10. SQL:1999 als OR-Datenmodell

- **Einordnung/Überblick**
- **Objekt-Relationale Erweiterungen (Auszug)**
  - Erweiterbares Typsystem
  - Benutzerdefinierte Typen und Routinen
  - Strukturierte Typen
  - Typ- und Tabellenhierarchien
  - Referenztypen
  - Kollektionstypen

DBS-Markt

Datenstrukturen \ Anfragen	einfach	komplex
komplex	OO	OR
einfach	Dateisysteme Video-Server	RM (SQL)

- **Einfache Daten, einfache Anfragen**
  - Datenstruktur ist dem System nicht bekannt
  - Künftig werden solche Systeme wahrscheinlich mit Anfragemöglichkeiten (z. B. SQL) ausgestattet
- **Einfache Daten, komplexe Anfragen**
  - RDBS: skalierbar, robust, Zugriff über Struktur und Inhalt
  - Begrenzte Unterstützung für komplexe, als BLOBs gespeicherte Daten
  - RDBS können diese BLOBs nicht indexieren, manipulieren oder über ihren Inhalt suchen
- **Komplexe Daten, einfache Anfragen**
  - Persistente komplexe Objekte, die durch Java, C++, SmallTalk, ... manipuliert werden
  - Begrenzte Skalierbarkeit in Bezug auf große Datenvolumina und große Anzahlen von Benutzer
- **Komplexe Daten, komplexe Anfragen**
  - OR-Server können komplexe Daten als Objekte handhaben
  - Benutzerdefinierte Funktionen lassen sich zur Manipulation der Daten im Server heranziehen
  - Erweiterbarkeit ist für Datentypen und Funktionen möglich

Objekt-Relationale Datenbank-Technologie

- **Funktionalität wird derzeit im wesentlichen durch den Standard SQL99 beschrieben**

- **Unterstützung von benutzerdefinierten Typen (UDT) bzw. Objektorientierung**
  - komplexe Datenstrukturen mit
  - komplexer Funktionalität definierbar
  - Vererbungshierarchie
  - . . .

  ➥ Repräsentation von Anwendungswissen im DB-System (Klassen-Bibliotheken)

- **Erweiterung von herkömmlichen Tabellen**
  - komplexe Spalten (Attribute, Wertebereiche)
  - Schachtelung
  - Referenzierung/Dereferenzierung
  - Tabellen mit Typbindung (typed tables) und Tabellenhierarchien
  - . . .

- **Erweiterungsinfrastruktur**
  - benutzerdefinierte Datentypen und Funktionen lassen sich in das ORDBS integrieren und sind in SQL nutzbar
  - Unterstützung durch spezielle Zugriffspfade und Speicherungsstrukturen
  - Integration mit DBS-Komponenten wie Optimizer, Synchronisation, Logging und Recovery
  - . . .

- **Regelsysteme**
  ➥Integritätssicherung, Geschäftsregeln

Objekt-Relationale Erweiterungen[1]

- **Benutzerdefinierte Typen (UDT) beschreiben die Anwendungsdaten**

- **Benutzerdefinierte Routinen (UDR) definieren ein Verhalten für die Anwendungsdaten**

- **Große Objekte (LOB) bis GByte werden unterstützt**

- **Diese Erweiterungen erlauben mächtigere SQL-Anfragen**

  ➥ einfachere und bessere Anwendungsentwicklung und -optimierung

  ➥ einfachere, schnellere und mächtigere SQL-Anfragen

  ➥ bessere Entscheidungsunterstützung, mächtigere Anfragegeneratoren

- **Trigger/Constraints erlauben**

  ➥ Verbesserung der Datenintegrität

  ➥ bessere Modellierung der Anwendungssemantik

  ➥ Implementierung von Anwendungsregeln („Geschäftsregeln“)

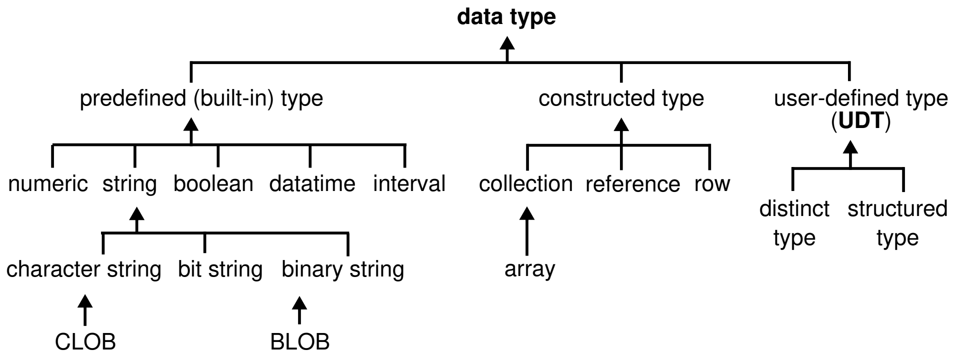
- **Ziel**
  <u>offene Architektur</u> für SQL-Klassenbibliotheken

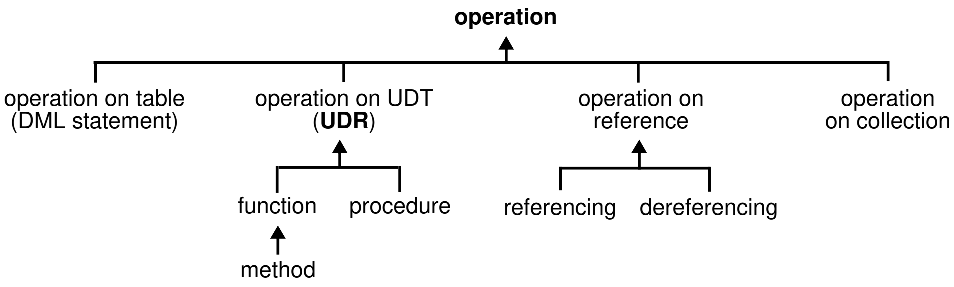
  ➥ erlaubt Anwendern die Integration von Funktionalität von externen Anbietern

1. The features of SQL99 can be crudely partitioned into its „relational features“ and its „object-oriented features“. „Relational“ is more appropriately categorized as „features that relate to SQL‘s traditional role and data model“. “Object-oriented features“ are focussed on adding support for object-oriented concepts to the SQL language.

Benutzerdefinierte Typen und Routinen(3)

- **Überblick**





Typklassifikation

- **Benutzerdefinierte Typen (UDT) und Routinen (UDR)**[1]
  - **Umbenannte Typen**
    - fördern *strong typing*, erlauben die Spezifikation von Verhalten
    - auch einzigartige Typen (*distinct types*) genannt
  - **Strukturierte Typen** (ADT-ähnlich, grob: Klassen in OO-Terminologie)
    - definieren bestimmte Objekteigenschaften
    - Zustand (Datenstrukturen) **+** Verhalten (zugehörige Routinen)
    - Bildung von Subtypen und dabei Nutzung von Vererbung:
      Ohne Typhierarchien keine Tabellenhierarchien!
    - Überladen, Überschreiben, spätes Binden
    - sind verwendbar als **Parametertypen** oder **Attributtypen** oder dienen zur **Definition von Tabellen**
      (die Zeilen mit Objekteigenschaften aufnehmen können)

PNR	Name	Ort
. . .	. . .	Straße Stadt Land PLZ

Staße	Stadt	Land	PLZ
. . .	. . .	. . .	. . .

1. The features of SQL:1999 can be crudely partitioned into its „relational features“ and its „object-oriented features“. „Relational“ is more appropriately categorized as „features that relate to SQL‘s traditional role and data model“. “Object-oriented features“ are focussed on adding support for object-oriented concepts to the SQL language.

Typklassifikation (2)

- **Konstruierte Typen (Typkonstruktoren)**
  - **Unbenamte Zeilen- und Spaltentypen**
    - sind aus vordefinierten, benutzerdefinierten und konstruierten Typen zusammengesetzt (haben keine Typnamen: „unnamed row types")
    - verwendbar als **Datentypen von Zeilen und Spalten**
    - Schachtelung von Zeilen (*nested rows*)
    - aber: keine Objekteigenschaften!
  - **Referenztypen**
    - definiert auf strukturierten Typen
    - eindeutige Identifikation der Zeilen (neben Primärschlüssel)
    - Pfadausdrücke, Navigation
    - erforderlich zur Referenzierung von Zeilen in Tabellen
  - **Kollektionstypen**
    - erlauben neben individuellen Werten auch die Speicherung von zusammengesetzten Werten (*composite values*) als Attributwerte
    - z. Zt. nur ARRAY

Umbenannte Typen

- **Eigenschaften**
  - verbesserte Modellierung, erhöhte Gewährleistung von Typsicherheit
  - jedoch keine Vererbung und keine Bildung von Subtypen (FINAL), immer INSTANTIABLE
  - Umbenennung des Typs gewöhnlich damit verbunden, ein zu seinem Basistyp unterschiedliches Verhalten zu erreichen

- **Beispiel**

  CREATE TYPE **ALTER** AS INTEGER FINAL;

- **Einfaches Arbeiten mit umbenannten Typen** verlangt (neben Casting-Funktionen)
  - Übernahme von Operationen vom Basistyp (Quellenfunktion) und/oder
  - Definition neuer, eigener Operationen

- **Quellenbasierte Funktionen**
  - Eine quellenbasierte Funktion ist eine neue Funktion, die auf einer bereits existierenden Quellenfunktion basiert
  - Quellenfunktionen von INTEGER wie +, –, SUM, AVG können vom Typ ALTER übernommen werden, um entsprechende Operationen direkt auszuführen:

    SELECT Alter1 + alter (Iq1) AS Kennziffer

  - Beispiel:

    "+" (ALTER, ALTER) sei vom Erzeuger des Typs ALTER als quellenbasierte Funktion angelegt:

```
CREATE FUNCTION "+" (ALTER, ALTER)
    RETURNS ALTER
    SOURCE "+" (INTEGER, INTEGER)
```

Strukturierter Typ als SQL-Objekttyp

- **Ziel:**
  Objektorientierung für die Zeilen von Tabellen

- **Strukturierter Typ**
  dient zur Typisierung der in einer Tabelle gespeicherten Objekte
  - Aus Attributen des strukturierten Typs werden „Spalten“ der darauf definierten Tabelle (*typed table*)
  - Eine **Extra-Spalte** definiert (eindeutige) REF-Werte für die Zeilen
    - systemgeneriert: REF IS SYSTEM GENERATED
    - benutzergeneriert: REF USING <vordefinierter Typ>
    - abgeleitet: REF <Attributliste> (UNIQUE NOT NULL)
  - Explizite Definition eines strukturierten Typs erlaubt seine Verwendung bei mehreren TABLE-Definitionen
  - Konzept ist wichtig für Tabellenhierarchien

- **Beispiel:**

```
CREATE TYPE Immob_t AS (                          // Strukturierter Typ

   ( Besitzer    REF (Person),
     E-Preis     GELD,
     V-Preis     GELD,
     . . .
     Ort         Adresse,
     Ansicht     Bitmap) INSTANTIABLE NOT FINAL
     REF IS SYSTEM GENERATED
     METHOD Gewinn RETURNS GELD;

CREATE METHOD Gewinn ... BEGIN <Programmcode> END;

CREATE TABLE Immobilien OF Immob_t          // Tabelle mit Typbindung
   (REF IS OID SYSTEM GENERATED)   // selbst-referenzierendes Attribut
```


Strukturierte Typen (2)

- **Schachtelung von strukturierten Typen ist möglich**

- **Automatisch generierte Funktionen (Methoden) bei CREATE TYPE**
  - **Konstruktor-Methode** erzeugt Instanzen vom strukturierten Typ, initialisiert mit Defaultwerten:
    Adresse () → Adresse
  - **Observer- und Mutator-Methoden** (O, M) manipulieren einzelne Attribute. Sie können
    - als Grundfunktion zur Implementierung von zusätzlichem Verhalten eingesetzt oder
    - an der Objektschnittstelle (Signatur) zur Verfügung gestellt werden. (a sei Adressen-Variable/Ausdruck)
    - nicht überladen werden.

      a.Straße () → CHAR (30)
      a.PLZ (INTEGER) → Adresse

- **Definition von weiteren Methoden/Funktionen, wie z. B. Gewinn**
  - in SQL oder
  - in einer externen Programmiersprache (wie C)

- **Zugriff mit O-Methoden: Beispiel**

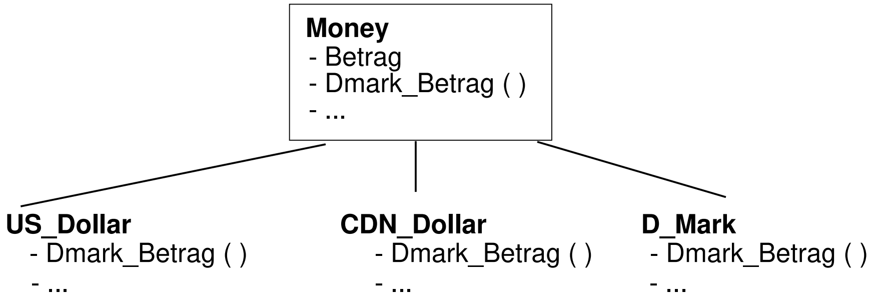
```
CREATE TABLE Adressenliste
    ( LfdNr      INTEGER PRIMARY KEY,
      Ort        Adresse);

SELECT   a.Ort.Stadt (), a.Ort.PLZ ()
INTO     :x, :y
FROM     Adressenliste a
WHERE    a.Ort.PLZ > 80000;
```

Strukturierte Typen und Vererbung

- **Strukturierter Typ kann Subtyp eines anderen strukturierten Typs sein**
  - Vererbung von Attributen und Verhalten (Methoden) vom Supertyp
  - Überschreibung von Verhalten
  - CREATE TYPE **Money** AS (Betrag ...) NOT INSTANTIABLE NOT FINAL;

- Bisher keine Unterstützung für **Mehrfachvererbung!** (➥ SQL4)



- **Substituierbarkeit:**
  jede Zeile kann eine Instanz eines verschiedenen Subtyps haben

```
CREATE TABLE  Immobilien_Info
  (Preis      Money,
   Besitzer   CHAR (40),
   Grundstück Adresse)
```

Preis	Besitzer	Grundstück
<US_Dollar> Betrag: 100.000	‘S.Weiss‘	<Adresse>
<CDN_Dollar> Betrag: 400.000	‘Dr.W.Gruen‘	<Adresse>
<D_Mark> Betrag: 150.000	‘D.Schwarz‘	<Adresse>

```
SELECT  Besitzer, Dmark_Betrag(Preis)
FROM      Immobilien_Info
WHERE     Dmark_Betrag(Preis) <
          D_Mark (500.000)
```

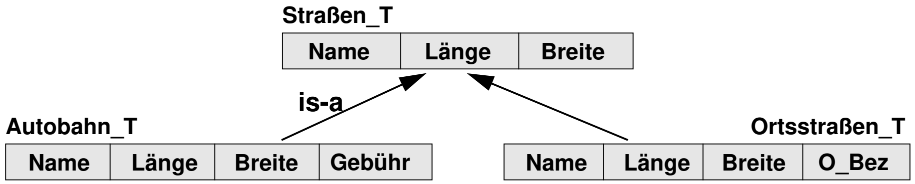
- **Dynamische Auswahl** (dynamic *dispatch* nur bei Methoden!) und **spätes Binden** der Werte von **Preis** auf der Basis des Typs **Money**

Typ- und Tabellenhierarchie

- **Strukturierter Typ Straßen_T**
  CREATE TYPE Straßen_T **AS** (
  Name **CHAR** (40),
  Länge **DECIMAL** (9,2),
  Breite **DECIMAL** (5,2)) **NOT FINAL . . .**;

- **Subtypen**
  CREATE TYPE Autobahn_T **UNDER** Straßen_T(Gebühr Money) **NOT FINAL ...**;
  CREATE TYPE Ortsstraßen_T **UNDER** Straßen_T(O_Bez Orte) **NOT FINAL ...**;

- **Typhierarchie**



- **Tabellenhierarchie**
  - **CREATE TABLE** Straßen *// Supertabelle Straßen*
    OF Straßen_T (**PRIMARY KEY** Name, . . .);
  - **CREATE TABLE** Autobahnen *// Subtabelle Autobahnen*
    OF Autobahn_T **UNDER** Straßen;
  - **CREATE TABLE** Ortsstraßen *// Subtabelle Ortsstraßen*
    OF Ortsstraßen_T **UNDER** Straßen;

Konstruierte Typen: Tupeltyp

- **Ziel: Einfacher Umgang mit zusammengesetzten Werten**
- **Tupeltyp (*ROW Type*) als Datentyp von zusammengesetzten Attributen**
  - Zusammengesetztes Attribut kann in **einer** Spalte gespeichert werden
  - Tupel kann als **ein** Argument an Routinen und als Rückgabewert von Funktionen dienen
  - entspricht Record-Typen in Programmiersprachen
- **Tupeltyp** (früher: *unnamed row type*)
  - Definition von geschachtelten Tabellenstrukturen
  - spezielle Operationen:
    Konstruktor, Zuweisung, Vergleich von zusammengesetzten Werten
- **Beispiel:**

```
CREATE TYPE Straßen_T AS (                              // Strukturierter Typ
    Name            VARCHAR (40),
    Verwaltung      ROW (Bezeich VARCHAR (20),           // Tupeltyp
                         Stadt   VARCHAR (30)),
    Geometrie       Polygon,
    Referenzpunkt   ROW ( Rechts GK_Koordinate,          // Tupeltyp
                          Hoch   GK_Koordinate,
                          Höhe   NN_Höhe)) NOT FINAL . . .;
```


Konstruierte Typen: Referenztypen

- **Ziel: Verweise (Referenzen) auf andere Objekte**
- **Eigenschaften**
  - Referenzen repräsentieren Beziehungen zwischen Objekten
  - Ein Objekt kann von vielen anderen referenziert werden
  - Referenz verweist immer auf eine Zeile (Objekt) einer getypten Tabelle, nicht auf ein Objekt in einer Spalte
- **Verwendung**
  - Referenztypen können mit Zeilentypen kombiniert werden
  - Sie erleichtern die Modellierung von Beziehungen zwischen Typen
  - Wertebereich von Referenzen kann durch die SCOPE-Klausel eingeschränkt werden
  - Bei mehreren Tabellendefinitionen mit gleichem strukturierten Typ lassen sich für Referenzen unterschiedliche Gültigkeitsbereiche festgelegen
- **Beispiel**

```
CREATE TYPE Kunden_T AS (
    KNR         INTEGER,
    Name        CHAR (50),
    Anschrift   Adresse)
    NOT FINAL REF USING INTEGER;                    //Strukturierter Typ

CREATE TABLE Kunden OF Kunden_T (
    PRIMARY KEY KNR,
    REF IS kid USER GENERATED);      // selbst-referenzierendes Attribut

CREATE TABLE Privat_Kunden OF Kunden_T (
    PRIMARY KEY KNR,
    REF IS pid USER GENERATED);      // selbst-referenzierendes Attribut
```

Referenztypen (2)

- **Beispiel** (Fortsetzung)

```
CREATE TYPE Konto_T AS (
    Konto_Nr     INTEGER,
    Kunde        REF (Kunden_T),   // Verweis auf den zugehörigen Kunden
    Typ          CHAR (1),
    Eröffnet     DATE,
    Zinsrate     DOUBLE PRECISION,
    Kontostand   DOUBLE PRECISION
    ) NOT FINAL REF Konto_Nr;

CREATE TABLE Konto OF Konto_T (
    PRIMARY KEY Konto_Nr,
    REF IS koid DERIVED,
    Kunde WITH OPTIONS SCOPE Kunden);
```

Kunden

kid	KNR	Name	Anschrift
	. . .	. . .	Straße Stadt Land PLZ

Privatkunden

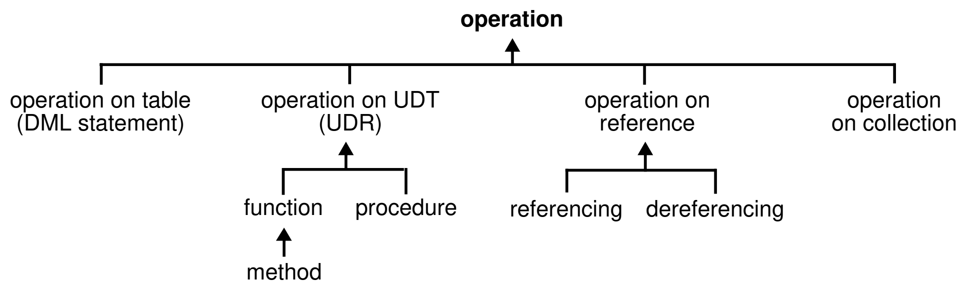
pid	KNR	Name	Anschrift
	. . .	. . .	Straße Stadt Land PLZ

Konto

koid	Konto_Nr	Kunde	Typ	. . .	Kontostand
	. . .	. . .	. . .	. . .	. . .

Von SQL aufrufbare Routinen (2)

- **Überblick**



- **Klassifikation nach Formen**
  - **Benutzerdefinierte Funktionen**
    - geben immer einzelne Werte als Ergebnis zurück
    - Überladbar, Typüberprüfung zur Übersetzungszeit
    - Auswahl über statische Parametertypen (statisches Binden)
  - **Benutzerdefinierte Prozeduren**
    - Aufgerufen durch eine CALL-Anweisung
    - Statisches Binden, kein Überladen
  - **Benutzerdefinierte Methoden**
    - sind Funktionen mit speziellen Aufrufkonventionen (p.JahresGehalt())
    - Überladbar, redefinierbar (überschreibbar)
    - „dynamic dispatch"; Laufzeit-Funktion von „SELF" bestimmt
    - Spätes Binden

SQL-Routinen

- **SQL-Prozeduren**
  - **Definitionsbeispiel**

```
CREATE PROCEDURE Kontoabfrage (IN KontoNr INT,
                               OUT Betrag DECIMAL (15,2))
BEGIN
   SELECT      Kontostand INTO Betrag
      FROM     Konten
      WHERE    Kontonummer = KontoNr;
   IF Betrag < 100
      THEN     SIGNAL Niedriger_Kontostand
   END IF;
END
```

  - **Aufruf durch CALL-Anweisung**

```
CALL Kontoabfrage (4711, Betrag);
```

- **Externe Prozeduren**

```
CREATE PROCEDURE Kontoabfrage (IN KontoNr INT,
                               OUT Betrag DECIMAL (15,2))
LANGUAGE C
EXTERNAL NAME 'Konten/Abfrage_Prozedur'
```


SQL-Routinen (2)

- **SQL-Funktionen**
  - **Definitionsbeispiel**

```
CREATE FUNCTION Kontoabfrage (KontoNr INT)
RETURNS DECIMAL (15,2)
   BEGIN
      DECLARE     Betrag DECIMAL (15,2);
      SELECT      Kontostand INTO Betrag
         FROM     Konten
         WHERE    Kontonummer = KontoNr;
      IF Betrag < 100
         THEN SIGNAL Niedriger_Kontostand
      END IF;
      RETURN Betrag
   END
```

  - **Funktionsaufruf als Teil eines Ausdrucks**

```
SELECT Kontonummer, Kontoabfrage (KontoNr)
FROM Konten
```

- **Externe Funktionen**

```
CREATE FUNCTION Kontoabfrage (KontoNr INT)
RETURNS DECIMAL (15,2)
LANGUAGE C
EXTERNAL NAME 'DBA/Konten/Abfrage'
```

Zusammenfassung

- **Objekt-Relationale Erweiterungen sind zentral**
  - Erweiterbares Typsystem
    - Benutzerdefinierte Typen (UDT) beschreiben die Anwendungsdaten
    - Benutzerdefinierte Routinen (UDR) definieren ein Verhalten für die Anwendungsdaten
  - Regeln und Trigger
  - Große Objekte (LOB) bis GByte werden unterstützt

- **Diese Erweiterungen erlauben mächtigere SQL-Anfragen**
  - einfachere und bessere Anwendungsentwicklung und -optimierung
  - einfachere, schnellere und mächtigere SQL-Anfragen
  - bessere Entscheidungsunterstützung, mächtigere Anfragegeneratoren

- **Trigger/Constraints erlauben**
  - Verbesserung der Datenintegrität
  - bessere Modellierung der Anwendungssemantik
  - Implementierung von Anwendungsregeln („Geschäftsregeln“)

- **Ziel: offene Architektur für SQL-Klassenbibliotheken**

➥ erlaubt Anwendern die Integration von Funktionalität von externen Anbietern

- **Wettbewerber**
  - IBM: DB2 Universal Database V7, mit Parallelität verfügbar
  - Informix Dynamic Server 2000
  - Oracle 8i
  - Microsoft SQL Server 7.5, Sybase Adaptive Server
  - CA Associates (OpenIngres ++?), Software AG (Adabas C ++?)

Zusammenfassung (2)

- **SQL:1999 wurde hier nur teilweise vorgestellt!**
  - SQL:1999 ist vollständig aufwärtskompatibel zu SQL2
  - Mächtigkeit einer modernen Programmiersprache (SQL/PSM)
  - Allgemeine Tabellenausdrücke, Rekursion
  - **Verbesserte Orthogonalität der Anfragesprache**

Pers (Pnr, ...) → Mitarbeit (Pnr, Jnr, ...) ← Proj (Jnr, ...)

Anfrage: Finde die Angestellten (PNR), die in allen Projekten mitarbeiten (mit ausschließlicher Hilfe mengentheoretischer Operationen).

```
(SELECT DISTINCT Pnr FROM Pers)
MINUS
(SELECT DISTINCT Pnr
FROM   (
            (SELECT  Pnr, Jnr
             FROM        (SELECT DISTINCT Pnr FROM Pers),
                         (SELECT DISTINCT Jnr FROM Proj)
            )
        MINUS
           ( SELECT DISTINCT Pnr, Jnr FROM Mitarbeit )
        )
)
```

- In der FROM-Klausel sind neben Basistabellen (und Views) auch durch ein geschachteltes SELECT berechnete Tabellen zulässig.
- Es werden zwei einspaltige (Pnr bzw. Jnr) berechnete Tabellen genutzt.

```
FROM   (SELECT DISTINCT Pnr FROM Pers),
       (SELECT DISTINCT Jnr FROM Proj)
```