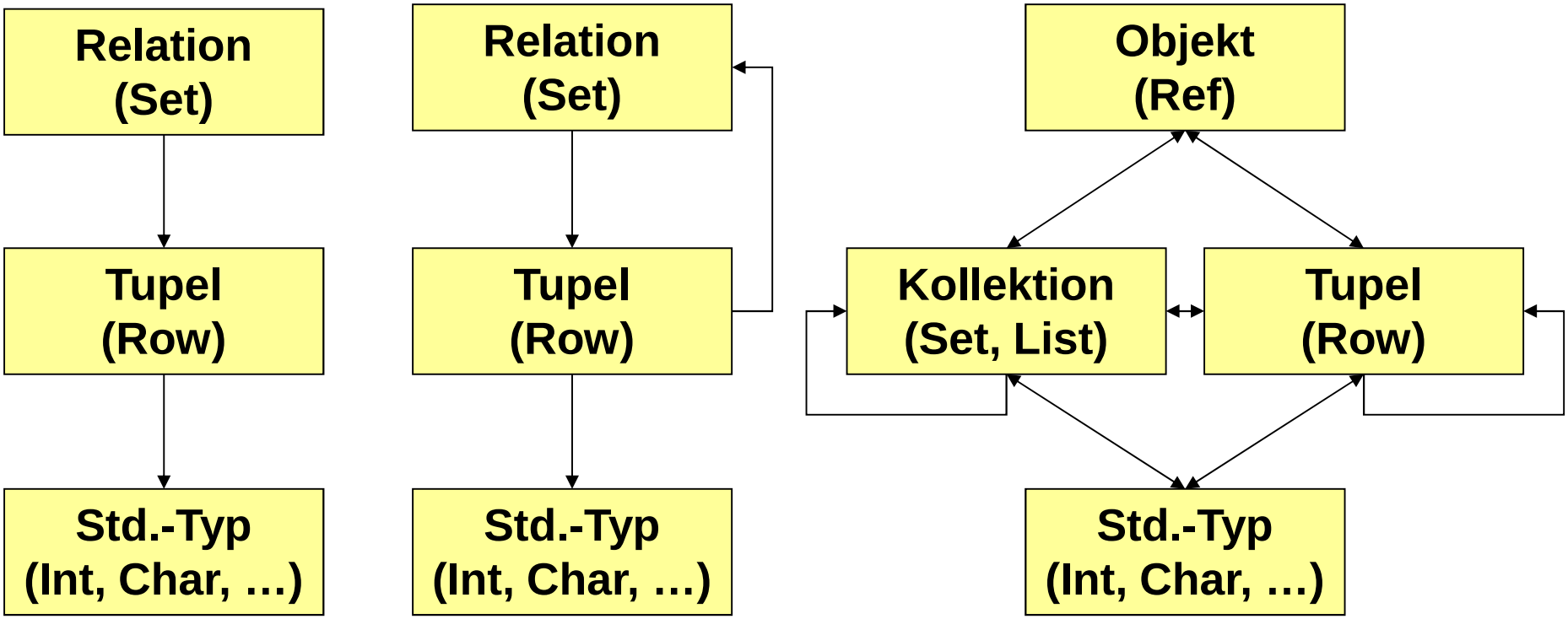
WP Datenbankdesign

Kapitel 6: Objektrelationale DBMS

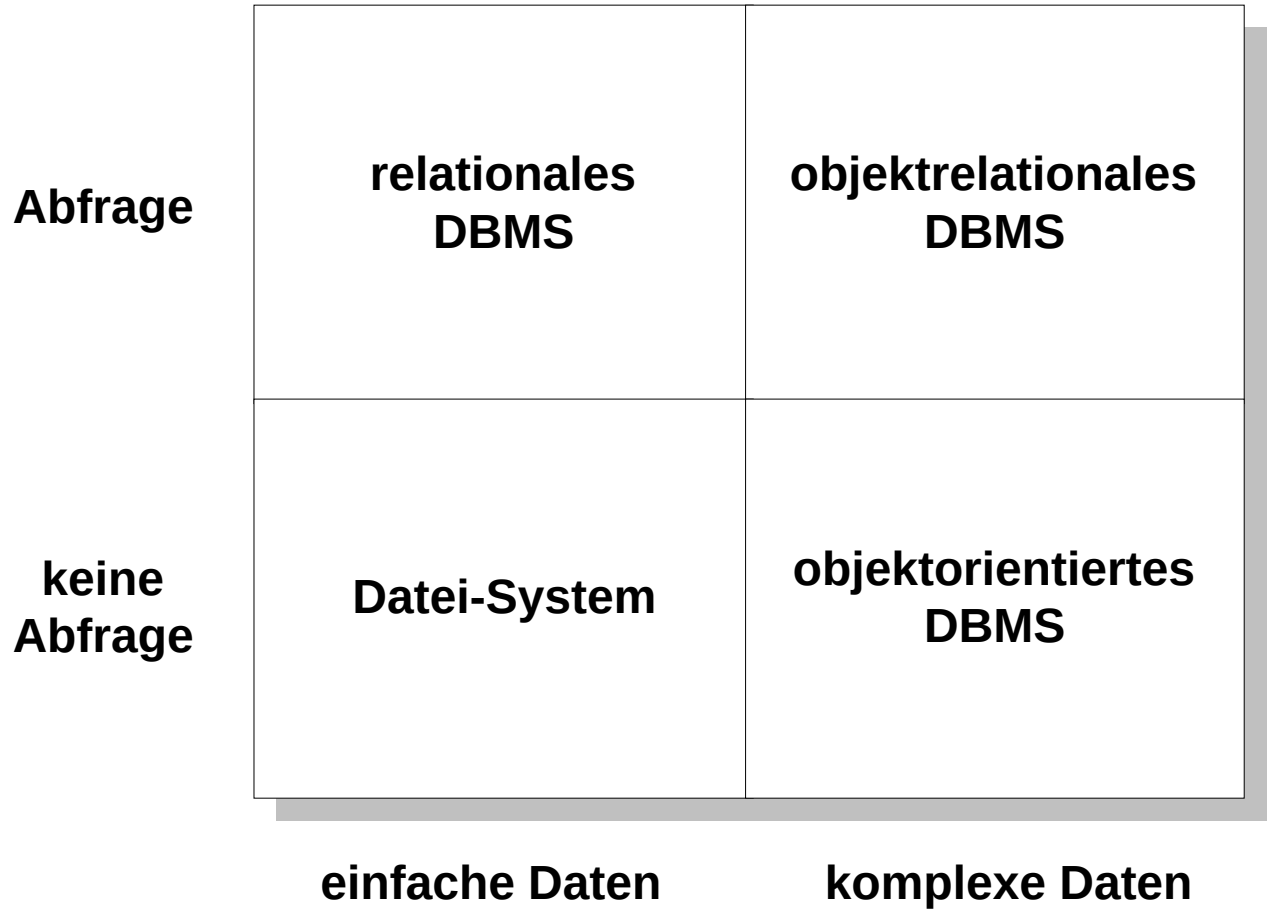
- Objektrelationale Datenbanken
- Object Types
- Beispiel: Object Types-Definition
- Object Tables mit Referenzen
- Collections (VARRAY, Nested table)
- Anwendungsmöglichkeiten von Nested tables

Relationale Datenbankmodelle



relationales Modell NF² objektrelationales Modell

Objektrelationale Datenbanken



Objektrelationale Datenbanken …

Eigenschaften

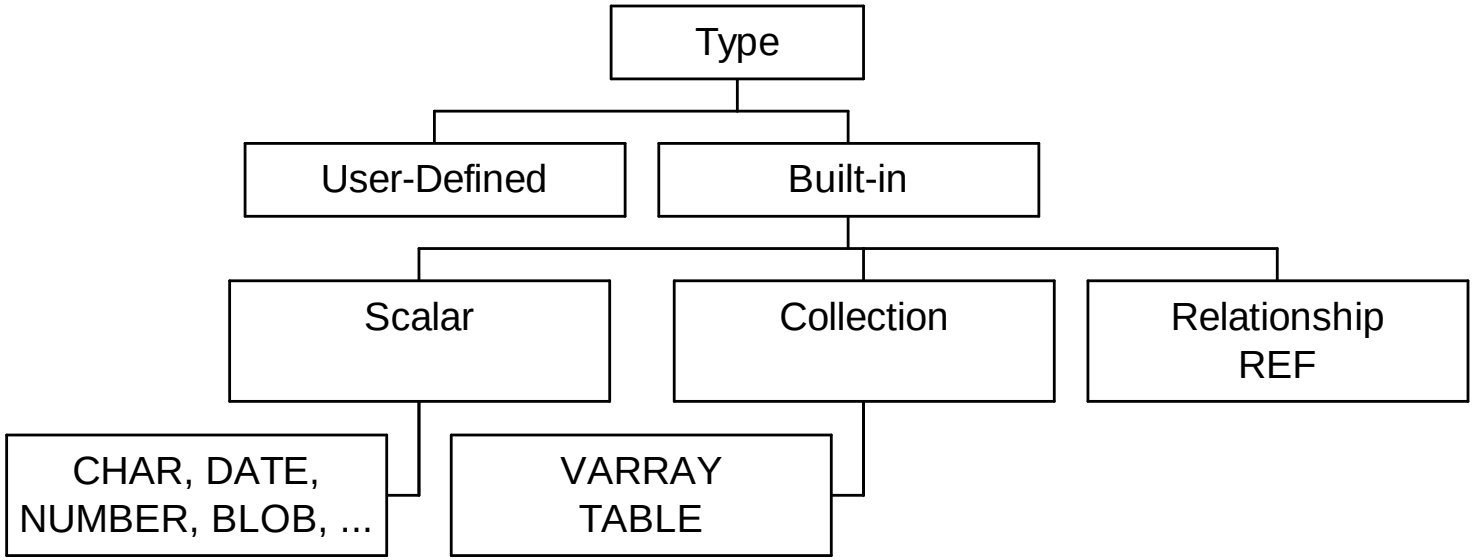
- Definition benutzerdefinierter Datentypen
- Unterstützung von Multimedia und LOBs (large data object)
- Kompatibilität zum objektorientierten SQL-Standard (SQL3)
- Unterstützung von VLDBs (very large database)
- Aufbauend auf RDBMS

Vorteile

- Modellierung von Objekten in der Datenbank
- Nicht nur flache Tabellen mit einfach strukturierten Attributen
- Reduzierung des „Impedance Mismatch“ zwischen Datenbank und Applikation
- Erzeugung wieder verwendbarer Objekte
- Abwärtskompatibilität

Object Types

Ein Object Type ist ein zusammengesetzter Datentyp definiert durch den Benutzer. Eine Object-Type Spezifikation muss alle Attribute und kann Methoden zur Implementierung des Verhaltens beinhalten. *Wichtig*: keine vollständige Kapselung, Zugriff auf Attribute möglich.



Mit Object Types kann man ...

- eine Tabelle mit einer Spalte als Object Type erzeugen
- einen anderen Object Type erzeugen (als Attribut)
- eine Object Table erzeugen

Relationale Tabellen mit einem Object Type

```
CREATE TYPE person_typ AS OBJECT
(           name    VARCHAR2(30),
            telefon  VARCHAR2(12),
            ort         VARCHAR2(25),
            ...      );

CREATE TABLE student
(           matrnr   NUMBER   PRIMARY KEY,
            person  person_typ);

INSERT INTO student VALUES (12345, person_typ(′Asterix′,
′0039014712′, ...) ;

SELECT s.person.ort FROM student s;
```

Member Methoden

```
Create Type Auftrag_Typ as Object
(       ANr             Integer,
        Datum           Date,
        RefLief         REF Lieferant_Typ,
        Menge           Integer.
        Preis           Real,
        Member Function Wert Return Real,
...     );
```

Deklaration

```
Create Type Body Aufrag_Typ as
Member Function Wert ( ) Return Real
is      W               Real  :=0;
Begin
   Select Menge*Preis into W  from Auftrag  a  where Value(a) = self;
   Return W;
End Wert;
...
End Auftrag_Typ;
```

Implementierung

Methoden erlauben Kapselung (encapsulation) der Attribute

Beispiel: Object Type Definition

```
CREATE OR REPLACE TYPE address AS OBJECT
( street            VARCHAR2(40),
  city              VARCHAR2(40),
  zip_code          VARCHAR2(7),
  MEMBER PROCEDURE ChangeAddress
( str IN VARCHAR2, cty IN VARCHAR2, zip IN VARCHAR2),
  MEMBER FUNCTION getCity RETURN VARCHAR2,
   ...
  MEMBER PROCEDURE setStreet (newStreet IN VARCHAR2)
);
```

Jeder Object-Typ hat einen Default-Konstruktor.

```
Address_var := address(‘ Av. Emilio Navarro‘,‘Coimbra‘,‘P-3000‘);
```

… und Methoden-Implementierung

```
CREATE OR REPLACE TYPE BODY address AS
  MEMBER PROCEDURE ChangeAddress
( str IN VARCHAR2, cty IN VARCHAR2, zip IN VARCHAR2) IS
  BEGIN          Street:=LTRIM(UPPER(str);
                 City:=LTRIM(UPPER(cty);
                 Zip_Code:=zip;
  END;
  MEMBER FUNCTION getCity RETURN VARCHAR2 IS
  BEGIN          RETURN city;
  END;
  MEMBER PROCEDURE setStreet (newStreet IN VARCHAR2) IS
  BEGIN          street := newStreet;
  END;
END;
/
```

Benutzung eines adress Objekts:

```
        Dbms_output.put_line(Address_var.getCity);
```

Beispiel: Objekte als Attribute

```
CREATE TABLE employee
( emp_id          NUMBER PRIMARY KEY,
  name            VARCHAR2(40),
  home_address  address    );

INSERT INTO employee values(1234,‘George Baker‘,
address(‘ Av. Emilio Navarro‘,‘Coimbra‘,‘P-3000‘) );

DECLARE  emp_home  address;
BEGIN  emp_home:= address(‘ Av. Emilio Navarro‘,‘Coimbra‘,‘P-3000‘);
          INSERT INTO employee values(1234,‘George Baker‘, emp_home);
END;

SELECT e.emp_id, e.name, e.home_address.city FROM employee e;

BEGIN  SELECT home_address INTO emp_home  FROM employee
                              WHERE emp_id = 1234;
emp_home.setStreet(‘Berliner Tor 3‘);
UPDATE employee SET home_address = emp_home WHERE emp_id = 1234;
END;
```

Beispiel: Objekte in Object Tables

```
CREATE OR REPLACE TYPE building AS OBJECT (
  BldgName          VARCHAR2(40),
  BldgAddress       address,
  ORDER MEMBER FUNCTION Compare (OtherBuilding IN building)
RETURN INTEGER  );

CREATE OR REPLACE TYPE BODY building AS
  ORDER MEMBER FUNCTION Compare (OtherBuilding IN building)
  RETURN INTEGER IS
     BldgName1     VARCHAR2(40);
     BldgName2     building.BldgName%TYPE;
  BEGIN            BldgName1 := upper(ltrim(rtrim(BldgName)));
                   BldgName2 := upper(ltrim(rtrim(OtherBuilding.BldgName)));
                   IF BldgName1 = BldgName2  THEN  RETURN  0;
                   ELSIF BldgName1 < BldgName2  THEN  RETURN -1;
                   ELSE    RETURN  1;  END IF;
    END;
END;

CREATE TABLE buildings OF building;
```

Beispiel: Objekte in Object Tables …

```
INSERT INTO buildings  values ( building('ET-Hochhaus',
        address('Berliner Tor 3',' Hamburg','D-20099' ) );
INSERT INTO buildings  values ( building('Hotel Astoria', Address_var ) );

SELECT * FROM buildings;
```

BLDGNAME	BLDGADDRESS(STREET, CITY, ZIP-CODE)
ET-Hochhaus	ADDRESS('Berliner Tor 3','Hamburg','20099')

```
SELECT  BldgName FROM buildings ORDER BY BldgName;
SELECT b.BldgAddress.street FROM buildings b;
```

Zugriff auf die Komponenten eines Objekts

```
DECLARE  this_building   building;
                CURSOR all_buildings IS
SELECT value (b) AS bldg FROM buildings b ORDER BY b.BldgName;
BEGIN   FOR one_building IN all_buildings
LOOP   this_building := one_building.bldg;
          dbms_output.put_line(this_building.BldgName || ' is located in ' ||
                                                  this_building.BldgAddress.city);
END LOOP;
COMMIT;  END;

DECLARE  one_building   building;
                CURSOR all_buildings IS
SELECT * FROM buildings b ORDER BY b.BldgName;
BEGIN   FOR one_building IN all_buildings
LOOP   dbms_output.put_line(one_building.Bldgname ||
          ' is located in ' || one_building.Bldgadress.ort);
END LOOP;
COMMIT;  END;
```

Object Tables mit Referenzen

```
CREATE TYPE personal_typ AS OBJECT
(   persnr          NUMBER,
    name           VARCHAR2(30) );
CREATE TYPE abteilung_typ AS OBJECT
(   abt             varchar2(3),
    manager        REF personal_typ );
```

CREATE TABLE personal OF personal_typ; CREATE TABLE abteilung OF abteilung_typ;

- diese Tabellen enthalten referenzierbare Objekte
- man könnte sagen: eine Tupel ist eine Instanz der Klasse
- object Identifier IOD ist weitweit eindeutig.

abteilung

abt	manager
EDV	→

REF

personal

persnr	name
1234	Petra

SELECT a.manager.name FROM abteilung a, personal p WHERE abtnr = ‘EDV’

Referenzen (REF, DEREF)

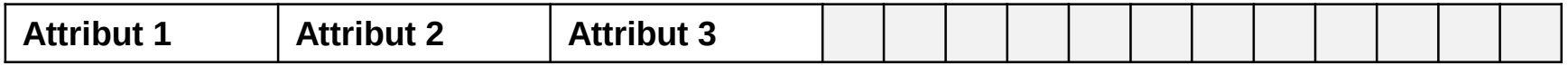
```
CREATE OR REPLACE PROCEDURE AssignManager
        (abtIn IN abteilung.abtnr%type,  mgrIn IN personal.persnr%type)  AS
BEGIN  UPDATE abteilung
SET  manager = (SELECT REF(p) FROM personal p WHERE persnr=mgrIn)
                WHERE  abtnr=abtIn
IF SQL%NOTFOUND THEN raise_application_error(...);  END IF;
END;

CREATE OR REPLACE FUNCTION GetManager
        (abtIn IN abteilung.abtnr%type)  RETURN VARCHAR2(30)  AS
DerManager  personal_typ;
BEGIN  SELECT DEREF(manager) INTO DerManager FROM abteilung
                WHERE abtnr=abtIn
IF DerManager IS NULL THEN RETURN  'Kein Manager'
ELSE RETURN  DerManager.name  END IF;
END;
/
```

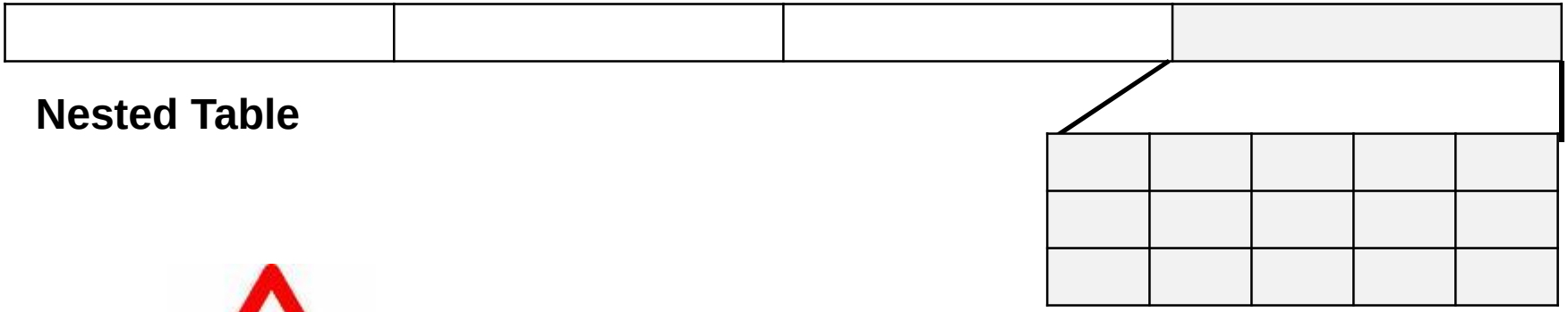
Collections (VArray, Nested Table)

Neue Datentypen: variabel lange Tabellen VARRAY,
nested tables (TABLE type)

- Ermöglichen die direkte Umsetzung von 1:n-Beziehungen
- Können skalare Werte oder Objekte enthalten
- Variabel lange Tabellen



Variabel lange Tabelle



Nested Table

Non first Normal Form

VArrays

- Ein VARARY enthält 0 oder mehr geordnete Elemente des gleichen Typs
- Speicherung wie ein Attribut, also ***inline*** (bis 4 KB Länge)
- Ein VARRAY ist aus Sicht von SQL eine Einheit, d. h. kein Zugriff auf die Elemente (geht über PL/SQL bzw. 3GL)
- Geeignet für geringe bekannte Elementanzahl

```
CREATE TYPE telefon_typ AS OBJECT
(       beschreibung            VARCHAR2(10),
        telefonnr               VARCHAR2(12) );

CREATE TYPE telefonliste_typ AS VARRAY(5) OF telefon_typ;
CREATE TYPE kunden_typ AS OBJECT
(       kundennr                NUMBER(5),
        name                    VARCHAR2(30),
        telefon_nummern         telefonliste_typ );

CREATE TABLE kunden OF kunden_typ;
```

Zugriff auf VArrays

Insert into kunden values (234, ‘Asterix‘, telefonliste_typ(telefon_typ(‘privat‘,‘04012345‘), telefon_typ(‘Handy‘,‘017098‘))) ;

select * from kunden;

KUNDENNR	KNAME	TELEFON_NUMMERN(BESCHREIBUNG, TELEFONNR)
123	Otto	
234	Asterix	TELEFONLISTE_TYP(TELEFON_TYP(’privat’, ’0401234’), TELEFON_TYP(’Handy’, ’017098’))

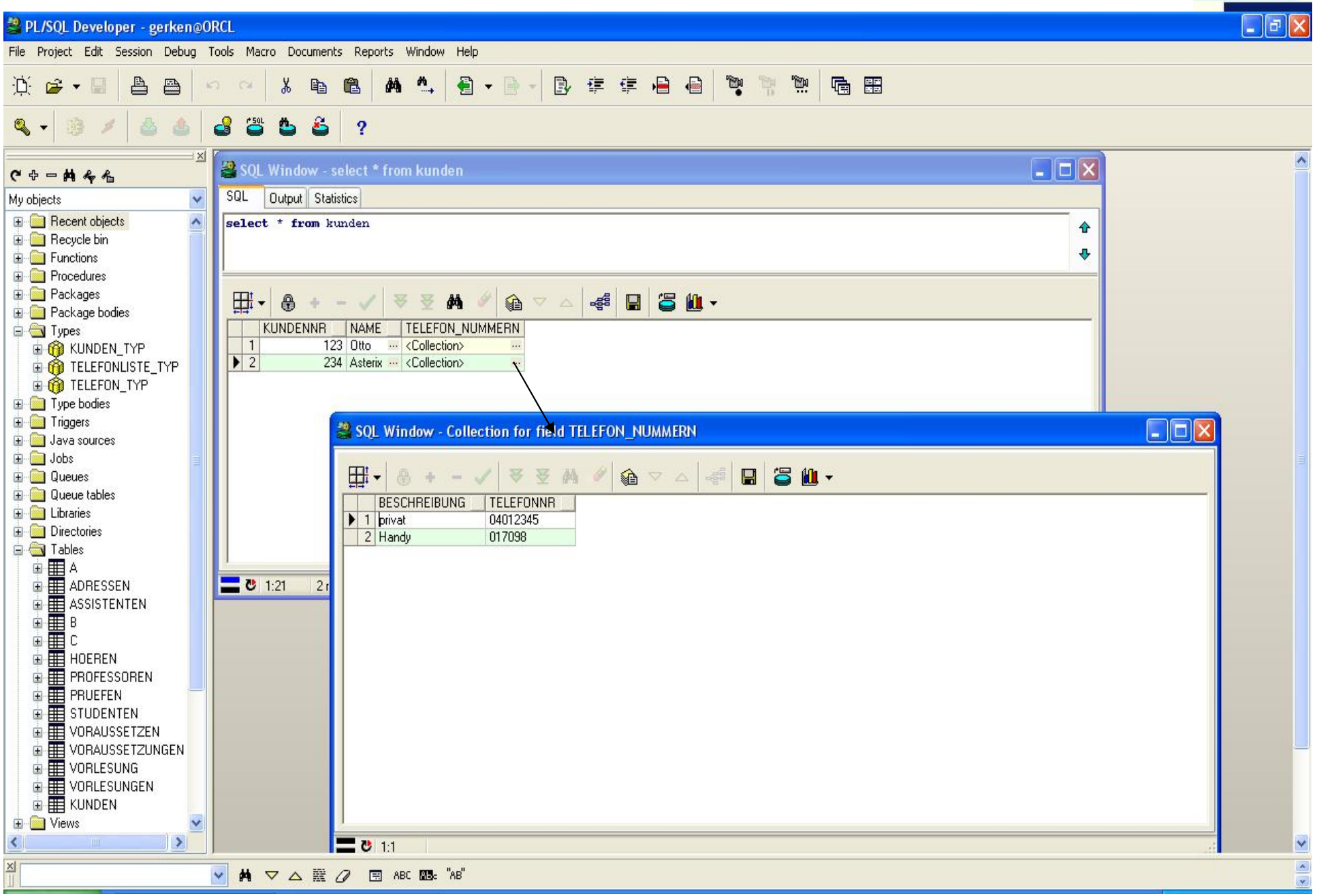
select k.telefon_nummern from kunden k;

TELEFON_NUMMERN(BESCHREIBUNG, TELEFONNR)
TELEFONLISTE_TYP(TELEFON_TYP(’privat’, ’0401234’), TELEFON_TYP(’Handy’, ’017098’))

select * from the (select k.telefon_nummern from kunden k where kundennr=234);

BESCHREIBUNG	TELEFONNR
privat	0401234
Handy	017098

Zugriff auf VArrays



Zugriff auf VArray-Elemente

```
DECLARE  Kunde kunden_typ;
BEGIN
        Select  value(k) into Kunde from kunden k  where kundennr=234;
        FOR j in 1..Kunde.telefon_nummern.count
        LOOP
        Dbms_output.put_line(Kunde.telefon_nummern(j).telefonnr);
        END LOOP;
END;

DECLARE  Tliste telefonliste_typ;
BEGIN
        Select  k.telefon_nummern into Tliste from kunden k where
                kundennr=234;
        FOR j in 1..Tliste.count
        LOOP
        Dbms_output.put_line(Tliste(j).beschreibung ||  ' ' || Tliste(j).telefonnr);
        END LOOP;
END;
```

Nested Tables

- Geeignet für 1:n-Beziehungen (master/detail)
- Speicherung outline, d. h. nicht in der Tabelle, aber im selben Tablespace
- Unterstützung für SQL

```
CREATE TYPE produkt_typ AS OBJECT
(       produktnr                       NUMBER(5),
        preis                           NUMBER(7,2) );
CREATE TYPE produkt_nested_typ AS TABLE OF produkt_typ;
CREATE TABLE bestellung
(       bestellnr                       NUMBER(5),
        kundennr                        NUMBER(5),
        bestelldatum                    DATE,
        produkte                        produkt_nested_typ )
NESTED TABLE produkte STORE AS bestellung_produkte;
```

Check-Klauseln für die Attribute der Nested table müssen in store as–Tabelle definiert sein. Beispiel: Alter table bestellung_produkte add constraint Preisgtzero check(preis > 0) ;

Primary-Key- und Foreign-Key-Angaben sind z.Zt. noch nicht möglich.

Nested Tables …

Insert into bestellung values (30, 981, to_date(´14-Feb-2007´), produkt_nested_typ (produkt_typ(80, 13.90), produkt_typ(110,11.50),produkt_typ(50,110.00)));

Insert into the (select produkte from bestellung where bestellnr=30) values (33,11.11);

NF²-Tabelle

BestNr	...	Produkte	
30	...	80	13.90
		110	11.50
		50	110.00
		33	11.11
31	...	50	110.00
		...	

Realisierung mit Oracle

BestNr	...	Id
30	...	X001
31	...	X002

Id	ProdNr	Preis
X001	80	13.90
X001	110	11.50
X001	50	...
X001	33	
X002	50	...

Id = systemgenerierter Nested table Identifier

Nested Table als Attribut

```
create type bestellungs_typ as object (TeileNr  number(5), menge
number(5));
create type bestellung_nested_typ as table of bestellungs_typ;

create table kunden2 (Kdnr  number(5), name varchar2(30));
insert into kunden2 values(1,'Peter');
insert into kunden2 values(2,'Irma');

alter table kunden2 add bestellungen bestellung_nested_typ
nested table bestellungen store as BestTab;

select * from kunden2;
```

KDNR	NAME	BESTELLUNGEN(TEILENR, MENGE)
1	Peter	
2	Irma	

Null-Werte

Zugriff auf Nested Tables

```
insert into the(select bestellungen from kunden2 where kdnr=2)
                values(200,44);
```

FEHLER in Zeile 1: ORA-22908: Referenz auf Tabellenwert NULL

```
update kunden2 set bestellungen =
Bestellung_nested_typ(Bestellungs_typ(100,33)) where kdnr=1;
select * from kunden2;
```

KDNR	NAME	BESTELLUNGEN(TEILENR, MENGE)
1	Peter	BESTELLUNG_NESTED_TYP(BESTELLUNGS_TYP(100, 33))
2	Irma	

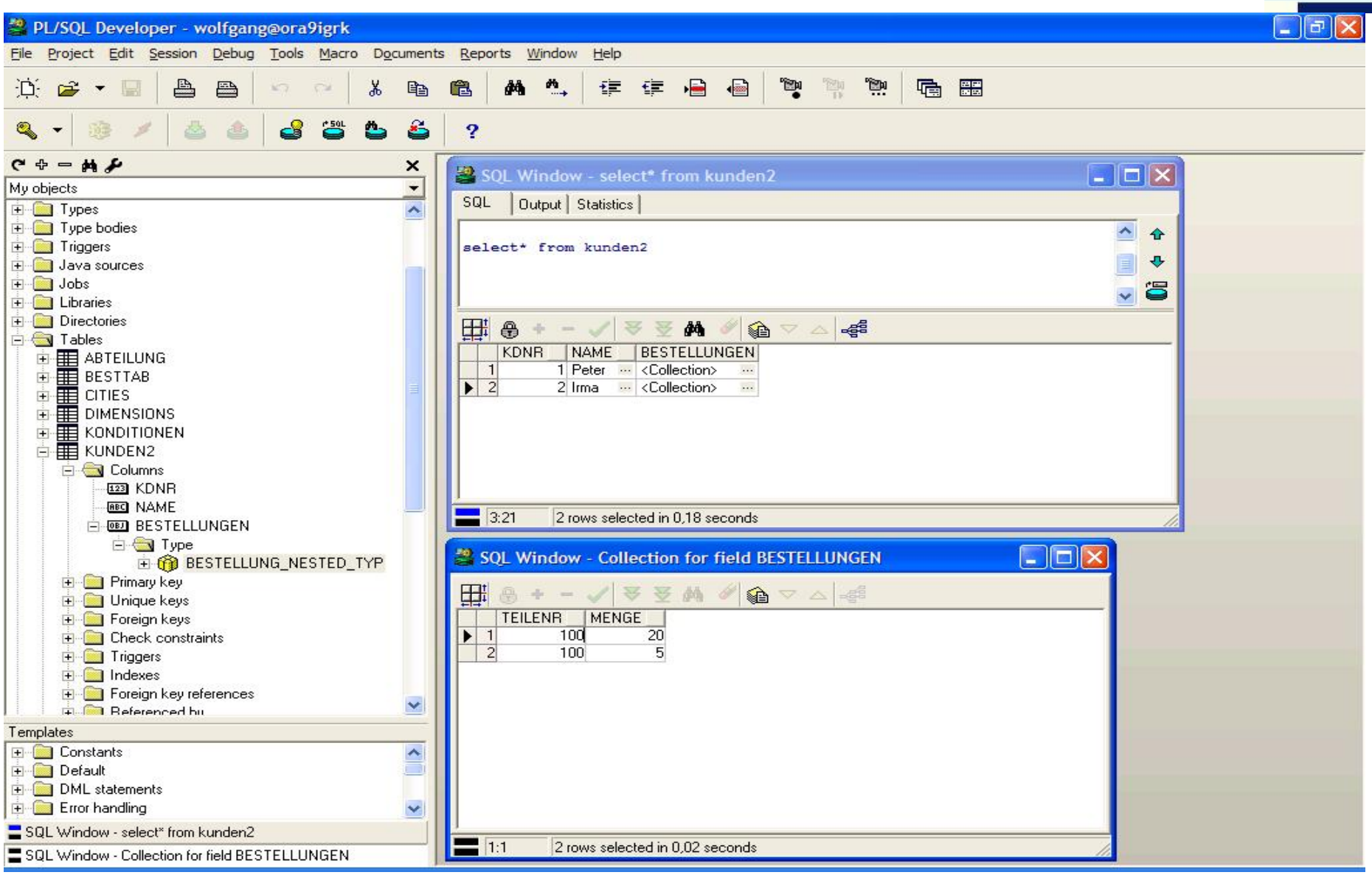
```
delete from the(select bestellungen from kunden2 where kdnr=1);
select * from kunden2;
```

KDNR	NAME	BESTELLUNGEN(TEILENR, MENGE)
1	Peter	BESTELLUNG_NESTED_TYP()
2	Irma	

leer ist ≠ NULL

```
insert into the(select bestellungen from kunden2 where kdnr=1)
                values(200,44);
```

Nested Tables im PLSQL-Developer



Der THE-Operator

desc bestellung;

Column Name	Null?	Type
BESTELLNR		NUMBER(5)
KUNDENNR		NUMBER(5)
BESTELLDATUM		DATE
PRODUKTE		RAW(56)

select * from the (select produkte from bestellung where bestellnr=30)

PRODUKTNR	PREIS
80	13.9
110	11.5
50	110
33	11.11

4 rows selected.

Restriktion möglich: z.B. where Preis < 12

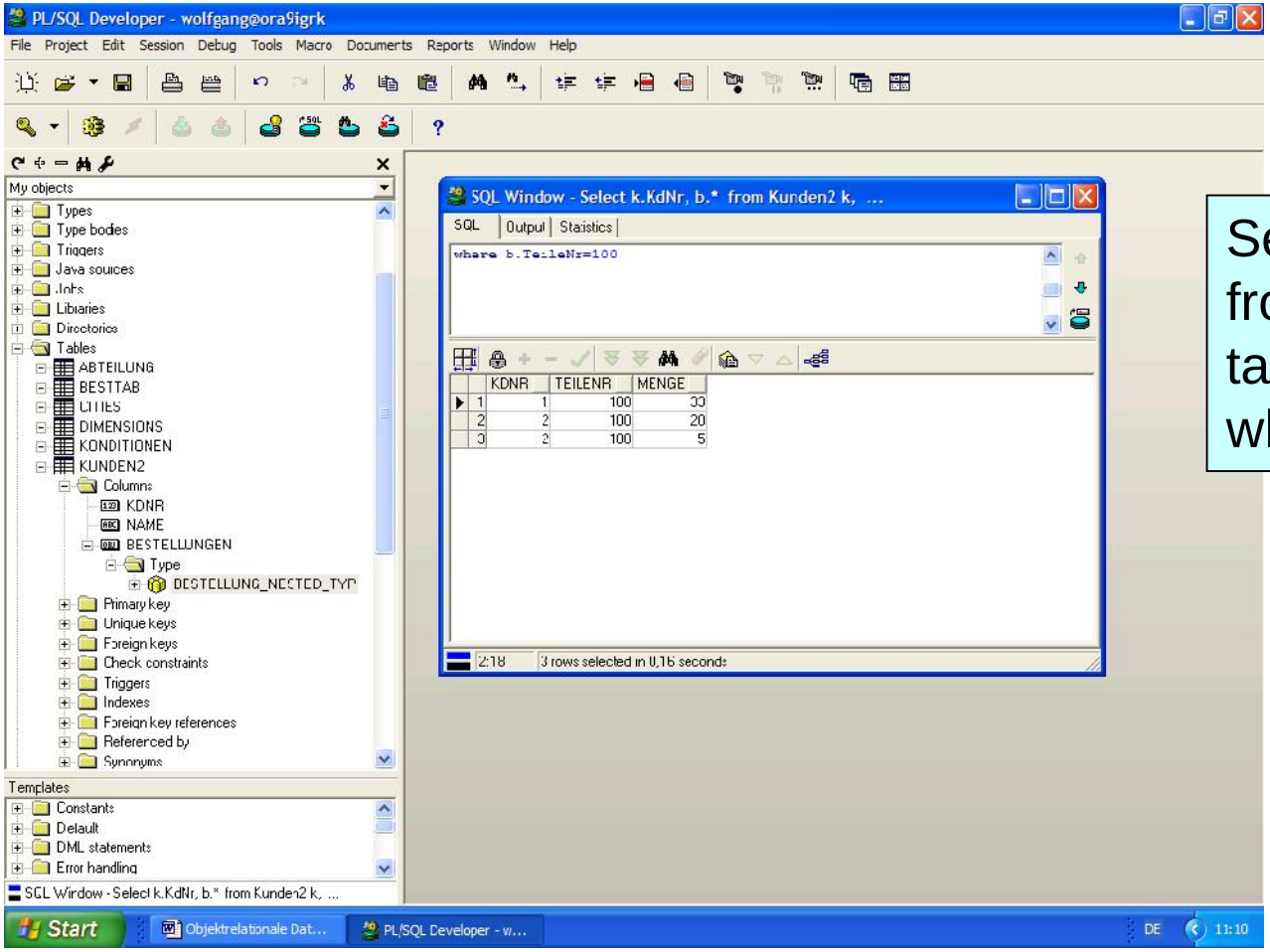
select * from the (select produkte from bestellung where bestellnr > 10)

PRODUKTNR PREIS

ORA-01427: single-row subquery returns more than one row

Der Table-Operator

Table( ) führt ein Type-Casting eines Nested-Table-Attributs auf eine Tabelle durch.




```
Select k.KdNr, b.*
from Kunden2 k,
table(k.Bestellungen) b
where b.TeileNr=200
```

Der Table-Operator

Select BestellNr, cursor(Select * from table (b.produkte)) from bestellung b;

```
BESTELLNR    CURSOR(SELECT*FROMTA
----------   ------------------------------------

      30              2

_____________________________________________

CURSOR NUMBER :              2
 PRODUKTNR     PREIS
_____________ _____
        80      13.9
       110      11.5
        50       110
        33     11.11
4 rows selected.
--------------------------------
1 row selected.
```

Queries bei Nested Tables ...

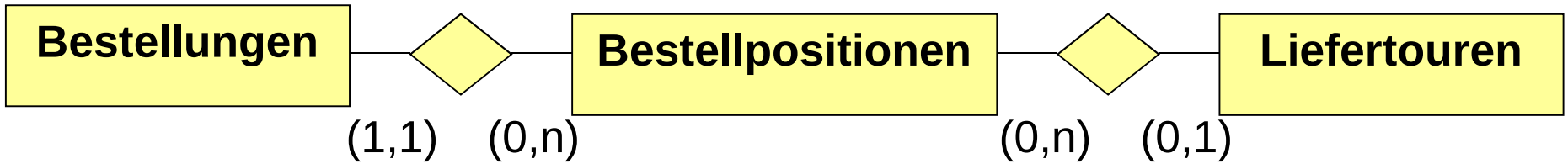
```
set serveroutput on;
Declare bnr bestellung.bestellnr%TYPE;
        pro bestellung.produkte%TYPE;
cursor b_csr is select bestellnr, produkte from bestellung;
Begin
open b_csr;
loop    fetch b_csr into bnr, pro;
        exit when b_csr%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Bestellung '||bnr);
        for ind in 1..pro.COUNT
        loop DBMS_OUTPUT.PUT_LINE('          Teil ' || pro(ind).produktnr);
        end loop;
end loop;
close b_csr;
end;
```

```
Bestellung 30
        Teil 80
        Teil 110
        Teil 50
        Teil 33
Bestellung 31
```

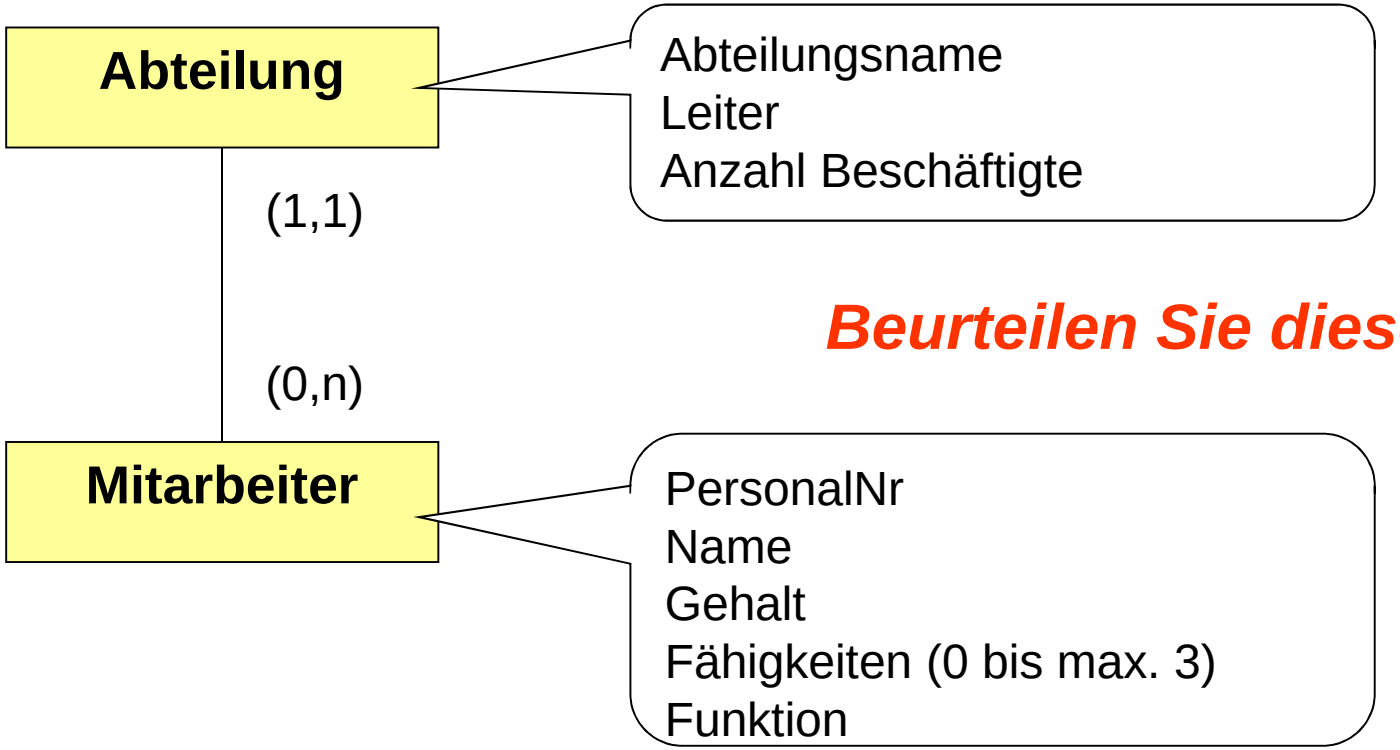
Anwendungsmöglichkeiten von Nested Tables

Nested Table: bei Komposition und Aggregation,
bei Assoziation mit (1,1); d.h. Existenzabhängigkeit

Zusätzlich für Varray: Wiederholungsgruppen geringer, bekannter Elementanzahl



Fallstudie (Nested Table, Varray)



Beurteilen Sie diesen Entwurf !

Beispiel-Anwendung