4. Objekt-relationale DBS

- SQL-Standardisierung
- Typkonstruktoren
  - ROW, ARRAY, MULTISET
  - UNNEST-Operation
- Benutzerdefinierte Typen und Funktionen (UDTs, UDFs)
  - DISTINCT-Typen
  - strukturierte Datentypen, typisierte Tabellen , REF-Typ
  - UDT-Kapselung
- Typhierarchien / Tabellenhierarchien (Subtypen, Subtabellen)
- *rekursive Anfragen / Berechnung der transitiven Hülle*
  - WITH-Anweisung
  - RECURSIVE UNION
- *Temporale Datenbanken mit SQL:2011*

Objekt-relationale 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 (ANSI)

1986 SQL86

- keine Integritätszusicherungen

1989 SQL89 (120 Seiten)

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

1992 SQL92 (SQL2)

- 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, **Rekursion**

SQL:2003, SQL:2008, SQL:2011, SQL:2016 : **XML**-Unterstützung (2003, 2006);
temporale DBS (2011), **JSON**-Unterstützung (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
  - 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:

```
ROW („Peter“, „Meister“)
```

  - Zugriff auf Tupelfeld mit Punktnotation:

```
SELECT * FROM Pers
WHERE
```

  - Vergleiche

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

ARRAY-Kollektionstyp

- wurde als erster Kollektionstyp im Standard eingeführt (SQL:1999)
- Spezifikation: <Elementtyp> ARRAY [<maximale Kardinalität>]
  - Elementtypen: alle Datentypen (z.B. Basisdatentypen, benutzerdefinierte Typen)
  - geschachtelte (mehrdimensionale) Arrays erst ab SQL:2003

<table>
<tr><th rowspan="2"><u>PNR</u></th><th colspan="2">Name</th><th rowspan="2">Sprachen</th></tr>
<tr><th>VName</th><th>NName</th></tr>
<tr><td rowspan="2">1</td><td rowspan="2">Markus</td><td rowspan="2">Nentwig</td><td>Deutsch</td></tr>
<tr><td>Englisch</td></tr>
</table>

```
CREATE TABLE Mitarbeiter
        (PNR      int Primary Key,
         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 <Tabellenalias> (<Attributliste>)(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“
```

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

PNR	Name		Sprachen
	VName	NName	
1234	Peter	Meister	Deutsch
			Englisch
			Französich

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

- Spezifikation: <Elementtyp> MULTISET
  - Elementtypen: alle Datentypen inklusive ROW, ARRAY und MULTISET
  - beliebige Schachtelung möglich

ANAME	AOrte		
	Ort	Mitarbeiter	
		Name	Beruf
Entwicklung	Leipzig	Müller	Programmierer
		Mann	Tester
	Halle	Baum	Architekt
Vertrieb	Leipzig	Schmidt	Kaufmann

```
CREATE TABLE Abt ( AName     VARCHAR(30),
                   AOrte  ROW(Ort VARCHAR(30),
                             Mitarbeiter  ROW (Name VARCHAR(30),
                                  Beruf VARCHAR(30)) MULTISET
                             ) MULTISET
                 )
```

MULTISET-Operatoren

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

MULTISET - Beispiel

<table>
<tr><th rowspan="3">ANAME</th><th colspan="3">AOrte</th></tr>
<tr><th rowspan="2">Ort</th><th colspan="2">Mitarbeiter</th></tr>
<tr><th>Name</th><th>Beruf</th></tr>
<tr><td rowspan="3">Entwicklung</td><td rowspan="2">Leipzig</td><td>Müller</td><td>Programmierer</td></tr>
<tr><td>Mann</td><td>Tester</td></tr>
<tr><td>Halle</td><td>Baum</td><td>Architekt</td></tr>
<tr><td>Vertrieb</td><td>Leipzig</td><td>Schmidt</td><td>Kaufmann</td></tr>
</table>

- Beispiel-Query: *Welche Abteilungen sind in Leipzig?*

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

Anwendungsbeispiel: Buch-Datenbank

Autor (<u>autorid</u>, nachname, vorname)

Buch_flach (<u>buchid</u>, titel, jahr, *verlagsid*)

Buch_Aut (*<u>buchid, autorid</u>*, rang)

Verlag (<u>verlagsid</u>, name, ort)

Schlagwort (<u>swid</u>, schlagwort)

Buch_SW (*<u>buchid, swid</u>*)

- Zielschema

buchid	titel	Verlags id	autoren		schlagworte
			nachname	vorname	
1	SQL	1	Müller	Hans	SQL
			Mann	Peter	Grundlagen
			Baum	Günther	Relational
2	ORDBS	2	Schmidt	Klaus	UDT

verlags id	Name	Ort
1	Springer	Hamburg
2	Hanser	München

```
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)
```

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*)

```
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
```

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 * 1.10`

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),
         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”
```

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 WITH OPTIONS SCOPE Punkt,
                   Punkt2 WITH OPTIONS SCOPE Punkt)
CREATE TABLE POLYEDER (VID   INT,
                  Flaechen ROW(FlaechenID INT,
                       Kanten REF(KanteT) SCOPE Kante MULTISET)
                           MULTISET,
                  CHECK CARDINALITY(Flaechen)>3)
```


DBS 2

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  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

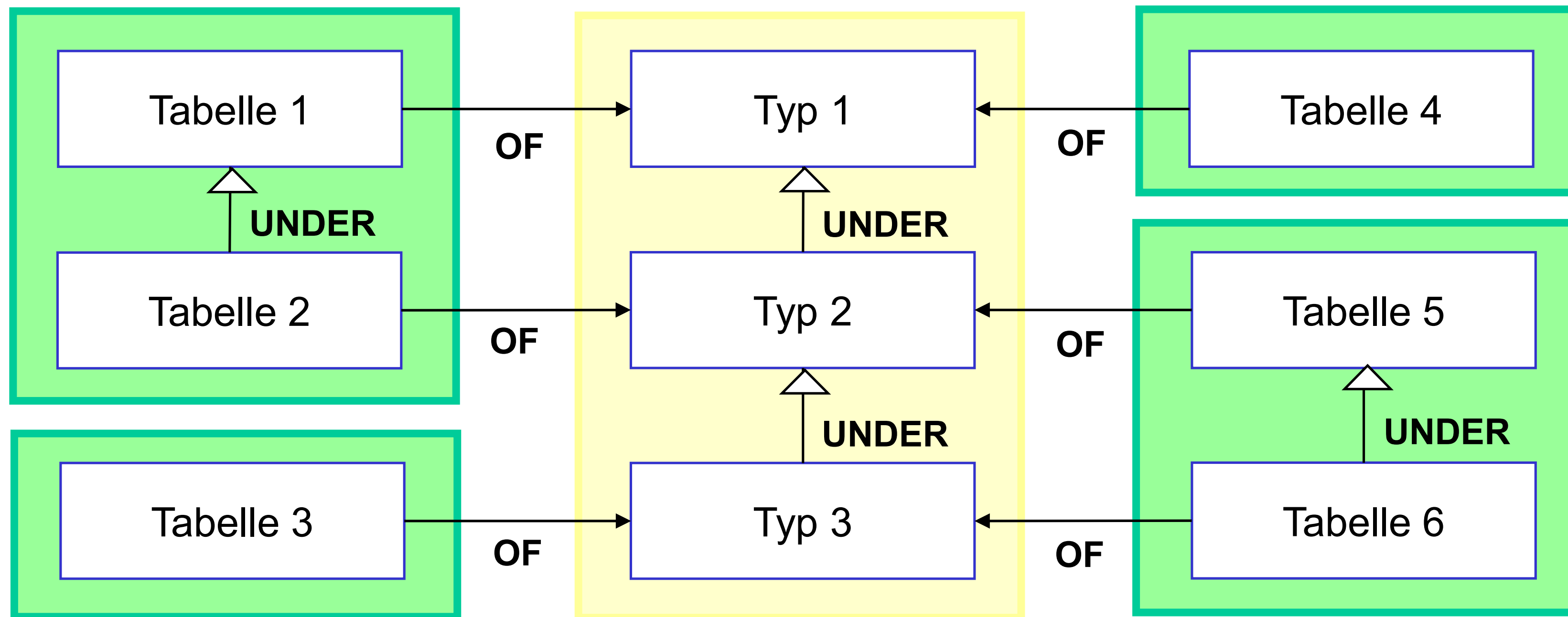
```
SELECT *
FROM ONLY Pers
WHERE Grundgehalt > 40000
```

```
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 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 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';
```

With-Anweisung

- Vergabe von Namen für Anfrageausdruck (benannte Anfrage)
  - entspricht temporärem View (für eine Anfrage)
  - u.a. nützlich bei mehrfacher Referenzierung von Teilanfragen (gemeinsamer Tabellenausdruck) oder Vorberechnung komplexer Teilergebnisse
- Spezifikation: WITH <R1> [(<Attributliste>)] AS (<Anfrageausdruck>) [,
  <R2> [(<Attributliste>)] AS (<Anfrageausdruck>), …]
  SELECT … *(Bezug auf R1,R2 und andere Tabellen)*
- **Beispiel** für Tabelle Verkauf (Kunde, Produkt, Ort, Anzahl, Betrag)

```
WITH Umsatz_O AS (SELECT Ort, SUM(Betrag) AS Umsatz FROM Verkauf
                      GROUP BY Ort),
     top_Orte  AS (SELECT Ort FROM Umsatz_O
                 WHERE Umsatz > (SELECT AVG(Umsatz)*3 FROM Umsatz_O)
SELECT Ort, Produkt, SUM(Anzahl) AS Anzahl, SUM(Betrag) AS Umsatz
FROM Verkauf
WHERE Ort IN (SELECT Ort FROM top_Orte)
GROUP BY Ort, Produkt;
```

Rekursion

- Berechnung rekursiver Anfragen (z. B. transitive Hülle) über rekursiv definierte Sichten (Tabellen)
- Anwendungsfälle
  - Stücklistenauflösung
  - Routenberechnung aus Einzelverbindungen
  - Auswertung über Vorgesetztenverhältnisse, Vorfahrenbeziehungen etc.
- Grundgerüst seit SQL:1999

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

SELECT ... From RekursiveTabelle WHERE ...
```

Rekursion: Beispiel

Regeln: *Vorfahr (K,V) <- Eltern (K,V)*
Vorfahr (K,V) <- Vorfahr (K,X), Eltern (X,V)

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

Eltern

Kind	*Elternteil*
John	Sabrina
Mary	Sabrina
Sabrina	Helmut
Helmut	Jennifer
...	...

Alle Vorfahren von „John“ ?

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

SELECT Generation, Vorfahr  FROM Vorfahren
   WHERE Kind="John"
```

Komplette transitive Hülle, danach Einschränkung auf „John“

Rekursion: Beispiel (Forts.)

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

Kind	*Elternteil*
John	Sabrina
Mary	Sabrina
Sabrina	Helmut
Helmut	Jennifer
...	...

Alle Vorfahren von „John“ ?

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

SELECT Generation, Vorfahr  FROM Vorfahren
```

Transitive Hülle nur für „John“

Temporale Datenbanken

- systemseitige Unterstützung zeitbehafteter / versionierter Datenbanken
- unterschiedliche Zeiten bzw. Zeitintervalle
  - Gültigkeitszeit (valid time)
    - kann in Vergangenheit, Gegenwart oder Zukunft liegen
    - durch Anwendung / Nutzer festzulegen
  - Transaktionszeit: Zeitpunkt der Änderung
    - kann nicht in Zukunft liegen
    - automatische Festlegung durch DBS
  - *Bitemporale Datenbanken* unterstützen beide Zeitarten (bei Speicherung, Änderung und Abfragen)
- Zeitunterstützung seit SQL:2011 standardisiert*
  - anwendungsversionierte Tabellen (application-time period table) für Gültigkeitszeit
  - systemversionierte Tabellen für Transaktionszeit
  - anwendungs- und systemversionierte (bitemporale) Tabellen

* K. Kulkarni, J. Michels: *Temporal features in SQL:2011.* Sigmod Record, Sep. 2012

Anwendungsversionierte Tabellen

- Verwendung von Zeitintervallen (PERIOD) mit Start- und Endzeitpunkt vom Typ DATE oder TIMESTAMP
  - halboffene Intervalle (Startzeitpunkt ist Teil des Intervalls, Endzeitpunkt nicht)
  - Primärschlüssel erfordert Hinzunahme des Zeitintervalls
  - Fremdschlüssel müssen für Bezugsintervall korrekt sein

```
CREATE TABLE PERS
(Pname VARCHAR(50) NOT NULL PRIMARY KEY,
 ANR VARCHAR(10),
 Starttermin  DATE NOT NULL,
 Endtermin DATE NOT NULL,
 PERIOD FOR Pzeitraum (Starttermin, Endtermin),
 PRIMARY KEY(Pname, Pzeitraum WITHOUT OVERLAPS),
 FOREIGN KEY(ANR,PERIOD Pzeitraum)REFERENCES ABT(ANR, PERIOD Azeitraum));
```

Pname	ANR	Starttermin	Endtermin
John	J13	15/11/2015	16/11/2016
John	J14	16/11/2016	01/05/2017
Tracy	K25	01/01/2016	31/12/9999

Anwendungsversionierte Tabellen (2)

- Intervallanpassungen bei Änderungen (UPDATE, DELETE) möglich

```
INSERT INTO PERS (Pname, ANR, Starttermin, Endtermin)
        VALUES ('John', 'J12', DATE '15-11-2013', DATE '15-11-2015'')

UPDATE PERS FOR PORTION OF PZeitraum
    FROM DATE '01-03-2016' TO DATE '01-07-2016'
SET  ANR = 'M12'' WHERE Pname = 'John'

DELETE FROM PERS FOR PORTION OF PZeitraum
   FROM DATE '01-08-2015' TO DATE '01-09-2016'
WHERE Pname = 'Tracy'
```

Pname	ANR	Starttermin	Endtermin
John	J13	15/11/2015	16/11/2016
John	J14	16/11/2016	01/05/2017
Tracy	K25	01/01/2015	31/12/9999

Anwendungsversionierte Tabellen (3)

- Ansatz ermöglicht zeitbezogene Auswertungen über Start/Endzeitpunkte

```
SELECT ANR FROM PERS
WHERE Pname = 'John' AND Starttermin >= DATE '01-12-2016'
      AND Endttermin <= DATE '01-12-2016';

SELECT count (distinct ANR) FROM PERS
WHERE Pname = 'John' AND Starttermin >= DATE '01-01-2013'
      AND Endtermin <= CURRENT_DATE ;
```

- oft einfacher mit Suchprädikaten für PERIOD-Intervalle: CONTAINS, OVERLAPS, EQUALS, PRECEDES, SUCCEEDS, IMMEDIATELY PRECEDES, IMMEDIATELY SUCCEEDS

```
SELECT ANR FROM PERS
WHERE Pname = 'John' AND Pzeitraum CONTAINS DATE '01-12-2016';

SELECT count (distinct ANR) FROM PERS
WHERE Pname = 'John' AND
      Pzeitraum OVERLAPS PERIOD (DATE '01-01-2013', CURRENT_DATE)
```

Systemversionierte Tabellen

- automatische Erzeugung und Anpassung von Zeitintervallen bzgl. Änderungszeit
  - obligatorische Attribute für Start- und Endzeitpunkte
  - nur aktuelle (nicht historische) Versionen von Tupeln können geändert/gelöscht werden
  - Integritätsbedingungen werden nur auf aktuellen Tupelversionen überwacht

```
CREATE TABLE PERS
    (Pname VARCHAR(50) NOT NULL PRIMARY KEY,
     ANR VARCHAR(10),
     sys_start TIMESTAMP(6) GENERATED ALWAYS AS ROW START,
     sys_end TIMESTAMP(6) GENERATED ALWAYS AS ROW END,
     PERIOD FOR SYSTEM_TIME (sys_start, sys_end),
     FOREIGN KEY (ANR) REFERENCES  ABT (ANR);
    )
    WITH SYSTEM VERSIONING;
```

Pname	ANR	sys_start	sys_end
John	J13	15/11/2015	31/12/9999
Tracy	K25	15/11/2015	31/12/9999

Systemversionierte Tabellen (2)

- UPDATE führt zu historischem Tupel und aktuellem Tupel

  Änderung am 15.5.2017:

```
UPDATE PERS  SET  ANR = 'M12' WHERE Pname = 'John'
```

- DELETE setzt Endzeitpunkt auf Transaktionszeitpunkt

  Löschen am 15.5.2017:

```
DELETE FROM PERS  WHERE Pname = 'Tracy'
```

Pname	ANR	sys_start	sys_end
John	J13	15/11/2015	31/12/9999
Tracy	K25	15/11/2015	31/12/9999

Queries auf systemversionierten Tabellen

- neue Suchprädikate
  - FOR SYSTEM_TIME AS OF <datetime value expression>
  - FOR SYSTEM_TIME BETWEEN < datetime 1> AND < datetime 2>
  - FOR SYSTEM_TIME FROM < datetime 1> TO < datetime 2>

```
SELECT ANR
FROM PERS FOR SYSTEM_TIME AS OF DATE '01-01-2016'
WHERE Pname = 'John'
```

```
SELECT count (distinct ANR)
FROM PERS FOR SYSTEM_TIME BETWEEN DATE '01-01-2015' AND CURRENT_DATE
WHERE Pname = 'John'
```

Bitemporale Tabellen

```
CREATE TABLE PERS
(Pname VARCHAR(50) NOT NULL,
 ANR VARCHAR(10),
 Starttermin  DATE NOT NULL,
 Endtermin DATE NOT NULL,
 sys_start DATE GENERATED ALWAYS AS ROW START,
 sys_end DATE GENERATED ALWAYS AS ROW END,
 PERIOD FOR Pzeitraum (Starttermin, Endtermin),
 PERIOD FOR SYSTEM_TIME (system_start, system_end),
 PRIMARY KEY (Pname, Pzeitraum WITHOUT OVERLAPS),
 FOREIGN KEY (ANR, PERIOD Pzeitraum)
              REFERENCES  ABT (ANR, PERIOD Azeitraum)
 ) WITH SYSTEM VERSIONING;
```

Pname	ANR	Starttermin	Endtermin	sys_start	sys_end
John	J13	15/11/2015	16/11/2016	15/11/2015	15/11/2016
John	J14	16/11/2016	31/12/9999	15/11/2016	31/12/9999
Tracy	K25	01/01/2016	31/12/9999	15/01/2016	31/12/9999

Bitemporale Tabellen (2)

Änderung am 15.05.2019 (Korrektur falsche Abteilungszugehörigkeit):

```
UPDATE PERS  SET  ANR = 'M12' WHERE Pname = 'John'
```

Am 01.06.2019 wird John für den Zeitraum 1.10.2019 bis 31.03.2020 an Abteilung M13 ausgeliehen

```
UPDATE PERS FOR PORTION OF Pzeitraum FROM DATE '01-10-2019'
                                     TO DATE '01-04-2020'
SET ANR = 'M13' WHERE Pname = 'John'
```

In welcher Abteilung war John am 01.01.2019 gemäß Datenbankstand vom 01.05.2019?

```
SELECT ANR  FROM PERS FOR SYSTEM_TIME AS OF DATE '01-05-2019'
WHERE Pname = 'John' AND  Pzeitraum CONTAINS DATE '01-01-2019'
```

Pname	ANR	Starttermin	Endtermin	sys_start	sys_end
John	J14	16/11/2016	31/12/9999	15/11/2016	31/12/9999
Tracy	K25	01/01/2016	31/12/9999	15/11/2016	31/12/9999

Zusammenfassung

- SQL-Standardisierung von objekt-relationen DBS
  - 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 im SQL-Standard: Rekursion, Tabellenfunktionen • • •
- Temporale Datenbanken seit SQL:2011
  - Unterstützung von Gültigkeits- und Transaktionszeit durch Zeitintervalle (PERIOD)
- hohe Komplexität