

# 3. Von relationalen zu objekt-relationalen DBS

- Beschränkungen des relationalen Datenmodells
  - Beispiel: Modellierung von 3D-Objekten
  - Impedance Mismatch
  - Anwendungsgebiete mit besonderen Anforderungen
- Entwicklung von Datenmodellen
- NF2-Ansatz
- OODBS und ORDBS
  - Objekttypen, Kapselung
  - Generalisierung, spätes Binden
  - komplexe Objekte: Objektidentität, Typkonstruktoren
  - ORDBS-Besonderheiten



## Objekt-Darstellung

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

### Ausprägungen

#### Schema

**ANGESTELLTER**

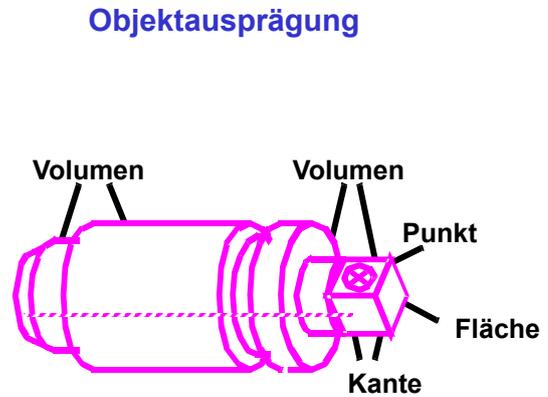
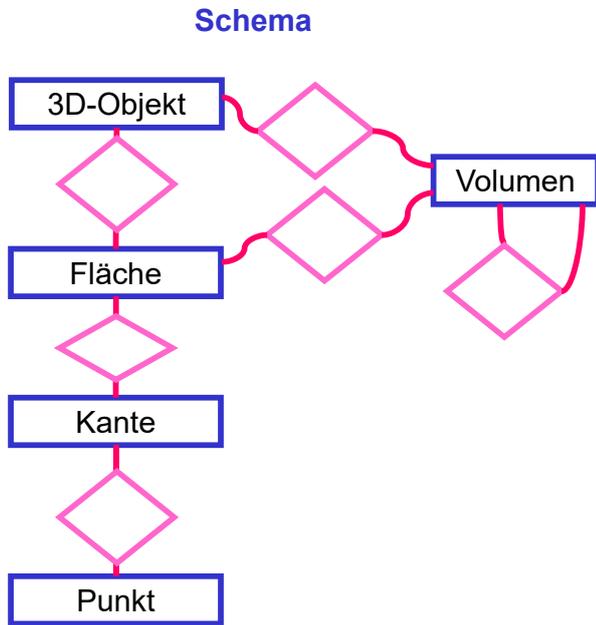
Satztyp (Relation)

PNR	NAME	TAETIGKEIT	GEHALT	ALTER
496	Lange	Pfoertner	3100	53
497	Christen	Kopist	3800	45
498	Franke	Kalligraph	4500	56

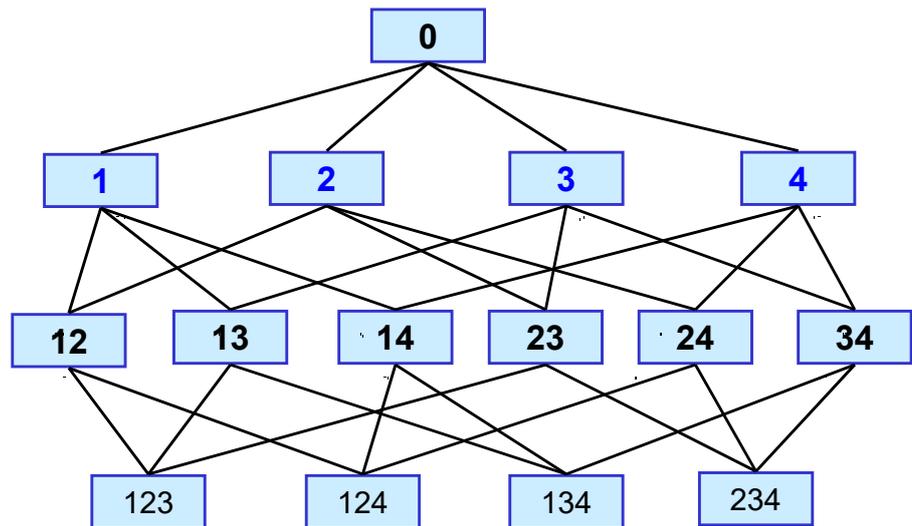
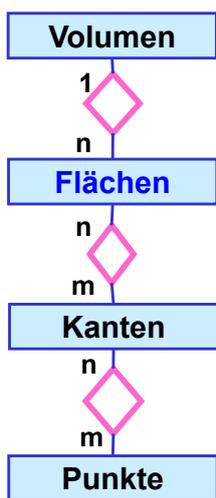


# Objekt-Darstellung (2)

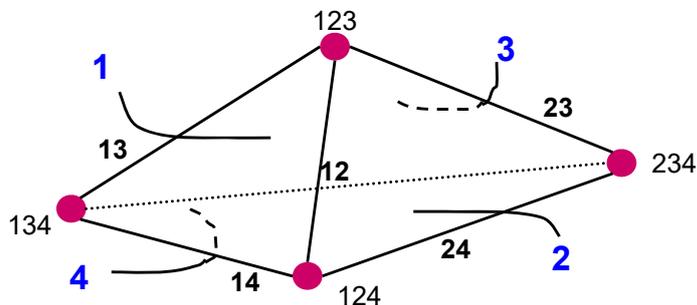
- CAD-Anwendung: das komplexe Objekt „Werkstück“ setzt sich aus einfacheren (komplexen) Objekten verschiedenen Typs zusammen



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

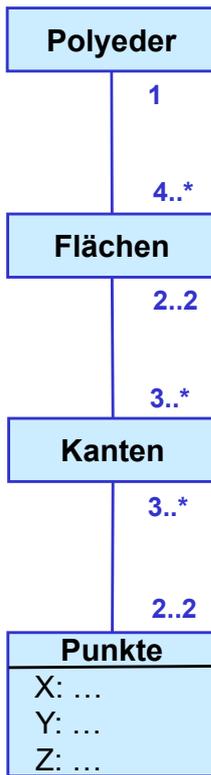


Beispiel: Tetraeder



# 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);
    
```

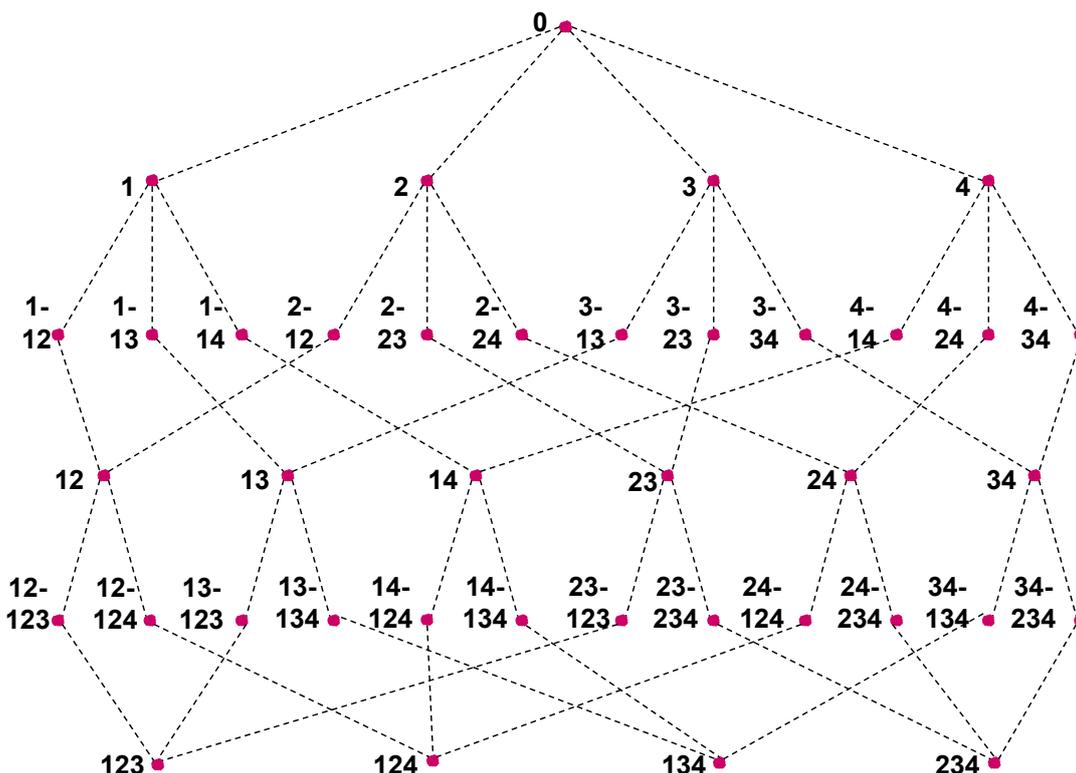
```

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 Fläche F NATURAL JOIN FK-Rel
      NATURAL JOIN Kante
      NATURAL JOIN KP-Rel
      NATURAL JOIN Punkt P
WHERE F.fid < 3
```

- aufwändige Join-Anfrage zur Rekonstruktion komplexer Objekte

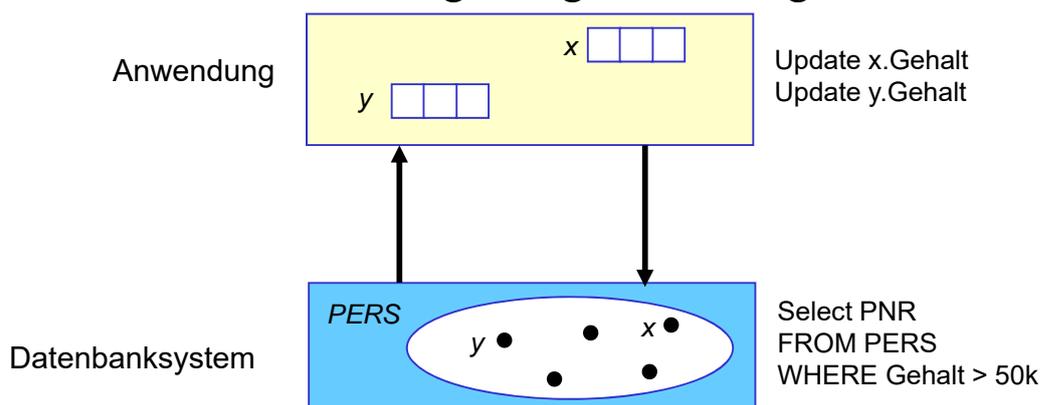
- Flächen, die mit Punkt (50,44,75) assoziiert sind

```
SELECT F.fid
FROM Punkt P NATURAL JOIN KP-Rel
      NATURAL JOIN Kante
      NATURAL JOIN FK-Rel
      NATURAL JOIN Fläche F
WHERE P.x = 50 AND P.y = 44 AND P.z = 75
```



## 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



# Heterogene Datenarten

- Multimedia-Daten (Bilder, Videos, Audio, Text ...)
  - großer Datenumfang und aufwändige Operationen
  - Speicherung in Dateien: fehlende Anfragemöglichkeiten, kein Transaktionsschutz ...
- Dokumente/Texte ohne feste Struktur, semistrukturierte Daten (XML, JSON)
  - erfordern unscharfe Textsuche (Homonym/Synonym-Probleme etc.), Relevanzbewertung (Precision, Recall), Ranking von Ergebnissen
  - semi-strukturierte Daten (Mischung von Text und strukturierten Daten): optionaler Schemaeinsatz
- Geoinformationssysteme: räumliche Daten und Operationen
- vernetzte Daten/Graph-Daten: schnelle Traversierung statt aufwändige Joins
- ...
- unterschiedliche Datenarten erfordern zugeschnittene Speicherungs- und Auswertungsmöglichkeiten
- Erweiterbarkeit von DBS auf unterschiedliche Datenarten von Daten oder Spezialsysteme (z.B. Information-Retrieval-Systeme, NoSQL)



## 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

*Anfrage:* ? Vorfahr (x, A)

*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) <-

*Regeln:*

R1: Vorfahr (x, y) <- Elternteil (x, y)  
R2: Vorfahr (x, y) <- Elternteil (x, z), Vorfahr (z, y)  
  
R3: Geschwister (x, y) <- Elternteil (z, x), Elternteil (z, y), x <math>\diamond</math> y



# Beschränkungen des Relationenmodells

- „Schema First“
  - Fokus auf strukturierte Daten mit starrem Schema
- nur einfach strukturierte Daten
  - einfache (Standard-) Datentypen
  - Sätze mit fester Anzahl atomarer Attribute (festes Satzformat)
- Relationenmodell ist "wertebasiert"
  - Identifikation von Daten durch Schlüsselwerte, oft über künstliche Schlüsselattribute
  - umständliche Modellierung von Beziehungen über Fremdschlüssel
- 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

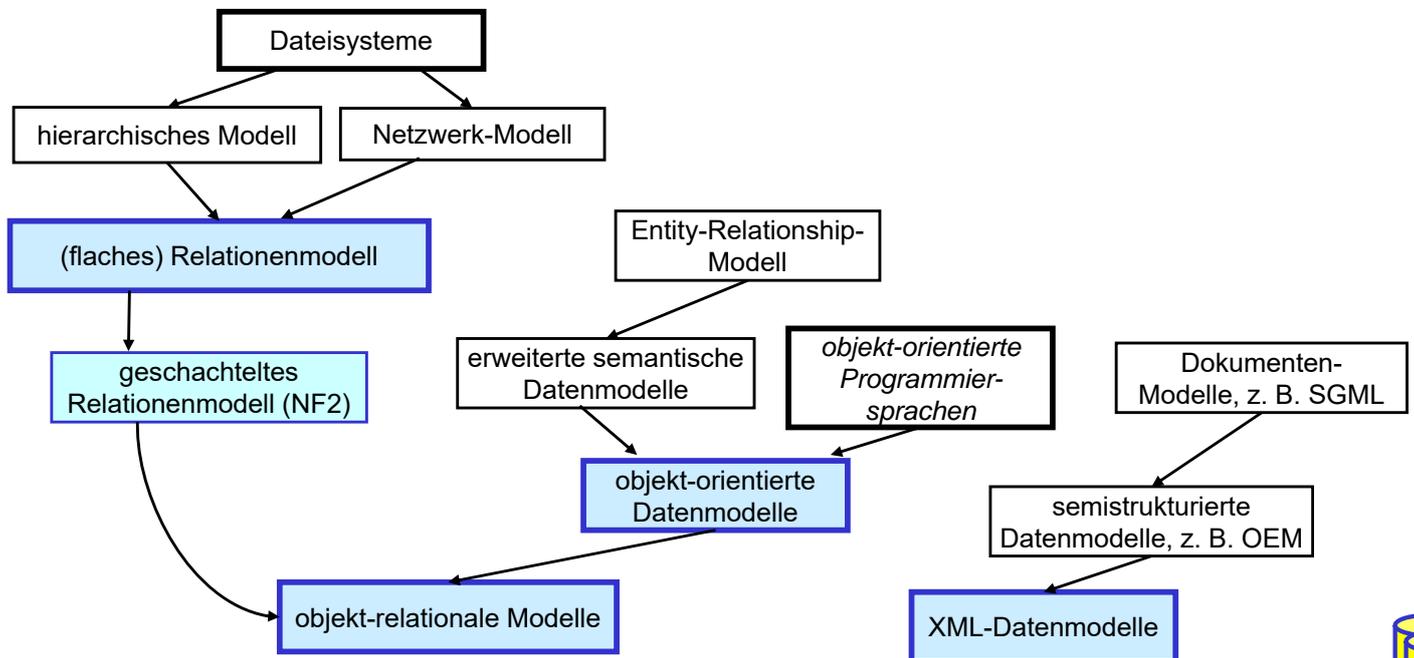


# Entwicklung von Datenmodellen

## ■ (logisches) Datenmodell

- generische Datenstruktur zum Aufbau von Datenbanken (z.B. Relationen im RM)
- Menge von Operatoren und modellinhärenten Integritätsbedingungen auf Datenstruktur

## ■ grobe historische Entwicklung:



## NF2-Modell

### ■ Non-First Normal Form (NF<sup>2</sup>)

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

### ■ 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		Kanten			
		Fld	Kld	Punkte			
				Pld	X	Y	Z
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



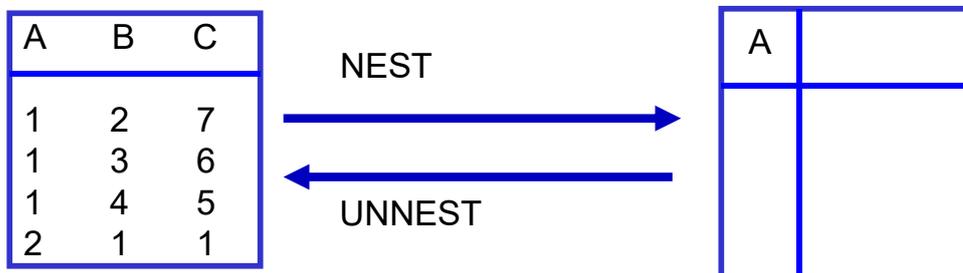
## NF<sup>2</sup>-Modell: Operatoren

### 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  $A_{n+1}, \dots, A_m$ )



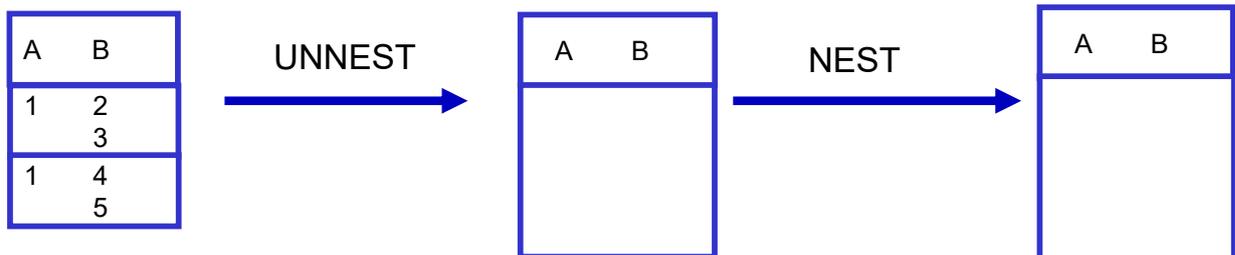
## NF2-Modell: Operatoren (2)

### ■ 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 !

- somit i.a.  $NEST(UNNEST(R)) \neq R$



## 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

$R1 \bowtie R2$


# Bewertung des NF2-Modells

## ■ Vorteile:

- einfaches Relationenmodell als Spezialfall enthalten
- Unterstützung komplex-strukturierter Objekte / Clusterung
- reduzierte Join-Häufigkeit
- 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 und ORDBS

## ■ objekt-orientierte Datenverwaltung erfordert

- Modellierung der Struktur (komplexe Objekte, Vererbung) +
- Verhalten (abstrakte/benutzerdefinierte Datentypen, Methoden)

## ■ 2 Ansätze

- Anreicherung von objekt-orientierten Programmiersprachen um DB-Eigenschaften (Persistenz, Integrität, ...) -> *persistente Programmiersprachen / objekt-orientierte DBS (OODBS)*
- Anreicherung von DBS um objekt-orientierte Konzepte -> *objektrelationale DBS (ORDBS)*



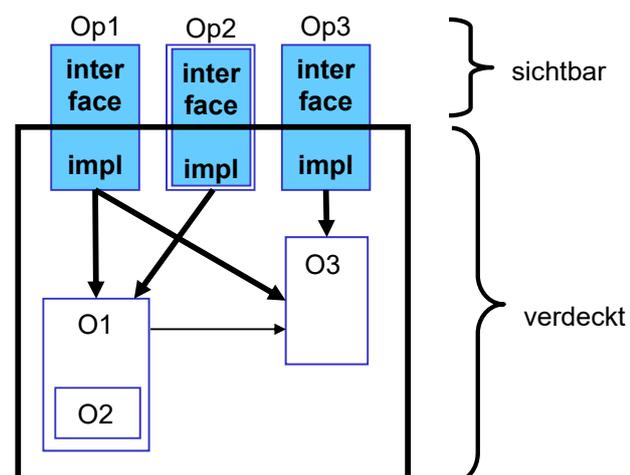
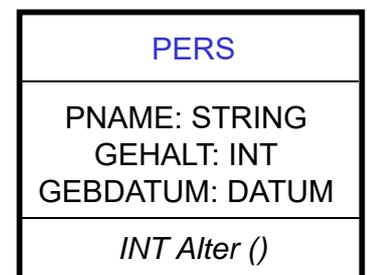
# Definition eines objekt-orientierten DBS

- OODBS muss DBS + objekt-orientiertes System sein
- *DBS-Aspekte*: Persistenz, Datenunabhängigkeit, Transaktionsverwaltung, Ad-Hoc-Anfragesprache
- *OOS-Aspekte*:
  - Objekttypen, Kapselung
  - Typ-/Klassenhierarchie mit Methoden, Vererbung, Überladen und spätes Binden
  - Objektidentität, komplexe Objekte
  - operationale Vollständigkeit
- OODBS (im Unterschied zu ORDBS)
  - Erweiterung objektorientierter Programmierung um DB-Sprache
  - Beseitigung des “impedance mismatch”/ bessere Effizienz
  - einheitliche Verwaltung transienter und persistenter Objekte
  - effiziente Traversierung komplexer (persistenter) Objektstrukturen

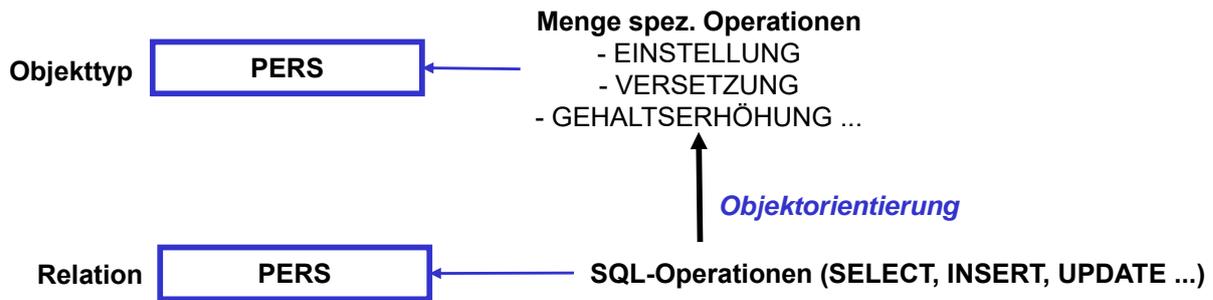


## Objekttypen / Kapselung

- 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)

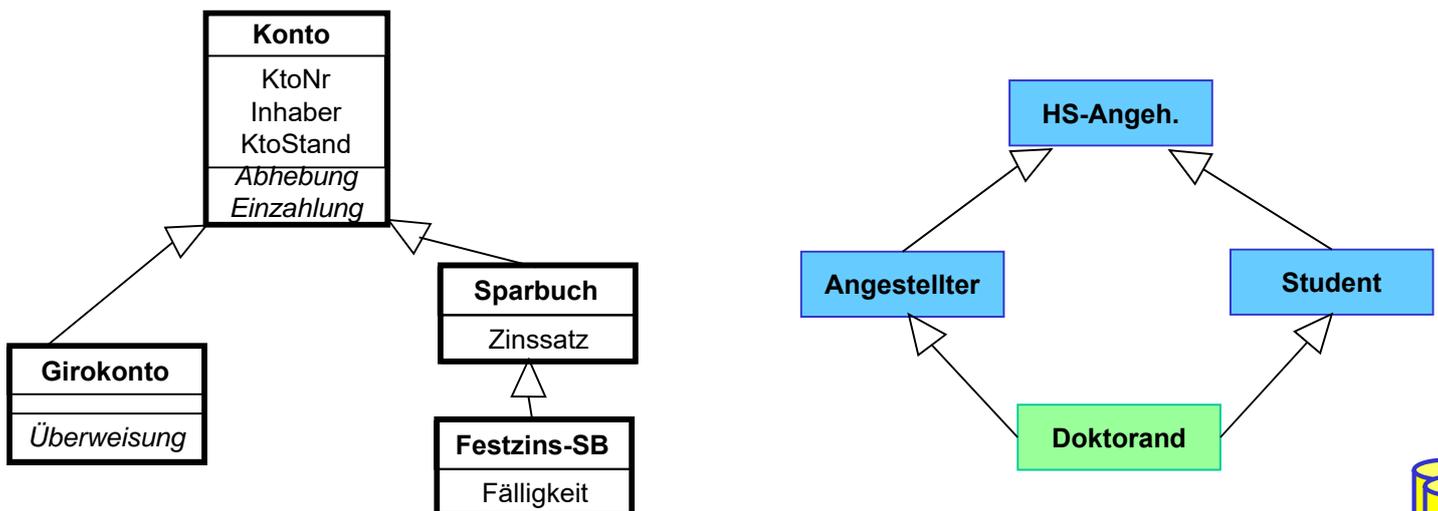


- Verwaltung von Objekttypen und Operationen im DBS
  - zusätzliche Anwendungsorientierung im DBS gegenüber Stored Procedures
  - verringerte Kommunikationshäufigkeit zwischen Anwendung und DBS
  - Reduzierung des "impedance mismatch"
- Vorteile der Kapselung: höherer Abstraktionsgrad
  - logische Datenunabhängigkeit, Datenschutz
- aber: strikte Kapselung oft zu restriktiv (unflexible, keine Ad-Hoc-Anfragen)
- Kompromiss: Zugriff aus ausgewählte Attribute durch get/put-Methoden



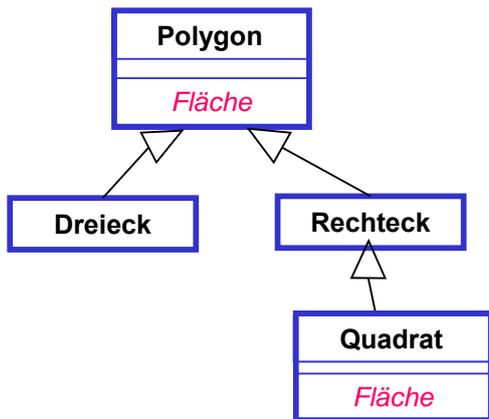
# Generalisierung

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



# Überladen (Overloading)

- derselbe Methodenname wird für unterschiedliche Prozeduren verwendet (polymorphe Methoden)
  - erleichtert Nutzung 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*)

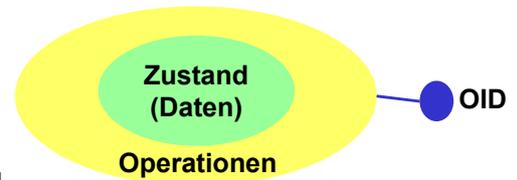


```
Quadrat q;
Rechteck r;
...
Polygon p1 = q;
Polygon p2 = r;
...
double f1=p1.Fläche();
double f2=p2.Fläche();
```

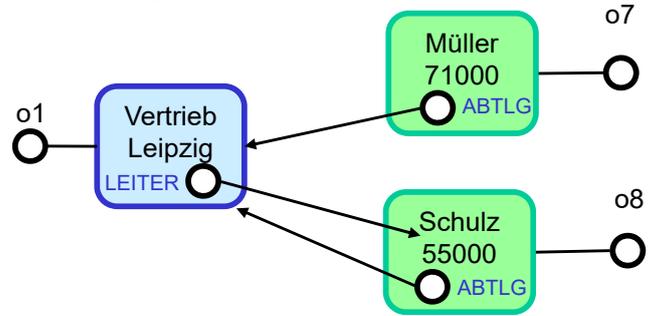
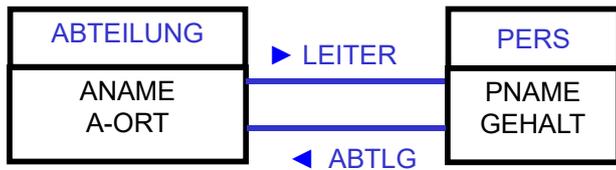


## Objektidentität

- 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 (OID- vs. Zustandsübereinstimmung)
  - keine Notwendigkeit künstlicher Primärschlüssel
  - Beziehungen realisierbar über stabile OID-Referenzen anstelle von Fremdschlüsseln
    - einfachere Realisierung komplexer (aggregierter) Objekte
    - effizienterer Zugriff auf Teilkomponenten (ohne Joins)



# OIDs: Komplexe Objekte



class ABT ( ANAME: STRING,  
A-ORT: STRING,  
LEITER: REF (PERS) ...

class PERS ( PNAME: STRING,  
GEHALT: INT,  
ABTLG: REF (ABT) (\* Referenz-Attribut \*) ...

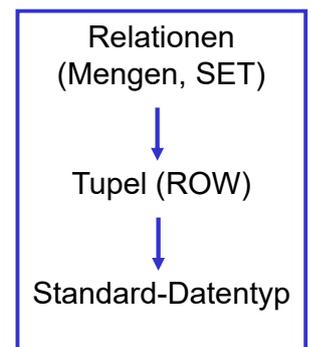
- Realisierung von Beziehungen über OIDs (Referenz-Attribute)
  - Objekt-Ids / Referenzen erlauben Bildung komplexer Objekte bestehend aus Teilobjekten
  - gemeinsame Teilobjekte ohne Redundanz möglich (referential sharing)
- implizite Dereferenzierung über *Pfadausdrücke* anstatt expliziter Verbundanweisungen
  - p->ABTLG.A-ORT (p sei PERS-Objekt)
  - a->LEITER.PNAME (a sei ABTEILUNG-Objekt)



## 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)
    - einfache Attribute (Standardtypen: Integer, Char, ...)
    - über Typkonstruktoren strukturierte / zusammengesetzte Attribute
    - Instanzen von (benutzerdefinierten) Objekttypen
  - Referenzen (über OID) auf benutzerdefinierte Objekttypen (Referenzsemantik)
  - Kombinationen



# 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:
 

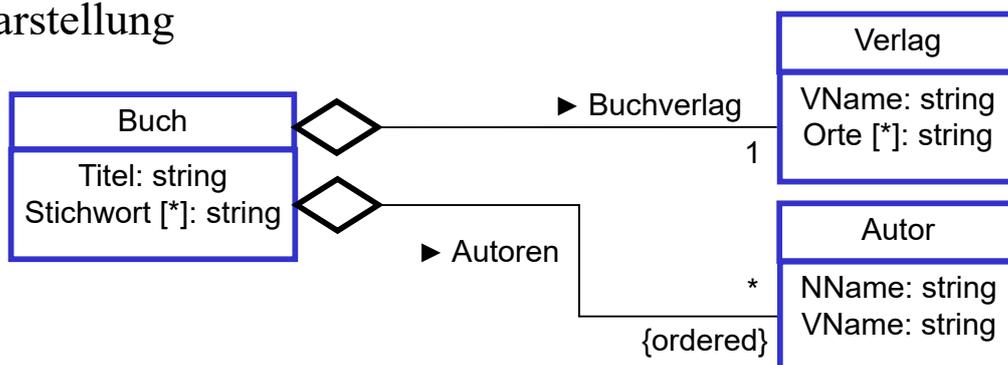
## Kollektionstypen

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



## Komplexe Objekte: Beispiel

- UML-Darstellung



- Verwendung von Typkonstruktoren:

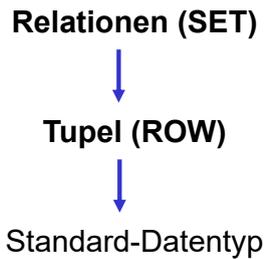
```
class AUTOR ( NName, VName: string )
```

```
class VERLAG ( VName: string,
               Orte: SET (string))
```

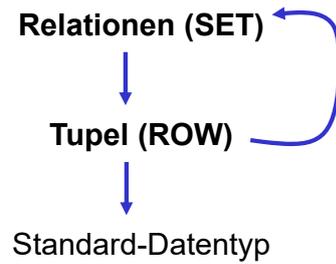
```
class BUCH ( Titel: string,
             Stichwort: SET (string),
             Buchverlag; REF (VERLAG),
             Autoren: LIST ( REF (AUTOR) ) )
```



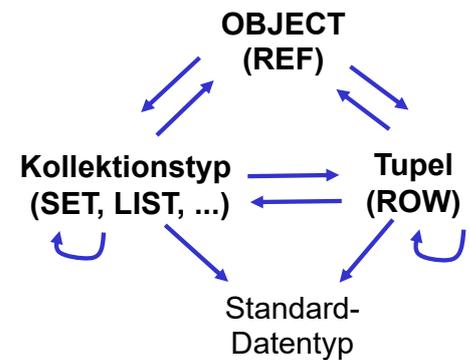
# Vergleich Datenmodelle



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 SQL:1999 und SQL:2003
- komplexe, nicht-atomare Attributtypen (z.B. relationenwertige Attribute)
- erweiterbares Verhalten über gespeicherte Prozeduren und benutzerdefinierte Datentypen und Funktionen



# 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 ...
- OODBS haben keine Marktbedeutung mehr -> ORDBS



## Zusammenfassung

- RM ist auf einfache strukturierte Daten beschränkt
  - unzureichende Unterstützung für komplexe Objekte, keine Erweiterbarkeit für neue Datenarten ...
- OODBS und ORDBS unterstützen
  - komplexe Objekte und Objektidentität
  - Typhierarchien und Vererbung
  - Erweiterbarkeit bezüglich Objekttypen und Verhalten
- strikte Kapselung zu inflexibel (Ad-hoc-Anfragemöglichkeit wichtig)
- ORDBS
  - Erweiterung des RM / SQL um objekt-orientierte Konzepte
  - NF2 erweitert Relationenmodell, jedoch unzureichend
- OODBS haben keine Marktbedeutung mehr
  - ermöglichen einheitliche Verarbeitung transienter und persistenter Daten, effiziente Navigation für komplexe Objekte
  - Teile der Eigenschaften über O/R-Mappingsoftware (z.B. Hibernate)

