

SQL: dynamische Integrität

Dynamische Integritätsbedingungen definieren zulässige Zustandsübergänge von Datenbankinstanzen, bzw. Aussagen über die Vergangenheit (**temporale Bedingungen**).

Beispiele:

Lebenszyklus von Objekten (Wechsel des Familienstandes), Gesetz "Gehälter dürfen nicht fallen", Bedingung "der aktuelle Energieverbrauch muß 5% unter dem durchschnittlichen Verbrauch der letzten 10 Jahre liegen", etc.

Gewährleistung der referentiellen Integrität

Referentielle Aktionen

Aufgabe **referentieller Aktionen** bzgl. einer C-Tabelle ist die Kompensierung von durch DELETE und UPDATE-Operationen auf einer zugehörigen P-Tabelle verursachten Verletzungen der Integrität.

Ist dies nicht möglich, so werden die gewünschten DELETE/UPDATE-Operationen nicht ausgeführt, bzw. zurückgesetzt.

Beispiel:

```
CREATE TABLE LT ...
  PRIMARY KEY (LNr, TNr),
  FOREIGN KEY (LNr) REFERENCES Lieferant
    ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (TNr) REFERENCES Teil
    ON DELETE CASCADE ON UPDATE CASCADE)
```

Syntax der REFERENCES-Klausel (vereinfacht)

```
REFERENCES base-table [ ( column-commalist ) ]
  [ ON DELETE { NO ACTION | RESTRICT | CASCADE | SET DEFAULT | SET NULL } ]
  [ ON UPDATE { NO ACTION | RESTRICT | CASCADE | SET DEFAULT | SET NULL } ]
```

Die Klauseln ON DELETE und ON UPDATE geben die bei DELETE und UPDATE auf den P-Tabellen zur Gewährleistung der referentiellen Integrität gewünschten **referentiellen Aktionen** auf den C-Tabellen an. Fehlt die ON DELETE- oder ON UPDATE-Klausel, so ist Default NO ACTION.

Bemerkung:

- Ein INSERT bzgl. der P-Tabelle oder ein DELETE bzgl. der C-Tabelle ist immer unkritisch.
- Ein INSERT bzgl. der C-Tabelle oder ein UPDATE bzgl. der C-Tabelle, die einen Fremdschlüsselwert erzeugen, zu dem kein Schlüssel in der P-Tabelle existiert, sind immer unzulässig; anderenfalls sind sie unkritisch.
- Die references-condition definiert somit lediglich die referentiellen Aktionen für DELETE und UPDATE bzgl. der P-Tabelle.

(einige) Lösch- und Änderungs-Regeln:**NO ACTION:**

Die DELETE/UPDATE-Operation auf der P-Tabelle wird zunächst ausgeführt; ob "dangling references" in der C-Tabelle entstanden sind wird erst nach Abarbeitung aller ausgelösten referentiellen Aktionen überprüft.

RESTRICT:

Die DELETE/UPDATE-Operation auf der P-Tabelle wird nur dann ausgeführt, wenn keine "dangling references" in der C-Tabelle entstehen.

CASCADE:

Die DELETE/UPDATE-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 bzgl. einer anderen Bedingung, so wird das DELETE/UPDATE bzgl. der dort festgelegten Lösch/Änderungs-Regel weiter behandelt.

Mehrdeutigkeiten

In Abhängigkeit von der Reihenfolge der Abarbeitung der FOREIGN KEY-Klauseln können in Abhängigkeit vom Inhalt der Tabellen unterschiedliche Ergebnisse resultieren.

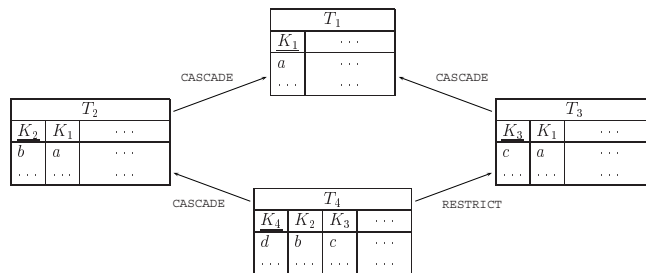
Beispiel:

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

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

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

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



Angenommen das Tupel in T1 wird gelöscht und es werden die referentiellen Aktionen von T3 vor T4 vor T2 betrachtet. Das Tupel in T3 wird gelöscht (CASCADE-Klausel in T3); das Tupel in T4 kann nicht gelöscht werden (RESTRICT-Klausel in T4). Zur Aufrechterhaltung der referentiellen Integrität wird das auslösende Löschen des Tupels in T1 und alle dadurch implizierten Löschungen wieder rückgesetzt.

Werden die referentiellen Aktionen von T2 vor T4 vor T3 betrachtet, so werden alle angegebenen Tupel gelöscht.

Ersetzt man RESTRICT durch NO ACTION, so wird unabhängig von der Reihenfolge der betrachteten referentiellen Aktionen das Ergebnis eindeutig.

Trigger

Trigger sind ein mächtiges und flexibles Programmierungskonzept, mit dem u.a. die Integrität gewährleistet, bzw. überprüft werden kann.

Trigger sind ein Spezialfall aktiver Regeln: **EventConditionAction-Paradigma**.

Beispiel:

Zu jeder Gehaltserhöhung wird noch ein Bonus von 100,-DM verrechnet.

```
CREATE TRIGGER AngestTrig
  AFTER UPDATE OF Gehalt ON Angest
  REFERENCING OLD AS oldrow NEW AS newrow
  WHEN (newrow.Gehalt > oldrow.Gehalt)
  SET newrow.Gehalt = newrow.Gehalt + 100
  FOR EACH ROW
```

Ereignis: Ein Trigger ist einer Tabelle zugeordnet. Er wird aktiviert durch das Eintreten eines Ereignisses (SQL-Anweisung): Einfügung, Änderung und Löschung von Zeilen seiner Tabelle.

Aktivierung: 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.

Granularität: Ein Trigger kann einmal pro aktivierender Anweisung (Statement-Trigger) oder einmal für jede betroffene Zeile (Row-Trigger) seiner Tabelle ausgeführt werden.

Transitions-Variablen: 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.

Bedingung: Ein aktivierter Trigger wird ausgeführt, wenn seine Bedingung erfüllt ist.

Rumpf: Der Rumpf eines Triggers enthält die auszuführenden SQL-Anweisungen.

Trigger-Syntax (vereinfacht):

```
CREATE TRIGGER trigger-name
[ BEFORE | AFTER | INSTEAD OF] [ INSERT | DELETE | UPDATE { OF col-name }0 ]
                                ON table-name
{ REFERENCING { [ OLD | NEW | OLD TABLE | NEW TABLE ] AS transition-variable }1
{ WHEN ( trigger-condition ) } triggered-SQL-statements
[ FOR EACH STATEMENT | FOR EACH ROW ]
```

Ausführung des Triggers als:

BEFORE-Trigger: Können verwendet werden, um vor der eigentlichen Ausführung der aktivierenden Anweisung die Integrität der Werte der betreffenden Tupel zu testen, bzw. um Werte von Attributen zu berechnen, die die in der aktivierenden Anweisung zunächst vorgesehenen Werte überschreiben.

AFTER-Trigger: Können zur Gewährleistung der Integrität verwendet werden, indem entsprechende weitere Anweisungen ausgeführt werden, bzw. die aktivierende Anweisung abgebrochen und zurückgesetzt wird.

INSTEAD OF-Trigger: Können verwendet werden, um anstatt der aktivierenden Anweisung die Anweisungen des Triggers auszuführen.

Sichtbarkeit von Änderungen:

Mittels der REFERENCING-Klausel wird festgelegt unter welchen Bezeichnern die Werte der geänderten Tupel vor, bzw. nach Ausführung der den Trigger auslösenden Operation verfügbar sind.

Bei einem Statement-Trigger bezieht sich dies auf eine Tabelle (OLD TABLE, NEW TABLE), bei einem Row-Trigger auf eine Zeile (NEW, OLD).

Es muß geklärt werden, ob die Effekte der den Trigger auslösenden Operation bei der Ausführung der Trigger-Anweisung in der dem Trigger zugeordneten Tabelle sichtbar sind, oder nicht.

INSERT: Bei einem BEFORE- oder INSTEAD OF-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.

DELETE: Bei einem BEFORE- oder INSTEAD OF-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.

UPDATE: Bei einem BEFORE, AFTER- oder INSTEAD OF-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- oder INSTEAD OF-Trigger sind die Änderungen nicht sichtbar in der Tabelle, bei einem AFTER-Trigger jedoch.

Diskussion am Beispiel einer Angestellten-Tabelle:

```
Angest(AngNr, Name, AbtNr, Arbeitscode, Projekt, Manager,
      MitArb, Gehalt, Bonus)
```

Before-Trigger:

Für jeden neuen Angestellten muß das Anfangsgehalt und der Bonus so sein, wie in der Tabelle StartGehalt vorgesehen.

```
CREATE TRIGGER AngestTrig2
  BEFORE INSERT ON Angest
  REFERENCING NEW AS newrow
  SET (newrow.Gehalt, newrow.Bonus) =
    (SELECT Gehalt, Bonus
     FROM StartGehalt
     WHERE Arbeitscode =
        newrow.Arbeitscode)
  FOR EACH ROW
```

Wichtige Angestellte dürfen nicht "gelöscht" werden.

```
CREATE TRIGGER AngestTrig4
  BEFORE DELETE ON Angest
  REFERENCING OLD AS oldrow
  WHEN (importance(oldrow.Arbeitscode, oldrow.Projekt) > 20)
  SIGNAL SQLSTATE '70011' ('wird noch gebraucht!')
  FOR EACH ROW
```

SIGNAL erzeugt hier einen speziellen Systemfehler, der rekursiv zum Abbruch mit Rücksetzen der auslösenden Anweisung führt.

Rekursive Trigger:

Trigger können selbst weitere Trigger aktivieren, wenn ein ausgelöster Trigger eine Tabelle modifiziert, über der selbst Trigger definiert sind. Eine solche Aktivierungsfolge kann zyklisch sein (**rekursive** Trigger); die Terminierung ist i.a. nicht gesichert.