



Moderne Datenbanken

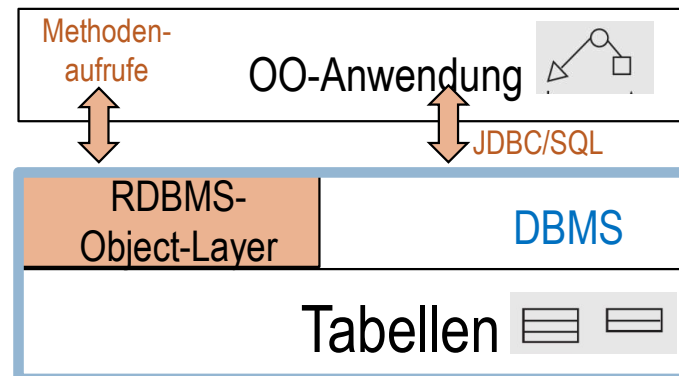
Negativ Beispiel: Objekt-Relationale Datenbanken

1	Objekt-Relationale Datenbankerweiterung	2
2	Anwendung der NF2-Erweiterung	10
2.1	Strukturierte Attribute	12
2.2	Mehrwertige Attribute	18
3	Anwendung objektorientierter Konzepte	33
3.1	Objekte identifizieren und referenzieren	34
3.2	Assoziationen	40
3.3	Aggregation	45
3.4	Vererbung	50
4	Fazit	56

Problemstellung

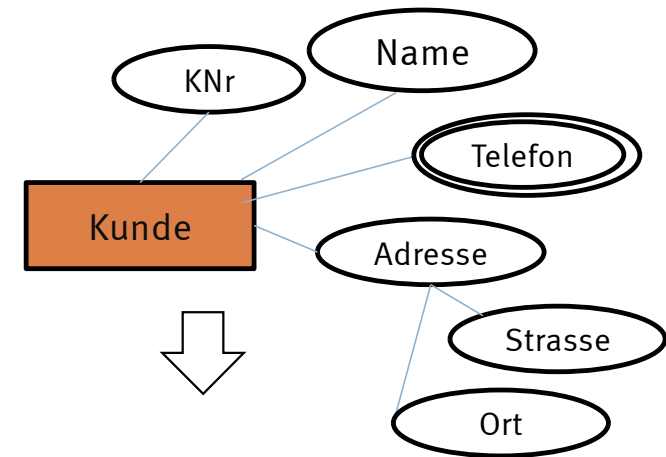


- Zusätzliche objektorientierte Abstraktionsschicht über dem relationalem Modell
 - Diese Objektebene (Object-Layer) ermöglicht nur eine einfache (und effiziente) Benutzung von Objekten
 - Beide Sichten – relationale und Objekt-relationale – können gleichzeitig benutzt werden



- Zielsetzung:
Erzeugung von benutzerdefinierten Objekten basierend auf
 - Oracle Datentypen
 - bereits definierten eigenen Datentypen
 - Objekt-Referenzen
 - Collection-Typen, z.B. Mengen, Arrays, eingebetteten Tabellen

- Erweiterungen in SQL:2003
 - Erweiterung um das **Non-First Normal Form (NF2)**-Konzept
 - Aufgabe der 1. Normalform
 - Attribute können nicht-atomar sein, z.B.
 - Collection: Menge von Werten innerhalb einer Spalte (Listen, Mengen, Array)
 - Relationen (eingeschachtelte Tabellen)
 - Erweiterung um objektorientierte Konzepte
 - Subtypkonzept mit Vererbung
 - Kapselung des Objektverhaltens
 - Objekttabellen als neue Tabellenart
 - Objekt-Identifikatoren (OID) und Referenzen (REF-Funktion)
 - Rekursion



KNr	Name	Adresse		Telefon	
		Strasse	Ort	TID	Tel
2310	Meitner	Xyz-str.3	Paris	123	0231
				124	0173
				153	0169

Ein (benutzerdefinierter) Objekt-Typ ist ein Schema-Objekt. Es enthält Attribute bestehend aus SQL-Datentypen und benutzerdefinierte Datentypen sowie Methoden, die das Verhalten der Objekte beschreiben.

- Syntax

```
CREATE TYPE <typ_name>
{AS OBJECT | UNDER <supertyp_name>}
  (<attributdefinitions- und methodendeklarationsliste>
   [<abbildungs- bzw- ordnungsmethode>]
  )
[[NOT] FINAL]
[[NOT] INSTANTIABLE]
```

- Beispiel

```
CREATE TYPE Person_Typ AS OBJECT
( PNr          INT,
  Nachname     VARCHAR(50),
)
```

- Die Typdefinition enthält keine Default-Werte und keine Integritätsbedingungen



- Statische Integritätsbedingungen sind für Objekttabellen definierbar

- PRIMARY KEY
- UNIQUE
- FOREIGN KEY
- NOT NULL
- CHECK (nicht auf Kollektionen!)
- SCOPE (Referenzbeschränkung - Wertebereich)

Integritätsbedingungen werden bei der Verwendung in den Tabellen definiert!



Beispiel:

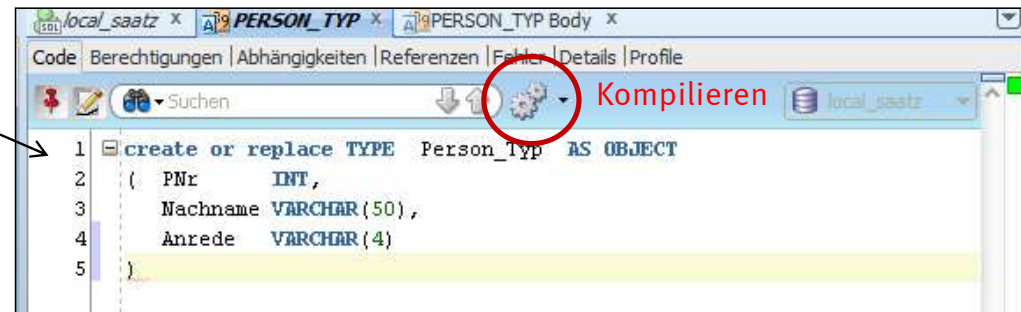
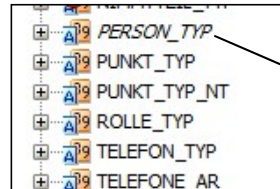
Objekttyp:

```
CREATE TYPE Person_Typ AS OBJECT  
( Pnr          Integer,  
  Nachname     VARCHAR(30),  
  Adresse       Adress_Typ  
)
```

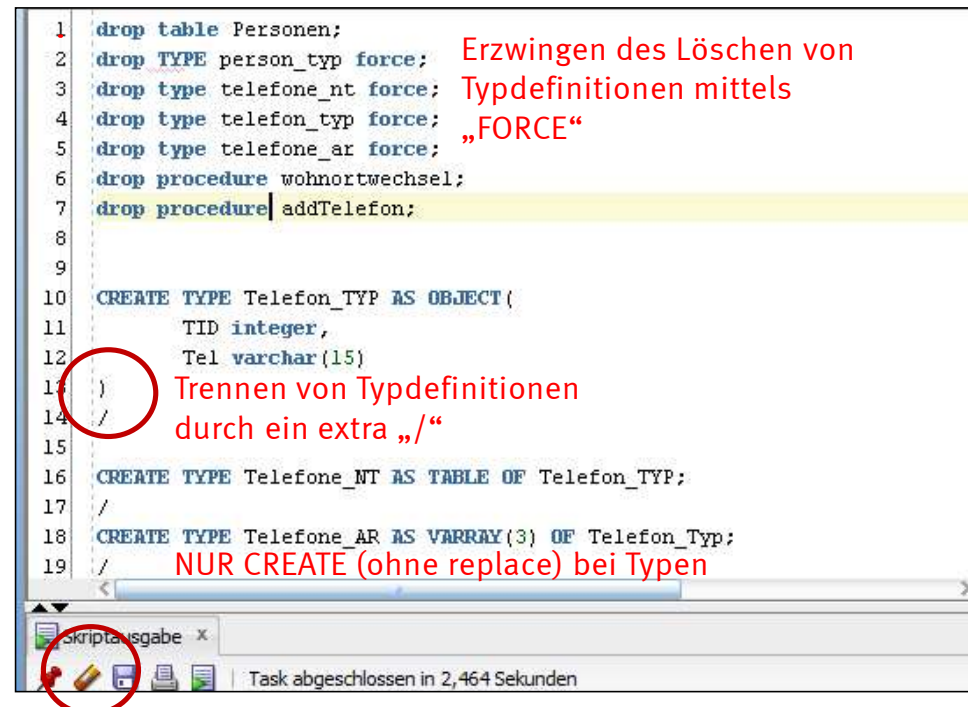
Objekttabelle:

```
CREATE TABLE Person OF Person_Typ  
( Pnr          PRIMARY KEY,                               Integritätsbedingungen  
  Nachname     NOT NULL,  
  CONSTRAINT address_constraint_Strasse UNIQUE (adresse.Strasse),  
  CONSTRAINT address_constraint_Ort    CHECK (adresse.Ort IS NOT NULL)  
)
```

Typen-
liste



Installationsskript erstellen



Ausgabe zwischendurch Löschen
(nächste Ausgabe wird angehängt)

Das Verhalten von Objekten wird in Methoden kodiert. **Member-Methoden** werden auf der Objektinstanz ausgeführt und besitzen einen SELF-Operator. Eine **statische** Methode, wird auf dem Objekttyp aufgerufen und besitzt **keinen** SELF-Operator.

Deklaration:

```
CREATE TYPE Person_Typ AS OBJECT
( PNr          INT,
  Nachname     VARCHAR(50),
  STATIC FUNCTION helloWorld RETURN VARCHAR2,
  MEMBER FUNCTION helloPerson RETURN VARCHAR2
)
```

Implementierung:

```
CREATE TYPE BODY Person_Typ AS
  STATIC FUNCTION helloWorld RETURN VARCHAR2 IS
  BEGIN
    RETURN 'Hallo Welt';
  END;
  MEMBER FUNCTION helloPerson RETURN VARCHAR2 IS
  BEGIN
    RETURN 'Hallo ' || self.nachname;
  END;
END;
```

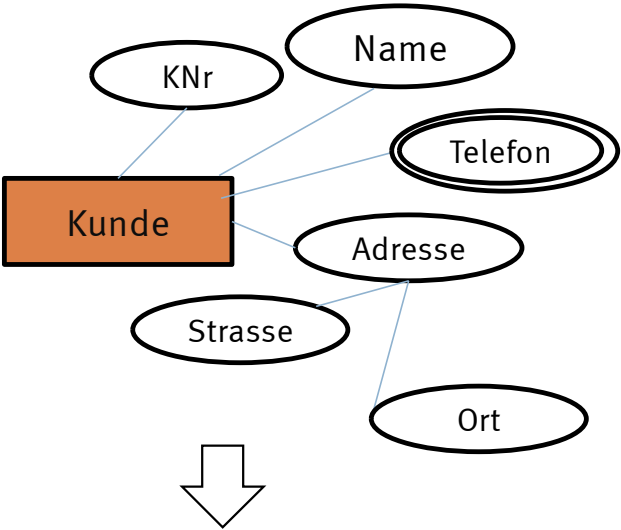
entspricht „this“ in JAVA

Verwendung:

```
Select Person_typ.helloWorld(), p.helloPerson() from Person p;
```

1	Objekt-Relationale Datenbankerweiterung	2
2	Anwendung der NF2-Erweiterung	10
2.1	Strukturierte Attribute	12
2.2	Mehrwertige Attribute	18
3	Anwendung objektorientierter Konzepte	33
3.1	Objekte identifizieren und referenzieren	34
3.2	Assoziationen	40
3.3	Aggregation	45
3.4	Vererbung	50
4	Fazit	56

Abbildung eines erweiterten ER-Modells



KNr	Name	Adresse		Telefon	
		Strasse	Ort	TID	Tel
2310	Meitner	Xyz-str.3	Paris	123	0231
				124	0173
				153	0169

1	Objekt-Relationale Datenbankerweiterung	2
2	Anwendung der NF2-Erweiterung	10
2.1	Strukturierte Attribute	12
2.2	Mehrwertige Attribute	18
3	Anwendung objektorientierter Konzepte	33
3.1	Objekte identifizieren und referenzieren	34
3.2	Assoziationen	40
3.3	Aggregation	45
3.4	Vererbung	50
4	Fazit	56

Benutzerdefinierter Objekt-Typ

Prof. Dr. I. M. Saatz

Moderne Datenbanken

FH Dortmund

- Ein (benutzerdefinierter) Objekt-Typ ist ein Schema-Objekt.
- Komponenten:
 - Name zur Identifizierung
 - Attributen zur Modellierung der Struktur des Objekts bestehend aus SQL-Datentypen und selbst wieder benutzerdefinierte Datentypen
 - Methoden, die das Verhalten der Objekte beschreiben
 - Z.B. automatisch erzeugter Objekt-Konstruktor
(innerhalb des DB-Kerns in PL/SQL bzw. Java, externe Zugriffsmethoden in C)

- Beispiel

- einfach

```
CREATE TYPE Adress_Typ AS OBJECT  
( Strasse      VARCHAR(50),  
  Ort          VARCHAR(30)  
)
```

- Verschachtelt

```
CREATE TYPE Person_Typ AS OBJECT  
( Nachname     VARCHAR(30),  
  Adresse      Adress_Typ  
)
```


Operationen auf benutzerdefinierten Typen

- Eine Instanz kann mit dem **Standard-Konstruktor** erzeugt werden.
Beispiel Standard-Konstruktor zum Typen Adress_Typ:

```
CREATE TYPE Adress_Typ AS OBJECT  
( Strasse      VARCHAR(50),  
  Ort          VARCHAR(30)  
)
```



Konstruktor

```
Adress_Typ()
```

- Attributzugriff erfolgt mittels Funktionsaufruf sowie Punkt-Operator
 - X.Attributname entspricht Attributname(X)
 - SET X.Attributname = Wert entspricht Attributname(X, Wert)

```
UPDATE Kunden k  
SET k.Adresse.Ort = 'Dortmund'  
WHERE k.knr = 2310
```

- Tabellendefinition:

```
CREATE TABLE Kunde  
( KNr          INTEGER PRIMARY KEY,  
  Adresse      Address_Typ  
)
```

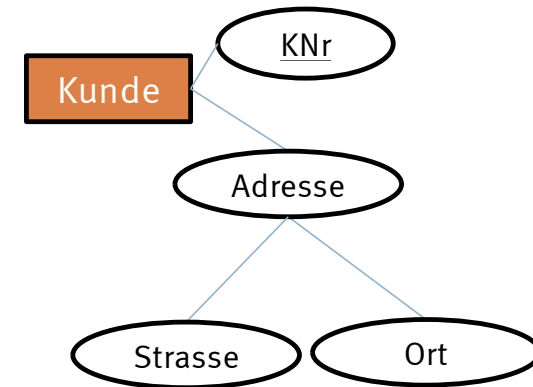
- Werten einfügen:

```
INSERT INTO Kunde VALUES  
( '2310',  
  Address_Typ('Strasse xy', 'Berlin'))
```

- Werte suchen:

- Zugriff auf Objekt-Typen über die Punkt-Notation

```
SELECT Kundenummer, k.Adresse.Strasse  
FROM Kunde k  
WHERE k.Adresse.Ort like 'Do%'
```



Konstruktoraufruf !

Punktnotation !

Benutzerdefinierte Objekttyp - Verwendung

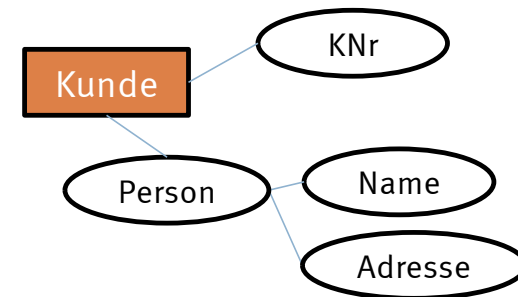
Prof. Dr. I. M. Saatz

Moderne Datenbanken

FH Dortmund

■ Tabellendefinition:

```
CREATE TABLE Kunde
( Kundennummer INTEGER PRIMARY KEY,
  Person Person_Typ
)
```



■ Werten einfügen:

```
INSERT INTO Kunde VALUES (
  ('2310',
   Person_Typ('Meitner', Adress_Typ('Strasse xy', 'Berlin'))
)
```

■ Werte suchen:

- Zugriff auf Objekt-Typen über die Punkt-Notation

```
SELECT Kundennummer, x.Person.Adresse.Strasse
FROM Kunde x
WHERE x.Person.Adresse.Ort like 'Do%'
```

Null-Werterhaltung in Pfadausdrücken:

x.Person IS.Adrresse IS NULL → x.Person.Adrresse.Ort IS NULL

Einsatz benutzerdefinierter Typen

Prof. Dr. I. M. Saatz

Moderne Datenbanken

FH Dortmund

Einsatzmöglichkeiten strukturierter Typen

1. als Attributtyp anderer strukturierter Typen

```
CREATE TYPE Person_Typ AS OBJECT(  
    PNR      Integer,  
    Name     VARCHAR(30),  
    Anschrift Address_Typ,  
    )
```

2. als Typ von Tabellenspalten in Tupeltabellen

```
CREATE TABLE person_tabelle(  
    stammdaten Person_typ,  
    bild       BLOB(1m)))
```

3. als Typ einer typisierten Tabelle (Objekttabelle)

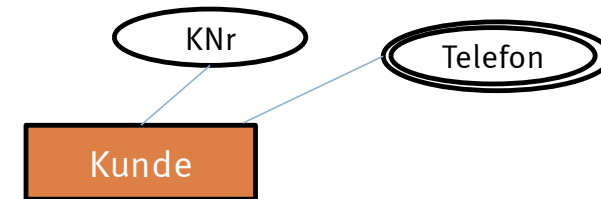
- Die Tabellenstruktur basiert auf einem abstrakten Datentyp.
- Einfügen von Objekten durch Konstruktoraufruf.
- Selektieren, Löschen, Ändern erfolgt wie bei „normalen“ Tabellen.

```
CREATE TABLE Person  
OF Person_Typ
```

4. als Parametertyp von Methoden, Funktionen und Prozeduren

5. als Typ von SQL-Variablen (in PL/SQL)

1	Objekt-Relationale Datenbankerweiterung	2
2	Anwendung der NF2-Erweiterung	10
2.1	Strukturierte Attribute	12
2.2	Mehrwertige Attribute	18
3	Anwendung objektorientierter Konzepte	33
3.1	Objekte identifizieren und referenzieren	34
3.2	Assoziationen	40
3.3	Aggregation	45
3.4	Vererbung	50
4	Fazit	56



■ Mehrwertige Attribute (→ Telefone)

- Feste Obergrenze
 - Konzept: Variable Arrays
- Variable Anzahl
 - Liste auf eigene Tabelle abbilden
Konzept: Nested Tables

KNr	...	Telefone		
		[0]	[1]	[2]
123	...	0231	0173	NULL

KNr	...	Telefone	
		TID	Tel
123		1	0231
		2	0173

- Mehrwertige Attribute **mit** fester Obergrenze (**Variable Arrays**)

- geordnete Multimenge mit indiziertem Zugriff
- Größe wird mit der Instanziierung festgelegt
- alle Elemente besitzen denselben Typ
- Elementindex als Iterator

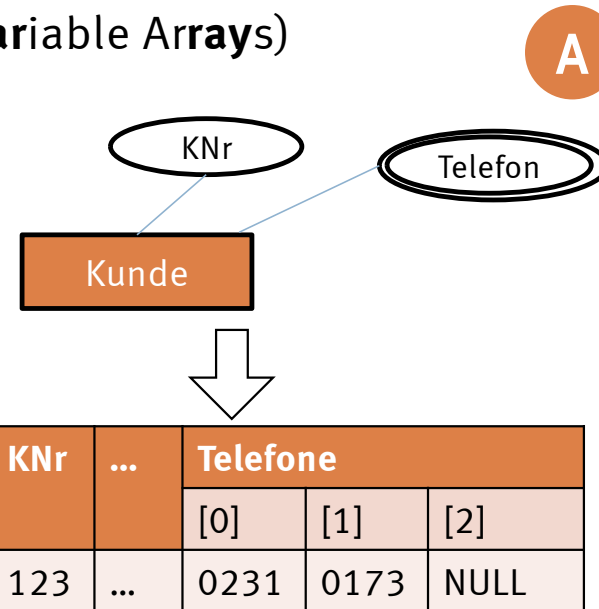
- Syntax

```
CREATE <arraytyp_name>  
  AS VARRAY (<arraygroesse>  
  OF <elementtyp>
```

- Beispiel:

```
CREATE TYPE Telefon_Typ  
  AS VARRAY(3)  
  OF VARCHAR2(15);
```

- VARRAY-Werte sind nicht direkt in SQL, sondern nur in PL/SQL nutzbar und werden nicht in einer separaten Tabelle abgespeichert.



■ Mehrwertige Attribute **mit** fester Obergrenze (**Variable Arrays**)

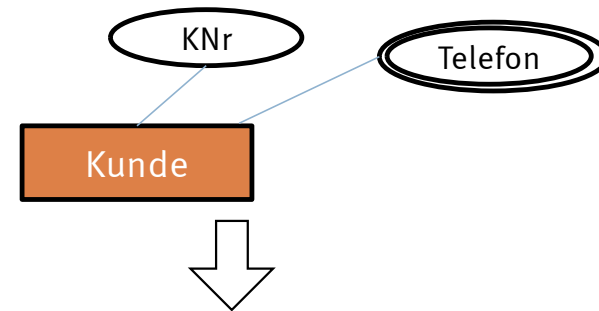
A

• Definition **Typdeklaration**

```
CREATE TYPE Telefon_Typ AS VARRAY(3)  
OF VARCHAR2(15);
```

• Verwendung

```
CREATE TABLE Kunde  
( KNr      Integer, ...  
  Telefone Telefon_Typ, ...  
)
```



KNr	...	Telefone		
		[0]	[1]	[2]
123	...	0231	0173	NULL

• Werte einfügen

```
INSERT INTO Kunde VALUES  
( '123', ... , Telefon_Typ('0231', '0173'))
```

Konstruktoraufruf !

• Selektion

```
SELECT Telefone FROM Kunde;
```

```
SELECT a.KNr, b.*  
FROM Kunde a, TABLE(a.Telefone) b;
```

A

1

```
SELECT Telefone FROM Kunde;
```

	TELEFONE
1	VARCHAR(0231,0173)

Anzahl: 2 Telefone

Telefone-Objekt

Anzahl: 3 Telefone

Null-Werte werden mitgezählt

Anm.: Null-Werte in einem VarArray ('0231...', NULL, '0173...') werden mitgezählt

2

```
SELECT a.KNr, b.*  
FROM Kunde a, TABLE(a.Telefone) b;
```

Oracle-Funktion

	 KNR	 COLUMN_VALUE
1	123 0231	
2	123 0173	

Die Table-Funktion kennzeichnet Iterationsbereiche, hier die zum Kunden a gehörenden eingebettete Telefonliste.

Eingebettete Tabellen - Nested Tables

B

- Mehrwertige Attribute **ohne** feste Obergrenze
Eigenschaften:
 - geordnete Multimenge mit wertbezogenem Zugriff
 - ohne Größenbeschränkungen
 - alle Elemente besitzen denselben Typ
 - Besitzt einen benannten Konstruktor (hier: Telefone_NT)
 - Auf die eingeschachtelte Tabelle kann ein Index definiert werden.

- Syntax

```
CREATE TYPE <tabellentyp_name>
AS TABLE OF <elementtyp>
```

- Beispiel

```
CREATE TYPE Telephone_NT
AS TABLE OF Telefon_TYP;
```

Telephone_NT	
TID	Tel
1	0231
2	0173

Telephone_NT

Telefon_Typ

■ Mehrwertige Attribute **ohne** feste Obergrenze

• Definition

```
CREATE TYPE Telefon_TYP AS OBJECT(  
    TID integer,  
    Tel varchar(15)  
);
```

```
CREATE TYPE Telefon_NT  
    AS TABLE OF Telefon_TYP;
```

• Verwendung

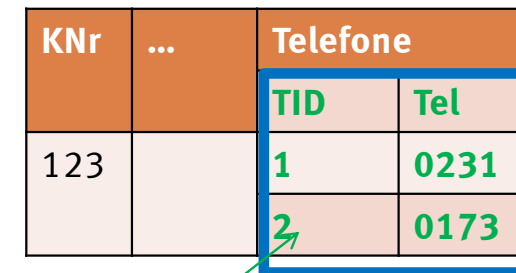
```
CREATE TABLE Kunden  
( KNr      INTEGER PRIMARY KEY,  
  Telefone  Telefon_NT  
)  
    NESTED TABLE Telefone  
    STORE AS Telefone_NT_TAB;
```

Anlegen der Haupttabelle Kunden
mit verdeckten Verweis
auf die eingebettete Tabelle

Eingebettete Tabelle Telefone_NT_TAB
wird eigenständig gespeichert
(weitere Store-Klauseln durch Komma trennen)

• Werte einfügen

```
INSERT INTO Kunden VALUES (  
    123, Telefon_NT (Telefon_typ(1,'0231'),Telefon_typ(2, '0173')));
```



KNr	...	Telefone	
		TID	Tel
123		1	0231
		2	0173

Telefon_Typ

Telefon_NT

B

Asymmetrischer Zugriff

Prof. Dr. I. M. Saatz

Moderne Datenbanken

FH Dortmund

DATA1	DATA2	DATA3	DATA4	NT_DATA
...	...	...	...	A
...	...	...	...	B
...	...	...	...	C
...	...	...	...	D
...	...	...	...	E



B

Storage Table

	NESTED_TABLE_ID	Values
Storage for nested table A	A	A11
	A	A12
	A	A13
Storage for nested table B	B	B21
	B	B22
	B	B23
	B	B24
Storage for nested table C	C	C31
	C	C32
Storage for nested table D	D	D41
	E	E51
Storage for nested table E	E	E52
	E	E53
	E	E54

http://docs.oracle.com/cd/B28359_01/appdev.111/b28371/adobjdes.htm#i446399

Nested Table - Integritätsbedingungen

```
CREATE TYPE Telefon_TYP AS OBJECT(  
    TID integer,  
    Tel varchar(15)  
);
```

```
CREATE TYPE Telephone_NT  
    AS TABLE OF Telefon_TYP;
```

KNr	...	Telefone	
123		TID	Tel
		1	0231
123		2	0173

Telefon_Typ

Telephone_NT

Integritätsbedingung auf der eingeschachtelten Tabelle

```
CREATE TABLE Kunden  
( KNr          INTEGER PRIMARY KEY,  
  Telefone     Telephone_NT  
)  
NESTED TABLE Telefone  
    STORE AS Telefone_NT_TAB (TID PRIMARY KEY);
```

Integritätsbedingung



Eingebettete Tabellen - Select

Prof. Dr. I. M. Saatz

Moderne Datenbanken

FH Dortmund

Folge des asymmetrischen Zugriffs:

Eine geschachtelte Tabelle ist z.B. ohne Basistabelle nicht zugreifbar, auch wenn sie eigenständig gespeichert wird.

Mit dem Table-Operator kann eine (flache) Tabelle von Telefon-Einträgen erzeugt werden:

```
SELECT KNr, n.TID, n.Tel
FROM Kunde, TABLE(Telefone) n;
```

	 KNr	 TID	 TEL
1	123	1	0231
2	123	2	0173

Werte in eingebettete Tabellen einfügen

KNr	Nachname	Telefone	
		TID	Tel
2310	Meitner	1	0231
		2	0173
		3	12345



B

Tupel in eine **innere Tabelle** einfügen

- Struktur:

```
INSERT INTO TABLE (<anfrage>)  
VALUES (...)
```

- Beispiel:

```
INSERT INTO  
TABLE( SELECT  telefone  
        FROM    Kunde  
        WHERE   Nachname= 'Meitner')  
VALUES(3, '12345');
```


Jede Tabellenspalte ist typisiert. Die Angabe der Typen einer Select-Anfrage erfolgt mit der Multiset-Funktion, da diese nicht automatisch ermittelt werden.

B

- Grundmuster

```
INSERT INTO <tabelle> (<kollektionsattribute>
VALUES ([CAST(MULTISET)](<anfrage>
      [AS <tabellentypkonstruktor>]
      )
```

- Beispiel: Zuordnung der Telefonnummern eines Kunden (Curie) zu einem anderen Kunden (Meitner)

```
INSERT INTO Kunde (Nachname, Telefone)
VALUES ('Meitner',
      (CAST(MULTISET( SELECT VALUE(t)
                      FROM Kunden ,TABLE(telefone) t
                      WHERE Nachname = 'Curie' )
      AS Telefon_Typ)
      )
```

Anm.: Der VALUE-Operator ermöglicht den Zugriff auf das ganze Objekt

Änderung innerer Tabellen

KNr	Nachname	Telefone
2310	Meitner	0231
		0173
		12345

KNr	Nachname	Telefone	
		TID	Tel
2310	Meitner	1	0231
		2	0173
		3	12345

B

■ Tupel einer inneren Tabelle ändern

• Struktur:

```
UPDATE TABLE (<anfrage>) SET ( ... )
```

• Beispiel:

(Telefone hier mit einem Attribut bzw. strukturiertem Attribut)

```
UPDATE
  TABLE( SELECT telefone
           FROM   Kunde WHERE Nachname = 'Meitner') h
```

```
SET VALUE(h) = '33333'
WHERE VALUE(h) = '12345';
```

Telefonliste (Nr)

```
SET h.TEL= '33333'
WHERE h.TID= 1;
```

Telefone (TID, Nr)

KNr	Nachname	Telefone
2310	Meitner	0231
		0173
		12345



- Tupel aus einer inneren Tabelle löschen

- Struktur:

```
DELETE FROM TABLE (<anfrage>) WHERE ( . . . )
```

- Beispiel:

```
DELETE FROM  
  TABLE( SELECT telefone  
           FROM   Kunde WHERE Nachname = 'Meitner') h  
WHERE VALUE(h) = '12345';
```

Vorsicht Falle!

Objekte in Tabellen dürfen nicht NULL werden, da sie sonst gegen die Tabellenstruktur verstoßen. Einzelne Einträge können nur eine leeren inneren Tabelle eingefügt werden.

B

KNr	Nachname	Telefone	
		TID	Tel
2310	Meitner	NULL	NULL



KNr	Nachname	Telefone	
		TID	Tel
2310	Meitner	NULL	



Liefere alle Kunden mit NULL als Tel-Attributwert
(mit Hilfe des Bezeichners COLUMN_VALUE)

```
SELECT k.Knr
FROM Kunde k, TABLE(k.Telefone) t
WHERE t.Tel.COLUMN_VALUE IS NULL
```

1	Objekt-Relationale Datenbankerweiterung	2
2	Anwendung der NF2-Erweiterung	10
2.1	Strukturierte Attribute	12
2.2	Mehrwertige Attribute	18
3	Anwendung objektorientierter Konzepte	33
3.1	Objekte identifizieren und referenzieren	34
3.2	Assoziationen	40
3.3	Aggregation	45
3.4	Vererbung	50
4	Fazit	56

1	Objekt-Relationale Datenbankerweiterung	2
2	Anwendung der NF2-Erweiterung	10
2.1	Strukturierte Attribute	12
2.2	Mehrwertige Attribute	18
3	Anwendung objektorientierter Konzepte	33
3.1	Objekte identifizieren und referenzieren	34
3.2	Assoziationen	40
3.3	Aggregation	45
3.4	Vererbung	50
4	Fazit	56

Relationales Modell

- **Tupel** werden durch Schlüsselattribute identifiziert
- Schlüsselattribute müssen **explizit** verwaltet werden
- Der Schlüsselwert ist **mehrdeutig**
 - Beispiel:
Kunde 123
Artikel 123

Objektorientiertes Modell

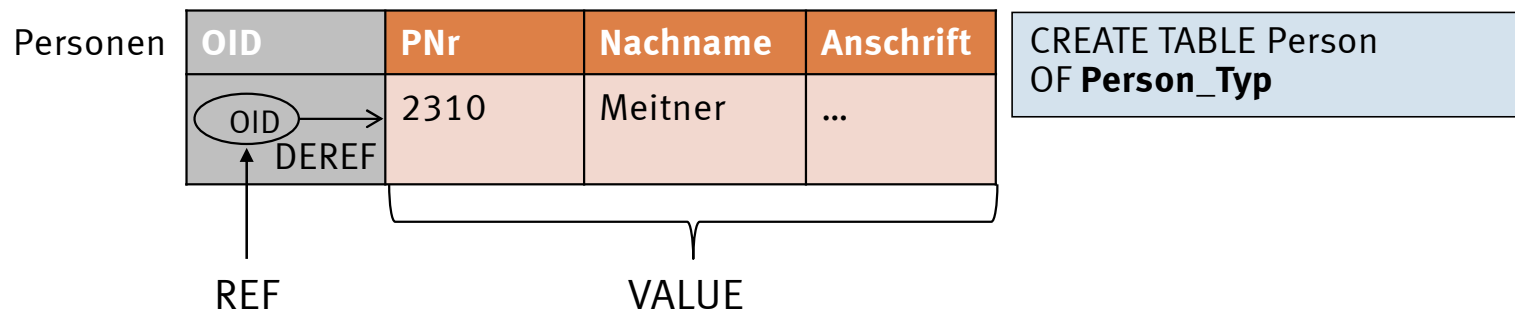
- **Objekte** werden über Objektidentitäten referenziert
- Jedes Objekt besitzt **implizit** eine Objektidentität (*object identifier*, OID)
- Die Objektidentität ist **eindeutig**
 - Beispiel:
OID von Kunde ≠ OID von Artikel

Realisierung von eindeutigen Objektreferenzen

?

- Jedes **Tupel** einer Objekttabelle ist ein eigenständiges Objekt
- Jedes Objekt einer Objekttabelle erhält eine **eindeutige OID** (durch DBMS automatisch gepflegt)
- ➔ Die Tupel einer Objekttabelle können von anderen Objekten innerhalb der Datenbank **referenziert** werden

- Verwaltung von OIDs
 - Ziel: Eindeutigkeit der OIDs (Es wird ein 16 Zeichen langer Schlüssel aufgebaut)
 - OID wird automatisch durch das DBMS vergeben
 - OID wird in einer verborgenen Tabellenspalte gespeichert (alte OIDs werden nicht wiederverwendet)



Eine Objekttabelle ist eine typisierte Tabelle, deren Typ durch strukturierten Typ festgelegt wird.

```
CREATE TABLE Person  
OF Person_Typ
```

■ Syntax

```
create table <tabellen_name>  
  of <objektyp-name>  
  [<substitutions-klausel>  
  [(<integritätsliste>)]  
  [<oid-klausel>]  
  [<store-klausel>]  
  [<attributsubstitutionsklauseln>]
```

Objektyp, den die Tabelle enthält

Statische Integritätsbedingungen

Je eine STORE-Klausel für
tabellenwertige Attribute

■ Eigenschaften:

- Spalten = Attribute des strukturierten Typs, erste Spalte: OID !
- Zeilen = Objekte des strukturierten Typs
- Zeilen einer Objekttabelle sind über OID referenzierbar
- Auf Zeilen sind Methoden aufrufbar
- OID-Klausel legt Art der Referenzgenerierung fest
 - OID wird vom System generiert oder vom Primärschlüssel abgeleitet. Eingebettete Objekte (nested tables) haben keine OID.

```
object identifier is {system generated | primary key }
```

- Zugriff auf die Objekt-ID (OID) eines Objektes
 - Die **REF**-Funktion liefert eine **Referenz** auf eine Instanz eines Objekt-Typs
 - Wert eines Referenztyps ist eine OID

```
CREATE TABLE Personen OF Person_Type;  
INSERT INTO Personen VALUES (1, 'Mama', NULL);  
SELECT REF(p) FROM Personen p;
```

REF(P)

1 oracle.sql.REF@8c6142e8

- REF benötigt Tabellen-Alias (hier p)
- die referenzierte Objekttable wird in der Anfrage nicht erwähnt, der Name muss nicht bekannt sein

- Zugriff auf das Objekt über die OID (**Dereferenzieren**):
 - Funktion **DEREF(OID)**, Kurzform: **VALUE()**, liefert das **Objekt**

```
SELECT p.Nachname FROM Personen p;  
SELECT DEREF(REF(p)).Nachname FROM Personen p;  
SELECT VALUE(p).Nachname FROM Personen p;
```

DEREF(REF(P))

SAATZ.PERSON_TYP(2310, 'Meitner', 'Frau')
SAATZ.PERSON_TYP(8345, 'Einstein', 'Herr')

Anmerkung

REF ist der einzige unbenannte Konstruktor in ORACLE (es gibt also kein CREATE TYPE ... AS REF !)

- Tabellendefinition mit Objektreferenz:

```
CREATE TYPE Person_Type AS OBJECT(  
  PID    INTEGER,  
  Nachname VARCHAR(30),  
  Mutter REF Person_Type  
)/  
CREATE TABLE Personen OF Person_Type;
```

- Objekt referenzieren:
 - Impliziter Join wird ausgeführt

```
INSERT INTO Personen  
VALUES (1, 'Kind', (Select REF(p) FROM Personen p));
```

where ~

- Zugriff auf Attributwerte des referenzierten Objekts:

```
SELECT Deref(p.Mutter).Nachname  
FROM Personen p  
WHERE Nachname='Kind';
```

- Achtung:
Das gesamte referenzierte Objekt wird zurückgeliefert

	DEREF(P.MUTTER).NACHNAME
1	Mama

1	Objekt-Relationale Datenbankerweiterung	2
2	Anwendung der NF2-Erweiterung	10
2.1	Strukturierte Attribute	12
2.2	Mehrwertige Attribute	18
3	Anwendung objektorientierter Konzepte	33
3.1	Objekte identifizieren und referenzieren	34
3.2	Assoziationen	40
3.3	Aggregation	45
3.4	Vererbung	50
4	Fazit	56

1:1-Beziehungen

Beziehung (binär)



1:1-Beziehung:

Erweitern des Objekttyps A um eine einwertige Referenz auf Objekttyp B (oder umgekehrt)
→ keine Abhängigkeit von Beziehungsdichte

Beispiel (Unidirektionale Assoziation):

A CREATE TYPE **Mutter_Typ** AS OBJECT
(Pnr Integer,
Nachname VARCHAR(30),
Partner REF Vater_Typ
)

CREATE TYPE **Vater_Typ** AS OBJECT
(Pnr Integer,
Nachname VARCHAR(30)
) **B**

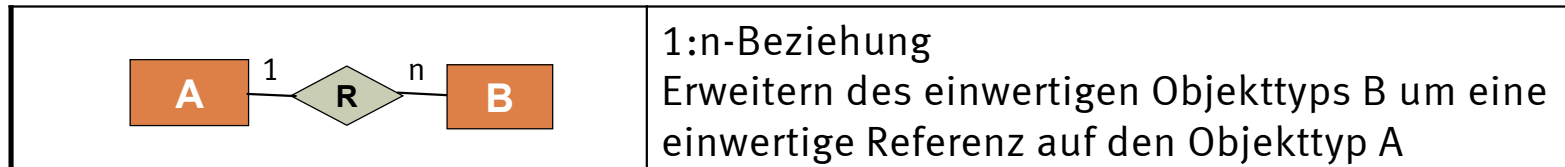
Oder Alternativ:

CREATE TYPE **Mutter_Typ** AS OBJECT
(Pnr Integer,
Nachname VARCHAR(30)
)

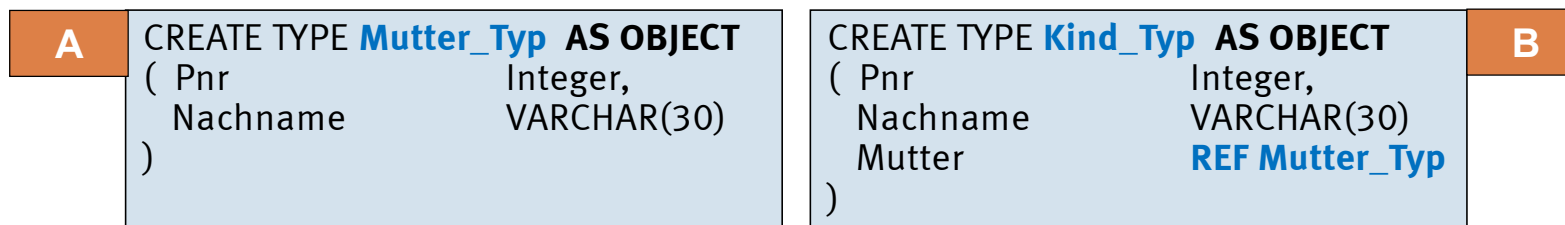
CREATE TYPE **Vater_Typ** AS OBJECT
(Pnr Integer,
Nachname VARCHAR(30),
Partner REF Mutter_Typ
)

Zur Abbildung von bidirektionalen Assoziationen werden beide Typen um eine Referenz erweitert.
Dann sind zusätzlich Trigger erforderlich, um Inkonsistenzen zu vermeiden.

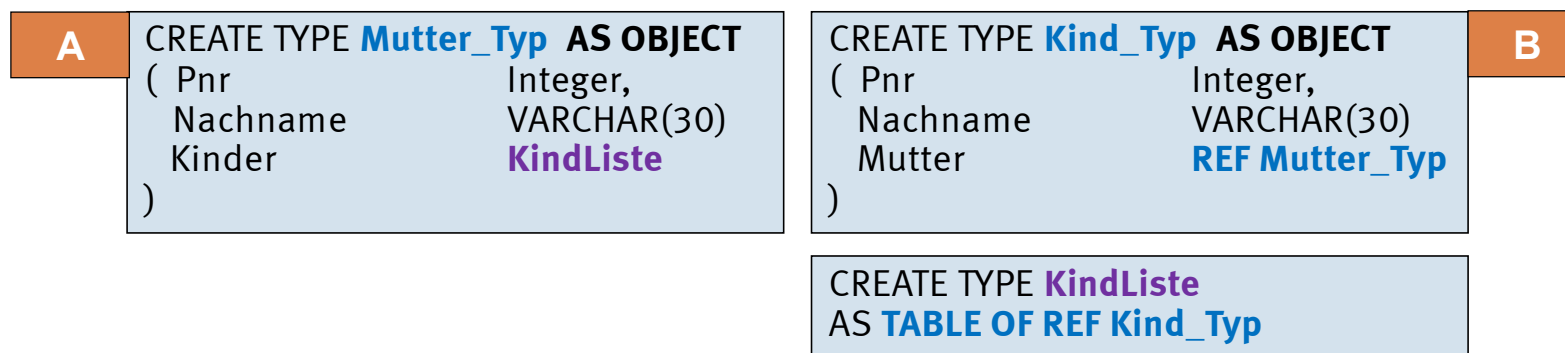
1:N-Beziehungen



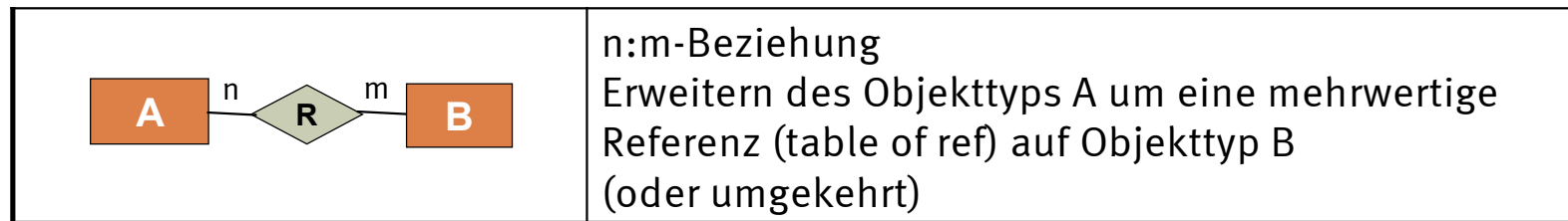
Beispiel (unidirektionale Assoziation):



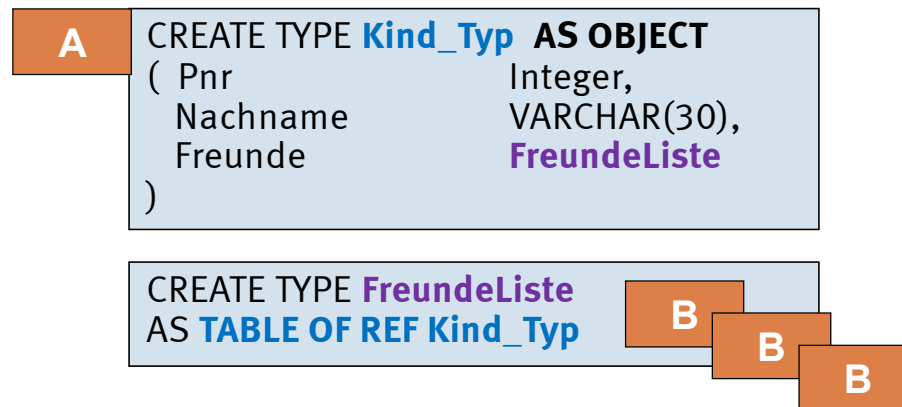
Beispiel (bidirektionale Assoziation):



N:M-Beziehungen



Beispiel:

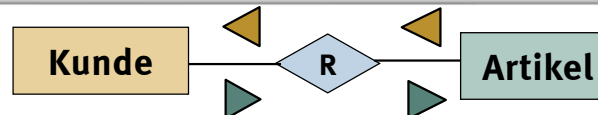


PNr	Nachname	Freunde
		OID[=PID]
1	Hans	2
		3
2	Anton	1
		3
3	Maria	1
		2

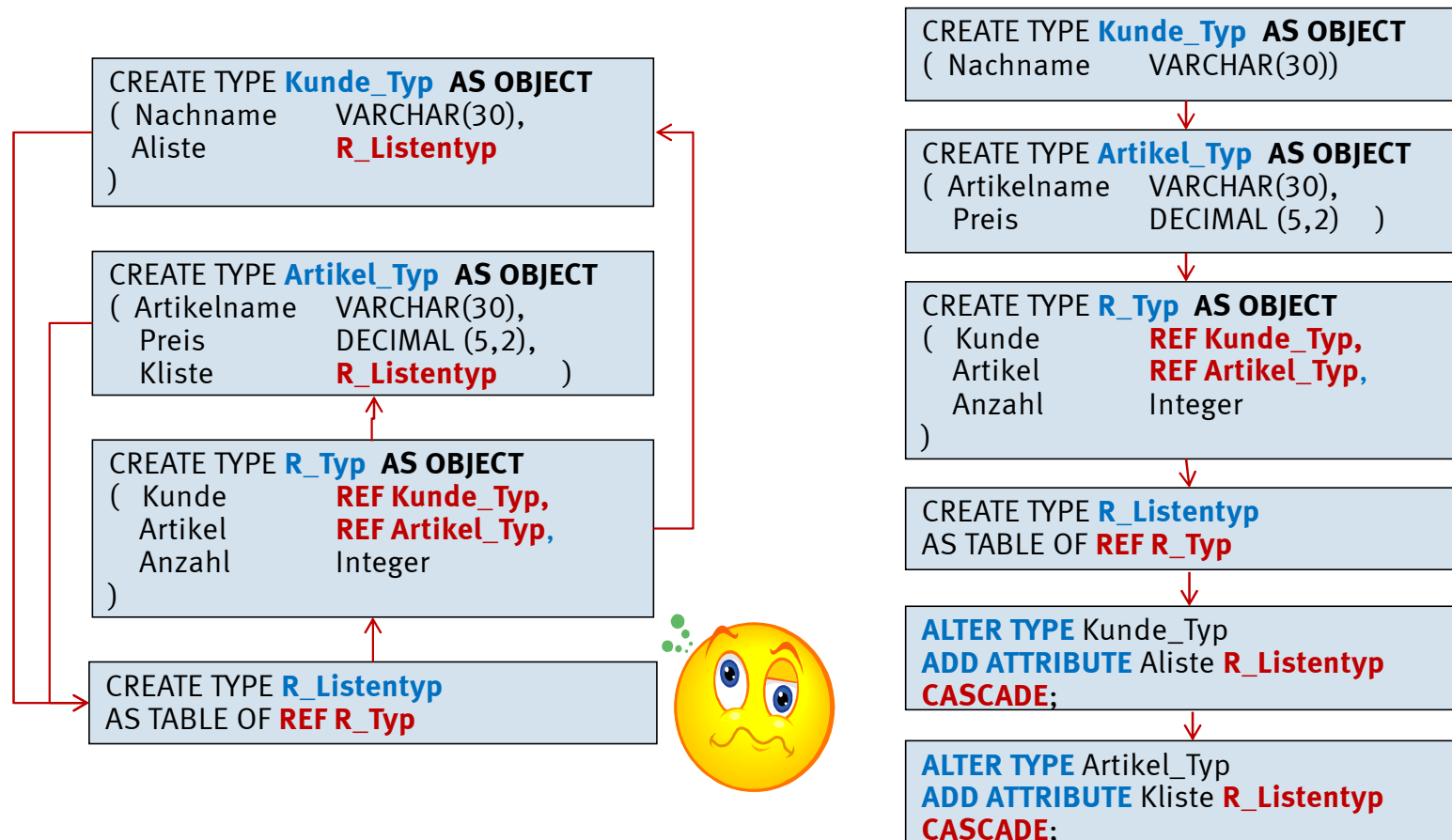
Nested Table !

Typdefinitionen bei zyklischen Referenzen

Bidirektionaler Zugriff:



3



Klauseln: CASCADE (propagieren der Änderung) oder INVALIDATE (ohne Validierung der Typänderung)

1	Objekt-Relationale Datenbankerweiterung	2
2	Anwendung der NF2-Erweiterung	10
2.1	Strukturierte Attribute	12
2.2	Mehrwertige Attribute	18
3	Anwendung objektorientierter Konzepte	33
3.1	Objekte identifizieren und referenzieren	34
3.2	Assoziationen	40
3.3	Aggregation	45
3.4	Vererbung	50
4	Fazit	56

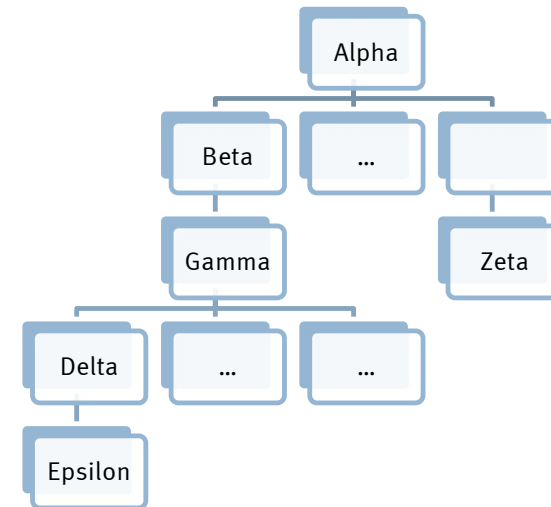
Die **Aggregation** wird objektrelational über Objektreferenzen umgesetzt.

Typdefinitionen:

```
CREATE TYPE Mitarbeiter_Typ (  
  PNR          INTEGER,  
  nachname     VARCHAR(30),  
  telefon      VARCHAR(10),  
  raum         VARCHAR(10),  
  vorgesetzter REF Mitarbeiter_typ  
)
```

Objekttabellen:

```
CREATE TABLE Person OF Mitarbeiter_Typ(  
  PNR PRIMARY Key  
)
```



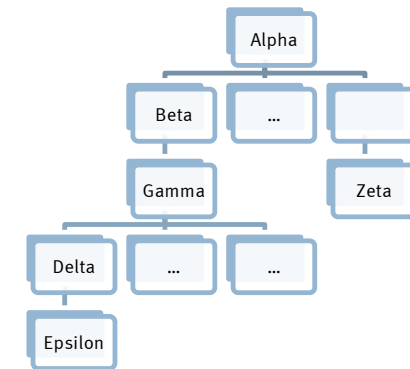
Liste der Namen der Mitarbeiter und des zugehörigen Vorgesetzten:

```
select Nachname, Deref(vorgesetzter).Nachname from Mitarbeiter
```

Skriptausgabe x Abfrageergebnis x

SQL | Alle Zeilen abgerufen: 5 in 0 Sekunden

	NACHNAME	Deref(VORGESETZTER).NACHNAME
1	alpha	(null)
2	beta	alpha
3	gamma	beta
4	delta	gamma
5	epsilon	delta



Liefere den „big boss“, der keinen Vorgesetzten besitzt:

```
select Nachname, Deref(vorgesetzter).Nachname from Mitarbeiter  
where vorgeseztter IS NULL
```

Skriptausgabe x Abfrageer... x

SQL | Alle Zeilen abgerufen: 1 in 0 Sekunden

	NACHNAME	Deref(VORGESETZTER).NACHNAME
1	alpha	(null)

„Dangling References“

Durch eine „Dangling-Referenz“ wird eine **nicht** vorhandene OID referenziert.

- Beispiel:

Löschen eines Objekts, welches noch referenziert wird

```
DELETE FROM Mitarbeiter WHERE Person.Nachname='beta'
```

Folge: Die Referenz zeigt danach ins Leere („dangling-Referenz“)



Keine
referentielle Integrität

<pre>select Nachname, Deref(vorgesetzter).Nachname from Mitarbeiter;</pre>	
Skriptausgabe x Abfrageergebnis x	
Alle Zeilen abgerufen: 4 in 0,062 Sekunden	
NACHNAME	Deref(VORGESETZTER).NACHNAME
1 alpha	(null)
2 gamma	(null)
3 delta	gamma
4 epsilon	delta

<pre>select Nachname, Deref(vorgesetzter).Nachname from Mitarbeiter where vorgesetzter is null;</pre>	
Skriptausgabe x Abfrageer... x	
Alle Zeilen abgerufen: 1 in 0 Sekunden	
NACHNAME	Deref(VORGESETZTER).NACHNAME
1 alpha	(null)

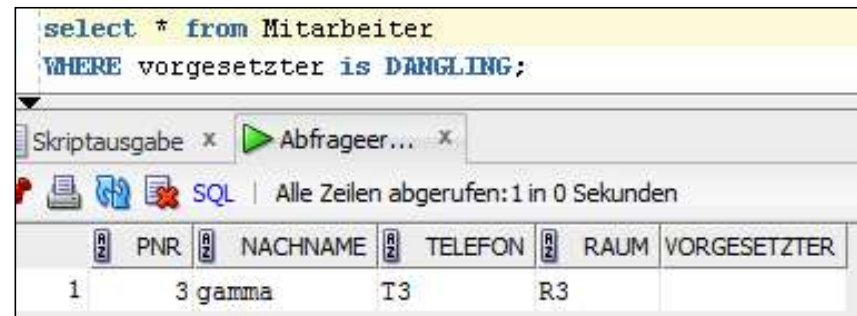
Referenz kann nicht aufgelöst werden

Anmerkung:

Problem wird **nicht** gelöst durch das Wiedereinfügen des Mitarbeiters beta, d.h. durch Einfügen eines neuen Datensatzes mit gleichem Inhalt, da dieser dann eine neue OID erhält.

Erkennen von „Dangling References“

- Ermittlung der fehlerhaften Referenzen mit dem Schlüsselworte Dangling:



The screenshot shows a database query tool interface. At the top, a SQL query is entered: `select * from Mitarbeiter WHERE vorgesetzter is Dangling;`. Below the query, there is a status bar indicating 'Skriptausgabe x' and 'Abfrageer... x'. A message states 'Alle Zeilen abgerufen: 1 in 0 Sekunden'. Below this, a table of results is displayed with columns: PNR, NACHNAME, TELEFON, RAUM, and VORGESETZTER. The first row shows the value '1' in the PNR column and '3 gamma' in the NACHNAME column, with other columns being empty.

PNR	NACHNAME	TELEFON	RAUM	VORGESETZTER
1	3 gamma	T3	R3	

- Korrektur der Dangling-Referenz, bspw. durch Setzen auf NULL

```
UPDATE Mitarbeiter SET vorgesetzter=NULL  
WHERE vorgesetzter IS Dangling
```

- Folge: Gamma ist nun auch ein „big boss“
- Alternativ: Zuordnung eines anderen Mitarbeiterobjekts zu Gamma als Vorgesetzten

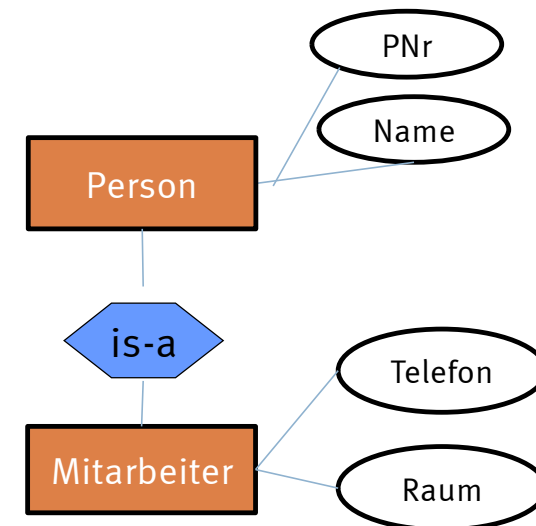
1	Objekt-Relationale Datenbankerweiterung	2
2	Anwendung der NF2-Erweiterung	10
2.1	Strukturierte Attribute	12
2.2	Mehrwertige Attribute	18
3	Anwendung objektorientierter Konzepte	33
3.1	Objekte identifizieren und referenzieren	34
3.2	Assoziationen	40
3.3	Aggregation	45
3.4	Vererbung	50
4	Fazit	56

- Subtypen (→ Generalisierung und Spezialisierung)
 - Subtyp erbt alle Attribute und Methoden des Supertyps (nur Einfachvererbung)
 - Geerbte Methoden sind überschreibbar (NOT FINAL)
- Beispiel
 - Supertyp (not final: Vererbung möglich)

```
CREATE TYPE Person_Typ AS OBJECT  
( PNr          CHAR(5),  
  Name         VARCHAR(50)  
) NOT FINAL;
```

- Subtyp

```
CREATE TYPE Mitarbeiter_Typ  
UNDER Person_Typ  
(  
  Telefon      CHAR(7),  
  Raum         VARCHAR(5) }  Attribut- und  
                           Methodendeklaration  
)
```



Aufbau von Typhierarchien

- Definition mittels UNDER-Klausel
 - Subtyp erbt alle Attribute und Methoden des Supertyps.
 - Supertyp muss selbst ein strukturierter Typ sein.
 - Subtyp darf maximal einen direkten Supertyp haben.
 - keine (direkte) Mehrfachvererbung möglich.
 - Subtyp kann geerbte Methoden überschreiben und überladen.
- Strukturierte Typen, die keine Subtypen sind, heißen Wurzeltypen

```
CREATE TYPE Person_Typ AS OBJECT  
( PNr          CHAR(5),  
  Name         VARCHAR(50)  
)NOT FINAL;
```

```
CREATE TYPE Mitarbeiter_Typ  
UNDER Person_Typ  
(  
  Telefon      CHAR(7),  
  Raum         VARCHAR(5)  
)
```


Beispiel Mitarbeiterhierarchie

Typdefinitionen:

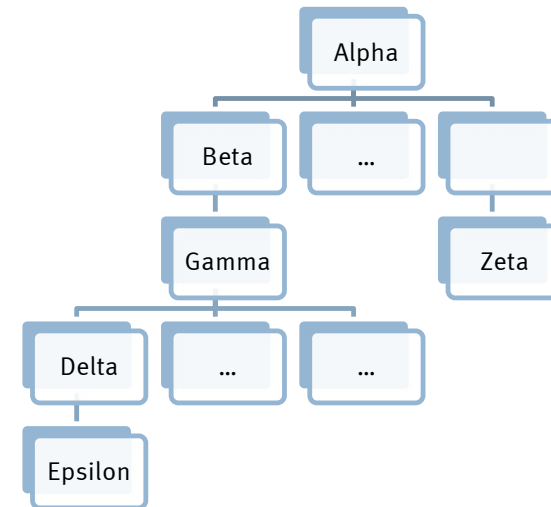
```
CREATE TYPE Person_Typ (  
    pnr          INTEGER,  
    nachname     VARCHAR(50)  
) NOT FINAL
```

```
CREATE TYPE Mitarbeiter_Typ UNDER Person_typ (  
    telefon      VARCHAR(10),  
    raum         VARCHAR(10),  
    vorgesetzter REF(Mitarbeiter_typ)  
)
```

Objekttabellen:

```
CREATE TABLE Person OF Person_Typ
```

```
CREATE TABLE Mitarbeiter OF Mitarbeiter_Typ(  
    PNR PRIMARY Key  
)
```



Substituierbarkeit: Eine Objekttabelle darf auch Instanzen eines Subtyps aufnehmen (Substituierbarkeit). Eine Instanz eines Subtyps ist überall verwendbar, wo der Supertyp erwartet wird.

Einfügen von Personen:

```
INSERT INTO Person  
VALUES (Mitarbeiter_Typ(2310, 'Meitner', 'T1', 'R1', null) )
```

Abfragen von Personen:

	PNR	NACHNAME
1	2310	Meitner

Abfragen des Mitarbeiter-Attributs Raum:

Verwendung des Subtyps (Treat() liefert NULL, falls Cast nicht erfolgreich)

```
SELECT TREAT(VALUE(p) AS Mitarbeiter_typ).Raum  
FROM Person p  
WHERE VALUE(p) IS OF TYPE (Mitarbeiter_Typ);
```

Instanz

instanceof

	(TREAT(VALUE(P)ASMITARBEITER_TYP)).RAUM
1	R1

Einfügen von Mitarbeitern:

```
INSERT INTO Mitarbeiter  
VALUES (Mitarbeiter_Typ(1, 'alpha', 'T1', 'R1', null) )
```

```
INSERT INTO Mitarbeiter VALUES  
(Mitarbeiter_Typ(2, 'beta', 'T2', 'R2',  
(SELECT REF(p) FROM Mitarbeiter p WHERE p.pnr=1)))
```

Abfragen von Mitarbeiter:

PNR	NACHNAME	TELEFON	RAUM	VORGESETZTER
1	1 alpha	T1	R1	{null}
2	2 beta	T2	R2	[SAATZ.MITARBEITER_TYP]

Abfrage von Person:

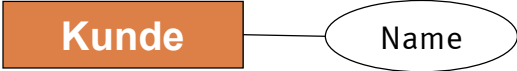
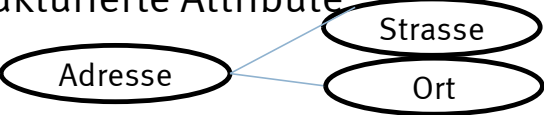
PNR	NACHNAME
1	2310 Meitner

Es erfolgt keine Subtabellen-
Bildung. Jeder Typ wird hier auf
eine eigene Tabelle abgebildet



1	Objekt-Relationale Datenbankerweiterung	2
2	Anwendung der NF2-Erweiterung	10
2.1	Strukturierte Attribute	12
2.2	Mehrwertige Attribute	18
3	Anwendung objektorientierter Konzepte	33
3.1	Objekte identifizieren und referenzieren	34
3.2	Assoziationen	40
3.3	Aggregation	45
3.4	Vererbung	50
4	Fazit	56

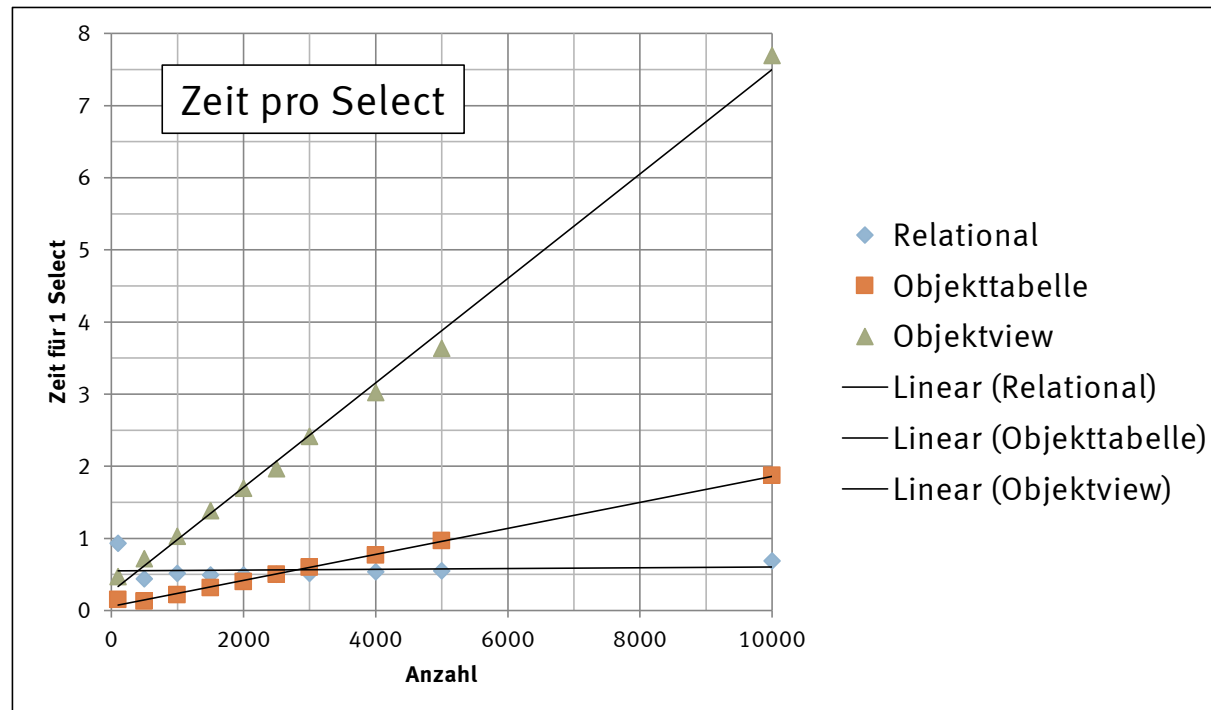
EER-Modell abbilden auf das OR-Datenbankmodell

Flache, einwertige Attribute 	Aus flachen, einwertigen Attributen werden Spalten der Relation mit flachen Typen
Strukturierte Attribute 	Für strukturierte Attribute wird ein strukturierter Objekttyp (Verbund der Basistypen) gebildet
Mehrwertiges Attribut 	- Feste Kardinalität (Obergrenze): VARARRAY - Variabler Kardinalität: Table-Objekttyp
Beziehung (binär) 	1:1-Beziehung: Erweitern des Objekttyps A um eine einwertige Referenz auf Objekttyp B (oder umgekehrt) → keine Abhängigkeit von Beziehungsdichte
	1:n-Beziehung Erweitern des einwertigen Objekttyps B um eine einwertige Referenz auf den Objekttyp A
	n:m-Beziehung Erweitern des Objekttyps A um eine mehrwertige Referenz (table of ref) auf Objekttyp B (oder umgekehrt)

EER-Modell und OR-Datenbankmodell

Aggregation (exklusiv)  <pre>graph LR; A[Fahrräder] --- B{Teil-von} --- C[Rahmen]</pre>	Abbildung über Objektreferenzen
Existenzabhängige Entitäten  <pre>graph LR; A[Gebäude] --- B{liegt in} --- C[Räume]</pre>	Eine starke Entität wird zu einer Objekttable, wenn diese komplexe Attribute enthält, ansonsten zu einer Tupel-Relation. Schwache Entitäten werden als komplexes Objekt (Typ-Struktur) in den Objekttyp der starken Entität eingebettet.
Generalisierung  <pre>graph LR; A[Personen] <-- B{is-a} <-- C[Kunden]</pre>	Vererbungsrelation über Vererbung von Objekttypen

Komplexitätsbetrachtung



Normiert über die Anzahl der durchgeführten Select-Anfragen ("Zeit für 1 Select") zeigt sich bei dem betrachteten Beispiel ein linearer Verlauf für den Zugriff über eine Objekttabelle und einem Objektview, während die Zugriffszeit über die relationale Tabelle (in diesem Bereich) in etwa konstant bleibt.

➔ Gesamtzugriffszeit (für den betrachteten Fall):

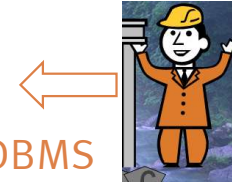
Relational Tabelle: $O(n)$

Objekttabelle, Objektview: $O(n^2)$

▪ Objektrelationale Datenbanken

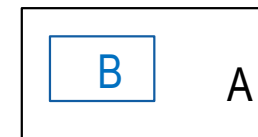
- Für große Datenmengen ist das Relationale Modell aus Gründen der Performance und der Maintenance zu bevorzugen.
- Verwendung von Objekttypen im Rahmen der Programmierung in PL/SQL.
- Die objekt-relationale Umsetzung unter Nutzung von Objekttabellen ist dann angebracht, wenn komplexe Objekte zu abzubilden sind, bspw. die Modellierung von Geoobjekten oder XML-Datentypen.

ORDBMS



• Beispiel:

- komplexe Struktur der Objekte
 - Methoden von geographischen Objekten
 - beinhaltet, überdeckt, liegt innerhalb, schneidet
 - Flächenberechnungen, Längen- und Umfangsberechnungen, Schwerpunktberechnung, Abstandsberechnung,
 - topographische und thematische Karten
 - Formulare
 - Text- und Bildobjekte



A beinhaltet B
A.contains(B)



Reflexion OR-Datenbankerweiterung

- 1) Wie wird die Objektidentität sichergestellt?
- 2) Wie werden Objekttypen definiert und genutzt?
- 3) Was ist eine Objekttabelle?
- 4) Wie können mehrwertige Attribute implementiert werden?
- 5) Wie können Objekte referenziert und Beziehungen umgesetzt werden?
- 6) Wie können Beziehungen zwischen Entitäten abgebildet werden?