

mvdb's: 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(6)) 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. Damit wir uns nicht um den Primärschlüssel kümmern müssen, erstellen wir eine Sequenz und erhöhen diese bei jedem Eintrag in die Tabelle ausleihen_log um eins.

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(6) 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     new_log_id number(6,0);
7 BEGIN
8     if inserting then
9         manipulation := 'INS';
10    elsif deleting then
11        manipulation := 'DEL';
12    elsif updating then
13        manipulation := 'UPD';
14    else
15        manipulation := 'ERR';
16    end if;
17
18    SELECT seq_ausleih_log_id.nextval INTO new_log_id FROM dual;
19    INSERT INTO ausleihen_log (log_id, change_user, change_date, change_type,
20        mnr_old, mnr_new, dvdnr_old, dvdnr_new, datum_old, datum_new, rueckgabe_old,
21        rueckgabe_new)
22    VALUES (new_log_id, user, sysdate, manipulation, :old.mnr, :new.mnr, :old.dvdnr,
23        :new.dvdnr, :old.datum, :new.datum, :old.rueckgabe, :new.rueckgabe);
24 END;
```

1.5 Tests

Um zu überprüfen, ob der von uns erstellte Trigger `ausleihen_logger` richtig funktioniert und alle verlangten Informationen in der Tabelle `ausleihen_log` eingetragen sind, führen wir für jede potentielle Änderung einen Tests durch.

1.5.1 Test INSERT

Mit folgendem SQL-Befehl fügen wir Daten in die Tabelle `Ausleihen` ein:

```
1 insert into ausleihen(mnr,dvdnr,datum,rueckgabe) VALUES
  ('M005',468123,'01.01.2000',null);
```

In der Tabelle `Ausleihen` sind nun die eingefügten Daten in der vierten Zeile ersichtlich:

	MNR	DVDNR	DATUM	RUECKGABE
2				
3	M001	468123	01.01.99	
4	M005	468123	01.01.00	
5	M002	158234	19.07.07	21.07.07
6	M004	158234	02.08.07	04.08.07
7	M003	269260	05.01.08	
8	M003	199004	05.01.08	
9	M001	310094	22.11.07	27.11.07
10	M001	468123	19.01.08	
11	M002	183669	30.11.07	01.12.07
12	M004	183669	27.12.07	03.01.08
13	M005	183669	15.01.08	
14	M001	183669	01.01.99	

In der Tabelle `Ausleihen_log` wurden die Daten mit der `LOG_ID` 22 eingefügt (Zeile 3):

	LOG_ID	CHANGEUSER	CHANGE_DATE		CHANGE_TYPE
2					
3	22	MVDBS10	24.02.13	15:19:50,000000000	INS

	MNR_OLD	MNR_NEW	DVDNR_OLD	DVDNR_NEW	DATUM_OLD	DATUM_NEW
2						
3		M005		468123		01.01.00

	RUECKGABE_OLD	RUECKGABE_NEW
2		
3		

Da es sich hier um ein `INSERT` handelt, sind alle `_old` Felder leer. Der Trigger hat also funktioniert.

1.5.2 Test UPDATE

Mit folgendem SQL-Befehl ändern wir Daten in der Tabelle `Ausleihen`:

```
1 update ausleihen
2 set mnr = 'M004'
3 where mnr = 'M005' and datum like '01.01.00';
```

In der Tabelle `Ausleihen` sind nun die geänderten Daten in der vierten Zeile ersichtlich:

	MNR	DVDNR	DATUM	RUECKGABE
1				
2				
3	M001	468123	01.01.99	
4	M004	468123	01.01.00	
5	M002	158234	19.07.07	21.07.07
6	M004	158234	02.08.07	04.08.07
7	M003	269260	05.01.08	
8	M003	199004	05.01.08	
9	M001	310094	22.11.07	27.11.07
10	M001	468123	19.01.08	
11	M002	183669	30.11.07	01.12.07
12	M004	183669	27.12.07	03.01.08
13	M005	183669	15.01.08	
14	M001	183669	01.01.99	

In der Tabelle `ausleihen_log` wurden die Daten mit der LOG_ID 23 eingefügt (Zeile 4):

	LOG_ID	CHANGEUSER	CHANGE_DATE	CHANGETYPE
1				
2				
3	22	MVDBS10	24.02.13 15:19:50,000000000	INS
4	23	MVDBS10	24.02.13 15:34:44,000000000	UPD

	MNR_OLD	MNR_NEW	DVDNR_OLD	DVDNR_NEW	DATUM_OLD	DATUM_NEW
1						
2						
3		M005		468123		01.01.00
4	M005	M004	468123	468123	01.01.00	01.01.00

	RUECKGABE_OLD	RUECKGABE_NEW
1		
2		
3		.
4		.

Nun sieht man, dass die `_old` Felder ebenfalls ausgefüllt sind mit den Werten vor dem Update. Der Trigger hat also funktioniert.

1.5.3 Test DELETE

Mit folgendem SQL-Befehl löschen wir Daten in der Tabelle `Ausleihen`:

```
1 delete from ausleihen
2 where mnr = 'M004' and datum like '01.01.00';
```

In der Tabelle `ausleihen` sind nun die gelöschten Daten nicht mehr ersichtlich:

	MNR	DVDNR	DATUM	RUECKGABE
1				
2				
3	M001	468123	01.01.99	
4	M002	158234	19.07.07	21.07.07
5	M004	158234	02.08.07	04.08.07
6	M003	269260	05.01.08	
7	M003	199004	05.01.08	
8	M001	310094	22.11.07	27.11.07
9	M001	468123	19.01.08	
10	M002	183669	30.11.07	01.12.07
11	M004	183669	27.12.07	03.01.08
12	M005	183669	15.01.08	
13	M001	183669	01.01.99	

In der Tabelle `Ausleihen_log` wurden die Daten mit der LOG_ID 24 eingefügt (Zeile 5):

	LOG_ID	CHANGEUSER	CHANGE_DATE	CHANGETYPE
1				
2				
3	22	MVDBS10	24.02.13 15:19:50,000000000	INS
4	23	MVDBS10	24.02.13 15:34:44,000000000	UPD
5	24	MVDBS10	24.02.13 15:41:14,000000000	DEL

1	MNR.OLD	MNR.NEW	DVDNR.OLD	DVDNR.NEW	DATUM.OLD	DATUM.NEW
2						
3		M005		468123		01.01.00
4	M005	M004	468123	468123	01.01.00	01.01.00
5	M005		468123		01.01.00	

1	RUECKGABE.OLD	RUECKGABE.NEW
2		
3		.
4		.
5		.

Da es sich nun um ein DELETE-Statement handelt, gibt es keine neuen Werte, daher sind die _new Felder leer. Der Trigger hat also funktioniert.

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:

```

1 create database link orion.helios.fhnw.ch
2 connect to mvdb10 identified by mvdb10
3 using 'orion'
```

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

```

1 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

```

1 CREATE OR REPLACE TRIGGER dvdkopien_insert
2   BEFORE INSERT 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;
```

2.3.2 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

Um zu überprüfen ob die Trigger richtig funktionieren, erstellen wir für jeden Fall zwei Tests, wobei einer der Tests das Erlaubte durchführt und der andere eine Exception verursacht.

2.4.1 Test Insert in DVD Kopien

2.4.2 Test Update in DVD Kopien

2.4.3 Test Delete in Filme

2.4.4 Test Update in DVD Kopien