

# 4. Objekt-relationale DBS

- SQL-Standardisierung
- SQL-Typsystem
  - Typkonstruktor ROW
  - Typkonstruktor ARRAY, UNNEST-Operation
  - Typkonstruktur MULTISSET
  - benutzerdefinierte Typen und Funktionen (UDTs, UDFs)
  - DISTINCT-Typen
  - strukturierte Datentypen, typisierte Tabellen , REF-Typ
  - UDT-Kapselung
- Generalisierung /Spezialisierung
  - Typ-/ Tabellenhierarchien (Subtypen, Subtabellen)
  - dynamisches Binden



## Objekt-relationale DBS (ORDBS)

- Erweiterung des relationalen Datenmodells und SQL um Objekt-Orientierung
- komplexe, nicht-atomare Attributtypen
- Erweiterbarkeit
  - benutzerdefinierte Datentypen
  - benutzerdefinierte Funktionen
- Bewahrung der Grundlagen relationaler DBS
  - deklarativer Datenzugriff mit SQL
  - Sichtkonzept etc.
- Standardisierung beginnend mit SQL:1999



# SQL-Standardisierung

1986 [SQL86](#)

- keine Integritätszusicherungen

1989 [SQL89](#) (120 Seiten)

- Basiskonzept der Referentiellen Integrität (Referenzen auf Primärschlüssel und Schlüsselkandidaten)

1992 [SQL92](#) (SQL2)

- Entry Level: ~ SQL89 + geringfügige Erweiterungen und Korrekturen
- Intermediate Level: Dynamic SQL, Join-Varianten, Domains ...
- Full Level (580 Seiten): Subquery in CHECK, Assertions, DEFERRED ...

1996 Nachträge zu SQL-92:

Call-Level-Interface, Persistent Stored Modules (Stored Procedures)

1999 [SQL:1999](#) (SQL3), ca. 3000 Seiten, mit Objekt-Erweiterungen

[SQL:2003](#), [SQL:2008](#), [SQL:2011](#), [SQL:2016](#) : XML-Unterstützung (2003, 2006);

temporale DBS (2011), JSON (2016)

diverse Erweiterungen, z.B. MERGE, OLAP-Abfragen, INSTEAD-Trigger



## Aufbau des SQL-Standards

Part 1: [SQL/Framework](#) (beschreibt Aufbau des Standards)

Part 2: [SQL/Foundation](#): Kern-SQL, objekt-relationale Erweiterungen, Trigger, ...

Part 3: [SQL/CLI](#): Call Level Interface

Part 4: [SQL/PSM](#): Persistent Storage Modules

Part 9: [SQL/MED](#): Management of External Data

Part 10: [SQL/OLB](#): Object Language Bindings (SQLJ)

Part 11: [SQL/Schemata](#): Information and Definition Schemas

Part 13: [SQL/JRT](#): SQL Routines and Types using Java

Part 14: [SQL/XML](#): XML-related Specifications

■ separater Standard [SQL/MM](#) (SQL Multimedia and Application Packages) mit derzeit sechs Teilen

- Framework, Full Text, Spatial, Still-Image, Data Mining, History



# SQL-Typsystem

## ■ erweiterbares Typkonzept seit SQL:1999

- vordefinierte Datentypen (inkl. Boolean, BLOB, CLOB)
- konstruierte Typen (**Konstruktoren**):
  - Tupel-Typen (ROW-Typ)
  - Kollektionstypen ARRAY und MULTISET (SQL:2003)
  - REF-Typ
- benutzerdefinierte Datentypen (**User-Defined Types, UDT**):  
**Distinct Types** und **Structured Types**

## ■ UDTs

- Definition unter Verwendung von vordefinierten Typen, konstruierten Typen und vorher definierten UDTs
- unterstützen Kapselung, Vererbung (Subtypen) und Overloading

## ■ alle Daten werden weiterhin innerhalb von Tabellen gehalten

- Definition von Tabellen auf Basis von strukturierten UDTs möglich
- Bildung von Subtabellen (analog zu UDT-Subtypen)



## Tupel-Typen (ROW-Typen)

### ■ Tupel-Datentyp (row type)

- Sequenz von Feldern (fields), bestehend aus Feldname und Datentyp:  
ROW (<feldname1> <datentyp1>, <feldname2> <datentyp2>, ...)
- eingebettet innerhalb von Typ- bzw. Tabellendefinitionen

### ■ Beispiel

```
CREATE TABLE Pers ( PNR          int ,
                    Name        ROW ( VName    VARCHAR (20) ,
                                      NName    VARCHAR (20) ) ,
                    ... );

ALTER TABLE Pers
  ADD COLUMN Anschrift ROW ( Strasse  VARCHAR (40) ,
                             PLZ      CHAR (5) ,
                             Ort      VARCHAR (40) );
```

### ■ geschachtelte Rows möglich



## ROW-Typen (2)

### ■ Operationen

- Erzeugung mit Konstruktor ROW:

```
INSERT INTO PERS (PNR, Name)
VALUES (123, ROW („Peter“, „Meister“))
```

- Zugriff auf Tupelfeld mit Punktnotation:

```
SELECT * FROM Pers
WHERE Name.NName = „Meister“
```

### ■ Vergleiche

```
ROW (1, 2) < ROW (2, 2)
```

```
ROW (2, 5) < ROW (2, 1)
```



## ARRAY-Kollektionstyp

### ■ Spezifikation: <Elementtyp> ARRAY [<maximale Kardinalität>]

- Elementtypen: alle Datentypen (z.B. Basisdatentypen, benutzerdefinierte Typen)
- geschachtelte (mehrdimensionale) Arrays erst ab SQL:2003

```
CREATE TABLE Mitarbeiter
(PNR          int,
 Name         ROW (VName VARCHAR (20),
                  NName VARCHAR (20)),
 Sprachen     VARCHAR(15) ARRAY [8], ... )
```



# ARRAY (2)

## ■ Array-Operationen

- Typkonstruktor ARRAY
- Element-Zugriff direkt über Position oder deklarativ (nach Entschachtelung)
- Bildung von Sub-Arrays, Konkatenation (||) von Arrays
- CARDINALITY
- UNNEST (Entschachtelung; wandelt Kollektion in Tabelle um)

```
INSERT INTO Mitarbeiter (PNR, Name, Sprachen)
VALUES ( 1234, ROW(„Peter“, „Meister“), ARRAY[„Deutsch“,„Englisch“])
```

```
UPDATE Mitarbeiter
SET Sprachen[3]=„Französisch“
WHERE Name.NName=„Meister“
```



## UNNEST-Operation

### ■ Umwandlung einer Kollektion (Array, Multiset) in Tabelle

```
UNNEST (<Kollektionsausdruck>) [WITH ORDINALITY]
```

- Verwendung innerhalb der From-Klausel

### ■ Anwendbarkeit von Select-Operationen

### ■ Beispiele

```
SELECT *
FROM UNNEST (ARRAY [1,2,3]) A (B)
```

*Welche Mitarbeiter sprechen französisch?*

```
SELECT
FROM Mitarbeiter
WHERE
```



# UNNEST-Operation (2)

## ■ Beispiele

*Welche Sprachen kennt der Mitarbeiter „Meister“?*

```
SELECT S.*
FROM Mitarbeiter AS M, UNNEST (M.Sprachen) AS S (Sprache)
WHERE M.Name.NName="Meister"
```

## ■ Ausgabe der Position innerhalb der Kollektion mit **Ordinality-Klausel**

```
SELECT S.*
FROM Mitarbeiter M,
      UNNEST(M.Sprachen)S(Sprache,Pos)WITH ORDINALITY
WHERE M.Name.NName="Meister"
```



# MULTISET-Kollektionstyp

## ■ Spezifikation: <Elementtyp> MULTISET

- Elementtypen: alle Datentypen inklusive ROW, ARRAY und MULTISET
- beliebige Schachtelung möglich

```
CREATE TABLE ABT (  AName      VARCHAR(30), ...
                   AOrte   VARCHAR(30)  MULTISET,
                   Mitarbeiter ROW (Name VARCHAR(30),
                                   Beruf VARCHAR(30)) MULTISET)
```

## ■ MULTISET-Operationen

- Typkonstruktor MULTISET:
  - MULTISET()
  - MULTISET (<Werteliste>)
- Konversion zwischen Multimengen und Tabellen:
  - UNNEST (<Multimenge>) bzw.
  - MULTISET (<Unteranfrage>)
- CARDINALITY



# MULTISET (2)

## ■ Weitere MULTISET-Operationen

- Duplikateliminierung über SET
- Duplikattest: <Multimenge> IS [NOT] A SET
- Mengenoperationen mit/ohne Duplikateliminierung:

<Multimenge1> **MULTISET** {**UNION** | **EXCEPT** | **INTERSECT**}  
[**DISTINCT** | **ALL**] <Multimenge2>

- Elementextraktion (für 1-elementige Multimenge):  
**ELEMENT** (MULTISET(17))
- Elementtest: <Wert> [NOT] **MEMBER** [OF] <Multimenge>
- Inklusionstest: <Multimenge1> [NOT] **SUBMULTISET** [OF] <Multimenge2>

## ■ Beispiel-Query: Welche Leipziger Abteilungen haben mehr als 20 Mitarbeiter?

```
SELECT AName
FROM ABT
WHERE
```



## Syntax der UDT-Definition (vereinfacht)

```
CREATE TYPE <UDT name>[[<subtype clause>][AS <representation>]
    [<instantiateable clause>]<finality>[<reference type spec>]
    [<cast option>] [<method specification list>]
<subtype clause> ::= UNDER <supertype name>           -- max. 1 Supertyp
<representation> ::= <predefined type> | [(<member>, ... ) ]

<instantiateable clause> ::= INSTANTIABLE | NOT INSTANTIABLE
<finality> ::= FINAL | NOT FINAL
<member> ::= <attribute definition>
<method spec> ::= <original method spec> | <overriding method spec>
<overriding method spec> ::= OVERRIDING <partial method spec>
<partial method spec> ::= [ INSTANCE | STATIC | CONSTRUCTOR ]
    METHOD <routine name> <SQL parameter list>
    <returns clause>

DROP TYPE <UDT name> [RESTRICT | CASCADE ]
```



# DISTINCT-Typen (Umbenannte Typen)

- Wiederverwendung vordefinierter Datentypen unter neuem Namen
  - einfache UDT, keine Vererbung (FINAL)
  - DISTINCT-Typen sind vom darunter liegenden (und verdeckten) Basis-Typ verschieden

```
CREATE TYPE Dollar AS REAL FINAL;
CREATE TYPE Euro AS REAL FINAL;

CREATE TABLE Dollar_SALES ( Custno INTEGER, Total Dollar, ...)
CREATE TABLE Euro_SALES ( Custno INTEGER, Total Euro, ...)

SELECT D.Custno
FROM Dollar_SALES D, Euro_SALES E
WHERE D.Custno = E.Custno AND D.TOTAL > E.TOTAL
```

- keine direkte Vergleichbarkeit mit Basisdatentyp (Namensäquivalenz)
- Verwendung von Konversionsfunktionen zur Herstellung der Vergleichbarkeit



## Strukturierte Typen: Beispiel

```
CREATE TYPE AdressTyp AS (Strasse VARCHAR (40),
                        PLZ CHAR (5),
                        Ort VARCHAR (40) ) NOT FINAL;

CREATE TYPE PersonT AS
(Name VARCHAR (40), Gehalt REAL, PNR int,
Anschritt AdressTyp,
Manager REF (PersonT),
Kinder REF (PersonT) ARRAY [10] )
INSTANTIABLE NOT FINAL
INSTANCE METHOD Einkommen ( ) RETURNS REAL;

CREATE TABLE Mitarbeiter OF PersonT
(Manager WITH OPTIONS SCOPE Mitarbeiter ... )

CREATE METHOD Einkommen() FOR PersonT
BEGIN RETURN Gehalt;
END;

SELECT Name, Anschrift.Ort
FROM Mitarbeiter
WHERE Einkommen() > 50000.0
```





# Typisierte Tabellen

```
CREATE TABLE Tabellename OF StrukturierterTyp [UNDER Supertabelle]
  [( [ REF IS oid USER GENERATED |
      REF IS oid SYSTEM GENERATED |
      REF IS oid DERIVED (Attributliste) ]
    [Attributoptionsliste] ) ]
```

Attributoption: Attributname **WITH OPTIONS** Optionsliste

Option: **SCOPE** TypisierteTabelle | **DEFAULT** Wert | Integritätsbedingung

- Tabellen: einziges Konzept (container), um Daten persistent zu speichern
- Typ einer Tabelle kann durch strukturierten Typ festgelegt sein: typisierte Tabellen (**Objekttabellen**)
  - Zeilen entsprechen Instanzen (Objekten) des festgelegten Typs
  - OIDs systemgeneriert, benutzerdefiniert oder aus Attribut(en) abgeleitet
- Bezugstabelle für REF-Attribute erforderlich (**SCOPE-Klausel**)
- Attribute können Array-/Multiset-, Tupel-, Objekt- oder Referenz-wertig sein



## REF-Typen

- dienen zur Realisierung von Beziehungen zwischen Typen bzw. Tupeln (OID-Semantik)

```
<reference type> ::= REF (<user-defined type>)[SCOPE <table name>]
  [REFERENCES ARE [NOT] CHECKED] [ON DELETE <delete_action> ]
<delete_action> ::= NO ACTION | RESTRICT | CASCADE | SET NULL | SET DEFAULT
```

  - jedes Referenzattribut muss sich auf genau eine Tabelle beziehen (SCOPE-Klausel)
  - nur typisierte Tabellen (aus strukturierten UDT abgeleitet) können referenziert werden
  - nur Top-Level-Tupel in Tabellen können referenziert werden

### ■ Beispiel

```
CREATE TABLE Abteilung OF AbteilungT;
```

```
CREATE TABLE Person(PNR          INT,
                    Name          VARCHAR (40),
                    Abt           REF (AbteilungT) SCOPE Abteilung,
                    Manager       REF (PersonT)   SCOPE Mitarbeiter,
                    Anschrift     AdressTyp, ... );
```



## REF-Typen (2)

- Dereferenzierung mittels **DEREF**-Operator (liefert alle Attributwerte des referenzierten Objekts)

```
SELECT  Deref (P.Manager).Name
FROM    Person P
WHERE   P.Name= "Meister"
```

- Kombination von Dereferenzierung und Attributzugriff: ->

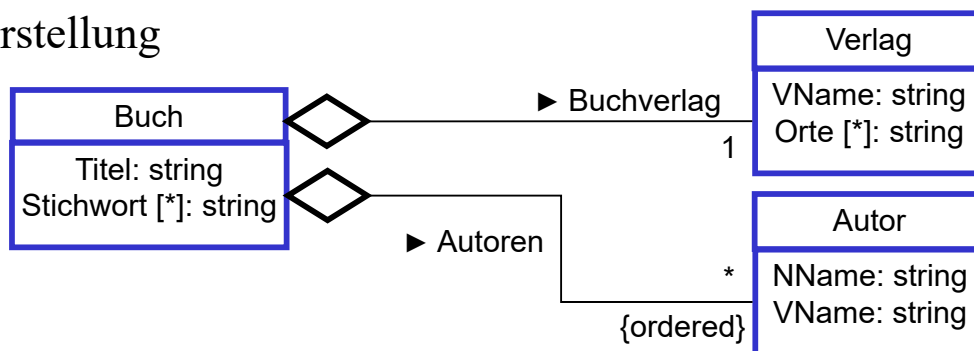
– Realisierung von **Pfadausdrücken**

```
SELECT P.Name, P.Abt -> AName
FROM Person P
WHERE P.Manager -> Name = "Schmidt" AND
      P.Anschrift.Ort = "Leipzig"
```



## Beispiel aus Kap. 3

- UML-Darstellung



- Objektrelationale Umsetzung mit SQL

```
CREATE TYPE AutorT(NName VARCHAR(40), VName VARCHAR(40)) NOT FINAL
CREATE TYPE VerlagsT (VName VARCHAR(40),
                     Orte VARCHAR (40) MULTISET) NOT FINAL
CREATE TABLE Autor OF AutorT;
CREATE TABLE Verlag OF VerlagsT;
CREATE TABLE BUCH(Titel VARCHAR(40),
                  Stichwort VARCHAR (40) MULTISET,
                  Buchverlag REF (VerlagsT) SCOPE Verlag,
                  Autoren REF (AutorT) SCOPE Autor ARRAY[20])
```



# UDT-Kapselung

- Kapselung: sichtbare UDT-Schnittstelle besteht aus Menge von Methoden
- auch Attributzugriff erfolgt ausschließlich über Methoden
  - für jedes Attribut werden implizit Methoden zum Lesen (Observer) sowie zum Ändern (Mutator) erzeugt
  - keine Unterscheidung zwischen Attributzugriff und Methodenaufruf
- implizit erzeugte Methoden für UDT AdressTyp

<b>Observer-Methoden:</b>	METHOD	Strasse ()	RETURNS VARCHAR (40);
	METHOD	PLZ ()	RETURNS CHAR (5);
	METHOD	Ort ()	RETURNS VARCHAR (40);
<b>Mutator-Methoden:</b>	METHOD	Strasse (VARCHAR (40))	RETURNS AdressTyp;
	METHOD	PLZ (CHAR(5))	RETURNS AdressTyp;
	METHOD	Ort (VARCHAR (40))	RETURNS AdressTyp;
- Attributzugriff wahlweise über Methodenaufruf oder Punkt-Notation (.)
  - a.x ist äquivalent zu a.x () , z.B. SELECT Anschrift.Ort() FROM ..
  - SET a.x = y ist äquivalent zu a.x (y), z.B. Anschrift.Ort („Leipzig“)



## Initialisierung von UDT-Instanzen

- DBS stellt Default-Konstruktor für instantiierbare UDTs bereit

```
CONSTRUCTOR METHOD PersonT () RETURNS PersonT
```

  - parameterlos, kann nicht überschrieben werden
  - besitzt gleichen Namen wie zugehöriger UDT
  - belegt jedes der UDT-Attribute mit Defaultwert (falls definiert)
  - Aufruf mit **NEW**
- Benutzer kann eigene Konstruktoren definieren, z.B. für Objektinitialisierungen (über Parameter)

```
CREATE CONSTRUCTOR METHOD PersonT (n varchar(40), a AdressTyp) FOR
PersonT RETURNS PersonT
BEGIN
    DECLARE p PersonT;
    SET p = NEW PersonT();
    SET p.Name = n;
    SET p.Anschrift = a;
    RETURN p;
END;

INSERT INTO Pers VALUES (NEW PersonT ("Peter Schulz", NULL))
```



# Generalisierung / Spezialisierung

- Spezialisierung in Form von Subtypen und Subtabellen
- nur Einfachvererbung (strukturierter Typ hat max. 1 Supertyp)
  - Supertyp muss auch strukturierter Typ sein
- Subtyp
  - erbt alle Attribute und Methoden des Supertyps
  - kann eigene zusätzliche Attribute und Methoden besitzen
  - Methoden von Supertypen können überladen werden (Overriding)
- Super-/Subtabellen sind typisierte Tabellen von Super-/Subtypen
- Instanz eines Subtyps kann in jedem Kontext genutzt werden, wo Supertyp vorgesehen ist (Substituierbarkeit)
  - Supertabellen enthalten auch Tupel von Subtabellen
  - Subtabellen sind Teilmengen von Supertabellen



## Subtypen / Subtabellen: Beispiel

```
CREATE TYPE PersonT AS (PNR INT, Name CHAR (20), Grundgehalt REAL, ...)
NOT FINAL
CREATE TYPE Techn-AngT UNDER PersonT AS (Techn-Zulage REAL, ...) NOT FINAL
CREATE TYPE Verw-AngT UNDER PersonT AS ( Verw-Zulage REAL, ...) NOT FINAL

CREATE TABLE Pers OF PersonT (PRIMARY KEY PNR)
CREATE TABLE Techn-Ang OF Techn_AngT UNDER Pers
CREATE TABLE Verw-Ang OF Verw-AngT UNDER Pers
INSERT INTO Pers VALUES (NEW PersonT (8217, 'Hans', 40500 ...))
INSERT INTO Techn-Ang VALUES (NEW Techn-AngT (NEW PersonT (5581, 'Rita',
...), 2300))
INSERT INTO Verw-Ang VALUES (NEW Verw-AngT (NEW PersonT (3375, 'Anna', ...),
3400))
```

heterogener Aufbau von Supertabellen, z.B. **PERS**:

PNR	Name	Techn-Zulage	Verw-Zulage
8217	Hans ...		
5581	Rita ...	2300	
3375	Anna ...		3400
...			



# Subtypen / Subtabellen: Anfrageeinschränkungen

## ■ Anfrageeinschränkungen auf homogene Ergebnismengen

- Zugriff auf Subtabellen (auf Blattebene)
- **ONLY**-Prädikat zur Einschränkung auf eine Tabelle (einen Typ) ohne Instanzen in Subtabellen
- **IS-OF**-Prädikat (bzw. IS OF ONLY) zur Einschränkung auf einen Subtyp

```
SELECT *  
FROM ONLY Pers  
WHERE Grundgehalt > 40000
```

```
SELECT Name  
FROM Pers  
WHERE Anschrift IS OF German-Address
```

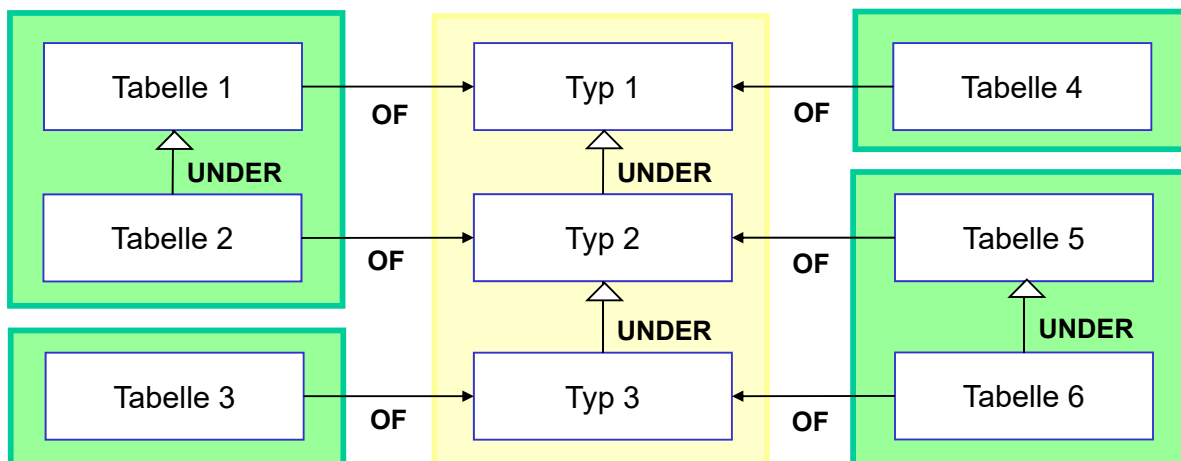
(Annahme: AddressTyp von Anschrift habe Subtypen German-Address etc.)



## Subtypen vs. Subtabellen

### ■ Typ- und Tabellenhierarchien müssen nicht 1:1 korrespondieren

- Typ einer Subtabelle (z.B. Tab. 2 und 6) muss direkter Subtyp des Typs der direkten Supertabelle sein
- nicht zu jedem strukturierten Typ muss (Objekt-)Tabelle existieren
- strukturierter Typ kann als Tabellentyp mehrerer (unabhängiger) Objekttabellen dienen
- Typ einer Wurzeltabelle muss nicht Wurzeltyp sein (z.B. Tab. 3/Typ 3)
- Typ einer Objekttable ohne Subtabellen kann Subtypen haben (z.B. Tab.2/Typ 2)



# Dynamisches Binden

- Overloading (Polymorphismus) von Funktionen und Methoden wird unterstützt
  - dynamische Methodenauswahl zur Laufzeit aufgrund spezifischem Typ
- Anwendungsbeispiel: polymorphe Methode Einkommen

```
CREATE TYPE PersonT AS (PNR INT, ... ) NOT FINAL
    METHOD Einkommen () RETURNS REAL, ...
CREATE TYPE Techn-AngT UNDER PersonT AS (Techn-Zulage REAL, ...)
    NOT FINAL
    OVERRIDING METHOD Einkommen () RETURNS REAL, ...
CREATE TYPE Verw-AngT UNDER PersonT AS (Verw-Zulage REAL, ...)
    NOT FINAL
    OVERRIDING METHOD Einkommen () RETURNS REAL,
```

```
CREATE TABLE Pers OF PersonT (...)
```

```
SELECT P.Einkommen()
FROM Pers P
WHERE P.Name = 'Anna';
```



# Zusammenfassung

- SQL-Standardisierung von objekt-relationen DBS
  - Kompatibilität mit existierenden SQL-Systemen + Objektorientierung
  - Unterstützung von Objekt-Identität (REF-Typen)
- erweiterbares Typsystem: signifikante Verbesserung der Modellierungsfähigkeiten
  - benutzerdefinierte Datentypen und Methoden (UDT, UDF)
  - DISTINCT Types
  - ROW: Tupel-Konstruktor
  - Kollektionstypen ARRAY und MULTISSET
- Typhierarchien und Einfach-Vererbung: Subtypen vs. Subtabellen

