

# 3. Von relationalen zu objekt-relationalen DBS

- Beschränkungen des relationalen Datenmodells
  - Beispiel: Modellierung von 3D-Objekten
  - Impedance Mismatch
  - Anwendungsgebiete mit besonderen Anforderungen
- Klassifikation von Datenmodellen
- NF2-Ansatz
- OODBS: Eigenschaften
  - Objekttypen, Kapselung
  - Generalisierung, spätes Binden
  - Komplexe Objekte: Objektidentität, Typkonstruktoren
- OODBS vs. ORDBS
- O/R-Mapping; Hibernate



## Objekt-Darstellung

- Standard-Anwendung: pro Objekt gibt es genau eine Satzausprägung, die alle beschreibenden Attribute enthält

### Ausprägungen

#### Schema

**ANGESTELLTER**

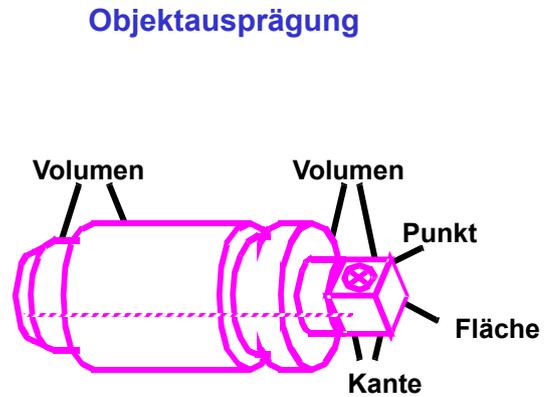
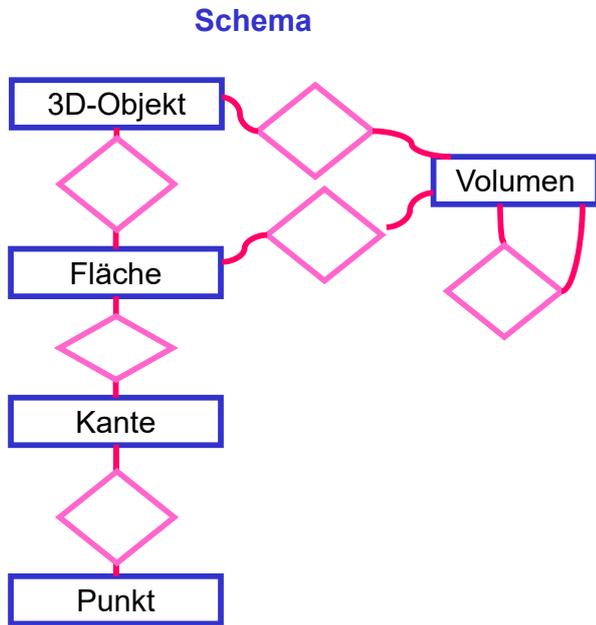
Satztyp (Relation)

PNR	NAME	TAETIGKEIT	GEHALT	ALTER
496	Peinl	Pfoertner	2100	63
497	Kinzinger	Kopist	2800	25
498	Meyweg	Kalligraph	4500	56

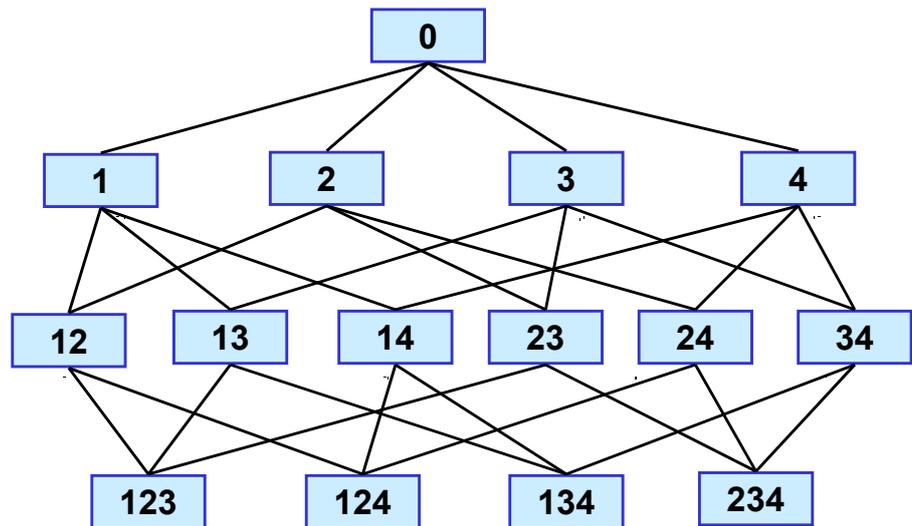
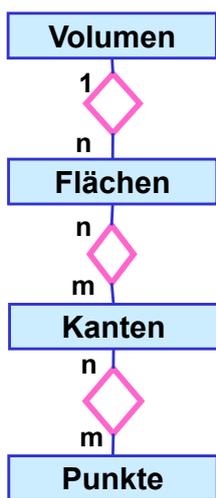


# Objekt-Darstellung (2)

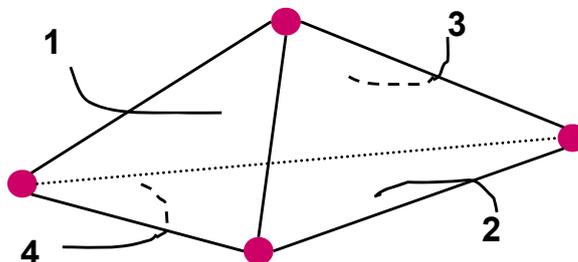
- CAD-Anwendung: das komplexe Objekt „Werkstück“ setzt sich aus einfacheren (komplexen) Objekten verschiedenen Typs zusammen



## Modellierung von 3D-Objekten im ER-Modell

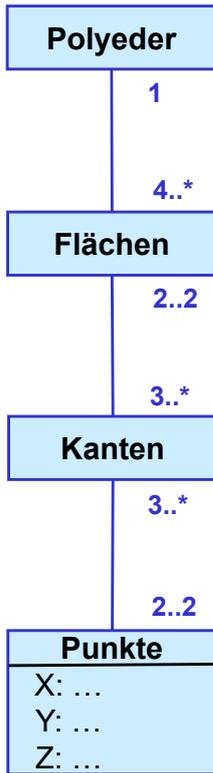


Beispiel: Tetraeder



# Modellierung von Polyedern im RM

UML



Modellierung im Relationenmodell

```

CREATE TABLE Polyeder
(polyid: INTEGER,
anzflächen: INTEGER,
PRIMARY KEY (polyid));
    
```

```

CREATE TABLE Fläche
(fid: INTEGER,
anzkanten: INTEGER,
pref: INTEGER,
PRIMARY KEY (fid),
FOREIGN KEY (pref)
REFERENCES Polyeder);
    
```

```

CREATE TABLE Kante
(kid: INTEGER,
ktyp: CHAR(5),
PRIMARY KEY (kid));
    
```

```

CREATE TABLE Punkt
(pid: INTEGER,
x, y, z: FLOAT,
PRIMARY KEY (pid));
    
```

```

CREATE TABLE FK_Rel
(fid: INTEGER,
kid: INTEGER,
PRIMARY KEY (fid, kid),
FOREIGN KEY (fid)
REFERENCES Fläche,
FOREIGN KEY (kid)
REFERENCES Kante);
    
```

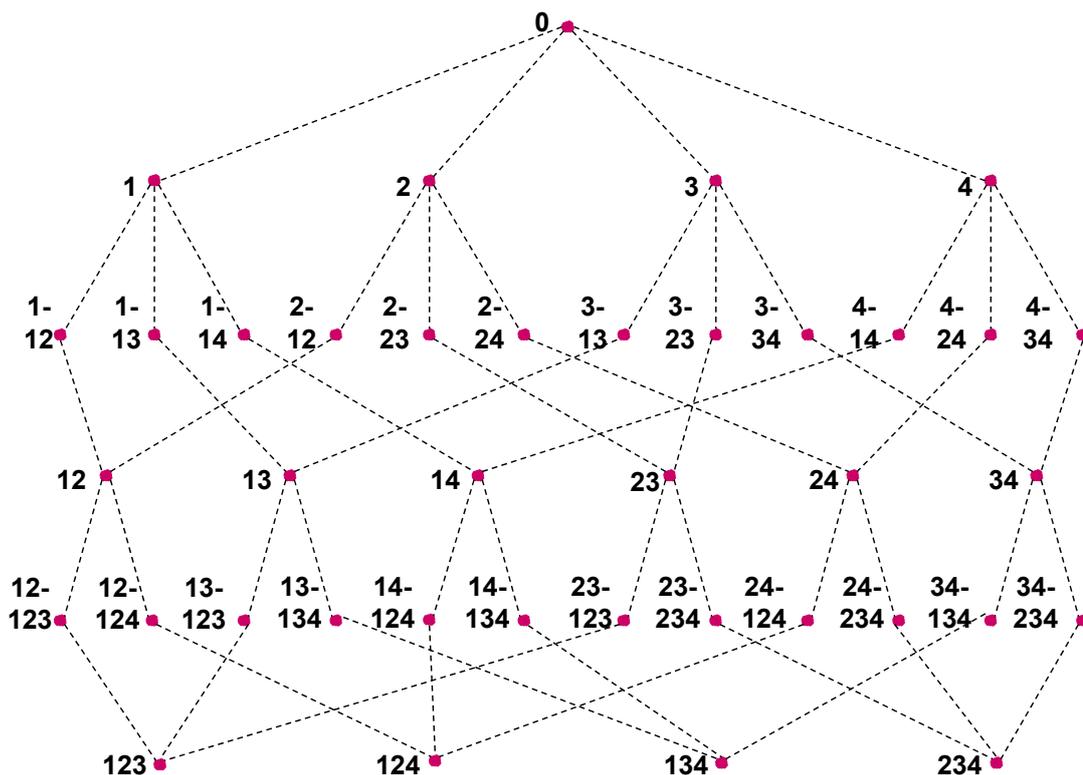
```

CREATE TABLE KP_Rel
(kid: INTEGER,
pid: INTEGER,
PRIMARY KEY (kid, pid),
FOREIGN KEY (kid)
REFERENCES Kante,
FOREIGN KEY (pid)
REFERENCES Punkt);
    
```



## Relationenmodell – angemessene Modellierung?

Darstellung eines Tetraeder mit vid = 0



Relationen

Polyeder

Fläche

FK-Rel

Kante

KP-Rel

Punkt



# Beispielanfragen

- Bestimme alle Punkte, die zu Flächenobjekten mit  $F.fid < 3$  gehören

```
SELECT F.fid, P.x, P.y, P.z
FROM Punkt P, KP-Rel S, Kante K, FK-Rel T, Fläche F
WHERE F.fid < 3
      AND P.pid = S.pid /* Rekonstruktion*/
      AND S.kid = K.kid /* komplexer Objekte*/
      AND K.kid = T.kid /* zur Laufzeit*/
      AND T.fid = F.fid;
```

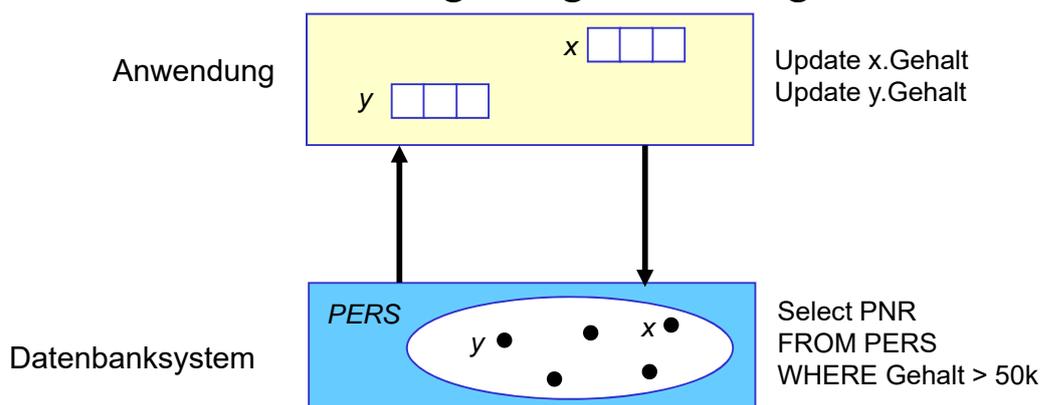
- symmetrischer Zugriff: Flächen, die mit Punkt (50,44,75) assoziiert sind

```
SELECT F.fid
FROM Punkt P, KP-Rel S, Kante K, FK-Rel T, Fläche F
WHERE P.x = 50 AND P.y = 44 AND P.z = 75
      AND P.pid = S.pid
      AND S.kid = K.kid
      AND K.kid = T.kid
      AND T.fid = F.fid;
```



## Impedance Mismatch

- Auseinanderklaffen von Datenbanksystem und Programmiersprachen:  
“*Impedance mismatch*” (Fehlanpassung)
  - “Struktur” wird durch DBS, “Verhalten” weitgehend von Anwendungsprogrammen (Programmiersprache) abgedeckt
  - unterschiedliche Datentypen und Operationen
  - Mengen- vs. Satzverarbeitung
  - unterschiedliche Behandlung transienter und persistenter Objekte
  - umständliche, fehleranfällige Programmierung



# Besondere DB-Anforderungen

- Multimedia-Daten (Bilder, Grafik, Ton, Audio, Video, Text ...)
  - großer Datenumfang und aufwändige Operationen
  - Dateispeicherung führt zur Ungleichbehandlung mit Datenbankinhalten (fehlende Anfragemöglichkeiten, Transaktionsschutz ...)
- Information-Retrieval-Systeme (IRS)
  - Verwaltung von Dokumenten/Texten (ohne feste Struktur)
  - unscharfe Textsuche (Homonym/Synonym-Probleme etc.)
  - Relevanzproblem (Precision, Recall)
  - Ranking entscheidend für große Ergebnislisten
- Semi-strukturierte Daten / XML-Dokumente
  - optionaler Schemaeinsatz
  - Mischung von Text und strukturierten Daten
- Erweiterbarkeit auf unterschiedliche Arten von Daten und Berechnungen
  - Geoinformationssysteme: räumliche Daten und Operationen
  - Bioinformatik: komplex strukturierte Daten (Proteinstrukturen ...)



# Wissensbasierte Anwendungen

- Verwaltung von Fakten (formatierte Daten = extensionale DB) und Regeln (intensionale DB)
  - Regeln dienen zur Ableitung von implizit vorhandenen Informationen
  - Nutzung nicht nur in KI-Anwendungen: Stücklistenauflösung, Wegeprobleme (Berechnung der transitiven Hülle)
- Hauptanforderung: effiziente Regelauswertung (Inferenz), Behandlung von Rekursion

## **Fakten:**

F1: Elternteil( C, A) <-  
F2: Elternteil (D, A) <-  
F3: Elternteil (D, B) <-  
F4: Elternteil (G, B) <-  
F5: Elternteil (E, C) <-  
F6: Elternteil (F, D) <-  
F7: Elternteil (H, E) <-

## **Anfrage:**

? Vorfahr (x, A)

## **Regeln:**

R1: Vorfahr (x, y) <- Elternteil (x, y)  
R2: Vorfahr (x, y) <- Elternteil (x, z), Vorfahr (z, y)



# Beschränkungen des Relationenmodells

- nur einfach strukturierte Daten
  - einfache (Standard-) Datentypen
  - Sätze mit fester Anzahl atomarer Attribute (festes Satzformat)
- Relationenmodell ist "wertebasiert"
  - Identifikation von Daten durch Schlüsselwerte.  
häufig künstliche Schlüsselattribute zu definieren
  - Modellierung von Beziehungen über Fremdschlüssel.  
umständliche Modellierung komplexer Strukturen
- oft schlechte Effizienz für anspruchsvolle Anwendungen
  - viele Joins
- geringe Semantik
  - Benutzer muss Bedeutung der Daten/Namen kennen
  - nur einfache Integritätsbedingungen
  - keine direkte Unterstützung der Abstraktionskonzepte (Generalisierung, Aggregation)

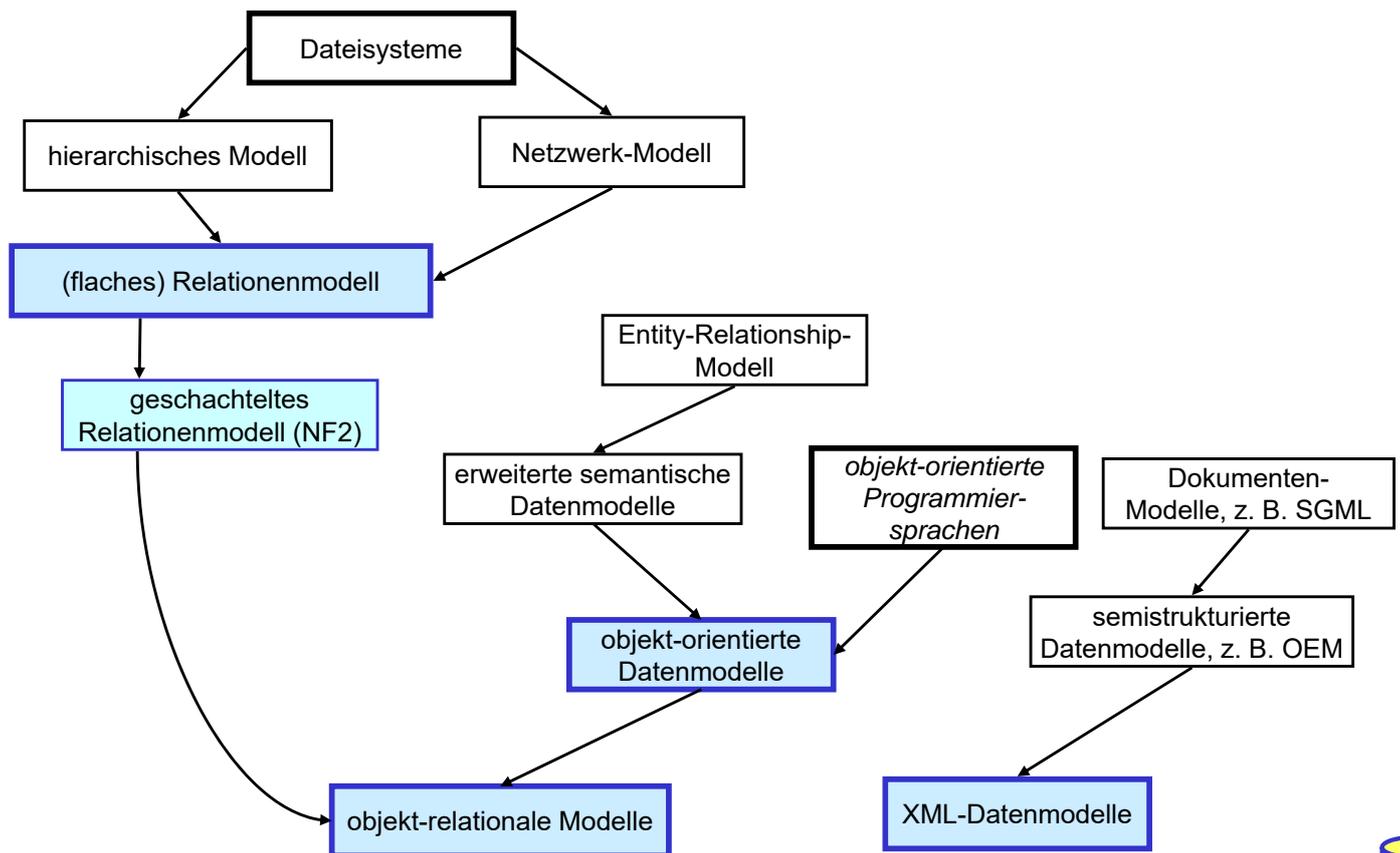


## Beschränkungen des Relationenmodells (2)

- unzureichende Spezifikation von "Verhalten" (Funktionen)
  - Verbesserung der Situation durch Stored Procedures
  - weitere Unterstützung durch benutzerdefinierte Methoden wünschenswert
- begrenzte Auswahlmächtigkeit der Anfragesprachen
  - keine Unterstützung von Rekursion (Berechnung der transitiven Hülle)
  - trotz SQL-Erweiterungen weiterhin Notwendigkeit allgemeine Programmiersprachen zu nutzen
- umständliche Einbettung in Programmiersprachen (impedance mismatch)
- auf kurze Transaktionen zugeschnitten (ACID)
  - Alles-oder-Nichts ungünstig für längere Verarbeitungsvorgänge



# Entwicklung von Datenmodellen



## NF2-Modell

### ■ Non-First Normal Form

- 'nested relations', unnormalisierte Relationen
- wie Relationenmodell, jedoch können Attributwerte auch Relationen (Mengen von Tupeln) sein

### ■ Polyeder-Beispiel

```

CREATE TABLE Volumen (
  VId          INT,
  Bez         CHAR(20),
  AnzFlaechen INT,
  Flaechen    SET ( ROW ( FId          INT,
                          AnzKanten  INT,
                          Kanten    SET ( ROW ( Kid          INT,
                                              Punkte SET ( ROW ( PId INT,
                                                                X INT,
                                                                Y INT,
                                                                Z INT ) )
                                              )
                          )
  )
)
  
```

**SET** Mengenkonstruktor, **ROW** Tupelkonstruktor



# NF<sup>2</sup>-Ausprägung (1 Tupel)

Volumen							
Vld	Bez	Flaechen		Punkte			
		Fld	Kanten	Pld	X	Y	Z
			Kld				
0	Tetraeder	1	12	123	0	0	0
				124	100	0	0
			13	123	0	0	0
				134	50	44	75
			14	124	100	0	0
				134	50	44	75
		2	12	123	0	0	0
				124	100	0	0
			23	123	0	0	0
				234	50	87	0
			24	124	100	0	0
				234	50	87	0
		3	13	123	0	0	0
				134	50	44	75
			23	123	0	0	0
				234	50	87	0
			34	134	50	44	75
				234	50	87	0
		4	14	124	100	0	0
				134	50	44	75
			24	124	100	0	0
				234	50	87	0
			34	134	50	44	75
				234	50	87	0



## NF<sup>2</sup>-Modell: Operatoren

### erweiterte relationale Algebra

- Erweiterung von Projektion, Selektion und Join auf geschachtelte Strukturen

### NEST-Operation

- Erzeugen geschachtelter Relationen aus flachen Relationen
- $NEST_{A_1, A_2, \dots, A_n: A}(R)$  fasst Attribute  $A_1, \dots, A_n$  zu neuem Attribut  $A$  zusammen, d.h.  $A$  entspricht SET (ROW ( $A_1, A_2, \dots, A_n$ ))
- mehrere ( $A_1, \dots, A_n$ )-Tupel werden dabei zu einer Menge zusammengefasst, wenn die Werte der Tupel auf den anderen R-Attributen  $A_{n+1}, \dots, A_m$  übereinstimmen (entspricht einem SQL Group-By auf  $A_{n+1}, \dots, A_m$ )

A	B	C
1	2	7
1	3	6
1	4	5
2	1	1




## NF2-Modell: Operatoren (2)

### ■ UNNEST-Operation

- Normalisierung (“Flachklopfen”) geschachtelter Relationen
- $UNNEST_{A:A_1, \dots, A_n}(R)$  überführt tabellenwertiges Attribut  $A = SET(ROW(A_1, \dots, A_n))$  in Menge einfacher Attribute  $A_1, \dots, A_n$
- $UNNEST_{A: A_1 \dots A_n}(NEST_{A_1, A_2, \dots, A_n: A}(R)) = R$

### ■ NEST und UNNEST sind dennoch i. a. nicht invers zueinander !

A	B
1	2
	3
1	4
	5

A	B

A	B

## NF2-Modell: Operatoren (3)

### ■ erweiterter natürlicher Join

**R1**

A	B	X	
		C	D
a1	b1	c1	d1
		c2	d2
		c1	d3
a2	b2	c1	d2
		c3	d2

**R2**

E	B	X	
		C	D
e1	b1	c1	d1
		c1	d3
		c3	d4
e3	b2	c3	d2

### ■ zwei Join-Semantiken für tabellenwertige Join-Attribute

- vollständige Übereinstimmung der Tabellenwerte für Verbundtupel
- nur tupelweise Übereinstimmung

**R1** ⋈ **R2**


# Bewertung des NF2-Modells

## ■ Vorteile:

- einfaches Relationenmodell als Spezialfall enthalten
- Unterstützung komplex-strukturierter Objekte
- reduzierte Join-Häufigkeit
- Clusterung einfach möglich
- sicheres theoretisches Fundament (NF2-Algebra)

## ■ Nachteile:

- überlappende/gemeinsame Teilkomponenten (n:m-Beziehungen) führen zu Redundanz
- unsymmetrischer Zugriff
- rekursiv definierte Objekte nicht zulässig
- keine Unterstützung von Generalisierung und Vererbung
- keine benutzerdefinierten Datentypen und Operationen



# OODBS

## ■ Ansätze zur objekt-orientierten Datenverwaltung

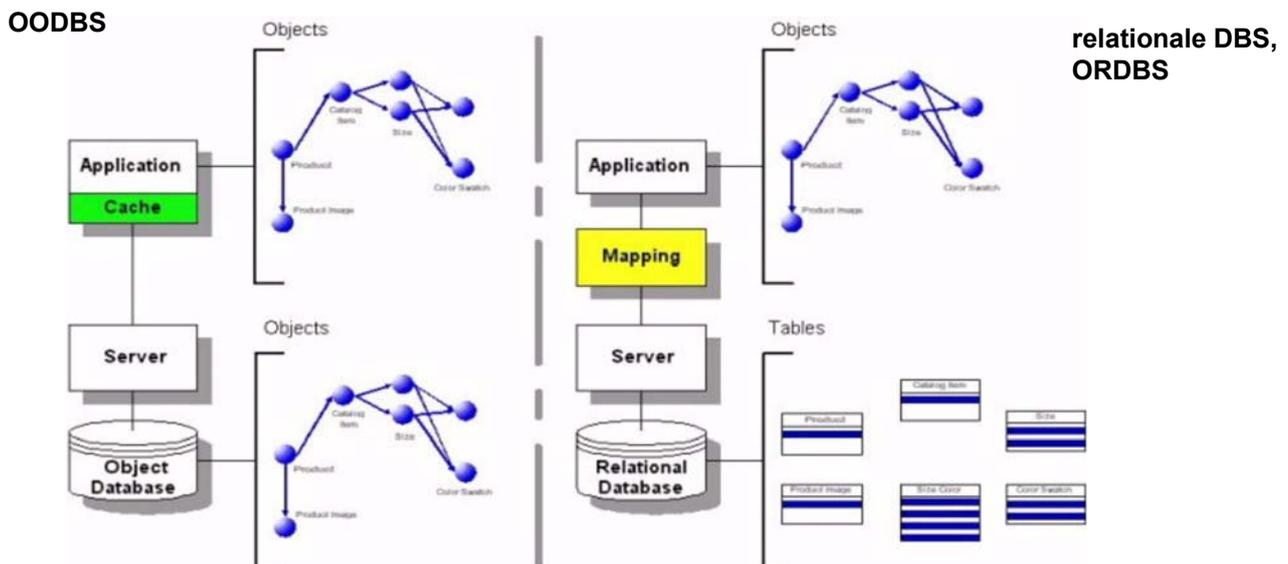
- Modellierung der Struktur (komplexe Objekte, Vererbung) + Verhalten (ADTs, Methoden)
- Anreicherung von OO-Programmiersprachen um DB-Eigenschaften (Persistenz, Integrität, ...) -> *persistente Programmiersprachen / objekt-orientierte DBS (OODBS)*
- Anreicherung von DBS um objekt-orientierte Konzepte -> *objekt-relationale DBS (ORDBS)*

## ■ OODBS

- Erweiterung objektorientierter Programmierung um DB-Sprache
- einheitliche Verwaltung transienter und persistenter Objekte
- effiziente Traversierung komplexer Objektstrukturen
- Beseitigung des “impedance mismatch”/ bessere Effizienz



# Effiziente Navigation mit OODBS



- sehr schnelle Navigation komplexer Objekte auch in Client/Server-Umgebungen ( $\mu\text{s}$  statt  $\text{ms}$ )
  - besonders wichtig für interaktive Designaufgaben, z.B. im CAD (Forderung  $10^5$  Objektreferenzen pro sec)
- transparente Abbildung zwischen physischem und virtuellem Speicher



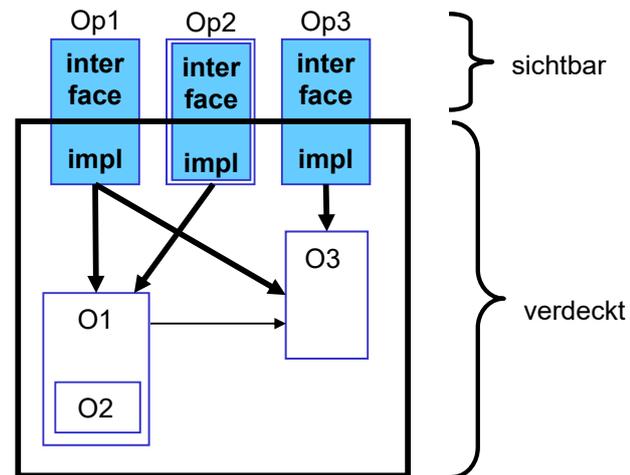
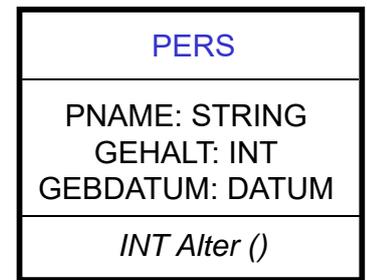
## Definition eines objekt-orientierten DBS

- OODBS muss DBS + objekt-orientiertes System sein
- DBS-Aspekte
  - Persistenz, Externspeicherverwaltung
  - Datenunabhängigkeit
  - Transaktionsverwaltung
  - Ad-Hoc-Anfragesprache
- OOS-Aspekte:
  - Objekttypen, Kapselung
  - Typ-/Klassenhierarchie, Vererbung, Überladen und spätes Binden
  - Objektidentität, komplexe Objekte
  - operationale Vollständigkeit
  - Erweiterbarkeit
  - Versionen
  - lang-lebige Transaktionen

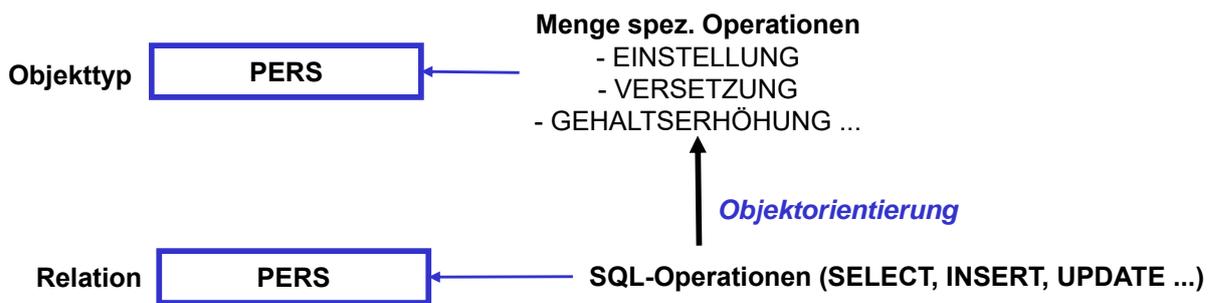


# Objekttypen / Kapselung

- Objekt = Struktur + Verhalten + OID
- Spezifikation durch **Objekttyp / Klasse**
  - Struktur: Attribute und ihre Wertebereiche
  - Verhalten: zulässige Operationen / Methoden
- Objekt = Instanziierung eines Typs mit konkreten Wertebelegungen der Attribute
- Strenge Objekt-Orientierung verlangt Kapselung (Information Hiding)
  - Verhalten des Objektes ist ausschließlich durch seine Operationen (Methoden) bestimmt
  - nur Namen und Signatur (Argumenttypen, Ergebnistyp) von Operationen werden bekannt gemacht
  - Struktur von Objekten wird verborgen
  - Implementierung der Operationen bleibt verborgen



## Kapselung (2)



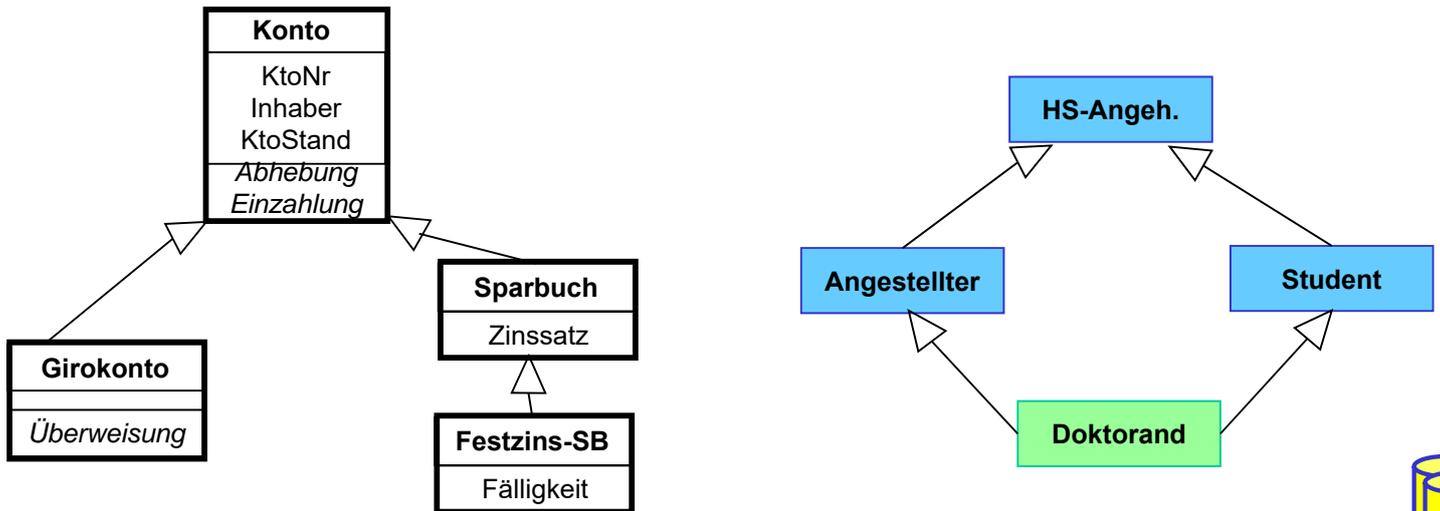
- Verwaltung von Objekttypen und Operationen im DBS
  - zusätzliche Anwendungsorientierung im DBS gegenüber Stored Procedures
  - verringerte Kommunikationshäufigkeit zwischen Anwendung und DBS
  - Reduzierung des "impedance mismatch"
- Vorteile der Kapselung: höherer Abstraktionsgrad
  - logische Datenunabhängigkeit, Datenschutz
- Aber: strikte Kapselung oft zu restriktiv
  - eingeschränkte Flexibilität
  - mangelnde Eignung für Ad-Hoc-Anfragen



# Generalisierung

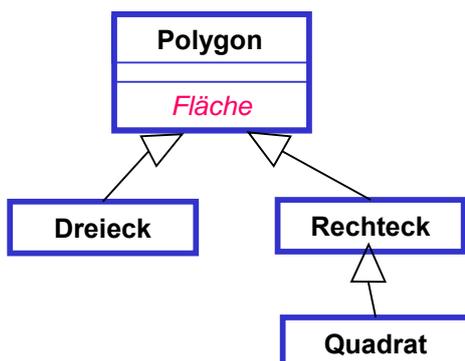
## ■ Generalisierungs-/Spezialisierungshierarchie (IS-A-Beziehung)

- Vererbung von Attributen, Methoden, Integritätsbedingungen ...
- Arten der Vererbung: einfach (Hierarchie) vs. mehrfach (Typverband)
- Prinzip der *Substituierbarkeit*: Instanz einer Subklasse B kann in jedem Kontext verwendet werden, in dem Instanzen der Superklasse A möglich sind (jedoch nicht umgekehrt)
  - impliziert, dass Klasse heterogene Objekte enthalten kann



# Überladen (Overloading)

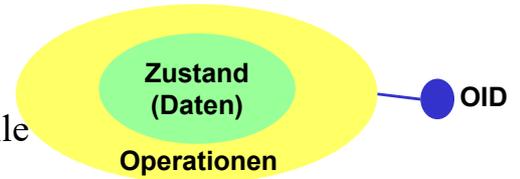
- derselbe Methodenname wird für unterschiedliche Prozeduren verwendet (polymorphe Methoden)
  - erleichtert Realisierung nutzender Programme und verbessert Software-Wiederverwendbarkeit
- Overloading innerhalb von Typ-Hierarchien:
  - Redefinition von Methoden für Subtypen (**Overriding**)
  - spezialisierte Methode mit gleichem Namen
- Überladen impliziert dynamisches (spätes) Binden zur Laufzeit (*late binding*)



# Objektidentität

## ■ OODBS: Objekt = (OID, Zustand, Operationen)

- OID: Identifikator
- Zustand: Beschreibung mit Attributen
- Operationen (Methoden): definieren externe Schnittstelle



## ■ Objektidentität

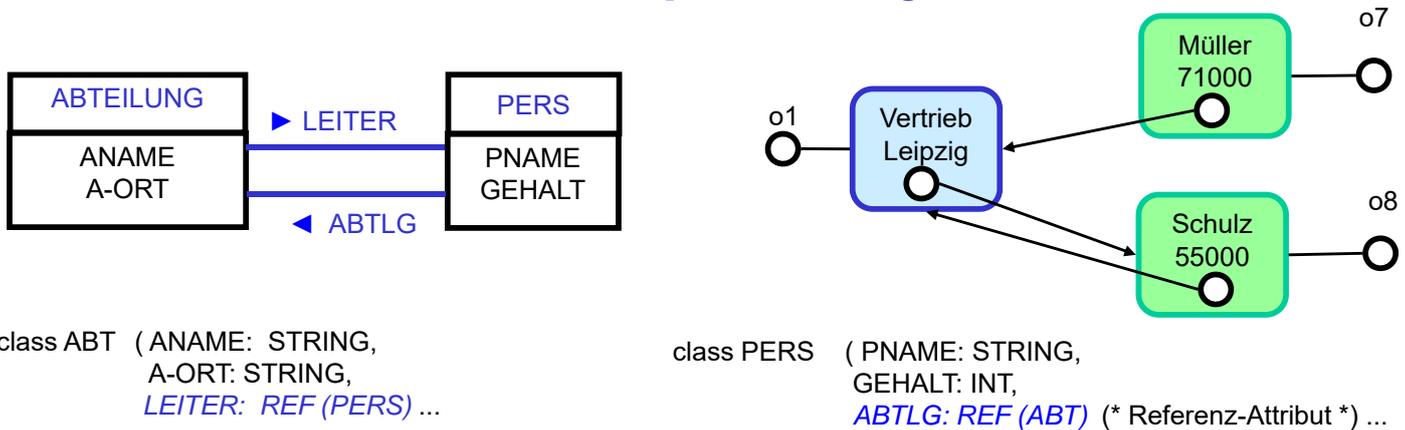
- systemweit eindeutige Objekt-Identifikatoren
- OID während Objektlebensdauer konstant, üblicherweise systemverwaltet
- OID tragen keine Semantik (<-> Primärschlüssel im RM)
- Änderungen beliebiger Art (auch des Primärschlüssels im RM) ergeben *dasselbe* Objekt

## ■ Vorteile gegenüber „wertebasiertem“ Relationenmodell

- Trennung von Identität und (Werte) Gleichheit
- Notwendigkeit künstlicher Primärschlüssel wird vermieden
- Beziehungen können über stabile OID-Referenzen anstelle von Fremdschlüsseln realisiert werden
- einfachere Realisierung komplexer (aggregierter) Objekte
- effizienterer Zugriff auf Teilkomponenten



## OIDs: Komplexe Objekte



## ■ Realisierung von Beziehungen über OIDs (Referenz-Attribute)

- Objekt-Ids / Referenzen erlauben Bildung komplexer Objekte bestehend aus Teilobjekten
- gemeinsame Teilobjekte ohne Redundanz möglich (referential sharing)

## ■ implizite Dereferenzierung über *Pfadausdrücke* anstatt expliziter Verbundanweisungen

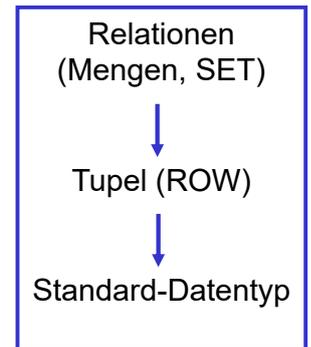
PERS->ABTLG->A-ORT



# Komplexe Objekte: Typkonstruktoren

## ■ Relationenmodell

- nur einfache Attribute, keine zusammengesetzte oder mengenwertige Attribute
- nur zwei Typkonstruktoren: Bildung von Tupeln und Relationen (Mengen)
- keine rekursive Anwendbarkeit von Tupel- und Mengenkonstruktoren



## ■ OODBS

- Objekte können Teilobjekte enthalten (Aggregation): eingebettet (Komposition, Wertesemantik) oder über OIDs referenziert
- Objektattribute können sein:
  - einfach (Standardtypen: Integer, Char, ...)
  - über Typkonstruktoren strukturiert / zusammengesetzt
  - Instanzen von (benutzerdefinierten) Objekttypen
  - Referenzen



# Komplexe Objekte: Typkonstruktoren

## ■ Typkonstruktoren zum Erzeugen strukturierter (zusammengesetzter)

Datentypen aus Basistypen

- **TUPLE** (ROW, RECORD)
- **SET**, **BAG** (MULTISET)
- **LIST** (SEQUENCE), **ARRAY** (VECTOR)

## ■ SET/BAG/LIST/ARRAY verwalten homogene Kollektionen:

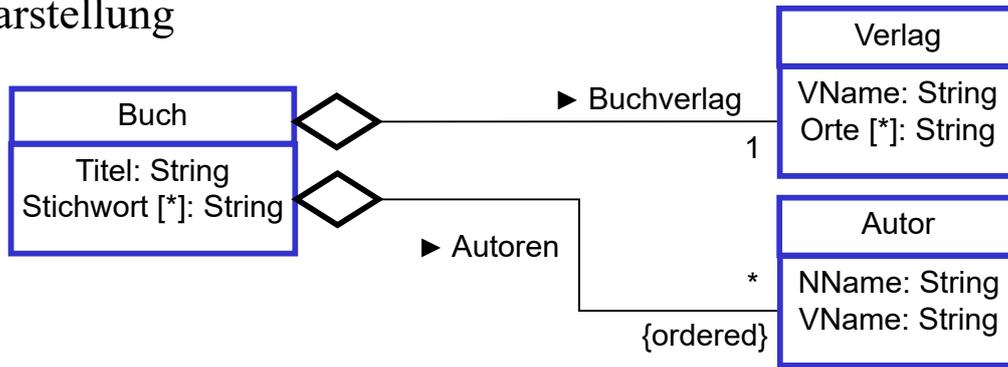
Kollektionstypen

Typ	Duplikate	Ordnung	Heterogenität	#Elemente	Elementzugriff über
<b>TUPLE</b>	JA	JA	JA	konstant	Namen
<b>SET</b>	NEIN	NEIN	NEIN	variabel	Iterator
<b>BAG</b>	JA	NEIN	NEIN	variabel	Iterator
<b>LIST</b>	JA	JA	NEIN	variabel	Iterator / Position
<b>ARRAY</b>	JA	JA	NEIN	konstant	Index



# Komplexe Objekte: Beispiel

## UML-Darstellung



## Verwendung von Typkonstruktoren:

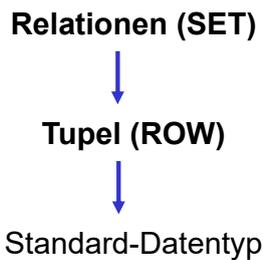
**class** AUTOR ( Nname, VName: String )

**class** VERLAG (

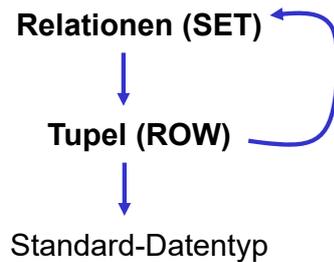
**class** BUCH (



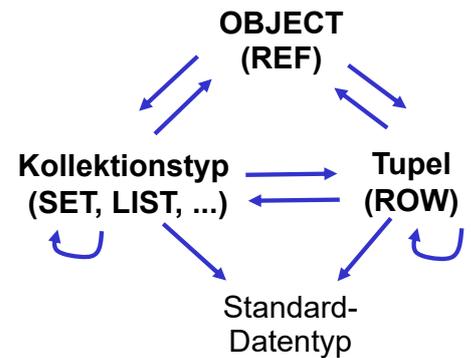
# Vergleich Datenmodelle



Relationenmodell



NF2



Objektmodell

- Ziel: beliebige (rekursive) Kombinierbarkeit der Typkonstrukturen
- OODBS-Standardisierung erfolgte im Rahmen der ODMG (Object Data Management Group): [www.odmg.org](http://www.odmg.org)



## Objekt-relationale DBS: Merkmale

- Erweiterung des relationalen Datenmodells um Objekt-Orientierung
- Bewahrung der Grundlagen relationaler DBS, insbesondere deklarativer Datenzugriff (Queries), Sichtkonzept etc.
- alle Objekte müssen innerhalb von Tabellen verwaltet werden
- Standardisierung von ORDBS v.a. in SQL1999 und SQL2003
- komplexe, nicht-atomare Attributtypen (z.B. relationenwertige Attribute)
- erweiterbares Verhalten über gespeicherte Prozeduren und benutzerdefinierte Datentypen und Funktionen (z.B. für Multimedia, ...)



## Grobvergleich nach Stonebraker

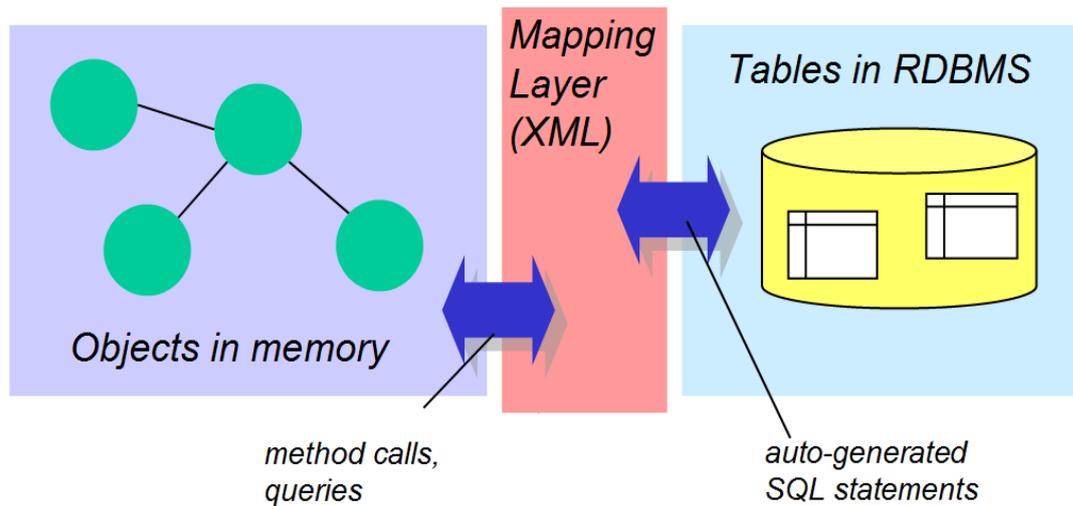
query	Relationale DBS	
no query	Dateisysteme	
	simple data	complex data

- kein Systemansatz erfüllt alle Anforderungen gleichermaßen gut
  - relationale DBS: einfache Datentypen, Queries, ...
  - OODBS: komplexe Datentypen, gute Programmiersprachen-Integration, hohe Leistung für navigierende Zugriffe
  - ORDBS: komplexe Datentypen, Querying ...
- geringe Marktbedeutung von OODBS gegenüber ORDBS

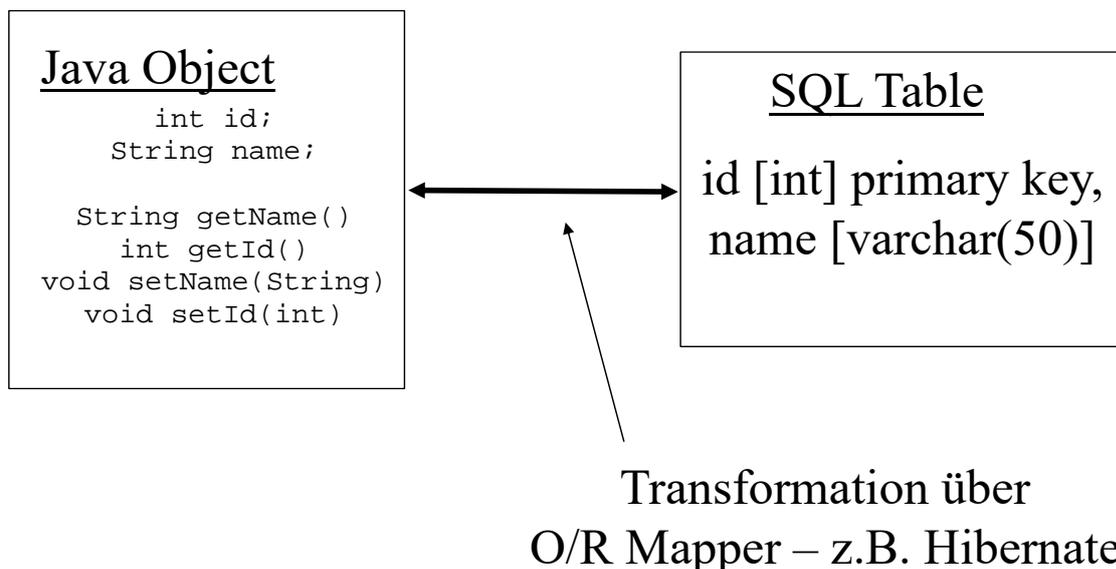


# O/R Mapping Frameworks

- Zwischenschicht zur Abbildung zwischen objektorientierten Anwendungen und relationalen Datenbanken (O/R-Mapping)
  - komfortablere Abbildung zwischen Anwendungsobjekten und Datenbank als mit SQL-Einbettung / CLI (z.B. JDBC), u.a. für komplexe Objekte
  - einheitliche Manipulation transienter und persistenter Objekte
  - größere Unabhängigkeit gegenüber DB-Aufbau



## Object/Relational Mapping



# O/R Mapping Frameworks

- Anforderungen
  - flexibles Mapping für Vererbungshierarchien, komplexe Objekte, ...
  - Persistenzverwaltung
  - Transaktionsverwaltung
  - Query-Unterstützung
  - Caching
- Unterstützung innerhalb von Software-Entwicklungsarchitekturen
  - J2EE: Enterprise Java Beans (EJB)
  - .NET-Framework
- leichtgewichtige Frameworks
  - Java Persistence API (JPA)
  - **Hibernate**
  - Entity framework (für .NET-Programmiersprachen wie C#)
  - Python SQLAlchemy
  - PHP Doctrine



## Hibernate



- Open-Source-Framework zum O/R-Mapping, u.a. für Java-Objekte
- gleichartige Verarbeitung transienter und persistenter Objekte
- flexible Mapping-Optionen über XML-Konfigurationsdateien
- Vererbungsoptionen:
  - table-per-class
  - table-per-class-hierarchy
  - table-per-subclass
- Query-Sprache HQL (SQL-Anfragen weiterhin möglich)
- Konfigurationsoptionen
  - Caching-Optionen (session, shared, distributed)
  - Lese/Ladestrategien („lazy loading“ von Objektmengen)
  - Schreibstrategien (WriteOnCommit, BatchUpdate)
  - Locking: optimistisch (timestamp) oder pessimistisch



# Hibernate Mapping: Beispiel

Person.java

```
public class Person {
    private java.util.Date birthday;
    private Name name;
    private String key;
    public String getKey() {
        return key;
    }
    private void setKey(String key) {
        this.key=key;
    }
    public java.util.Date getBirthday() {
        return birthday;
    }
    public void setBirthday(java.util.Date birthday) {
        this.birthday = birthday;
    }
    public Name getName() {
        return name;
    }
    public void setName(Name name) {
        this.name = name;
    }
    .....
    .....
}
```

person.hbm.xml

```
<hibernate-mapping>
  <class name="Person" table="person"
    <id name="key" column="pid" type="string"
      /> >
    <generator class="native"/>
  </id>
  <property name="birthday" type="date"/>
  <component name="Name" class="Name"
    <property name="initial" />
    <property name="first" />
    <property name="last" />
  </component>
</class>
</hibernate-mapping>
```

person table

Column Name	Data Type	Length	Allow Nulls
pid	varchar	20	
birthday	datetime	8	✓
initial	char	1	✓
[first]	varchar	50	✓
[last]	varchar	50	✓

```
public class Name {
    char initial;
    String first;
    String last;
    public String getFirst() {
        return first;
    }
    void setFirst(String first) {
        this.first = first;
    }
    public String getLast() {
        return last;
    }
    void setLast(String last) {
        this.last = last;
    }
    public char getInitial() {
        return initial;
    }
    void setInitial(char initial) {
        this.initial = initial;
    }
}
```

Name.java



# Hibernate Anwendung: Beispiel

```
Session s = factory.openSession();
Transaction tx = null;
try {
    tx = s.beginTransaction();
    Name n = new Name();
    Person p = new Person();
    n.setFirst („Robin“); n.setLast („Hood“);
    p.setName(n); ...
    s.save(p);
    tx.commit();

    Query q1 = s.createQuery("from Person");
    List l = q1.list()
    //Ausgabe ...

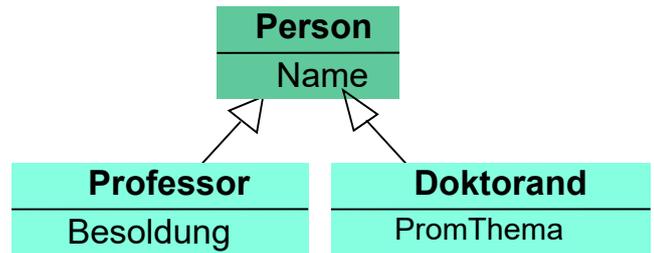
    s.close();
} catch (HibernateException e) {
    e.printStackTrace();
}
```



# Hibernate Generalisierung (1)

- Table-per-subclass: vertikale Partitionierung
- Beispiel (Ausschnitt)

```
<class name="Person" table="PERSON">  
  <id name="id" type="long" column="ID"> ... </id>  
  <property name="Name" column="NAME"/>  
  <joined-subclass name="Professor" table="PROF">  
    <key column="ID"/>  
    <property name="Besoldung" column="BESOLDUNG"/>  
  </joined-subclass>  
  <joined-subclass ...  
</class>
```



PERSON	
ID	NAME
1	Rahm
2	Franke

PROF	
ID	BESOLDUNG
1	C4

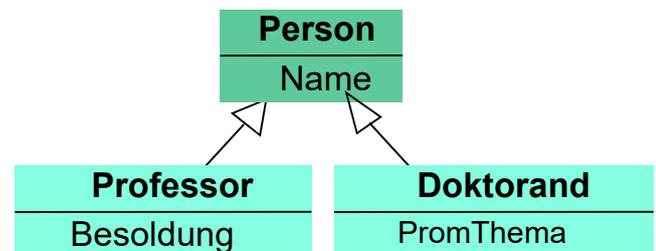
DOKTORAND	
ID	PROMTHEMA
2	Privacy



# Hibernate Generalisierung (2)

- Table-per-(concrete) class: Horizontale Partitionierung
- Beispiel (Ausschnitt)

```
<class name="Person" table="PERSON">  
  <id name="id" type="long" column="ID"> ... </id>  
  <property name="Name" column="NAME"/>  
  <union-subclass name="Professor" table="PROF">  
    <property name="Besoldung" column="BESOLDUNG"/>  
  </union-subclass>  
  <union-subclass ...  
</class>
```



- Spezialfall, wenn Spezialisierung vollständig
  - Andernfalls auch eine Tabelle für Superklasse Person.

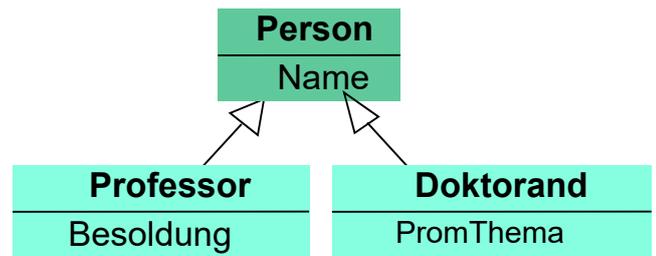
PROF		
ID	NAME	BESOLDUNG
1	RAHM	C4

DOKTORAND		
ID	NAME	PROMTHEMA
2	Franke	Privacy



# Hibernate Generalisierung (3)

- **Table-per-class-hierarchy:**  
eine Tabelle pro Hierarchie ("wide table")
  - Spezialisierung durch Discriminator-Spalte



- **Beispiel (Ausschnitt)**

```
<class name="Person" table="PERSON">
  <id name="id" type="long" column="ID"> ... </id>
  <discriminator column="TYP" type="string"/>
  <property name="Name" column="NAME"/>
  <subclass name="Professor" discriminator-value="Prof">
    <property name="Besoldung" column="BESOLDUNG"/>
  </subclass>
  <subclass ...
</class>
```

PERSON				
ID	NAME	TYP	BESOLDUNG	PROMTHEMA
1	Rahm	Prof	C4	NULL
2	Franke	Doktorand	NULL	Privacy



## Zusammenfassung

- **RM unterstützt anspruchsvollere Anwendungen nur bedingt**
  - Multimedia-DBS, Deduktive DBS, Geo-DBS, CAD-DBS, Bio-DBS, XML-DBS, ...
- **OODBS (und ORDBS) unterstützen**
  - komplexe Objekte und Objektidentität
  - Typhierarchien und Vererbung
  - Erweiterbarkeit bezüglich Objekttypen und Verhalten
- **strikte Kapselung zu inflexibel (Ad-hoc-Anfragemöglichkeit wichtig)**
- **Bewertung OODBS gegenüber ORDBS**
  - einheitliche Bearbeitung transienter und persistenter Daten über objekt-orientierte Programmierschnittstelle (enge Programmiersprachenintegration)
  - hohe Leistung für navigierende Zugriffe, z.B. in Entwurfsanwendungen (CAD)
  - geringe Marktbedeutung
- **ORDBS**
  - Erweiterung des RM / SQL um objekt-orientierte Konzepte
  - NF2 erweitert Relationenmodell, jedoch unzureichend
- **O/R-Mapping: persistente Objekte mit relationalen DB**

