Oracle: Abstrakte Datentypen:

Oracle bietet zwei mögliche Arten um abstrakte Datentypen zu implementieren:
Varying Array
Nested Table

Varying Array (kunde)

kdnr	kdname	gekaufteArt
1	Mustermann	1 4 5 8
2	Müller	2 5 6 8

Kunde beinhaltet in einem Attribut einen Array (hier: gekaufteArt) indem Daten innerhalb eines Feldes gespeichert werden.

Nested Table (kunde)

kdnr	kdname	lieferposition			
1	Mustermann	liefposnr	artnr	anzahl	preis
		0	1	2	7.99
		1	4	3	3.33
		2	5	2	1.23
		3	8	1	9.87
2	Müller	liefposnr	artnr	anzahl	preis
		0	2	6	0.34
		1	5	3	1.23
		2	6	9	3.40
		3	8	1	9.87

Kunde beinhaltet in einem Attribut (hier: lieferposition) eine eingebettete Tabelle

Erstellen eines Varying Array

Neuer Objekt Typ muss deklariert werden, der aus einem einfachen Datentyp besteht:

```
CREATE TYPE table_ty AS OBJECT (
     Attributname einfacherDatentyp)
```

Aus dem neuen Objekt Typ muss nun ein Varying Array erstellt werden:

```
CREATE OR REPLACE TYPE table_va AS VARRAY(x)
          OF table_ty
```

[das x in der VARRAY() Klausel gibt die maximale Anzahl an Werte innerhalb des Arrays an]

Im letzten Schritt kann der neu erstellte Datentyp in einer jeden Tabelle verwendet werden:

```
CREATE TABLE tablename (
     Attributname1 einfacherDatentyp,
     Attributname2 table_va
)
```

Für das obige Beispiel:

```
CREATE TYPE kunde_ty AS OBJECT (
     gekaufteArt Integer)
```

```
CREATE OR REPLACE TYPE kunde_va AS VARRAY(50)
          OF kunde_ty
```

```
CREATE TABLE kunde (
     kdnr Integer,
     kdname VarChar2(50),
     gekaufteArt kunde_va
)
```

INSERT

Einfügeoperationen in eine Tabelle mit Varying Array funktioniert mit Hilfe eines Konstruktor-Aufrufs.

```
INSERT INTO tablename (Attributname1,
     Attributname2) VALUES (Wert1,
     tablename_va(wert2, wert3, wert4, ..., wertN))
```

Für das Kundenbeispiel:

```
INSERT INTO kunde (kdnr, kdname, gekaufteArt)
VALUES (3,'Schmitz', kunde_va(5,6,7,8))
```

SELECT
Ein SELECT funktioniert wie ein herkömmliches SELECT, aber um eine TABLE()-Klausel erweitert

```
SELECT B.Attributname, A.* FROM tablename B,
      TABLE(B.tablename_va) A
```

Für das Kundenbeispiel:

```
SELECT B.kdnr, A.* FROM kunde B,
      TABLE(B.kunde_va) A
```

Erstellen eines Nested Table
Analog zum Varying Array muss zunächst ein neuer Objekt Typ erstellt werden.

```
CREATE TYPE table_ty AS OBJECT (
      Attributname einfacherDatentyp)
```

Als nächstes wird anstatt mit der varray-Klausel mit dem Schlüsselwort AS TABLE OF gearbeitet.

```
CREATE TYPE table_nt AS TABLE OF table_ty
```

Im letzten Schritt wird wieder die Tabelle angelegt, jedoch diesmal mit expliziter Angabe unter welchem Namen die geschachtelte Tabelle gespeichert werden soll. Man kann hier von Datenkapselung sprechen, da die Daten innerhalb dieser geschachtelten Tabelle nur über ihre übergeordnete Tabelle erreichbar sind. Die Nested Table wird unter dem Attributnamen AttributNameNT in A verwaltet.

```
CREATE TABLE tablename (
      Attributname1  einfacherDatentyp,
      AttributNameNT table_nt
) NESTED TABLE AttributNameNT STORE AS table_nt_tab
```

Beispiel für obige Kundentabelle:

```
CREATE TYPE kunde_ty AS OBJECT (
     liefposnr Integer,
     artnr     Integer,
     anzahl    Integer,
     preis     decimal(10,2)
)
```

```
CREATE TYPE kunde_nt AS TABLE OF kunde_ty
```

```
CREATE TABLE kunde (
     kdnr    Integer,
     kdname Varchar2(30),
     lieferposition kunde_nt
) NESTED TABLE lieferposition STORE AS kunde_nt_tab
```

INSERT

```
INSERT INTO tabname VALUES (attribut1,
     table_nt(table_ty(attribut2, attribut3)))
```

Kundenbeispiel:

```
INSERT INTO kunde VALUES (1,'Schmidt',
     kunde_nt(kunde_ty(1,13,2,3.56)))
```

Möchte man direkt mehrere Lieferpositionen für einen Kunden einfügen werden diese mit einem Komma separiert.

```
INSERT INTO kunde VALUES (1,'Schmidt',
     kunde_nt(kunde_ty(1,13,2,3.56),
           kunde_ty(2,14,9,27.89))
```

Möchte man noch keine Zeilen in den Nested Table einfügen, muss beim INSERT der Wertauf null gesetzt werden.

```
INSERT INTO kunde VALUES (1,'Schmidt',null)
```

Einfügen einer neuen Zeile in einen Nested Table

```
INSERT INTO TABLE (SELECT AttributNameNT FROM
     tablename WHERE attribut1 = wert) VALUES (
          table_ty(wert1, wert2,...))
```

Beispiel: Für Kunde 1 Lieferposition hinzufügen

```
INSERT INTO TABLE (SELECT lieferposition FROM
     kunde WHERE kdnr = 1) VALUES (
          kunde_ty(4,3,5,2.9))
```

SELECT

Zeilen eines Nested Table lassen sich über die TABLE()-Klausel abfragen. Die TABLE()-Klausel ist Bestandteil der FROM-Klausel (wie beim natürlichen JOIN, wo eine zweite Tabelle B Bestandteil der FROM-Klausel war). In der TABLE-Klausel wird der symbolische Tabellenname B mit der NESTED TABLE von A verbunden, die in A unter dem Attributnamen AttributNameNT verwaltet wird.

```
SELECT B.* FROM tablename A,
     TABLE (A.AttributNameNT) B
     WHERE A.Attribut = Wert
```

Beispiel 1: Anzeige aller Lieferpositionen des Kunden 2

```
SELECT B.* FROM kunde A,
     TABLE (A.lieferposition) B
     WHERE A.kdnr = 2
```

Beispiel 2: Anzeige aller Lieferpositionen des Kunden 2 mit Kundendaten

```
SELECT B.*, A.kdnr, A.kdname FROM kunde A,
     TABLE (A.lieferposition) B
     WHERE A.kdnr = 2
```

JDBC-Beispiel für Nested Table Anfrage

```
public static void main(String args[])throws IOException{

    // Datenbankverbindung erstellen mit der "connect()"-Methode
    try {
      Connection con               = connect();

      // Vorbereitung der JDBC Objekte
      Statement Stmt;
      ResultSet RS;

      String SQL;
      String kdname;
      int kdnr, liefposnr, artnr, anzahl;
      double preis;

      // Erezeugen eines Statements aus der DB-Verbindung
      Stmt = con.createStatement();

      // Eine SQL Select Anweisung
      SQL  = "SELECT A.kdnr, A.kdname, B.* FROM kunde A,
            TABLE (A.lieferposition ) B
            WHERE A.kdnr = 2";

      // SQL-Anweisung ausführen und Ergebnis in ResultSet
      // schreiben
      RS   = Stmt.executeQuery(SQL);

      // Das ResultSet Datensatzweise durchlaufen
      while(RS.next()){
        kdnr            = RS.getInt("kdnr");
        kdname          = RS.getString("kdname");
        liefposnr       = RS.getInt("liefposnr");
        artnr           = RS.getInt("artnr");
        anzahl          = RS.getInt("anzahl");
        preis           = RS.getDouble("preis");

        System.out.println("Kundennr. : "+ kdnr);
        System.out.println("Kundenname: "+ kdname);
        System.out.println("Lieferpositionsnr: "+liefposnr);
        System.out.println("Artikelnr: "+artnr);
        System.out.println("Anzahl: "+anzahl);
        System.out.println("Preis: "+preis);
      }
      ...
    } catch (SQLException e){}
}
```