## 4.14.3 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))
```

einfache Wertebereichsbedingungen mittels `NONNUll, DEFAULT` und `CREATE DOMAIN`.

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

```
CREATE DOMAIN meineNamen VARCHAR(35) DEFAULT '?'
CREATE TABLE Land (
   LName meineNamen,
   ⋮
```

## 4.14.4 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(L.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 (NOT EXISTS (
   SELECT LCode FROM Lage
         GROUP BY LCode
         HAVING (SUM(Prozent) <> 100))))
```

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 = HStadt AND S.LCode = LCode) >
         (SELECT AVG(T.Einwohner) FROM Stadt T
            WHERE T.LCode = 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(T.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:

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

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

# 4.15 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 vorgenomen werden.
- Spalten und Integritätsbedingungen können mittels `DROP` entfernt, bzw. mittels `ADD` nachträglich eingefügt werden.

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
```

**DEFERRED und IMMEDIATE Constraints.**

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 (
... );
CREATE TABLE T2 (
...
   CONSTRAINT C2
      FOREIGN KEY ... REFERENCES T1
      INITIALLY DEFERRED );
ALTER TABLE T1 ADD CONSTRAINT C1
   FOREIGN KEY ... REFERENCES T2
   INITIALLY DEFERRED;
   ⋮
INSERT INTO T1 ( ... ) VALUES ( ...);
INSERT INTO T2 ( ... ) VALUES ( ...);
SET CONSTRAINTS C1, C2 IMMEDIATE;
```

- Bei zyklischen Beziehungen kann zunächst die zuerst zu definierende Tabelle ohne `REFERENCES`-Klausel definiert werden.
- Nach erfolgter Definition der zweiten Tabelle die `REFERENCES`-Klausel mittels `ALTER Table` nachholen.
- Außerdem kann festgelegt werden, wann eine Integritätsbedingung überprüft werden soll.
- Zu jeder Integritätsbedingung kann mittels `IMMEDIATE` und `DEFERRED` festgelegt werden, ob sie direkt nach Ausführung einer SQL-Anweisung, oder nach Ausführung einer sie enthaltenden Transaktion überprüft werden soll.
- Diese Angaben können nachträglich modifiziert werden. Bedingungen können als `DEFERRABLE` oder `NOT DEFERRABLE` definiert werden. `INITIALLY` definiert den gültigen Modus für eine Transaktionen zu Beginn ihres Ablaufs. Mittels der `SET CONSTRAINTS`-Anweisung kann dann während der Ausführung einer Transaktion eine Bedingungen `IMMEDIATE` oder `DEFERRED` werden.

## 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 existierenden 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 hat den Inhalt von Stadt und zusätzlich für jede Stadt den Anteil an der Gesamtbevölkerung ihres Landes.

```
CREATE TABLE Stadt_2 AS (
   SELECT S1.*, (
      SELECT S1.Einwohner/SUM(S2.Einwohner)
      FROM Stadt S2 WHERE S1.LCode = S2.LCode ) AS Anteil
   FROM Stadt S1 )
   WITH DATA
```

# 4.16 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 und Objekte

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).

## Rechte

- 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}
```

- Mit `GRANT OPTION` erhaltene Rechte können weitergereicht werden.

## Verwaltung von Rechten

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`.

Beispiel:

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 auf den Inhalt einer Tabelle geschlossen werden könnte.
- Jeder Benutzer, der ein `SELECT`-Recht 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.

```
GRANT INSERT ON Land TO Tutor
```

Es folgen eine Reihe von durch Benutzer Admin vorgenommene REVOKE-Anweisungen.

```
REVOKE INSERT ON Land FROM Tutor
```

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.

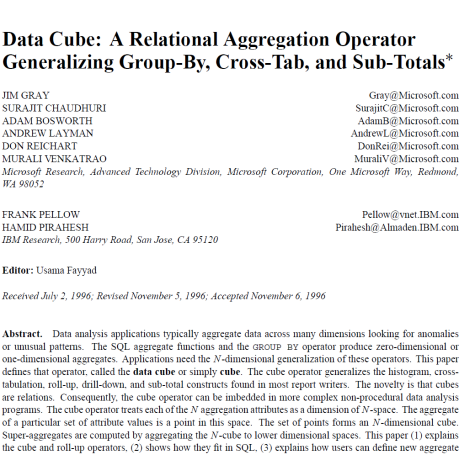
Angenommen, Admin führt anstatt der letzten Anweisung

```
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.

# 4.17 empfohlene Lektüre



**Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals***

JIM GRAY Gray@Microsoft.com
SURAJIT CHAUDHURI SurajitC@Microsoft.com
ADAM BOSWORTH AdamB@Microsoft.com
ANDREW LAYMAN AndrewL@Microsoft.com
DON REICHART DonRei@Microsoft.com
MURALI VENKATRAO MuraliV@Microsoft.com
*Microsoft Research, Advanced Technology Division, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052*

FRANK PELLOW Pellow@vnet.IBM.com
HAMID PIRAHESH Pirahesh@Almaden.IBM.com
*IBM Research, 500 Harry Road, San Jose, CA 95120*

**Editor:** Usama Fayyad

*Received July 2, 1996; Revised November 5, 1996; Accepted November 6, 1996*

**Abstract.** Data analysis applications typically aggregate data across many dimensions looking for anomalies or unusual patterns. The SQL aggregate functions and the GROUP BY operator produce zero-dimensional or one-dimensional aggregates. Applications need the *N*-dimensional generalization of these operators. This paper defines that operator, called the **data cube** or simply **cube**. The cube operator generalizes the histogram, cross-tabulation, roll-up, drill-down, and sub-total constructs found in most report writers. The novelty is that cubes are relations. Consequently, the cube operator can be imbedded in more complex non-procedural data analysis programs. The cube operator treats each of the *N* aggregation attributes as a dimension of *N*-space. The aggregate of a particular set of attribute values is a point in this space. The set of points forms an *N*-dimensional cube. Super-aggregates are computed by aggregating the *N*-cube to lower dimensional spaces. This paper (1) explains the cube and roll-up operators, (2) shows how they fit in SQL, (3) explains how users can define new aggregate functions for cubes, and (4) discusses efficient techniques to compute the cube. Many of these features are being added to the SQL Standard.

[5]

[5] In: Data Mining and Knowledge Discovery 1, 1997. Kann gegoogelt werden.