Praktikum: Datenbanken
Woche 7: Noch mehr SQL
(SELECT-Statement und Trigger)

Raum: LF230 Abgabe bis **6.-8.6. 2005**

Datum	
Gruppe	
Vorbereitung	
Präsenz	

Rekursive Anfragen

In einem vollen SELECT-Statement ist es auch möglich, rekursive Anfragen zu stellen. Rekursive Anfragen sind solche Anfragen, die sich in ihrer Definition auf sich selbst beziehen. Derart kann man etwa die *transitive Hülle* eines Tupels oder Stücklisten berechnen.

db2s1e81.pdf, S. 601

Ein kurzer Ausflug, um das Konzept der Rekursion zu erklären:

> **Rekursion**
> *aus Wikipedia, der freien Enzyklopädie*
> (`http://de.wikipedia.org/wiki/Rekursion`)
>
> Rekursion bedeutet Selbstbezüglichkeit. Sie tritt immer dann auf, wenn etwas auf sich selbst verweist oder mehrere Dinge aufeinander, so dass merkwürdige Schleifen entstehen. So ist z.B. der Satz "Dieser Satz ist unwahr" rekursiv, da er von sich selber spricht. [...]
>
> Dabei ist Rekursion ein allgemeines Prinzip zur Lösung von Problemen, das in vielen Fällen zu "eleganten" mathematischen Lösungen führt. Als Rekursion bezeichnet man den Aufruf bzw. die Definition einer Funktion durch sich selbst. Ohne geeignete Abbruchbedingung geraten solche rückbezüglichen Aufrufe in einen sog. infiniten Regress [Endlosrekursion].
>
> [...] Die Definition von rekursiv festgelegten Funktionen ist eine grundsätzliche Vorgehensweise in der funktionalen Programmierung. Ausgehend von einigen gegebenen Funktionen (wie zum Beispiel unten die Summen-Funktion) werden neue Funktionen definiert, mithilfe derer weitere Funktionen definiert werden können.

Hier ein Beispiel für eine Funktion $sum : N_0 \rightarrow N_0$, die die Summe der ersten n Zahlen berechnet:

$$sum(n) = \begin{cases} 0, & \text{falls } n = 0 \text{ (Rekursionsbasis)} \\ sum(n-1) + n, & \text{falls } n \neq 1 \text{ (Rekursionsschritt)0} \end{cases}$$

Bei einer rekursiven Anfrage (RA) muß man folgende Regeln beachten:

- Jedes Fullselect, das Teil einer RA ist, muß mit SELECT beginnen (aber nicht mit SELECT DISTINCT). Es darf nicht geschachelt sein. Als Vereinigung **muß** UNION ALL benutzt werden.
- Im WITH-Teil müssen Spaltennamen explizit angegeben werden.
- Das erste Fullselect in der ersten Vereinigung darf sich nicht auf die zu definierende Tabelle beziehen. Es bildet die Rekursionsbasis.
- Die Datentypen und Längen der Spalten der zu definierenden Tabelle werden durch die SELECT-Klausel der Rekursionsbasis festgelegt.
- Kein Fullselect in der rekursiven Definition darf eine Aggregatfunktion, eine GROUP BY- oder eine HAVING-Klausel enthalten. Die FROM-Klauseln in der rekursiven Definition dürfen höchstens eine Referenz auf die zu definierende Tabelle besitzen.
- Es dürfen keine Subqueries verwendet werden.

Wenn die Daten Zyklen enthalten können, dann ist es möglich, dass durch eine RA eine Endlosrekursion erzeugt wird. Dann ist es wichtig, eine Abbruchbedingung zu definieren:

- Die zu definierende Tabelle muß ein Attribut enthalten, das durch eine INTEGER-Konstante initialisiert wird, und regelmäßig erhöht wird.
- In jeder WHERE-Klausel muß eine Prüfbedingung auftreten.

Die Auswertung einer rekursiven Anfrage läuft wie folgt ab: Zuerst werden die Tupel der Rekursionsbasis berechnet, dann werden ausgehend von diesen Tupeln gemäß des zweiten Teils des Fullselects (nach dem UNION ALL) weitere Tupel berechnet. Dabei werden jedoch nur diejenigen Tupel aus der rekursiv definierten Relation verwendet, die beim letzten Berechnungsschritt neu hinzugekommen sind (beginnend also mit den Tupeln der Rekursionsbasis). Kamen in einem Berechnungsschritt keine neuen Tupel hinzu, so endet die Berechnung.

Beispiel:

Gegeben sei eine Tabelle mit Bauteilen, in denen die Komponten für Werkstücke festgehalten werden. Dabei können diese Komponenten selber wiederum aus Einzelteilen zusammengesetzt sein.

```
CREATE TABLE bauteile (
       stueck     VARCHAR(8),
       komponente VARCHAR(8),
       menge      INTEGER);
```

Um nun zu ermitteln, welche Basiskomponenten insgesamt nötig sind, um ein Beispiel-Bauteil herzustellen, kann man eine rekursive Anfrage benutzen. In der folgenden RA ist die im ersten Teil der WITH-Klausel definierte Query `rek` die Rekursionsbasis, im zweiten Teil der WITH-Klausel folgt dann die Definition der Rekursion. Die so rekursiv definierte Ergebnisrelation wird dann im SELECT DISTINCT-Teil benutzt.

```
WITH rek (stueck, komponente, menge) AS (
        SELECT r.stueck, r.kompenente, r.menge
        FROM bauteile r
        WHERE r.stueck = 'Beispiel-Bauteil'
   UNION ALL
        SELECT kind.stueck, kind.komponente, kind.menge
        FROM rek vater,
             bauteile kind
        WHERE vater.komponente = kind.stueck
     )
SELECT DISTINCT stueck, komponente, menge
FROM rek
ORDER BY stueck, komponente, menge
```

Trigger

Als Trigger (deutsch Auslöser) bezeichnet man in SQL eine Anweisungsfolge (eine Abfolge von Aktionen), die ausgeführt wird, wenn eine verändernde Anweisung auf einer bestimmten Tabelle ausgeführt werden soll. Wir erinnern uns aus früherer Übung, dass verändernde Anweisungen für Tabellen DELETE, INSERT und UPDATE sind.

Man kann Trigger zum Beispiel nutzen, um

- neu eingefügte oder zu verändernde Tupel auf ihre Gültigkeit bezüglich vorgegebener Bedingungen zu überprüfen,
- Werte zu generieren,
- in anderen Tabellen zu lesen (etwa zur Auflösung von Querverweisen) oder zu schreiben (etwa zur Protokollierung)

Insbesondere kann man mit Triggern Regeln in der Datenbank modellieren. Einen Trigger erstellt man mit dem CREATE TRIGGER-Statement, mit DROP TRIGGER wird er gelöscht.

db2s2e81.pdf, S. 414

Man kann Trigger entweder so schreiben, dass sie für jedes zu ändernde Tupel einmal (FOR EACH ROW) oder für jedes verändernde Statement einmal (FOR EACH STATEMENT) ausgeführt werden. Letzteres ist allerdings nur für die sogenannten AFTER-Trigger erlaubt, die im Anschluß an eine verändernde Operation tätig werden. In diesem Fall wird der Trigger auch dann abgearbeitet, wenn kein Tupel von dem DELETE- oder dem UPDATE-Statement betroffen ist.

Über die REFERENCING-Klausel werden Namen für die Übergangsvariablen festgelegt: `OLD AS name` bzw. `NEW AS name` definiert `name` als Name für die Werte des betroffenen Tupels vor bzw. nach der Ausführung der auslösenden Operation. Entsprechend definieren `OLD_TABLE AS identifier` bzw. `NEW_TABLE AS identifier` diesen als Name für eine hypothetische Tabelle, welche die betroffenen Tupel vor bzw. nach der auslösenden Operation enthält. Dabei gilt :

auslösendes Ereignis und Zeitpunkt	**ROW-Trigger darf benutzen**	**STATEMENT-Trigger darf benutzen**
BEFORE INSERT	NEW	-
BEFORE UPDATE	OLD, NEW	-
BEFORE DELETE	OLD	-
AFTER INSERT	NEW, NEW_TABLE	NEW_TABLE
AFTER UPDATE	OLD, NEW, OLD_TABLE, NEW_TABLE	OLD_TABLE, NEW_TABLE
AFTER DELETE	OLD, OLD_TABLE	OLD_TABLE

Über vorhandene Trigger auf einer Datenbank kann man die System-Tabelle `SYSIBM.SYSTRIGGERS` konsultieren:

```
SELECT name,text
FROM SYSIBM.SYSTRIGGERS
```

Beispiel:

Angenommen, wir führten eine Tabelle `opl` (Mitgliedschaften in Organisationen pro Land), dann könnte ein Trigger für das Einfügen von neuen Tupeln in die Tabelle `schauspieler` eventuell derart aussehen:

```
CREATE TRIGGER neue_mitgliedschaft
   AFTER INSERT ON Mitglied
   REFERENCING NEW AS n
   FOR EACH ROW MODE DB2SQL
   UPDATE fps
      SET anzahl_mitgliedschaften = anzahl_mitgliedschaften + 1
      WHERE n.land = opl.land
```

Und ein Einfügen eines Tupels in `schauspieler` würde automatisch zu einem UPDATE der Tabelle `fps` führen:

```
INSERT INTO opl (land, anzahl_mitgliedschaften)
VALUES ('Fantasia', 0);

INSERT INTO mitglied (land, organisation, art)
VALUES ('Fantasia', 'Der Club', 'member');

SELECT anzahl_mitgliedschaften
FROM opl
WHERE land='Fantasia'

ANZAHL
------
     1
```

Noch ein Beispiel:

```
CREATE TRIGGER check_year
   NO CASCADE BEFORE INSERT ON politik
   REFERENCING NEW AS n
   FOR EACH ROW MODE DB2SQL
   IF n.unabhaendigkeit > current date
   THEN
      SIGNAL SQLSTATE '75000'
      SET MESSAGE_TEXT='Kein Einfuegen von zukünftigen Ländern möglich.';
   END IF
```

Dieser Trigger verbietet das Einfügen neuer Tupel in `politik`, bei denen das Unabhängigkeitsjahr in der Zukunft liegt. Der Versuch führt zu einem Fehler und das Statement wird nicht ausgeführt.

Aufgaben

Zur Vorbereitung:

Aufgabe V7.1 Dokumentation

Macht Euch mit Hilfe der angegebenen Handbuchabschnitte, bzw. der Online-Dokumentation oder anderer Literatur mit der Syntax des WITH...SELECT-Statements und des CREATE TRIGGER-Statements vertraut.

Aufgabe V7.2 SELECT-Statement

Was sind andere Nutzen für einen in der WITH-Klausel definierten Tabellenausdruck außer der Verwendung in rekursiven Anfragen.

Aufgabe V7.3 Referentielle Integrität

In einer der vergangenen Wochen haben wir Constraints als eine Möglichkeit kennengelernt, referentielle Integritätsbedingungen festzulegen. Schreibe nun Trigger, die bei der Änderung eines Landescodes oder eines Provinznamen diese Umbenennungen in der gesamten Datenbasis ausführen.

Aufgabe P7.4 Transitive Hülle

(a) Bestimmt ausgehend von Deutschland alle Länder, die man bereisen kann, ohne dabei durch ein Land zu reisen, dass kein NATO-Mitglied ist. Bricht diese Rekursion von alleine ab? Wenn nicht, was für eine Abbruchbedingung könnte man setzen?

(b) Bestimmt alle Quellflüsse (direkt *und* indirekt des Zaire und berechne die Gesamtlänge des Flussnetzes.

Aufgabe P7.5 Abhängige Werte

Erweitert die Relation, in der ihr Landesdaten speichert (`Land` in der Beispieldatenbank) um eine neue Spalte `Bevölkerungsdichte`, die stets den aktuellen Wert $\frac{Bevoelkerung}{Gebiet}$ enthält. Füllt diese Spalte mit den korrekten Werten.

Schreibt dann Trigger, die folgendes leisten:

(a) Wird die Einwohnerzahl oder Fläche verändert, so wird der neue Wert der Bevölkerungsdichte berechnet.

(b) Wird die Bevölkerungsdichte verändert, so soll die Einwohnerzahl neu bestimmt werden.

Überlegt Für und Wider einer solchen Implementation.