4.9 Datentyp TABLE (vereinfacht)

Definition einer Tabelle am Beispiel

```
CREATE TABLE Stadt (
   SName       VARCHAR(50),
   PName       VARCHAR(50),
   LCode       CHAR(4),
   Einwohner   INTEGER,
   LGrad       NUMBER,
   BGrad       NUMBER,
   PRIMARY KEY (SName,PName,LCode) )
```


4.10 Konstruierte Datentypen

Ein Datentyp heißt *konstruiert*, sofern seine Werte aus Werten anderen Typen, sogenannter *Element-Typen*, zusammengesetzt sind.

- Der Datentyp `ARRAY` fasst mehrere Werte seines Element-Typs geordnet zusammen, die über einen Index referenziert werden können.
- Der Typ `ROW` lässt hingegen zu, dass Werte unterschiedlicher Element-Typen geordnet zusammengefasst werden und das der Zugriff auf die einzelnen Komponenten über Bezeichner, analog zu Spaltenbezeichern, ermöglicht wird.
- Der Datentyp `MULTISET` fasst mehrere Werte eines Element-Typs, möglicherweise mit Duplikaten, zu einer ungeordneten Menge zusammen.

```
CREATE TABLE Land (
   ⋮
   Provinzen VARCHAR(50) ARRAY[20]
   Organisationen
      ROW(Organiation VARCHAR(50), Art VARCHAR(20)) MULTISET
CREATE TABLE Stadt (
   ⋮
   Koordinaten ROW(LGrad NUMBER, BGrad NUMBER )
```


- Die fünfte Provinz eines Landes wird mittels `Provinzen[5]` referenziert.
- Der Längengrad innerhalb der Koordinaten einer Stadt kann mittels eines Pfadausdrucks der Form `Koordinaten.LGrad` angesprochen werden.
- Mittels `('EU','member') ELEMENT Organisationen` wird getestet, ob die Mitgliedschaften eines Landes bezüglich der EU den *member*-Status hat.

4.11 Einfügen, Löschen und Ändern

Einfügen

- Mittels `INSERT` kann eine neue Zeile, oder eine Menge von neuen Zeilen in eine Tabelle `T` eingefügt werden.
- Werden nicht zu allen Attributen von `T` Werte gegeben, so werden für die fehlenden Werte möglicherweise vorgesehene Default-Werte, bzw. der Nullwert `null` genommen.
- Die Angabe der Spaltennamen kann entfallen, wenn die Werte in der Reihenfolge der Spaltennamen in der `CREATE`-Anweisung definiert werden.

Aufnahme eines neuen Mitgliedes in die EU.

```
INSERT INTO Mitglied (LCode, Organisation, Art)
   VALUES ('PL', 'EU', 'member')
```

Alle Länder die Mitglied in irgendwelchen Organisationen sind, die aber noch nicht in der Relation `Land` auftreten, werden in diese Relation übernommen.

```
INSERT INTO Land ( LCode )
   SELECT DISTINCT M.LCode
      FROM Mitglied M
      WHERE NOT EXISTS (
         SELECT L.LCode
            FROM Land L
            WHERE L.LCode = M.LCode)
```

Sequenznummern

Beim Einfügen von Zeilen muss die Eindeutigkeit des Primärschlüssels gewährleistet sein.

Beispiel Sequenznummerngenerator.

```
CREATE SEQUENCE LandSEQ AS INTEGER
     START WITH 1       /*kleiner (größer) oder gleich MAXVALUE (MINVALUE) sein*/
     INCREMENT BY 1     /*eine positive (negative) Zahl*/
     MINVALUE 1         /*muss kleiner MAXVALUE sein*/
     MAXVALUE 100000    /*muss größer als MINVALUE sein*/
     NO CYCLE           /*CYCLE: wenn MAXVALUE (MINVALUE) erreicht, */
                        /*beginne wieder mit MINVALUE (MAXVALUE)*/
                        /*Mehrfachverwendung von Werten führt u.U. zu einem */
                        /*Laufzeitfehler*/


INSERT INTO Land
   (LandNr, LName, HStadt, Fläche)
   VALUES ( NEXT VALUE FOR LandSEQ,'Bavaria','Munich',70)
```


Beispiel Identitätsspalte.

```
CREATE TABLE Land
   LandNr INTEGER GENERATED ALWAYS AS IDENTITY
   ( START WITH 1
   INCREMENT BY 1
   MINVALUE 1
   MAXVALUE 100000
   NO CYCLE),
   ⋮

INSERT INTO Land
   (LName, HStadt, Fläche)
   VALUES ( 'Bavaria', 'Munich', 70)
```

generierte Spalten

Der Wert eines Attributes ergibt sich automatisch aus den Werten anderer Attribute desselben Tupels.

Beispiel.

```
CREATE TABLE Land
   ( LandNr INTEGER GENERATED ALWAYS AS IDENTITY ( ...),
   ⋮,
   Fläche      NUMBER,
   Einwohner   NUMBER,
   Dichte GENERATED ALWAYS (Einwohner / Fläche))
```


Löschen

- `DELETE FROM T WHERE P`
- Es werden alle Tupel aus `T`, für die der bedingte Ausdruck `P` wahr ist, markiert und anschließend aus `T` entfernt.

Beispiel Löschen des gesamten Inhalts der Tabelle `Stadt`.

```
DELETE FROM Stadt
```

Löschen von ausgewählten Zeilen.

```
DELETE FROM Stadt
   WHERE Einwohner <
      (SELECT AVG(Einwohner) FROM Stadt)
```

Ändern

```
UPDATE T
   SET A_1 = val_1,..., A_n = val_n
   WHERE P
```

- Anstelle eines direkten Wertes innerhalb einer Zuweisung kann auch ein skalarer Ausdruck stehen.

Im Zuge der Euro-Umstellung, werden die Angaben des Bruttosozialprodukt angepasst.

```
UPDATE Land
   SET BruttoSP =
      CASE BruttoSP
         WHEN LCode = 'D' THEN BruttoSP * 0,5
         WHEN LCode = 'F' THEN BruttoSP * 0,16
         ELSE NULL
      END
```


4.12 Sichten

- Eine Sicht `V` ist eine durch einen Anfrageausdruck `E` definierte Tabelle:
- `CREATE VIEW V AS`
  `<E>`
- Im Unterschied zu den als Sicht definierten Tabellen bezeichnen wir die mittels `CREATE TABLE` definierten Tabellen als *Basistabellen*.
- Bezeichner von Sichten dürfen in SQL überall stehen, wo ein Tabellenbezeichner stehen darf.

Definiere zu der Tabelle `Benachbart` eine bezüglich Symmetrie abgeschlossene Tabelle `symBenachbart` in Form einer Sicht.

Benachbart

LCode1	LCode2
CH	D
CH	F
CH	I
D	F
I	F

```
CREATE VIEW symBenachbart AS
   SELECT LCode1 AS Von, LCode2 AS Nach
   FROM Benachbart
      UNION
   SELECT LCode2 AS Von, LCode1 AS Nach
   FROM Benachbart
```

Welche Länder sind zu Deutschland benachbart?

```
SELECT Nach FROM symBenachbart
   WHERE Von = 'D'
```


Materialisierte und virtuelle Sichten

- Ein Datenbanksystem kann Sichten entweder bei Bedarf jeweils neu berechnen, oder eine einmal berechnete Sicht für weitere Bearbeitungen permanent speichern. Im ersten Fall redet man von einer *virtuellen* Sicht, im zweiten Fall von einer *materialisierten* Sicht.
- Soll eine Anfrage bearbeitet werden, die sich auf eine virtuelle Sicht bezieht, so wird vor Ausführung der Anfrage der Name der Sicht durch den sie definierenden Ausdruck ersetzt (*Anfrage-Modifizierung*).
- Gegenüber einer materialisierten Sicht hat eine virtuelle Sicht den Vorteil, dass ihr Inhalt garantiert dem aktuellen Zustand der Datenbank entspricht.
- Standardmäßig ist eine Sicht einer Datenbank virtuell. Materialiserte Sichten werden typischerweise für sogenannte *Datenlager* (*Data-Warehouses*) eingesetzt; zu ihrer Aktualisierung ist häufig ein erheblicher organisatorischer und systemtechnischer Aufwand vonnöten.
- Im folgenden betrachten wir ausschließlich virtuelle Sichten.
- Eine interessante Frage ist, ob in einer virtuellen Sicht Einfügen, Löschen und Ändern von Zeilen erlaubt sein kann, oder nicht.

Ändern von Sichten

Sei $\mathcal{R}$ ein Datenbank-Schema.

- Eine *Datenbankänderung* ist eine Funktion t von der Menge der Instanzen zu $\mathcal{R}$ auf sich selbst.
- Eine *Sicht* ist eine Funktion f von der Menge aller Instanzen zu $\mathcal{R}$ in die Menge aller Instanzen zu S, wobei S das durch die Sicht definierte Relationsschema ist.
- Eine *Sichtänderung* ist eine Funktion u von der Menge aller Instanzen zu S auf sich selbst.
- Sei u eine Sichtänderung und t eine Datenbankänderung, so dass für jede Datenbank-Instanz $\mathcal{I}$ gilt:

$$u(f(\mathcal{I})) = f(t(\mathcal{I})),$$

dann nennen wir t eine *Transformation* von u.

- Auf einer Sicht sind grundsätzlich nur solche Änderungen zulässig, zu denen eine Transformation existiert.

Beispiel

$r =$

A	B
a	b
x	b

$s =$

B	C
b	c
b	z

$v = r \bowtie s =$

A	B	C
a	b	c
a	b	z
x	b	c
x	b	z

insert (y,b,c) in v

Es existiert keine Transformation.

delete (a,b,c) in v

Es existiert keine Transformation.

update (a,b,c) zu (y,b,c) in v

Es existiert keine Transformation.

Projektionssicht

Einfügen, Löschen und Ändern ist nur dann erlaubt, wenn der Schlüssel der Basistabelle komplett in der Sicht vorhanden ist.

Informationen über Länder sind nur anonymisiert erlaubt.

```
CREATE VIEW LandInfo AS
   SELECT Fläche, Einwohner
      FROM Land

INSERT INTO LandInfo VALUES (250000,20)
```

INSERT nicht zulässig.

Selektionssicht

- Aufgrund von Einfügungen und Änderungen können als unerwünschter Seiteneffekt Zeilen aus der Sicht herausfallen und damit in anderen Sichten erkennbar werden.
- Dieser Seiteneffekt kann durch Hinzunahme der Klausel `WITH CHECK OPTION` zu der Sichtdefinition verhindert werden.

Beschränkung auf Großstädte.

```
CREATE VIEW Großstadt AS
   SELECT *
      FROM Stadt
      WHERE Einwohner >= 1000

UPDATE Großstadt SET Einwohner=Einwohner*0.9
```

Verbundsicht

In SQL-92 und in SQL:1999 werden eine Reihe von Regeln diskutiert, deren Erfülltsein die Existenz einer Transformation sichern. Diese Regeln garantieren im Wesentlichen ein eindeutiges Rückverfolgen der Sichtänderung zu einzelnen Zeilen in den Basistabellen.

Ordne jeder Stadt ihren Kontinent zu.

```
CREATE VIEW StadtInfo AS
   SELECT S.SName, L.Kontinent
      FROM Stadt S, Lage L
      WHERE S.LCode = L.LCode

INSERT INTO StadtInfo VALUES ('Freiburg', 'DreiLänderEck')
```

INSERT nicht zulässig.

In-line/temporary-table Sichten mittels `WITH`

Vergleiche:

```
SELECT DISTINCT S.LCode FROM Stadt S
WHERE (SELECT AVG(Einwohner) FROM Stadt WHERE LCode = S.LCode) =
      (SELECT MIN(Einw) FROM (SELECT AVG(Einwohner) AS Einw
                              FROM Stadt GROUP BY Lcode));
```

mit

```
WITH
avgEinwohner AS (SELECT LCode, AVG(Einwohner) AS Einw
                 FROM Stadt GROUP BY Lcode),
minEinwohner AS (SELECT MIN(Einw) as minE FROM avgEinwohner)
SELECT LCode FROM avgEinwohner S
WHERE S.Einw = (SELECT minE FROM minEinwohner);
```

WITH führt zu einer klareren Struktur,

im Sinne einer lokal definierten

- virtuellen Sicht (*in-line view*),
- materialisierten Sicht (*temporären Tabelle*).

4.13 SQL und Programmiersprachen

- SQL hat eine tabellenorientierte Sematik.
- SQL hat eine im Vergleich zu einer Programmiersprache eingeschränkte Mächtigkeit, so dass es typischerweise ermöglicht wird, SQL von einem in einer gängigen Programmiersprache geschriebenen Programm aus aufzurufen.
- Anwendungen können so unter Ausnutzung der vollen Mächtigkeit einer Programmiersprache Datenbank-Instanzen verarbeiten.
- *Impedance-Mismatch*!

Ansätze der Integration

(A) Erweiterung von SQL um imperative Sprachelemente zur Formulierung von *in der Datenbank gespeicherten* benutzerdefinierten Funktionen und Prozeduren (*SQL-Erweiterung*).

(B) Einbettung von SQL-Ausdrücken in eine Programmiersprache (*statisches SQL*).

(C) Übergabe eines datenabhängig gebildeten SQL-Ausdrucks an eine Datenbank während der Ausführung eines Programms (*dynamisches SQL*).

(A) SQL-Erweiterung[4]

- *Deklaration* von *Variablen*,
- *Zuweisung* von Werten an Variable,
- *Sequenz* von Anweisungen,
- *bedingte Anweisungen* und *Wiederholungsanweisungen*,
- Funktionen und Prozeduren.

Funktionen können als parametrisierbare virtuelle Sichten betrachtet werden.

[4] Oracle verwendet teilweise eine andere Syntax!

Tabellen

Benachbart

LCode1	LCode2
CH	D
CH	F
CH	I
D	F
I	F

symBenachbart

LCode1	LCode2
CH	D
D	CH
CH	F
F	CH
CH	I
I	CH
⋮	⋮

Berechne zu der Tabelle `Benachbart` den symmetrischen Abschluss.

```
CREATE FUNCTION symBenachbart()
RETURNS TABLE (
      LCode1 CHAR(4),
      LCode2 CHAR(4) )
RETURN (SELECT * FROM Benachbart
      UNION
      SELECT LCode2 AS LCode1, LCode1 AS LCode2 FROM Benachbart )

SELECT * FROM TABLE (symBenachbart()) T
```

Gib zu einem Land alle erreichbaren Länder mit der Mindestanzahl Grenzübergänge an.

```
CREATE FUNCTION ErreichbarVon(Start CHAR(4))
RETURNS TABLE ( Nach CHAR(4), Anzahl INTEGER )
BEGIN
CREATE TABLE Erreichbar (
      Nach CHAR(4),
      Anzahl INTEGER );
DECLARE alt, neu INTEGER;

/* Initialisiere mit den direkt erreichbaren Ländern */
INSERT INTO Erreichbar
      SELECT T.LCode2 AS Nach, 1 AS Anzahl
      FROM TABLE (symBenachbart()) T WHERE T.LCode1 = Start;

/* Initialisiere Abbruchbedingung */
SET alt = 0;
SET neu = (SELECT COUNT(*) FROM Erreichbar);

/* Berechne iterativ die indirekt erreichbaren Länder */
WHILE (alt <> neu) DO
      SET alt = neu;
      INSERT INTO Erreichbar
            SELECT DISTINCT B.LCode2, (A.Anzahl + 1)
            FROM Erreichbar A, TABLE (symBenachbart()) B
            WHERE A.Nach = B.LCode1 AND B.LCode2 <> Start AND B.LCode2 NOT IN
                  (SELECT Nach FROM Erreichbar);
      SET neu = (SELECT COUNT(*) FROM Erreichbar);
END WHILE;

RETURN Erreichbar;
END
```


Initialisiere `Erreichbar` mit den *direkt* erreichbaren Ländern.

```
INSERT INTO Erreichbar
      SELECT T.LCode2 AS Nach, 1 AS Anzahl
      FROM TABLE (symBenachbart()) T WHERE T.LCode1 = Start;
```

Die Berechnung soll terminieren, wenn kein neues *indirekt* erreichbares Land hinzugekommen ist.

```
SET neu = (SELECT COUNT(*) FROM Erreichbar);
WHILE (alt <> neu) DO
      SET alt = neu;
      INSERT INTO Erreichbar
      ...
      SET neu = (SELECT COUNT(*) FROM Erreichbar)
END WHILE;
```

Zyklen und Mehrfachberechnungen sollen vermieden werden.

- `DISTINCT` berücksichtigt den Fall, dass zum selben Land mehrere Wege gleicher Länge existieren können.
- Zyklen werden berücksichtigt, indem ein weiteres Land nur dann hinzugefügt wird, wenn das neu berechnete Land nicht bereits als erreichbares Land bekannt ist.
- Zyklen zum Start werden ebenfalls ausgeschlossen.

```
INSERT INTO Erreichbar
    SELECT DISTINCT B.LCode2, (A.Anzahl + 1)
    FROM Erreichbar A, TABLE (symBenachbart()) B
    WHERE A.Nach = B.LCode1 AND B.LCode2 <> Start AND B.LCode2 NOT IN
        (SELECT Nach FROM Erreichbar);
```


(B) statisches SQL

- Stehen die auszuführenden SQL-Anfragen bereits zur Übersetzungszeit eines Programms als Teil des Programmcodes fest, dann redet man von einer *statischen* Einbettung von SQL in eine Programmiersprache.
- Eine datenabhängige Änderung der Anfragen während der Ausführung des Programms ist dann nicht mehr möglich.
- Die Ergebnisse einer SQL-Anfrage werden innerhalb des Programms mittels eines Cursors zugänglich gemacht.

```
EXEC SQL DECLARE StadtCursor CURSOR FOR
   SELECT DISTINCT S.SName, L.LName
      FROM Stadt S, Land L
      WHERE S.LCode = Land.LCode;
```

```
  EXEC SQL OPEN StadtCursor;
```

```
  EXEC SQL FETCH StadtCursor INTO :stadtName, :landName;
```

```
  EXEC SQL CLOSE StadtCursor;
```


(C) dynamisches SQL

- Können wir einen SQL-Ausdruck während der Ausführung eines Programms in Form einer Zeichenkette an das Datenbanksystem übergeben, so redet man von *dynamischem* SQL.
- Standardisierte Schnittstellen: ODBC und JDBC. Erweiterung von Java: SQLJ.

Berechnen einer Adjazenzmatrix

```
<?php
sql_connect('Mondial','lausen','buch');
$AdjazenzMatrix = array();
$query = 'SELECT * FROM Benachbart';
$result = sql_query($query);
while ($row = sql_fetch_assoc($result)) {
      $i = $row['von']; $j = $row['nach'];
      $AdjazenzMatrix[$i][$j] = 1; }
      ...
?>
```