5. Objekt-relationale DBS, SQL:2003

- SQL-Standardisierung
- Large Objects: BLOBS, CLOBS
- Typkonstruktoren
  - ROW, ARRAY
  - UNNEST-Operation
  - MULTISET
- Benutzerdefinierte Typen und Funktionen (UDTs, UDFs)
  - DISTINCT-Typen
  - Strukturierte Datentypen, Typisierte Tabellen
  - REF-Typ
- Typhierarchien / Tabellenhierarchien (Subtypen, Subtabellen)
- Rekursive Anfragen

Objektrelationale DBS (ORDBS)

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

SQL-Standardisierung

1986 SQL86

- DDL- und DML-Anweisungen
- keine Integritätszusicherungen

1989 SQL89 (120 Seiten)

- Revision von SQL86
- 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

2003 SQL:2003

Aufbau des SQL:2003-Standards

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

Part 2: SQL/Foundation: objektrelationale 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 fünf Teilen
  - Framework, Full Text, Spatial, Still-Image, Data Mining

Typsystem von SQL:1999 und SQL:2003

- erweiterbares Typkonzept
  - vordefinierte Datentypen
  - konstruierte Typen (Konstruktoren): REF, Tupel-Typen (ROW-Typ), Kollektionstypen ARRAY, MULTISET (SQL:2003)
  - 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)
- neue vordefinierte Datentypen: Boolean, Large Objects

Large Objects

- Verwaltung großer Objekte im DBS (nicht in separaten Dateien)
  - umgeht große Datentransfers und Pufferung durch Anwendung
  - Zugriff auf Teilbereiche
- 3 neue Datentypen:
  - BLOB (Binary Large Object)
  - CLOB (Character Large Object): Texte mit 1-Byte Character-Daten
  - NCLOB (National Character Large Objects): 2-Byte Character-Daten für nationale Sonderzeichen (z. B. Unicode)

```
CREATE TABLE  Pers  (PNR          INTEGER,
                     Name         VARCHAR (40),
                     Vollzeit     BOOLEAN,
                     Lebenslauf   CLOB (75K),
                     Unterschrift BLOB (1M),
                     Bild         BLOB (12M))
```

Large Objects (2)

- indirekte Verarbeitung großer Objekte über Locator-Konzept (ohne Datentransfer zur Anwendung)
- unterstützte Operationen
  - Suchen und Ersetzen von Werten (bzw. partiellen Werten)
  - LIKE-Prädikate, CONTAINS, POSITION, SIMILAR TO *„SQL (1999 | 2003)“*
  - Konkatenation ||, SUBSTRING, LENGTH, IS [NOT] NULL ...

```
Bsp.:   SELECT Name FROM Pers
        WHERE  CONTAINS (
         AND POSITION (
```

- einige Operationen sind auf LOBs nicht möglich
  - Schlüsselbedingung
  - Kleiner/Größer-Vergleiche
  - Sortierung (ORDER BY, GROUP BY)

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:

    ROW („Peter“, „Meister“)

  - Zugriff auf Tupelfeld mit Punktnotation:

    ```
    SELECT * FROM Pers
    WHERE
    ```

  - Vergleiche

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

ARRAY-Kollektionstyp

- in SQL:1999 wird nur Kollektionstyp ARRAY unterstützt
  - kein Set, List ...
- Spezifikation: <Elementtyp> ARRAY [<maximale Kardinalität>]
  - Elementtypen: alle Datentypen (z.B. Basisdatentypen, benutzerdefinierte Typen)
  - geschachtelte (mehrdimensionale) Arrays erst ab SQL:2003

```
CREATE TABLE Mitarbeiter
            (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 (Name, Sprachen)
VALUES ( 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

  Welche Sprachen kennt der Mitarbeiter „Meister“?

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

  Welche Mitarbeiter sprechen französisch?

  SELECT
  FROM Mitarbeiter
  WHERE

- 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 (SQL:2003)

- 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
  - Duplikatelminierung 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>

Syntax der UDT-Definition (vereinfacht)

CREATE TYPE <UDT name> [[<subtype clause>] [AS <representation>]
[<instantiable clause>] <finality> [<reference type specification>]
[<cast option>] [<method specification list>]

<subtype clause> ::= UNDER <supertype name>

<representation> ::= <predefined type> | [**(** <member> , ... **)**]

<instantiable clause> ::= INSTANTIABLE | NOT INSTANTIABLE

<finality> ::= FINAL | NOT FINAL

<member> ::= <attribute definition>

<method spec> ::= <original method spec> | <overriding method spec>

<original method spec> ::= <partial method spec> <routine characteristics>

<overriding method spec> ::= OVERRIDING <partial method spec>

<partial method spec> ::= [INSTANCE | STATIC | CONSTRUCTOR]
METHOD <routine name> <SQL parameter declaration 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
```

- keine direkte Vergleichbarkeit mit Basisdatentyp (Namensäquivalenz)
- Verwendung von Konversionsfunktionen zur Herstellung der Vergleichbarkeit (CAST)
  - UPDATE Dollar_SALES SET Total = Total * 1.16

Strukturierte Typen: Beispiel

```
CREATE TYPE AdressTyp
            (Strasse   VARCHAR (40),
             PLZ       CHAR (5),
             Ort       VARCHAR (40) ) NOT FINAL;
CREATE TYPE PersonT
            (Name                   VARCHAR (40),
            Anschrift               AdressTyp,
            PNR                     int,
            Manager                 REF (PersonT),
            Gehalt                  REAL,
            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;
```

Typisierte Tabellen

CREATE TABLE Tabellenname 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 sytemgeneriert, 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)
FROM        Person P
WHERE  P.Name= “Meister”
```

- Kombination von Dereferenzierung und Attributzugriff: -> (Realisierung von Pfadausdrücken)

```
SELECT P.Name
FROM Person P
WHERE P.Manager -> Name = “Schmidt” AND
                      P.Anschrift.Ort = “Leipzig”
```

Funktionen und Methoden

- Routinen (Funktionen und Prozeduren) als eigenständige Schemaobjekte bereits in SQL/PSM
- Methoden: beziehen sich auf genau einen UDT
- Realisierung aller Routinen und Methoden über prozedurale SQL-Spracherweiterungen oder in externer Programmiersprache (C, Java, ...)
- Namen von SQL-Routinen und Methoden können überladen werden (keine Eindeutigkeit erforderlich)
  - bei SQL-Routinen wird zur Übersetzungszeit anhand der Anzahl und Typen der Parameter bereits die am “besten passende” Routine ausgewählt
  - bei Methoden wird dynamisches Binden zur Laufzeit unterstützt

	SQL-Routinen	**Externe Routinen**
Aufruf in SQL (SQL-invoked routines)	SQL-Funktionen (inkl. Methoden) und SQL-Prozeduren	externe Funktionen und Prozeduren
Externer Aufruf	nur SQL-Prozeduren (keine Funktionen)	(nicht relevant für SQL)

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

```
Observer-Methoden:  METHOD  Strasse ()              RETURNS VARCHAR (40);
                    METHOD  PLZ ()      RETURNS CHAR (5);
                    METHOD  Ort ()      RETURNS VARCHAR (40);
Mutator-Methoden:   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 ()
  - SET a.x = y ist äquivalent zu a.x (y)

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))
```

Tabellenwertige Funktionen

- Seit SQL:2003 können benutzerdefinierte Funktionen eine Ergebnistabelle zurückliefern

  RETURNS TABLE (<column list> **)**

- Beispiel

```
CREATE FUNCTION ArmeMitarbeiter ()
RETURNS TABLE (PNR INT, Gehalt DECIMAL (8,2));

 RETURN SELECT PNR, Gehalt FROM PERS
        WHERE Gehalt < 20000.0);
```

- Nutzung in FROM-Klausel

```
SELECT *
FROM TABLE (ArmeMitarbeiter ())
```

Generalisierung / Spezialisierung

- Spezialisierung in Form von Subtypen und Subtabellen
- nur Einfachvererbung
- Supertyp muss 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 (PNR INT, Name CHAR (20), Grundgehalt REAL, ...) NOT FINAL
CREATE TYPE Techn-AngT **UNDER** PersonT (Techn-Zulage REAL, ...) NOT FINAL
CREATE TYPE Verw-AngT **UNDER** PersonT (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:**

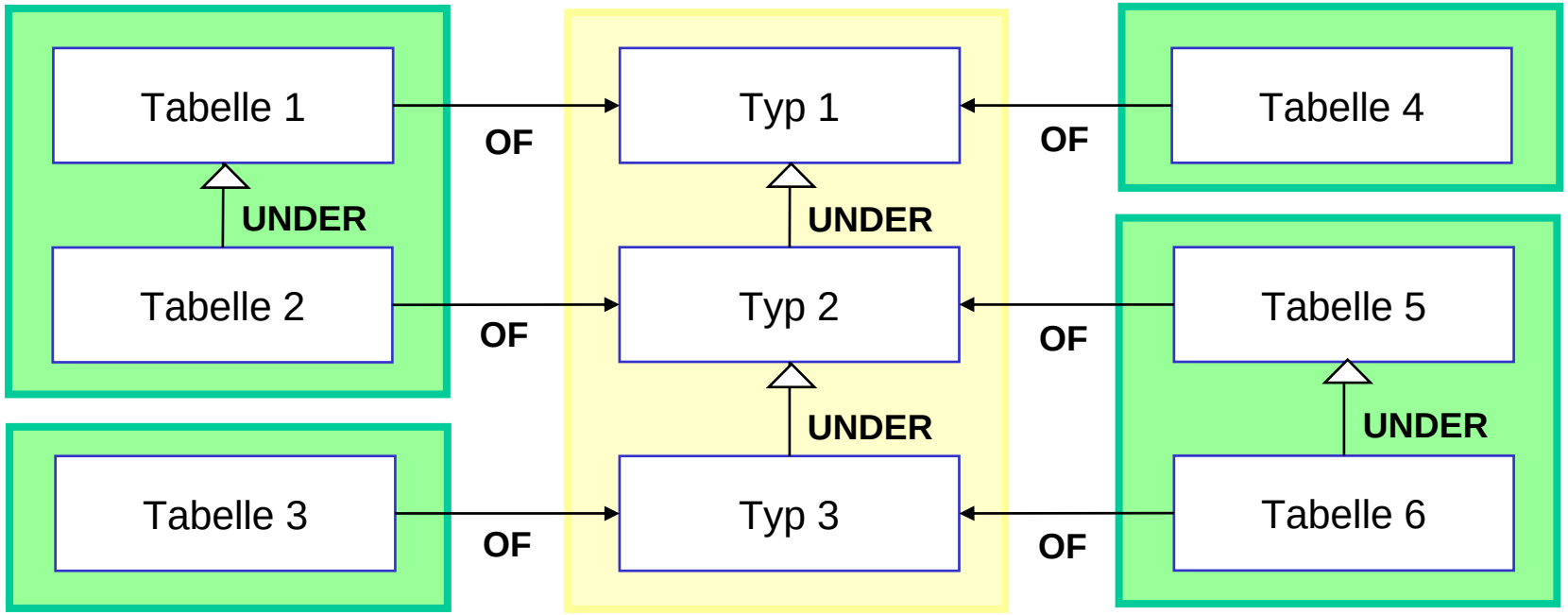
PNR	Name	Techn-Zulage	Verw-Zulage
8217	Hans ...		
5581	Rita ...	2300	
3375	Anna ...		3400
...			

Homogene Ergebnismengen durch Zugriff auf Subtabellen bzw. mit ONLY-Prädikat

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

Subtypen vs. Subtabellen

- Typ- und Tabellenhierarchen müssen nicht 1:1 korrespondieren
  - Typ einer Subtabelle 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
  - Typ einer Objekttabelle ohne Subtabellen kann Subtypen haben



Dynamisches Binden

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

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

With-Anweisung in SQL:1999

- Vergabe von Namen für Anfrageausdruck (benannte Anfrage)
  - v. a. falls Anfrage mehrfach referenziert wird und damit wiederverwendet werden kann (gemeinsamer Tabellenausdruck)
  - entspricht temporärem View (für eine Anfrage)
- Spezifikation: WITH <Anfragename> [(<Attributliste>)]
  AS (<Anfrageausdruck>)
- Beispiel

```
SELECT Name FROM Pers
WHERE    Grundgehalt > 0.5 * (SELECT MAX (Verw-Zulage) FROM Verw-Ang) AND
         Grundgehalt < 2 * (SELECT MAX (Verw-Zulage) FROM Verw-Ang)
```

Umformulierung:

```
WITH MaxVZulage (Vmax) AS (SELECT MAX (Verw-Zulage) FROM Verw-Ang)
SELECT Name  FROM Pers
WHERE
```

Rekursion

- Berechnung rekursiver Anfragen (z. B. transitive Hülle) über rekursiv definierte Sichten (Tabellen)
- Grundgerüst

```
WITH RECURSIVE RekursiveTabelle (...) AS
( SELECT ... FROM Tabelle WHERE ...
  UNION
  SELECT ... From Tabelle, RekursiveTabelle     WHERE ...)
SELECT ... From RekursiveTabelle WHERE ...
```

- Beispiel

```
CREATE TABLE Eltern (Kind   CHAR (20), Elternteil  CHAR (20));
```

Alle Vorfahren von „John“ ?

```
WITH RECURSIVE Vorfahren (Generation, K, V) AS
( SELECT (1, Kind, Elternteil
   FROM Eltern
   WHERE Kind=“John“
UNION
SELECT V.Generation+1, E.Kind, V.V
   FROM Eltern E, Vorfahren V
   WHERE E.Elternteil = V.K)
SELECT Generation, V  FROM Vorfahren
```

Rekursion (2)

- Syntax (WITH-Klausel)

```
<query expression> ::=    [WITH [RECURSIVE] <with list> ] <query expression body>
<with list element> ::=   <query name> [ ( <with column list> ) ] AS ( <query expression> )
                                [SEARCH <search order> SET <sequence column>]
                                [CYCLE <cycle column list>]
<search order> ::=        DEPTH FIRST BY <sort specification list> |
                          BREADTH FIRST BY <sort specification list>
```

- Merkmale
  - verschiedene Suchstrategien (Depth First, Breadth First)
  - lineare oder allgemeine Rekursion
- Zyklenbehandlung durch Einschränkung der Rekursionstiefe
  - z. B. durch zusätzliche Attribute, die Schachtelungstiefe codieren (z. B. Attribut in Vorfahren-Tabelle, welches Generationen mitzählt)

Zusammenfassung

- SQL:2003 Standardisierung
  - Kompatibilität mit existierenden SQL-Systemen + Objektorientierung
  - Objekt-Identität (REF-Typen)
  - erweiterbares Typsystem: signifikante Verbesserung der Modellierungsfähigkeiten
  - Benutzerdefinierte Datentypen und Methoden (UDT, UDF)
  - DISTINCT Types
  - ROW: Tupel-Konstruktor
  - Kollektionstypen ARRAY und MULTISET
  - Typhierarchien und Vererbung: Subtypen vs. Subtabellen
  - Zahlreiche weitere Fähigkeiten: Rekursion, Tabellenfunktionen • • •
- Hohe Komplexität