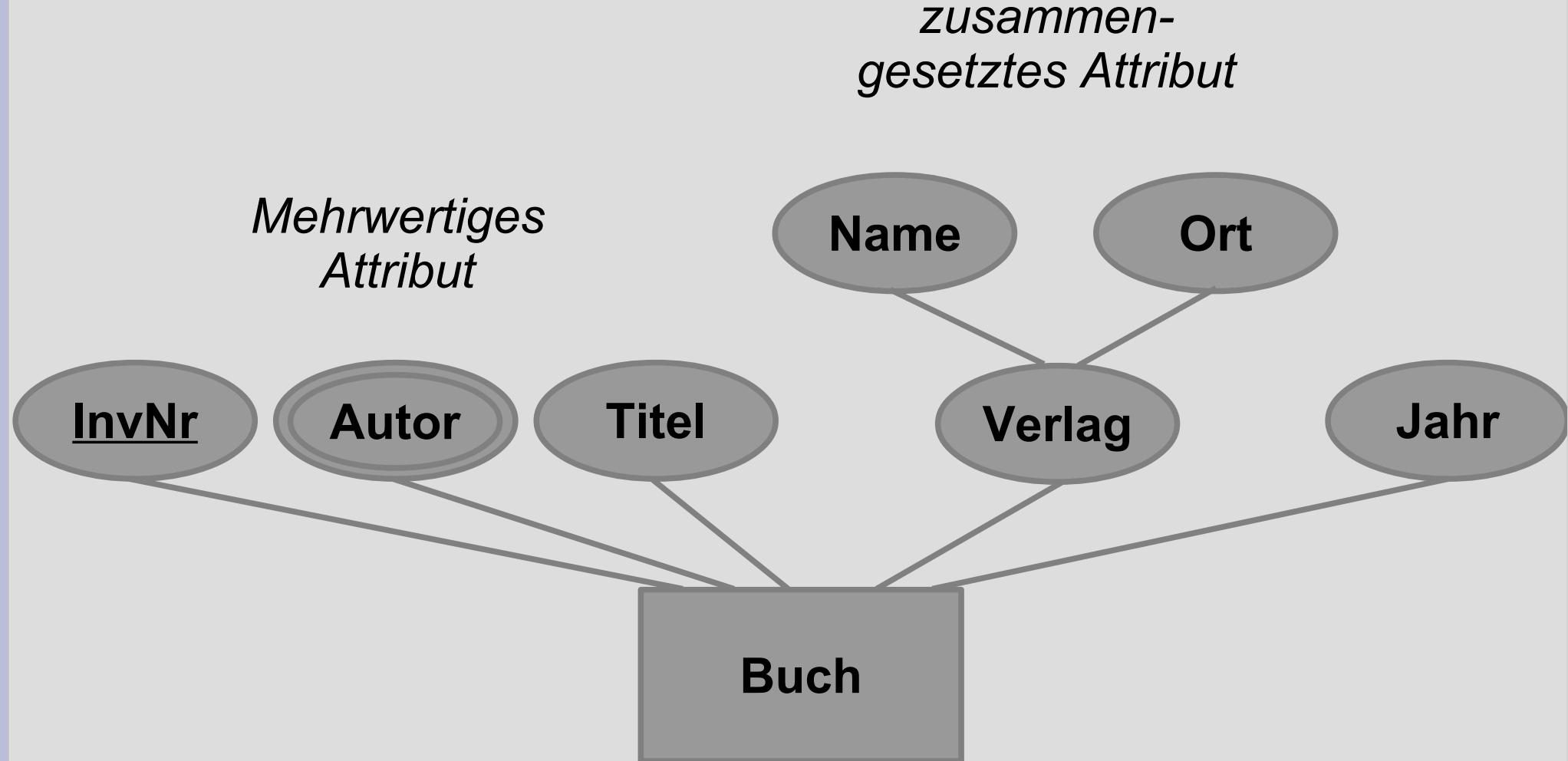
„vierstellige Zahlen“

„Characterstrings der Länge 12“

„Characterstrings der Länge 30“

„Characterstrings der Länge 15“

Entity-Diagramm für Beispiel Buch



Entity-Deklaration

Def.:

Eine Entity-Deklaration hat die Form $E = (X, K)$.

Sie besteht aus einem Format X und einem Primärschlüssel K , welcher aus (einwertigen) Elementen von X zusammengesetzt ist.

Die Elemente eines Formates X werden dabei wie folgt notiert:

- Einwertige Attribute: A
- Mehrwertige Attribute: $\{A\}$
- Zusammengesetzte Attribute: $A(B_1, \dots, B_k)$

Üblicherweise werden Entity-Deklarationen auch als Entity-Typ bezeichnet.

Kurzbezeichnungen für Domains

Def.:

Ist $E = (X, K)$ eine Entity-Deklaration, so bezeichne $\text{attr}(E)$ die Menge aller in X vorkommenden Attributnamen. Mit jedem $A \in X$, welches nicht einer Zusammensetzung voransteht, sei eine Wertemenge $W(A)$ assoziiert. Für jedes $A \in X$ sei:

$$\text{dom}(A) := \begin{cases} W(A) & \text{falls } A \text{ einwertig} \\ 2^{W(A)} & \text{falls } A \text{ mehrwertig} \\ W(B_1) \times \dots \times W(B_k) & \text{falls } A \text{ aus einwertigen} \\ & B_1 \dots B_k \text{ zusammengesetzt} \end{cases}$$

Darstellung des Formats X einer Entity-Deklaration E

Def.:

Es sei $E = (X, K)$ eine Entity-Deklaration mit $X = (A_1, \dots, A_m)$. Zu diesem A_i sei $\text{dom}(A_i)$ dessen Domain gemäß der vorigen Definition, $1 \leq i \leq m$.

- Ein Entity e ist ein Element des Kartesischen Produkts aller Domains, d.h.

$$e \in \text{dom}(A_1) \times \dots \times \text{dom}(A_m).$$

- Ein Entity-Set E^t (zum Zeitpunkt t) ist eine Menge von Entities, welche K erfüllt, d.h.

$$E^t \subseteq \text{dom}(A_1) \times \dots \times \text{dom}(A_m).$$

Wir bezeichnen E^t auch als den Inhalt bzw. als den aktuellen Wert (engl. Instance) des Typs E zur Zeit t .

Inhalt des Typs E zur Zeit t

Def. Format X einer Entity-Deklaration E :

Es sei $E = (X, K)$ eine Entity-Deklaration mit $X = (A_1, \dots, A_m)$. Zu diesem A_i sei $\text{dom}(A_i)$ dessen Domain gemäß der vorigen Definition, $1 \leq i \leq m$.

- Ein Entity e ist ein Element des Kartesischen Produkts aller Domains, d.h.

$$e \in \text{dom}(A_1) \times \dots \times \text{dom}(A_m).$$

- Ein Entity-Set E^t (zum Zeitpunkt t) ist eine Menge von Entities, welche K erfüllt, d.h.

$$E^t \subseteq \text{dom}(A_1) \times \dots \times \text{dom}(A_m).$$

Wir bezeichnen E^t auch als den Inhalt bzw. als den aktuellen Wert (engl. Instance) des Typs E zur Zeit t .

Beispiel für Entity-Typ *Buch*

Zu einem Zeitpunkt t könnte folgendes Entity-Set vorliegen:

Mit: $Buch^t = \{b_1, b_2, b_3\}$

$b_1 = (123, \{\text{'Vossen'}, \text{'Witt'}\}, \text{'DB2 Handbuch'}, (\text{'Addison-Wesley'}, \text{'Bonn'}), 1990)$

$b_2 = (125, \{\text{'Vossen'}, \text{'Witt'}\}, \text{'SQL/DS Handbuch'}, (\text{'Addison-Wesley'}, \text{'Bonn'}), 1988)$

$b_3 = (130, \{\text{'Witt'}\}, \text{'OO Programmierung'}, (\text{'Oldenbourg'}, \text{'München'}), 1992)$

Nicht vorkommen darf z.B.:

$b_4 = (123, \{\text{'Vossen'}\}, \text{'Transaktionsverarbeitung'}, (\text{'Hüthig'}, \text{'Heidelberg'}), 1990)$

(es würde den Schlüssel InvNr verletzen)

Zusammenfassung Entities

- Entity-Sets
 - Klassifikation von Objekten mit gemeinsamen Eigenschaften
 - Beschreibung der zeitinvarianten Struktur:
Entity-Typ (Attribute und ein Schlüssel)
- Entities
 - Aggregationen von Werten auffassen
 - Entity-Typen: Aggregationen von Attributen
 - Zusammengesetzte Attribute: Aggregationen anderer Attribute
- Entity-Deklarationen sind vollständig graphisch darstellbar
 - Entity-Deklaration als Rechteck mit Namen
 - Attribute als Kreise oder Ovale, mit Kante mit der zugehörigen Deklaration verbunden
 - (Primär-) Schlüssel unterstrichen
 - Wertebereiche nicht dargestellt
 - Mehrwertige Attribute in Doppelkreise eingeschlossen

Relationships

Def.:

- Eine Relationship-Deklaration hat die Form $R = (\text{Ent}, Y)$. Dabei ist R der Name der Deklaration, Ent bezeichnet die Folge der Namen der Entity-Deklarationen, zwischen denen eine Beziehung definiert werden soll, und Y ist eine (möglicherweise leere) Folge von Attributen (der Beziehung).

(Fortsetzung nächste Folie)

Relationships (2)

Def. (Fortsetzung):

- Sei Ent = (E_1, \dots, E_k) , und für beliebiges, aber festes t sei E_i^t der Inhalt der Entity-Deklaration E_i , $1 \leq i \leq k$. Ferner sei $Y = (B_1, \dots, B_n)$. Ein Relationship r ist das Element des Kartesischen Produktes aus allen E_i^t und den Domains der B_j , d.h.

$$r \in E_1^t \times \dots \times E_k^t \times \text{dom}(B_1) \times \dots \times \text{dom}(B_n)$$

bzw.

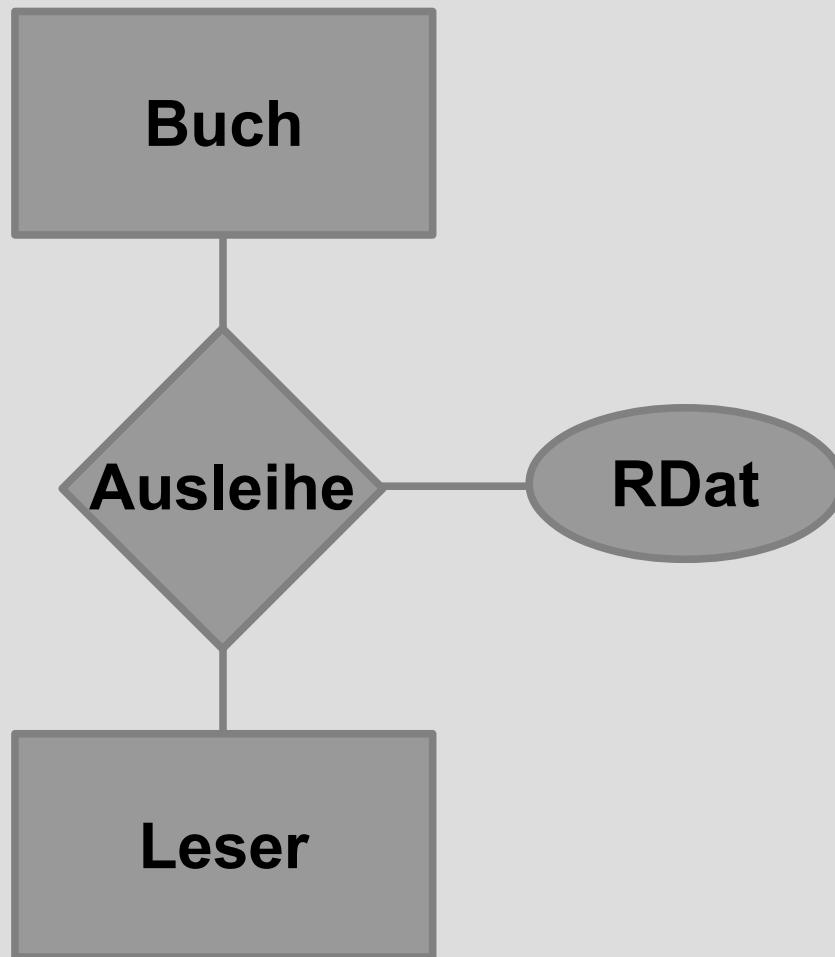
$$r = (e_1, \dots, e_k, b_1, \dots, b_n)$$

mit

$$e_i \in E_i^t \text{ für } 1 \leq i \leq k \text{ und } b_j \in \text{dom}(B_j) \text{ für } 1 \leq j \leq n$$

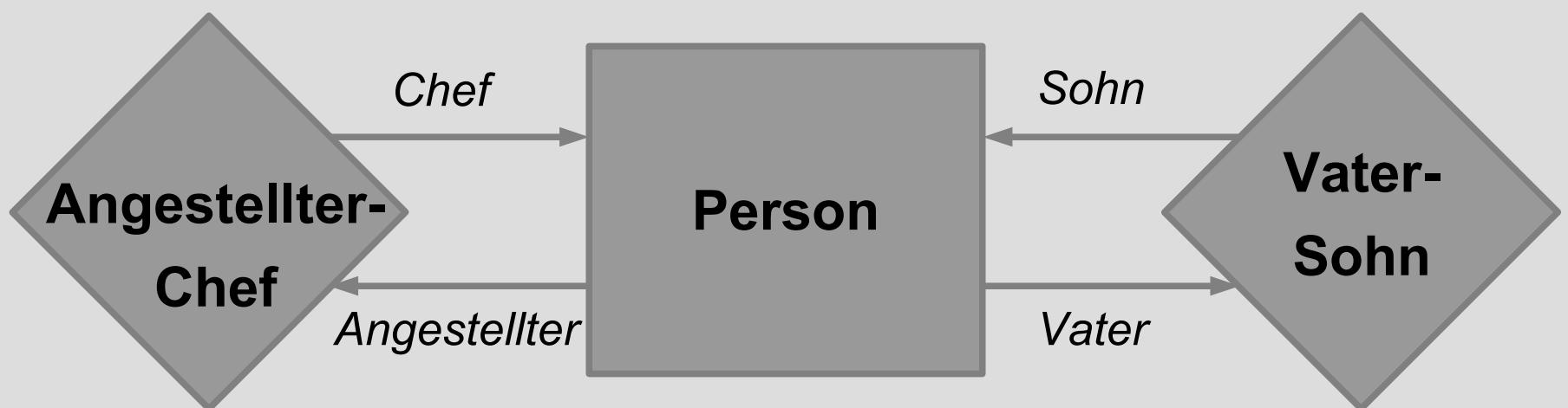
- Ein Relationship-Set R^t zur Zeit t ist eine Menge von Relationships
$$R^t \subseteq E_1^t \times \dots \times E_k^t \times \text{dom}(B_1) \times \dots \times \text{dom}(B_n)$$

Beziehung zwischen Büchern und Lesern



Relationship-Deklaration:
Raute mit dem Namen der Deklaration
eventuelle Attribute in Kreisen und mit Kanten mit der Raute verbinden

Rekursive Beziehung zwischen Personen

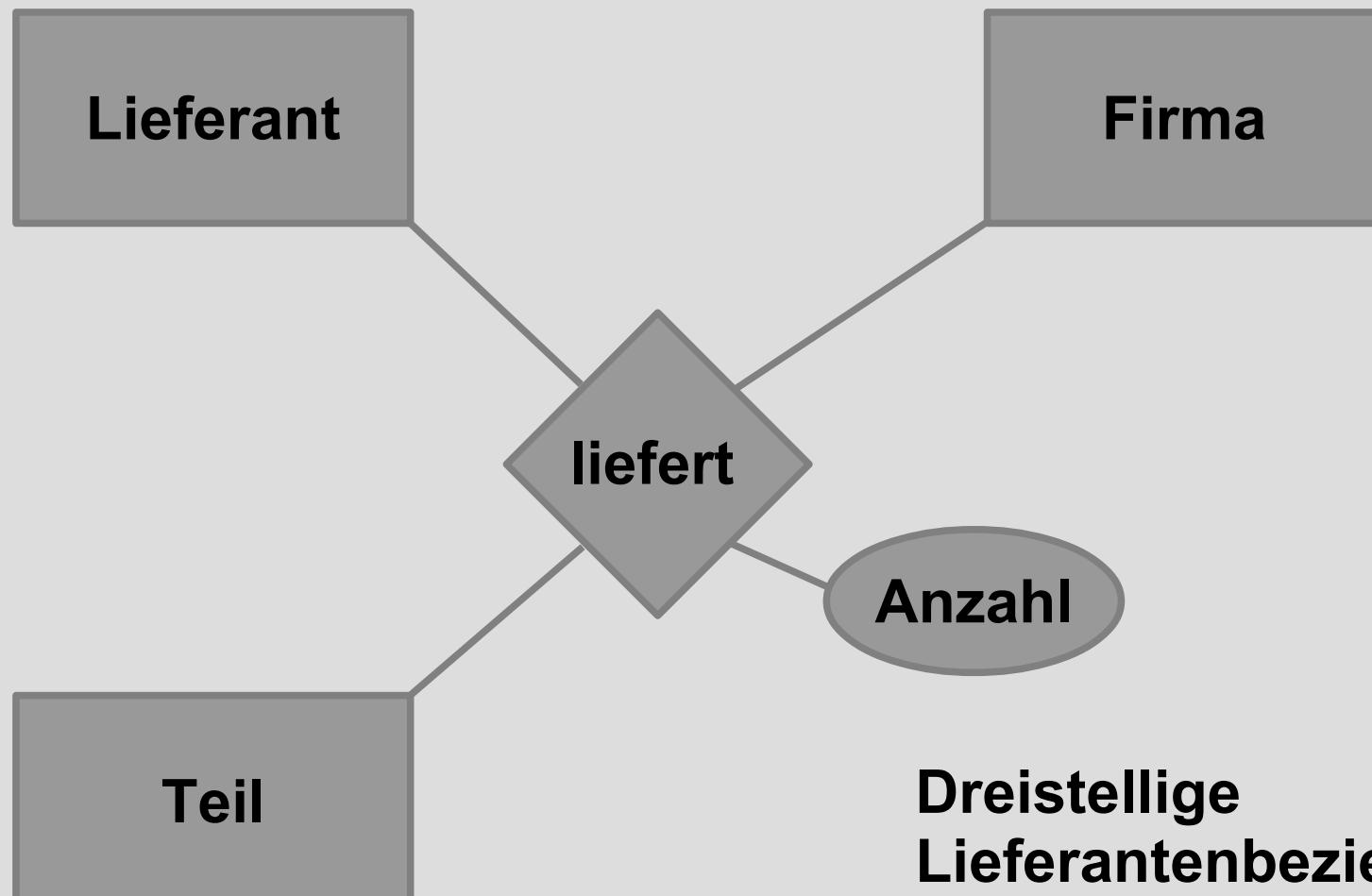


Vereinfachungen

Darstellung konkreter Relationships oder Relationship-Sets:

- Überflüssige Klammern werden weggelassen
- Für alle an einem Relationship beteiligten Entities ist ein (Primär-)Schlüssel vereinbart
- Für die Darstellung eines Relationships reicht die Angabe der Schlüsselwerte der jeweiligen Entities (da diese damit eindeutig festgelegt sind)

Stelligkeit einer Relationship-Deklaration



**Dreistellige
Lieferantenbeziehung
 $\text{grad}(R)$**

Komplexität

$$\text{comp}(R, E_i) = (m, n) : \Leftrightarrow (\forall t)(\forall e_i \in E_i^t) |\{r \in R^t | r[E_i] = e_i\}| \begin{cases} \geq m \\ \leq n \end{cases}$$

Beispiel der Lieferantenbeziehung:

Relationship-Deklaration:

liefert = (*Lieferant*, *Teil*, *Firma*), (*Anzahl*)

$\text{comp}(\text{liefert}, \text{Lieferant}) = (0, \infty)$

$\text{comp}(\text{liefert}, \text{Teil}) = (0, 1)$

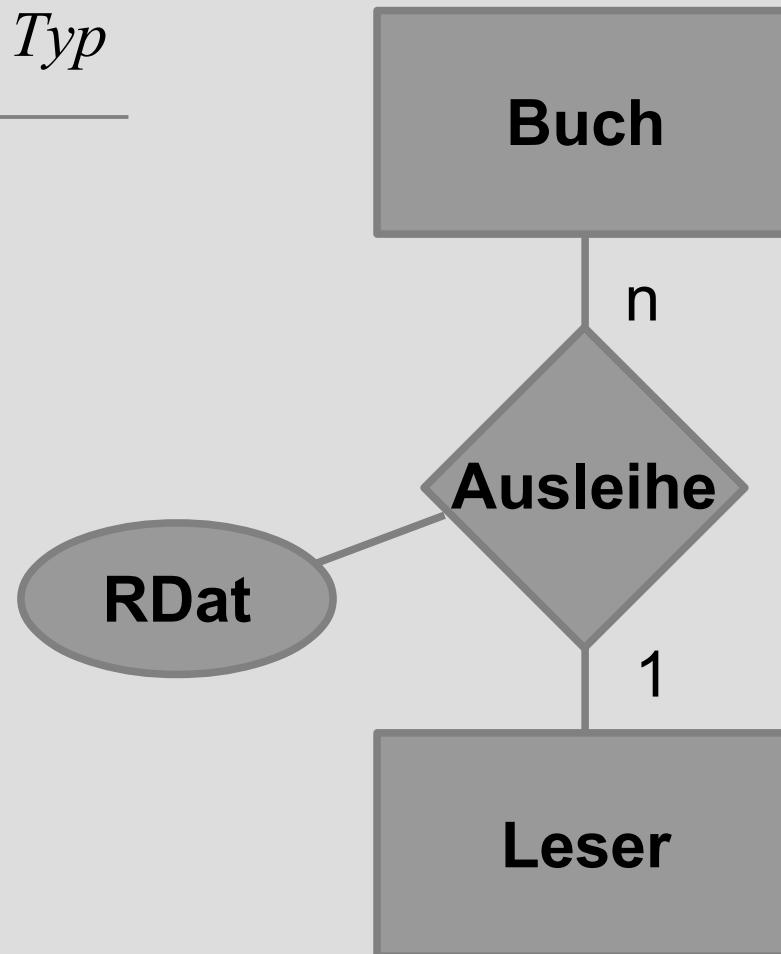
$\text{comp}(\text{liefert}, \text{Firma}) = (1, 3)$

Komplexitätstypen für zweistellige Beziehungen

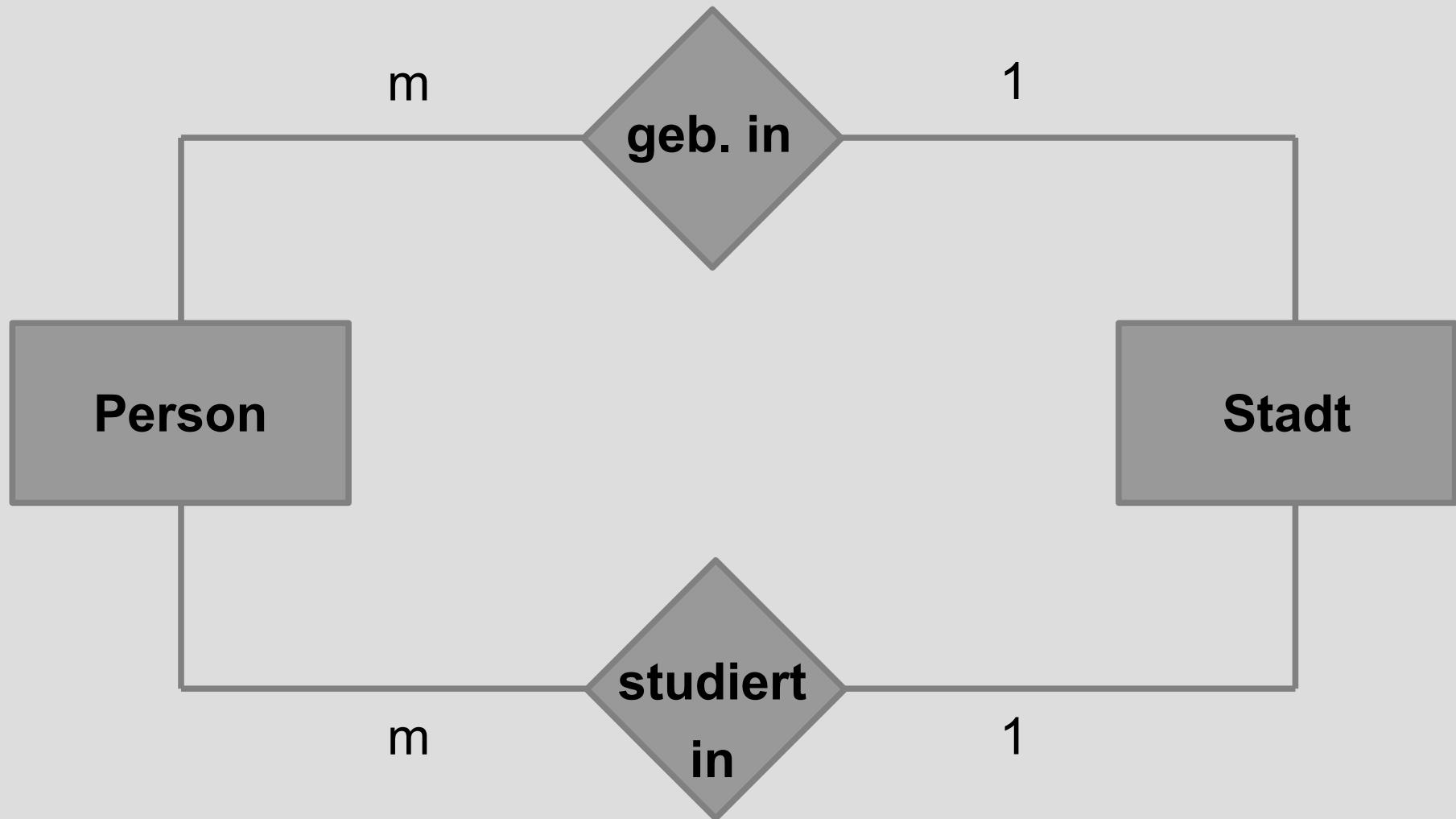
$comp(R, E_1) \in$	$comp(R, E_2) \in$	R ist vom Typ
$\{(0,1), (1,1)\}$	$\{(0,1), (1,1)\}$	$1:1$
$\{(0,\infty), (1,\infty)\}$	$\{(0,1), (1,1)\}$	$1:n$
$\{(0,\infty), (1,\infty)\}$	$\{(0,\infty), (1,\infty)\}$	$m:n$

Problem:

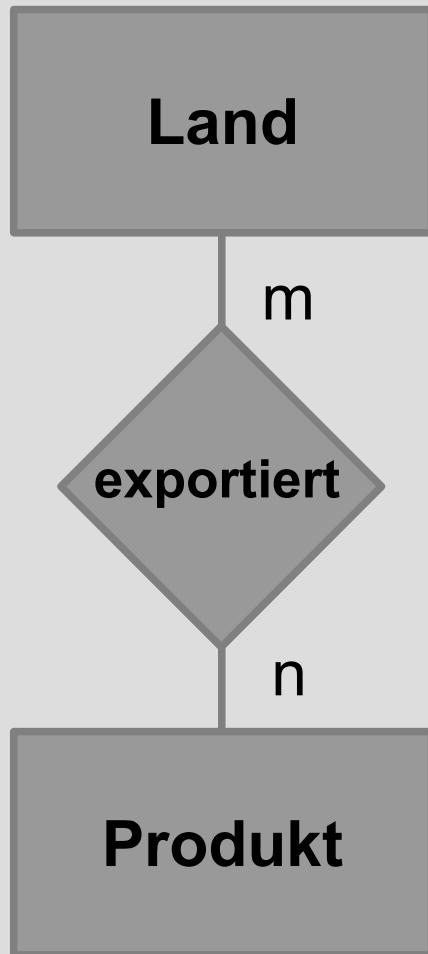
$1:n$ oder $n:m$ gibt keine oberen Schranken für n und m an.
(Im Gegensatz zur comp-Notation)



Beispiel Many One Beziehungen



Beispiel Many Many Beziehung



Keine Restriktionen an die Entity-Paare eines Relationship-Sets

IS-A Beziehungen

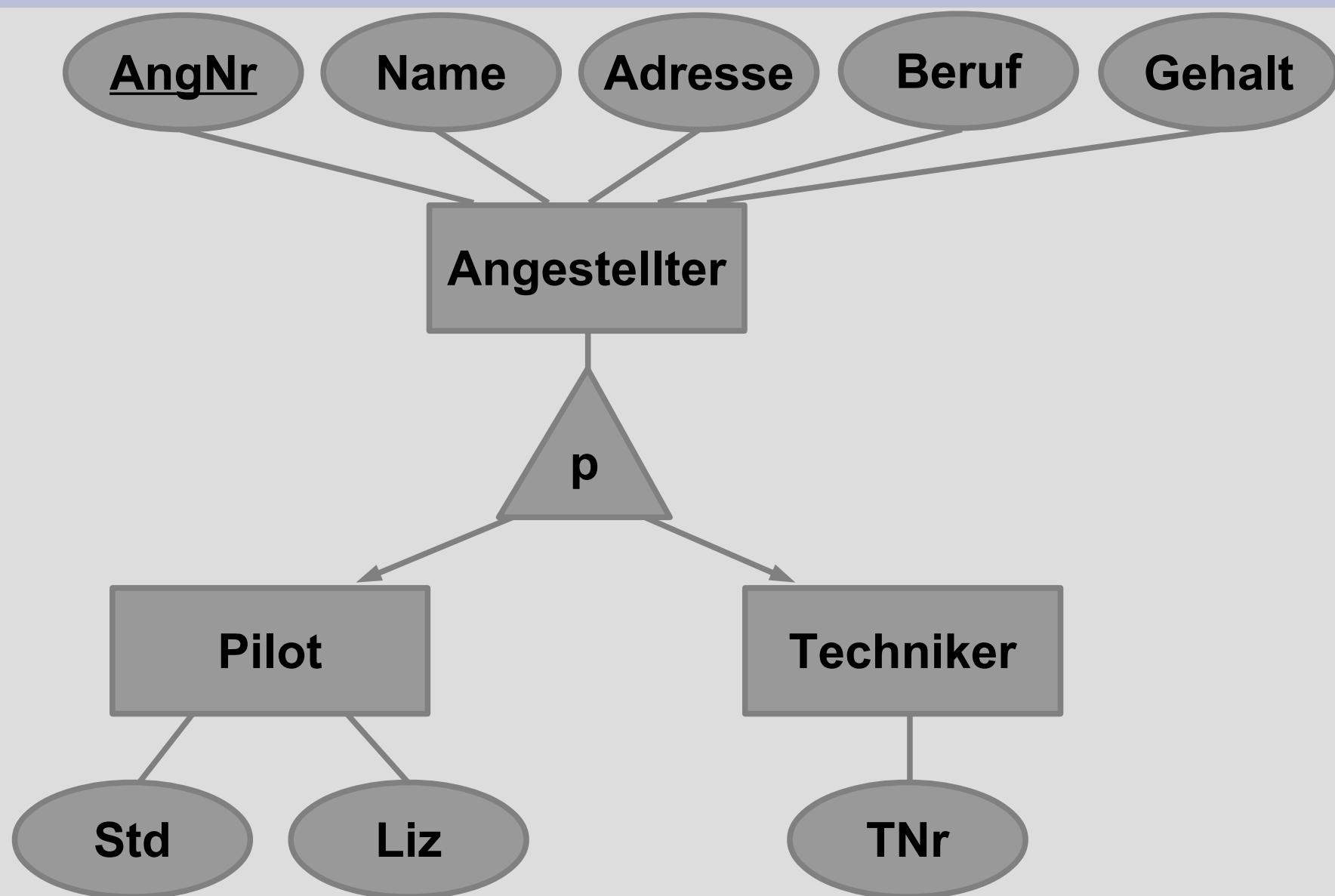
Def.:

Sind $E_1 = (X_1, K_1)$ und $E_2 = (X_2, K_2)$ zwei Entity-Deklarationen, so besteht zwischen diesen eine IS-A Beziehung (der Form $E_1 \text{ IS-A } E_2$) falls gilt:

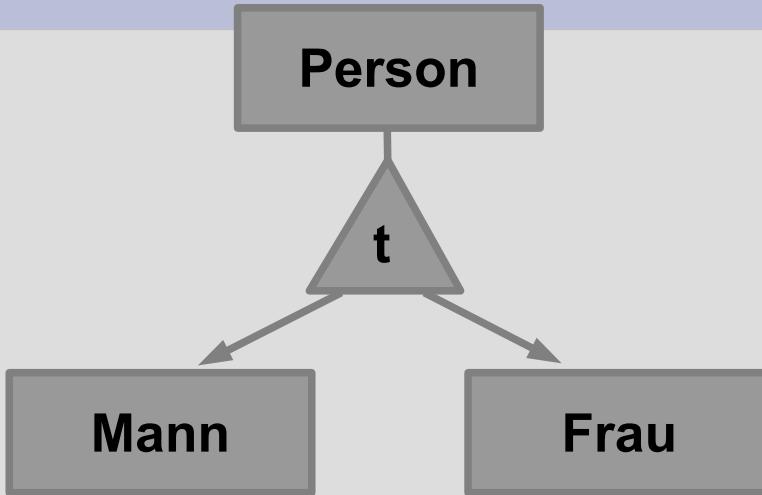
- Alle Elemente von X_2 kommen in X_1 vor;
- Zu jedem Zeitpunkt t gilt: Für jedes $e_1 \in E_1^t$ existiert ein $e_2 \in E_2^t$ mit $e_1(A) = e_2(A)$ für jedes Attribut $A \in X_2$.

Wir schreiben auch kurz $E_1 \subseteq E_2$.

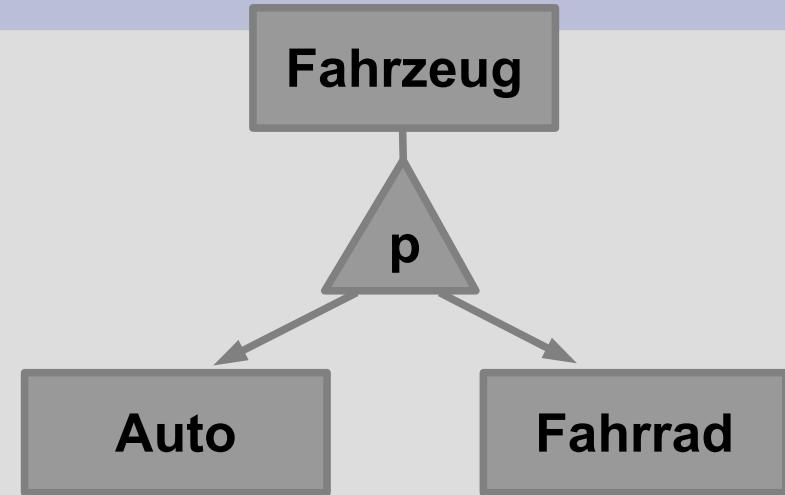
IS-A Beziehung



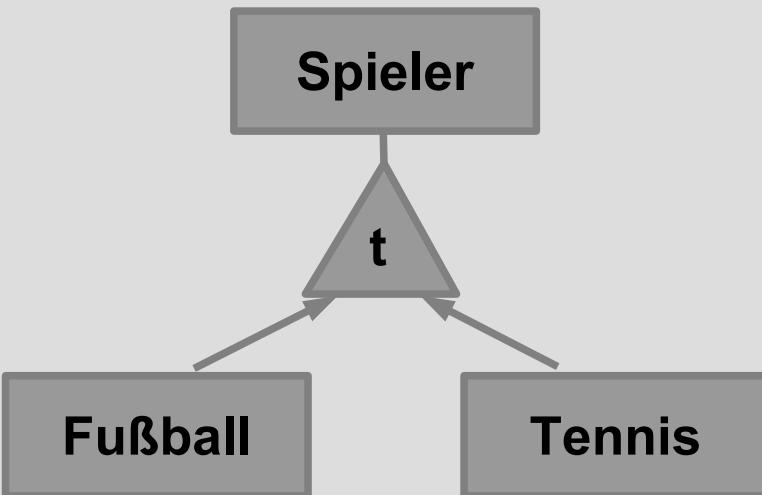
Spezialisierung einer Entity-Deklaration



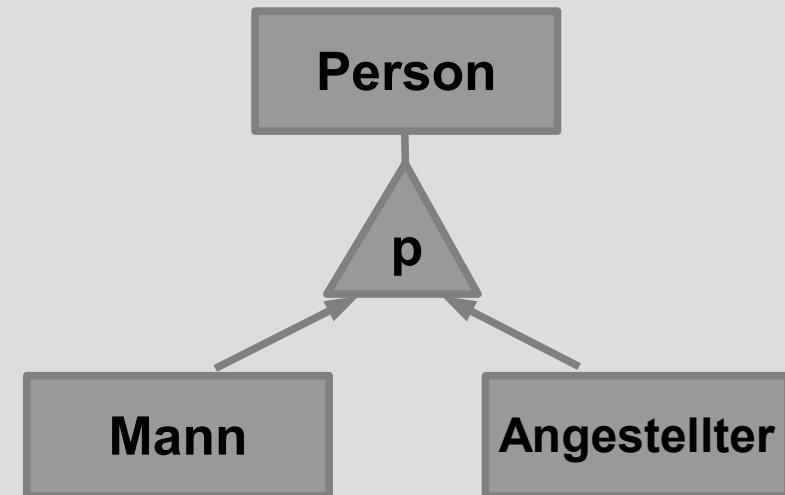
total, disjunkt



partiell, disjunkt



total, nicht disjunkt



partiell, nicht disjunkt

Totale und disjunkte Spezialisierungen

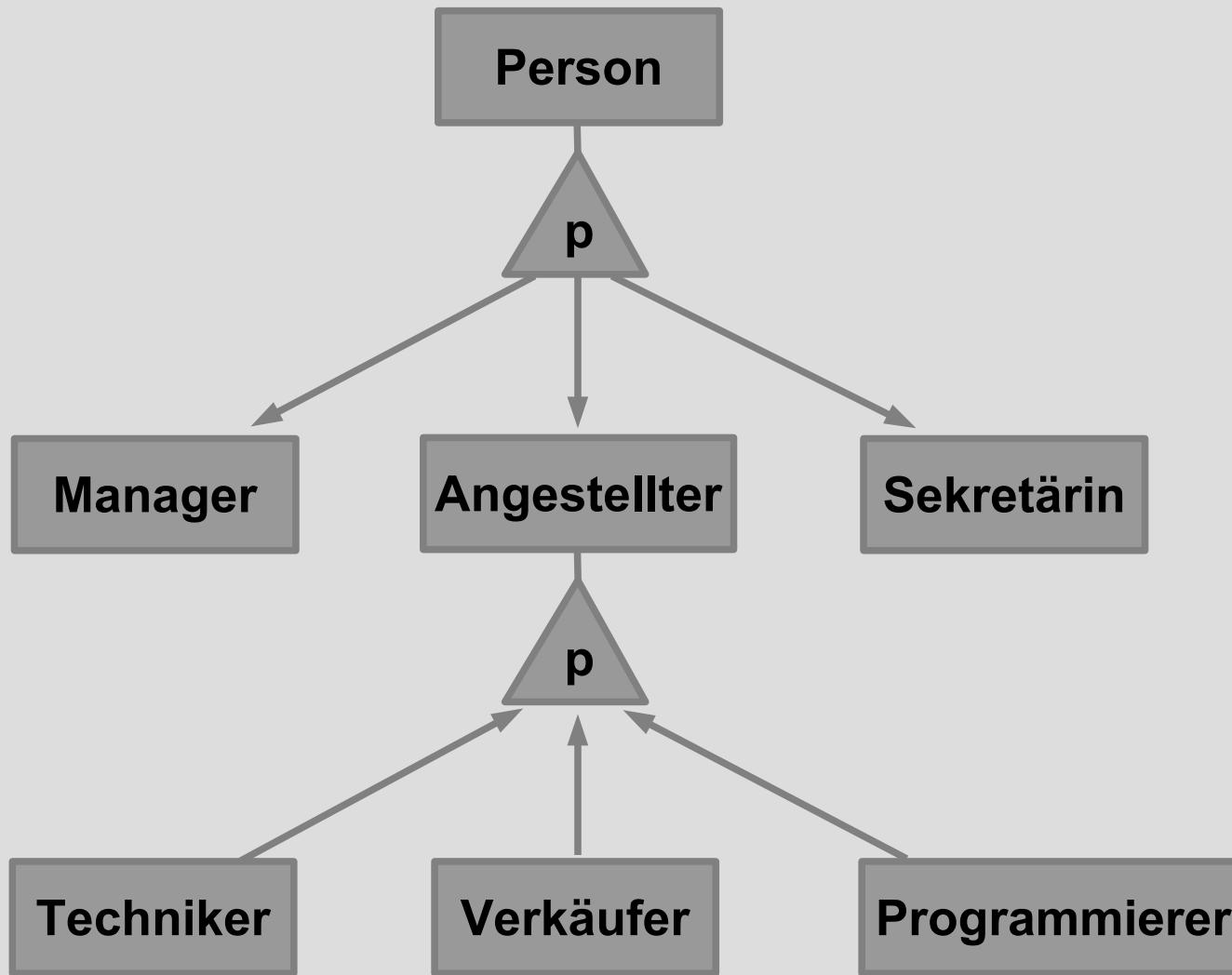
Def.:

Es seien E, E_1, \dots, E_k Entity-Deklarationen, $k \geq 2$, und es gelte, dass alle E_i , $1 \leq i \leq k$ zu derselben Spezialisierung von E gehören (also insbesondere E_i IS-A E für $1 \leq i \leq k$). Die IS-A Beziehung heißt:

- Total, falls gilt:
$$(\forall t) \cup_{i=1}^k E_i^t = E^t$$
- Disjunkt, falls gilt:

$$(\forall t)(\forall i, j \in \{1, \dots, k\}, i \neq j) E_i^t \cap E_j^t = \emptyset$$

Spezialisierungshierarchie



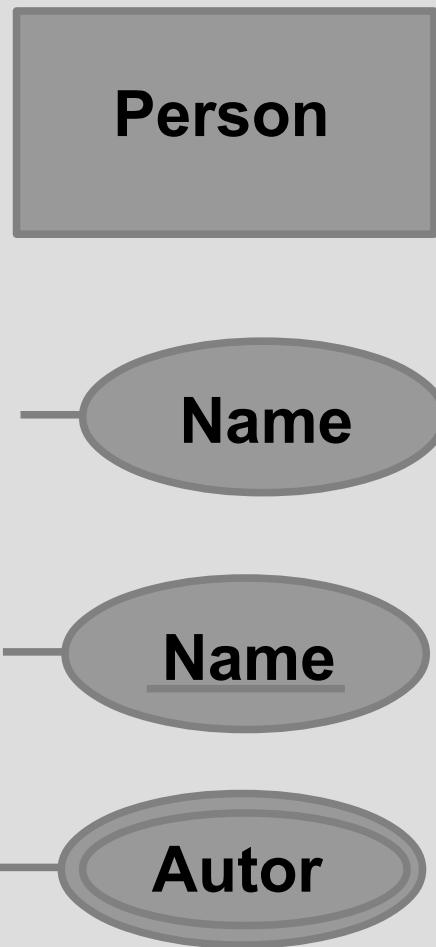
Zusammenfassung ER-Modell

- Entity-Deklarationen mit
 - Namen
 - einwertigen, mehrwertigen und zusammengesetzten Attributen und deren Wertebereichen
 - Primärschlüssel
- IS-A Beziehungen als Spezialisierungen von Entity-Deklarationen mit Typ-Festlegung
 - partiell / total
 - disjunkt / nicht disjunkt
- Relationship-Deklarationen mit
 - Namen
 - beteiligten Entity-Deklarationen
 - ggf. eigenen Attributen und deren Wertebereichen
 - Komplexitäts- bzw. Typfestlegung

Konzeptioneller Entwurf mit dem ER-Modell – Top-Down Vorgehensweise

- Entity-Verfeinerung
 - Ersetzen durch neue Typen mit relevanten Beziehungen
 - Spezialisierung in Subtypen
 - Zerlegung in voneinander unabhängige Typen, die weder in Beziehung stehen noch Spezialisierungen sind
 - Entity-Typ mit Attributen versehen, Primärschlüssel
- Relationship-Verfeinerung
 - Zerlegung in zwei oder mehr Relationships
 - Relationship durch Folge von Beziehungen ersetzen (eventuell weitere Entity-Typen einbeziehen)
 - Relationship mit Attributen versehen
- Attribut-Verfeinerung
 - Ersetzen durch mehrere Attribute
 - Ersetzen durch zusammengesetztes Attribut

Grafische Notation lokaler ER-Konstrukte

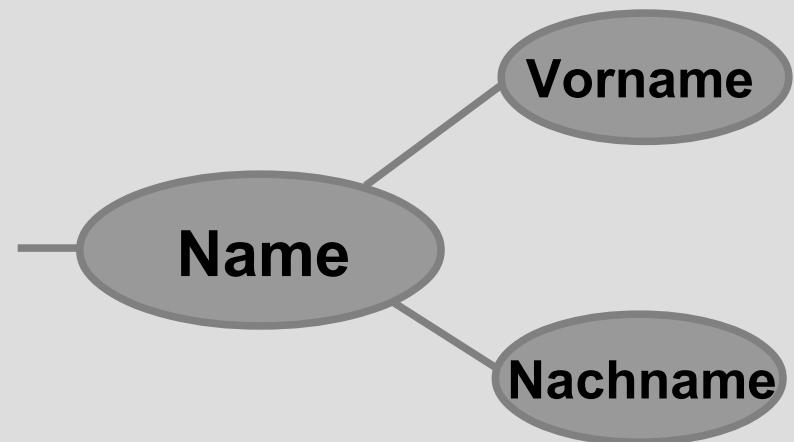


Entität (Entity)

Attribut

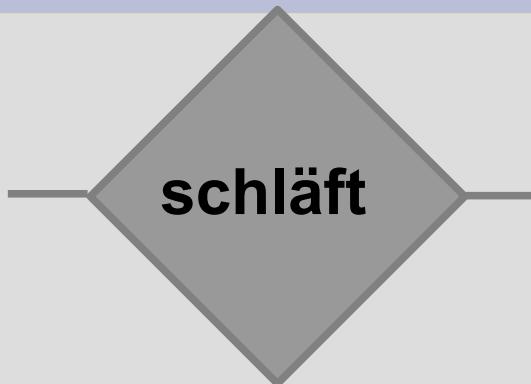
Schlüsselattribut

**mehrwertiges
Attribut**

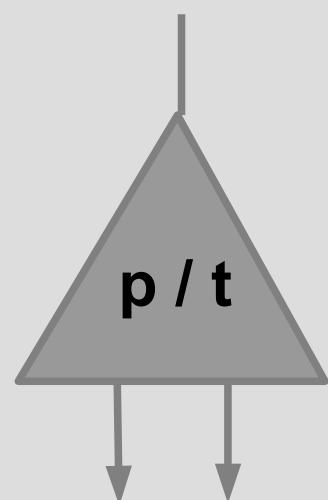


**zusammengesetztes
Attribut**

Grafische Notation lokaler ER-Konstrukte (2)



Relationship

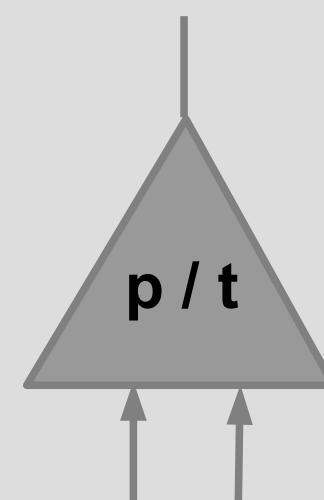


disjunkt

IS-A Beziehung

p – partiell

t - total



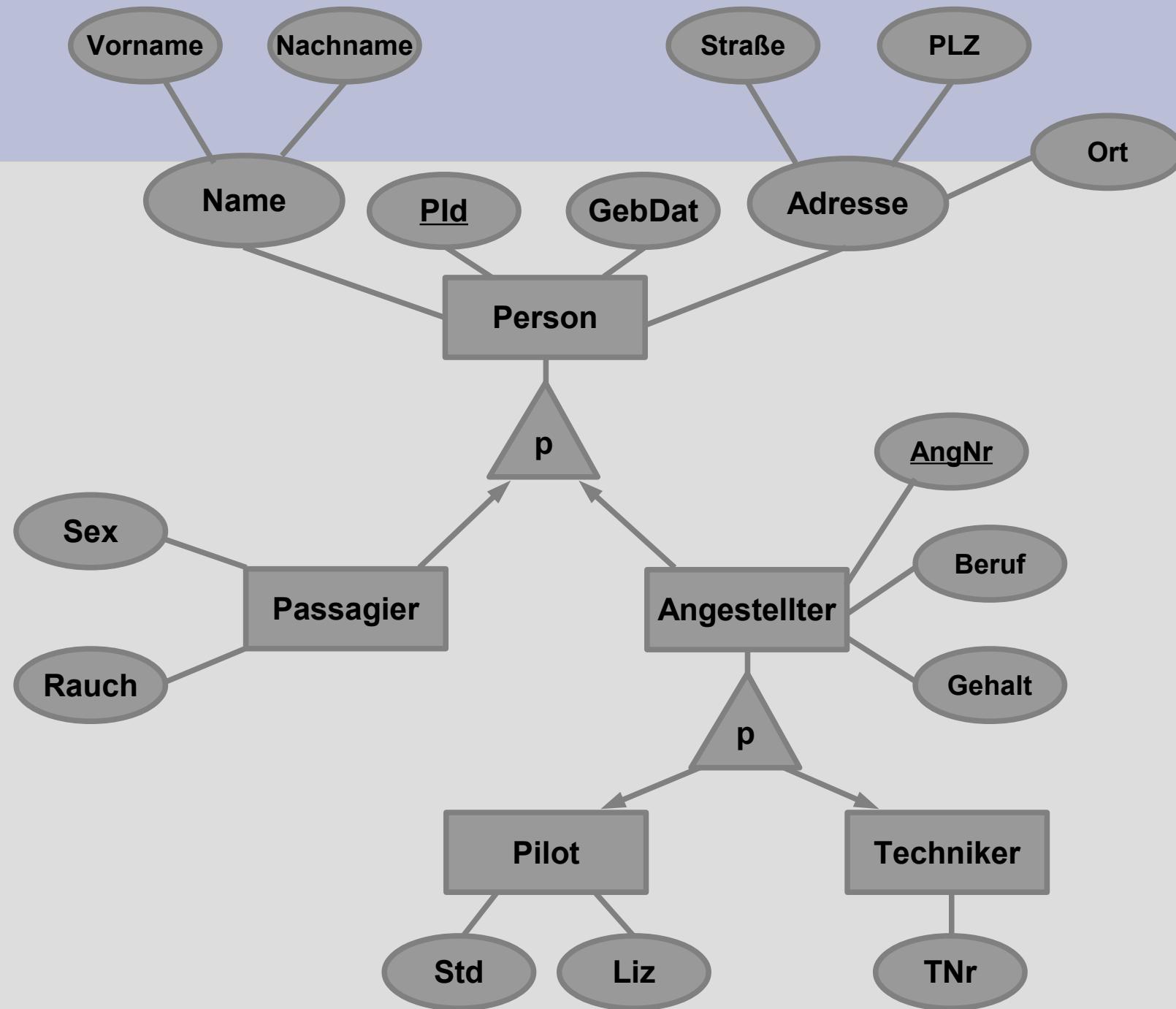
nichtdisjunkt

Beispiel Fluggesellschaft

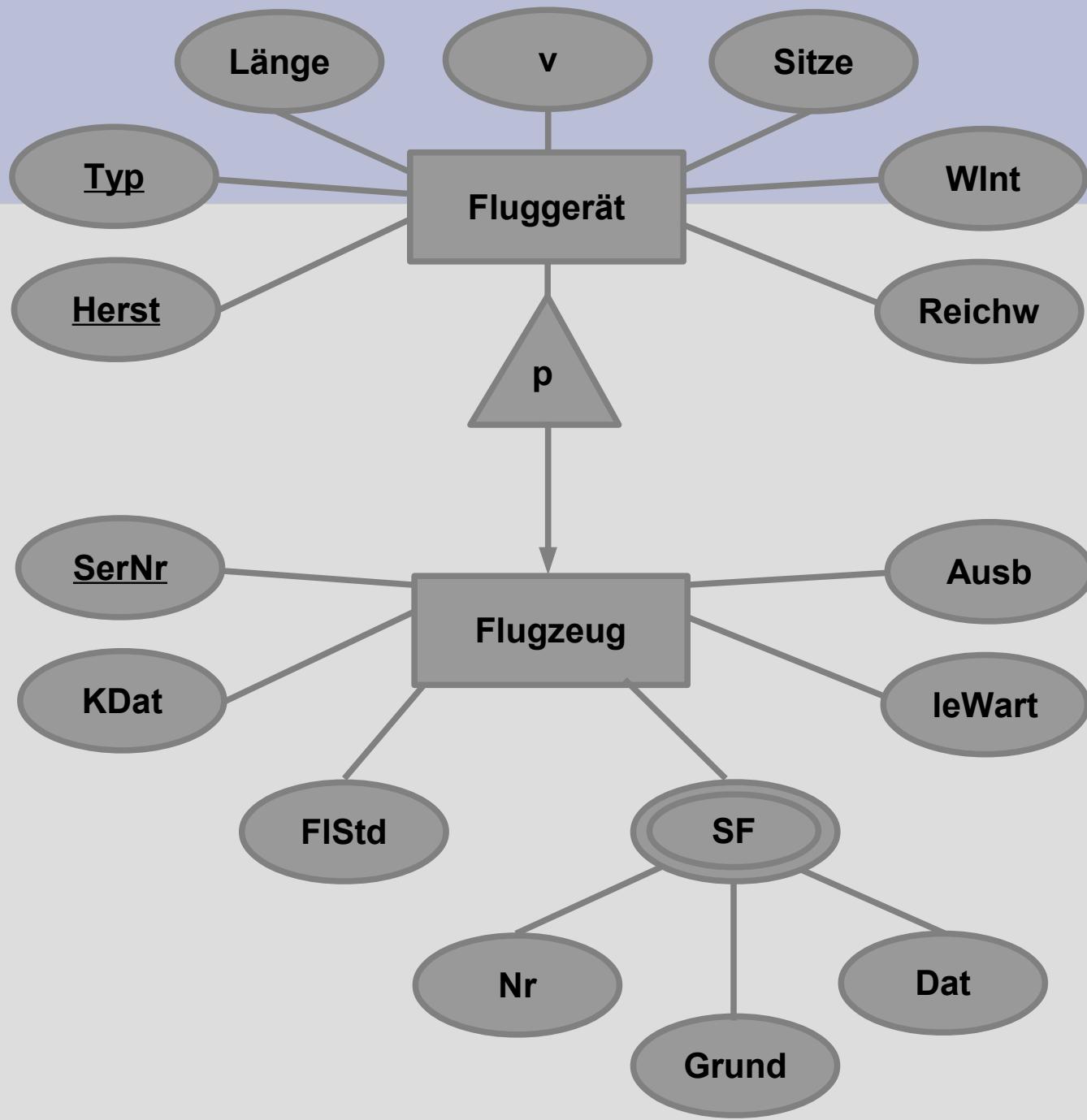
Folgende Daten sind zu speichern (Anforderungen):

- Personen
 - Passagiere
 - Angestellte
- Flugzeuge
- Flüge

Fluggesellschaft



Fluggesellschaft (2)



Qualitätsmerkmale ER-Diagramm

- Vollständigkeit
 - Schwierig zu prüfen (nur Vergleich Anforderungsanalyse mit ER Diagramm ist möglich)
- Korrektheit
 - Syntaktische Korrektheit
 - Semantisch korrekt
- Minimalität
- Lesbarkeit
 - Übersichtliche Anordnung / Größen der Symbole
 - Spezialisierungshierarchien beginnen mit dem allgemeinsten Typ
 - Symmetrien werden betont
 - Diagramm möglichst kreuzungfrei
- Modifizierbarkeit

Aufgabe 1: Supermarkt

Supermarkt-Datenbank:

- Angestellte:
 - Nummer, Name, Adresse, Abteilung
- Abteilung:
 - Name, die Angestellten, der Manager, angebotene Artikel
- Verkaufte / angebotene Artikel:
 - Bezeichnung, Lieferant, Preis, Lieferanten-Waren-nummer, Supermarkt-Warennnummer
- Lieferant:
 - Name, Adresse, an Supermarkt gelieferte Waren mit Preisen

Aufgabe:

Erstellen Sie das ER-Diagramm der Supermarkt-Datenbank.

Bereiten Sie sich darauf vor, Ihr ER-Diagramm in einem 5-minütigen Vortrag zu entwickeln und zu diskutieren.

Aufgabe 2: Bibliothek

Bibliotheks-Datenbank:

- Leser:
 - Nummer, Name, Adresse, Herkunft (Seminargruppe / Sonst)
- Buch:
 - Titel, Autor(en), Erscheinungsjahr, Verlag, ISBN
- Ausleihe:
 - Rückgabedatum, Leser, Verlängerung
- Sponsor:
 - Name, Adresse, an Bibliothek gesponsorte Werke mit Preis

Aufgabe:

Erstellen Sie das ER-Diagramm der Bibliotheks-Datenbank.

Bereiten Sie sich darauf vor, Ihr ER-Diagramm in einem 5-minütigen Vortrag zu entwickeln und zu diskutieren.

Aufgabe 3: Autovermietung

Autovermietungs-Datenbank:

- Kunde:
 - Nummer, Name, Adresse
- Fahrzeug:
 - Klasse (PKW, LWK, Motorrad ...), Typ, Farbe, Mietpreis
- Ausleihe:
 - Rückgabedatum, Mieter, Verlängerung, Schäden
- Hersteller:
 - Name, Adresse, Fahrzeug, Sonderkonditionen für Vermieter

Aufgabe:

Erstellen Sie das ER-Diagramm der Autovermietungs-Datenbank.

Bereiten Sie sich darauf vor, Ihr ER-Diagramm in einem 5-minütigen Vortrag zu entwickeln und zu diskutieren.

Das relationale Datenmodell

- Geht auf E.F. Codd zurück
- Seit Mitte der 80er Jahre de-facto Standard
- Sprache des relationalen Modells:
 - Relationen
 - Integritätsbedingungen
 - Schemata
- Herleitung einer relationalen Datenbankbeschreibung
 - ER-Diagramm
 - Schemadefinition mit Sprache SQL

Beispiel einer relationalen Datenbank

Buch	InvNr	ErstAutor	WeitereAut	Titel	...
	123	Date	n	Intro DBS	
	234	Jones	y	Algorithms	
	345	King	n	Operating Syst.	

Ausleihe	InvNr	LeserNr	Rückdat
	123	225	22-04-1998
	234	347	31-07-1998

Leser	LeserNr	Name	...
	225	Peter	
	347	Laura	

Tupel und Relation

Def.

- Ein Tupel über X ist eine (bis auf weiteres totale) Abbildung:

$$\mu: X \rightarrow \text{dom}(X) \text{ für die gilt } (\forall A \in X) \mu(A) \in \text{dom}(A)$$

Zu gegebener Attributmenge X bezeichne dann $\text{Tup}(X)$ die Menge aller Tupel über X ; man beachte, daß es sich bei $\text{Tup}(X)$ um eine Menge (ohne doppelte Elemente) handelt.

- Eine Relation r über X ist eine (endliche) Menge von Tupeln über X , d.h. $r \subseteq \text{Tup}(X)$
Mit $\text{Rel}(X)$ bezeichnen wir die Menge aller Relationen über X .

Partielle Relation

- Nullwertebereich $N = \{\delta_1, \delta_2, \delta_3, \dots\}$
 $(\forall A \in X) \ dom(A) \cap N = \emptyset$
- Partielles Tupel X ist eine Abbildung:
 $\mu : X \rightarrow dom(X) \subset N$ für die gilt:
 $(\forall A \in X) \mu(A) \in dom(A) \vee \mu(A) \in N$
- Relation heißt partiell, falls sie partielle Tupel enthält

Intrarelationale Datenabhängigkeiten

Beispiel: Primärschlüssel

Def.

Sei X eine Attributmenge, $K \subseteq X$.

- 1) K heißt Schlüssel für $r \in \text{Rel}(X)$, falls gilt:
 - (a) $(\forall \mu, \nu \in r) \quad \mu[K] \rightarrow \mu = \nu$
 - (b) für keine echte Teilmenge $K' \subset K$ gilt (a).
- 2) Eine Schlüsselabhängigkeit $K \rightarrow X$

bezeichnet folgende semantische Bedingung:

Sei $r \in \text{Rel}(X)$:

$$(K \rightarrow X)(r) := \begin{cases} 1 & \text{falls } K \text{ Schlüssel für } r \\ 0 & \text{sonst} \end{cases}$$

Intrarelationale Abhängigkeiten (2)

Def:

Ein Relationenschema hat die Form $R = (X, \Sigma_x)$, es umfaßt einen Namen (R), eine Attributmenge bzw. ein Relationenformat X und einen Menge Σ_x intrarelationaler Datenabhängigkeiten.

Falls aus dem Zusammenhang heraus klar ist, welche Menge X gemeint ist, schreiben wir auch Σ statt Σ_x .

Relationale Datenbanken

Def.

Es sei $\mathcal{R} = \{ R_1, \dots, R_k \}$ eine (endliche) Menge von Relationenschemata (in 1NF) der Form $R_i = (X_i, \Sigma_i)$ für $1 \leq i \leq k$.

- Eine (relationale) Datenbank d (über \mathcal{R}) ist eine Menge von (Basis-) Relationen, $d = \{ r_1, \dots, r_k \}$, mit $r_i \in \text{Rel}(X_i)$ für $1 \leq i \leq k$. $\text{Dat}(\mathcal{R})$ bezeichne die Menge aller Datenbanken über \mathcal{R} .
- Eine Datenbank $d \in \text{Dat}(\mathcal{R})$ heißt punktweise konsistent, falls $r_i \in \text{Sat}(R_i)$ gilt für alle $r_i \in d$. $\text{Sat}(\mathcal{R})$ bezeichne die Menge aller punktweise konsistenten Datenbanken über \mathcal{R} .

Interrelationale Abhangigkeiten

- Relationen stehen i.A. Zueinander in Beziehung
- Formal: Abbildung σ von $\text{Dat}(\mathcal{R})$ in $\{0, 1\}$
 - $\sigma(d) = 1$, falls die Datenbank d die Abhangigkeit erfillt
- Eine Menge interrelationale Abbildungen ist dann ebenfalls eine solche Abbildung:
 - $\Sigma_{\mathcal{R}} : \text{Dat}(\mathcal{R}) \rightarrow \{0, 1\}$
 - Datenbank erfillt $\Sigma_{\mathcal{R}}$, falls jede Abhangigkeit erfillt wird

Inklusionsabhängigkeiten

- Erlauben insbesondere die Darstellung von IS-A-Beziehungen
 - Ist 1, falls in beiden beteiligten Relationen zwei Attribute den gleichen Wert haben
 - Sonst 0
- Englisch: Inclusion Dependency (IND)

Transformation ER-Diagramm in Relationenmodell

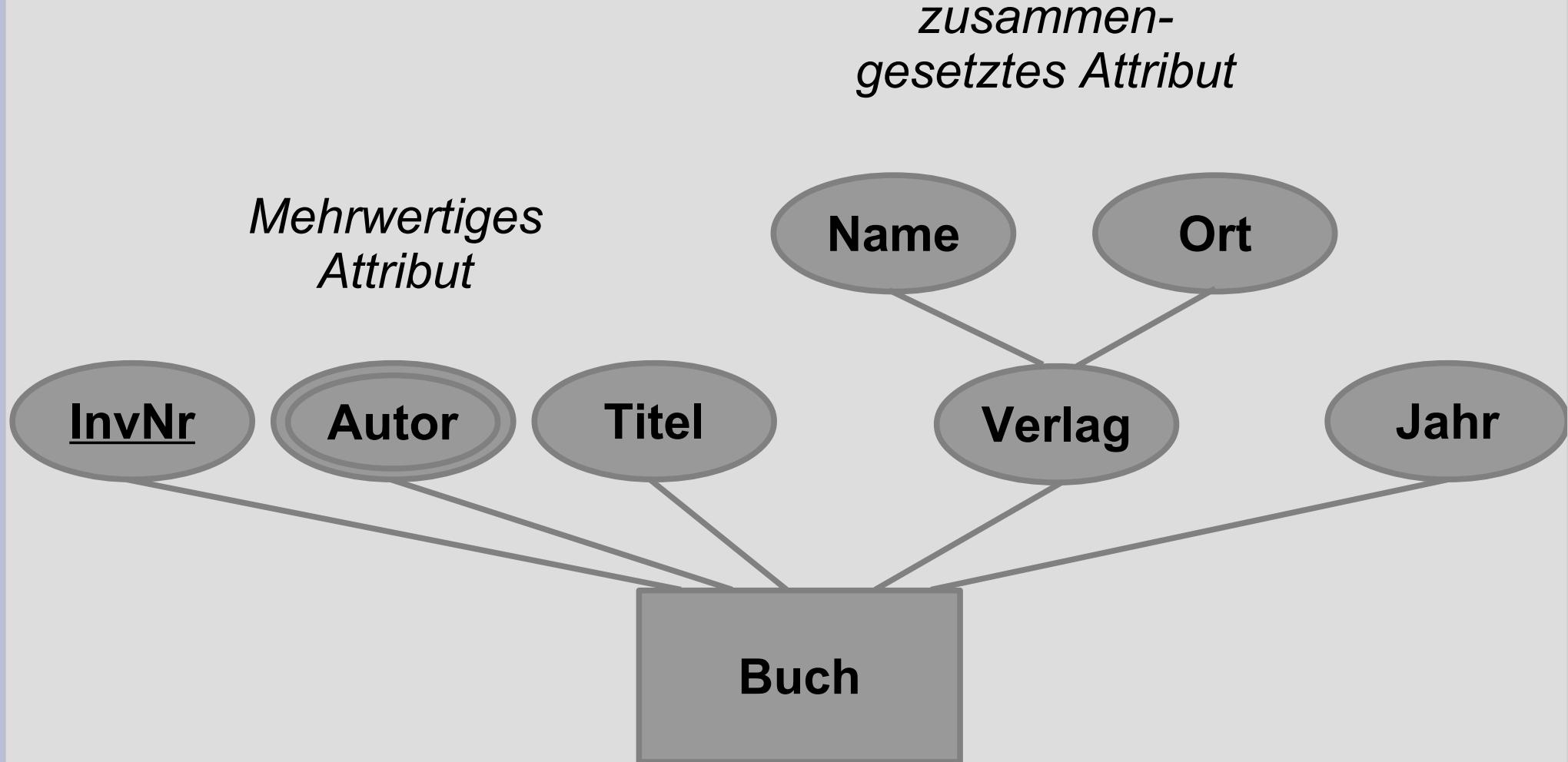
Vorgehensweise / Transformationsregeln:

- Jeder Entity-Typ wird in ein Relationenschema transformiert
- Jeder Relationship-Typ wird ebenfalls in ein Relationenschema transformiert

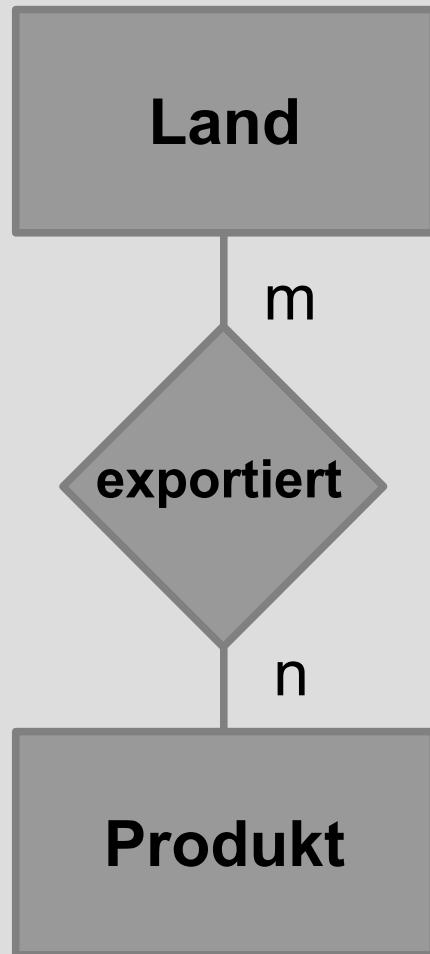
Ausnahmen:

- Zweistellige 1:1 oder 1:n Beziehung (hier reicht die Hinzunahme von Attributen zu bereits existierenden Relationenschemata)
- IS-A Beziehungen werden allein über INDs ausgedrückt

Entity-Diagramm für Beispiel Buch



Beispiel Many Many Beziehung



Keine Restriktionen an die Entity-Paare eines Relationship-Sets

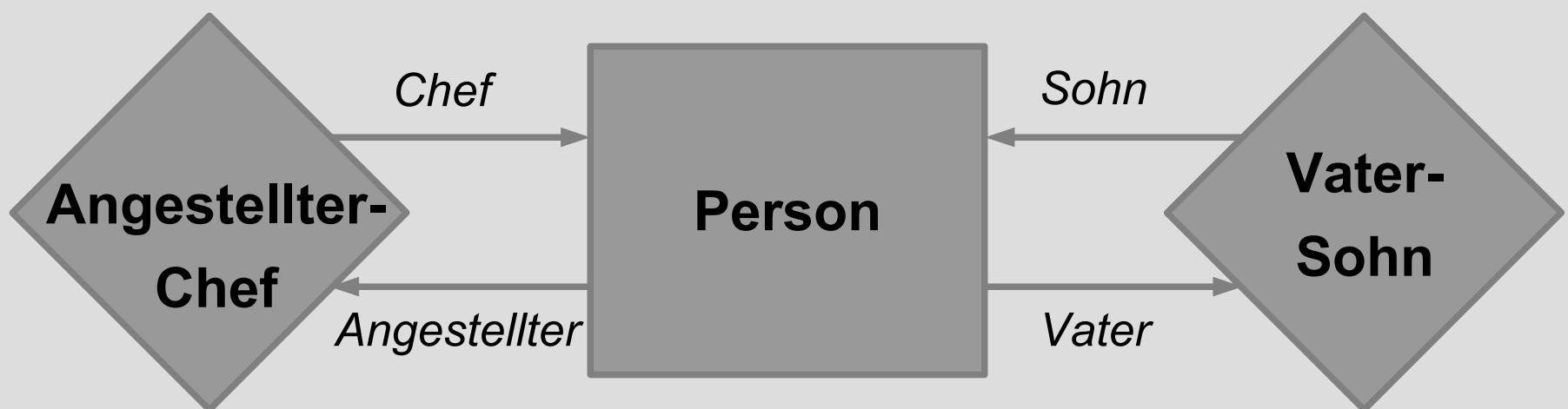
Beispiel einer relationalen Datenbank

Buch	InvNr	ErstAutor	WeitereAut	Titel	...
	123	Date	n	Intro DBS	
	234	Jones	y	Algorithms	
	345	King	n	Operating Syst.	

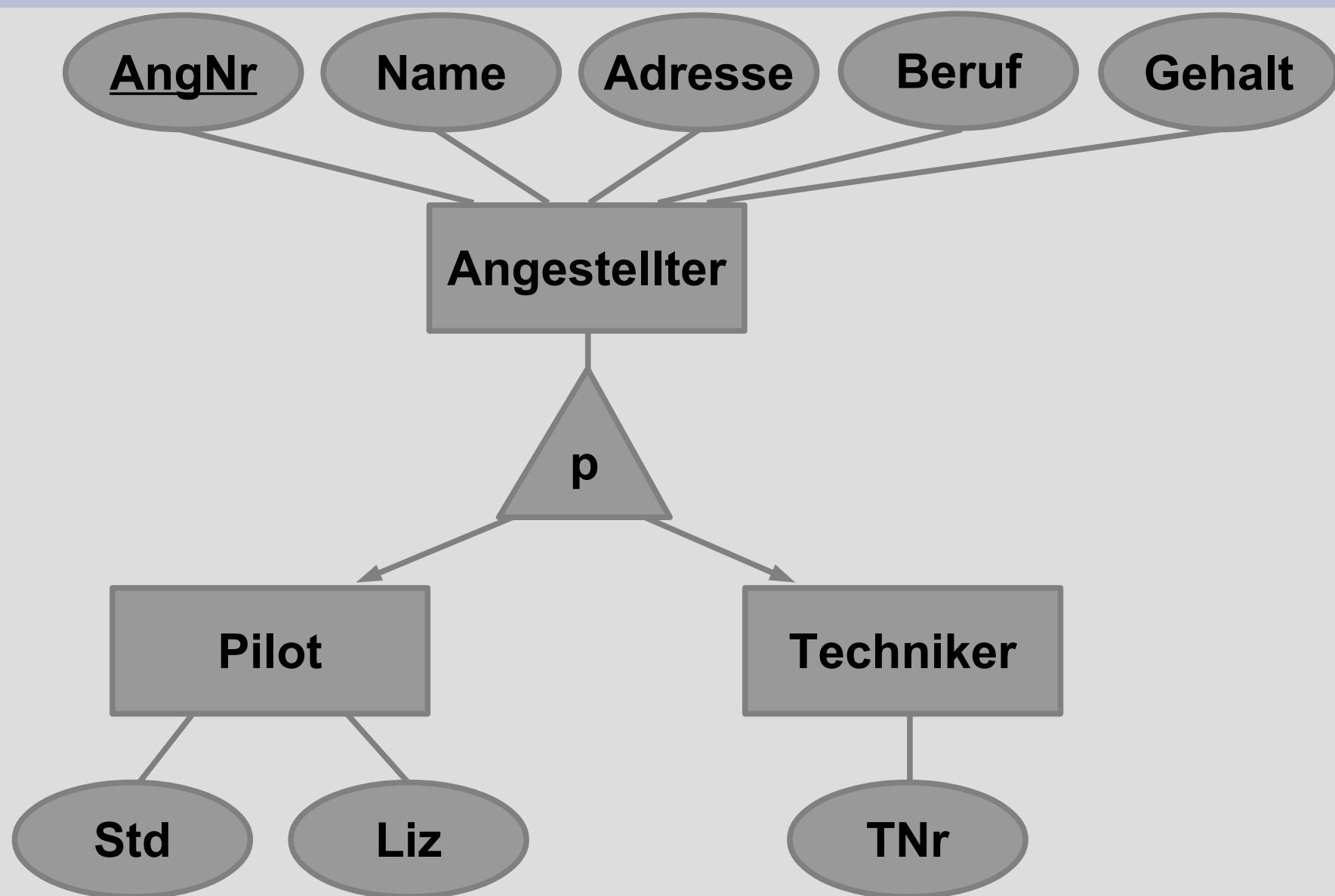
Ausleihe	InvNr	LeserNr	Rückdat
	123	225	22-04-1998
	234	347	31-07-1998

Leser	LeserNr	Name	...
	225	Peter	
	347	Laura	

Rekursive Beziehung zwischen Personen

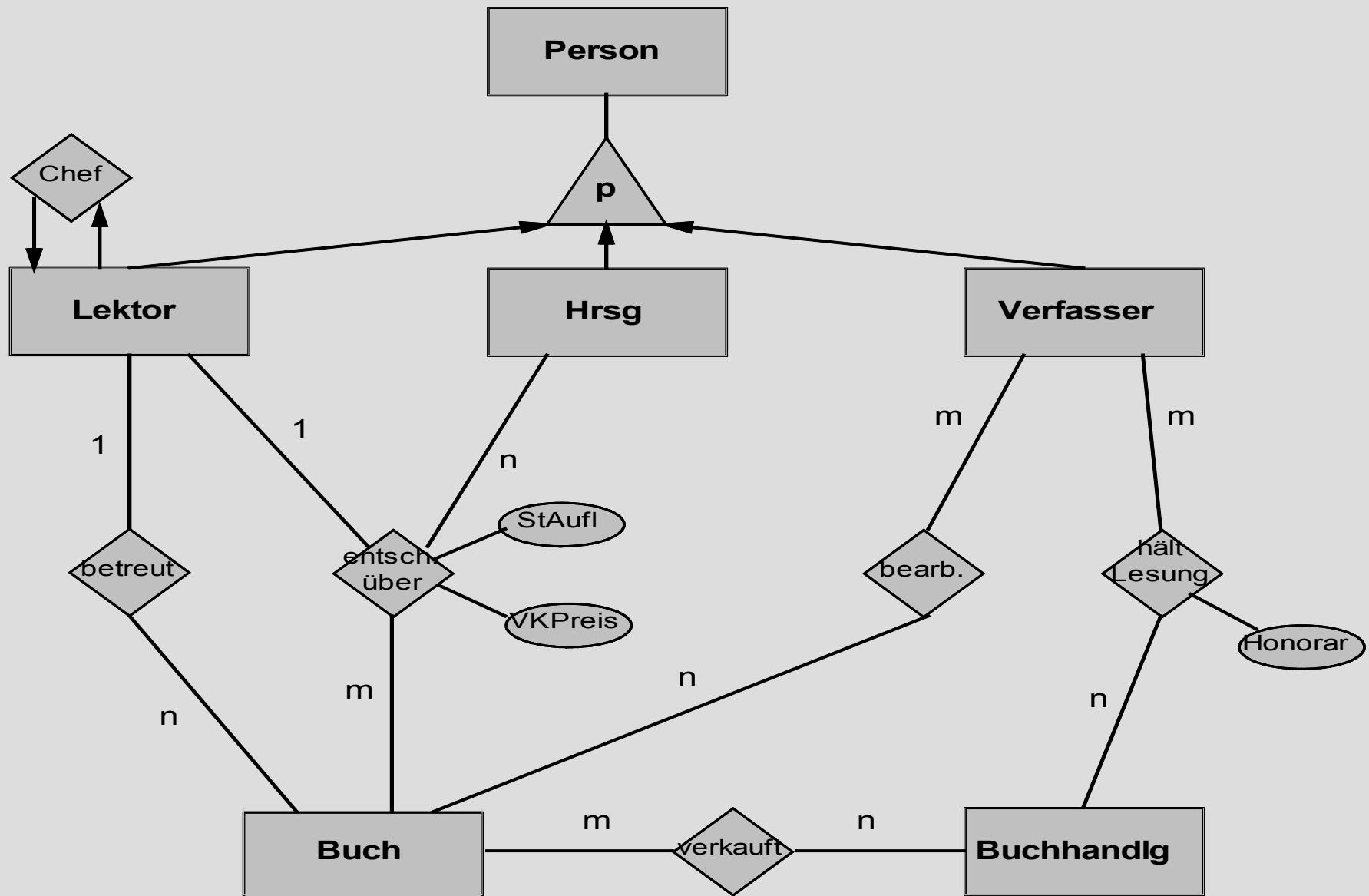


IS-A Beziehung



Übung: Buchverlag

(siehe separates Dokument)



5 Integrität in relationalen Datenbanken

- Arten von Integritätsbedingungen
- Funktionale Abhängigkeiten
- Allgemeine Implikation
- Inklusionsabhängigkeiten
- Assertions und Trigger
- Aktive Datenbanken

Arten von Integritätsbedingungen

Integritätskontrolle: Verhinderung semantischer Fehler bzw. semantisch unsinniger Zustände

Möglichkeiten Zuständigkeit für Integritätskontrolle:

- Anwendungs-Designer oder Benutzer
 - DBMS besitzt keine Kontrolle über Konsistenz
- Transaktions-Designer
 - Transaktionsprogramme sorgen für Konsistenz
 - Selten verwendet
- DBMS
 - Spezieller Monitor im DBMS
 - Weist Transaktionen zurück, die Integritätsbedingungen verletzten

Arten von Integritätsbedingungen

- Statische Bedingungen
 - Einschränkung der Menge der möglichen Datenbankzustände gegenüber den Möglichkeiten des Schemas
- Transitionale (oder halb-dynamische) Bedingungen
 - Einschränkung der möglichen Zustandsübergänge
 - Also Art und Weise, von einem gegebenen zu einem neuen Zustand zu gelangen
- Dynamische Integritätsbedingungen
 - Verallgemeinerung der Transitionalen Bedingungen
 - Beziehen sich nicht auf genau 2, sondern mehrere Zustände, welche in zeitlicher Abfolge erreicht bzw. durchlaufen werden
 - Auch *temporale* Bedingungen genannt

Beispiele für statische Integritätsbedingungen (1)

- Domain- oder Attributbeschränkungen
 - Ober- oder Untergrenzen
Jahr between 2000 and 2010
 - Festlegung bestimmter Werte
Typ in ('M', 'S')
 - Not-Null Bedingungen
ISBN int not null
- Tupel-Bedingungen
 - Betreffen einzelne Tupel innerhalb einer Relation
 - z. B. Faxnummer von Telefonnummer verschieden

Beispiele für statische Integritätsbedingungen (2)

- Relationen-Bedingungen
 - Schlüssel-Bedingungen
primary key (ISBN)
 - Aggregat-Bedingungen
Summe aller Gehälter darf eine Obergrenze nicht überschreiten
 - Rekursive Bedingungen
z. B. Datenbank für Zugverbindungen: „jeder Ort ist von jedem anderen aus erreichbar“
 - Referentielle Bedingungen
Buch [Lektor] \neq Lektor [Kuerzel] bzw.
foreign key (Lektor) references Lektor

Integritätsbedingungen in SQL

- Schlüsselbedingungen
- Referentielle Integrität und Fremdschlüssel
- Attribut-Bedingungen
 - not null Zusatz
 - check Klausel
- Tupel-Bedingungen
 - check Klausel
- Weitere Bedingungen:
 - Domain-Bedingungen
 - Spezifische Wertebereiche für einzelne Attribute
 - Globale Constraints
 - z. B. Über Trigger realisierbar

Funktionale Abhängigkeiten

Def.: Funktionale Abhängigkeit

- Es sei V eine Attributmenge, $X, Y \subseteq V$. Eine funktionale Abhängigkeit (functional dependency, FD) $X \rightarrow Y$ bezeichnet folgende semantische Bedingung: Sei $r \in \text{Rel}(V)$:

$$(X \rightarrow Y)(r) := \begin{cases} 1 & \text{falls } (\forall \mu, \nu \in r)(\mu[X] = \nu[X] \Rightarrow \mu[Y] = \nu[Y]) \\ 0 & \text{sonst} \end{cases}$$

Test auf funktionale Abhangigkeit

Test, ob Relation $r \in \text{Rel}(V)$ eine FD der Form $X \rightarrow Y$ erfullt:

- Sortieren r nach den Werten in den Attributen aus X so, da Tupel mit gleichen X -Eintragen hintereinander stehen
- Prufen, ob jede „neue“ Menge von Tupeln mit gleichem X -Wert auch gleiche Y -Werte besitzt
 - Falls ja, ist FD gegeben

Implikation

Def.: Implikation

- Sei F eine FD-Menge über einer Attributmenge V und f eine FD mit $\text{attr}(f) \neq V$.
- F impliziert f , kurz $F \sqrt{ } f$, falls $\text{Sat}(F) \neq \text{Sat}(\{f\})$ gilt
- d.h. Jede Relation (über V), welche F erfüllt, erfüllt auch f .

Membership-Problem

Def.: Hülle (closure)

$$F^+ := \{ f \mid attr(f) \subseteq \cup_{g \in F} attr(g) \wedge F \downarrow f \}$$

- Es folgt unmittelbar: $F \subseteq F^+$
- „Implikation testen“ bedeutet damit:
 - Gehört FD f zur Hülle einer bestimmten FD Menge F
 - Wird Membership-Problem funktionaler Abhängigkeiten genannt

Regeln zur Bestimmung gewisser Elemente der Hülle

- Ableitungsregeln:

$$(A1) \quad Y \subseteq X \Rightarrow X \rightarrow Y$$

$$(A2) \quad X \rightarrow Y \wedge Y \rightarrow Z \Rightarrow X \rightarrow Z \quad (\text{Transitivität})$$

$$(A3) \quad X \rightarrow Y \wedge Z \subseteq W \Rightarrow XW \rightarrow YZ \quad (\text{Erweiterung})$$

- Anwendung auf Berechnung einer Hülle:

$$(1) \quad Y \subseteq X \Rightarrow X \rightarrow Y \in F^+ \quad (\text{sogar für } F = \emptyset)$$

$$(2) \quad \{X \rightarrow Y, Y \rightarrow Z\} \subseteq F \Rightarrow X \rightarrow Z \in F^+$$

$$(3) \quad X \rightarrow Y \in F \wedge Z \subseteq W \Rightarrow XW \rightarrow YZ \in F^+$$

Anwendungen des Membership-Algorithmus

- Test auf Äquivalenz
 - Zwei FD-Mengen F und G (über derselben Attributmenge V)
 - Membership-Algorithmus ermöglicht Test, ob $F \approx G$
- Test auf Redundanz
- Berechnung eines Schlüssels für Relationenformat V
 - Beginnen mit Initialisierung $K=V$
 - Sukzessives Entfernen von Attributen aus K
 - Test, ob verbleibende Menge noch die gesamte Attributmenge V funktional impliziert
 - Problem: wir bekommen irgendeinen Schlüssel

Redundanz

- Redundanz einer FD-Menge F :
 - F ist als semantische Spezifikation umfangreicher als nötig
- Gesucht wird redundanzfreie Spezifikation, welche hinsichtlich der Menge zulässiger Relationen das gleiche leistet wie F
- Suchen nach einer Basis

Def.: Basis

- Eine Überdeckung G von F heißt (Abhängigkeits-) Basis von F , falls G nonredundant und minimal ist.

Allgemeine Implikation

- Verallgemeinerungen folgender eingeführter Begriffe sind möglich:
 - Implikation
 - Hülle einer Menge von Abhängigkeiten
- Unbeschränkte Implikation
- Endliche Implikation
 - Beschränkt sich auf mengen endlicher Relationen
 - Für praktische Anwendungen offensichtlich ausreichend

Inklusionsabhängigkeiten

- Erlauben insbesondere die Darstellung von IS-A-Beziehungen
 - Ist 1, falls in beiden beteiligten Relationen zwei Attribute den gleichen Wert haben
 - Sonst 0
- Englisch: Inclusion Dependency (IND)

Inklusionsabhängigkeiten (2)

- FDs und INDs sind einzeln betrachtet intuitiv einsichtig
- Die Interaktionen zwischen beiden können komplex sein
 - Zwei INDs und eine FD implizieren manchmal eine weitere IND

Assertions

- Spezifikation von Integritätsbedingungen bisher:
 - key- oder check-Klauseln in Tabellendefinition
- Definition von Integritätsklauseln auch außerhalb von Tabellendefinitionen möglich
 - *Assertions* (Zusicherungen)
 - Nicht von allen relationalen Systemen realisiert
 - Formulierung unabhängig von der Tabellendefinition
 - Effekt oft auch mit check-Klauseln erreichbar

Beispiel Assertion

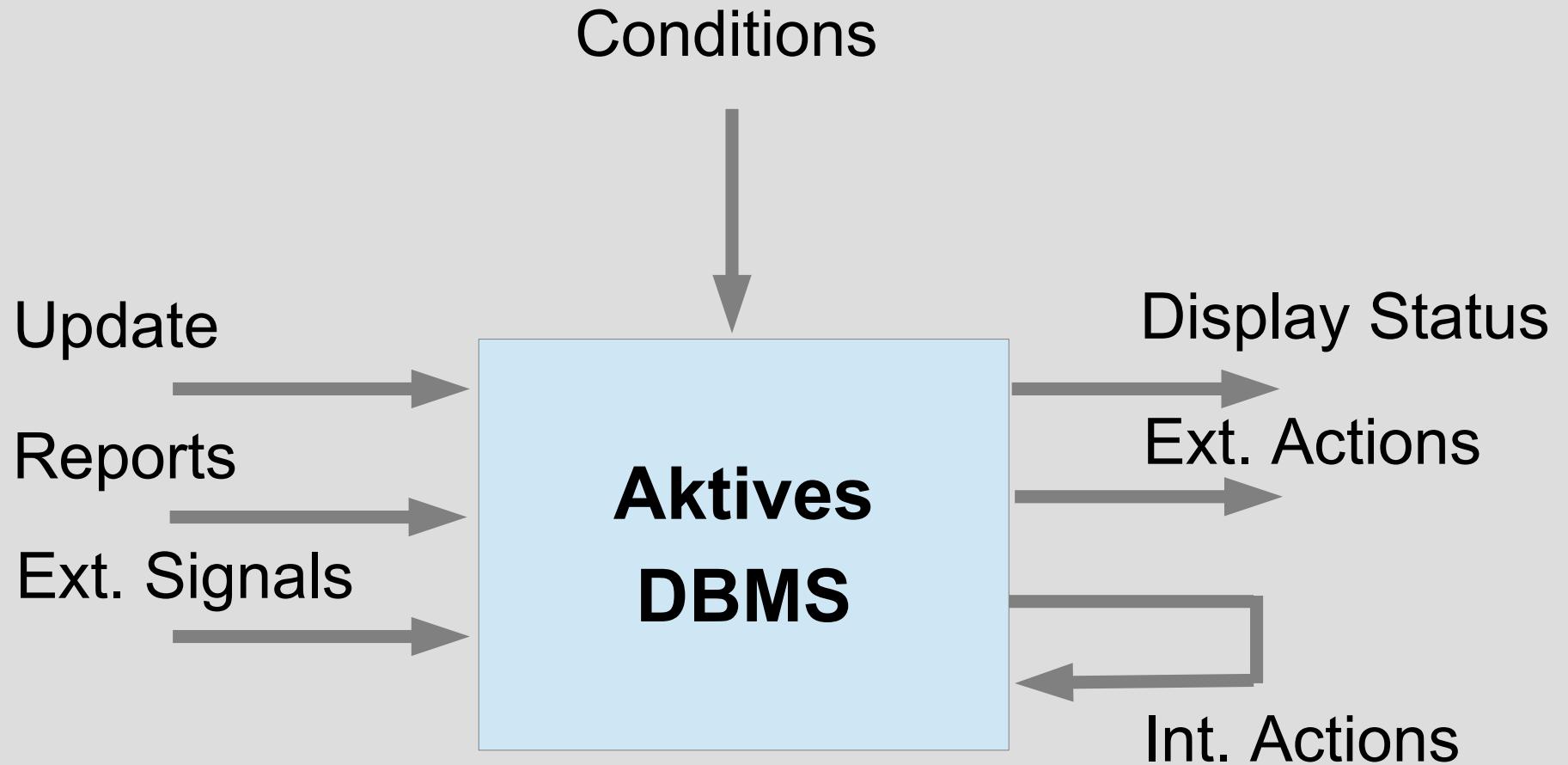
```
create assertion GueltigesFachgebiet  
check (not exists (select distinct Fachgebiet from Lektor  
except select distinct Fachgebiet from Buch) )
```

- Die definierte Bedingung lässt sich via `drop assertion` wieder löschen
- Zu beachten:
 - Logischer Ausdruck der Form „jedes x erfüllt B“
 - Ist umzuformulieren in:
„kein x erfüllt nicht B“
- Assertions sind unabhängig von Tabellendefinitionen

Möglichkeiten zur Integritätsüberwachung

- Transaktionsorientiert
 - Erfolgt im Kontext eines Programms, welches die Datenbank ändern möchte
 - Klassisches Monitor-Programm
- Ereignisorientiert
 - Ereignis-Bedingung-Aktionen – Regeln
 - Als Trigger bezeichnete Programme realisiert
 - Wichtig (Effizienz): welche Bedingung soll Trigger auslösen
 - Heute zunehmend eingesetzt (z. B. DB2)

Aktive Datenbanken



Ausführungsmodelle für ECA-Regeln

- Direktes Feuern (Immediate Firing)
 - Regel wird gefeuert, sobald Bedingung wahr wird (auch innerhalb Transaktion)
- Verzögertes Feuern (Deferred Firing)
 - Bis Ende der laufenden (auslösenden) Transaktion verzögert
- Gleichzeitiges Feuern (Concurrent Firing)
 - Jede Regel-Aktion erzeugt eigenen Prozess
 - Parallel Ausführung der Prozesse

6 Algebraischer Entwurf mit der Normalformtentheorie

- Erste Normalform
- Zweite Normalform
- Dritte Normalform (3NF)
- Boyce / Codd-Normalform (BCNF)
- Vierte Normalform (4NF)
- Fünfte Normalform (5NF)
- Weitere Normalisierungsaspekte

Ziele der Normalisierung

- Redundanzvermeidung als Basis der Anomaliefreiheit
- Vermeidung unnötiger Abhängigkeiten, die Performanceeinbußen bei Einfüge-, Lösch- und Änderungsoperationen nach sich ziehen
- Senkung der Anzahl beteiligter Tabellen bei der Modifikation der Datenbank
- Erhöhung des Dokumentationsgrades des entstehenden Datenmodells (Ziel: Verständlichkeit)

Zentrale Forderung: Anomalienfreiheit

- **Einfügeanomalie:** Durch das Hinzufügen eines korrekten neuen Tupels werden konsistent vorliegende Daten in einen inkonsistenten Zustand überführt.
- **Löschanomalie:** Durch die Entfernung eines Tupels entsteht ein inkonsistenter Datenbestand.
- **Aktualisierungsanomalie:** Durch die Änderung eines vorhandenen Tupels entsteht ein inkonsistenter Datenbestand.

Universalrelation

Universalrelation:

- Ausgangspunkt der algebraischen Normalisierung
 - Nur ein Relationenschema
 - Urrelation / Universalrelation genannt
- Umfasst alle Attribute
- Umfasst alle funktionalen Abhangigkeiten zwischen den Attributen

Im Fall mehrerer Relationenschemata:

- Auf jedes einzelne separat anwenden

Erste Normalform

Def.: Erste Normalform (1NF)

- Eine Relation ist dann in erster Normalform, wenn ihre Domänen (=Wertausprägungen der Attribute) nur einfache (atomare) Werte besitzen.

Test auf Einhaltung der ersten Normalform

- Die Relation sollte keine nicht-atomaren Attribute oder verschachtelte Relationen enthalten.
- Der algorithmische Ableitungsprozeß aus dem konzeptuellen Schema stellt durch die Organisation der Repräsentationstypen sicher, daß Attribute ausschließlich durch atomare Werte repräsentiert werden.

Zweite Normalform

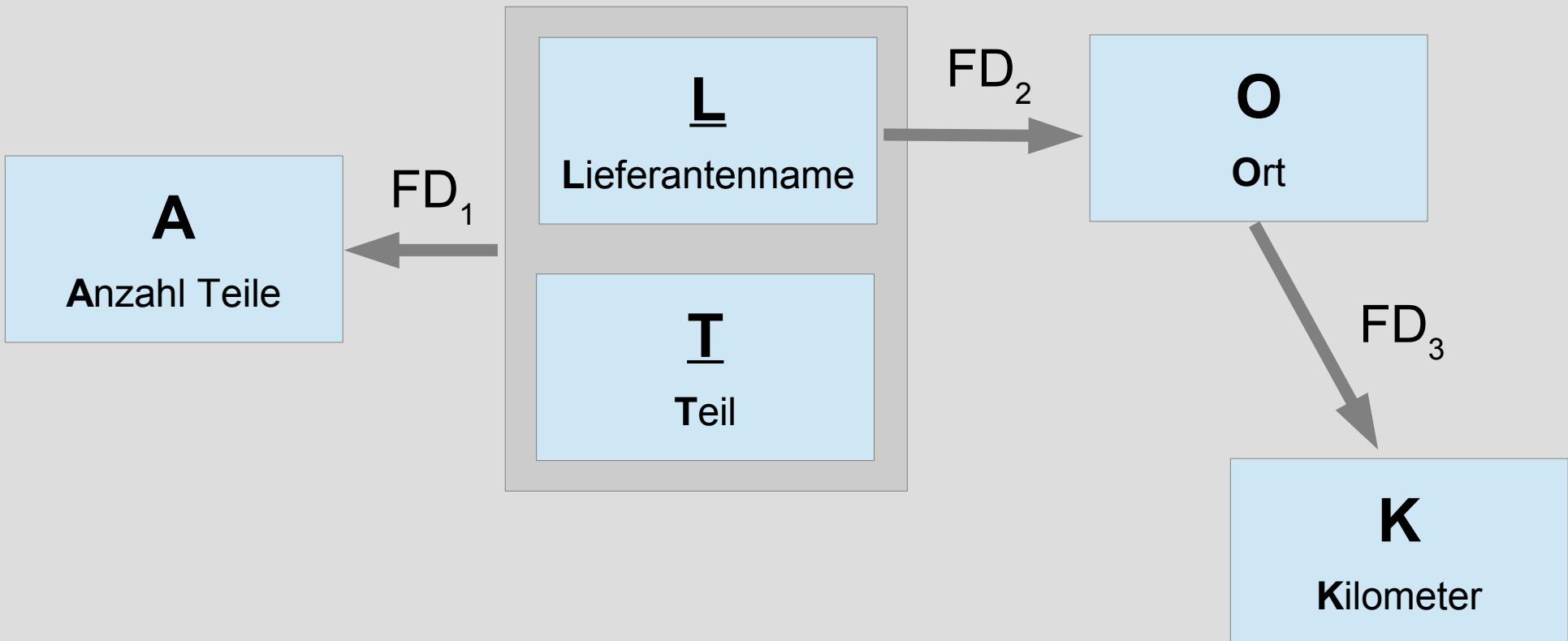
- Grundlage: Konzept der vollen funktionalen Abhangigkeit

Volle funktionale Abhängigkeit

Def.: Volle funktionale Abhängigkeit

- Ein Attribut y einer Relation ist vollfunktional abhängig von einem Attribut x wenn gilt, daß jede Ausprägung von x genau eine Ausprägung von y bestimmt und y nicht abhängig von Teilattributen von x ist.
- Im Zeichen: $x \rightarrow y$.
- Die vollfunktionale Abhängigkeit wird häufig als FD (functional dependency) abgekürzt.

Beispiel für volle funktionale Abhängigkeiten

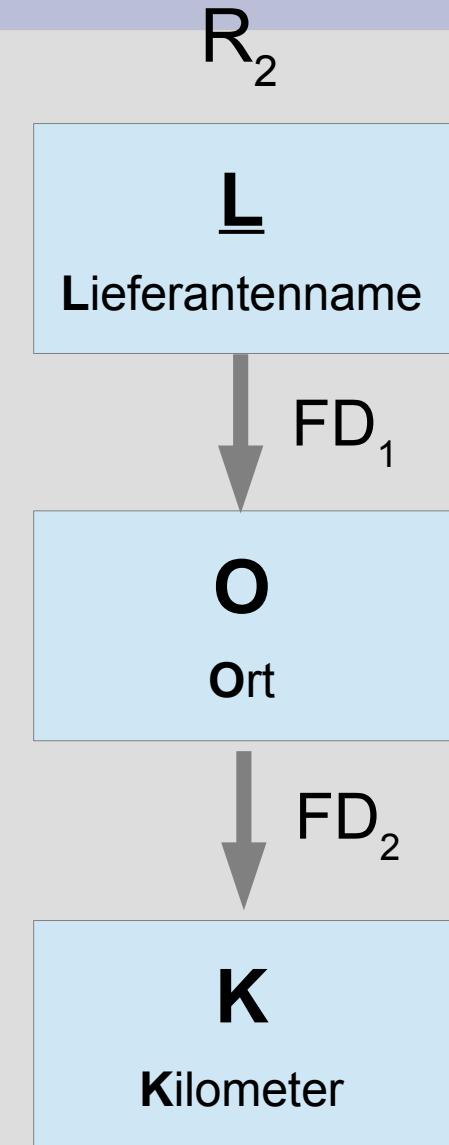


Zweite Normalform (2NF)

Def.: Zweite Normalform (2NF)

- Eine Relation ist in 2NF genau dann, wenn sie in 1NF ist und jedes Nichtschlüsselattribut voll funktional abhängig von einem Schlüsselkandidaten ist.

Beispiel in 2NF



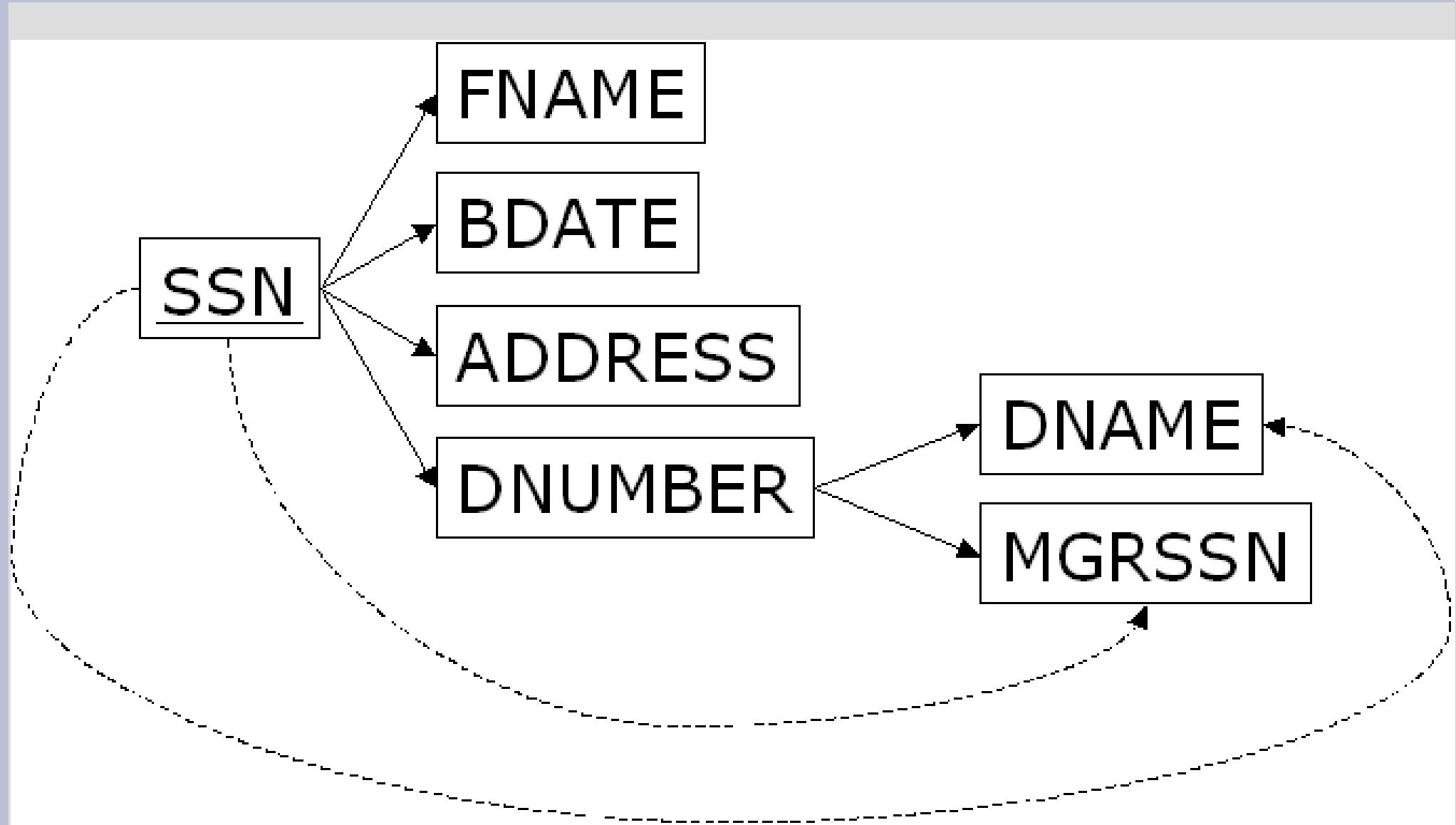
Transitive Abhangigkeit

- Erweiterung der zweiten Normalform: Zusatzlich zur voll funktionalen Abhangigkeit wird die **transitive Abhangigkeit** eingefuhrt und betrachtet

Def.: Transitive Abhangigkeit

- In einer Relation R ist ein Attribut z transitiv von einem Attribut x abhangig dann und nur dann, wenn z voll funktional von y und y voll funktional von x abhangig ist.
- Im Zeichen: $x \rightarrow\!\!\!-\!\!\!> z$

Transitive Abhängigkeiten



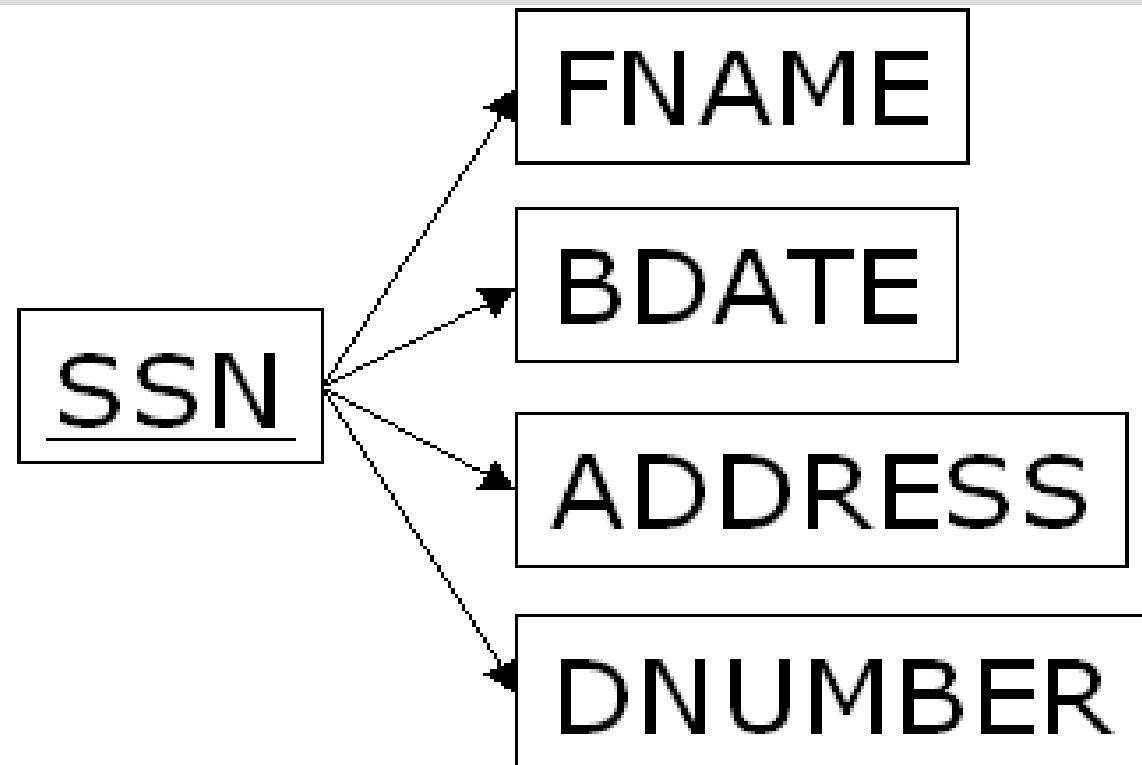
Dritte Normalform (3NF)

Def.: Dritte Normalform (3NF)

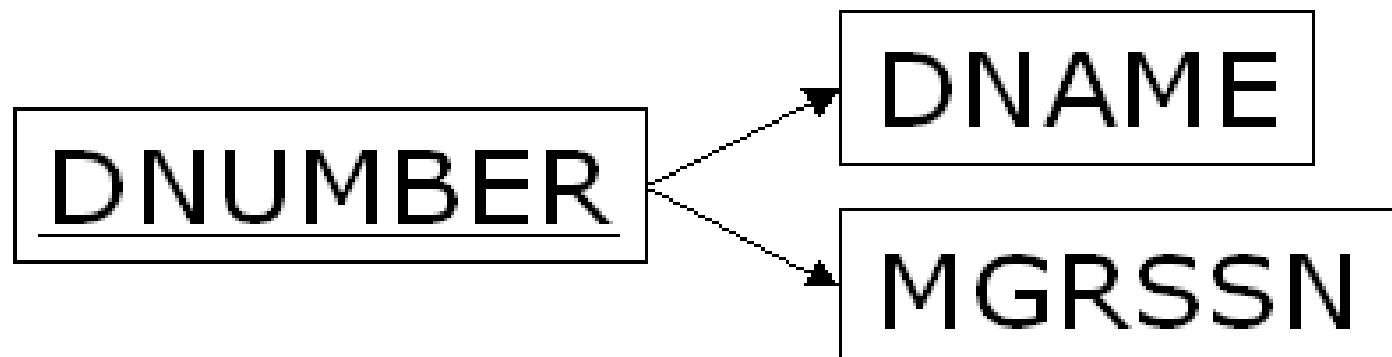
- Eine Relation ist dann und nur dann in 3NF, wenn sie in 2NF ist und jedes Nicht-Schlüsselattribut nicht-transitiv abhängig ist von einem Schlüsselkandidaten
- sie ist auch in 3NF, wenn sich eine transitive Abhängigkeit ausschließlich über Bestandteile des Schlüsselkandidaten herleiten lässt.

Relation in 3NF

R1:



R2:



Boyce/Codd-Normalform (BCNF)

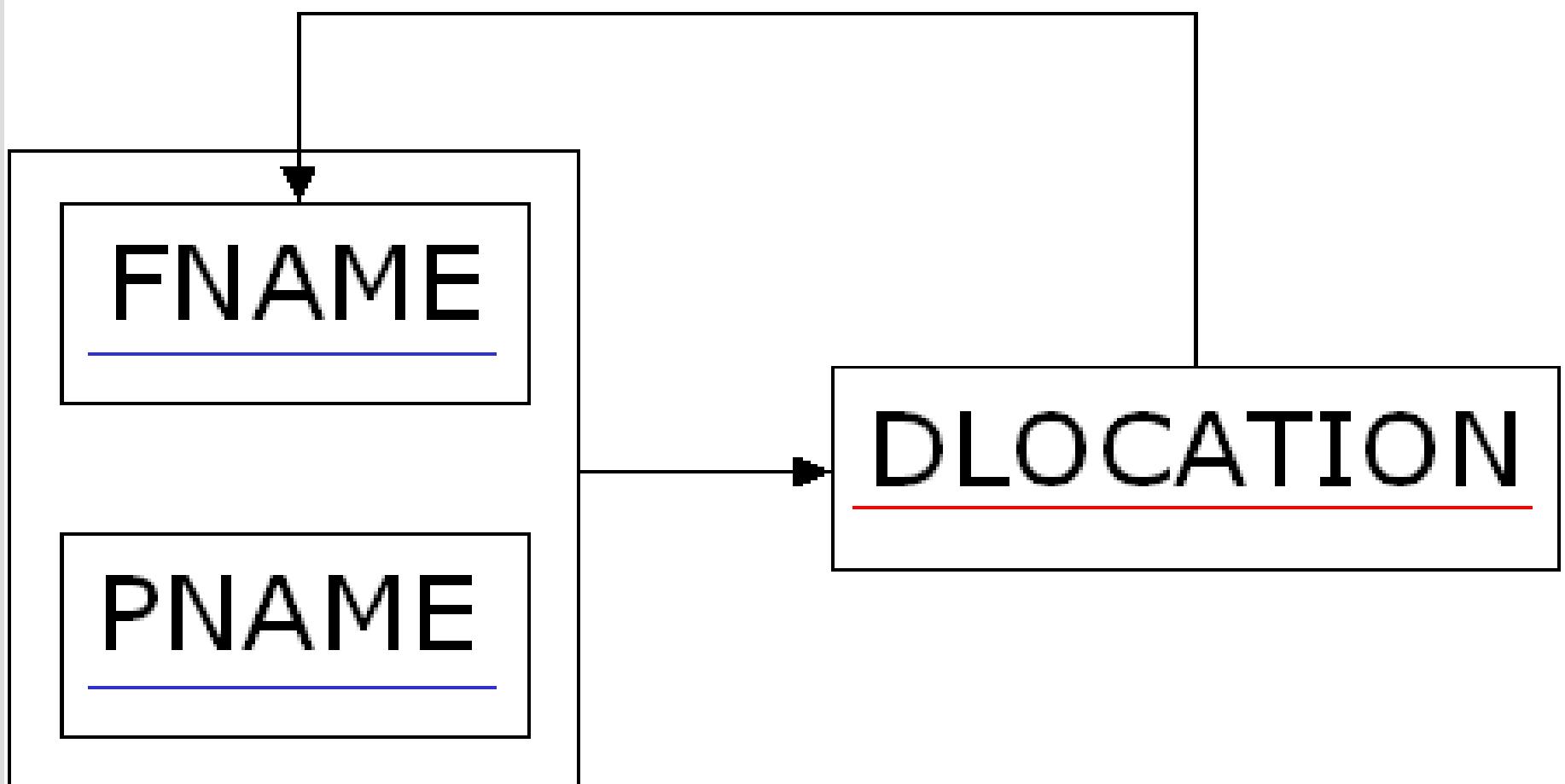
Def.: Boyce/Codd-Normalform

- Eine Relation ist dann und nur dann in BCNF, wenn sie in 3NF ist und gleichzeitig jede Determinante Schlüsselkandidat ist.

Def.: Determinante

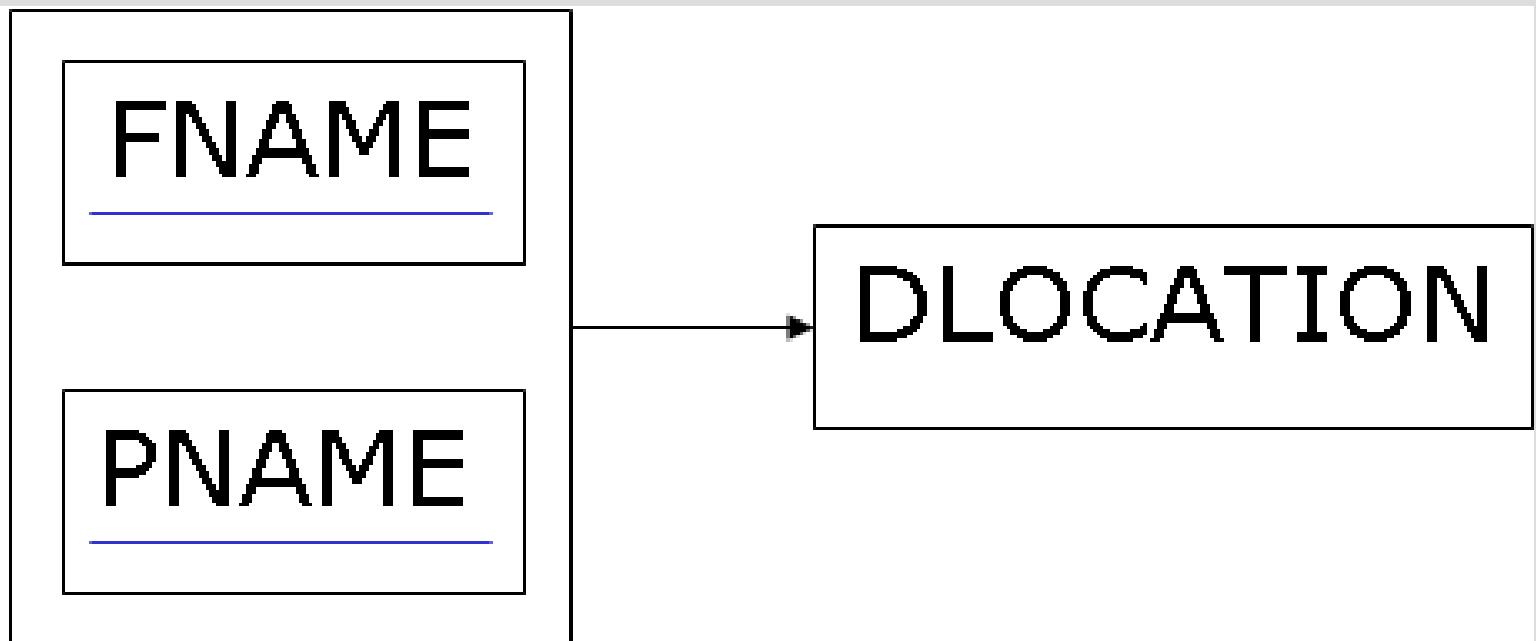
- Eine Determinante ist ein (eventuell zusammengesetztes) Attribut, von dem ein anderes voll funktional abhängig ist.

Funktionale Abhängigkeiten

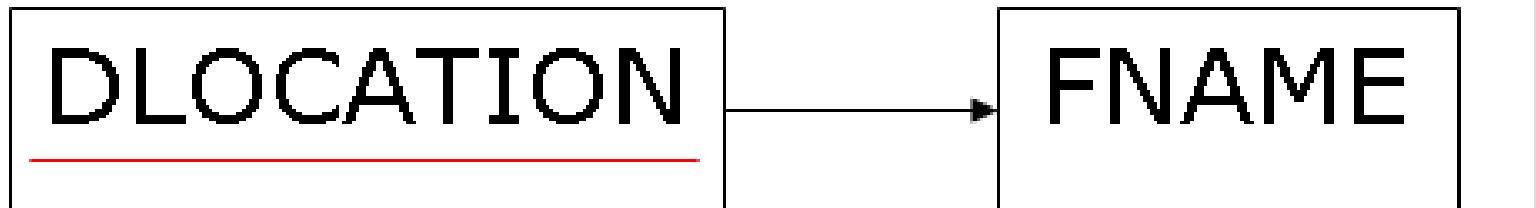


Relation in BCNF

R1:



R2:



Vierte Normalform (4NF)

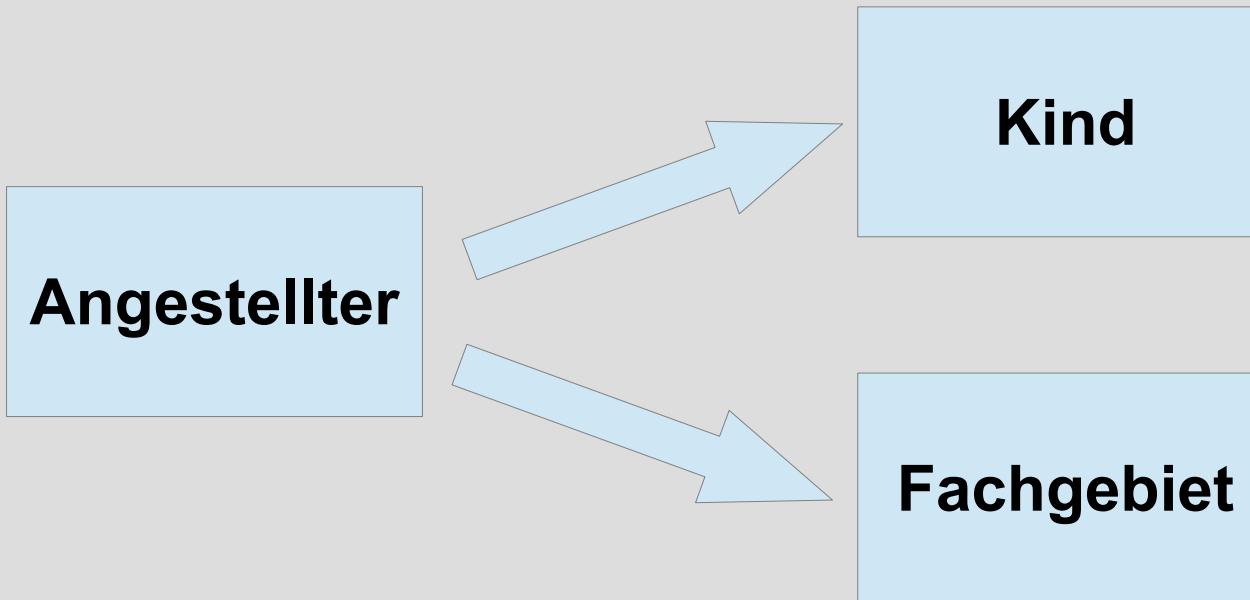
- Neben funktionalen gibt es noch weitere Abhangigkeiten:

Def.: Mehrwertige Abhangigkeit

- Eine mehrwertige Abhangigkeit (multivalued dependency, MVD) ist eine Abhangigkeit, die einem einem Attribut eine Menge verschiedener Werte zuordnet.

Beispiel mehrwertige Abhängigkeiten

R



Mehrwertige Abhangigkeit / Vierte Normalform (4NF)

Def.: Triviale mehrwertige Abhangigkeit

- Eine triviale mehrwertige Abhangigkeit ist eine mehrwertige Abhangigkeit zwischen Attributmengen X und Y der Relation R fur die gilt: Y ist eine Teilmenge von X oder die Vereinigung von X und Y bildet R.

Def.: Vierte Normalform

- Eine Relation ist dann in vierter Normalform, wenn sie nur noch triviale mehrwertige Abhangigkeiten enthalt.

Fünfte Normalform

Def.: Fünfte Normalform

- Eine Relation ist dann in fünfter Normalform (5NF), bei ihrer Zerlegung durch Projektionen und deren anschließender Kombination durch Verbundoperationen keine Tupel gebildet werden, die nicht Bestandteil der Ausgangsrelation waren.

Beispiel: Ausgangsrelation

FNAME	DNUMBER	DLOCATION
John	5	Bellaire
John	5	Houston
James	1	Houston

Ur:

Ursprüngliche Relation

Beispiel: Schritt 1

FNAME	DNUMBER
John	5
John	5
James	1

R₁₁:

SELECT Ur.FNAME, Ur.DNUMBER INTO R11 FROM Ur

Beispiel: Schritt 2

FNAME	DLOCATION
John	Bellaire
John	Houston
James	Houston

R₁₂:

SELECT Ur.FNAME, Ur.DLOCATION INTO R12 FROM Ur

Beispiel: Schritt 3

DNUMBER	DLOCATION
5	Bellaire
5	Houston
1	Houston

R₁₃:

SELECT Ur.DNUMBER, Ur.DLOCATION INTO R13 FROM Ur

Beispiel: Schritt 4

FNAME	DNUMBER	DLOCATION
John	5	Bellaire
John	5	Houston
James	1	Houston

R₂₁:

```
SELECT R11.FNAME, R11.DNUMBER, R12.DLOCATION  
INTO R21 FROM R11 INNER JOIN R12 ON  
R11.FNAME=R12.FNAME
```

Beispiel: Schritt 5

FNAME	DNUMBER	DLOCATION
John	5	Bellaire
John	5	Houston
James	1	Houston

R₂₂:

```
SELECT R11.FNAME, R11.DNUMBER, R13.DLOCATION  
INTO R22 FROM R11 INNER JOIN  
R13 ON R11.DNUMBER = R13.DNUMBER
```

Beispiel: Ergebnis

FNAME	DNUMBER	DLOCATION
John	5	Bellaire
John	5	Houston
James	1	Houston
James	5	Houston
John	1	Houston

R₂₃:

```
SELECT R12.FNAME, R12.DLOCATION, R13.DNUMBER  
INTO R23 FROM R12 INNER JOIN  
R13 ON R12.DLOCATION=R13.DLOCATION
```

Beispiel: Eliminierung des unechten Tupels

```
SELECT R21.FNAME, R22.DNUMBER, R23.DLOCATION  
INTO RESULT FROM (R21 INNER JOIN R22 ON  
(R21.FNAME = R22.FNAME) AND  
(R21.DNUMBER = R22.DNUMBER) AND  
(R21.DLOCATION = R22.DLOCATION)) INNER JOIN  
R23 ON (R22.FNAME = R23.FNAME) AND  
(R22.DNUMBER = R23.DNUMBER) AND  
(R22.DLOCATION = R23.DLOCATION) AND  
(R21.FNAME = R23.FNAME) AND  
(R21.DNUMBER = R23.DNUMBER) AND  
(R21.DLOCATION = R23.DLOCATION)
```

FNAME	DNUMBER	DLOCATION
John	5	Bellaire
John	5	Houston
James	1	Houston

Test auf 5NF

Test auf Einhaltung der fünften Normalform:

- Wenn durch Projektions- und Verbundoperationen keine unechten Tupel entstehen, dann ist die Relation in 5NF.

Höchstmögliche erreichbare Normalisierungsform

- 5NF ist die höchstmögliche über den Normalisierungsprozeß erreichbare Normalform
- Nicht jede Relation kann bis in 5NF gebracht werden
- Der Normalisierungsprozeß endet generell mit der höchstmöglichen Normalform
(nicht zwingend 5NF, orientiert sich an den modellierten Informationszusammenhängen)

Weitere Normalisierungsaspekte

- Inklusionsabhängigkeit
- Template-Abhängigkeiten
- Domain-Key-Normalform (DKNF)

Inklusionsabhängigkeit

Inklusionsabhängigkeit (inclusion dependence, ID):

- Beziehungen zwischen Super- und Subtypen, insbesondere die Attributentsprechung
- Drei Inferenzregeln:
 - IDIR₁: (Reflexivität) Die Inklusionsabhängigkeit ist reflexiv.
 - IDIR₂: (Attributentsprechung) Verfügen zwei Typen über dieselben Attribute, dann entsprechen sie sich.
 - IDIR₃: (Transitivität) Ist *B* Untertyp von *A* und *C* Untertyp von *B*, dann ist auch *C* Untertyp von *A*.
- bisher keine Normalformen vorgeschlagen

Template-Abhängigkeiten

Idee: Template-Abhängigkeit fußen auf der Definition einer Reihe von Hypothesetupeln und daraus abgeleiteten Konklusionstupeln

- gültige Ausprägungen der Datenbank werden abstrahiert beispielhaft aufgezeigt
- einfache Formulierung von Intrarelations-abhängigkeiten, die sich auf konkrete Wertausprägungen einzelner Attribute beziehen.

Beispiel: kein Angestellter darf mehr verdienen, als sein Vorgesetzter

Domain-Key-Normalform (DKNF)

Grundannahme: wird für jedes Attribut einer Relation eine Domäne (d.h. die Menge der zugelassenen Wertbelegungen) angegeben, so verschwinden alle Änderungsanomalien

- Gleichzeitig fordert die DKNF die eindeutige Identifikation jedes Attributs einer Relation durch einen Schlüssel.

Praxis: die Angabe einer allgemein formulierten eindeutigen Domäne kann mitunter zu Schwierigkeiten führen --> die Prüfung auf Einhaltung dieser Normalform ist mit erheblichen technischen Umsetzungsschwierigkeiten verbunden

7. Objektbasierte Modelle

- **Objektbasierte Modelle**
- **Datenbanksprachen**
 - Codd-vollständige (relationale) Sprachen
 - Sprachstandard SQL (Ergänzungen zum 2. Semester)
- **Datenbanksystemtechnik**

Objektbasierte Modelle

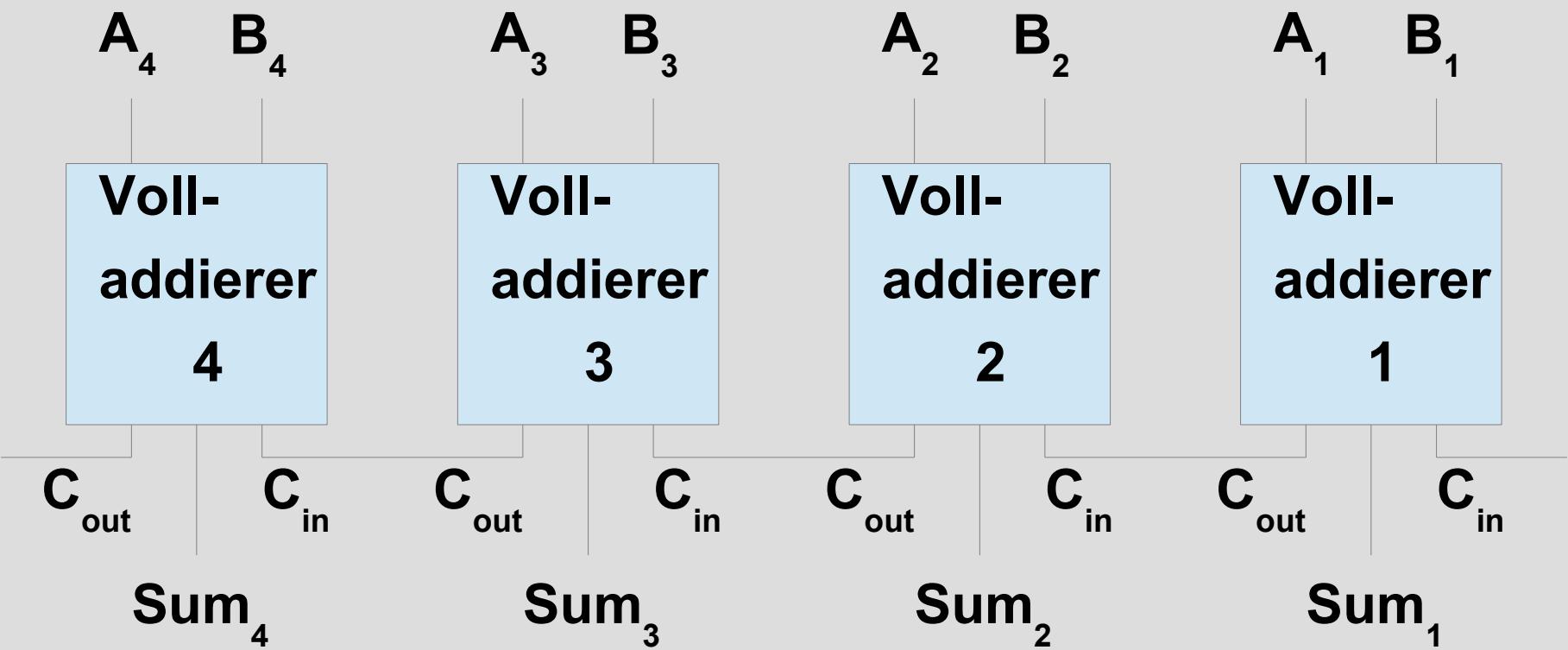
- Unterscheidung:
 - Objektorientierte Modelle
 - Obektrelationale Modelle
- Eigenschaften

Grundlagen

Anforderungen an Datenmodell:

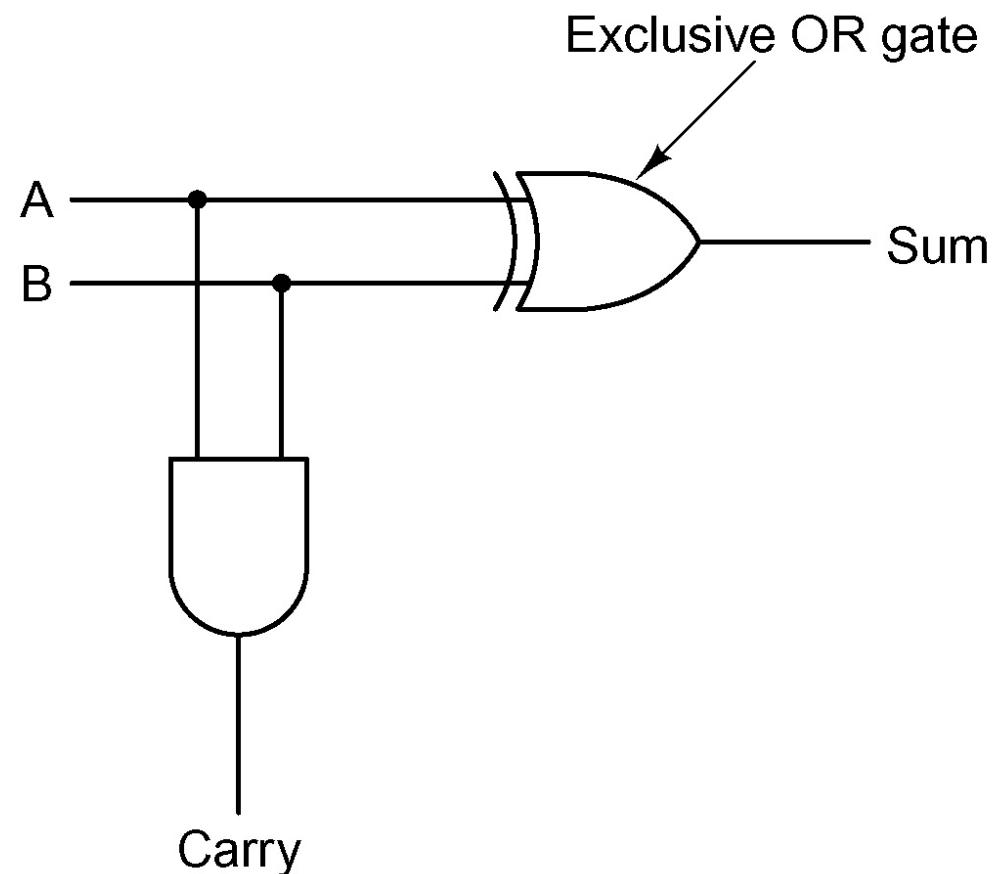
- Direkte Darstellbarkeit bzw. Modellierbarkeit stark strukturierter Information
- „hohe“ Datentypen (Beschreibung u. Manipulation graphischer und audiovisueller Daten)
- Bereitstellung Typ-spezifische Operationen zur Spezifikation bzw. Modellierung von Verhalten (anwendungsspezifische Operationen auf Daten bzw. Instanzen von Datentypen)
- Unterstützung mächtiger Sprachen zur adäquaten Nutzung des Datenmodells

Beispiel: 4-Bit-Ripple-Carry-Adder



Halbaddierer

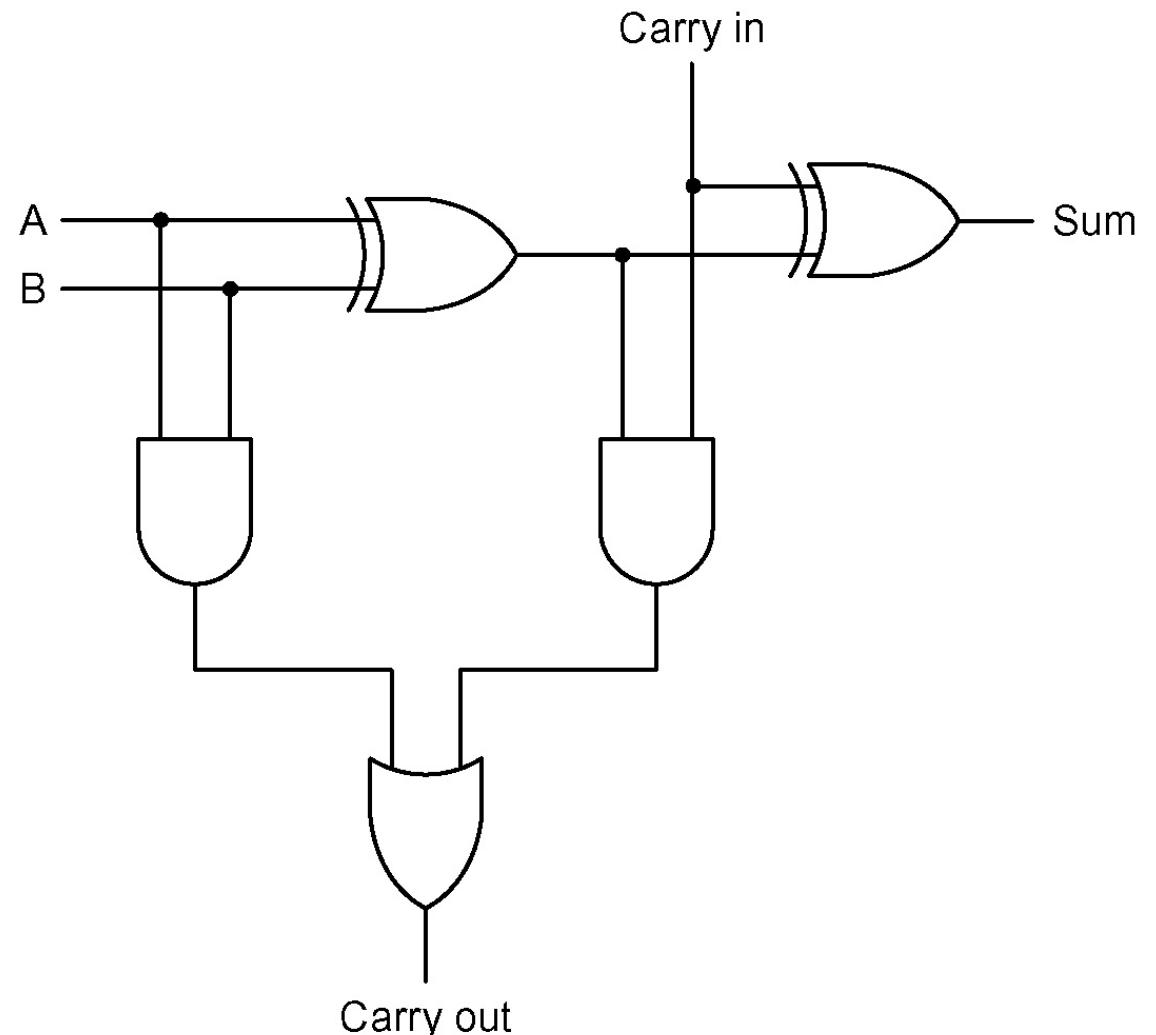
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Tanenbaum, A.S. (2001): Computerarchitektur. Pearson Studium.

Volladdierer

A	B	Carry in	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



4-Bit-Ripple-Carry-Adder

<i>Komponente</i>	<i>Anzahl</i>
<i>4-Bit-Ripple-Carry-Adder</i>	
<i>Inputs</i>	8
<i>Outputs</i>	5
<i>Volladdierer</i>	4
<i>Inputs</i>	3
<i>Outputs</i>	2
<i>Oder-Gatter</i>	1
<i>Inputs</i>	2
<i>Outputs</i>	1
<i>Halbaddierer</i>	2
<i>Inputs</i>	2
<i>Outputs</i>	2
<i>Xor-Gatter</i>	1
<i>Inputs</i>	2
<i>Outputs</i>	1
<i>Und-Gatter</i>	1
<i>Inputs</i>	2
<i>Outputs</i>	1

Relationale Darstellung des 4-Bit-Ripple-Carry-Adder

Chip	ChipID	ChipName
	125	4-Bit-RC-Adder

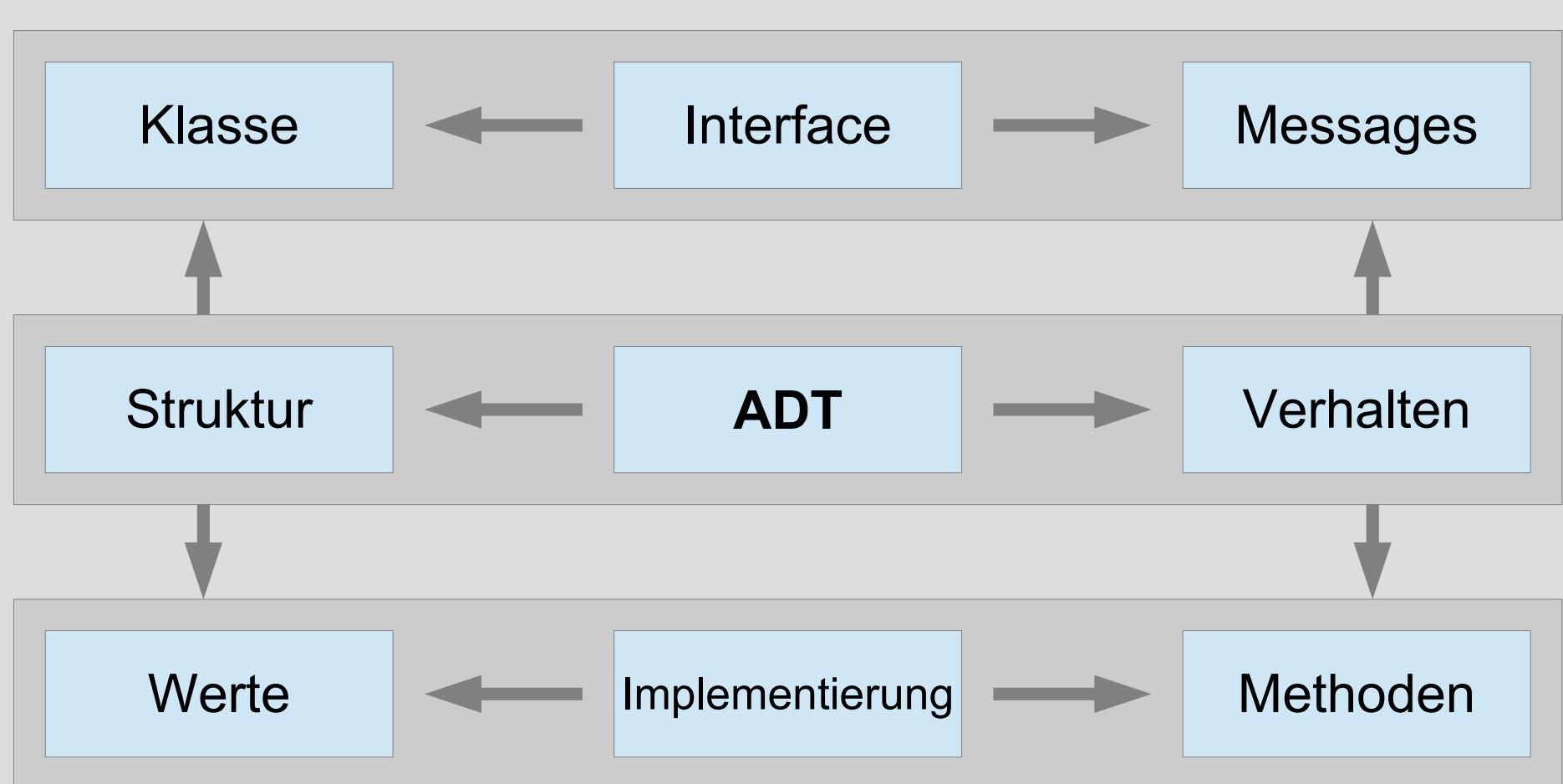
Modul	ModID	ModName	KompVonChip
	201	VA1	125
	202	VA2	125
	203	VA3	125
	203	VA4	125

SubModul	SubID	SubName	KompVonMod
	301	HA11	201
	302	HA12	201
	303	ODER1	201
	304	HA21	202
	305	HA22	202
	306	ODER2	202
	307	HA31	203
	308	HA32	203
	309	ODER3	203
	310	HA41	204
	311	HA42	204
	312	ODER4	204

Objektorientierung in höheren Programmiersprachen

- Erster „Meilenstein“ höherer Sprachen: FORTRAN (Mitte der 50er Jahre)
- Konzept / Begriff Objekt: Simula (Anfang 60er Jahre)
 - Konventionelle Sicht: aktive Funktionen operieren auf passiven Objekten
 - Neu: aktive Objekte reagieren auf Nachrichten anderer Objekte
- Weiterentwicklung des „Einkapselns“ von Struktur und Verhalten in den 70er Jahren
 - Konzept der abstrakten Datentypen:
 - Daten-Abstraktion (data abstraction)
 - Verbergen von Information (information hiding)

Abstrakte Datentypen (ADT)



Paradigma der Objektorientierung

- Jede Entität einer gegebenen Anwendung wird modelliert als Objekt mit eigener Identität
 - Objekte lassen sich aus anderen zusammensetzen
 - Ein Objekt kann von mehreren referenziert werden (Object Sharing)
- Objekt kapselt Struktur und Verhalten
- Zugriff auf internen Zustand nur über Nachrichten
 - Objekte kommunizieren durch Austausch von Nachrichten (Message Passing)
- Objekte gemeinsamer Struktur und gemeinsamen Verhalten in Klasse gruppiert
 - Jedes Objekt ist Instanz einer Klasse
 - Message Passing also unter Rückgriff auf die von der Klasse zentral gehaltenen Informationen
- Klasse kann als Spezialisierung ein oder mehrerer anderer Klassen definiert werden

Kennzeichen objektorientierter Datenmodelle

Zusammenführung von verschiedenen Konzepten:

- Bereich Programmiersprachen:
 - Abstrakte Datentypen
 - Prinzip der Kapselung
- Software-Engineering:
 - Code-Erweiterbarkeit und Wiederverwendbarkeit
 - Prinzip der Modularisierung
- Künstliche Intelligenz (KI):
 - Ideen / Ansätze zur Wissensrepräsentation
 - Techniken / Methoden zur Klassifikation
- Bereich Datenbanken (Datenmodellierung):
 - Verallgemeinerungen des Relationenmodells

Modellierungseigenschaften

- Darstellbarkeit komplexer Objekte
- Objekte lassen sich mit eigener Identität versehen
- Schema der Datenbank besteht aus Klassen
 - Attribute der Klassen jeweils von bestimmten Typ
- Klassen lassen sich in Vererbungshierarchie anordnen
 - Gegenseitiges (und zyklisches) Referenzieren möglich
- Struktur und Verhalten eines Objektes müssen sich kapseln lassen
 - Klasse verfügt neben Strukturinformation über eine Spezifikation der Methoden, die auf den Objekten der Klasse ausführbar sind
- Methodennamen lassen sich in der Vererbungshierarchie überschreiben
- Fügt Benutzer dem Modell neue Typen hinzu, werden diese und vordefinierte gleichbehandelt

Beispiel Automobilvertrieb

- Personen (*Namen, Alter, Wohnsitz, Fuhrpark*)
- Fahrzeuge (*Modellbezeichnung, Farbe, Hersteller*)
- Automobile: spezielle Fahrzeuge (*zusammengesetzt aus Antrieb und Karosse*)
- Fahrzeugantrieb (*zusammengesetzt aus Motor und Getriebe*)
- Ottomotor (*Leistung und Hubraum*)
- Firmen (*Namen, Sitz, Manager, Angestellte*)
- Angestellte sind Personen mit *Qualifikationen, Gehalt und Familienmitgliedern*
- Adressen (*Straße und Ort*)

Klassenbegriff - Relationenschema

- In unserem Kontext spielt der Klassenbegriff die gleiche Rolle wie der Begriff Relationenschema im relationalen Modell
- Unterschiede:
 - Jedes Relationenschema hat bis auf die verwendeten Attribute den gleichen Typ
 - Jede Klasse:
 - kann eigenen Typ besitzen (legt Struktur der erlaubten Werte fest)
 - Es gibt eine Menge von Objekten mit eigener Identität
 - Wert der Objekte aus dem Domain des Typs, welcher der Klasse zugeordnet ist

Beispiel eines Typs: „Datenstruktur“ für Personaldaten

```
[ Name: [ Vorname: string, Nachname: string ],  
  Alter: integer, Verheiratet: boolean,  
  Wohnsitz: [ Strasse: string, Ort: string,  
             PLZ: integer ],  
  Kinder < string >,  
  Arbeitsstelle:  
    { [ Bezeichnung: [ Name: string,  
                      Sitz: string ],  
        Prozentsatz: integer ] } ]
```

{ }	Mengenbildung
[]	Tupelbildung
< >	Listenbildung

Beispiel eines Typs: „Datenstruktur“ für Person

`C = { Person, Adresse, Firma }` sei Menge von Klassenanmen

```
[ Name: [ Vorname: string, Nachname: string ],
  Alter: integer, Verheiratet: boolean,
  Wohnsitz: Adresse,
  Kinder < Person >,
  Arbeitsstelle:
    { [ Bezeichnung: Firma,
        Prozentsatz: integer ] } ]
```

Definition von Typen folgender Klassen:

Adresse: [Strasse: string, Ort: string, Plz: integer]
Firma: [Name: string, Sitz: Adresse]

Spezialisierung

Typ Person:

```
[ Name: [ Vorname: string, Nachname: string ],  
  Alter: integer, Verheiratet: boolean,  
  Wohnsitz: Adresse,  
  Kinder < Person > ]
```

Typ Angestellter:

```
[ Name: [ Vorname: string, Nachname: string ],  
  Alter: integer, Verheiratet: boolean,  
  Wohnsitz: Adresse,  
  Kinder < Person >,  
  Gehalt: integer ]
```

Definitionsmöglichkeiten realer Systeme

Zusätzliche Aspekte realer System im Vergleich zu obigen Beispielen:

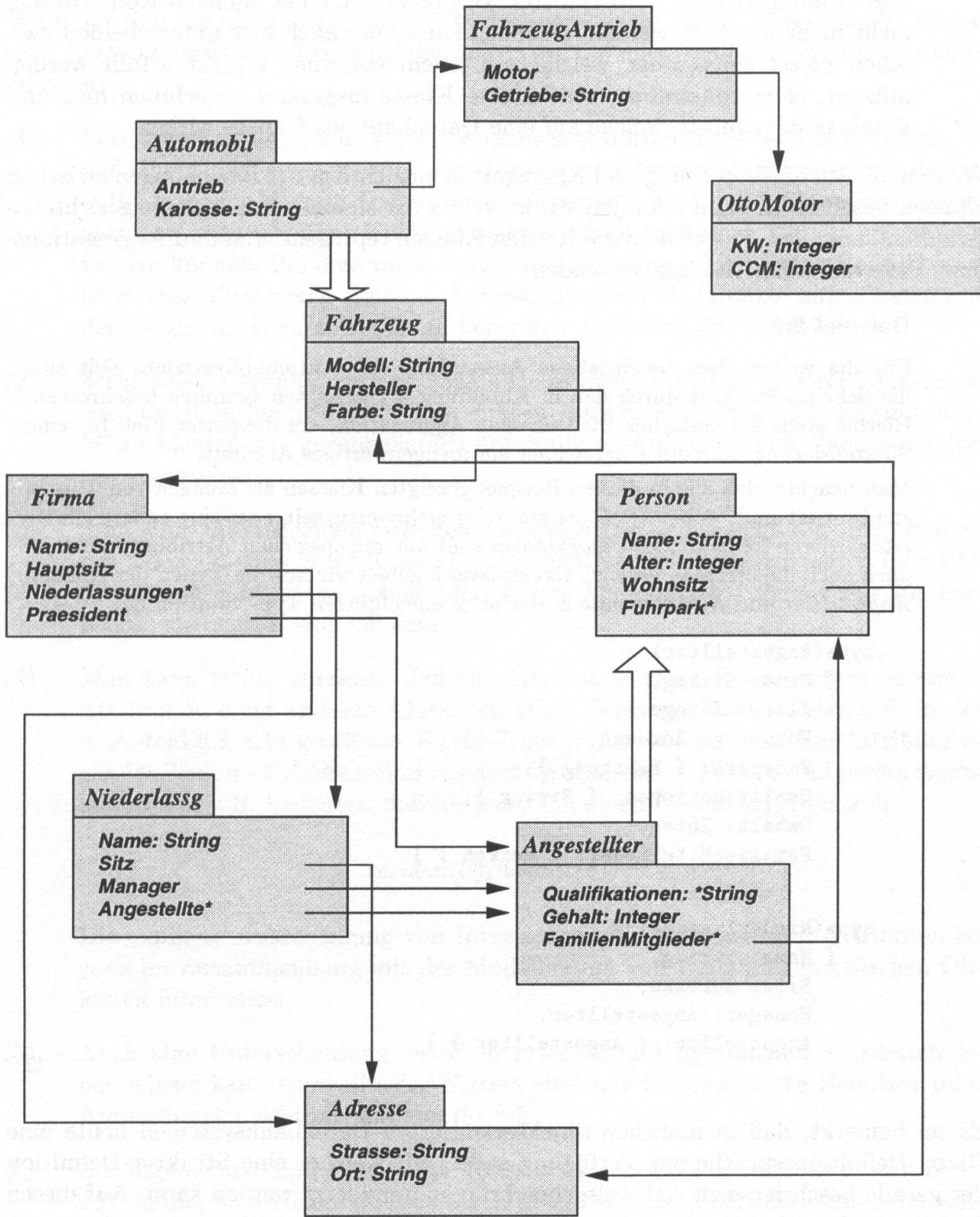
- Typen dürfen benannt sein
 - Typ-Deklarationen können wiederverwendet werden
- Mehrfach-Vererbung
 - Ist eine Klasse Unterklasse mehrerer anderer Klassen, so ist sie gewöhnlich Subtyp jeder dieser Klassen
 - Auflösung von Namenskonflikten ggf. erforderlich
- Unterscheidung Instanz-Attribute und Klassen-Attribute
 - Instanz-Attribute beschreiben Eigenschaften eines jeden Objekts einer Klasse
 - Klassen-Attribute sind nur für ganze Klasse sinnvoll
- Attribut einer Klasse kann invers zum Attribut einer anderen Klasse sein

Definitionsmöglichkeiten realer Systeme (2)

Zusätzliche Aspekte realer System im Vergleich zu obigen Beispielen (2):

- Unterscheidung private und öffentliche Attribute ist eventuell sinnvoll
- Bestimmte Klassen sind systemseitig vordefiniert
 - Oft existiert Klasse Object als Wurzel Vererbundshierarchie
 - Wurzelklasse umfasst üblicherweise nur Methoden (und keine Struktur)
- Integritätsbedingungen oft nicht vorhanden
 - „klassische“ Datenabhängigkeiten verlieren im Objektorientierten Kontext oft an Bedeutung
 - z.B. Explizite Schlüssel bei Objekt-Identität nicht unbedingt erforderlich

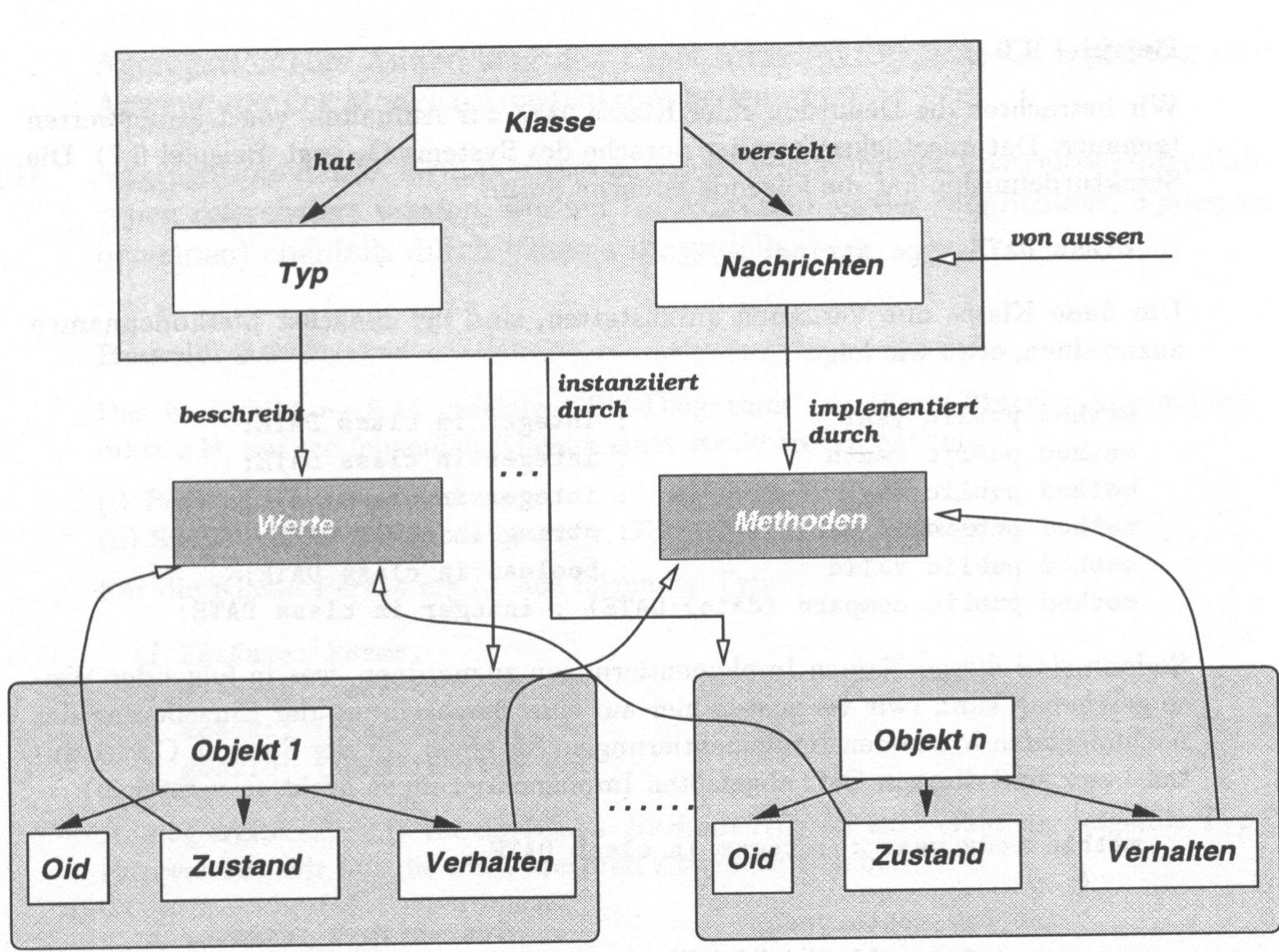
Klassenverband zum Beispiel Autovertrieb



Transformation ER in ein Objektmodell

- Entity-Deklarationen --> Klassen
 - Primärschlüssel eventuell verzichtbar (Objekt-Identität)
- Relationship-Deklarationen werden ähnlich wie bei einer Transformation in Relationenmodell behandelt
- IS-A Beziehungen führen auf Unterklassen bereits erzeugter Klassen
 - Eventuell vereinbarte Charakteristika einer solchen Beziehung (wie Disjunktheit verschiedener Spezialisierungen) sind nicht explizit ausdrückbar

Zusammenfassung Objektmodell



Object Database Standard

- Object Management Group (OMG)
 - Object Request Broker (ORB)
 - Common Object Request Broker Architecture (CORBA)
- Object Database Management Group (ODMG)
 - Untergruppe der OMG
 - Ziel Standardisierung Objektorientierter Datenbanken
- Komponenten:
 - Objektmodell: ausgehend vom OMG-Modell
 - Datendefinitionssprache: Object Definition Language, ODL
 - Anfragesprache: Object Query Language, OQL
 - Anbindungen an OO-Programmiersprachen: C++, Smalltalk, Java

Grundzüge des ODMG-Modells

- Zentrale Modellierungskonstrukte:
 - Objekt (mit Identität)
 - Literal (ohne Identität)
- Kategorisierung von Objekten und Literalen anhand von Typen
 - Alle Elemente eines Typs haben gleiche (Zustands-)Struktur und gleiches Verhalten
- Menge von Eigenschaften definiert Zustand eines Objektes
 - Diese Eigenschaften sind Attribute des Objektes oder Beziehungen zu anderen Objekten
- Menge von Operationen legt Verhalten des Objektes fest
 - Diese können auf einem Objekt des betreffenden Typs ausgeführt werden

Typ-Begriff im ODMG-Modell

Spezifikation

- Interface: Verhaltensbeschreibung
- Literal: Zustandsbeschreibung
- Klasse: Kombination aus Verhaltens- und Zustandsbeschreibung

Implementierung

- Repräsentation (Datenstruktur, abgeleitet aus dem abstrakten Zustand des Typs)
- Menge von Methoden

Vordefinierte Typen im ODMG-Modell

- Object_Type
 - Atomic_Object
 - Collection_Object
 - Set < T >
 - Bag < T >
 - List < T >
 - Array < T >
 - Dictionary < T, V >
 - Structured_Object
 - Date
 - Time
 - Timestamp
 - Interval
- Literal_Type
 - Atomic_Literal
 - Long
 - Short
 - Unsigned Long
 - Unsigned Short
 - Float
 - Double
 - Char
 - Boolean
 - Octet
 - String
 - Enum < T >

Vordefinierte Typen im ODMG-Modell (2)

- Literal_Type (Fortsetzung)
 - Collection_Literal
 - Set < T >
 - Bag < T >
 - List < T >
 - Array < T >
 - Dictionary < T, V >
 - Structured_Literal
 - Date
 - Time
 - Timestamp
 - Interval
 - Structure < T >

Vordefinierte Typen im ODMG-Modell (2)

- Literal_Type (Fortsetzung)
 - Collection_Literal
 - Set < T >
 - Bag < T >
 - List < T >
 - Array < T >
 - Dictionary < T, V >
 - Structured_Literal
 - Date
 - Time
 - Timestamp
 - Interval
 - Structure < T >

Vordefinierte Operationen für einige Subtypen

```
Interface Collection : Object {
    exception InvalidCollectionType{ };
    exception ElementNotFound{any element; };
    unsigned long cardinality();
    boolean is_empty();
    boolean is_ordered();
    boolean allows_duplicates();
    boolean contains_element(in any element);
    void insert_element(in any element);
    void remove_element(in any element)
        raises(ElementNotFound);
    Iterator create_iterator(in boolean stable);
    BidirectionalIterator create_bidirectional_iterator
        (in boolean stable)
        raises(InvalidCollectionType);
};
```

Object Definition Language (ODL)

- ODL ist eine Spezifikationssprache, Definition von:
 - Interfaces,
 - Typen,
 - Klassen (für Objekte)
- Soll die Portabilität von Datenbank-Schemata zwischen verschiedenen Systemen unterstützen

Definition eines Typs in ODL

- Erfolgt durch Spezifikation eines Interfaces oder einer Klasse:

```
interface          ::= interface_dcl | forward_dcl
interface_dcl     ::= interface_header {[ interface_body ]}
forward_dcl       ::= INTERFACE identifier
interface_header  ::= INTERFACE identifier [ inheritance_spec ]

class             ::= class_header { interface_body }
class_header      ::= CLASS identifier
                      [ EXTENDS scopedName ]
                      [ inheritance_spec ]
                      [ type_property_list ]
```

Schlüsselwörter: Großschreibung
(geschweifte Klammern stets Teil der Syntax, deuten keine Alternativen an)

Beispiel: Objekt Student

```
class Student
  ( extent students
    key ( student_id ))
{
  attribute String name;
  attribute String student_id;
  Struct Address { String college,
                    String room_number }
  relationship Set<Section>
    takes inverse Section::is_taken_by;
} ;
class TA extends Employee : Student
()
{ relationship Section assists
  inverse Section::has_TA; } ;
```

Grundzüge objektrelationaler Datenmodelle

Warum objektrelationale Modelle?

- Relationale Datenbanksysteme weithin eingeführt
- Vielzahl von Benutzerschnittstellen, -oberflächen und Werkzeuge für relationale Systeme verfügbar
- Anwender setzen zunehmend Objekttechnologien ein
- Relationales Modell mit flacher Tabellenstruktur bildet Daten auf Bildschirm / Papier adäquat ab
- Viele Schemata objektorientierter Datenbanken sind auf der obersten Stufe record- bzw. Tupelstrukturiert, erst darunterliegende Schachtelungsstufen verwenden Typen und Konstruktoren Gebrauch gemacht

Modellierungsfähigkeiten objektrelationaler Systeme

- Neue Typen lassen sich aus (Basis-)Typen definieren
 - Diese können im Kontext von SQL-Deklarationen verwendet werden
 - An benutzerdefinierte Typen lassen sich Prozeduren binden (Operationen, die nur auf diesem Typ ausführbar sind)
- Komplexe Objekte werden im SQL-Kontext unterstützt
- Relation kann als Spezialisierung einer anderen definiert werden
 - Erbt deren Attribute und ggf. Prozeduren
- Regeln können bei Updates eine Integritätssicherung garantieren

Objektorientierte Datenbanksprachen

- Konzepte von Sprachen für objektbasierte Datenmodelle:
 - Objektorientierte
 - Objektrelationale
- Beispielhafte Darstellung / Bemerkungen zu:
 - ODMG-Sprache **Object Query Language**
- Nicht behandelt werden:
 - SQL:1999 (wächst zunehmend mit OQL zusammen)
 - O₂ (entwickelt von Altair in Frankreich, vertrieben von der Firma O₂ Technology)
 - Object Store von Object Design

Spezielle Eigenschaften objektorientierte Datenbanksprachen

- Viele Sprachkonzepte sind Verallgemeinerungen relationaler Sprachen
- Prototypische Rolle der Relationenalgebra im relationalen Modell ist im Bereich objektorientierter Sprachen unerreicht

Eigenschaften

- Anwendungsunabhängigkeit
- Deskriptivität
- Optimierbarkeit
- Abgeschlossenheit
- Vollständigkeit
- Mächtigkeit
- Effizienz

Object Query Language (OQL)

- Von der ODMG definiert
- Grundlage ist das ODMG Datenmodell
- Eigenschaften
 - Deklarativ und optimierbar, nicht Turing-vollständig („klassische“ Anfragesprache)
 - An SQL angelehnt
 - Menge wird nicht als Anfragemedium favorisiert, z.B. Tupelstrukturen und Listen werden gleichwertig zu Mengen behandelt
 - Keine expliziten Änderungs-Befehle oder Operationen (zur Änderung von Objekten müssen entsprechende Methoden geschrieben werden)

SELECT in OQL

```
query      ::= SELECT [DISTINCT ] proj_attributes
              FROM variable_decl { , variable_decl }
              [ WHERE query ]
              [ GROUP BY partition_attributes ]
              [ HAVING query ]
              [ ORDER BY sort-crit { , sort-crit } ]

proj_attributes ::= projection { , projection } | *
projection      ::= query | identifier: query
                  | query AS identifier
variable_decl   ::= query [ [ AS ] identifier ]
partition_attributes ::= projection { , projection }
sort-crit       ::= query [ { ASC | DESC } ]
```

Objektrelationale Sprachen

Beispiel Postgres

- Nachfolger des relationalen Systems Ingres
 - University of California at Berkeley
- Datenmodell
 - Basis Relationenmodell
 - Typ eines Attributes:
 - atomar oder
 - strukturiert (Struktur legt Benutzer fest)
- Struktur-Definition
 - Typ-Name
 - Länge der internen Darstellung in Byte
 - Prozeduren zur Konvertierung von einer externen in eine interne Darstellung und umgedreht
 - Default-Wert

Schema Addierer in Postgres

```
CREATE CHIP ( CHIPID = INT4 ,  
              CHIPNAME = CHAR[ 30 ] ;  
              MODULE = POSTQUEL )  
KEY ( CHIPID )
```

```
CREATE MODUL ( MODID = INT4 ,  
                 MODNAME = CHAR[ 20 ] ;  
                 SUBMODULE = POSTQUEL )  
KEY ( MODID )
```

```
CREATE SUBMODUL ( SUBID = INT4 ,  
                   SUBNAME = CHAR[ 4 ] ;  
KEY ( SUBID )
```

Postgres-Beschreibung der Datenbank für den Addierer

<i>Chip</i>	<i>ChipId</i>	<i>ChipName</i>	<i>Module</i>
	125	4-Bit-Addierer	select * from Modul where ModId \geq 201 and ModId \leq 204
	:		

<i>Modul</i>	<i>ModId</i>	<i>ModName</i>	<i>SubModule</i>
	201	VA1	select * from SubModul where SubId \geq 301 and SubId \leq 303
	202	VA2	select * from SubModul where SubId \geq 304 and SubId \leq 306
	203	VA3	select * from SubModul where SubId \geq 307 and SubId \leq 309
	204	VA4	select * from SubModul where SubId \geq 310 and SubId \leq 312

<i>SubModul</i>	<i>SubId</i>	<i>SubName</i>
	301	HA11
	302	HA12
	303	ODER1
	304	HA21
	305	HA22
	306	ODER2
	307	HA31
	308	HA32
	309	ODER3
	310	HA41
	311	HA42
	312	ODER4

Ausblick

Bisher:

- Datenmodelle und Datenbankentwurf
 - Normalisierung und algorithmischer Schema-Entwurf
 - Integrität in relationalen Datenbanken
 - Objektbasierte Modelle

Weiterer Verlauf:

- Datenbanksprachen
 - Codd-vollständige (relationale) Sprachen
 - Sprachstandard SQL (Ergänzungen zum 2. Semester)
- Datenbanksystemtechnik

8. Datenbanksprachen

- Codd-vollständige (relationale) Sprachen
- Sprachen für Objektrelationale Datenbanken

Codd-vollständige (relationale) Sprachen

Zentrale Sprachparadigmen des Relationenmodells:

- Prozedurale Algebra
(Relationenalgebra)
- Deskriptive Kalküle
(Relationenkalküle)

Relationenalgebra

- Grundlegende Operationen auf den Relationen einer Datenbank
(gemeint ist eine flache relationale DB)
- Anfrageverarbeitung ist mengenorientiert
 - Relationale Operationen erzeugen aus einer oder mehreren Tupelmengen (Relationen) eine neue Tupelmenge (Relation)
 - Benutzer „navigiert“ also relationenweise (und nicht länger satzweise)
 - Deshalb bezeichnet man die Elemente einer relationalen DB als *Basisrelationen* (die abgeleiteten Relationen werden nicht in der DB gespeichert)

Projektion

Sei $R = (X, .)$ ein Relationenschema, $r \in \text{Rel}(X)$ und $Y \subseteq X$:

Def.: Projektion

$$\pi_Y(r) := \{y[Y] \mid \mu \in r\}$$

heißt Projektion von r auf Y . Dabei bezeichnet $\mu[Y]$ die Einschränkung des Tupels μ auf Y ($\mu[Y] \in \text{Tup}(Y)$).

Selektion

Sei $R = (X, .)$ ein Relationenschema, $r \in \text{Rel}(X)$ und $Y \subseteq X$:

Def.: Selektion

Es sei $A \in X$, $a \in \text{dom}(A)$ und $\Theta \in \{<, \leq, >, \geq, =, \neq\}$:

$$\sigma_{A\Theta a}(r) := \{\mu \in r \mid \mu(A) \Theta a\}$$

heißt Selektion von r bezüglich $A \Theta a$.

(Für den Fall $\Theta \in \{<, \leq, >, \geq\}$ sei dabei unterstellt,
dass $\text{dom}(A)$ geordnet ist.)

Es seien $A, B \in X$ mit $\text{dom}(A) = \text{dom}(B)$ und $\Theta \in \{<, \leq, >, \geq, =, \neq\}$:

$$\sigma_{A\Theta B}(r) := \{\mu \in r \mid \mu(A) \Theta \mu(B)\}$$

heißt Selektion von r bezüglich $A \Theta B$.

(Für den Fall $\Theta \in \{<, \leq, >, \geq\}$ sei dabei unterstellt,
dass $\text{dom}(A)$ bzw. $\text{dom}(B)$ geordnet sind.)

Rechenregeln

Es sei $r \in \text{Rel}(X)$:

$$(1) \quad Z \subseteq Y \subseteq X \Rightarrow \pi_Z(\pi_Y(r)) = \pi_Z(r)$$

$$(2) \quad Z, Y \subseteq X \wedge Z \cap Y \neq \emptyset \Rightarrow \pi_Z(\pi_Y(r)) = \pi_{Z \cap Y}(r)$$

$$(3) \quad \sigma_{C_1}(\sigma_{C_2}(r)) = \sigma_{C_2}(\sigma_{C_1}(r))$$

$$(4) \quad A \in Y \subseteq X \Rightarrow \pi_Y(\sigma_{A \Theta a}(r)) = \sigma_{A \Theta a}(\pi_Y(r))$$

$$A, B \in Y \subseteq X \Rightarrow \pi_Y(\sigma_{A \Theta B}(r)) = \sigma_{A \Theta B}(\pi_Y(r))$$

oder allgemeiner

$$\text{attr}(C) \subseteq Y \subseteq X \Rightarrow pi_Y(\sigma_C(r)) = \sigma_C(\pi_Y(r))$$

Umbenennung

Sei $R = (X, .)$ ein Relationenschema, $A \in X, B \notin X - \{A\}$,
 $\text{dom}(A) = \text{dom}(B)$ und $r \in \text{Rel}(X)$;
ferner sei $X' := (X - \{A\}) \cup \{B\}$.

$$\rho_{B \leftarrow A}(r) = \{\mu \in \text{Tup}(X') \mid (\exists v \in r) \mu[B] = v[A] \wedge \mu[X' - \{B\}] = v[X - \{A\}]\}$$

heißt Umbenennung (renaming) von r bezüglich A/B .

Vereinigung und Differenz

Seien $r, s \in \text{Rel}(X)$:

Def.: Vereinigung

$$r \cup s := \{\mu \in \text{Tup}(X) \mid \mu \in r \wedge \mu \in s\}$$

Def.: Differenz

$$r - s := \{\mu \in r \mid \mu \notin s\}$$

Damit kann man auch den Durchschnitt bestimmen:

$$r \cap s := r - (r - s)$$

Verbund

Def.: (natürlicher) Verbund

Es seien X_1, \dots, X_n Attributmengen und $r_i \in \text{Rel}(X_i)$ für $1 \leq i \leq n$:

$$\bowtie_{i=1}^n r_i := \{\mu \in \text{Tup}(\cup_{i=1}^n X_i) \mid (\forall i, 1 \leq i \leq n) \mu[X_i] \in r_i\}$$

Für $n=2$ ergibt sich daraus:

$$r_1 \bowtie r_2 := \{\mu \in \text{Tup}(X_1 X_2) \mid \mu[X_1] \in r_1 \wedge \mu[X_2] \in r_2\}$$

Ausdrücke der Relationenalgebra

Def.: Ausdrücke der Relationenalgebra:

Es sei $\mathbf{D} = (\mathbf{R}, \Sigma_{\mathbf{R}})$ ein Datenbankschema mit $\mathbf{R} = \{R_1, \dots, R_k\}$.

Die Menge RA (genauer $RA_{\mathbf{D}}$) der Ausdrücke der Relationenalgebra (über \mathbf{D}) wird rekursiv wie folgt definiert:

- (1) $R_i \in RA$ für alle $R_i \in \mathbf{R}$
- (2) Sind $E_1, E_2 \in RA$, A ein Attribut des Formats der durch E_1 definierten Relation, B nicht Element dieses Formats und C eine Selektionsbedingung, $X \subseteq \cup_{i=1}^k$, so ist
 $\sigma_C(E_1), \pi_X(E_1), \rho_{B \leftarrow A}(E_1), E_1 \bowtie E_2, E_1 \cup E_2, E_1 - E_2 \in RA$;
- (3) nur solche Ausdrücke gehören zu RA , welche durch wiederholte (und endlich häufige) Anwendung von (1) und (2) entstehen.

Semantik

Sei $D = (R, .)$ und $E \in RA$. Die Auswertung von E bezüglich $d \in \text{Dat}(R)$ wird wie folgt definiert:

$$v_E(d) := \begin{cases} r_i & \text{falls } E = R_i \\ \sigma_c(v_{E_1}(d)) & \text{falls } E = \sigma_C(E_1) \\ \pi_X(v_{E_1}(d)) & \text{falls } E = \pi_C(E_1) \\ \rho_{B \leftarrow A}(v_{E_1}(d)) & \text{falls } E = \rho_{B \leftarrow A}(E_1) \\ v_{E_1}(d) \bowtie v_{E_1}(d) & \text{falls } E = E_1 \rightarrow E_2 \\ v_{E_1}(d) \cup v_{E_1}(d) & \text{falls } E = E_1 \cup E_2 \\ v_{E_1}(d) - v_{E_1}(d) & \text{falls } E = E_1 - E_2 \end{cases}$$

Erklärung der Abkürzungen

$D = (R, \Sigma_R)$	Datenbankschema
d	Datenbank
$r_i \in Rel(R)$	Relation r_i ist Element der Menge aller Relationen über die Attributmenge R
σ_X	Selektion
$\pi_{B \leftarrow A}$	Projektion
$v_{E_1}(d)$	ist der hier auszuwertende semantische Ausdruck
\bowtie	(natürlicher) Verbund (natural join)
\cup	Vereinigung
-	Differenzmenge

Beispiel Web-Anwendung

Machine	<u>IPaddress</u>	DomainName	OStype
	string	string	string
	128.176.159.168 128.176.158.86 128.176.6.1 :	ariadne.uni-ms.de helios.uni-ms.de www.uni-ms.de	Unix Unix Unix

Document	<u>URL</u>	Content	Created
	string	string	date
	www.uni-ms.de/index.html www.uni-ms.de/db.html	Text Text :	18.10.1997 20.10.1997

ConnStructure	<u>SourceURL</u>	<u>TargetURL</u>
	string	string
	www.uni-ms.de/index.html www.uni-ms.de/index.html	www.uni-ms.de/index.html www.uni-ms.de/db.html :

User	<u>EmailAddress</u>	Name
	string	string
	lechten@helios.uni-ms.de vossen@helios.uni-ms.de	Jens Gottfried :

SessionLog	<u>User</u>	<u>ClientIP</u>	SeqNo	URL
	string	string	integer	string
	lechten@... lechten@... :	128.176.159.168 128.176.159.168	1 2	www.uni-ms.de/index.html www.uni-ms.de/db.html

Vossen, G. (2000): Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme. Oldenbourg Wissenschaftsverlag.

Eigenschaften der Relationenalgebra

- RA ist abgeschlossen
 - Relationen werden stets in Relationen überführt
 - Ergebnis der Auswertung einer RA-Operation kann stets als „Eingabe“ für die nächste dienen
- RA ist eine sichere Sprache
 - Ergebnisse sind von endlicher Kardinalität (d.h. enthalten endlich viele Tupel)
 - Das gilt nicht für alle Alternativen zur RA!
- RA kann auch zur Spezifikation von (externen) Sichten über einem konzeptionellen Schema **D** eingesetzt werden
 - Benutzersicht kann durch Ausdruck $E \in RA$ ausgedrückt werden
- RA Ausdrücke lassen sich effizient auswerten
- Ausdruckskraft der RA ist beschränkt

Relationenkalküle

- **RA** ist eine prozedurale Sprache
 - Ausdruck einer prozeduralen Sprache gibt an, **wie** das Ergebnis berechnet werden kann
- **Relationenkalküle:**
 - Beschreiben die Tupel einer Ergebnisrelation durch **Eigenschaften**
 - Liefern keine Berechnungsprozedur mit
- **Äquivalent:**
 - Zwei gleich ausdrucksstarke Sprachen
- **Codd-vollständig**
 - Sprache ist so ausdrucksstark wie RA

Relationenkalküle (2)

- Relationen-Tupelkalkül (RTK)
 - Mit Sprache der Prädikatenlogik erster Stufe verwandt
 - Variablen werden für Tupel einer Relation verwendet
 - Aus praktischen Gründen gewisse Einschränkungen notwendig
- Relationen-Domainkalkül (RDK)
 - Wesentlicher Unterschied zu RTK:
 - Verwendung von Variablen für die Elemente von Wertebereichen (Domainvariablen) anstelle von Tupelvariablen

Einzelheiten: vgl. Literatur

10. Datenbanksystemtechnik

- Interne Organisation von Datenbanken
 - (Verarbeitung und Optimierung von Anfragen)
 - (Physischer Datenbankentwurf und Tuning)

Interne Datenbank- und Speicherorganisation

- Grundlagen des Aufbaus der Systemebene
- Organisationsformen des Sekundärspeichers
 - Plattenspeicher
 - File-organisationsformen

Anwendersicht:

- Teil des physischen Datenbankentwurfs
- beeinflußt Effizienz einer Anfrageauswertung unmittelbar

Entwicklersicht:

- welche Datenstrukturen benötigt ein bestimmtes System

Plattenspeicher

- Performance zentraler Aspekt einer Datenbank (also die Transaktionsrate)
 - Anfragen / Update müssen auf den Datenstrukturen effizient ausgeführt werden
 - dauerhafte Speicherung großer Datenmengen
- Übliche Speicherhierarchie:
 - Primärspeicher
 - Cache
 - Hauptspeicher
 - Sekundärspeicher
 - magnetische Platten
 - Tertiärspeicher (Archivierung)
 - magnetische Bänder
 - optische Platten

Komponenten der Speicherverwaltung

Zentrale Komponenten eines Datenbanksystems auf der Funktionsebene der Speicherverwaltung:

- Geräte- und Sekundärspeichermanager
 - Data Manager
 - verwaltet Hardware-Betriebsmittel
 - Zugriff auf gespeicherte Daten
 - Zuweisung von Plattenplatz
- Puffermanager
 - Datentransfer zwischen Primär- und Sekundärspeicher

Plattenspeicher wird meist vom OS verwaltet!

Blöcke und Blockzugriffe

- Optimierung von Plattenzugriffen
- Ausgleich der verschiedenen Zugriffsgeschwindigkeiten auf Primär- und Sekundärspeicher
-> Transfer einer zusammenhängenden Folge von Sektoren (Block)

Logische Sicht:

- Unterteilung von Files in Blöcke

Zuteilung von Plattenplatz an Files durch OS:

- zusammenhängend (wenig Kopfbewegungen)
- verkettete Liste von Blöcken (sequentieller Zugriff gut unterstützt)
- indiziert (zu jedem Block wird ein Index gespeichert)

RAID-Architekturen

- Plattsenspeicher ist preiswert geworden
- größere Anzahl von Platten in einem Rechner
- RAID-0 Striping
- RAID-1 Spiegelung
- RAID-2 Bit-Striping + Codierung (zusätzliche Bits, z.B. Hamming-Codes)
- RAID-3 Bit-Striping + Paritätsbit auf Zusatzplatte
- RAID-4 blockorientiertes Striping + Paritätsblock auf zusätzlicher Platte
- RAID-5 blockorientiertes Striping mit verteilter Parität
- RAID-6 Reed-Solomon-Codes (Ausfall von 2 Platten)

Pufferverwaltung

- Datenbank bzw. ihre Files sind in Blöcke fester Größe partitioniert
- Transfer von Blöcken zwischen Primär- und Sekundärspeicher notwendig

Ziele:

- Möglichst wenig Blöcke transferieren!
- Soviel Blöcke als möglich im Hauptspeicher belassen.

Lösung: Pufferbereiche

Pufferverwaltung (2)

Ähnlich Verwaltung virtueller Speicher, aber:

- bei vollem Puffer Blöcke entfernen
 - OS verwendet z.B. Least Recently Used
 - in Datenbank können Blöcke eventuell sofort wieder ausgelagert werden (z.B. nach Verbundrechnung)
- OS kann auszulagernde Blöcke im Primärspeicher häufig einfach überschreiben (z.B. Programmcode)
 - Datenbank muß immer erst prüfen, ob Daten geändert wurden
- Datenbanksystem muß nach Fehler kontrolliert wieder anlaufen
 - regelmäßige Sicherheitskopien --> während Sicherung nicht in Sekundärspeicher schreiben!

Files

Logische Sicht: die transferierten Blöcke enthalten die Records eines Files

- jedes OS verfügt über ein Filesystem
 - DBMS wird für persistente Speicherung meist darauf zurückgreifen
 - DBMS speichert Daten in Files
- File:
 - ein oder mehrere Blöcke
 - jeder Block ein oder mehrere Records
 - relationale Datenbank:
 - Relation <--> File
 - Tupel <--> Records
 - Blocklänge fest, Länge Records variabel – genaue Zuordnungsvorschriften erforderlich
- DBMS muß Operationen des Nutzers an Datenbank zu Operationen auf Files transformieren

Organisation der Files

- OS stellt nur einige Fileoperationen zur Verfügung:
 - create, delete, open, close, append, read, write
- DBMS muß Fileblöcke zur Bearbeitung des Auftrages in Primärspeicher transferieren:
 - Anfragen
 - Updates
 - Löschen usw.
- Nutzer an schneller Bearbeitung interessiert
- Forderung seitens DBMS: vorhandene Files so organisiert, daß die effiziente Ausführung der Operationen möglich ist

Sequentielle Files

- Relation mit n Tupeln
- jedes Tupel in einem Record gespeichert
- Block nehme b Records auf

Y	L1	Smith	London	20	Block 1
	L2	Jones	Paris	10	
	L3	Blake	Paris	30	
Y	L4	Clark	London	20	Block 2
	L5	Adams	Athen	30	
	L6	Hart	Chicago	80	
Y	L7	Parker	New York	70	Block 3
	L8	James	Athen	25	
					(ein Freiplatz)

Sequentielle Files

- Optimierung Suchaufwand durch Sortierung
- Einfügen neuer Sätze in sortiert gespeichertem File:
 - einfügen auf beliebigem Freiplatz
 - neue Sortierung
 - Stelle im File bestimmen, an die Datensatz gehört
 - alle nachfolgenden Sätze um einen Platz nach hinten verschieben
 - im entstandenen Freiplatz einfügen
- Löschen:
 - Freiplätze in der Sortierung zugelassen oder nicht?

Indexierung

- Beschleunigung
 - des Zugriffs auf Records
 - von Suchoperationen
- zusätzliches „Inhaltsverzeichnis“
- Beispiele für einen (dichten) Index:

Name	„Adresse“
Adams	5
Blake	3
Clark	4
Hart	6
James	8
Jones	2
Parker	7
Smith	1

Stadt	„Adresse“
Athen	5
Chicago	6
London	1
New York	7
Paris	2

Spezielle Indexstrukturen

- Indexierung dient der Beschleunigung des Zugriffs
 - Weg zum Speicherort wird durch Berechnungen oder Abkürzungspointer vereinfacht
- Zwei Klassen:
 - Geordnete Strukturen
 - Auf Suchschlüsseln basierende Ordnung
 - Hash-Strukturen
 - Ungeordnet
 - Unterstellen Gleichverteilung der vorkommenden Schlüssel
- Unterscheidungsmerkmale:
 - Art des Zugriffs
 - Zugriffszeit
 - Aufwand für Einfügen / Löschen
 - Platzbedarf

ISAM

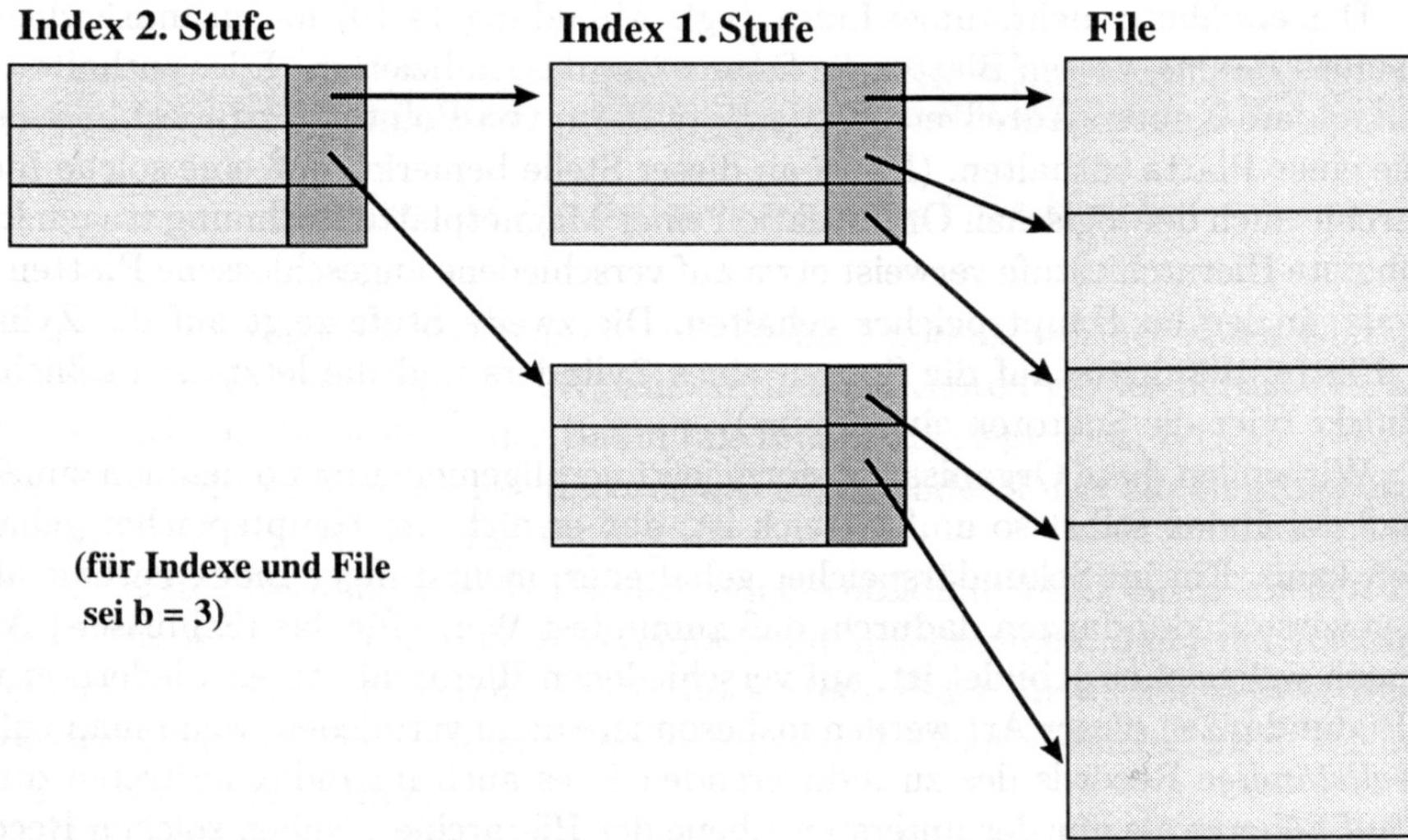
- Index für einen (etwa nach aufsteigenden Werten des Primärschlüssels) gespeicherten File
- Insbesondere bei dünnem Index besteht Zugriff auf den File aus Index-Suche und sequentiellem Zugriff auf die Records selbst
- Indexed Sequential Access Method (index-sequentielles File)
- Index für den Primärschlüssel: Primärindex
- Index für Sekundärschlüssel oder Attributkombination: Sekundärindex
- Zur Verkürzung der Suchzeit kann der Index selbst wie ein File behandelt und indiziert werden

Invertierte Liste

Stadt	„Adresse(n)“
Athen	5,8
Chicago	6
London	1,4
New York	7
Paris	2,3

- Sonderfall Sekundärindex:
 - Alle Records erfaßt (und nicht nur einer)
- **Invertierte Liste**

Hierarchische Organisation von Indexen



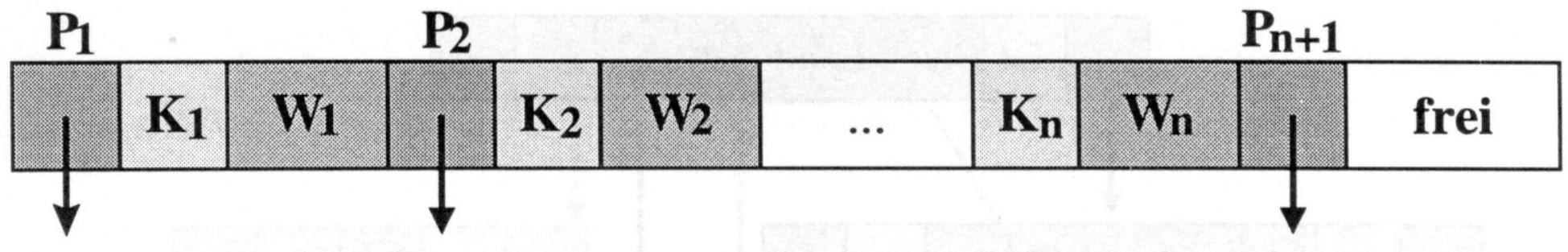
Baumstrukturen für eindimensionale Punktdaten

- Nachteil index-sequentielle Organisation von Files:
 - mit zunehmender Filegröße nimmt Performance von Anfrage- und Updateoperationen ab
 - kann nur durch Reorganisation vermieden werden (aufwendig)
- Strukturen gesucht, die effiziente Suchoperationen ermöglichen und zur Reorganisation nur lokale Änderungen erfordern
 - mehrstufiger Index hat Baumform
- Annahme: Index so umfangreich, daß nicht im Primärspeicher
 - ein mehrstufiger Index im Sekundärspeicher enthält gewöhnlich gewisse Redundanzen
 - wird vermieden, indem wir Records im Index zulassen

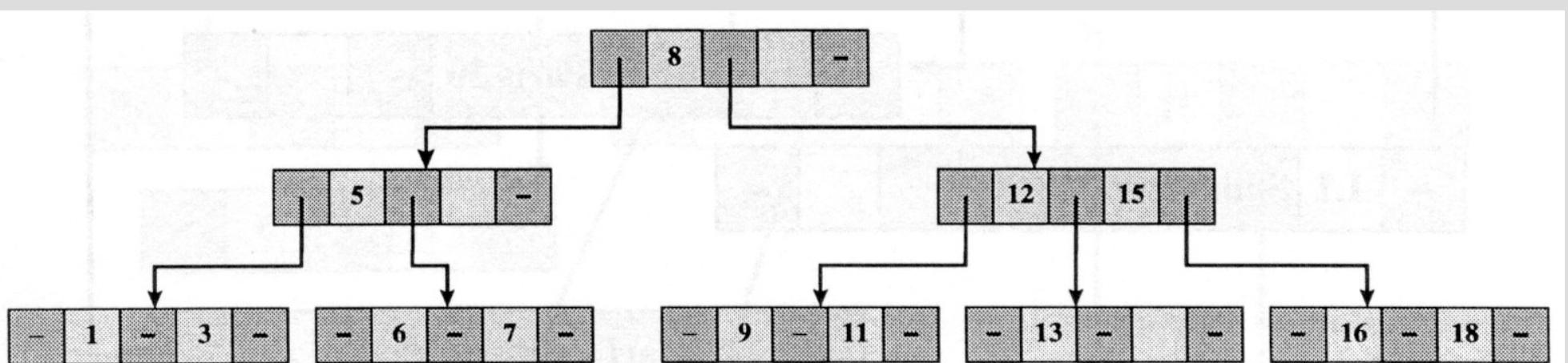
B-Baum

- Balancierter Baum oder Bayer-Baum (nach Autor)
- Gerichteter Baum, jeder Pfad von Wurzel zu Blatt hat die gleiche Länge (= Höhe des Baumes)
 - jeder Knoten repräsentiert einen Speicherblock fester Länge, enthält mindestens k und maximal $2k$ Records (konstanter Länge)
 - Record besteht aus Schlüssel- und Nichtschlüsselanteil
 - jeder Block ist nach aufsteigenden Schlüsselwerten sortiert und jeder Vater stellt einen Index für seine Söhne dar
 - die Wurzel hat keinen oder mindestens 2 Söhne
 - jeder innere Knoten, welcher weder Wurzel noch Blatt ist, hat die für ihn maximal mögliche Zahl von Söhnen, d.h. er hat $n+1$ Söhne, falls er n Schlüsselwerte enthält

B-Baum: Knoten und Baum

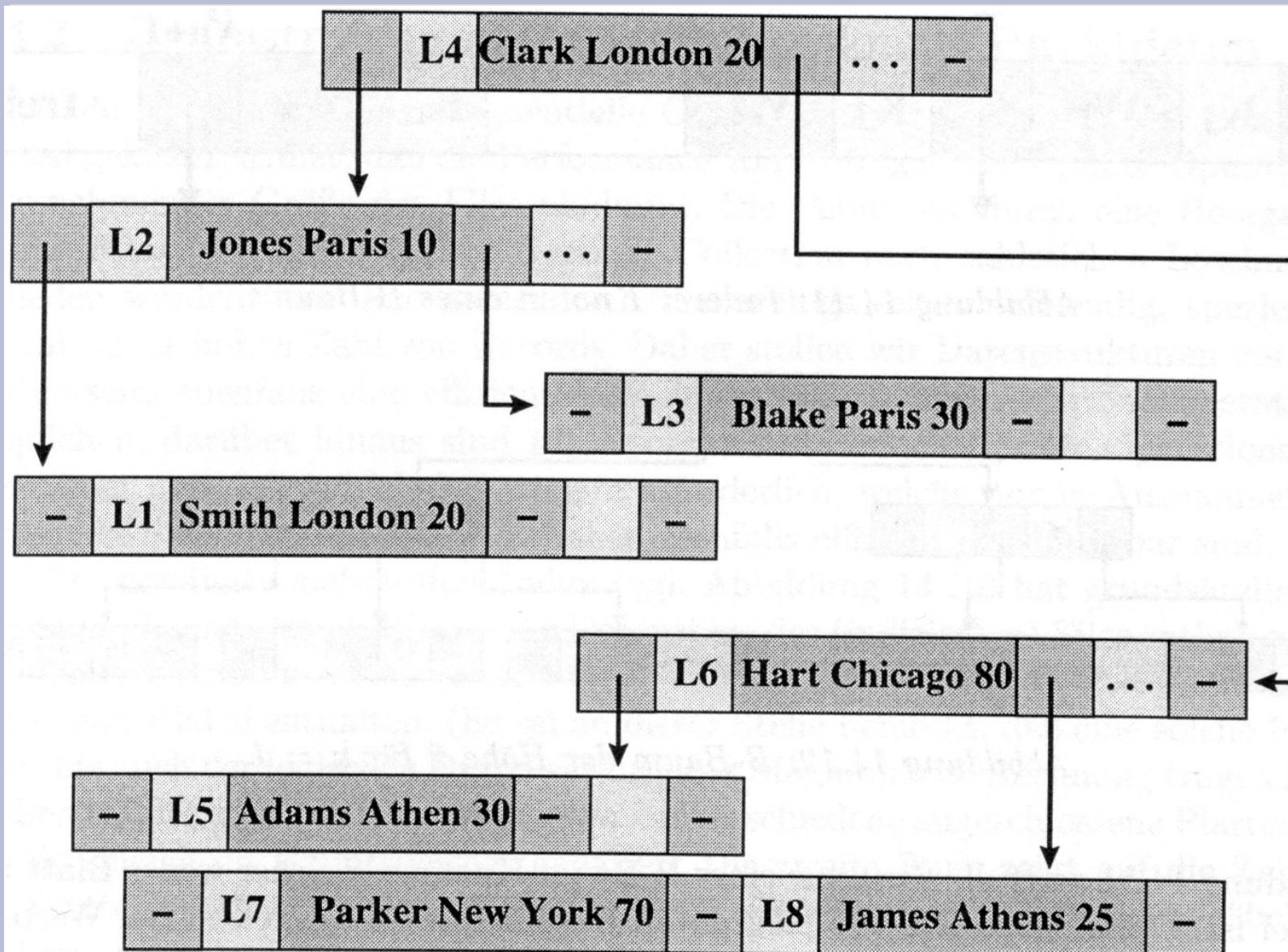


Innerer Knoten eines B-Baums.



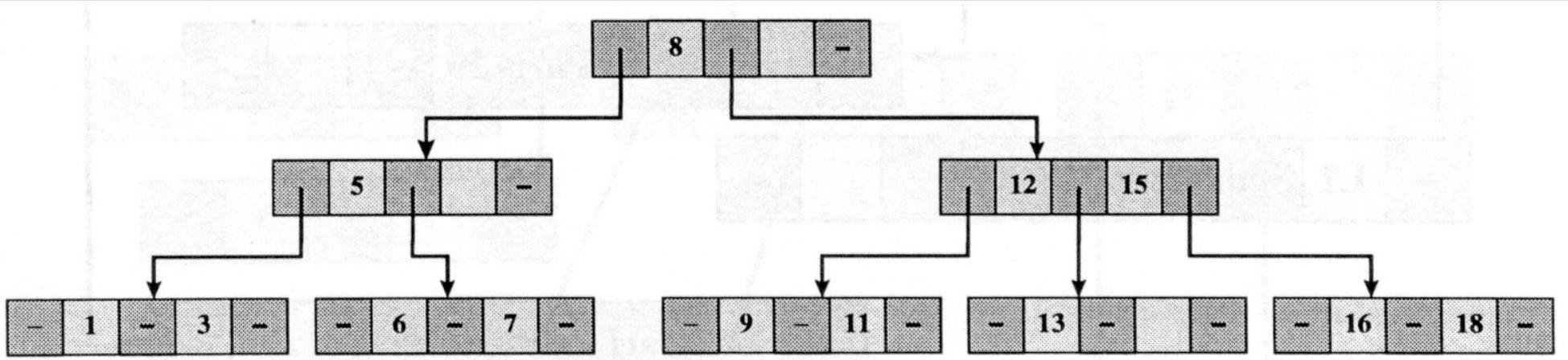
B-Baum der Höhe 2 für $k=1$.

Beispiel eines B-Baums



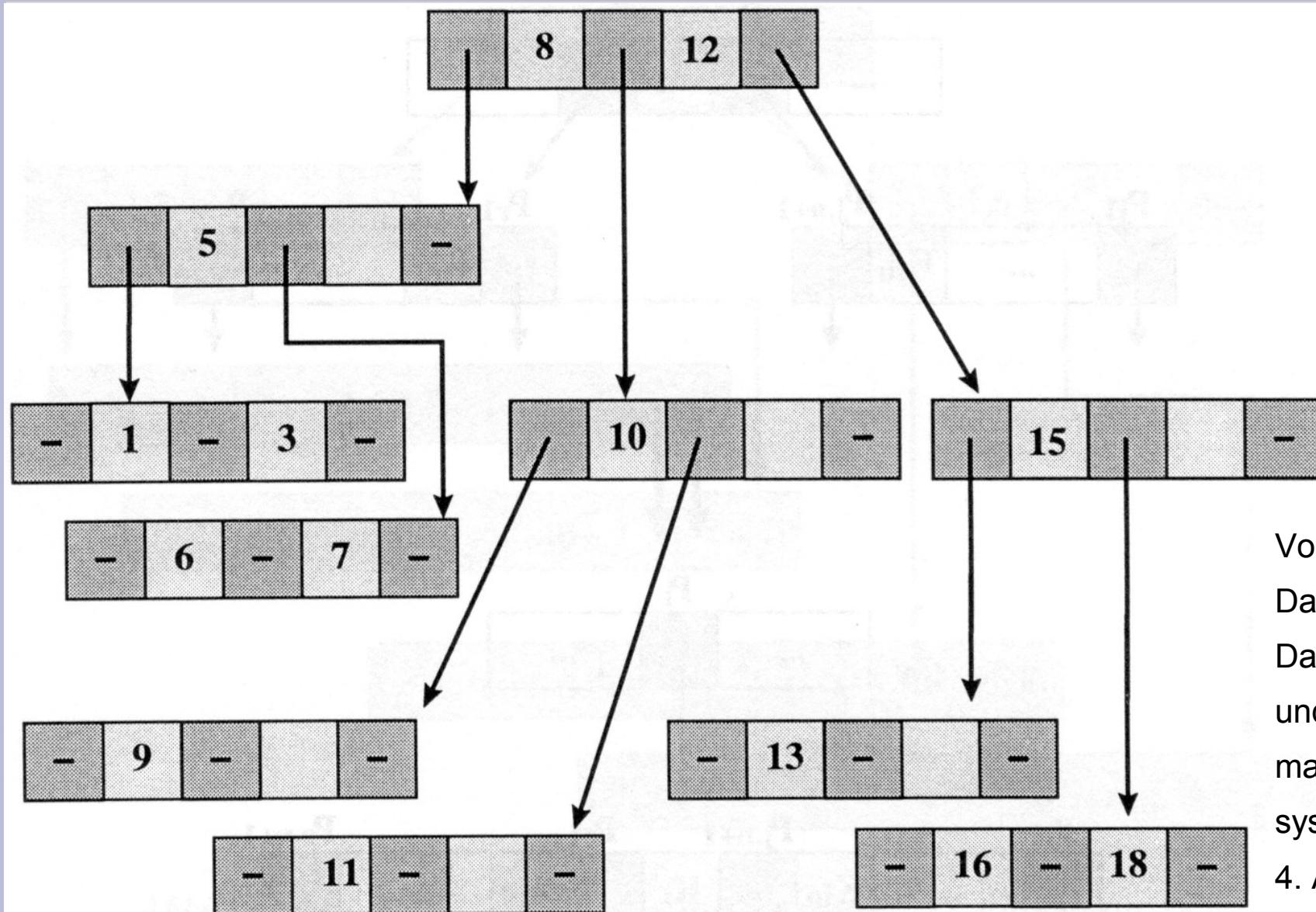
Vossen, G. (2000):
Datenmodelle,
Datenbanksprachen
und Datenbank-
management-
systeme.
4. Aufl., Oldenbourg.

B-Baum vor Einfügen



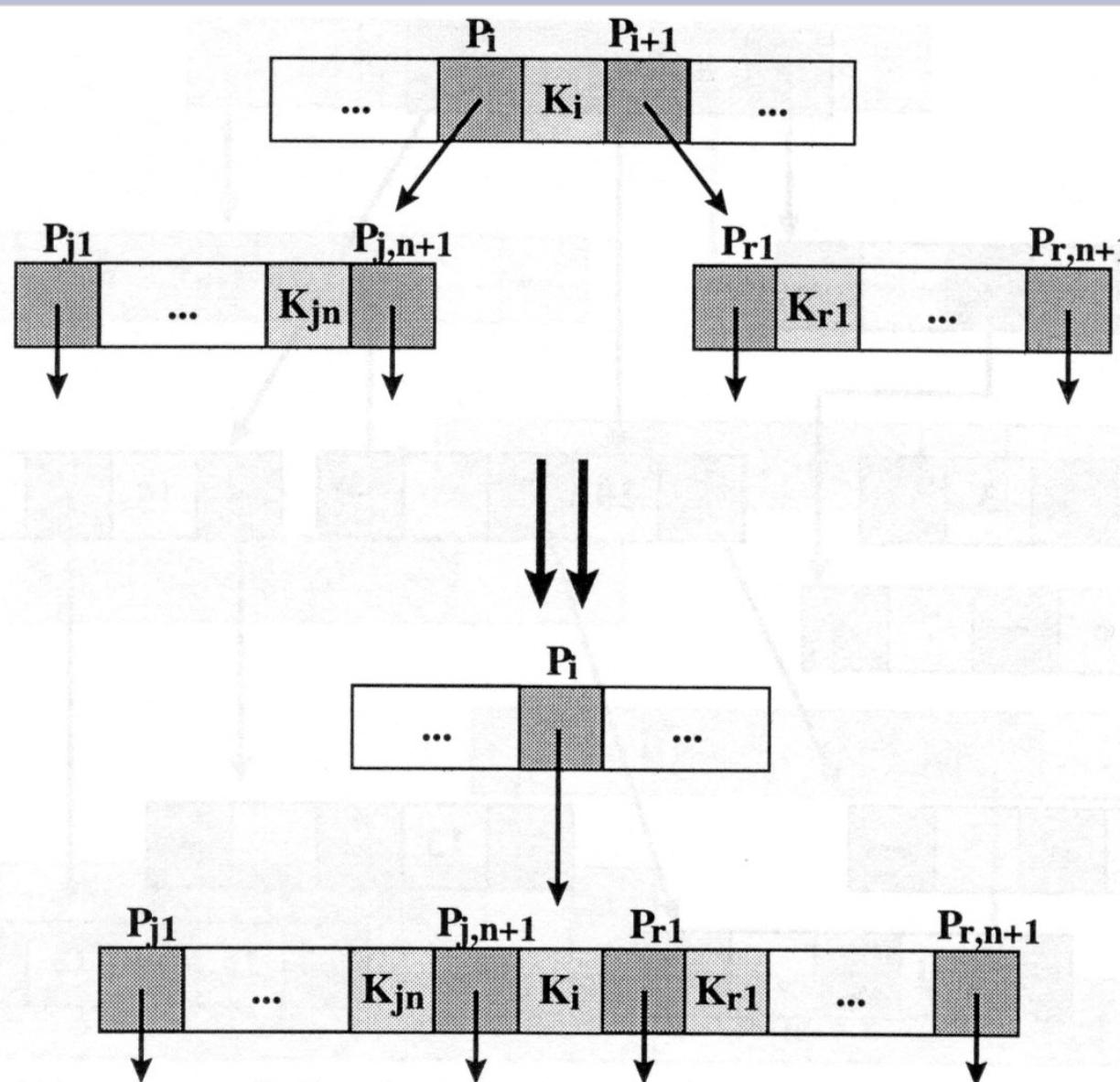
B-Baum der Höhe 2 für $k=1$.

B-Baum nach Einfügen von 10



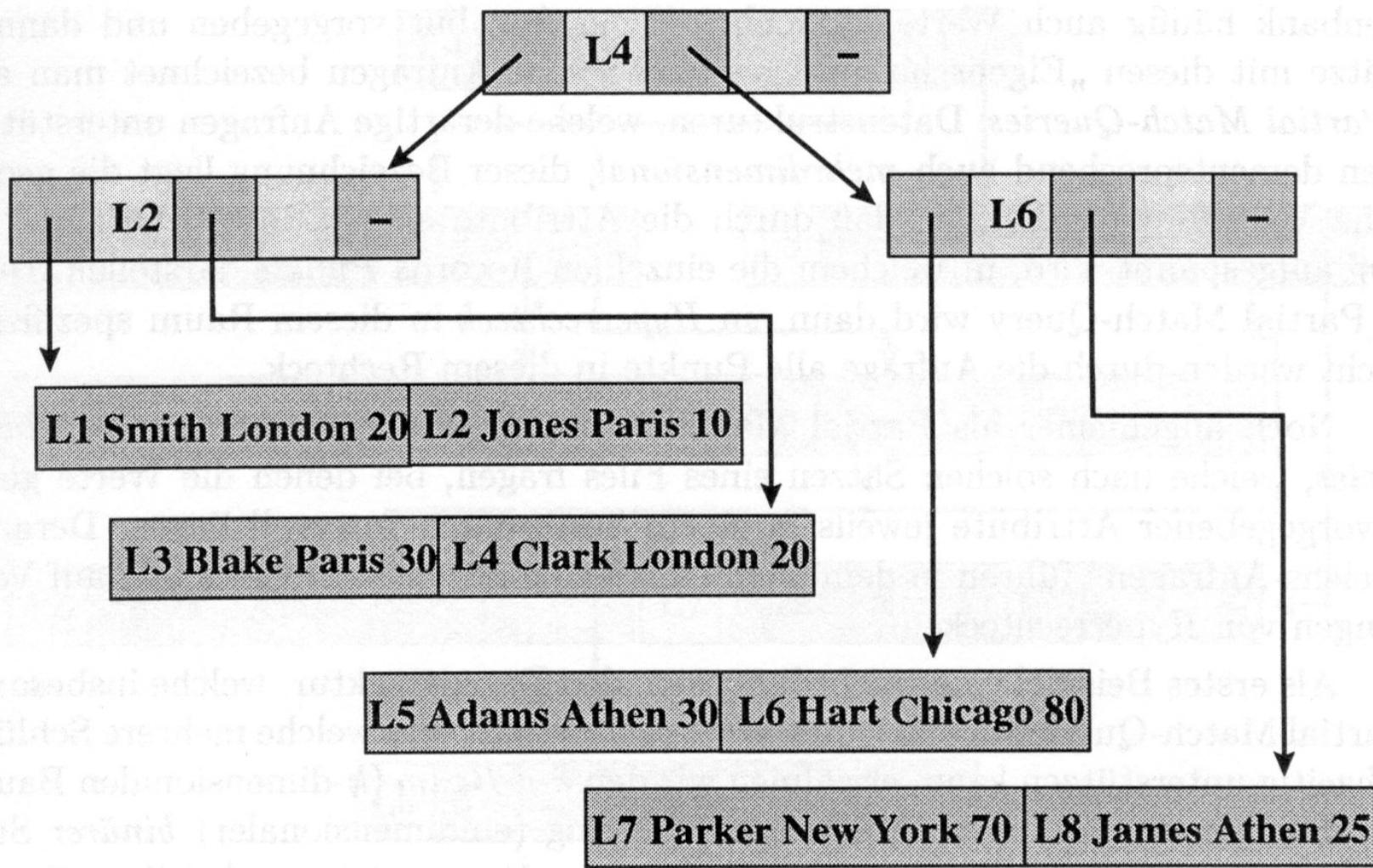
Vossen, G. (2000):
Datenmodelle,
Datenbanksprachen
und Datenbank-
management-
systeme.
4. Aufl., Oldenbourg.

Zusammenfassung von Knoten nach einer Löschung aus B-Baum

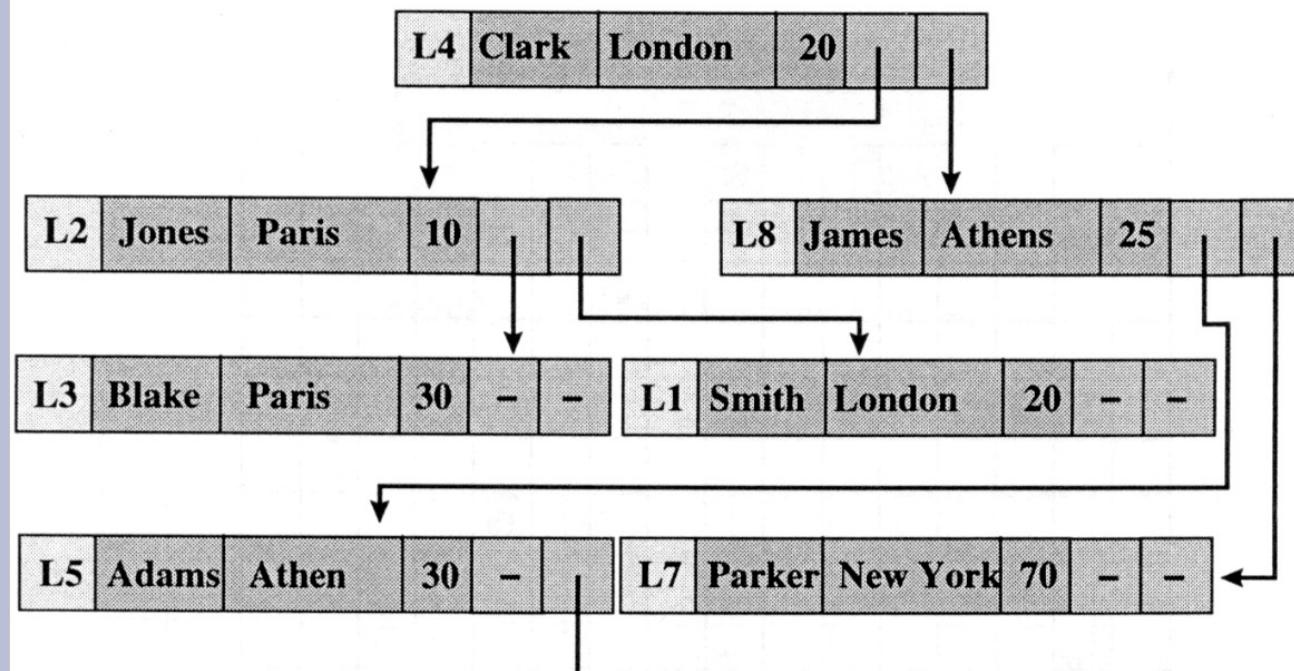


Vossen, G. (2000):
Datenmodelle,
Datenbanksprachen
und Datenbank-
management-
systeme.
4. Aufl., Oldenbourg.

B*-Baum



3-d-Baum und 2-d-Baum-Knoten



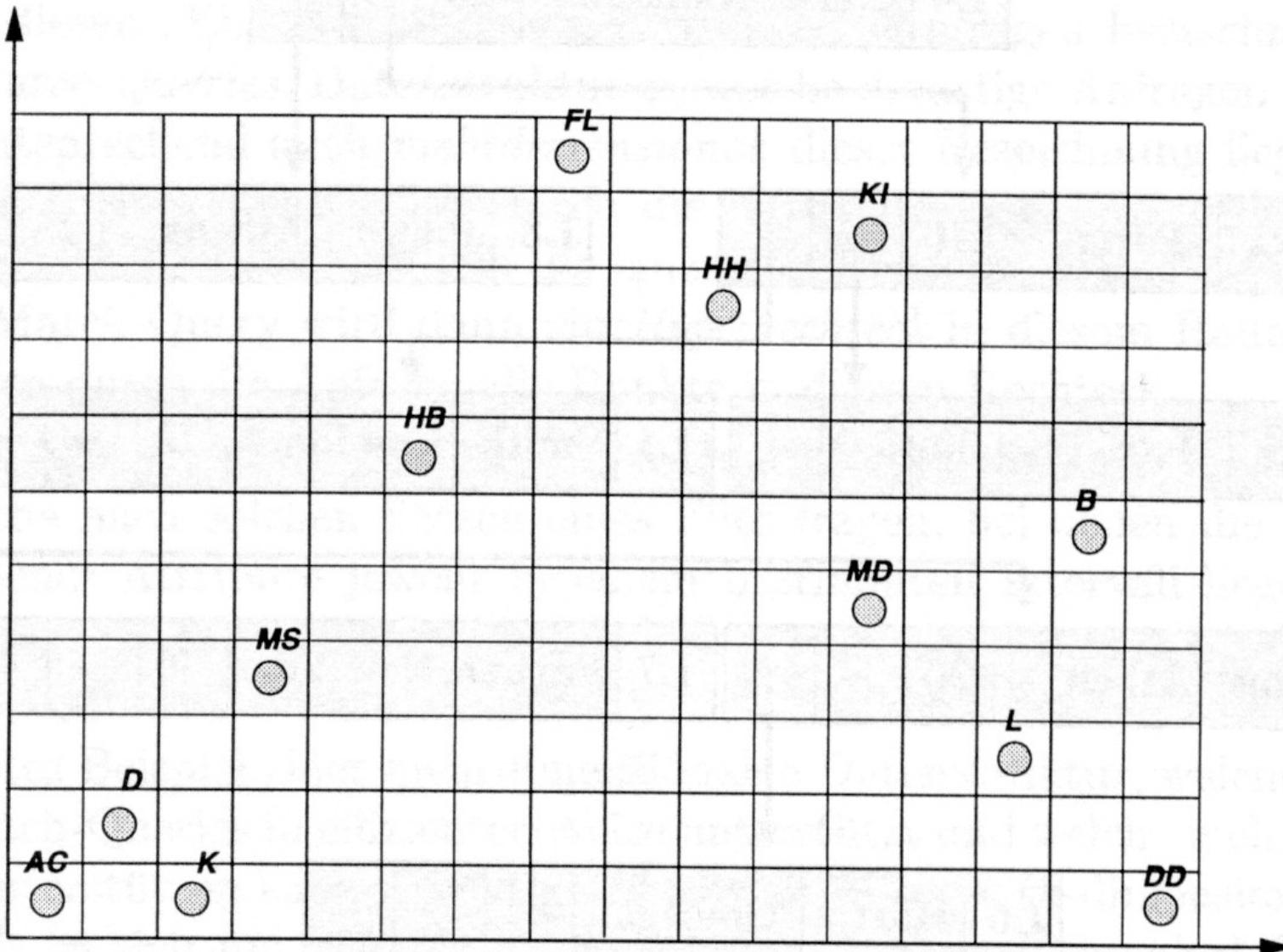
3-d-Baum



2-d-Baum-Knoten

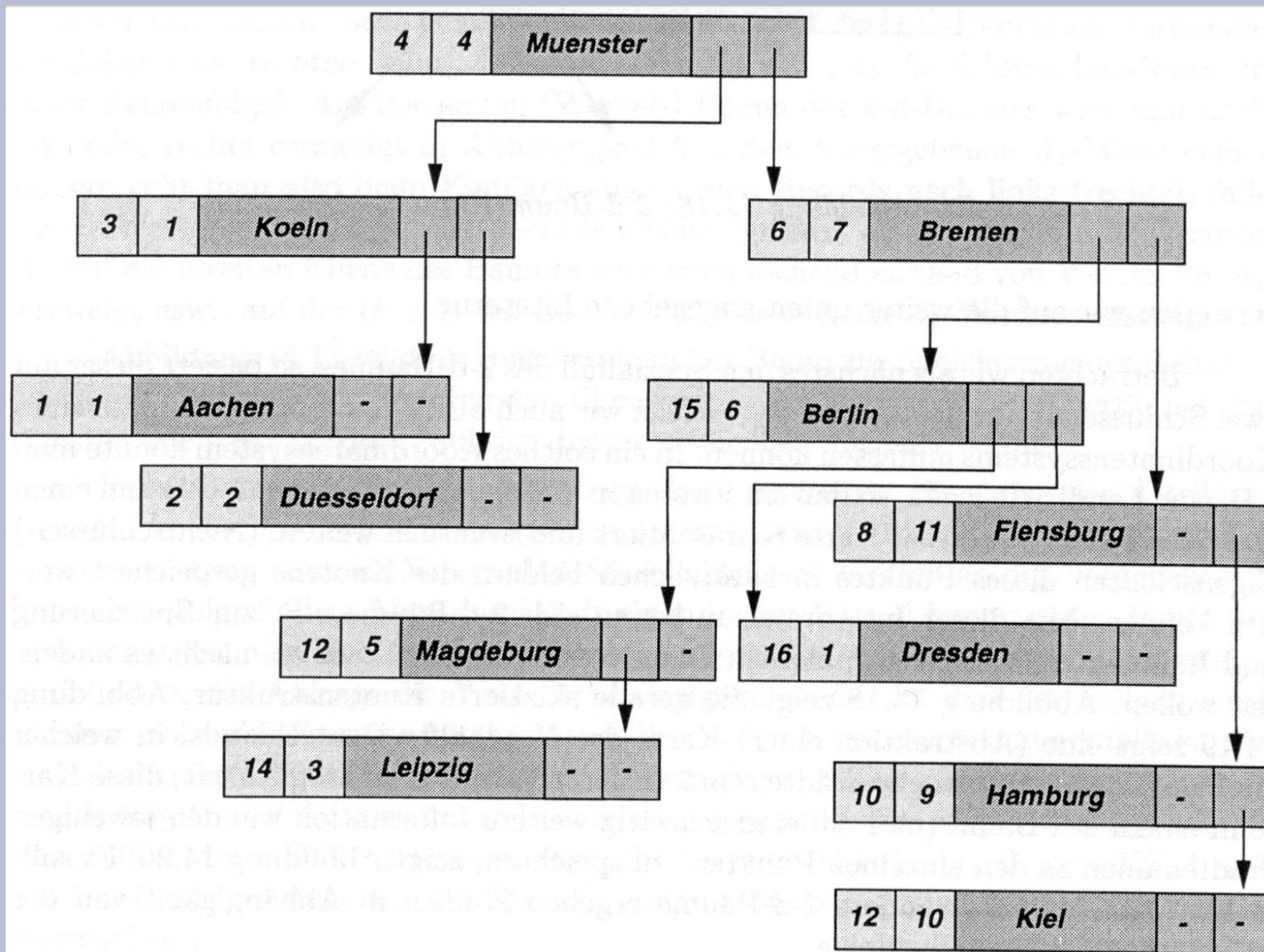
Vossen, G. (2000):
Datenmodelle,
Datenbanksprachen
und Datenbank-
management-
systeme.
4. Aufl., Oldenbourg.

Stilisierte Karte der Nordhälfte Deutschlands



Vossen, G. (2000):
Datenmodelle,
Datenbanksprachen
und Datenbank-
management-
systeme.
4. Aufl., Oldenbourg.

2-d-Baum zur Repräsentation der Karte



Vossen, G. (2000):
Datenmodelle,
Datenbanksprachen
und Datenbank-
management-
systeme.
4. Aufl., Oldenbourg.

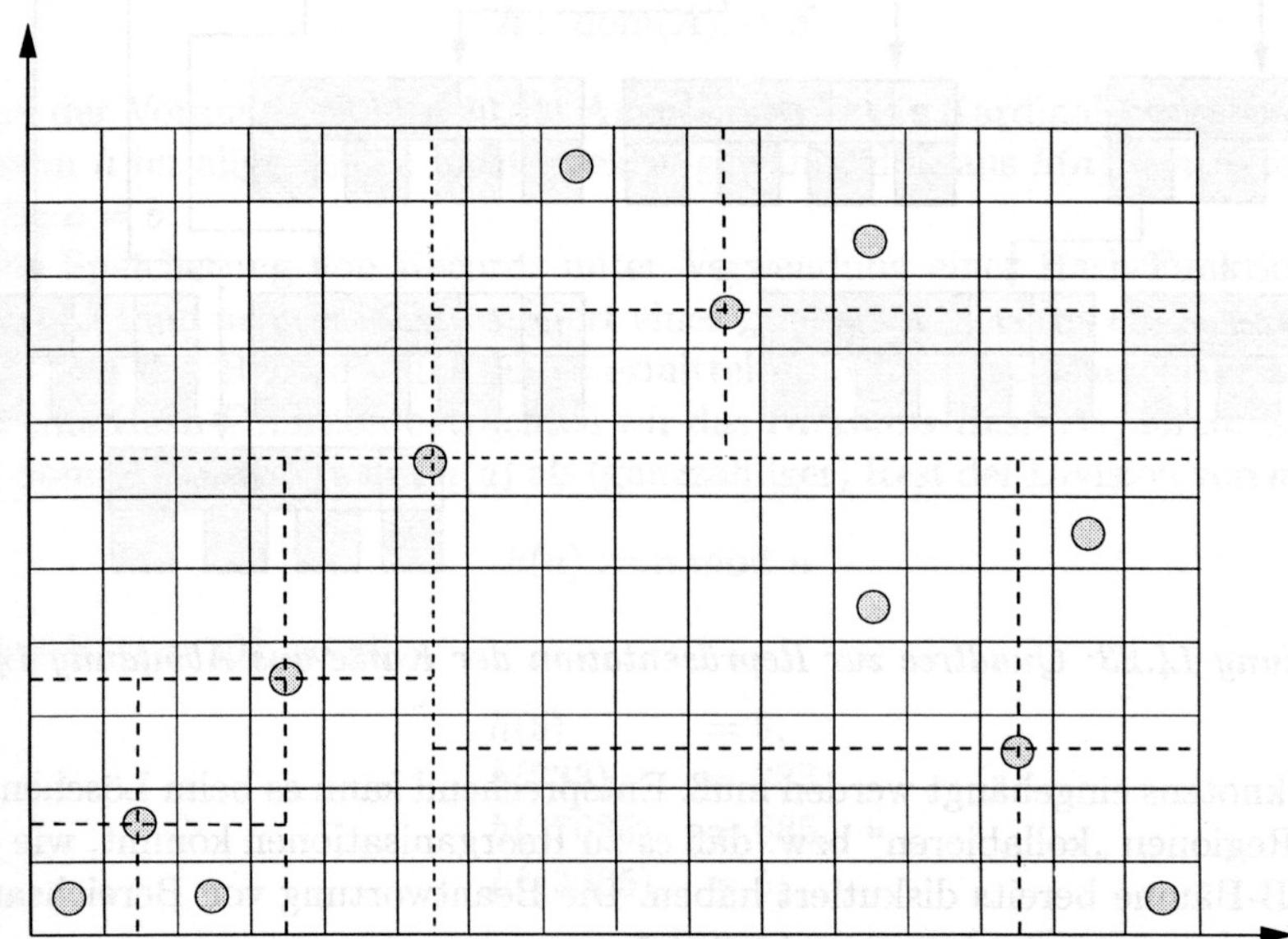
Zerlegung der Nordhälfte Deutschlands in Regionen

X	Y	Daten	
NW	SW	NO	so

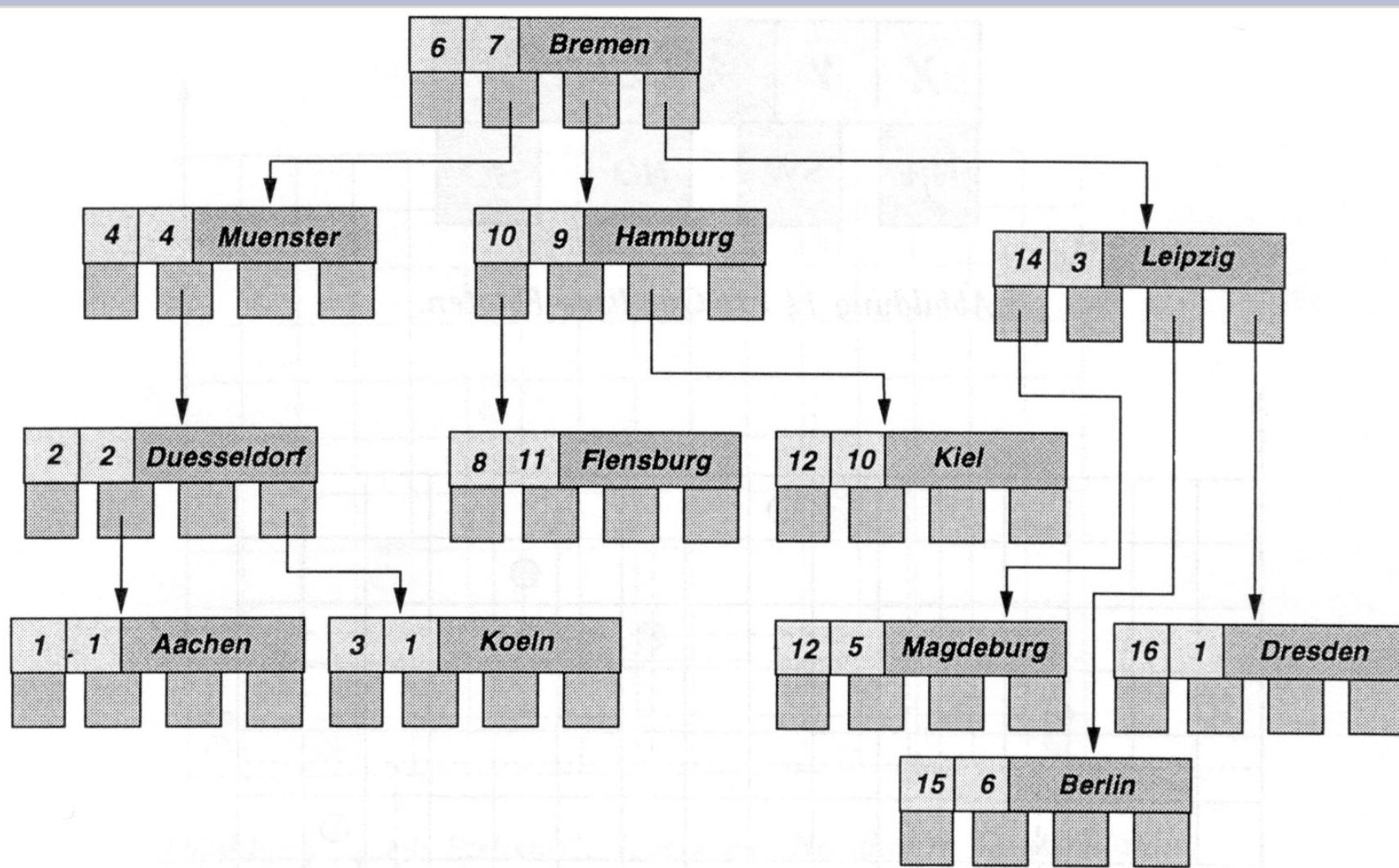


Quadtree-Knoten

Vossen, G. (2000):
Datenmodelle,
Datenbanksprachen
und Datenbank-
management-
systeme.
4. Aufl., Oldenbourg.



Quadtree zur Repräsentation der Karte



Hash-Organisationsformen

- Bisher:
 - Bestimmter Pfad war bei jedem Zugriff zu durchlaufen
 - d.h. man muß den Index traversieren
- Hash-Technik vermeidet das:
 - Adresse eines Schlüsselwertes wird direkt aus vorgegebenen Schlüsselwerten berechnet
 - Records werden dabei über den Speicherbereich gestreut --> gestreute Speicherung

Grundidee

- gegeben sei Record-Typ, bestehend aus Attributen
- Schlüssel sei A , besteh o.B.d.A. aus einem Attribut mit numerischen Werten
 $\text{dom}(A) = \{1, 2, \dots, 10^i\}$
- nicht alle 10^i Records sind im allgemeinen gleichzeitig in der Datenbank gespeichert
 - im Falle einer Relation r_i befindet sich aktuell nur die aktive Domain von A im Speicher
 - im allgemeinen gilt:
 $| \text{adom}(A, r_i) | \ll | \text{dom}(A) |$
 - es ist nicht erforderlich (und in der Praxis meist unmöglich) Speicherplatz für die Menge aller möglichen Records bereitzustellen

Hash-Funktion

- Eine Hash-Funktion ist eine Abbildung von $\text{dom}(A)$ in S :

$$h: \text{dom}(A) \rightarrow S$$

- aufgrund der Voraussetzung bzw. der Annahme über die Kardinalitäten von $\text{dom}(A)$ und S kann h im allgemeinen nicht injektiv gewählt werden, d.h.

*aus $h(a) = h(b)$
folgt nicht notwendig
 $a = b$*

Beispiel für Hash-Funktion

- Divisions-Rest-Verfahren:

Für $|S| = n$ und $a \in dom(a)$
Sei $h(a) := a \ mod \ n$

- Beispiel für $n = 10^3$:

$$h(5) = 5$$

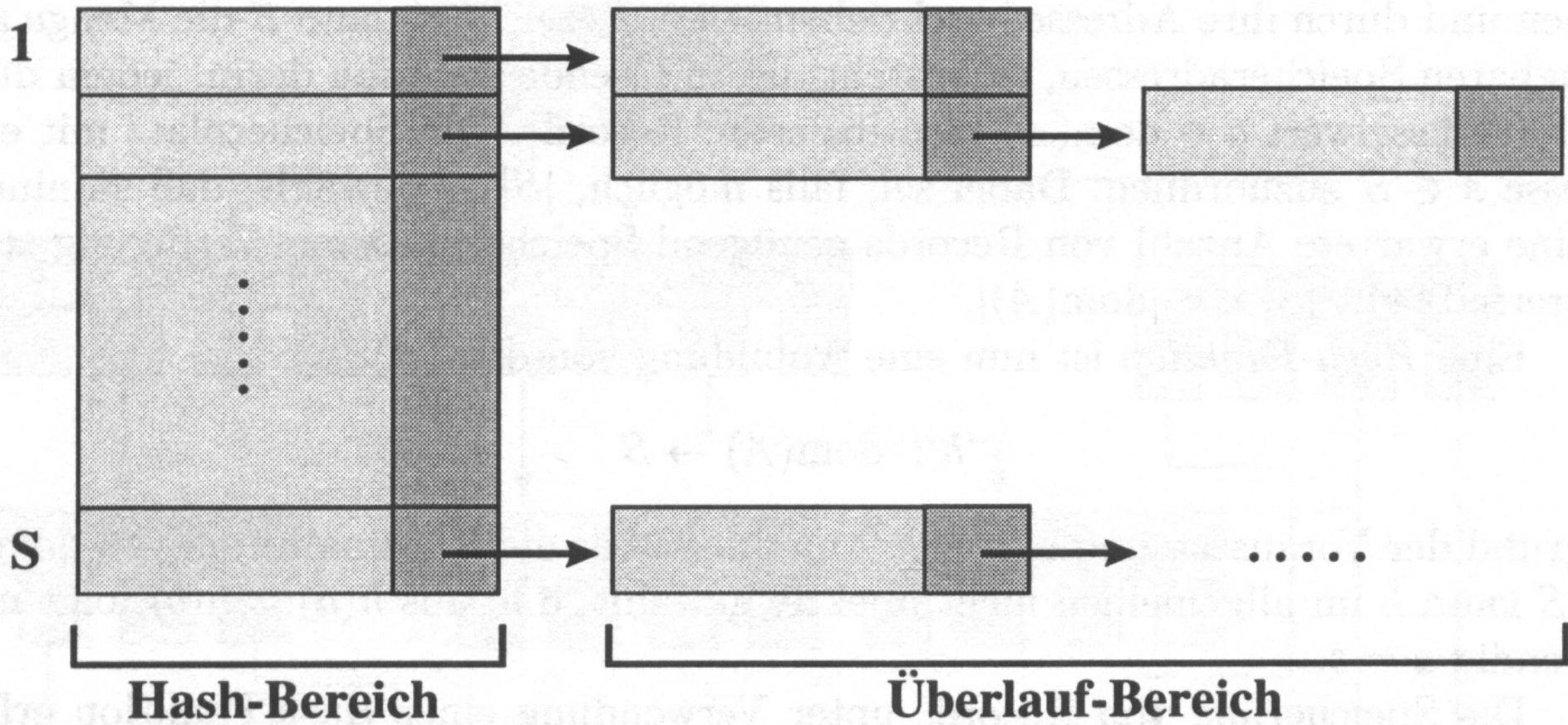
$$h(233) = 233$$

$$h(27685) = 685$$

$$h(43005) = 5$$

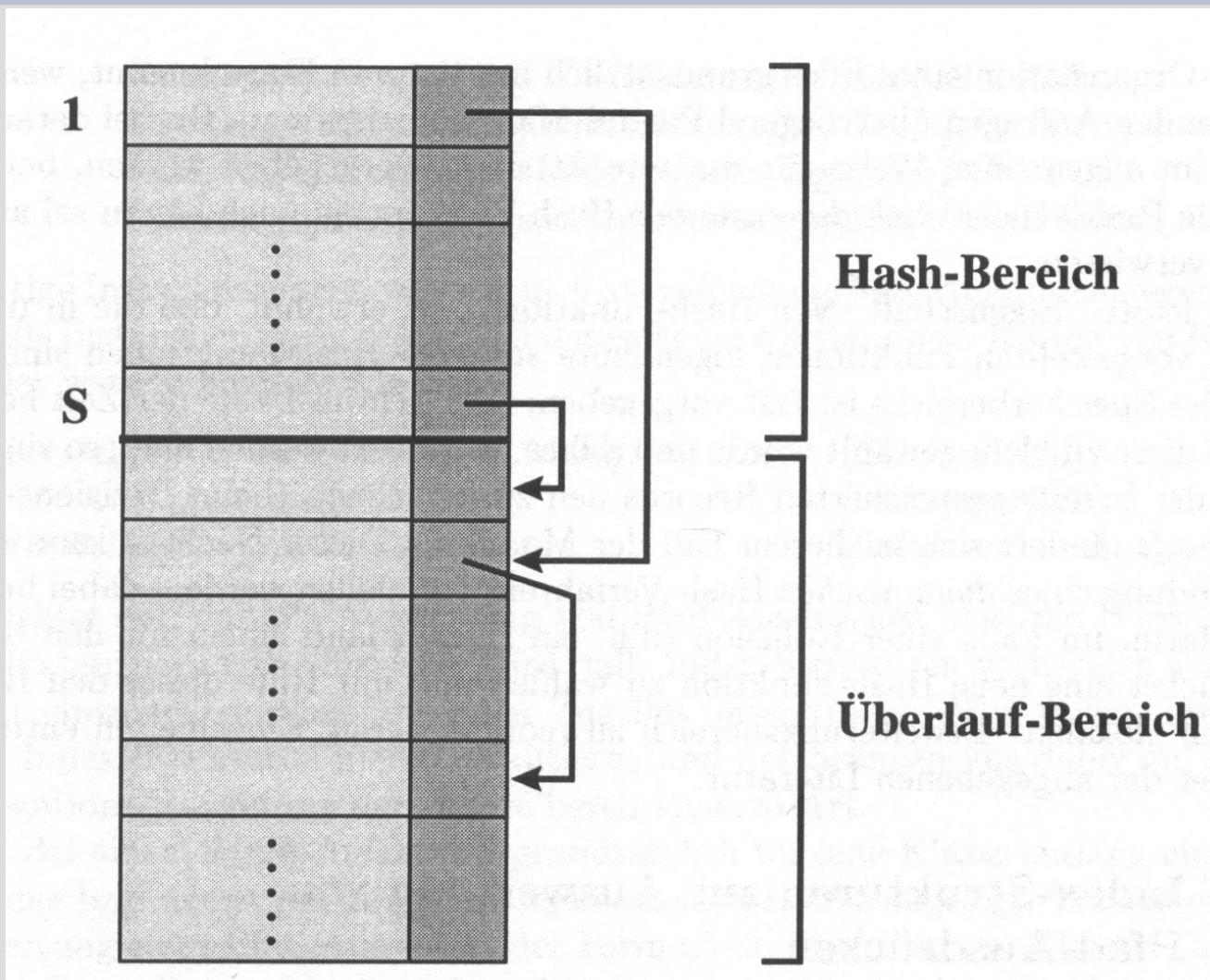
- Den Records mit den Schlüsselwerten $A=5$ und $A=43005$ wird der gleiche Speicherplatz zugewiesen.

Hashing mit Überlaufbereich pro Adresse



Vossen, G. (2000): Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme.
4. Aufl., Oldenbourg.

Hashing mit separatem Überlaufbereich



Vossen, G. (2000): Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme.

4. Aufl., Oldenbourg.

Offenes Hash-Verfahren

- im Falle einer Kollision wird mit zweiter Hash-Funktion Ersatzadresse ermittelt
 - ist auch diese schon belegt, iterativ weiter bis $|S|$

Primäradresse: $h_0(a)$ für $a \in \text{dom}(A)$

und eine Folge von $|S|-1$ Ersatzadressen:

$h_1(a), h_2(a), \dots, h_{n-1}(a)$ mit $n = |S|$

- Beispiel: Divisions-Rest-Verfahren mit Sondieren:

$$h_i(a) := \begin{cases} a \bmod n & \text{falls } i=0 \\ (h_0(a)+i) \bmod n & \text{sonst} \end{cases}$$

Index-Strukturen zur Auswertung von Pfadausdrücken

- treten in objektorientierten Datenbanken auf
- Pfad-Instanzierung:
 - Folge von Objekten, die im Rahmen einer Auswertung entlang des benutzten Pfades gefunden werden
 - Beispiel für in der Anfrage auszuwertenden Pfad:
Fahrzeug.Hersteller.Niederlassungen.Sitz.Ort
 - mögliche Pfad-Instanzierung:
Fahrzeug[i].Hersteller[j].Niederlassung[k].Sitz[l].Ort.Leipzig
- spezielle Indexstrukturen können diese Auswertung beschleunigen

Beispiel: Speicherorganisation bei DB2

- in benannten Teilen des Sekundärspeichers:
 - Tablespace
 - Indexspace
- DB2 Datenbank ist logisch zusammengehörende Menge von:
 - Tabellen und Tablespace
 - Indexen und Indexspace
- Datenbanken werden Storage Groups zugeordnet
- Physische Speicherobjekte sind
 - Bytes --> für Attributwerte oder systeminterne Größen
 - Records --> kleinstes adressierbares Speicherobjekt
 - Pages --> Einheit für Datentransfer zwischen Haupt- und Sekundärspeicher (4 kB oder 32 kB)

11. SQLite

- **Was ist SQLite?**
- **Verwendung**
- **Informationen:**
<http://www.sqlite.org>

Was ist SQLite?

- Programmbibliothek, die ein relationales Datenbanksystem enthält
- Wenige hundert kByte groß
- Implementiert einen großen Teil des SQL-92 Standards
 - Transaktionen, Unterabfragen, Sichten, Trigger, benutzerdefinierte Funktionen
- Für eingebettete Systeme entwickelt
- Was bietet es nicht?
 - Client-Server Architektur
 - Parallele Zugriffe

Tools

- Schnittstellen für viele Programmiersprachen, u.a.
 - ODBC- und JDBC-Treiber
- Frontend für Konsole/Scripte:
 - sqlite3
- Graphisches Frontend:
 - sqlitebrowser

Datenbank anlegen

- Datenbank anlegen:

```
sqlite3 test.db
```

- Tabelle erstellen:

```
sqlite> CREATE TABLE test (
...>     one VARCHAR(10),
...>     two SMALLINT);
```

- Daten einfügen:

```
sqlite> INSERT INTO test VALUES ('Hello!', 10);
sqlite> INSERT INTO test VALUES ('Goodbye!', 20);
```

- Daten abrufen:

```
SELECT * FROM test;
```

Einbinden in C-Programme

```
#include <stdio.h>
#include <sqlite3.h>
static int callback(void *NotUsed, int argc, char **argv, char **azColName)
{
    int i;
    for(i=0; i<argc; i++)
    {
        printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
    }
    printf("\n");
    return 0;
}
int main(int argc, char **argv)
{
    sqlite3 *db;
    char *zErrMsg = 0;
    int rc;
    if( argc!=3 )
    {
        fprintf(stderr, "Usage: %s DATABASE SQL-STATEMENT\n", argv[0]);
        return(1);
    }
    rc = sqlite3_open(argv[1], &db);
    if( rc )
    {
        fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return(1);
    }
    rc = sqlite3_exec(db, argv[2], callback, 0, &zErrMsg);
    if( rc!=SQLITE_OK )
    {
        fprintf(stderr, "SQL error: %s\n", zErrMsg);
        sqlite3_free(zErrMsg);
    }
    sqlite3_close(db);
    return 0;
}
```

Programm übersetzen

- Mit dem gcc-Compiler auf der Kommandozeile:
`gcc -Wall c_interface.c -lsqlite3 -o c_interface`
- Voraussetzungen:
 - gcc und Entwicklungstools
 - SQLite und Header-Dateien
(Linux: Package SQLite development)

Sprachen für Objektrelationale Datenbanken – Beispiel PostgreSQL

- Ingres
 - Relationales Datenbanksystem
 - University of California at Berkeley
- Postgres
 - Kurzform von Post Ingres
 - Erweiterbares System
 - Unterstützung komplexer Objekte
 - Benutzer-definierbare Datentypen, Operatoren und Zugriffsmethoden
 - Bereitstellung von Interferenzmechanismen
- Neue Bezeichnung: PostgreSQL

Literatur

- Peter Eisentraut: PostgreSQL. Das offizielle Handbuch. 1. Auflage, mitp-Verlag 2003. ISBN: 3-8266-1337-6.
- Stefan Kunick: Der Start mit PostgreSQL. URL: <http://www.postgresql.de>
- Ressourcen im WWW:
 - URL: <http://www.postgresql.de>
 - URL: <http://www.postgresql.org>

Hinweise: Die Bücher sind als PDF auf fileserv.babeipzig.de im Info-Verzeichnis verfügbar!

Start mit PostgreSQL

Beispiel: SuSE Linux (bis Version 9.3)

- PostgreSQL mittels yast2 installieren
- Als User postgres (su – postgres):
 - Mit psql -d template1 an Datenbank anmelden
(Datenbank template1 ist standardmäßig angelegt)
 - User und Datenbank anlegen:

```
CREATE USER userxx PASSWORD 'Geheim';
CREATE DATABASE userxx OWNER userxx TEMPLATE
template0;
GRANT ALL PRIVILEGES ON DATABASE userxx TO
userxx;
```
 - Berechtigungen und Zugang zur Datenbank
 - /var/lib/pgsql/data/pg_hba.conf
 - /var/lib/pgsql/data/postgresql.conf

Sprachen installieren

- Standardmäßig sind keine Sprachen in den Datenbanken aktiviert
- Installation einer im Template enthaltenen Sprache in eine Datenbank:
 - Als User postgres (nicht an Datenbank angemeldet)
 - `createlang plpgsql datenbank`
- Ersatzweise kann auch der SQL-Befehl CREATE LANGUAGE eingesetzt werden

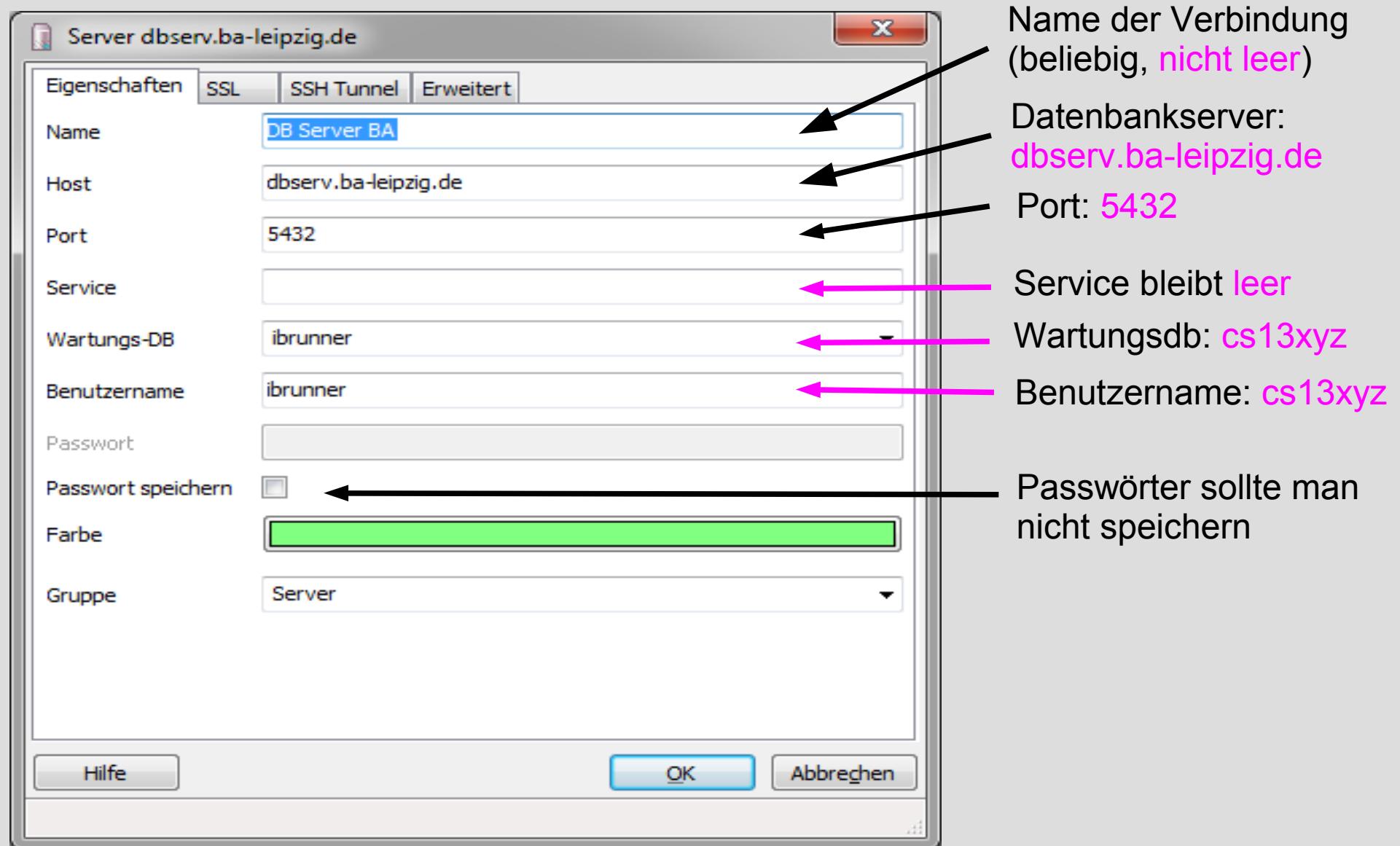
Tools für den Zugriff

- Vgl. die unter Literatur genannten Websites

Beispiele:

- pgAdmin (wird hier genutzt)
- DBTool
 - In Java realisiert
 - Freeware
 - Installation nicht erforderlich
- Datastudio
 - Private / akademische Nutzung kostenlos
 - In Java realisiert
 - Installation nicht erforderlich

Anmeldung am Datenbankserver mit pgAdmin III portable



Definition eines Tabellenschemas

```
CREATE TABLE table-name (
    column-name-1 type-1,
    column-name-2 type-2, ...)
[INHERITS (list of column-names)];
```

Beispiel

```
CREATE TABLE person (
    name CHAR(25),
    city CHAR(20),
    birthdate DATE PRIMARY KEY)
                        WITH OIDS;
CREATE TABLE professor (
    dept CHAR(20),
    field CHAR(50),
    tenure BOOL )
                        INHERITS ( person ) WITH OIDS;
CREATE TABLE student (
    studid CHAR(12),
    college CHAR(10),
    major CHAR(10),
    level CHAR(20))
                        INHERITS ( person ) WITH OIDS;
```

Views

- Sichten (Views) sparen umfangreiche Eingaben

Beispiel:

- View definieren:

```
CREATE VIEW studentview AS  
SELECT name, birthdate, college, major, level  
FROM student;
```

- View anwenden:

```
SELECT * FROM studentview WHERE major='IT';
```

- View löschen:

```
DROP VIEW studentview;
```

Mehrwertige Attribute

- **Definition Array ohne Dimensionen:**

```
CREATE TABLE Student (
    name          VARCHAR(30),
    pay_by_quarter INTEGER[],
    schedule      TEXT[][]
);
```

- **Definition mit Angabe der Dimension:**

```
CREATE TABLE tictactoe (
    squares      integer[3][3]
);
```

- **Alternativ für eindimensionale Arrays konform SQL:1999:**

```
pay_by_quarter integer ARRAY[4],
```

Mehrwertige Attribute

- Beispiel für Update:

```
UPDATE Student SET  
geschwister='{"Max","Erika"}'  
WHERE vorname='Susi';
```

- Einzelnen Wert abfragen:

```
SELECT geschwister[1] FROM student WHERE  
name='Susi';
```

Funktionen

- PostgreSQL kennt eine Reihe Standardfunktionen
- Eigene Funktionen lassen sich bei Bedarf schreiben:
 - Lassen sich in SQL-Befehle einbauen
 - Werden auf dem Server ausgeführt
 - Arten von Funktionen:
 - Mit SQL geschriebene Funktionen
 - Mit prozeduralen Sprachen geschriebene Funktionen (z.B. PL/pgSQL, PL/TCL, PL/Perl, PL/Python)
 - Interne Funktionen
 - C-Funktionen

Im Rahmen der Lehrveranstaltung werden nur SQL-Funktionen behandelt.

Standardfunktionen

- Mathematische Funktionen
- Zeichenkettenfunktionen
- Datums- und Uhrzeitfunktionen
- DateTime Arithmetik

Mathematische Funktionen

- ABS(zahl)
- PI()
- ROUND(Zahl,n)
- RANDOM()
- SQRT(Zahl)
- CEIL(Zahl)
- FLOOR(Zahl)
- MOD(Zahl1, Zahl2)

Zeichenkettenfunktionen

- ASCII(Zeichen)
- CHAR(Zahl)
- LOWER(Text)
- UPPER(Text)
- POSITION(t in T)
- LENGTH(Text) oder CHAR_LENGTH(Text)
- STRING || STRING
- SUBSTRING(t, s, n) oder SUBSTRING(t FROM s FOR n)

Zeichenkettenfunktionen

Text trimmen

- **TRIM([leading | trailing | both][Zeichen] FROM String)**
- **BTRIM(Text,z)**
- **LTRIM(Text)**
- **RTRIM(Text)**

Zeichenkettenfunktionen

Text umwandeln

- INITCAP(Text)
- TO_HEX(Zahl)
- REPEAT(Text, n)
- TRANSLATE(Text, von, nach)

Beispiel: Text verschleiern

- Die Zahl 3450 soll nicht sofort als solche erkannt werden:

```
SELECT  
TRANSLATE('3450', '0123456789', 'ABCDEFGHIJ')
```

- Klartext wieder sichtbar machen:

```
SELECT  
TRANSLATE('DEFA', 'ABCDEFGHIJ', '0123456789')
```

- **Achtung:** keine sichere Verschlüsselung (Cäsar-Code)

Datums- und Uhrzeitfunktionen

- CURRENT_DATE
- CURRENT_TIME (mit Zeitzone)
- CURRENT_TIMESTAMP (mit Zeitzone)
- LOCALTIME (ohne Zeitzone)
- LOCALTIMESTAMP (ohne Zeitzone)
- NOW() (entspricht CURRENT_TIMESTAMP)
- TIMEOFDAY()
(Rückgabewert Text anstelle TIMESTAMP)

Berechnungen mit Datums- und Zeitangaben

- AGE(TIMESTAMP, TIMESTAMP)
- EXTRACT(FIELD FROM TIMESTAMP)
- SELECT DATE '2013-11-26' + Interval '22 Days'
- SELECT DATE '2013-11-26' - Interval '17 Days'
- SELECT
 DATE '2013-11-26' + Interval '11 Weeks' AS "+ 11 Wochen",
 DATE '2013-11-26' + Interval '4 Months' AS "+ 4 Monate",
 DATE '2013-11-26' + Interval '2 Year' AS "+ 2 Jahre"
- SELECT TIMESTAMP '2013-11-26 12:52:11' + Interval '8 Hours'

SQL-Funktionen

```
CREATE OR REPLACE FUNCTION
    name(parameter)
RETURNS art AS $$

    [SQL-Befehl;]
    . . .
$$ LANGUAGE SQL;
```

SQL-Funktionen (2)

```
CREATE OR REPLACE FUNCTION
    geschwister(TEXT, INT)
RETURNS VARCHAR(30) AS
$$
    SELECT geschwister[$2] FROM student
WHERE name=$1
$$ LANGUAGE SQL;
```

PL/pgSQL

- Procedural Language/PostgreSQL Structured Query Language
- an ADA angelehnt
 - strukturierte Programmiersprache
 - statische Typenbindung
- Funktionen müssen deklariert werden
 - Funktionskopf
 - Funktionsrumpf

Beispiel einfache Funktion

```
CREATE OR REPLACE FUNCTION
Arith_Mittel(Integer, Integer) RETURNS NUMERIC
AS $$

    SELECT ($1 + $2)/2.0;
$$ LANGUAGE SQL;
```

```
SELECT Arith_Mittel(2,4);
```

```
DROP FUNCTION Arith_Mittel(Integer, Integer);
```

Funktion mit Variablen-deklaration

```
CREATE or REPLACE FUNCTION
Arith_Mittel(Integer, Integer)
RETURNS NUMERIC AS $$

DECLARE
Zahl1 Alias FOR $1;
Zahl2 Alias FOR $2;
Ergebnis Numeric;
BEGIN
Ergebnis := (Zahl1 + Zahl2)/2.0;
RETURN Ergebnis;
END;
$$ LANGUAGE plpgsql;
```

Tabelle als Parameter

```
CREATE or REPLACE FUNCTION  
  Preis_mit_MwSt(Artikel)  
RETURNS NUMERIC AS $$  
  SELECT ROUND($1.Nettopreis*1.19,2);  
$$ LANGUAGE SQL;
```

```
SELECT ArtikelName, Nettopreis,  
  Preis_ohne_Mwst(Artikel)  
FROM Artikel WHERE ArtikelNr = 324;
```

Zeile/Tabelle als Rückgabewert

- Rückgabewert Zeile:
RETURNS TableName
- Rückgabewert Tabelle:
RETURNS SETOF TableName

Beispiel: Zeile zurückliefern

```
CREATE OR REPLACE FUNCTION
  Lesen_Artikel(Integer)
RETURNS Bestellung AS $$ 
  SELECT * FROM Artikel WHERE ArtikelNr = $1;
$$ LANGUAGE SQL;
```

Beispiel: Tabelle zurückliefern

```
CREATE OR REPLACE FUNCTION
  Lesen_Artikel(Integer)
RETURNS SETOF Artikel AS $$ 
  SELECT * FROM Artikel WHERE LieferantenNr = $1;
$$ LANGUAGE SQL;
```

```
SELECT * FROM Lesen_Artikel(108)
```

Trigger

- Lassen sich vor oder nach Ereignis aktivieren
- Trigger ruft eine Funktion auf
 - Rückgabewert der Funktion:
 - NULL-Wert oder
 - Record- bzw. Zeilenwerttyp
 - Before-Trigger:
 - NULL-Wert: keine weiteren Aktionen
 - Anderer Wert: Aktionen laufen weiter
 - Kommt bei einer NEW Aktion ein Zeilenwert zurück, werden die ggf. geänderten Daten übernommen

Trigger

- Lassen sich vor oder nach Ereignis aktivieren
- Trigger ruft eine Funktion auf
 - Rückgabewert der Funktion:
 - NULL-Wert oder
 - Record- bzw. Zeilenwerttyp
 - Before-Trigger:
 - NULL-Wert: keine weiteren Aktionen
 - Anderer Wert: Aktionen laufen weiter
 - Kommt bei einer NEW Aktion ein Zeilenwert zurück, werden die ggf. geänderten Daten übernommen

Realisierung eines Triggers (1)

Beispieltabelle:

```
CREATE TABLE versuch (
    vnr VARCHAR(5) PRIMARY KEY,
    vname VARCHAR(30),
    datum TIMESTAMP,
    benutzer VARCHAR(30));
```

Realisierung eines Triggers (2)

Funktion anlegen:

```
CREATE OR REPLACE FUNCTION
    Versuch_Datum_geaendert()
RETURNS trigger AS $$

BEGIN
    IF NEW.vnr IS NULL THEN
        RAISE EXCEPTION 'Feld vnr nicht gefüllt';
    END IF;
    IF NEW.vname IS NULL THEN
        RAISE EXCEPTION 'Feld vname nicht gefüllt';
    END IF;
    NEW.datum := NOW();
    NEW.benutzer := CURRENT_USER;
    RETURN NEW;
END
$$ LANGUAGE plpgsql;
```

Realisierung eines Triggers (3)

Trigger anlegen:

```
CREATE TRIGGER Versuch_Datum_geaendert BEFORE  
    INSERT OR UPDATE ON versuch  
FOR EACH ROW EXECUTE PROCEDURE  
Versuch_Datum_geaendert();
```

Beispiel Trigger

```
CREATE OR REPLACE FUNCTION check_kuerzel() RETURNS TRIGGER AS $$  
DECLARE  
    ikuerzel CHAR(3);  
BEGIN  
    IF NEW.kuerzel IS NULL THEN  
        RAISE EXCEPTION 'Empty kuerzel.';  
    END IF;  
    SELECT INTO ikuerzel datenpool.kuerzel FROM (  
        SELECT kuerzel FROM lektor  
        UNION  
        SELECT kuerzel from verfasser  
        UNION  
        SELECT kuerzel FROM herausgeber) datenpool  
    WHERE datenpool.kuerzel = NEW.kuerzel;  
    -- If there is an matching kuerzel, raise an exception  
    IF FOUND THEN  
        RAISE EXCEPTION 'Invalid kuerzel.';  
    END IF;  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;
```

Constraints

- Bei der Neuanlage einer Tabelle:

```
CREATE TABLE artikel (artikel_nr varchar(15),  
artikel_name varchar(25),  
artikel_preis numeric CHECK (artikel_preis>0));
```

- Mit Definition einer Fehlermeldung:

```
CREATE TABLE artikel (artikel_nr varchar(15),  
artikel_name varchar(25),  
artikel_preis numeric  
CONSTRAINT "Korrekte Preis angeben!"  
CHECK (artikel_preis>0));
```

- Prüfung auf Wert Null:

```
artikel_name varchar(25) NOT NULL
```

- Doppelte Vergabe des gleichen Wertes verhindern:

```
artikel_name varchar(25) UNIQUE
```

Constraints (2)

- **Constraint nachträglich einfügen:**

```
ALTER TABLE artikel ADD  
    CHECK (artikel_preis > '0');
```

- **Oder:**

```
ALTER TABLE artikel ADD  
    CONSTRAINT "Doppelte Artikel-Nummer"  
    UNIQUE (artikel_nr);
```

- **Für das Null-Constraint:**

```
ALTER TABLE artikel ALTER  
    COLUMN artikel_nr SET NOT NULL;
```

- **Constraint entfernen (außer Null-Constraint):**

```
ALTER TABLE artikel DROP CONSTRAINT "Doppelte  
Artikel-Nummer";
```

- **Null-Constraint entfernen:**

```
ALTER TABLE artikel ALTER  
    COLUMN artikel_nr DROP NOT NULL;
```

Transaktionen

- Zusammenfassung einer Reihe von Befehle
- Sind **atomar**: werden vollständig ausgeführt oder gar nicht
- Aus der Sicht anderer Transaktionen passiert die komplette Transaktion oder gar nichts
- Eine Transaktion sieht die unvollständigen Änderungen der anderen nicht
- Vom DBMS als abgeschlossen gemeldete Transaktion wurde garantiert auf dauerhaftes Medium geschrieben

Beispiel: Kontoverbuchung

```
UPDATE konten SET
    kontostand = kontostand - 100.00
    WHERE name = 'Anne';
UPDATE zweigstellen SET
    kontostand = kontostand - 100.00
    WHERE name = (SELECT zweig_name FROM
konten WHERE name = 'Anne');
UPDATE konten SET
    kontostand = kontostand + 100.00
    WHERE name = 'Bernd';
UPDATE zweigstellen SET
    kontostand = kontostand + 100.00
    WHERE name = (SELECT zweig_name FROM
konten WHERE name = 'Bernd');
```

Transaktionen (2)

- SQL-Befehle der Transaktion werden eingebettet:
BEGIN;
...
[SQL-Befehl;]
...
COMMIT;
- Transaktion abbrechen (z.B. wegen unzureichender Deckung des Kontos):
ROLLBACK statt COMMIT

Schema Addierer in Postgres

Achtung: Der Datentyp POSTQUEL wird aktuell nicht mehr unterstützt!

```
CREATE CHIP ( CHIPID = INT4 ,
    CHIPNAME = CHAR[ 30 ] ;
    MODULE = POSTQUEL )
KEY ( CHIPID )
CREATE MODUL ( MODID = INT4 ,
    MODNAME = CHAR[ 20 ] ;
    SUBMODULE = POSTQUEL )
KEY ( MODID )
CREATE SUBMODUL ( SUBID = INT4 ,
    SUBNAME = CHAR[ 4 ] ;
KEY ( SUBID )
```