4. Objekt-relationale DBS

- SQL-Standardisierung
- SQL-Typsystem
  - Typkonstruktor ROW
  - Typkonstruktor ARRAY, UNNEST-Operation
  - Typkonstruktur MULTISET
  - 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, SQL:2023 : XML-Unterstützung (2003, 2006); temporale DBS (2011), JSON (2016, 2023), PQL Propertygraph-Anfragen (2023) diverse Erweiterungen, z.B. MERGE, OLAP-Abfragen, …

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“
```

Array-Beispiele PostgresSQL (LOTS)

SQL Anfrage

```
SELECT cardinality (ARRAY['Deutsch','Französisch','Englisch'])
```

cardinality
3

SQL Anfrage

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

b
1
2
3

SQL Anfrage

```
SELECT titel, autoren
FROM (select buchid, titel,
      array (select textcat(vornamen,nachname) from  buch_aut natural join autor where buchid=B.buchid) from buch B)
      as Buecher (bid, titel, autoren)
WHERE cardinality (autoren)=3
```

zeige Datensätze 1 - 20 (234 insgesamt)

Zeige: 20 Datensätze, beginnend ab 21 > >>

titel	autoren
A comprehensive guide to AI and expert systems	{"Robert J.Levine","Diane E.Drang","Barry Edelson"}
Expertensysteme: Werkzeuge u. Anwendungen	{"Paul Harmon","Rex Maus","William Morrisey"}
Softwaresystem zur Formelmanipulation. Praktisches Arbeiten mit den Computer-Algebra-Systemen REDUCE, MACSYMA und DERIVE	{"Bernhard Kutzler","Franz Lichtenberger","Franz Winkler"}
Datendrucker	{"Walter E.Proebster","Helmut P.Louis","Erich Eissfeldt"}
Basiskenntnis Bibliothek : Fachkunde für Assistentinnen und Assistenten an Bibliotheken ; die theoretischen und praktischen Grundlagen eines Bibliotheksberufes	{"Günter Rötcher","Klaus-Peter Böttger","Ursula Ankerstein"}
Clipper 5.0	{"Frank Anders","Thomas Behrensdorf","Malte Borges"}
EDV-Grundwissen : eine Einführung in Theorie und Praxis der modernen EDV	{"Manfred Precht","Nikolaus Meier","Joachim Kleinlein"}
Java : Programmierhandbuch und Referenz	{"Stefan Strobel","Stefan Middendorf","Rainer Singer"}
Ganzheitliches Informationsmanagement	{"Jörg Biethan","Harry Mucksch","Walter Ruf"}
Der LATEX-Begleiter	{"Michel Goossens","Frank Mittelbach","Alexander Samarin"}
Algebraische Spezifikation abstrakter Datentypen	{"Hans-Dieter Ehrich","Martin Gogolla","Udo W.Lipeck"}
Objektorientierte Sprachkonzepte und diskrete Simulation	{"Thomas Frauenstein","Uwe Pape","Olaf Wagner"}
Windows im Netzwerk : optimale Installation, effektive Speicherverwaltung und praktische Problemlösungen für Windows 3.1, Windows für Workgroups und Windows NT	{"Michael Scholz","Lorenz Moosmüller","Birgit Smadja"}
Datenbanken zur rechnergestützten Auftragsabwicklung in kleinen und mittleren Unternehmen	{"Wolfgang Michesl","Gerhard Steinmetz","Wolfgang Kaiser"}
Draw Perfect : Grafik, Texte, Layout	{" K. J.Klein","Peter Brunswicker"," H. J.Kehnen"}
PlanPerfect-Praxis : Grafik, Makros, Kalkulation, Formeln, Analysen, Controlling	{"Peter Brunswicker","Karl J.Klein","Heinrich Kehnen"}
Datensicherung im System der EDV	{"Dieter Horn","Norbert Busch","Jürgen Kirbach"}

ANY-Test (PostgresSQL-spezifisch)

SQL Anfrage

```
with Pers (Name,Sprachen) as (values('Meister',ARRAY['Deutsch','Englisch']),
('Baum',ARRAY['Deutsch','Französisch']),
('Schmied',ARRAY['Deutsch','Französisch','Englisch']))
select Name from Pers where 'Französisch' = ANY (Sprachen)
```

zeige Datensätze 1 - 2 (2 insgesamt)

Zeige: 2 Datensätze, beginnend ab

name
Baum
Schmied

- UNNEST in SELECT- statt FROM-Klausel:

```
select Name from Pers where 'Französisch' in (select unnest (Sprachen))
```


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

 <instantiable 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,
             Anschrift          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 M
   WHERE M.Einkommen() > 50000.0
```

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)

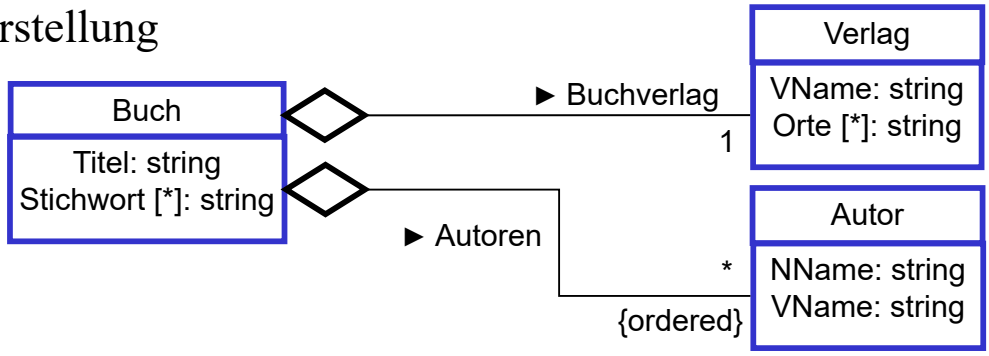
```
SELECT  DEREF (P.Manager).Name
FROM    Person P
WHERE   P.Name= “Meister”
```

- 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

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 () , 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

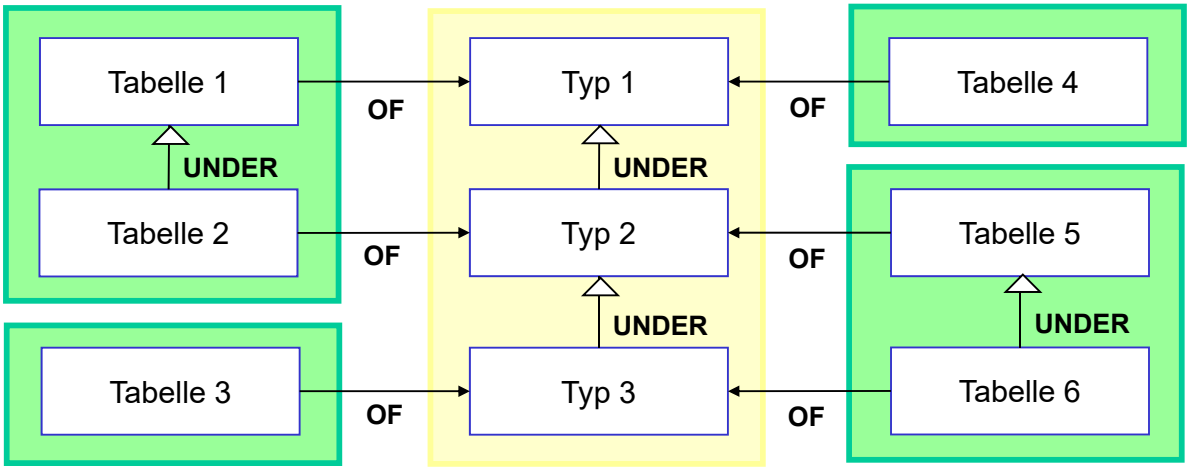
```
SELECT *
FROM ONLY Pers
WHERE Grundgehalt > 40000
```

```
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 Objekttabelle 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 MULTISET
- Typhierarchien und Einfach-Vererbung: Subtypen vs. Subtabellen

https://pingo.coactum.de/533527

Welche Typkonstruktoren können gemäß den objektrelationalen Erweiterungen des SQL-Standards für Attributdefinitionen genutzt werden?

Zeit zum Abstimmen: 0:18

Wählen Sie alle richtigen Antwortmöglichkeiten aus:

- ☐ LIST
- ☐ MULTISET
- ☐ DEREF
- ☐ TUPLE
- ☐ ARRAY

Abstimmen!

https://pingo.coactum.de/533527

Objekttabellen (typisierte Tabellen)

Zeit zum Abstimmen: 0:25

Wählen Sie alle richtigen Antwortmöglichkeiten aus:

- ☐ Sätze einer typisierten Tabelle erhalten eine Objekt-ID zur Referenzierung
- ☐ Nur typisierte Tabellen können REF-Attribute besitzen
- ☐ Objekttabellen können nicht auf Basis von DISTINCT TYPES erzeugt werden
- ☐ In einer Tabelle können nur Objekte einer einzigen Objekttabelle referenziert werden
- ☐ Sätze einer geschachtelten Tabelle können nicht referenziert werden

Abstimmen!

https://pingo.coactum.de/533527

Vererbung in SQL

Zeit zum Abstimmen: 0:24

Wählen Sie alle richtigen Antwortmöglichkeiten aus:

- ☐ **Subtabellen müssen typisiert sein**
- ☐ **Subtabellen der selben Supertabelle müssen von unterschiedlichen strukturierten Typen abgeleitet sein**
- ☐ **Der Typ einer Subtabelle kann kein Wurzeltyp sein**
- ☐ **Die Methoden einer Subtabelle sind auch für Instanzen der Supertabelle anwendbar (Substituierbarkeitsprinzip)**

Abstimmen!

Datenbanksysteme 2 533527

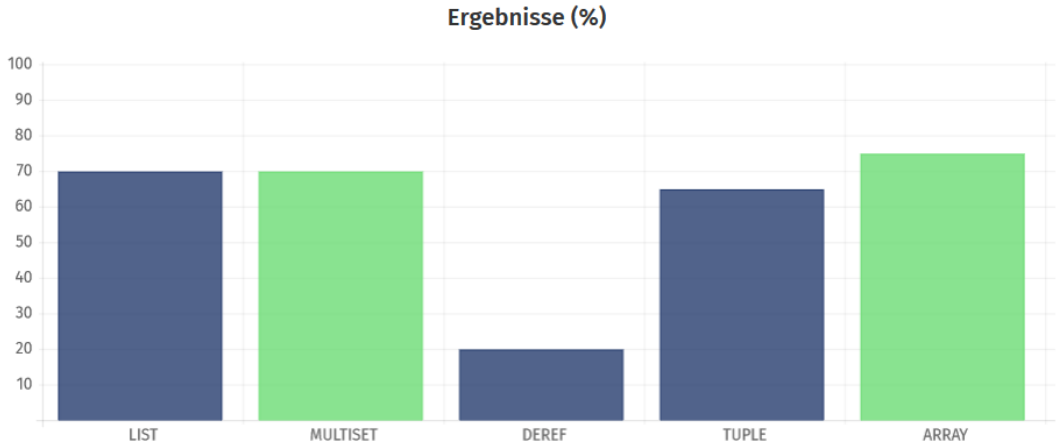
Welche Typkonstruktoren können gemäß den objektrelationalen Erweiterungen des SQL-Standards für Attributdefinitionen genutzt werden?

Dies ist eine Multiple-Choice-Umfrage.

Teilnehmer: **20**

Antwortmöglichkeiten:

- 14 70% LIST
- 14 70% MULTISET
- 4 20% DEREF
- 13 65% TUPLE
- 15 75% ARRAY



Datenbanksysteme 2 533527

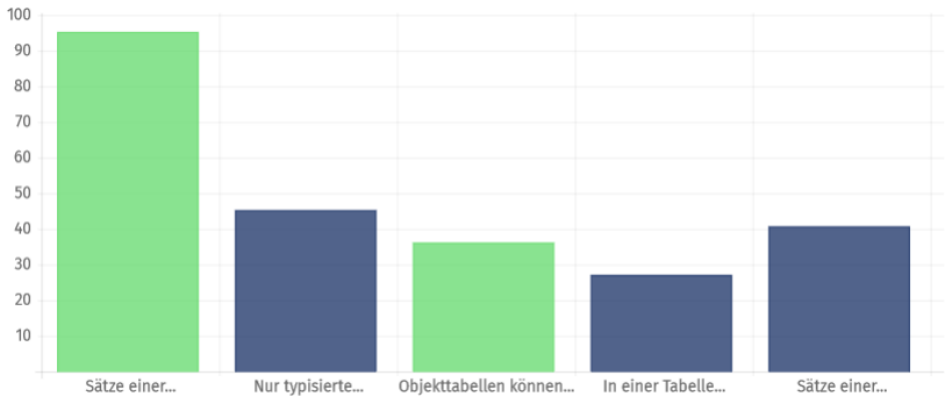
Objekttabellen (typisierte Tabellen)

Dies ist eine Multiple-Choice-Umfrage.

Teilnehmer: **22**

Antwortmöglichkeiten:

- 21 95% Sätze einer typisierten Tabelle erhalten eine Objekt-ID zur Referenzierung
- 10 45% Nur typisierte Tabellen können REF-Attribute besitzen
- 8 36% Objekttabellen können nicht auf Basis von DISTINCT TYPES erzeugt werden
- 6 27% In einer Tabelle können nur Objekte einer einzigen Objekttabelle referenziert werden
- 9 41% Sätze einer geschachtelten Tabelle können nicht referenziert werden



Datenbanksysteme 2 533527

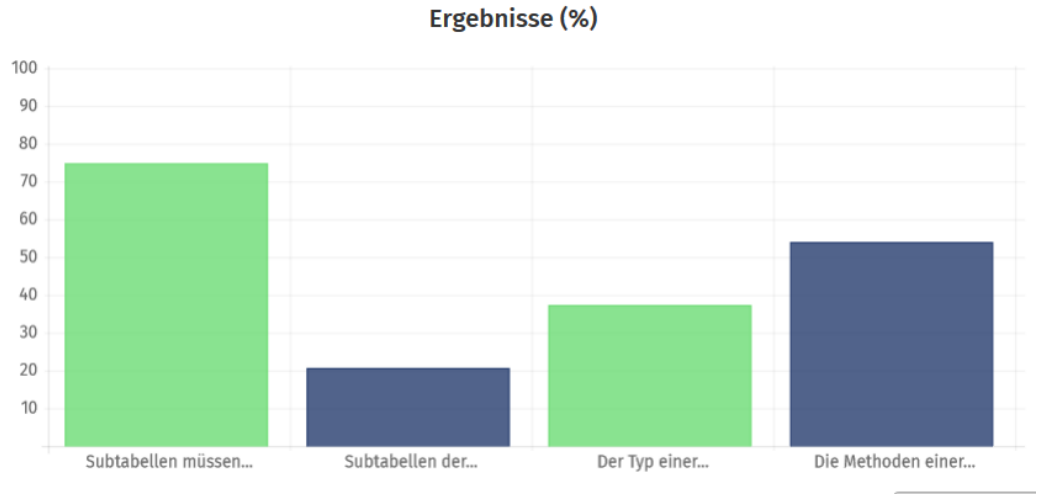
Vererbung in SQL

Dies ist eine Multiple-Choice-Umfrage.

Teilnehmer: **24**

Antwortmöglichkeiten:

- 18 75% Subtabellen müssen typisiert sein
- 5 21% Subtabellen der selben Supertabelle müssen von unterschiedlichen strukturierten Typen abgeleitet sein
- 9 38% Der Typ einer Subtabelle kann kein Wurzeltyp sein
- 13 54% Die Methoden einer Subtabelle sind auch für Instanzen der Supertabelle anwendbar (Substituierbarkeitsprinzip)



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)

Anwendungsbeispiel: Buch-Datenbank

Autor (autorid, nachname, vorname)
Buch_flach (buchid, titel, jahr, *verlagsid*)
Buch_Aut (*buchid, autorid*, rang)
Verlag (verlagsid, name, ort)
Schlagwort (swid, schlagwort)
Buch_SW (*buchid, swid*)

- Nutzung von Kollektionstypen

```
CREATE TABLE Buch (buchid int primary key,
                   titel  VARCHAR(50), jahr  int,
                  verlagsid int references Verlag,
                   autoren ROW (nachname VARCHAR(30),
                                vorname VARCHAR(30)) ARRAY[20],
                   schlagworte VARCHAR(30) MULTISET)
```

- Simulation der **NEST**-Operation

```
INSERT INTO Buch(buchid,titel,jahr,verlagsid,autoren,schlagworte)
  SELECT buchid, titel, jahr, verlagsid,
   ARRAY(SELECT nachname,vorname FROM Autor NATURAL JOIN Buch_Aut BA
                    WHERE BA.buchid = B.buchid  ORDER BY rang),
  MULTISET(SELECT schlagwort FROM Schlagwort NATURAL JOIN Buch_SW BS
                    WHERE BS.buchid = B.buchid)
  FROM Buch_flach B
```

Polyeder-Modellierung mit SQL:2003

```
CREATE TYPE PunktT AS (X FLOAT, Y FLOAT, Z FLOAT)
                  INSTANTIABLE NOT FINAL;
CREATE TYPE KanteT AS (Punkt1 REF (PunktT), Punkt2 REF (PunktT))
                  INSTANTIABLE NOT FINAL,
                  INSTANCE METHOD Laenge() RETURNS REAL;

CREATE TABLE Punkt OF PunktT (REF IS pid SYSTEM GENERATED);
CREATE TABLE Kante OF KanteT (REF IS kid SYSTEM GENERATED,
                     Punkt1 REF(PunktT) SCOPE Punkt,
                     Punkt2 REF(PunktT) SCOPE Punkt)
CREATE TABLE POLYEDER (VID   INT,
                       Flaechen ROW (FlaechenID INT,
                           Kanten REF (KanteT) SCOPE Kante MULTISET)
                                 MULTISET,
                      CHECK CARDINALITY(Flaechen)>3)
```


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 ())
```