Trigger

Im bisherigen Unterricht wurden Datenbanksysteme weitgehend als **"passive"** SW-Systeme beschrieben: Aktionen werden vom DBMS **nicht selbstständig** ausgeführt, sondern nur, wenn es dazu aufgefordert wird (Ausnahme: `CASCADE` bzw. `SET NULL` bei FK-Spalten).

Trigger bieten nun die Möglichkeit **SQL-Anweisungen automatisch ausführen** zu lassen, wenn in einer Tabelle ein neuer Datensatz **eingefügt** wird oder ein bestehender Datensatz **geändert** bzw. **gelöscht** wird. In diesem Fall übernimmt das DBMS eine **"aktive"** Rolle.
Die Unterscheidung zwischen "aktiven DBMS" und "passiven DBMS" erfolgt somit abhängig davon, ob ein DBMS auch selbstständig SQL-Anweisungen ausführen kann.

Trigger werden mittels DDL definiert. Dabei wird festgelegt **bei welcher Tabelle** und **bei welchem auslösenden Ereignis** der Trigger jeweils automatisch aktiviert werden soll und **welche Verarbeitung** dabei durchgeführt werden soll. Für **jede Kombination** aus Tabelle und auslösendem Ereignis ist **ein eigener Trigger** zu definieren.
Beispiel: in zwei Tabellen jeweils `INSERT` überwachen → 2 Trigger erforderlich,
in einer Tabelle jeweils `INSERT`, `DELETE` + `UPDATE` überwachen → 3 Trigger erforderlich.

Einsatzmöglichkeiten

Trigger können eingesetzt werden, um:

- redundante Daten automatisch zu aktualisieren
  (z.B. um Statistikdaten zu erstellen bzw. zu aktualisieren),
- auf Integritätsverletzungen gezielt zu reagieren
  (z.B. um unzulässige Werte automatisch zu korrigieren),
- bei Änderungen die Differenz `wert_neu – wert-alt` zu bestimmen
  und anhand dieser Differenz gezielt zu reagieren (z.B. um Statistiken zu aktualisieren),
- Aktivitäten in der Datenbank zu dokumentieren
  (z.B. um Versionskennzeichen bei Änderungen zu erhöhen, Log-Files zu erstellen, etc.).

Trigger in PostgreSQL

*"It is **not currently possible** to write a trigger function **in the plain SQL function language**."*
[PostgreSQL Documentation, Chapter 36: Triggers]

Abhilfe: Trigger mittels PL/pgSQL definieren.
[PostgreSQL Documentation, Chapter 40.6: Trigger Procedures]

Trigger in MySQL

Offizielle Dokumentation – siehe Kapitel 20:

"Support for triggers is included beginning with MySQL 5.0.2."

"Prior to MySQL 5.0.10, triggers cannot contain direct references to tables by name."

*"MySQL 5.1 **does not support** triggers using FOR EACH **STATEMENT**."*

Allgemeine Syntax

```
DROP TRIGGER <trigger_name>;

CREATE TRIGGER <trigger_name>
     BEFORE | AFTER
     INSERT | DELETE | UPDATE
     ON <tab_name>
     FOR EACH ROW | STATEMENT
BEGIN
     <trigger_verarbeitung>
END;
```

Beispiele (MySQL)

```
mysql> DELIMITER //
mysql> CREATE TRIGGER preis_check BEFORE UPDATE ON speise
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.preis < 0 THEN
    ->         SET NEW.preis = 0;
    ->     ELSEIF NEW.preis > 100 THEN
    ->         SET NEW.preis = 100;
    ->     END IF;
    -> END;//
mysql> DELIMITER ;
mysql>
```

Durch die Angaben `DELIMITER //` sowie `DELIMITER ;` wird die Zeichenfolge `//` als **Abschluss der Kommandoeingabe** definiert. Damit können innerhalb der Trigger-Definition **mehrere SQL-Anweisungen** jeweils durch `;` getrennt verwendet werden, ohne dass dies bei der Eingabe der Trigger-Definition zur Unterbrechung oder vorzeitigen Abarbeitung der Eingabe führt. Achtung: nach `DELIMITER` muss unbedingt ein Leerzeichen folgen!

Im Trigger kann auf die Spalten der **getriggerten SQL-Anweisung** zugegriffen werden:

- `OLD.spalte` repräsentiert **den alten Wert** bzw. den Wert **vor der Änderung**,
- `NEW.spalte` repräsentiert **den neuen Wert** bzw. den Wert **nach der Änderung**.

Der **neue Wert** kann mit `SET NEW.spalte = <neuer_Wert>;` **überschrieben** werden.

Trigger können auch DML-Anweisungen für **Änderungen in anderen Tabellen** enthalten. Diese SQL-Anweisungen müssen **ohne `OLD.` und ohne `NEW.` formuliert werden**, da sich `OLD.` bzw. `NEW.` jeweils nur auf den Datensatz der getriggerten Tabelle beziehen!

Trigger werden häufig **zur Vermeidung des Lost-Update-Problems** eingesetzt.
Im folgenden Beispiel wird die Versionsnummer in der Tabelle `rechnung` um 1 erhöht, sobald ein neuer Datensatz in die Tabelle `bestellung` eingefügt wird.
Für `DELETE` und `UPDATE` sind noch **zwei weitere ähnliche Trigger erforderlich**!

```
mysql> DELIMITER //
mysql> CREATE TRIGGER bestellung_a_i
    -> AFTER INSERT ON bestellung
    -> FOR EACH ROW
    -> BEGIN
    ->     UPDATE rechnung
    ->     SET    version = version + 1
    ->     WHERE  rid = NEW.rid;
    -> END;//
mysql> DELIMITER ;
mysql>
```

Übungen (MySQL)

Portiere die zuletzt verwendete Datenbank `restaurant` von PostgreSQL nach MySQL und ergänze das DDL-Script um folgende Trigger-Definitionen:

a. Wenn bei einer zu speichernden `rechnung` die Spalte `datum` den Wert `NULL` hat, soll sie jeweils durch den aktuellen Wert `CURRENT_DATE` ersetzt werden.

b. Wenn in der Tabelle `speise` ein `preis` geändert wird, soll ein zusätzlicher Datensatz in der Tabelle `preisaenderung` eingefügt werden. In der Spalte `datum` soll das aktuelle Datum gespeichert werden und in der Spalte `aenderung` soll die Preisänderung gespeichert werden. `PK (snr, datum).`

c. Wenn in der Tabelle `bestellung` ein Datensatz gelöscht wird, soll ein zusätzlicher Datensatz in der Tabelle `bestellstorno` gespeichert werden.

d. In der Tabelle `statistik` soll datumsabhängig die Anzahl aller im Restaurant angebotenen Speisen dokumentiert werden. Erstelle alle erforderlichen Definitionen!

e. Die `ENGINE=MYISAM` gestattet die Verwendung von Triggern. Realisiere eine 1:N-Beziehung mit der `ENGINE=MYISAM`, d.h. das FK-Constraint und die `CASCADE`-Verarbeitung sollen mittels Triggern nachgebildet werden.

Kontrollfragen

Beantworte folgende Kontrollfragen:

- Sind mehrere Trigger für dasselbe Aktivierungs-Event (z.B. `BEFORE INSERT ON xxx`) zulässig?
- Können innerhalb `BEGIN … END` mehrere SQL-Anweisungen definiert werden?
- Können Trigger die Anweisungen `START TRANSACTION`, `COMMIT` oder `ROLLBACK` enthalten?
- Was bewirkt das Constraint `NOT NULL` in Kombination mit einem `BEFORE`-Trigger?
  `IF NEW.xxx IS NULL THEN SET NEW.xxx = ... END IF;`
- Kann in einem Trigger auf Datensätze einer anderen Tabelle zugegriffen werden?
- Welche der folgenden Kombinationen / Zugriffe sind zulässig?
  Ergänze die folgende Tabelle! Begründe die Aussagen!

		BEFORE			**AFTER**		
		INSERT	DELETE	UPDATE	INSERT	DELETE	UPDATE
NEW.xxx	lesen			ja 1)			
	ändern			ja 1)			
OLD.xxx	lesen	nein 2)			nein 2)		
	ändern	nein 2)					

Begründungen:

1) siehe Beispiel auf S. 2 in trigger.pdf
2) "vor" einem INSERT existiert der Datensatz noch nicht → kein OLD
3) …
4) …
…