Kapitel 3: Der SQL-Standard

- SQL ist die in der Praxis am weitesten verbreitete Datenbanksprache für relationale Datenbanken.
- Die Historie von SQL geht zurück bis 1974, die Anfangszeit der Entwicklung relationaler Datenbanken.
- Der Sprachumfang von SQL ist einer permanenten Weiterentwicklung und Standardisierung unterworfen. Derzeit relevant sind der Stand von 1992, 1999 und 2003 entsprechend bezeichnet mit SQL-92, SQL:1999 und SQL:2003.
- Auf dem Markt befindliche Datenbanksysteme realisieren weitgehend die im Standard SQL-92 definierten Konzepte und bereits viele Konzepte der Standards SQL:1999 und SQL:2003.
- SQL und XML: später.

3.1 Überblick

Ein *Anfrageausdruck* in SQL besteht aus einer `SELECT`-Klausel, gefolgt von einer `FROM`-Klausel, gefolgt von einer `WHERE`-Klausel.

SFW-Ausdruck

```
SELECT A_1,...,A_n   (...Attribute der Ergebnisrelation)
   FROM R_1,...,R_m  (...benötigte Relationen)
   WHERE F           (...Auswahlbedingung)
```

äquivalenter Algebraausdruck

$$\pi[A_1, \ldots, A_n](\sigma[F](R_1 \times \cdots \times R_m))$$

äquivalenter Kalkülausdruck

$$\{(X_{i_1} : A_1, \ldots, X_{i_n} : A_n) \mid R_1(\vec{X}_1) \wedge \ldots \wedge R_m(\vec{X}_m) \wedge F\}$$

Mondial-Datenbank Teil 1

Land

LName	LCode	HStadt	Fläche
Austria	A	Vienna	84
Egypt	ET	Cairo	1001
France	F	Paris	547
Germany	D	Berlin	357
Italy	I	Rome	301
Russia	RU	Moscow	17075
Switzerland	CH	Bern	41
Turkey	TR	Ankara	779

Provinz

PName	LCode	Fläche
Baden	D	15
Bavaria	D	70,5
Berlin	D	0,9
Ile de France	F	12
Franken	D	null
Lazio	I	17

Stadt

SName	LCode	PName	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

Mondial-Datenbank Teil 2

Lage

LCode	Kontinent	Prozent
D	Europe	100
F	Europe	100
TR	Asia	68
TR	Europe	32
ET	Africa	90
ET	Asia	10
RU	Asia	80
RU	Europe	20

Mitglied

LCode	Organisation	Art
A	EU	member
D	EU	member
D	WEU	member
ET	UN	member
I	EU	member
I	NAM	guest
TR	UN	member
TR	CERN	observer

3.2 Einfache Anfragen

Welche Städtenamen sind in der Tabelle Stadt gespeichert?

```
SELECT SName FROM Stadt
```

Welche unterschiedlichen Namen haben die Städte?

```
SELECT DISTINCT SName FROM Stadt
```

..., die mehr als 1 Mio. Einwohner haben?

```
SELECT DISTINCT SName FROM Stadt
   WHERE ..................
```

Was ist bekannt über die Städte, die mehr als 1 Mio. Einwohner haben?

```
SELECT * FROM Stadt
   WHERE Einwohner > 1000
```

Kennzeichne die Tatsache, dass eine Stadt mehr als 1 Mio. Einwohner hat, durch den Wert 'Großstadt' einer neuen Spalte mit Namen StadtKategorie.

```
SELECT SName, 'Großstadt' AS StadtKategorie FROM Stadt
   WHERE Einwohner > 1000
```

Erstelle eine Liste der Länder, deren Namen mit 'G' anfängt oder mit 'y' aufhört?

```
SELECT * FROM Land
   WHERE LName LIKE 'G%' OR LName LIKE '%y'
```

Erstelle eine Liste der Länder, deren dritter Buchstabe des Namen 'y' ist?

```
SELECT * FROM Land
   WHERE LName LIKE '__y%'
```

Gib zu jedem Land die zugehörigen Städte an.

```
SELECT DISTINCT Stadt.SName AS Stadt, Land.LName AS Land
   FROM Stadt, Land
   WHERE Stadt.LCode = Land.LCode
```

... Verwende optionale Korrelationsnamen.

```
SELECT DISTINCT S.SName, L.LName
   FROM Stadt S, Land L
   WHERE S.LCode = L.LCode
```

Bestimme alle Paare von Ländern, die im selben Kontinent liegen.

```
SELECT DISTINCT L1.LCode AS Land1, L2.LCode AS Land2
   FROM Lage L1, Lage L2
   WHERE L1.Kontinent = L2.Kontinent
      AND L1.LCode < L2.LCode
```

Algorithmus (nested-loop-Semantik)

FOR each Tupel t_1 in Relation R_1 DO
FOR each Tupel t_2 in Relation R_2 DO
$\vdots$
FOR each Tupel t_m in Relation R_m DO
IF Die WHERE-Klausel ist erfüllt nach Ersetzen der Attributnamen in F durch die entsprechenden Werte der gerade betrachteten Tupel $t_1, \ldots, t_m$.
THEN Bilde ein Antwort-Tupel aus den Werten der in der SELECT-Klausel angegebenen Attributen $A_1, \ldots, A_n$ bezüglich der gerade betrachteten Tupel $t_1, \ldots, t_m$.

JOIN, NATURAL JOIN und OUTER JOIN

Erstelle eine Liste der Mitgliedsländer der EU.

```
SELECT L.LName, M.*
   FROM Land L JOIN Mitglied M
   ON L.LCode = M.LCode
   WHERE M.Organisation = 'EU'
```

... Mache den natürlichen Verbund in der Anfrage explizit.

```
SELECT L.LName, M.*
   FROM Land L NATURAL JOIN Mitglied M
   WHERE M.Organisation = 'EU'
```

Bestimme die Mitgliedschaften zu den einzelnen Ländern mit Fläche größer 500000.

```
SELECT L.LCode AS Land, M.Organisation AS Org
   FROM Land L LEFT OUTER JOIN Mitglied M
   ON L.LCode = M.LCode
   WHERE L.Fläche > 500
```

Sortierung

Sortiere die Zeilen der Tabelle `Stadt` aufsteigend nach `LCode` und für gemeinsame Werte zu LCode absteigend nach dem Breitengrad.

```
SELECT * FROM Stadt
   ORDER BY LCode ASC, BGrad DESC
```

3.3 Basis-Datentypen

SQL bietet eine Fülle von unterschiedlichen Datentypen an, mittels derer die Wertebereiche der Spalten einer Tabelle festgelegt werden können.

- `INTEGER`, `SMALLINT`
- `NUMERIC`, `DECIMAL`. Angabe Anzahl Ziffern insgesamt und Anzahl Kommastellen.
- `REAL`, `DOUBLE PRECISION`, `FLOAT`
- `CHARACTER`, `CHARACTER VARYING`, `CHARACTER LARGE OBJECT`
- `BIT`, `BIT VARYING`, `BINARY LARGE OBJECT`
- `BOOLEAN`
- `DATE`, `TIME`, `TIMESTAMP`, `INTERVALL`, ...

CAST und CASE

- `CAST` erlaubt eine *explizite* Typkonversionen zwischen unterschiedlichen Typen;
- `CASE` beschränkt die Konversionen im Wesentlichen auf Wertumwandlungen innerhalb eines Typs.

Erstelle eine Tabelle mit Spalten `LName` und `LCode`, die zu jedem Land den Namen auf die ersten zwei Zeichen reduziert und anstatt `'D'` den Wert `'BRD'` von `LCode` verwendet.

```
SELECT CAST(LName AS VARCHAR(2)),
   CASE LCode
      WHEN 'D' THEN 'BRD'
      ELSE LCode
   END
   FROM Land
```

3.4 Nullwerte

- SQL bietet die Prädikate `IS NULL` und `IS NOT NULL` an, um auf Existenz von Nullwerten prüfen zu können.
- In Ausdrücken der Form A+B, A+1, etc. ist das Resultat `null`, wenn einer der Operanden `null` ist.
- Ausdrücke mit Vergleichsoperatoren der Form A=B, A<>B, A<B, etc. haben den Wahrheitswert `UNKNOWN`, wenn mindestens einer der beteiligten Operanden den Wert `null` besitzt.
- SQL liegt eine dreiwertige Logik zugrunde. (`t=TRUE`, `f=FALSE`, `u=UNKNOWN`).

Bestimme alle Provinzen, zu denen die Fläche bekannt ist.

```
SELECT * FROM Provinz
   WHERE Fläche IS NOT NULL
```

Wahrheitswerte

AND	t	u	f
t	t	u	f
u	u	u	f
f	f	f	f

OR	t	u	f
t	t	t	t
u	t	u	u
f	t	u	f

NOT	
t	f
u	u
f	t

3.5 Anfragen mit Aggregierungsfunktionen

COUNT, MIN, MAX, SUM und AVG

Wieviele Länder gibt es in der Tabelle Land, wie groß ist die maximale, die minimale Fläche und die durchschnittliche Fläche aller Länder?

```
SELECT COUNT(LName),MAX(Fläche),
       MIN(Fläche),AVG(Fläche)
   FROM Land
```

Wieviele Länder haben eine Mitgliedschaft bzgl. der EU?

```
SELECT COUNT(*) AS AnzEU
   FROM Mitglied
   WHERE ......................
```

Wieviele unterschiedliche Organisationen werden in Mitglied aufgeführt?

```
SELECT COUNT(DISTINCT Organisation) FROM Mitglied
```

Besonderheiten

- `SELECT LName, MAX(Einwohner) FROM Land` ist syntaktisch nicht zulässig.
- Aggregierungsfunktionen ignorieren für ihre Berechnungen Nullwerte.
- Eine Ausnahme ist `COUNT(*)`; hier werden auch alle Zeilen, in denen alle Spalten `null` sind, mitgezählt.

3.6 Anfragen mit Gruppierungen

- Mittels einer *Gruppierung* können wir eine virtuelle Struktur über einer Tabelle definieren.
- Die Gruppierungsattribute fassen alle Zeilen der Tabelle jeweils zu einer Gruppe zusammen, die bezüglich aller Gruppierungsattribute gleiche Werte haben und zusätzlich die in einer optionalen `HAVING`-Klausel festgelegten Bedingungen erfüllt.
- Anfragen über einer gruppierten Tabelle betrachten die einzelnen Gruppen zusammen mit den Gruppierungsattributen analog zu einer Zeile einer Tabelle.

Wie groß ist die durchschnittliche Einwohnerzahl der Städte der jeweiligen Länder?

```
SELECT LCode, AVG(Einwohner) FROM Stadt
   GROUP BY LCode
```

In welchen Ländern ist die durchschnittliche Einwpohnerzahl kleiner 2 Mio.?

```
SELECT LCode, AVG(Einwohner) FROM Stadt
   GROUP BY LCode
   HAVING ......................
```

Bestimme die drei Städte mit den größten Einwohnerzahlen.

```
SELECT MAX(A.Einwohner) AS Rang1,
       MAX(B.Einwohner)AS Rang2,
       MAX(C.Einwohner) AS Rang3
   FROM Stadt A, Stadt B, Stadt C
   WHERE (A.Einwohner > B.Einwohner)
       AND (.............................)
```

Bestimme die drei Städte mit den größten Einwohnerzahlen.

```
SELECT DISTINCT COUNT(*) AS Rang, A.Einwohner
   FROM Stadt A, Stadt B
   WHERE (A.Einwohner <= B.Einwohner)
       GROUP BY A.Einwohner
       HAVING ..............
       ORDER BY Rang
```

SFW-Ausdruck

```
SELECT A_1, ..., A_n          Liste der Attribute
   FROM R_1, ..., R_m         Liste der Relationen
   WHERE F                    Bedingung
   GROUP BY B_1, ..., B_k     Liste der Gruppierungsattribute
   HAVING G                   Gruppierungsbedingung
   ORDER BY H                 Sortierordnung
```

Für die Auswertungsreihenfolge gilt: FROM-Klausel vor WHERE-Klausel vor GROUP-Klausel vor HAVING-Klausel vor ORDER-Klausel vor SELECT-Klausel.

3.7 Anfragen mit Mengenoperatoren

UNION, INTERSECT und EXCEPT.

- Die beteiligten Tabellen müssen zueinander *kompatible* Spaltentypen haben.
- Die Resultatspalte bekommt dann jeweils den allgemeineren Typ.

Welche Länder sind Teil von Europa und Asien?

```
SELECT LCode FROM Lage
   WHERE Kontinent = 'Europe'
INTERSECT
SELECT LCode FROM Lage
   WHERE Kontinent = 'Asia'
```

Welche Landcodes treten in der Relation Land oder der Relation Lage auf?

```
SELECT LCode, 'Stadt' AS Kategorie FROM Stadt
UNION
SELECT LCode, 'Lage' AS Kategorie FROM Lage
```

Welche Landcodes treten in der Relation Land und nicht in der Relation Lage auf?

```
SELECT LCode FROM Land
EXCEPT
SELECT LCode FROM Lage
```

Duplikate

- Duplikate werden berücksichtigt, sofern die Varianten `UNION ALL`, `INTERSECT ALL`, `EXCEPT ALL` verwendet werden. Anderenfalls wird standardmäßig `DISTINCT` angenommen.
- Im Falle einer Verwendung von `ALL` verhalten sich die Operatoren wie folgt. Hat der erste Operand n Duplikate einer Zeile und der zweite Operand m, wobei $0 \leq n, m$, dann hat das Ergebnis bei `UNION` $n + m$, bei `INTERSECT` $min(n, m)$, und bei `EXCEPT` $max(n - m, 0)$ Duplikate dieses Tupels.

3.8 Geschachtelte Anfragen

Eine Anfrage heißt *geschachtelt*, wenn sie in der `SELECT-`, `FROM-`, oder `WHERE-`, bzw. `HAVING`-Klausel selbst wieder eine SQL-Anfrage enthält.

Welche Länder befinden sich gemeinsam mit Russland auf einem Kontinent?

```
SELECT DISTINCT LCode FROM Lage
   WHERE Kontinent IN
      (.....................................................)
```

Welche Länder haben eine größere Fläche als mindestens ein anderes Land mit europäischem Anteil?

```
SELECT LName FROM Land
   WHERE Fläche > ANY
      (SELECT Fläche FROM Land, Lage
         WHERE Land.LCode = Lage.LCode AND
            ..............................)
```

Welche Länder haben eine größere Fläche als alle anderen Länder mit europäischem Anteil?

```
SELECT LName FROM Land L1
   WHERE Fläche > ALL
      (SELECT Fläche FROM Land L2, Lage L
         WHERE ........................ AND
            ..................... AND
            ........................)
```

Bei Verwendung von *Korrelationsvariablen* wird die Teilanfrage pro möglicher Wertekombination der Korrelationsvariablen ihrer übergeordneten Anfragen einmal ausgeführt.

Zu welchen Ländern ist genau ein Kontinent bekannt?

```
SELECT LName FROM Land L1
   WHERE UNIQUE
      (SELECT L2.LCode FROM Lage L2
         WHERE L1.LCode = L2.LCode)
```

Zu welchen Ländern ist genau ein Kontinent bekannt?

```
SELECT LName FROM Land L1
   WHERE 1 =
      (.............................
         ..........................)
```

Division!

Welche Länder sind Mitglied in denselben Organisationen wie Österreich?

```
SELECT DISTINCT LCode
   FROM Mitglied M
   WHERE NOT EXISTS
      ((SELECT DISTINCT Organisation FROM Mitglied
         WHERE LCode = 'A')
      EXCEPT
      (SELECT DISTINCT Organisation FROM Mitglied
         WHERE LCode = M.LCode))
```

Division alternativ.

Welche Länder sind Mitglied in denselben Organisationen wie Österreich?

```
SELECT DISTINCT LCode
   FROM Mitglied M1
   WHERE NOT EXISTS (
      SELECT * FROM Mitglied M2
         WHERE M2.LCode = 'A' AND NOT EXISTS (
            SELECT * FROM Mitglied M3
               WHERE M3.LCode = M1.LCode AND
                  ......................................))
```

Gleichheit!

Welche Organisationen haben alle europäischen Länder als Mitglied und nur gerade diese?

```
(SELECT DISTINCT Organisation FROM Mitglied M
   WHERE NOT EXISTS
      (SELECT LCode FROM Lage
         WHERE Lage.Kontinent = 'Europe') EXCEPT
      (SELECT LCode FROM Mitglied
         WHERE Organisation = M.Organisation))
AND NOT EXISTS
      (SELECT LCode FROM Mitglied
         WHERE Organisation = M.Organisation) EXCEPT
      (SELECT LCode FROM Lage
         WHERE Lage.Kontinent = 'Europe'))
```

3.9 Struktur der Syntax

Anfrageausdruck

Orthogonalität der Syntax: Ein Ausdruck, der eine Tabelle definiert, ist überall dort zulässig ist, wo eine Tabelle stehen darf.

- Ein Anfrageausdruck definiert eine Tabelle.
- Jeder Tabellenbezeichner ist ein Anfrageausdruck.
- Jeder SFW-Ausdruck ist ein Anfrageausdruck.
- Ein Verbundausdruck ist ebenfalls ein Anfrageausdruck.
- Die üblichen Mengenoperatoren können verwendet werden, um Anfrageausdrücke zu bilden.
- Ein *Tabellen-Konstruktor* der Form `VALUES ('a1',...,'an'), ('b1',...,'bn'), ...` ist ein Anfrageausdruck.

Liste die Namen, die für Städte und Länder verwendet werden.

```
SELECT Name
   FROM (SELECT SName AS Name
            FROM Stadt UNION
         SELECT LName AS Name
            FROM Land) T
```

Berechne die Anzahl der Menschen aller Länder, die in der größten Stadt ihres Landes leben.

```
SELECT SUM(GroßStädter)
   FROM (SELECT LCode, MAX(Einwohner) AS Großstädter
            FROM Stadt
            GROUP BY LCode) T
```

Lateral-Klausel.

Bestimme zu jedem Land die Summe der Einwohner aller Städte.

```
SELECT L.LCode, E.GesamtEinwohner
   FROM Land L, LATERAL (
      SELECT SUM(Einwohner) AS GesamtEinwohner
      FROM Stadt S
      WHERE S.LCode = L.LCode ) E
```

Skalarer Anfrageausdruck

Anstelle eines Wertes, bzw Spaltenbezeichners, ist auch ein geklammerter Tabellenausdruck zulässig, sofern er *skalar* ist, d.h. genau einen Wert definiert.

Bestimme zu jeder Stadt den Mittelwert der Einwohnerzahl aller Städte, die weniger Einwohner haben als sie selbst.

```
SELECT SName, Einwohner,
       (SELECT AVG(Einwohner) FROM Stadt S2
          WHERE S2.Einwohner <= .............)
             AS kleinerMittelwert
   FROM Stadt S1
```

Bestimme diejenigen asiatischen Länder, deren Flächenanteil in Asien kleiner ist als der Anteil der Türkei in Asien.

```
SELECT DISTINCT LCode, Prozent FROM Lage
   WHERE Kontinent = 'Asia' AND
      Prozent <
         (SELECT Prozent FROM Lage
            WHERE LCode = 'TR' AND Kontinent = 'Asia')
```

Bedingte Ausdrücke

- Ein Ausdruck, dem ein Wahrheitswert zugeordnet werden kann, ist ein bedingter Ausdruck.
- Bedingte Ausdrücke sind Teil einer `WHERE`-, `HAVING`- und `ON`-Klausel.
- Wesentlich für die korrekte Verwendung bedingter Ausdrücke ist die Miteinbeziehung des Auftretens von Nullwerten.

Welche Städte sind keine Hauptstadt von irgendeinem Land?

```
SELECT SName FROM Stadt S
   WHERE S.SName NOT IN (SELECT HStadt FROM Land)
```

3.10 Konstruierte Datentypen

Ein Datentyp heißt *konstruiert*, sofern seine Werte aus Werten anderen Typen, sogenannter *Elementtypen*, 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 Elementtypen 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 Elementtyps, 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.

Kollektionstypen `Multiset, Array`

- Zum Datentyp `MULTISET` werden eine Fülle von Operatoren angeboten, die die üblichen Mengenoperatoren wie Vereinigung (`MULTISET UNION`, oder auch Durchschnitt `MULTISET INTERSECT` enthalten.
  `MULTISET`s können auch auf Gleichheit getestet werden, wobei zwei `MULTISET`s genau dann gleich sind, wenn sie die gleichen Elemente in der gleichen Anzahl enthalten.
- Innerhalb eines SFW-Ausdrucks können Elemente vom Typ `ROW` im Prinzip überall dort stehen, wo ein skalarer Wert stehen kann.
  `ROW`-Elemente werden komponentenweise, analog zu Vektoren, verglichen.
  `ROW(1,2) <= ROW(2,1)` liefert den Wahrheitswert `FALSE`, jedoch `ROW(1,2) <= ROW(1,2)` den Wert `TRUE`.
- Der `UNNEST`-Operator wandelt ein Element eines Kollektionstyps in eine Tabelle um.

Erstelle eine Tabelle, in der jede einzelne Provinz zusammen mit ihrem Land eine Zeile bildet.

```
SELECT L.LName, P.Provinz
   FROM Land L, UNNEST(Provinzen) P (Provinz)
```

3.11 OLAP: ROLLUP und Cube

Unter *Online Analytical Processing* (*OLAP*) werden Anwendungen verstanden, die nicht auf den eigentlichen operativen Daten eines Unternehmens ablaufen, sondern auf integrierten, typischerweise historischen Daten eines Unternehmens.

- Anfragen dieser Art sind typisch für entscheidungsunterstützende Systeme (*Decision-Support*) und
- *Data-Warehouse*-Anwendungen.
- SQL:1999 führt zur Unterstützung solcher Anwendungen die Operatoren `ROLLUP` und `CUBE` ein.

Erstelle eine Liste in der pro Kontinent, Land und Provinz die Anzahl Städte berechnet wird.

```
SELECT C.Kontinent, L.LCode, P.PName, COUNT(*) AS Num
    FROM Lage C
    JOIN Land L ON C.LCode = L.LCode
    JOIN Provinz P ON L.LCode = P.LCode
    JOIN Stadt S ON P.PName = S.PName AND P.LCode = S.LCode
    GROUP BY C.Kontinent, L.LCode, P.PName
    ORDER BY C.Kontinent, L.LCode, P.PName
```

Erstelle eine Liste in der pro Kontinent, Land und Provinz die Anzahl Städte berechnet wird, einschließlich der Verdichtungen pro Land, Kontinent und alle Kontinente.

```
SELECT C.Kontinent, L.LCode, P.PName, COUNT(*) AS Num
   FROM Lage C
   JOIN Land L ON C.LCode = L.LCode
   JOIN Provinz P ON L.LCode = P.LCode
   JOIN Stadt S ON P.PName = S.PName AND P.LCode = S.LCode
   GROUP BY ROLLUP (C.Kontinent, L.LCode, P.PName)
   ORDER BY C.Kontinent, L.LCode, P.PName
```

NULLs mit unterschiedlicher Ursache müssen unterschieden werden können!

```
SELECT C.Kontinent, L.LCode, P.PName, COUNT(*) AS Num
   FROM Lage C LEFT OUTER JOIN Land L ON C.LCode = L.LCode
   LEFT OUTER JOIN Provinz P ON L.LCode = P.LCode
   LEFT OUTER JOIN Stadt S ON P.PName = S.PName AND
                P.LCode = S.LCode
   GROUP BY ROLLUP (C.Kontinent, L.LCode, P.PName)
   ORDER BY C.Kontinent, L.LCode, P.PName
```

GROUPING-Spalten.

```
SELECT C.Kontinent, L.LCode, P.PName, COUNT(*) AS Num,
   GROUPING(P.PName) AS ProvinzNullTest
   ...
```

Analysiere die Tabelle `Mitglied` indem alle möglichen Gruppierungen über `Organisation` und `Art` betrachtet werden.

```
SELECT Organisation AS Org, Art, COUNT(*) AS Num
   FROM Mitglied
   GROUP BY CUBE (Organisation, Art)
   ORDER BY Organisation
```

3.12 Rekursion

Unter einer *rekursiven* Anfrage verstehen wir eine Anfrage, die eine Tabelle T definiert, wobei in diese Definition T selbst wiederum eingeht.

Tabellen

symBenachbart

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

Benachbart

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

Erreichbar

Von	Nach	Übergänge
CH	D	1
CH	F	1
CH	I	1
D	CH	1
D	F	1
D	I	2
F	CH	1
F	D	1
F	I	1
I	CH	1
I	D	2
I	CH	1

Gib zu jedem Land alle Länder an, die von diesem aus auf dem Landweg erreichbar sind, einschließlich der minimalen Anzahl der benötigten Grenzübergänge.

```
WITH RECURSIVE
Erreichbar(Von, Nach, Übergänge) AS (
   SELECT LCode1 AS Von, LCode2 AS Nach, 1
      FROM symBenachbart
   UNION ALL
   SELECT A.Von, B.LCode2 AS Nach, A.Übergänge+1
      FROM Erreichbar A, symBenachbart B
         WHERE A.Nach = B.LCode1 )

SELECT Von, Nach, MIN(Übergänge) AS Übergänge
   FROM Erreichbar WHERE Von <> Nach
   GROUP BY Von, Nach
```

Auswertung terminiert im Allgemeinen nicht!

Anfrage mit korrekter Abbruchbedingung.

```
WITH RECURSIVE
Anzahl (N) AS (
SELECT COUNT(DISTINCT LCode1) FROM symBenachbart ),
Erreichbar(Von, Nach, Übergänge) AS (
SELECT LCode1 AS Von, LCode2 AS Nach, 1
FROM symBenachbart
    UNION ALL
SELECT A.Von, B.LCode2 AS Nach, A.Übergänge + 1
FROM Erreichbar A, symBenachbart B
WHERE A.Nach = B.LCode1 AND A.Von <> B.LCode2 AND
   A.Übergänge < (SELECT N FROM Anzahl) )

SELECT Von, Nach, MIN(Übergänge) AS Übergänge
   FROM Erreichbar WHERE Von <> Nach
   GROUP BY Von, Nach
```

Teilschritte der Auswertung

- Zunächst initialisieren wir die zu berechnende temporäre Tabelle `Erreichbar` durch die Tabelle `symBenachbart`.
- Die Semantik der rekursiven Teilanfrage ergibt sich dann durch so lange fortgesetztes induktives Auswerten der rekursiven Teilanfrage, wie neue Zeilen (inklusive Duplikaten) für `Erreichbar` ermittelt werden können, die die Bedingungen der `WHERE`-Klauseln der rekursive Teilanfrage erfüllen.
- In einem letzten Teilschritt kann dann das gewünschte Ergebnis der minimalen Anzahl benötigter Grenzübergänge aus der induktiv berechneten Tabelle mittels der primären (nachgeschalteten) Anfrage extrahiert werden.

alternative Formulierung

```
WITH RECURSIVE
Anzahl (N) AS (
SELECT COUNT(*) FROM Benachbart ),
Erreichbar(Von, Nach, Übergänge, Symm) AS (
SELECT LCode1 AS Von, LCode2 AS Nach, 1, 0
FROM Benachbart
    UNION ALL
SELECT Nach AS Von, Von AS Nach, Übergänge, 1
FROM Erreichbar WHERE Symm < 1
    UNION ALL
SELECT Von, LCode2 AS Nach, Übergänge+1, 0
FROM Erreichbar, Benachbart
WHERE Nach = LCode1 AND Von <> LCode2 AND
   Übergänge < (SELECT N FROM Anzahl)
    UNION ALL
SELECT Von, LCode1 AS Nach, Übergänge+1, 0
FROM Erreichbar, Benachbart
WHERE Nach = LCode2 AND Von <> LCode1 AND
   Übergänge < (SELECT N FROM Anzahl))
```

Bemerkungen

- Für die rekursiven Teilanfragen gelten eine Reihe von syntaktischen Restriktionen. Beispielsweise sind in den rekursiven Teilanfragen keine Aggregierungsfunktionen zulässig.
- Darüber hinaus darf die rekursiv definierte Tabelle wohl in der FROM-Klausel einer rekursiven Teilanfrage auftreten, aber nicht in einer Teilanfrage von dieser. Nicht erlaubt:

```
WHERE A.Nach = B.LCode1 AND A.Von <> B.LCode2
   AND NOT EXISTS (
      SELECT * FROM Erreichbar C
      WHERE A.Von = C.Von AND B.LCode2 = C.Nach
         AND A.Übergänge >= (C.Übergänge - 1) )
```

weitere Klauseln

- Der SQL-Standard sieht eine spezielle Klausel `CYCLE` vor, mittels derer eine systemkontrollierte Terminierung erreicht werden kann.
- Mittels der `SEARCH`-Klausel kann die Sortierfolge der Zeilen der rekursiven Tabelle in Analogie zu einer Tiefen- und Breitensuche eines Graphen gesteuert werden.
- Gemäß dem Standard können in einer `WITH RECURSIVE`-Klausel mehrere Tabellen rekursive definiert werden.

3.13 Anfragen über Bäumen

Baum

- Hierarchisch angeordnete Menge von Objekten, die wir als *Knoten* bezeichnen.
- Ein Knoten dieser Menge wird als die *Wurzel* ausgezeichnet.
- Zu jedem Knoten p außer der Wurzel, existiert genau ein (unmittelbarer) Vorgängerknoten q. Dieser Knoten q wird als *Elterknoten* von p bezeichnet, bzw. p als ein *Kindknoten* von q. Ein Knoten ohne Kindknoten heißt *Blatt* und zwei Knoten mit demselben Elterknoten nennen wir *Geschwisterknoten*.
- Eine Folge von Knoten $p_1, p_2, \ldots, p_k$, wobei p_i Elterknoten von p_{i+1}, $1 \leq i < k$, heißt *Pfad* der Länge k von p_1 nach p_k.
- Zu jedem von der Wurzel verschiedenen Knoten existiert genau ein Pfad ausgehend von der Wurzel; die Länge dieses Pfades ist die *Tiefe* des Knotens im Baum; die Länge des längsten Pfades von der Wurzel zu einem Blatt ergibt die *Höhe* des Baumes.

Tabelle

Gliederung

Untereinheit	Bezeichnung	Obereinheit
Deutschland	Staat	Mondial
Baden-Württemberg	Land	Deutschland
RegBez.Freiburg	Bezirk	Baden-Württemberg
Br.-Hochschwarzwald	Kreis	RegBez.Freiburg
St.Peter	Gemeinde	Br.-Hochschwarzwald
France	Staat	Mondial
Alsace	Region	France
Bas-Rhin	Departement	Alsace
Sélestat-Erstein	Bezirk	Bas-Rhin
Sélestat	Kanton	Sélestat-Erstein
Ebersmunster	Gemeinde	Sélestat
Schweiz	Staat	Mondial
Graubünden	Kanton	Schweiz
Albula	Bezirk	Graubünden
Schanfigg	Kreis	Albula
Arosa	Gemeinde	Schanfigg

Bestimme die Wurzel des Baumes.

```
SELECT DISTINCT Obereinheit AS Wurzel
   FROM Gliederung G1
   WHERE 0 =
      ( SELECT COUNT(*) FROM Gliederung G2
         WHERE G1.Obereinheit = G2.Untereinheit )
```

Bestimme alle Blätter des Baumes.

```
SELECT Untereinheit AS Blatt
   FROM Gliederung G1
   WHERE 0 =
      ( SELECT COUNT(*) FROM Gliederung G2
         WHERE G1.Untereinheit = G2.Obereinheit )
```

Bestimme zu den einzelnen Staaten die maximale Tiefe der Gliederung der Verwaltungseinheiten.

```
WITH RECURSIVE
Ebene(Staat, Knoten, N) AS (
SELECT Untereinheit AS Staat,
Untereinheit AS Knoten, 0
FROM Gliederung
WHERE Obereinheit = 'Mondial'
    UNION ALL
SELECT Staat, Untereinheit AS Knoten, N+1
FROM Ebene, Gliederung
WHERE Knoten = Obereinheit )

SELECT Staat, MAX(N)
FROM Ebene GROUP BY Staat
```

Abbruchkriterium?

Bestimme zu jeder Gemeinde die übergeordneten Verwaltungseinheiten.

```
WITH RECURSIVE
Gemeinde(Name, Obereinheit, N) AS (
SELECT Untereinheit AS Name, Obereinheit, 0
FROM Gliederung
WHERE Bezeichnung = 'Gemeinde'
    UNION ALL
SELECT G1.Name, G2.Obereinheit, G1.N+1
FROM Gemeinde G1, Gliederung G2
WHERE G1.Obereinheit = G2.Untereinheit )
SELECT * FROM Gemeinde
ORDER BY Name, N
```

Abbruchkriterium?

3.14 Einfügen, Löschen und Ändern

Einfügen

- Mittels `INSERT` kann eine neue Zeile, oder eine Menge von neuen Zeilen in eine Tabelle 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
   SELECT DISTINCT M.LCode
      FROM Mitglied M
      WHERE NOT EXISTS (
         SELECT L.LCode
            FROM Land L
            WHERE .....................)
```

Sequenznummern

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

Beispiel zur Verwendung einer Sequenznummer innerhalb der Tabelle Land.

```
CREATE SEQUENCE LandSEQ AS INTEGER
   START WITH 1
   INCREMENT BY 1
   MINVALUE 1
   MAXVALUE 100000
   NO CYCLE

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

Beispiel für eine Generierte Spalten; der zughörige Wert wird automatisch berechnet.

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

Löschen

- `DELETE FROM T WHERE P`
- Während des Löschens von einzelnen Zeilen einer Tabelle kann sich der betreffende Wert der `WHERE`-Klausel ändern. In SQL-Implementierungen werden deshalb vor dem Löschen alle zu löschenden Tupel markiert und erst danach gelöscht.

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
```

3.15 Sichten

- Eine Sicht `V` ist eine durch einen Anfrageausdruck `E` definierte Tabelle:
- ```
  CREATE VIEW V AS
      <E>
  ```
- Im Unterschied zu denen 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.

```
CREATE VIEW symGrenze AS
   SELECT LCode1 AS Von, LCode2 AS Nach
   FROM Benachbart
      UNION
   SELECT B1.LCode2 AS Von, B1.LCode1 AS Nach
   FROM Benachbart B1
      WHERE NOT EXISTS (
      SELECT * FROM Benachbart B2
      WHERE B1.LCode2 = B2.LCode1 AND
          B1.LCode1 = B2.LCode2)
```

Welche Länder sind zu Deutschland benachbart?

```
SELECT Nach FROM symGrenze
   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 Einfügen/Löschen/Ändern in einer Sicht.

$r =$

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

Konsequenzen des Einfügens von (a, b, d) in v?

Projektionssicht

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)
```

Selektionssicht

- Aufgrund von 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 einer einzelnen Zeile in einer Basistabelle.

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')
```

Es existiert keine Transformation zu der Einfügeoperation.

3.16 SQL und Programmiersprachen

- SQL hat somit eine tabellenorientierte Sematik.
- SQL hat eine im Vergleich zu einer Programmiersprache eingeschränkte Mächtigkeit, so dass 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

- Erweiterung von SQL um imperative Sprachelemente zur Formulierung von in der Datenbank gespeicherten benutzerdefinierten Funktionen und Prozeduren (*SQL-Erweiterung*).
- Einbettung von SQL-Ausdrücken in eine Programmiersprache (*statisches SQL*).
- Übergabe eines SQL-Ausdrucks an eine Datenbank während der Ausführung eines Programms (*dynamisches SQL*).

SQL-Erweiterung

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

Berechne zu der Tabelle `Benachbart` den symmetrischen Abschluss.

```
CREATE FUNCTION symBenachbart()
RETURNS TABLE (
   LCode1 VARCHAR(30),
   LCode2 VARCHAR(30) )
RETURN (SELECT * FROM Benachbart
   UNION
   SELECT LCode2 AS LCode1, LCode1 AS LCode2
   FROM Benachbart )
```

Gib zu jedem Land alle Länder an, die von diesem aus auf dem Landweg erreichbar sind, einschließlich der minimalen Anzahl der benötigten Grenzübergänge.

```
CREATE FUNCTION Erreichbar()
RETURNS TABLE ( Von, Nach VARCHAR(30), Übergänge INTEGER )
BEGIN
CREATE TABLE Erreichbar (
      Von, Nach VARCHAR(30),
      Übergänge INTEGER NOT NULL );
DECLARE alt, neu INTEGER;
SET alt = 0;
INSERT INTO Erreichbar
      SELECT T.LCode1 AS Von, T.LCode2 AS Nach, 1 AS Übergänge
      FROM TABLE (symBenachbart()) T;
SET neu = (SELECT COUNT(*) FROM Erreichbar);
WHILE (alt <> neu) DO
      SET alt = neu;
      INSERT INTO Erreichbar
      SELECT DISTINCT A.Von, B.LCode2, (A.Übergänge + 1)
      FROM Erreichbar A, TABLE (symBenachbart()) B
      WHERE A.Nach = B.LCode1 AND A.Von <> B.LCode2
            AND NOT EXISTS (
                  SELECT * FROM Erreichbar C
                  WHERE A.Von = C.Von AND B.LCode2 = C.Nach
                        AND A.Übergänge > (C.Übergänge - 1) );
      SET neu = (SELECT COUNT(*) FROM Erreichbar)
END WHILE;
RETURN Erreichbar;
END
```

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;
```

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; }
      ...
?>
```

3.17 Integrität und Trigger

- Im Allgemeinen sind nur solche Instanzen einer Datenbank erlaubt, deren Relationen die der Datenbank bekannten *Integritätsbedingungen* (IB) erfüllen.
- Integritätsbedingungen können *explizit* durch den Benutzer definiert werden, durch die Definition von konkreten Schemata *implizit* erzwungen werden, oder bereits dem relationalen Datenmodell *inhärent* sein.
- *Inhärente Bedingungen:* Attributwerte sind skalar; Relationen, abgesehen von Duplikaten, verhalten sich wie Mengen, d.h. ohne weitere Angaben haben sie insbesondere keine Sortierung.
- *Implizite Bedingungen:* Werte der Attribute eines Primärschlüssels dürfen keine Nullwerte enthalten
- *Explizite Bedingungen:* Werden als Teil der `CREATE TABLE`-Klausel, bzw. der `CREATE SCHEMA`-Klausel definiert.

- Integritätsbedingungen sind von ihrer Natur aus deklarativ: sie definieren die zulässigen Instanzen, ohne auszudrücken, wie eine Gewährleistung der Integrität implementiert werden kann.
- Komplementär hierzu bieten Datenbanksysteme einen *Trigger*-Mechanismus an, mit dem in Form von Regeln definiert werden kann, welche Aktionen zur Gewährleistung der Integrität vorgenommen werden sollen, bzw., wie Verletzungen behandelt werden sollen.
- Mittels Trigger können wir insbesondere die Zulässigkeit von *Zustandsübergängen* kontrollieren, was mit Integritätsbedingungen aufgrund ihres Bezugs zu gerade einem Zustand nicht möglich ist.

3.17.1 Fremdschlüsselbedingungen

- *Fremdschlüsselbedingungen* werden als Teil der `CREATE TABLE`-Klausel definiert.
- Sie sind formal sogenannte *Inklusionsabhängigkeiten*: zu jedem von `null` verschiedenen Fremdschlüsselwert in einer Zeile der *referenzierenden* Tabelle, der C- (child-) Tabelle, existiert ein entsprechender Schlüsselwert in einer Zeile der *referenzierten* Tabelle, der P- (parent-) Tabelle.
- Man redet hier auch von *referentieller* Integrität. Zur Definition von Fremdschlüsselbedingungen steht die `FOREIGN KEY`-Klausel zur Verwendung in der C-Tabelle zur Verfügung.

Die Spalte LCode innerhalb der Tabelle Provinz enthält Werte des Schlüssels LCode der Tabelle Land. Zu jeder Zeile in Provinz muss eine Zeile in Land existieren, deren Schlüsselwert gleich dem Fremdschlüsselwert ist.

```
CREATE TABLE Provinz (
   PName VARCHAR(35),
   LCode VARCHAR(4),
   FlächeNUMBER
   PRIMARY KEY (PName, LCode),
   FOREIGN KEY (LCode) REFERENCES Land (LCode) )
```

Die Zeilen der Tabelle `Grenze` enthalten jeweils zwei unterschiedliche Fremdschlüssel, die beide Werte des Schlüssels `LCode` der Tabelle `Land` annehmen.

```
CREATE TABLE Grenze (
   LCode1VARCHAR(4),
   LCode2VARCHAR(4),
   Länge INTEGER,
   PRIMARY KEY (LCode1, LCode2),
   FOREIGN KEY (LCode1) REFERENCES Land (LCode),
   FOREIGN KEY (LCode2) REFERENCES Land (LCode) )
```

Für die Tabelle `Stadt` sind zwei Fremdschlüsselbeziehungen relevant. Einmal müssen die referenzierten Länder in der Tabelle zu `Land` existieren, und zum andern entsprechend die Provinzen. Letzterer Fremdschlüssel besteht aus zwei Spalten. Die Zuordnung der einzelnen Spalten des Fremd- und Primärschlüssels ergeben sich aus der Reihenfolge des Hinschreibens.

```
CREATE TABLE Stadt (
   :
   :
   PRIMARY KEY (SName, LCode, PName),
   FOREIGN KEY (LCode) REFERENCES Land (LCode),
   FOREIGN KEY (LCode, PName)
      REFERENCES Provinz (LCode, PName) )
```

referentielle Aktionen im Überblick

Zur Gewährleistung der referentiellen Integrität werden sogenannte *referentielle Aktionen* zur Ausführung bezüglich der C-Tabellen definiert. Aufgabe dieser Aktionen ist die Kompensierung von durch `DELETE`- und `UPDATE`-Operationen auf der zugehörigen P-Tabelle verursachten Verletzungen der Integrität.

- Änderungen der P-Tabelle werden auf die C-Tabelle übertragen (Aktion `CASCADE`).
- Die Änderung der P-Tabelle wird im Falle einer Verletzung der referentiellen Integrität einer C-Tabelle abgebrochen (Aktion `NO ACTION` und Aktion `RESTRICT`).
- Der Fremdschlüsselwert der C-Tabelle wird angepaßt (Aktion `SET NULL` und Aktion `SET DEFAULT`).

Wird der Code eines Landes geändert oder das Land gelöscht, so sollen die neuen Codes bei den zugehörigen Provinzen nachgezogen werden, bzw. auch die Provinzen des gelöschten Landes gelöscht werden.

```
CREATE TABLE Provinz (
   ⋮
   FOREIGN KEY (LCode) REFERENCES Land (LCode)
      ON DELETE CASCADE ON UPDATE CASCADE
```

Werden Provinzen gelöscht, so sollen ihre Städte weiter in der Datenbank bestehen bleiben, wobei der betreffende Fremdschlüsselwert Nullwerte erhält. Änderungen eines Provinzschlüssels sollen auf die betroffenen Städte übertragen werden.

```
CREATE TABLE Stadt (
   ⋮
   FOREIGN KEY (LCode, PName)
      REFERENCES Provinz (LCode, PName)
      ON DELETE SET NULL ON UPDATE CASCADE))
```

Bemerkung

- Einfügen bezüglich der P-Tabelle oder Löschen bezüglich der C-Tabelle ist für die referentielle Integrität immer unkritisch.
- Einfügen bezüglich der C-Tabelle oder Ändern bezüglich der C-Tabelle, die einen Fremdschlüsselwert erzeugen, zu dem kein Schlüssel in der P-Tabelle existiert, sind immer primär unzulässig, da von Änderungen in den C-Tabellen im Allgemeinen kein sinnvoller Rückschluss auf Änderungen der P-Tabellen möglich ist; anderenfalls sind die Änderungen unkritisch.

referentielle Aktionen

NO ACTION: Die Operation auf der P-Tabelle wird zunächst ausgeführt; ob Dangling References in der C-Tabelle entstanden sind wird erst nach Abarbeitung aller durch die Operation auf der P-Tabelle direkt oder indirekt ausgelösten referentiellen Aktionen überprüft.

RESTRICT: Die Operation auf der P-Tabelle wird nur dann ausgeführt, wenn durch ihre Anwendung keine Dangling References in der C-Tabelle entstehen.

CASCADE: Die Operation auf der P-Tabelle wird ausgeführt. Erzeugt die DELETE/UPDATE-Operation Dangling References in der C-Tabelle, so werden die entsprechenden Zeilen der C-Tabelle ebenfalls mittels DELETE entfernt, bzw. mittels UPDATE geändert. Ist die C-Tabelle selbst P-Tabelle bezüglich einer anderen Bedingung, so wird das DELETE/UPDATE bezüglich der dort festgelegten Lösch/Änderungs-Regel weiter behandelt.

SET DEFAULT: Die Operation auf der P-Tabelle wird ausgeführt. In der C-Tabelle wird der entsprechende Fremdschlüsselwert durch die für die betroffenen Spalten in der C-Tabelle festgelegten DEFAULT-Werte ersetzt; es muss jedoch gewährleistet sein, daß entsprechende Schlüsselwerte in den P-Tabellen existieren.

SET NULL: Die Operation auf der P-Tabelle wird ausgeführt. In der C-Tabelle wird der entsprechende Fremdschlüsselwert spaltenweise durch NULL ersetzt. Vorausetzung ist hier, daß Nullwerte zulässig sind.

Bei Verwendung von RESTRICT können in Abhängigkeit von der Reihenfolge der Abarbeitung der FOREIGN KEY-Klauseln in Abhängigkeit vom Inhalt der Tabellen potentiell unterschiedliche Ergebnisse resultieren.

```
CREATE TABLE T1 (... PRIMARY KEY K1)

CREATE TABLE T2 ( ... PRIMARY KEY K2
   FOREIGN KEY (K1) REFERENCES T1 (K1)
      ON DELETE CASCADE)

CREATE TABLE T3 (... PRIMARY KEY K3
   FOREIGN KEY (K1) REFERENCES T1 (K1)
      ON DELETE CASCADE)

CREATE TABLE T4 (... PRIMARY KEY K4
   FOREIGN KEY (K2) REFERENCES T2 (K2)
      ON DELETE CASCADE
   FOREIGN KEY (K3) REFERENCES T3 (K3)
      ON DELETE RESTRICT)
```

zum potentiellen Nichtdeterminismus

- Um nichtdeterministische Ausführungen dieser Art auszuschließen, wird vorgeschlagen, die Implementierung nach einer Strategie vorzunehmen, in der im Wesentlichen vor Berücksichtigung einer `RESTRICT`-Aktion alle `CASCADE`-Aktionen ausgeführt werden.
- Diese Strategie klärt offensichtlich obige Unbestimmtheit.
- Alternativ können gewisse Kombinationen von referentiellen Aktionen verboten werden. Ersetzt man `RESTRICT` durch `NO ACTION` in obigem Beispiel, so wird das Endergebnis wieder eindeutig, unabhängig von der Reihenfolge der betrachteteten referentiellen Aktionen.

3.17.2 Bedingungen über Werte

Zu jeder Tabelle werden typischerweise ein *Primärschlüssel* und möglicherweise weitere Schlüssel festgelegt (`UNIQUE`-Klausel).

In jeder Instanz zu der Tabelle `Land` können alle Zeilen eindeutig durch ihren Spaltenwert zu `LCode`, oder alternativ, durch ihren Spaltenwert zu `LName` identifiziert werden.

```
CREATE TABLE Land (
   LName        VARCHAR(35) UNIQUE,
   LCode        VARCHAR(4) PRIMARY KEY,
   ...
```

Identifizierendes Kriterium für die Tabelle `Stadt` sei `SName`, `LCode`, `PName`, bzw. alternativ, `LGrad`, `BGrad`.

```
CREATE TABLE Stadt (
   ...
   PRIMARY KEY (SName,LCode,PName),
   UNIQUE (LGrad,BGrad))
```

NONNUll, DEFAULT und CREATE DOMAIN

```
    LName VARCHAR(35) NONNULL
    Prozent NUMBER DEFAULT 100
```

```
CREATE DOMAIN meineNamen VARCHAR(35) DEFAULT '?'

CREATE TABLE Land (
    LName meineNamen,
    ⋮
```

3.17.3 Statische Integrität

CHECK-Klausel

- *Statische* Integrität definiert unter Verwendung der CHECK-Klausel, welche Instanzen eines Schemas zulässig sind.
- Mittels der CHECK-Klausel können die zulässigen Werte eines Datentyps und die für eine Spalte einer konkreten Tabelle zu verwendenden Werte weiter eingeschränkt werden.
- Darüberhinaus können beliebige, mittels SQL-Anfrageausdrücken gebildete, Bedingungen über den Instanzen der Tabellen eines Schemas ausgedrückt werden.

Wertebereichsbedingung

```
CREATE DOMAIN meineStädte CHAR(15),
   DEFAULT 'Paris',
   CHECK (VALUE IN ('Berlin', 'Paris',
                    'London', 'Rom'))
```

Spaltenbedingungen

```
CREATE TABLE Stadt (
   SName        meineStädte,
   ⋮
   LGrad        NUMBER
                CHECK (LGrad BETWEEN -180 AND 180),
   BGrad        NUMBER
                CHECK (BGrad BETWEEN -90 AND 90),
   ...)
```

Spalten- und Tabellenbedingung: Die Summe aller Anteile an unterschiedlichen Kontinenten eines Landes muss 100 ergeben.

```
CREATE TABLE Lage (
   LCode        VARCHAR(4)
   Kontinent    VARCHAR(35)
   Prozent      NUMBER
                CHECK (Prozent BETWEEN 0 AND 100),
   CHECK (100 = (SELECT SUM(Prozent) FROM Lage L
      WHERE LCode = L.LCode)))
```

Assertion

- Spalten- und Tabellenbedingungen sind erfüllt, wenn jede Zeile der betreffenden Tabelle sie erfüllt.
- Spalten- und Tabellenbedingungen sind somit implizit $\forall$-quantifiziert über den Zeilen der Tabelle.
- Alternativ können wir die explizitere Form einer `ASSERTION` wählen

Die Summe aller Anteile an unterschiedlichen Kontinenten eines Landes muss 100 ergeben.

```
CREATE ASSERTION AssertLage (
   CHECK ((SELECT COUNT(*) FROM Lage
      GROUP BY LCode
      HAVING (.....................) = 0))
```

Tabellenbedingung: Die Hauptstadt eines Landes muss mehr Einwohner als der Durchschnitt aller Städte dieses Landes haben.

```
CREATE TABLE Land (
   ...
   CHECK ((SELECT S.Einwohner FROM Stadt S
         WHERE S.SName = Land.HStadt AND S.LCode = Land.LCode) >
         (SELECT AVG(Einwohner) FROM Stadt T
            WHERE T.LCode = Land.LCode))
```

Assertion: Die Hauptstadt eines Landes muss mehr Einwohner als der Durchschnitt aller Städte dieses Landes haben.

```
CREATE ASSERTION AssertLand (
   CHECK (NOT EXISTS (
      (SELECT S.Einwohner FROM Stadt S, Land L
      WHERE S.SName = L.HStadt AND
         S.LCode = L.LCode AND
         S.Einwohner <=
            (SELECT AVG(Einwohner) FROM Stadt T
               WHERE T.LCode = L.LCode)))
```

Überprüfen von Tabellenbedingungen

- SQL überprüft nur dann eine Tabellenbedingung, wenn die betreffende Tabelle eine nicht-leere Instanz hat.
- Solange eine Tabelle `T` keine Zeilen enthält, ist somit jede beliebige ihrer Tabellenbedingungen erfüllt, sowohl beispielsweise widersprüchliche, als auch Bedingungen der folgenden Form.

Ist die Instanz zu `T` leer, so ist die Bedingung verletzt; diese Verletzung bleibt jedoch unerkannt, da die Bedingung nicht überprüft wird.

```
CHECK ((SELECT COUNT(*) FROM T) > 0)
```

3.17.4 Dynamische Integrität und Trigger

- *Dynamische* Integrität beschäftigt sich mit der Formulierung von Integritätsbedingungen, die definieren, welche Zustandsübergänge auf den Tabellen zu einem Datenbank-Schema erlaubt sind.
- Sie müssen es uns dazu ermöglichen, in einem Ausdruck sowohl den alten, wie auch den neuen Zustand der Instanzen ansprechen zu können.
- Zur Gewährleistung der dynamischen Integrität bietet SQL einen mächtigen *Trigger*-Mechanismus. Trigger sind ein Spezialfall *aktiver Regeln*, in denen in Abhängigkeit von eingetretenen Ereignissen, sofern gewisse Bedingungen erfüllt sind, definierte Aktionen auf einer Datenbank ausgeführt werden (**E**vent**C**ondition**A**ction-Paradigma).
- Innerhalb SQL sind die auslösenden Operationen gerade Einfügungen, Löschungen und Änderungen von Zeilen der Tabellen.

weitere Anwendungen

- Prüfen der Zulässigkeit von Werten vor der Durchführung von Änderungen, um so im Falle von Integritätsverletzungen diese korrigieren zu können.
- Protokollieren von auf sicherheitskritischen Tabellen vorgenommene Änderungen, z.B. mit Angabe der Benutzeridentifikation und Zugriffszeit.
- Implementierung von Änderungsoperationen auf Sichten.
- Definition von für Anwendungen verbindlichen (Geschäfts-)Regeln.

Ändert sich die Einwohnerzahl einer Stadt, dann soll die Einwohnerzahl der betreffenden Provinz angepaßt werden.

```
CREATE TRIGGER EinwohnerzahlAnpassen
   AFTER UPDATE OF Einwohner ON Stadt
   REFERENCING OLD AS Alt NEW AS Neu
   FOR EACH ROW
   UPDATE Provinz
      SET Einwohner=Einwohner-Alt.Einwohner+
         Neu.Einwohner
      WHERE LCode=Alt.LCode AND ................
```

Die Tabelle Grenze soll antisymmetrisch sein; d.h., für je zwei Länder darf eine Nachbarschaftsbeziehung nur einmal enthalten sein.

```
CREATE TRIGGER antiSymGrenze
   BEFORE INSERT ON Grenze
   REFERENCING NEW AS Neu
   FOR EACH ROW
   WHEN EXISTS ( SELECT * FROM Grenze G
      WHERE G.LCode1=Neu.LCode2 AND
         G.LCode2=Neu.LCode1 )
   BEGIN
      SIGNAL SQLSTATE '75001';
      SET Message='Grenze bereits vorhanden'
   END
```

Ergibt die Summe der Anteile an den Kontinenten für ein Land einen kleineren Wert als 100, so wird die Differenz dem Kontinent Atlantis zugeordnet.

```
CREATE TRIGGER Atlantis
   AFTER INSERT ON Lage
   FOR EACH STATEMENT
   WHEN EXISTS ( SELECT * FROM Lage
      GROUP BY LCode
      HAVING (SUM(Prozent) < 100) )
   BEGIN
      INSERT INTO Lage
      SELECT L1.LCode, 'Atlantis', (
         100 - ( SELECT SUM(L2.Prozent)
            FROM Lage L2
            WHERE L2.LCode = L1.LCode) )
      FROM Lage L1
      GROUP BY L1.LCode
      HAVING (.....)
   END
```

Trigger

- Ein Trigger ist einer Tabelle zugeordnet. Er wird aktiviert durch das Eintreten eines Ereignisses (SQL-Anweisung): Einfügung, Änderung und Löschung von Zeilen.
- Der Zeitpunkt der Aktivierung ist entweder vor oder nach der eigentlichen Ausführung der entsprechenden aktivierenden Anweisung in der Datenbank. Ein Trigger kann die Ausführung der ihn aktivierenden Anweisung verhindern.
- Ein Trigger kann einmal pro aktivierender Anweisung (Statement-Trigger) oder einmal für jede betroffene Zeile (Row-Trigger) seiner Tabelle ausgeführt werden.
- Mittels Transitions-Variablen `OLD` und `NEW` kann auf die Zeilen- und Tabellen-Inhalte vor und nach der Ausführung der aktivierenden Aktion zugegriffen werden. Im Falle von Tabellen-Inhalten handelt es sich dabei um hypothetische Tabellen, die alle betroffenen Zeilen enthalten.
- Ein aktivierter Trigger wird ausgeführt, wenn seine Bedingung erfüllt ist.
- Der Rumpf eines Triggers enthält die auszuführenden SQL-Anweisungen.
- Bei einem `BEFORE`-Trigger sind die einzufügenden Tupel nicht sichtbar in der Tabelle; es kann jedoch zu ihnen mittels `NEW` oder `NEW TABLE` zugegriffen werden. Bei einem `AFTER`-Trigger sind sie zusätzlich in der Tabelle zugreifbar.
- Bei einem `BEFORE`-Trigger sind die zu löschenden Tupel sichtbar in der Tabelle, bei einem `AFTER`-Trigger nicht; es kann zu ihnen mittels `OLD` oder `OLD TABLE` zugegriffen werden.
- Bei einem `BEFORE`- oder `AFTER`-Trigger kann zu den alten und neuen Werten der zu ändernden Tupel mittels `OLD/NEW` oder `OLD TABLE/NEW TABLE` zugegriffen werden. Bei einem `BEFORE`-Trigger sind die Änderungen nicht sichtbar in der Tabelle, bei einem `AFTER`-Trigger jedoch.

Bemerkungen

- Trigger können selbst weitere Trigger aktivieren, wenn ein ausgelöster Trigger eine Tabelle modifiziert, über der selbst Trigger definiert sind. Eine Transaktion kann somit während ihrer Ausführung eine ganze Reihe von Triggern auslösen.
- Die Reihenfolge der Ausführung dieser Trigger ist ohne weitere Kontrolle nicht vorhersehbar.
- Um eine deterministische Ausführung zu gewährleisten, sind Einschränkungen an die möglichen Triggerdefinitionen, bzw. Anforderungen an ihre Ausführung zu berücksichtigen.
- Eine Aktivierungsfolge von Triggern kann insbesonder zyklisch sein (*rekursive* Trigger); die Terminierung einer solchen Folge ist im Allgemeinen nicht gesichert.
- Im SQL-Standard nicht berücksichtigt werden `INSTEAD OF`-Trigger, die in einigen Datenbanksystemen implementiert sind.
- Wird ein solcher Trigger aktiviert, dann werden *anstelle* der auslösenden Operation die Operationen des Triggers ausgeführt.
- Ein sinnvolles Anwendungsgebiet für `INSTEAD OF`-Trigger ist die Realisierung von Änderungsoperationen auf Sichten auf den zugehörenden Basistabellen.

3.18 Zugriffskontrolle

- Datenbanken enthalten häufig vertrauliche Informationen, die nicht jedem Anwender zur Verfügung stehen dürfen.
- Außerdem wird man nicht allen Anwendern dieselben Möglichkeiten zur Verarbeitung der Daten einräumen wollen, da Änderungen der Daten unter Umständen kritisch sind, auch wenn die Daten an sich nicht vertraulich sind.
- Zugriffsrechte können nicht nur einzelnen Benutzern zugewiesen werden, sondern es können Zugriffsrechte auch an *Rollen* gebunden werden.

Rollen

- `CREATE ROLE <Rollenname>`
- `DROP ROLE <Rollenname>`
- `GRANT <Rollenname> TO <Benutzerliste>`
- `REVOKE <Rollenname> FROM <Benutzerliste>`

Benutzer

Jeder Benutzer wird durch die spezielle Kennung `PUBLIC` identifiziert; `PUBLIC` erteilte Rechte sind automatisch für alle Benutzer gültig.

- Zugriffskontrolle mittels `GRANT` und `REVOKE`.
- Objekte, die mit Zugriffsrechten versehen werden können, sind unter anderem Tabellen, Spalten, Sichten, Wertebereiche (Domains) und Routinen (Funktionen und Prozeduren).
- Die möglichen Rechte sind `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `REFERENCES`, `USAGE`, `TRIGGER` und `EXECUTE`, wobei nicht jedes Recht für jede Art von Objekten angewendet werden kann.
- Syntax:

```
GRANT <Liste von Rechten>
ON <Objekt>
TO <Liste von Benutzern> [WITH GRANT OPTION]

REVOKE [GRANT OPTION FOR] <Liste von Rechten>
ON <Objekt>
FROM <Liste von Benutzern> {RESTRICT | CASCADE}
```

Verwaltung von Rechten über Basistabellen und Sichten

Der Erzeuger einer Basistabelle, hat zu dieser Tabelle alle für eine Tabelle möglichen Rechte, d.h. die Rechte `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `REFERENCES` und `TRIGGER`.

Angenommen der Benutzer `Admin` hat alle Tabellen der Mondial-Datenbank erzeugt und besitzt somit alle Rechte. Das Leserecht zu der Tabelle `Land` soll dem Benutzer `PUBLIC` erteilt werden Außerdem, sollen den Benutzern `Assistent` und `Tutor` die Rechte zum Lesen, Einfügen, Löschen und Ändern zugeteilt werden in der Weise, dass diese Benutzer diese Rechte auch anderen Benutzer erteilen dürfen. Schließlich soll der Benutzer `SysProg` die Rechte `REFERENCES` und `TRIGGER` erhalten.

```
GRANT SELECT ON Land TO PUBLIC

GRANT SELECT, INSERT, DELETE, UPDATE
ON Land TO Assistent, Tutor WITH GRANT OPTION

GRANT REFERENCES, TRIGGER
ON Land TO SysProg
```

Bemerkungen

- Die Definition von Fremdschlüsseln, Integritätsbedingungen und Triggern darf nur bei Besitz entsprechender Rechte erlaubt sein, da sonst indirekt der Inhalt der Tabelle `Land` geschlossen werden könnte.
- Jeder Benutzer, der ein `SELECT`-Recht zu `Land` besitzt, darf eine Sicht über dieser Tabelle definieren.
- Wird eine Sicht über mehreren Tabellen definiert, dann muss das `SELECT`-Recht zu allen diesen Tabellen zugeteilt sein. Weitere Rechte zu der Sicht existieren nur dann, wenn diese Rechte auch für alle der Sicht zugrunde liegenden Tabellen besessen werden.

zum REVOKE

Ein Recht R heißt *verlassen*, wenn das Recht, das für seine Zuteilung erforderlich war, zurückgezogen wurde und keine weitere Zuteilung von R vorgenommen wurde, deren erforderlichen Rechte noch existieren.

- Die Option `CASCADE` veranlaßt zusätzlich zu der Rücknahme des in der `REVOKE`-Klausel benannten Rechts auch die Zurücknahme aller verlassenen Rechte.
- die Option `RESTRICT` führt zum Abbruch der `REVOKE`-Anweisung, wenn verlassene Rechte resultieren.

Benutzer Assistent teilt dem Benutzer Tutor ein INSERT-Recht für Land zu; es folgen eine Reihe von durch Benutzer Admin vorgenommene REVOKE-Anweisungen.

```
GRANT INSERT ON Land TO Tutor
REVOKE INSERT ON Land FROM Tutor RESTRICT
```

Tutor behält das Recht, da er es unabhängig auch von Assistent erhielt.

```
REVOKE INSERT ON Land FROM Assistent CASCADE
```

Jetzt verliert sowohl Assistent, als auch Tutor das Recht. Angenommen, Admin führt anstatt der letzten Anweisung

```
REVOKE GRANT OPTION FOR INSERT ON Land
FROM Assistent CASCADE
```

aus. Jetzt behält Assistent das INSERT-Recht, jedoch Tutor verliert es, da die Erlaubnis für die Vergabe des Rechts an ihn zurückgezogen wurde.

3.19 Arbeiten mit Schema-Definitionen

Alle mit `CREATE` definierten Konstrukte sind Teil eines *Datenbankschemas*.

- SQL bietet Anweisungen an, mit denen existierende Schemata erweitert, oder auch einmal festgelegte Definitionen innerhalb eines Schemas wieder entfernt oder geändert werden können.
- Um einen nachträglichen Bezug zu existierenden Definitionen zu haben, müssen diese Definitionen mit einem Namen versehen werden.
- Die Zuordnung eines Namens ist auch sinnvoll, um im Falle von auftretenden Datenbankfehlern, wie Integritätsverletzungen, einen konkreten Bezug innerhalb einer Fehlernachricht zu bekommen.

Definition eines Schemas.

```
CREATE SCHEMA MondialDatenbank
```

Änderungen eines Schemas

- Mittels einer `DROP`-Anweisung können existierende mittels `CREATE` erzeugte Wertebereiche, Tabellen, Sichten und Assertions entfernt werden.
- Mittels `ALTER` können nachträglich Änderungen vorgenommen werden.
- Interessant ist hier insbesondere das Entfernen von existierenden Spalten und Integritätsbedingungen mittels `DROP`, bzw. das nachträgliche Einfügen mittels `ADD`.

Die Tabelle `Land` wird um eine Spalte `Einwohner` erweitert; des Weiteren wird die Spalte `Hauptstadt` entfernt.

```
ALTER TABLE Land
    ADD COLUMN Einwohner NUMBER

ALTER TABLE Land
    DROP COLUMN HStadt
```

Zwischen Tabellen T1 und T2 bestehe eine zyklische referentielle Beziehung. Aufgrund des gegenseitigen Bezuges ist beim Einfügen von Zeilen mit gegenseitigem Bezug eine Verletzung der Integrität bei direkter Überprüfung nicht vermeidbar.

```
CREATE TABLE T1 CONSTRAINT C1
FOREIGN KEY ... REFERENCES T2
INITIALLY DEFERRED

CREATE TABLE T2 CONSTRAINT C2
FOREIGN KEY ... REFERENCES T1
INITIALLY DEFERRED

   :

INSERT INTO T1 ( ... ) VALUES ( ...).
INSERT INTO T2 ( ... ) VALUES ( ...).
SET CONSTRAINTS C1, C2 IMMEDIATE.
```

LIKE- und AS-Klausel

- SQL:2003 bietet die `CREATE TABLE LIKE`-, bzw. `CREATE TABLE AS`-Klausel an.
- Im ersteren Fall wird die komplette Spaltendefinition einer exsitierenden Tabelle in die neu zu definierende Tabelle übernommen, wobei zusätzlich weitere neue Spalten hinzugenommen werden können.
- Im zweiten Fall wird die neue Tabelle mittels einer beliebigen SFW-Anweisung definiert. Es können somit beliebige Spalten aus existierenden Tabellen ausgewählt werden und es wird gleichzeitig eine Instanz der neuen Tabelle erzeugt.
- In beiden Varianten der `CREATE`-Klausel sind die neuen Tabellen unabhängig von ihren Ursprüngen.

Die Tabelle `Stadt_1` ist wie `Stadt` definiert und enthält zusätzlich eine Spalte `Fläche`.

```
CREATE TABLE Stadt_1 (
   LIKE         Stadt
   Fläche       NUMBER)
```

Die Tabelle `Stadt_2` enthält zusätzlich zu `Stadt` den Anteil an der Gesamtbevölkerung ihres Landes, samt Inhalt.

```
CREATE TABLE Stadt_2 AS
   (SELECT Stadt.*,
      (SELECT SUM(Stadt.Einwohner)/Land.Fläche
      FROM Stadt JOIN Land ON Stadt.LCode = Land.LCode
      GROUP BY Land.LCode, Land.Fläche) AS Anteil
   FROM Stadt) WITH DATA
```