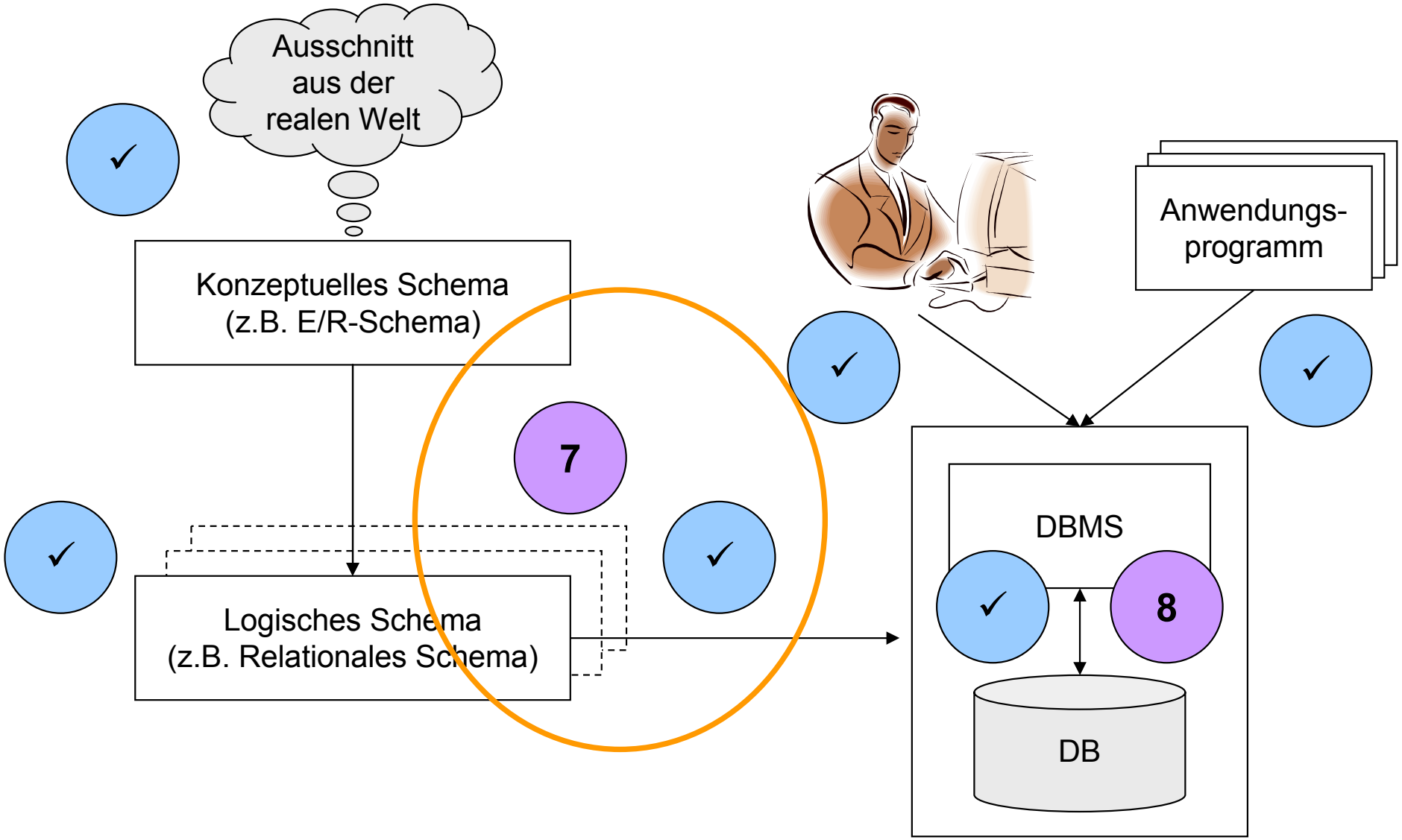
Aufbau Vorlesung Datenbanken II – Kapitel 6, 7 und 8



- **Weitere Datenbankmodelle**

7.2.1 Ein einführendes Beispiel (1) – Hörsaalübung

Ein **Kunde** (kundennr, name) kann mehrere Aufträge geben, jeder **Auftrag** (auftragsnr, eingangsdatum, bearbeitungsdatum) gehört zu genau einem Kunden.

Zu einem Kunden kann es mehrere Adressen geben (z.B. Lieferadresse, Postadresse etc.), aber jede **Adresse** (plz, ort, strasse, hausnr) ist eindeutig einem Kunden zugeordnet.

Zu jedem Kunden können maximal drei **Telefonnummern** gespeichert werden.

Zu jedem Kunden können beliebig viele **Kontakte** (Namen von Kontaktpersonen) gespeichert werden.

Wie implementiert man beim Übergang zur Designsicht die Beziehungen?

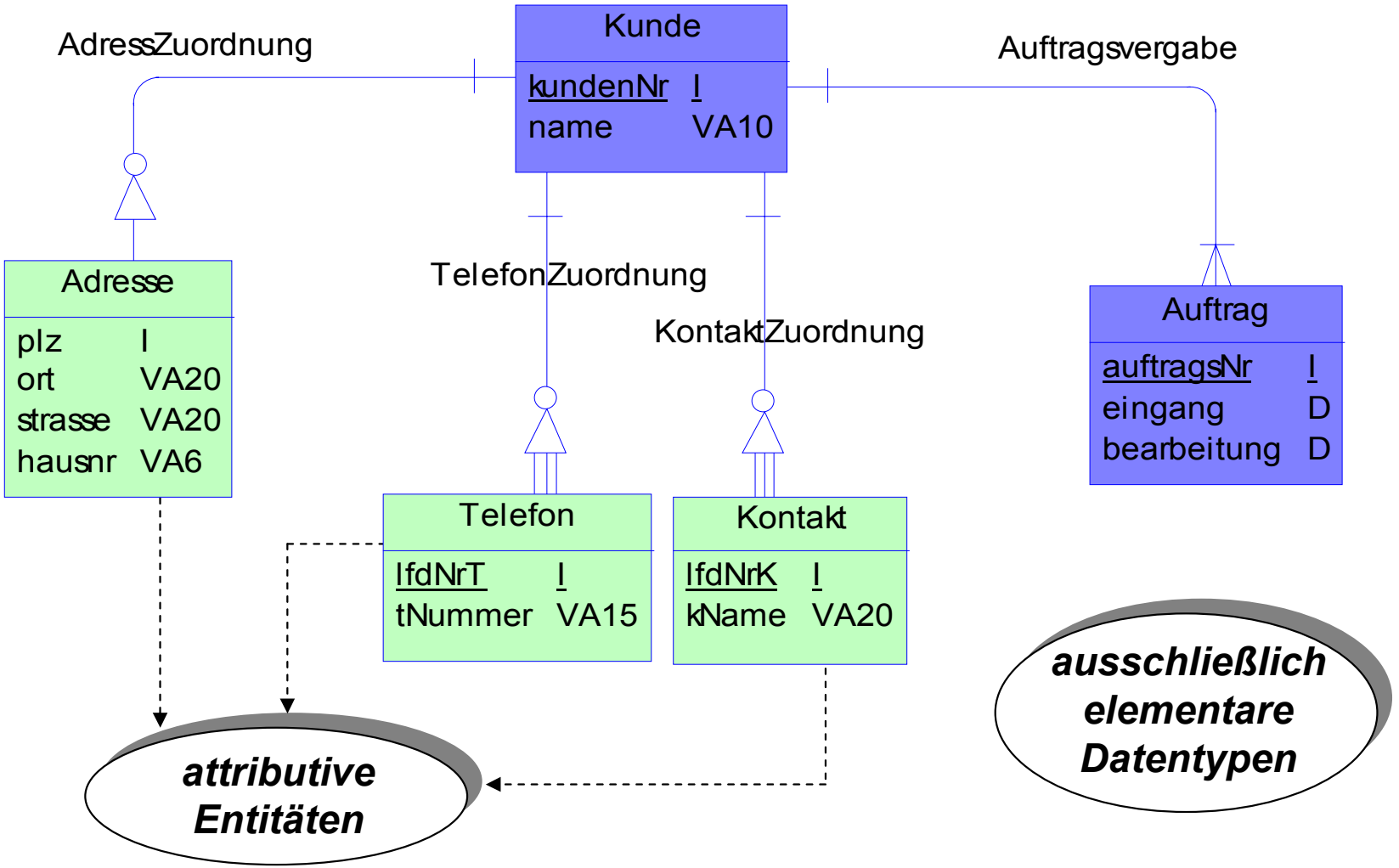
Gibt es Freiheitsgrade für die Wahl der Navigationsrichtung?

Modellieren Sie für die oben beschriebene Situation

- ein E/R-Diagramm (konzeptionelles Modell – Analyse),
- ein entsprechendes relationales (physisches) Modell (Design), sowie
- ein UML-Klassendiagramm (Analysesicht),
- ein entsprechendes Java-Klassendiagramm (Designsicht)

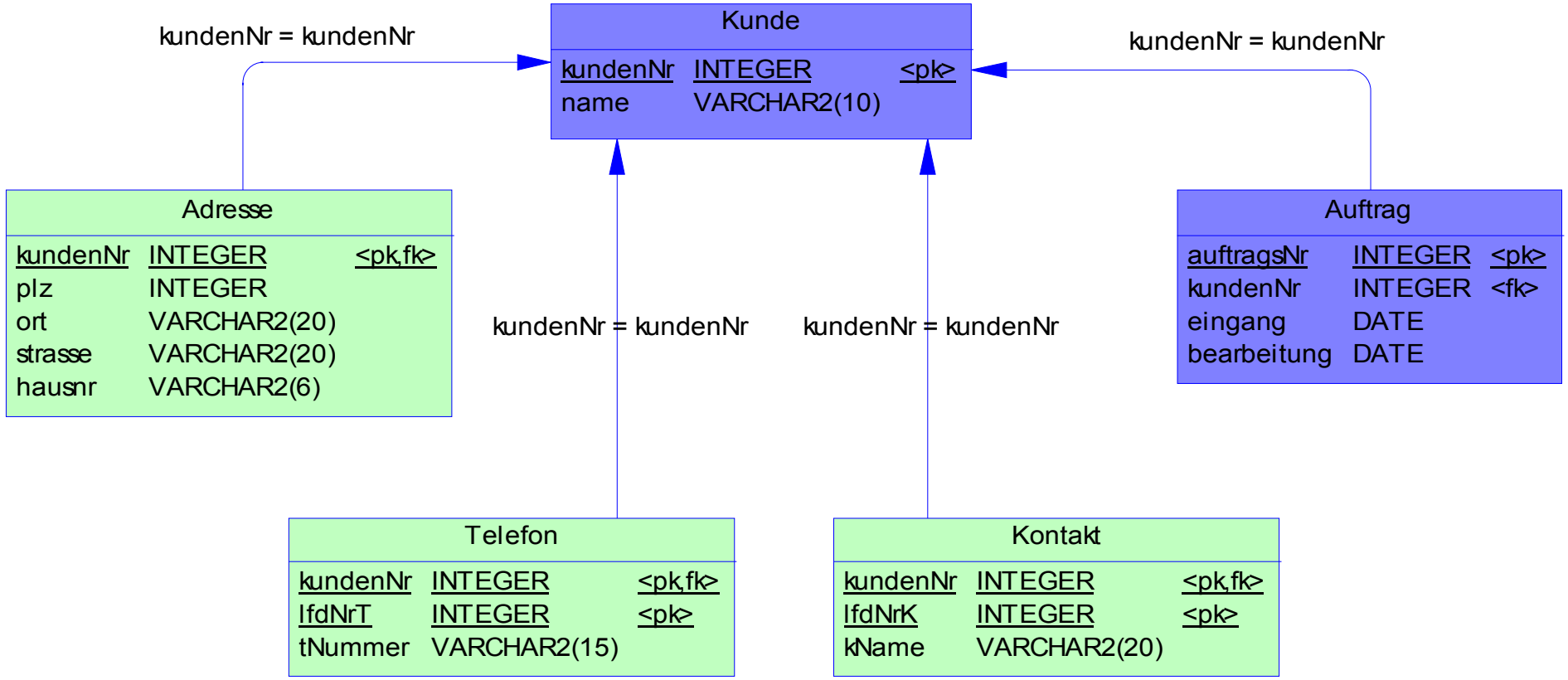
Welche Datentypen bieten sich an?

E/R-Diagramm zur „Auftragsverwaltung“ – **Analysesicht**



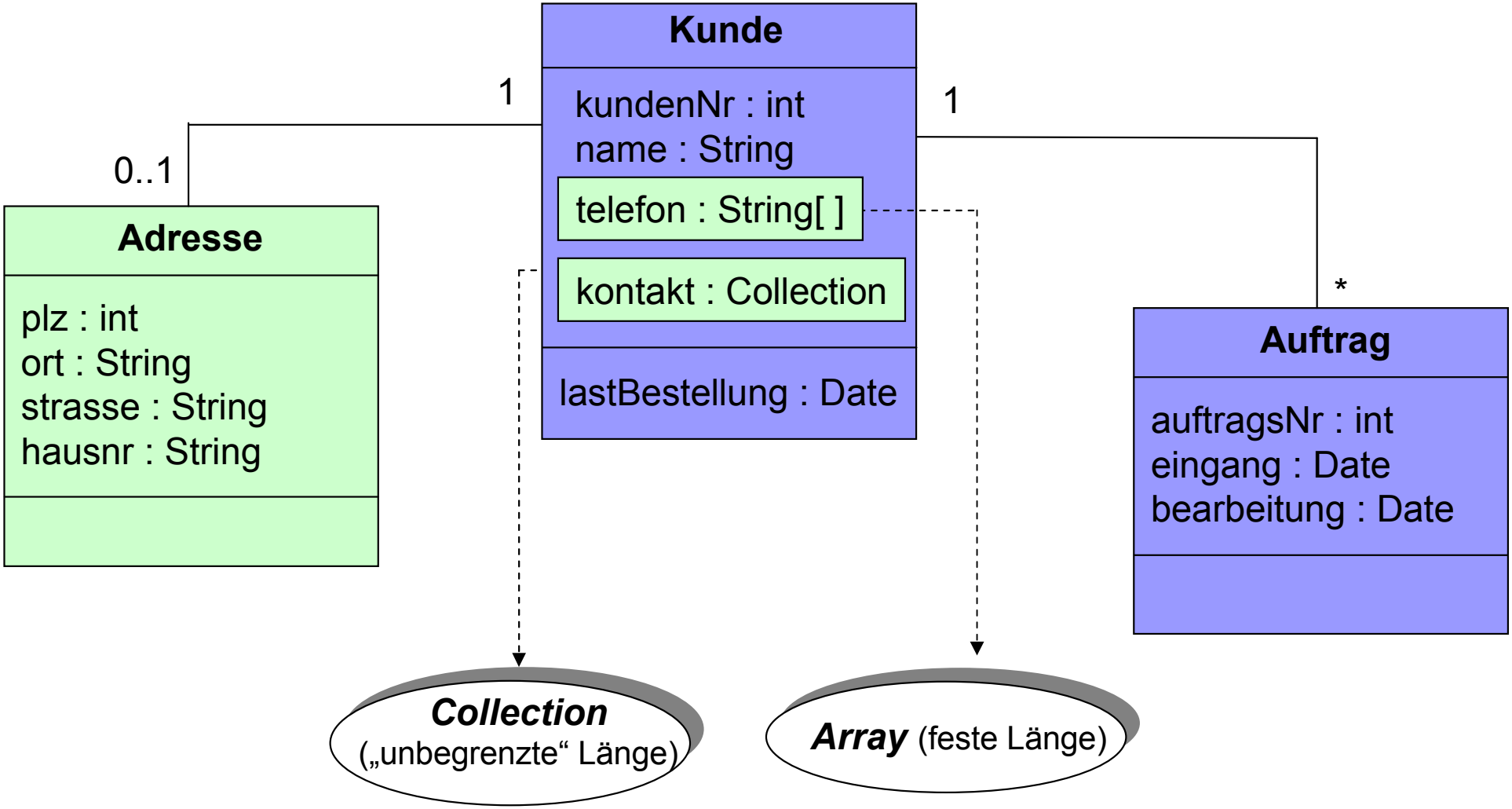
7.2.1 Ein einführendes Beispiel (3) – Hörsaalübung

relationales Schema zur „Auftragsverwaltung“ – **Designsicht** (Oracle 9i)



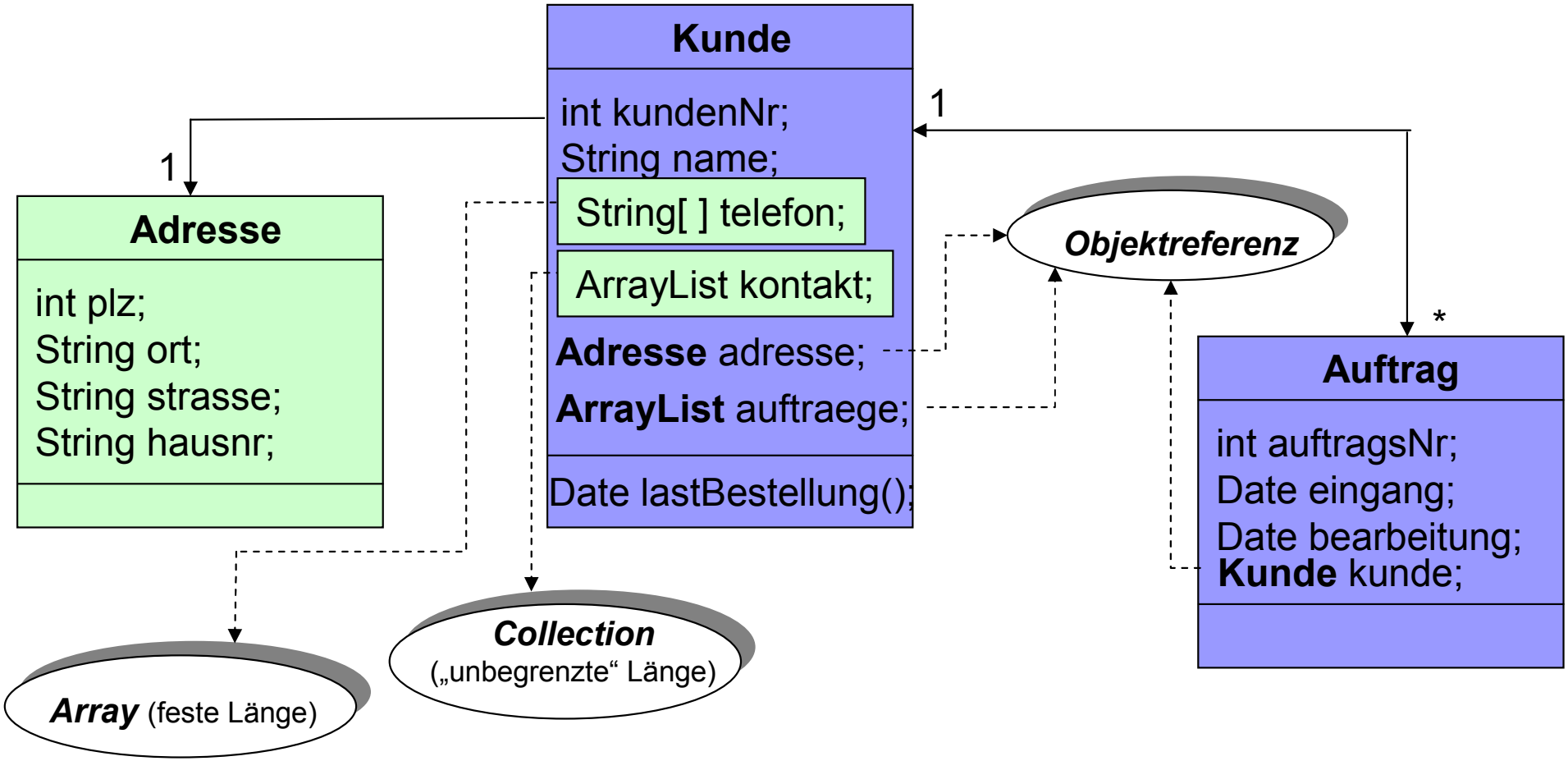
- Die Implementierung der Beziehungen erfolgt durch **Foreign-Key-Spalten**.
- Die **Navigationsrichtung** (Referenz) erfolgt **immer „in Richtung der 1“** (1. CNF!)

UML Klassendiagramm zur „Auftragsverwaltung“ – **Analysesicht**



7.2.1 Ein einführendes Beispiel (5) – Hörsaalübung

UML Klassendiagramm zur „Auftragsverwaltung“ – **Designsicht** (Java)



- Die Implementierung der Beziehungen erfolgt durch **Referenzattribute**.
- Die **Navigationsrichtung** kann **beliebig** implementiert werden. Mehrfachreferenzen werden durch Attribute vom Typ *Collection* implementiert!

7.2.2 SQL3, SQL99 und SQL4 (1)

Objektrelationale Datenbanken stellen eine Erweiterung relationaler Datenbanken um objektorientierte Konzepte dar. Der SQL-Standard SQL3 (teilweise realisiert in SQL99) unterstützt hierfür die folgenden objektorientierten Konzepte:

- **abstrakte Datentypen (ADT)**
- **Spezialisierungshierarchien** für Tabellen **(Relationenhierarchien)** und Typen, und
- **Objektidentitäten** für komplexe Tupel in Relationen, die dann auch **Klassen** genannt werden können.

- Die Operationen in objektrelationalen Datenbanken erfolgen weiterhin *nur* relational, da das zu Grunde liegende Datenmodell nach wie vor relational ist!

- Der von ANSI und ISO projektierte SQL3-Standard wurde teilweise 1999 als SQL99 veröffentlicht, während in SQL99 unberücksichtigte Erweiterungen und Änderungen nun für den SQL4-Standard projektiert werden.

- **Abstrakte Datentypen (ADT)**

  Abstrakte Datentypen werden mit der create type-Anweisung generiert und enthalten *Attribute* und *Funktionen* bzw. *Prozeduren*, die unterschiedliche *Sichtbarkeiten* haben können:

Sichtbarkeiten

public
protected
private

Syntax:
```
CREATE TYPE  <Typbezeichner>
(   <Sichtbarkeit>  <Attribut> <Datentyp>,
                    ...
    <Sichtbarkeit> FUNCTION | PROCEDURE
                    <Funktionsname>(Parameterliste)
    nur bei Functions RETURNS <Datentyp>
                    BEGIN
                    ...
                    END, ...
)
```

- Für jedes Attribut werden automatisch zwei Methoden generiert:
  - eine *Observer-Funktion*, die den lesenden Zugriff auf das Attribut unterstützt und also eine Anfrage-Methode darstellt,
  - eine *Mutator-Prozedur*, die den schreibenden Zugriff auf das Attribut unterstützt und einer Update-Methode entspricht. Vgl. auch die Analogie zu get- und set-Methoden bei Java!

7.2.2 SQL3, SQL99 und SQL4 (3)

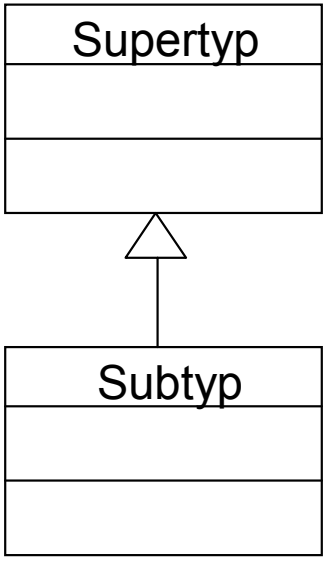
Syntax:
```
CREATE TYPE  <Typbezeichner>
(   <Sichtbarkeit>  <Attribut> <Datentyp>,
                    ...
    <Sichtbarkeit> FUNCTION | PROCEDURE
                    <Funktionsname>(Parameterliste)
    nur bei Functions RETURNS <Datentyp>
                    BEGIN
                    ...
                    END, ...
)
```

Deklarationen von **Funktionen** und **Prozeduren** werden als **Stored Procedures** im System abgelegt (vgl. Kapitel 6).

- Man unterscheidet zwischen
  - **Objekt-ADT**: mit Objektidentität (OID), Konstruktor mit dem Namen des ADT, Destruktor destroy
  - **Wert-ADT**: keine Objektidentität, keine Destruktoren.
- Im Beispiel dieses Kapitels ist *Typ_Adresse* ein Wert-ADT, die ADTs *Typ_Kunde* und *Typ_Auftrag* sind Objekt-ADTs:
  - Die Instanzen vom *Typ_Kunde* und *Typ_Auftrag* werden in eigenen Tabellen verwaltet und sind über OIDs adressierbar.
  - Eine Instanz vom *Typ_Adresse* ist als Attributwert vollständig in der Kundentabelle enthalten und nicht über eine OID adressierbar.

- **Typ- und Tabellenhierarchie**
  Für eine *Typhierarchie* wird der Subtyp als Spezialisierung eines Supertyps mit der under-Klausel definiert:

Syntax: **CREATE TYPE** <Subtyp> **UNDER** <Supertyp>
(
...
)



- Zur Verwendung einer Methode des Supertyp, die im Subtyp überschrieben wurde, kann man mit Hilfe der as-Klausel ein type casting vornehmen:

  <Methodenname>(<subtyp-Ausprägung> **as** <Supertyp>)

- Umgekehrt kann auch für eine Ausprägung des Supertyps durch casting auf einen Subtyp eine spezialisierte Methode des Subtyps aufgerufen werden - mit Hilfe der treat...as-Klausel:

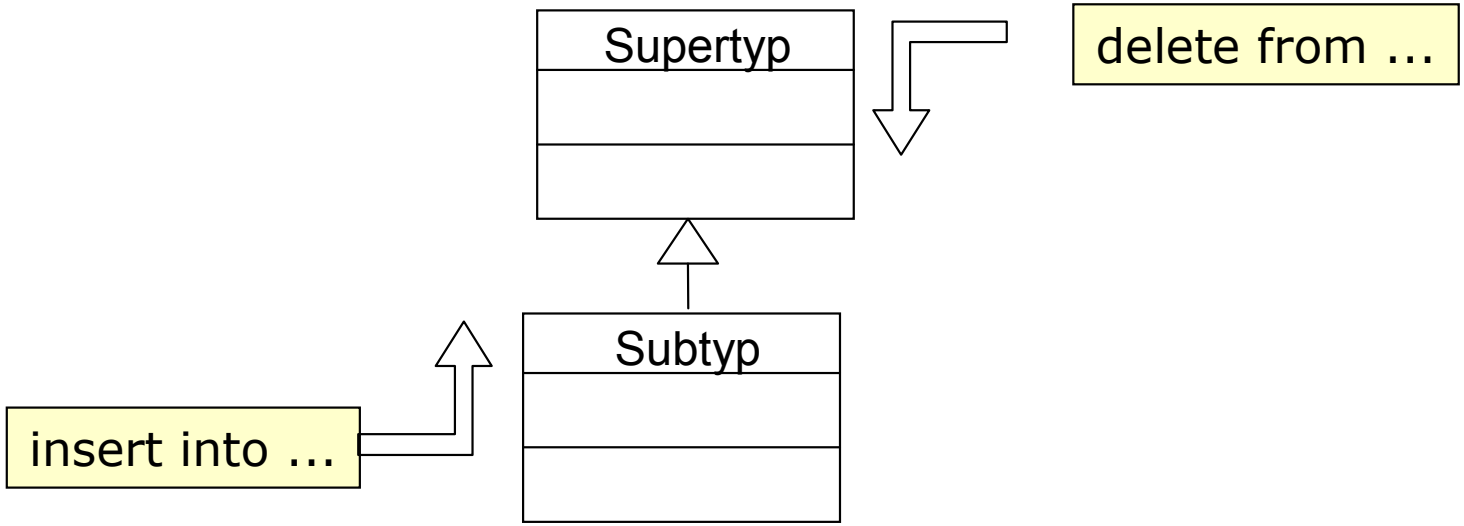
  <Methodenname>(**treat** <supertyp-Ausprägung> **as** <Subtyp>)

- Die *Tabellenhierarchie* entspricht der Klassenhierarchie als objektorientiertes Konzept. DieTabellenhierarchie wird dann - auf der Basis einer Typenhierarchie - wie folgt definiert:

Syntax: **CREATE TABLE** <Subtable<> **OF** <Subtyp>
UNDER <Supertable> **OF** <Supertyp>
(
...
)

Die Subtabelle enthält alle Attribute der Supertabelle!

- Jedes **insert** in die Subtabelle wird auf der Supertabelle fortgesetzt,
- Jedes **delete** der Supertabelle wird auf die Subtabelle fortgesetzt.



ab Oracle8

Umsetzung der objektrelationalen SQL3-Konzepte in Oracle8

- ***Abstrakte Datentypen (ADT)***

  Die Unterscheidung von Objekt-ADT und Wert-ADT erfolgt ab Oracle8 nur implizit durch die Art der Nutzung - wir sprechen deshalb im folgenden häufig von **Objekttypen**.

  Es stehen bislang zwei Klassen von Kollektionstypen zur Verfügung:
  - **Arrays** und
  - **geschachtelte Tabellen**.

- ***Typ- und Tabellenhierarchie***

  Die Unterstützung von Spezialisierungs- und Vererbungshierarchien auf Objekttypen und Tabellen ist noch nicht vollständig umgesetzt.

ab Oracle8

- Ein **Objekttyp** stellt eine Schablone mit Attributen und Methoden eines Objektes der „realen Welt“ dar (in Anlehnung an die objektorientierte Klassendeklaration).

Syntax:
```
CREATE TYPE  <Objecttyp> AS OBJECT
(   <Attribut> <Datentyp>,
                    ...
   MEMBER FUNCTION | PROCEDURE <Funktionsname>[(Parameterliste)]
   nur bei Functions RETURN <Datentyp>;
 )
```

- Wir wollen den **Typ_Adresse** definieren, um ihn anschließend in einem größeren Objektmodell zu verwenden:

```
CREATE TYPE Typ_Adresse AS OBJECT
( plz       NUMBER(5),
  ort       VARCHAR2(20),
  strasse   VARCHAR2(20),
  hausnr    VARCHAR2(5)
);
```

7.2.3 Generierung von Tabellentypen – VArray-Typen

ab Oracle8

- Außerdem benötigen wir einen **Tabellentyp**, ...

Syntax: **CREATE TYPE** <Tabellentyp>
AS TABLE OF <Domäne>;

... mit dem wir eine Liste von Kontaktpersonen verwalten können:

```
CREATE TYPE Typ_Kontakt
AS TABLE OF  VARCHAR2(20);
```

- Im Unterschied zu Tabellentypen ist ein **VArraytyp** geordnet und in ihrer Größe begrenzt:

Syntax: **CREATE TYPE** <VArraytyp>
AS VARRAY(<Grösse>)
OF <Domäne>;

- Wir benötigen für unser Beispiel einen Datentyp, mit dem wir eine geordnete Liste von maximal 3 Telefonnummern pro Datensatz verwalten können, also einen VArraytyp **Typ_Telefon:**

```
CREATE TYPE Typ_Telefon
AS VARRAY(3)
OF VARCHAR2(20);
```

ab Oracle8

- Objekt-, Table- und VArraytypen können auf zwei Arten verwendet werden:
  1. als *Wertebereich eines Attributes* innerhalb einer Tabellen-Deklaration, oder
  2. zur *Definition einer Tabelle*, für die jede Zeile direkt dem Objekttyp entspricht (**Objekttabelle**)

- *zu 1.:*

  Wir verwenden die drei oben definierten, komplexen Datentypen, um einen weiteren Objekttyp **Typ_Kunde** zu definieren:

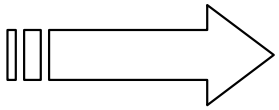
```
CREATE TYPE Typ_Kunde AS OBJECT (
   KNr       INTEGER,
   Name      VARCHAR2(20),
   Adresse   Typ_Adresse,
   Telefon   Typ_Telefon,
   Kontakt   Typ_Kontakt
);
```

ab Oracle8

- *zu 2.:*

  Wir deklarieren die **Objekttabelle T_Kunde**, für die jede Zeile direkt dem Objekttyp Typ_Kunde entspricht:

Syntax: **CREATE TABLE** <Tabelle>
OF <Typ>;



```
CREATE TABLE T_Kunde
OF Typ_Kunde;
```

- Die Tabellen-Deklaration kann um die bekannten constraints - implizit und explizit - erweitert werden.

```
CREATE TABLE T_Kunde
OF Typ_Kunde (
     KNr primary key);
```

- *Constraints* werden immer erst bei der Tabellen-Deklaration und nicht schon bei der Typ-Deklaration angegeben!

ab Oracle8

- Der zuvor definierte Tabellentyp **Typ_Kontakt** kann nun wie folgt genutzt werden:

  Intern werden die geschachtelten Tabellen über eine vom System erzeugte Fremdschlüsselbeziehung verwaltet, die im Prinzip dem Auslagern von mehrwertigen Attributen zur Gewährleistung der 1CNF entspricht.

  Der Tabellenname dieser „attributiven Tabelle“ wird über die **store as-Klausel** definiert.

  Im Gegensatz hierzu können VArray-Typen immer nur als Ganzes gelesen oder gespeichert werden.

```
CREATE TABLE T_Kunde
OF Typ_Kunde(
      KNr primary key)
NESTED TABLE Kontakt
STORE AS Kunde_Kontakt_Liste;
```

ab Oracle8

- **Deklaration**
  Bei der Typdefinition werden *nur* die Methodennamen und Rückgabetypen definiert:

```
CREATE TYPE Typ_Kunde AS OBJECT (
   ...
   MEMBER FUNCTION Last_Bestellung
   RETURN date
);
```

- **Implementierung**
  Die Implementierung der Methoden erfolgt mittels PL/SQL getrennt (vgl. Folie 7 / 23).

- **Aufruf**
  Methoden vom Typ Function können innerhalb einer select-Anweisung wie Attribute genutzt werden:

```
select t.KNr, t.Last_Bestellung()
from T_Kunde t
where t.KNr = 123;
```

7.2.3 Referenzen auf Objekttypen – der Datentyp REF (1)

ab Oracle8

- Mit Hilfe von Objektidentifikatoren können Referenzen von Objekttabellen untereinander implementiert werden. Der Datentyp REF kann - in Analogie zur Fremdschlüssel-Definition - als Referenz auf Objekte in Objekttabellen eingesetzt werden.
- Jedes Auftragsobjekt referenziert genau ein Kundenobjekt:

```
CREATE TYPE Typ_Auftrag AS OBJECT (
   ANr        INTEGER,
   Eingang    DATE,
   Bearb      DATE,
   RefKunde REF Typ_Kunde
);
```

Die entsprechende Tabellendeklaration lautet:

```
CREATE TABLE T_Auftrag
OF Typ_Auftrag (
    ANr primary key);
```

ab Oracle8

- Andererseits wollen wir die Objekte unsere Tabelle T_Kunde referenzieren lassen auf Objekte der Auftragstabelle T_Auftrag, die basierend auf dem Objekttyp Typ_Auftrag definiert ist:
- Ein Kunde kann mehrere Aufträge haben, es ist also notwendig, in der Tabelle T_Kunde einen Kollektions-Datentyp zur Verfügung zu stellen, der die Objektreferenzen auf alle Auftragsobjekte *eines* Kundenobjektes verwaltet:

```
CREATE TYPE Typ_AuftragTab AS TABLE
OF REF Typ_Auftrag;
```

- Das neue Referenz-Attribut AuftragRef können wir nun in unserem Objekttyp Typ_Kunde verwenden:

```
CREATE TYPE Typ_Kunde AS OBJECT (
   ...
   RefAuftrag  Typ_AuftragTab
);
```

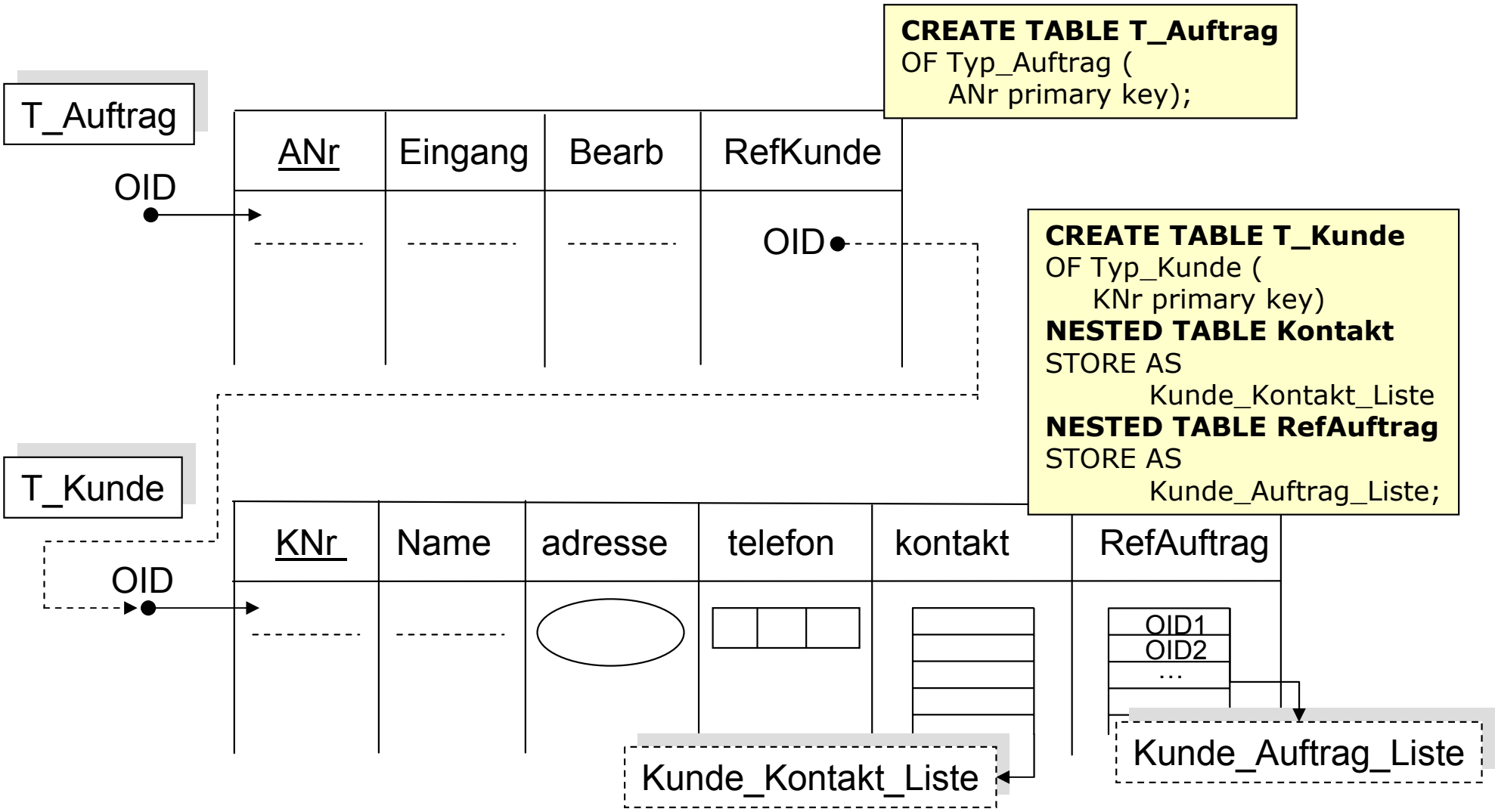
- Zur Vermeidung von „Kreis-Referenzen“ bei der Deklaration ist für alle ADT eine **Vorwärtsdeklaration** möglich:

```
CREATE TYPE Typ_Auftrag;
```

7.2.3 Die Tabellen T_Auftrag und T_Kunde ...

ab Oracle8

... sind Container für die Objekt-ADTs Typ_Auftrag bzw. Typ_Kunde:



ab Oracle8

- In Analogie zum this-Pointer (C++) bzw. der this-Objektreferenz (Java) kann innerhalb einer Methode auf die Attribute des aktuellen Objektes mit **self.<Attribut>** zugegriffen werden, wobei das Schlüsselwort self optional ist. self ist ein Parameter, der nicht notwendig explizit übergeben werden muss.

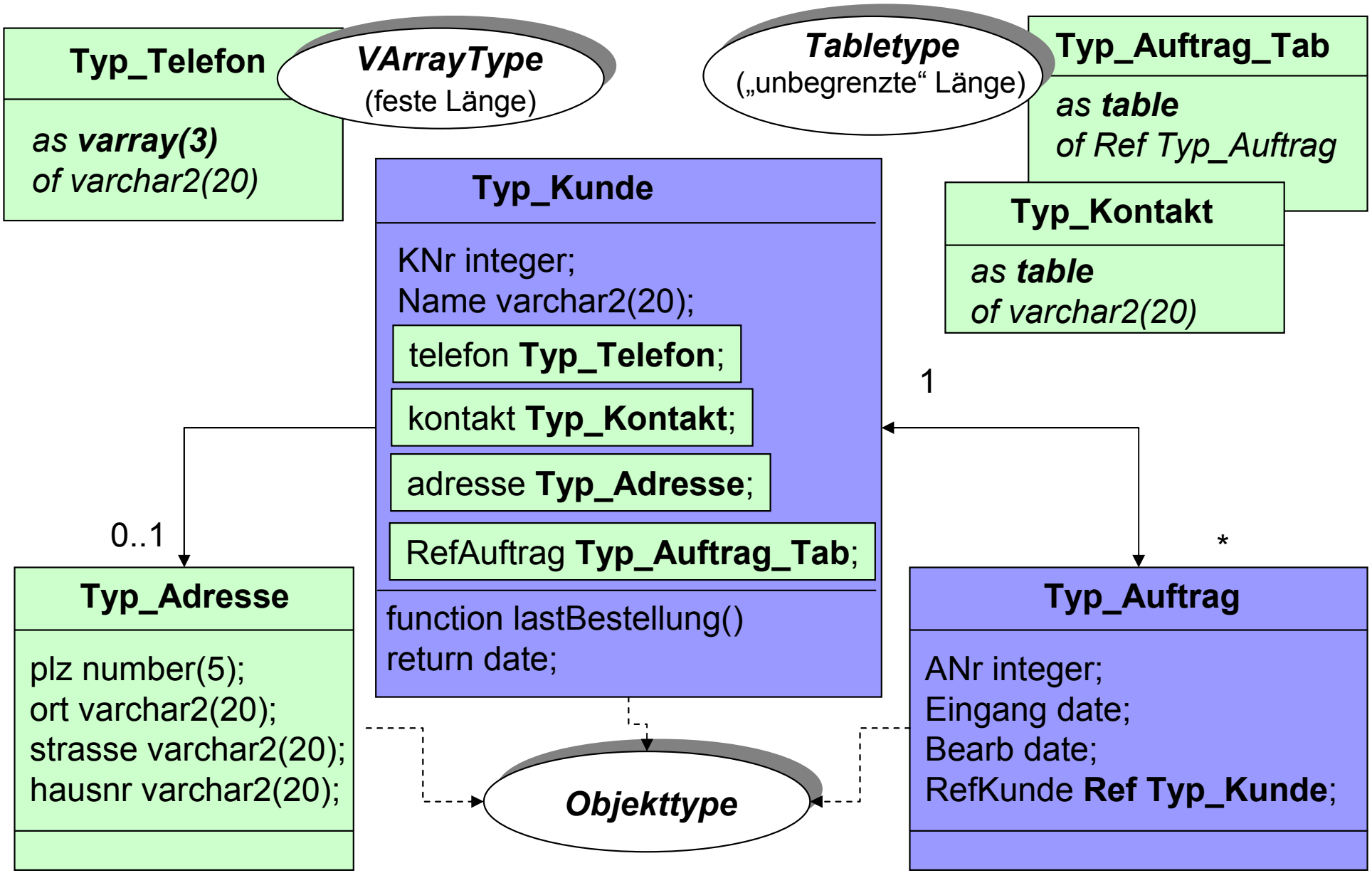
- Formulierung der Implementierung unter Verwendung der Objektreferenzen des Attributs *RefAuftrag* in *Typ_Kunde*:

```
CREATE OR REPLACE TYPE BODY Typ_Kunde AS
  MEMBER FUNCTION Last_Bestellung
  RETURN date is
            lastdate date;
  BEGIN
    SELECT MAX(a.column_value.Eingang) INTO lastdate
    FROM    table(self.RefAuftrag) a;
    RETURN lastdate;
  END;
 END;
/
```

→ zum Operator column_value vergleiche die Folie 7 / 35!

7.2.3 Objekt-relationales Klassendiagramm zur «Auftragsverwaltung»

ab Oracle8



ab Oracle8

- Beim **Einfügen von Tupeln** können die Werte entweder als *Konstanten*, oder über die *Berechnung eines* select-Blocks übergeben werden:

```
INSERT INTO T_Kunde (KNr,Name,Adresse,Telefon,Kontakt)
VALUES (8,'Schmidt',
            Typ_Adresse(64295,'DA','X-Str.','45'),
            Typ_Telefon('111','222',NULL),
            Typ_Kontakt('Kontaktmann 5','Kontaktmann 6')
            );
```

- Beim **Einfügen vonTupeln in innere Tabellen** verwendet man die **TABLE-Klausel** in Verbindung mit einer select-Anweisung. Die select-Anweisung hinter TABLE muss genau ein Tupel identifizieren und darf nur ein Attribut eines Tabellentyps selektieren:

```
INSERT INTO TABLE (
      SELECT Kontakt
      FROM T_Kunde
      WHERE KNr = 8
)
VALUES ('Kontaktmann 7');
```

7.2.3 Update (1)

ab Oracle8

- Belegen von *Referenzen* - einem bestimmten Auftrag wird ein Kunde zugewiesen:

```
UPDATE T_Auftrag a
SET    a.RefKunde = (SELECT REF(k)
                     FROM   T_Kunde k
                     WHERE  k.KNr = 10)
WHERE  a.Anr = 101;
```

- Belegen von *Objekten* - einem bestimmten Kunden wird eine Adresse zugewiesen:

```
UPDATE T_Kunde k
SET    k.Adresse =
       Typ_Adresse(55122,'MZ','X-Weg','20')
WHERE  k.name = 'Huber';
```

- Belegen von *Arrays* - einem bestimmten Kunden werden Telefonnummern zugewiesen:

```
UPDATE T_Kunde k
SET    k.Telefon =
       Typ_Telefon('T1','T2','T3')
WHERE  k.name = 'Huber';
```

ab Oracle8

- Belegen von *inneren Tabellen* - einem bestimmten Kunden werden Auftragsreferenzen zugewiesen:

```
UPDATE T_Kunde k
SET      k.RefAuftrag =
CAST(MULTISET (SELECT REF(a)
                      FROM   T_Auftrag a
                      WHERE a.RefKunde.knr = 10)
AS Typ_Auftragtab);
```

MULTISET berechnet eine Kollektion aus dem Ergebnis einer Select-Anweisung, der **CAST**-Operator wandelt das Ergebnis auf den geeigneten Datentyp.

ab Oracle8

- Das Löschen von *Objekten* und *Referenzen* innerhalb eines Tupels erfolgt dadurch, dass man ihnen mittels eines Updates den Wert NULL zuweist.

  Attribute vom Typ einer Kollektion können mit der leeren Kollektion oder mit NULL belegt werden.

  Alle Tupel einer *inneren Tabelle* lassen sich auch mit einer DELETE TABLE-Anweisung löschen:

```
DELETE TABLE (
        SELECT Kontakt
        FROM T_Kunde
        WHERE Name = 'Huber'
);
```

7.2.3 Select (1)

ab Oracle8

- Beim Lesen von Daten unterscheiden wir
  - *normale Tabellen* mit Tupeln,
  - *Objekttabellen* mit Objekten, die eine Objektidentität besitzen und die auch aus anderen Tabellen referenziert werden können, sowie
  - *Referenzen* auf Objekte, die selbst wieder manipulierbare Einheiten sind.

- Zugriff auf Tupel:

Syntax: **SELECT** <Var>.<Attribut$_1$>,...,<Var>.<Attribut$_n$>
FROM <Objekttabelle> <Var> ...;

Im Ergebnis einer solchen Anfrage werden die Attributwerte als Ergebnistupel (<Attrwert$_1$>,...,<Attrwert$_n$>) zusammengestellt.

Der Select-Ausdruck * steht nach wie vor als Platzhalter für alle Attribute der Tabelle.

ab Oracle8

- Jeder benutzerdefinierte Typ besitzt einen vordefinierten **Konstruktor**, der den selben Namen hat wie der Typ und, im Falle eines Objekttyps, die Objekttypstruktur als Parameterliste verwendet.
- Zu Tabellen- und VArray-Typen gibt es ebenfalls Konstruktoren um konstante Kollektionen zu erzeugen. Solche Konstruktoren werden z.B. bei der Anzeige von Attributen eines Typs verwendet:

```
SELECT k.KNr, k.name, k.adresse
FROM   T_Kunde k;
```

```
KNr   name   adresse(plz,ort,strasse,hausnr),
----- ------ ----------------------------------------
111   Xxx    Typ_Adresse(64295,'DA','A-Str.','45')
222   Yyy    Typ_Adresse(55122,'MZ','B-Str.','120g')
```

- Zugriff auf Referenzen:

Syntax: **SELECT REF**(<Var>)
FROM <Objekttabelle> <Var> ...;

Im Ergebnis dieser Anfrage werden die Referenzen auf die Objekte als Ergebnis angezeigt, also die Objektidentifikatoren (OID) in ihrer internen Darstellung (in HEX-Darstellung). Zum Dereferenzieren von Objektreferenzen kann die Funktion **DEREF** verwendet werden: sie wandelt einen OID in ein Objekt um.

ab Oracle8

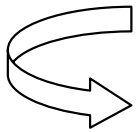
- Zugriff auf Objekte:

Syntax: **SELECT VALUE**(<Var>)
FROM <Objekttabelle> <Var> ...;

- Im Ergebnis dieser Anfrage werden die Objekte der Tabelle bei interaktiver Ausgabe am Bildschirm wie folgt ausgegeben, wobei alle Attribute der Objekttabelle auftreten:
<Objekttyp> (<Attrwert$_1$>,...,Attrwert$_n$>)

```
SELECT VALUE(k)
FROM   T_Kunde k;
```

```
Value(k)   (KNr name adresse(plz,ort,strasse,hausnr)),
---------- ---- ---- ----------------------------------------
Typ_Kunde (111 Xxx  typ_adresse(64295,'DA','A-Str.','45'),
Typ_Kunde (222 Yyy  typ_adresse(55122,'MZ','B-Str.','120g'),
```



```
          typ_telefon(),          typ_kontakt()     )
          ------------------------------------------
          typ_telefon(T1,T2,T3), typ_kontakt(K1)    )
          typ_telefon(T4,T5,T6), typ_kontakt(K2,K3))
```

ab Oracle8

- Durch Verknüpfung der **Tupelvariablen <Var>** mit unterschiedlichsten Objekten / Referenzen und Attributen durch den **Punkt-Operator** können beliebige **Pfadausdrücke** gebildet werden, die in Select- und Update-Operationen Verwendung finden können.
- Beispiele:

```
select  k.name, k.adresse
from    t_kunde k
where k.adresse.ort = 'DA';
```

```
name    adresse(plz,ort,strasse,hausnr),
------  ------------------------------------
Xxx     Typ_Adresse(64295,'DA','A-Str.','45')
```

```
select a.RefKunde
from   t_auftrag a;
```

→ liefert die OID für den referenzierten Kunden

```
select a.RefKunde.Knr
from   t_auftrag a;
```

→ liefert die Knr des referenzierten Kunden

7.2.3 Select (5)

ab Oracle8

```
select deref(a.RefKunde)
from t_auftrag a;
```

→ liefert alle Attribute eines Kundenobjektes

```
select a.ANr
from   t_auftrag a
where a.RefKunde.name='Schmidt';
```

→ liefert alle Auftragsnummern von Aufträgen des Kunden 'Schmidt'.

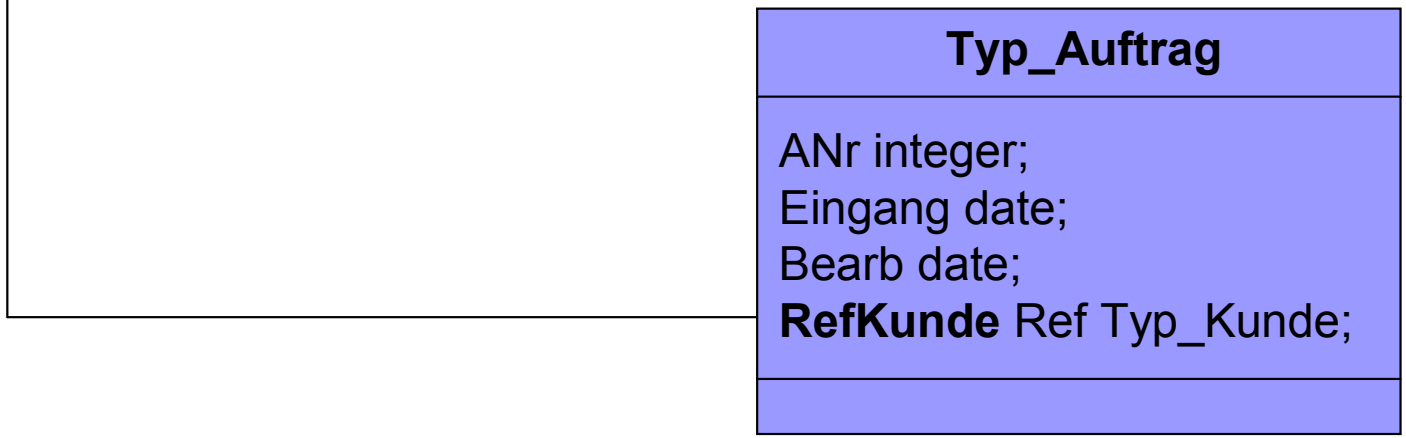
```
select a.ANr
from   t_auftrag a, T_Kunde k
where a.RefKunde = REF(k)
and    a.RefKunde.name='Schmidt';
```

ab Oracle8

- Dereferenzierung bei einfachen Referenzen:

```
-- Auftragsobjekte referenzieren ihr
-- Kundenobjekt

select a.ANr
from   t_auftrag a
where a.RefKunde.name='Schmidt';
```

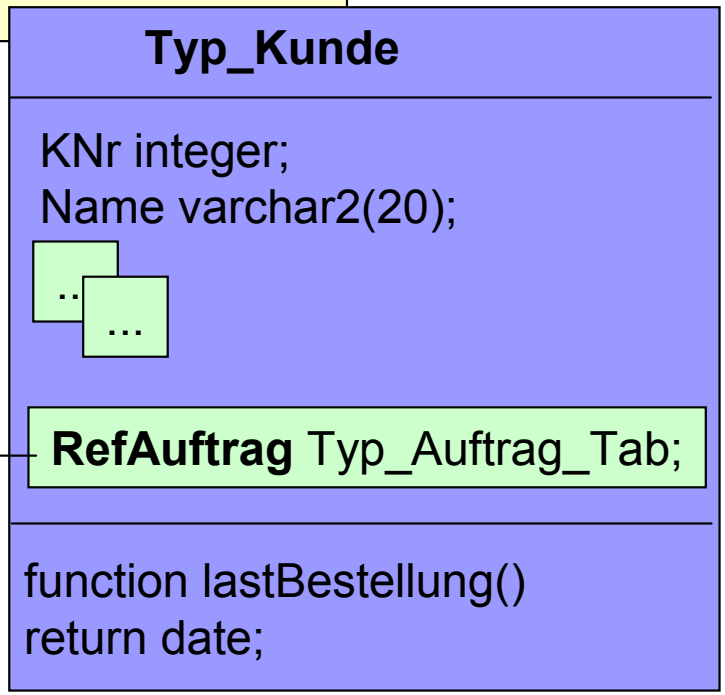


ab Oracle8

- Dereferenzierung bei Mehrfachreferenzen:

```
-- Kundenobjekte referenzieren ihre Auftragsobjekte

select a.column_value.anr
from   t_kunde k, table(k.RefAuftrag) a
where k.name = 'Schmidt';
```



- Die OIDs referenzierter Objekte im HEX-Code innerhalb eines Attributes vom Typ table_type können zeilenweise dereferenziert werden durch den Operator column_value:

<zeilenreferenz>.**column_value.**<Attribut>