**Praktikum Datenbanken / DB2**
**Woche 8: Trigger, SQL-PL**

Betreuer: Sascha Kriewel, Tobias Tuttas
Raum: LF 230
Bearbeitung: 26., 27. und 29. Juni 2006

| | |
|---|---|
| **Datum** | |
| **Team (Account)** | |
| | |
| **Vorbereitung** | |
| **Präsenz** | |

Aktuelle Informationen, Ansprechpartner und Material unter:
`http://www.is.inf.uni-due.de/courses/dbp_ss07/index.html`

# Wochenziele

In dieser Woche soll es um `Trigger` gehen. Zudem werden einige frühere Konzepte noch einmal wiederholt.

# 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 früheren Übungen, 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 `afps` (Anzahl Folgen pro Serie), dann könnte ein Trigger für das Einfügen von neuen Tupeln in die Tabelle `serienfolge` eventuell derart aussehen:

```
CREATE TRIGGER neue_folge
   AFTER INSERT ON serienfolge
   REFERENCING NEW AS neu
   FOR EACH ROW MODE DB2SQL
   UPDATE afps
      SET anzahl_folgen = anzahl_folgen + 1
      WHERE neu.serientitel = afps.serientitel
        AND neu.drehjahr = afps.drehjahr
```

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

```
INSERT INTO afps (serientitel, drehjahr, anzahl_folgen)
VALUES ('Babylon 5 - The Telepath Tribes', 0);

INSERT INTO serienfolge (serientitel, drehjahr, staffelnr,
        folgennr, folgentitel)
VALUES ('Babylon 5 - The Telepath Tribes', 2007, 1, 1,
        'Forget Byron');

INSERT INTO serienfolge (serientitel, drehjahr, staffelnr,
        folgennr, folgentitel)
VALUES ('Babylon 5 - The Telepath Tribes', 2007, 1, 2,
        'Bester''s Best');

SELECT anzahl_folgen AS Anzahl
FROM afps
WHERE land='Babylon 5 - The Telepath Tribes'

ANZAHL
------
     2
```

**Noch ein Beispiel:**

```
CREATE TRIGGER check_ausleihe
   NO CASCADE BEFORE INSERT ON ausleihe
   REFERENCING NEW AS neu
   FOR EACH ROW MODE DB2SQL
   IF neu.ausleihdatum > current date
   THEN
      SIGNAL SQLSTATE '75000'
      SET MESSAGE_TEXT =
        'Ein Ausleihvorgang kann nicht im voraus eingefügt werden.';
   END IF
```

Dieser Trigger verbietet das Einfügen neuer Tupel in `ausleihe`, bei denen das Aus-

leihdatum in der Zukunft liegt. (Es sollen also nur bereits stattgefundene Ausleihvorgänge eingetragen werden.) Der Versuch führt zu einem Fehler und das Statement wird nicht ausgeführt.

## Übersicht über die neuen Befehle

| | |
|---|---|
| create trigger *name* ... | erstelle einen neuen Trigger *name* |
| drop trigger *name* ... | lösche den Trigger *name* |

# Aufgaben

## Vorbereitung (Hausaufgaben)

### V1: Verwendung von Triggern

(a) Überlegt Euch mindestens zwei weitere Beispiele in Eurer eigenen Datenbank, wo Ihr Trigger sinnvoll einsetzen könntet.

(b) Vergleicht Trigger mit Constraints. Werden beide für dieselbe Zwecke verwendet, oder gibt es Unterschiede? Wann ist es sinnvoller, mit Triggern statt mit Constraints zu arbeiten?

### V2: Verwendung von Funktionen

Welche Unterschiede seht Ihr in der Verwendung von Triggern bzw. benutzerdefinierten Funktionen? Wann sind Trigger sinnvoller, wann eigene Funktionen? Gibt es in Eurer eigenen Datenbank weitere sinnvolle Anwendungensmöglichkeiten für skalare oder für tabellenwertige Funktionen?

### V3: Wiederholung

Wiederholt das Material zu Datenbank-Indexen aus der Woche 3.

## Präsenz

### P1: Indexe

Lasst Euch zu den existierenden Tabellen Eurer Datenbank die bestehenden Indexe anzeigen.

```
describe indexes for table TABELLENNAME show detail
```

Versucht (ggf. mit Hilfe von Online-Dokumentation zu verstehen, was diese Angaben bedeuten. Warum existieren auch für solche Tabellen Indexe, für die keine explizit angelegt wurden?

Erstellt einen eigenen Index für eine Eurer Tabellen (z.B. zur beschleunigten Suche nach einem Attributpaar).

### P2: Update- und Delete-Trigger

Betrachtet den Beispiel-Trigger `neue_folge` aus dem Vorbereitungsmaterial. Legt eine Tabelle `afps` (Anzahl Folgen pro Serie), sowie den angepassten Trigger für Eure Datenbank an.

Schreibt dann zusätzliche Update- und Delete-Trigger, die entsprechend die Anzahl der Serienfolgen bei einem Löschen eines Folgentupels verringern, bzw. die Anzahl der Folgen bei einer Änderung des Seriennamens entsprechend anpasst.

### P3: Trigger zum Überprüfen von Telefonnummern

Schreibt einen Trigger `checkFon`, um beim Einfügen von Telefonnummern deren Syntax zu überprüfen. Zulässige Telefonnummern sollen ausschließlich aus den Zeichen '/', '␣' (Leerzeichen), '-' und Ziffern bestehen.

Entspricht die Syntax nicht der Vorgabe, soll das Einfügen mit einer Fehlermeldung verweigert werden.

**Zum Schluss ...**
**stoppt bitte Eure Datenbank-Manager-Instanz auf `salz`,**
**loggt Euch von `salz` und vom lokalen Rechner aus,**
**aber schaltet den Rechner nicht ab.**