mvdbs: Übung Trigger

Yanick Eberle, Pascal Schwarz

24. Februar 2013

1 Aufgabe 1 - Event Logging

1.1 Lösungsidee

Wir erstellen einen Trigger, welcher bei den SQL Statements, die potenziell Änderungen an der Tabelle „Ausleihen“ bewirken, ausgelöst wird. Wie in der Aufgabenstellung beschrieben sind dies die folgenden SQL-Befehle:

- INSERT
- UPDATE
- DELETE

Aufgrund der Anforderung sowohl die alten wie auch die neuen Werte zu protokollieren, muss unser Trigger jeweils vor dem Statement ausgeführt werden. In einem solchen Trigger haben wir Zugriff auf die neuen Werte.

1.2 Tabelle Ausleihen_Log

Die Log-Tabelle enthält einen eigenen Primary Key (Number(6,0)). Die geforderten Angaben (User, welcher die Änderung vorgenommen hat (VARCHAR2(20)), Art der Änderung (VARCHAR(3)) sowie Zeitpunkt der Änderung (TIMESTAMP)) werden jeweils in einem Attribut abgelegt.

Zusätzlich erhält die Log-Tabelle für jedes Attribut der Tabelle Ausleihen zwei Attribute. In *Feldname*_old wird der Wert vor der Änderung, in *Feldname*_new der Wert nach der Änderung festgehalten. Diese Attribute haben jeweils den selben Datentyp wie das jeweilige Attribut in der Tabelle „Ausleihen“.

1.3 Trigger für Protokollierung

Da ein Statement grundsätzlich mehrere Zeilen der Tabelle „auf einmal“ verändern kann, muss der Trigger mit der Granularität „FOR EACH ROW“ definiert werden.

Wie bei der Lösungsidee bereits beschrieben wird der Trigger vor den Events INSERT, UPDATE und DELETE ausgelöst.

1.4 SQL Statements

1.4.1 Ausleihen_Log

```
1  CREATE TABLE AUSLEIHEN_LOG
2  (
3    log_id NUMBER(6, 0) NOT NULL
4  , CHANGE_USER VARCHAR2(20) NOT NULL
5  , CHANGE_DATE TIMESTAMP NOT NULL
6  , CHANGE_TYPE VARCHAR2(3) NOT NULL
7  , MNR_OLD VARCHAR2(4)
8  , MNR_NEW VARCHAR2(4)
9  , DVDNR_OLD NUMBER(6, 0)
10 , DVDNR_NEW NUMBER(6, 0)
11 , DATUM_OLD DATE
12 , DATUM_NEW DATE
13 , RUECKGABE_OLD DATE
14 , RUECKGABE_NEW DATE
15 , CONSTRAINT AUSLEIHEN_LOG_PK PRIMARY KEY (log_id) ENABLE
16 );
```

1.4.2 Trigger

```
1  CREATE OR REPLACE TRIGGER ausleihen_logger
2    BEFORE UPDATE OR INSERT OR DELETE ON ausleihen
3    FOR EACH ROW
4    DECLARE
5      manipulation varchar2(3);
6    BEGIN
7      if inserting then
8        manipulation := 'INS';
9      elsif deleting then
10       manipulation := 'DEL';
11     elsif updating then
12       manipulation := 'UPD';
13     else
14       manipulation := 'ERR';
15     end if;
16
17     INSERT INTO ausleihen_log (change_user, change_date, change_type, mnr_old,
           mnr_new, dvdnr_old, dvdnr_new, datum_old, datum_new, rueckgabe_old,
           rueckgabe_new)
18     VALUES (user, sysdate, manipulation, :old.mnr, :new.mnr, :old.dvdnr, :new.dvdnr,
           :old.datum, :new.datum, :old.rueckgabe, :new.rueckgabe);
19 END;
```

1.5 Tests

2 Aufgabe 2 - Referential Integrity

2.1 Lösungsidee / Vorbereitung

Für das Verschieben der Tabelle „Filme“ muss der Foreign Key Constraint „DK_FM_FK“ auf der Tabelle „DVDKopien“ zunächst entfernt werden. Ansonsten kann die Tabelle nicht entfernt werden.

Als nächster Schritt wird ein Database-Link auf dem Server telesto (dort sind alle Tabellen ausser Filme) erstellt:

```
create database link orion.helios.fhnw.ch
connect to mvdbs10 identified by mvdbs10
using 'orion'
```

Damit die entfernte Tabelle so benutzt werden kann als wäre sie auf diesem Server wird noch ein SYNONYM erstellt:

```
create synonym filme for filme@orion.helios.fhnw.ch;
```

Wir brauchen einen entsprechenden Link auch von der anderen Seite her.

2.2 Entwurf der Trigger

In den folgenden Fällen muss unser Trigger eingreifen:

1. Löschen eines Datensatzes aus Filme, auf den sich noch mindestens ein Datensatz aus DVDKopien bezieht. Wird vor dem Event DELETE auf der Tabelle Filme für jede Zeile angewendet. Sollte sich noch ein Datensatz aus DVDKopien auf den zu löschenden Film-Eintrag beziehen, so wird eine Exception geworfen.
2. Ändern des Primärschlüssels (katalognr) eines Datensatzes aus Filme, auf den sich noch mindestens ein Datensatz aus DVDKopien bezieht. Der Trigger muss vor dem Event UPDATE auf der Spalte katalognr wiederum für jede betroffene Zeile angewendet werden. Auch hier soll eine Exception geworfen werden, falls einer der alten Werte in DVDKopien benutzt wurde.
3. Einfügen eines Datensatzes in DVDKopien - die angegebene katalognr muss in Filme existieren. Um dies zu prüfen verwenden wir einen BEFORE INSERT-Trigger, welcher wiederum pro Zeile angestossen wird.
4. Änderung von FK in DVDKopien, auch der neue Wert muss in Filme existieren. Dieser Fall wird mit einem Trigger geprüft, der vor dem Update auf der katalognr-Spalte der Tabelle DVDKopien gefeuert wird.

2.3 SQL Trigger

2.3.1 Insert in DVDKopien

```
CREATE OR REPLACE TRIGGER dvdkopien_insert
  BEFORE INSERT on DVDKopien
  FOR EACH ROW

  DECLARE
    katalognr_found_count NUMBER(2,0) := 0;

  BEGIN
    SELECT count(katalognr) INTO katalognr_found_count FROM filme WHERE katalognr =
        :new.katalognr;

    dbms_output.put('count(katalognr) in filme: ');
    dbms_output.put_line(katalognr_found_count);

    if katalognr_found_count < 1 then
      raise_application_error(-20000, 'film mit angegebener katalognr existiert
          nicht');
    end if;
```

```
17
18    END;
```

2.3.2 Update in DVDKopien

```
1   CREATE OR REPLACE TRIGGER dvdkopien_update_katalognr
2     BEFORE UPDATE OF katalognr ON DVDKopien
3     FOR EACH ROW
4
5     DECLARE
6       katalognr_found_count NUMBER(2,0) := 0;
7
8     BEGIN
9       SELECT count(katalognr) INTO katalognr_found_count FROM filme WHERE katalognr =
            :new.katalognr;
10
11      dbms_output.put('count(katalognr) in filme: ');
12      dbms_output.put_line(katalognr_found_count);
13
14      if katalognr_found_count < 1 then
15        raise_application_error(-20000, 'film mit angegebener katalognr existiert
              nicht');
16      end if;
17
18    END;
```

2.3.3 Delete aus Filme

```
1   CREATE OR REPLACE TRIGGER filme_delete
2     BEFORE DELETE on filme
3     FOR EACH ROW
4
5     DECLARE
6       katalognr_found_count NUMBER(6,0) := 0;
7
8     BEGIN
9       SELECT count(katalognr) INTO katalognr_found_count FROM dvdkopien WHERE katalognr
            = :old.katalognr;
10
11      dbms_output.put('count(katalognr) in dvdkopien: ');
12      dbms_output.put_line(katalognr_found_count);
13
14      if katalognr_found_count > 0 then
15        raise_application_error(-20000, 'es gibt noch dvdkopien dieses filmes');
16      end if;
17
18    END;
```

2.3.4 Update in Filme

```
1   CREATE OR REPLACE TRIGGER filme_update_katalognr
2     BEFORE UPDATE OF katalognr ON filme
3     FOR EACH ROW
4
5     DECLARE
6       katalognr_found_count NUMBER(6,0) := 0;
7
8     BEGIN
9       SELECT count(katalognr) INTO katalognr_found_count FROM dvdkopien WHERE katalognr
            = :old.katalognr;
10
11      dbms_output.put('count(katalognr) in dvdkopien: ');
12      dbms_output.put_line(katalognr_found_count);
13
14      if katalognr_found_count > 0 then
15        raise_application_error(-20000, 'es gibt noch dvdkopien dieses filmes');
```

```
16        end if;
17
18    END;
```

2.4 Tests