

# mvdb's: Übung Trigger

Tobias Lerch, Yanick Eberle, Pascal Schwarz

26. 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)
2 SELECT 'M005',468123,'01.01.2000',null FROM dual
3 UNION ALL SELECT 'M005',183669,'01.01.2000',null FROM dual;
```

In der Tabelle `ausleihen` sind nun die eingefügten Daten in der 4. und 5. Zeile ersichtlich:

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

In der Tabelle `ausleihen_log` wurden die Daten mit der `LOG_ID` 22 und 23 eingefügt (Zeile 3 und 4):

	LOG_ID	CHANGEUSER	CHANGE.DATE	CHANGE.TYPE
2				
3	22	MVDBS10	24.02.13 15:19:50,000000000	INS
4	23	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
4		M005		183669		01.01.00

  

	RUECKGABE.OLD	RUECKGABE.NEW
2		
3		.
4		.

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
4 datum LIKE '01.01.00';
```

In der Tabelle ausleihen sind nun die geänderten Daten in der 4. und 5. Zeile ersichtlich:

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

In der Tabelle ausleihen\_log wurden die Daten mit der LOG\_ID 24 und 25 eingefügt (Zeile 5 und 6):

	LOG_ID	CHANGEUSER	CHANGE_DATE		CHANGE_TYPE
1					
2					
3	22	MVDBS10	24.02.13	15:19:50,000000000	INS
4	23	MVDBS10	24.02.13	15:19:50,000000000	INS
5	24	MVDBS10	24.02.13	15:34:44,000000000	UPD
6	25	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		183669		01.01.00
5	M005	M004	468123	468123	01.01.00	01.01.00
6	M005	M004	183669	183669	01.01.00	01.01.00

  

	RUECKGABE_OLD	RUECKGABE_NEW
1		
2		
3		.
4		.
5		.
6		.

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
3      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` 26 und 27 eingefügt (Zeile 7 und 8):

LOG_ID	CHANGEUSER	CHANGE_DATE	CHANGE_TYPE
22	MVDBS10	24.02.13 15:19:50,000000000	INS
23	MVDBS10	24.02.13 15:19:50,000000000	INS
24	MVDBS10	24.02.13 15:34:44,000000000	UPD
25	MVDBS10	24.02.13 15:34:44,000000000	UPD
26	MVDBS10	24.02.13 15:41:14,000000000	DEL
27	MVDBS10	24.02.13 15:41:14,000000000	DEL

  

MNR_OLD	MNR_NEW	DVDNR_OLD	DVDNR_NEW	DATUM_OLD	DATUM_NEW
	M005		468123		01.01.00
	M005		183669		01.01.00
M005	M004	468123	468123	01.01.00	01.01.00
M005	M004	183669	183669	01.01.00	01.01.00
M004		468123		01.01.00	
M004		183669		01.01.00	

  

RUECKGABE_OLD	RUECKGABE_NEW

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:

```

1 CREATE DATABASE LINK telesto.janus.fhnw.ch
2 CONNECT to mvds10 identified by mvds10
3 USING 'telesto';

```

Auch hier erstellen wir ein SYNONYM

```

1 CREATE SYNONYM dvdkopien FOR dvdkopien@telesto.janus.fhnw.ch;

```

## 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 =
10         :new.katalognr;
11
12     dbms_output.put('count(katalognr) in filme: ');
13     dbms_output.put_line(katalognr_found_count);
14
15     if katalognr_found_count < 1 then
16         raise_application_error(-20000, 'film mit angegebener katalognr existiert
17         nicht ');
18     end if;
19 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 =
10      :new.katalognr;
11
12     dbms_output.put('count(katalognr) in filme: ');
13     dbms_output.put_line(katalognr_found_count);
14
15     if katalognr_found_count < 1 then
16       raise_application_error(-20000, 'film mit angegebener katalognr existiert
17       nicht');
18     end if;
19
20   END;
```

### 2.3.3 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
10      = :old.katalognr;
11
12     dbms_output.put('count(katalognr) in dvdkopien: ');
13     dbms_output.put_line(katalognr_found_count);
14
15     if katalognr_found_count > 0 then
16       raise_application_error(-20000, 'es gibt noch dvdkopien dieses filmes');
17     end if;
18
19   END;
```

### 2.3.4 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
10      = :old.katalognr;
11
12     dbms_output.put('count(katalognr) in dvdkopien: ');
13     dbms_output.put_line(katalognr_found_count);
14
15     if katalognr_found_count > 0 then
16       raise_application_error(-20000, 'es gibt noch dvdkopien dieses filmes');
17     end if;
18
19   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

Wir fügen in die Tabelle DVDKopien Daten mit einem vorhandenen FK katalognr ein:

```
1 INSERT INTO dvdkopien (dvdnr,katalognr,fnr)
2 SELECT 10001,2468,'F1' FROM dual
3 UNION ALL SELECT 10002,2468,'F2' FROM dual;
```

Es wird keine Exception geworfen. In der Tabelle DVDKopien sind die eingefügten Daten nun ersichtlich (Zeile 3 und 4):

	DVDNR	KATALOGNR	FNR
3	10001	2468	F1
4	10002	2468	F2
5	199004	2028	F1
6	468123	2028	F2
7	269260	1245	F1
8	183669	1245	F3
9	329270	1245	F4
10	178643	2239	F2
11	389653	2239	F4
12	158234	1062	F3
13	139558	2468	F2
14	469118	2468	F2
15	310094	1062	F2

Für den zweiten Fall versuchen wir Daten einzufügen mit einem ungültigen FK katalognr:

```
1 INSERT INTO dvdkopien (dvdnr,katalognr,fnr)
2 SELECT 10003,2468,'F1' FROM dual
3 UNION ALL SELECT 10004,9999,'F2' FROM dual;
```

Es wird eine Exception geworfen von unserem Trigger der die Meldung 'film mit angegebener katalognr existiert nicht' ausgibt.

```
1 Fehler beim Start in Zeile 1 in Befehl:
2 INSERT INTO dvdkopien (dvdnr,katalognr,fnr)
3 SELECT 10003,2468,'F1' FROM dual
4 UNION ALL SELECT 10004,9999,'F2' FROM dual
5 Fehlerbericht:
6 SQL-Fehler: ORA-20000: film mit angegebener katalognr existiert nicht
7 ORA-06512: in "MVDBS10.DVDKOPIEN.INSERT", Zeile 11
8 ORA-04088: Fehler bei der Ausfuehrung von Trigger 'MVDBS10.DVDKOPIEN.INSERT'
9 20000. 00000 - "%s"
10 *Cause:      The stored procedure 'raise_application_error'
11              was called which causes this error to be generated.
12 *Action:     Correct the problem as described in the error message or contact
13              the application administrator or DBA for more information.
```

Es wurden keine Daten in die Tabelle DVDKopien eingefügt:

	DVDNR	KATALOGNR	FNR
3	10001	2468	F1
4	10002	2468	F2
5	199004	2028	F1
6	468123	2028	F2



7	269260	1245	F1
8	183669	1245	F3
9	329270	1245	F4
10	178643	2239	F2
11	389653	2239	F4
12	158234	1062	F3
13	139558	2468	F2
14	469118	2468	F2
15	310094	1062	F2

Wir haben ein Multi-Row-Insert mit zwei Zeilen durchgeführt, bei welchem eine Zeile korrekt war und die andere einen ungültigen FK katalognr enthalten hat. Da es sich um eine Transaktion handelt, darf alles oder gar nichts eingefügt werden. Daher ist es korrekt, dass das ganze Statement nicht ausgeführt wurde und somit keine Daten in die Tabelle DVDKopien eingefügt wurden.

### 2.4.2 Test Update in DVD Kopien

Nun werden wir einen bestehenden Eintrag in der Tabelle DVDKopien anpassen und die katalognr mit einem anderen gültigen Wert ersetzen:

```
1 UPDATE dvdkopien
2 SET katalognr = 2028
3 WHERE dvdnr LIKE '1000%';
```

Es wird keine Exception geworfen. In der Tabelle DVDKopien sind die geänderten Daten nun ersichtlich (Zeile 3 und 4):

	DVDNR	KATALOGNR	FNR
3	10001	2028	F1
4	10002	2028	F2
5	199004	2028	F1
6	468123	2028	F2
7	269260	1245	F1
8	183669	1245	F3
9	329270	1245	F4
10	178643	2239	F2
11	389653	2239	F4
12	158234	1062	F3
13	139558	2468	F2
14	469118	2468	F2
15	310094	1062	F2

Für den zweiten Fall versuchen wir den FK katalognr durch einen ungültigen FK zu ersetzen:

```
1 UPDATE dvdkopien
2 SET katalognr = 9999
3 WHERE dvdnr LIKE '1000%';
```

Es wird eine Exception geworfen von unserem Trigger der die Meldung 'film mit angegebener katalognr existiert nicht' ausgibt.

```
1 Fehler beim Start in Zeile 1 in Befehl:
2 UPDATE dvdkopien
3 SET katalognr = 9999
4 WHERE dvdnr LIKE '1000%'
5 Fehlerbericht:
6 SQL-Fehler: ORA-20000: film mit angegebener katalognr existiert nicht
7 ORA-06512: in "MVDBS10.DVDKOPIEN.UPDATEKATALOGNR", Zeile 11
```

```

8  ORA-04088: Fehler bei der Ausfuehrung von Trigger 'MVDBS10.DVDKOPIEN_UPDATEKATALOGNR'
9  20000. 00000 - "%s"
10 *Cause:      The stored procedure 'raise_application_error'
11              was called which causes this error to be generated.
12 *Action:      Correct the problem as described in the error message or contact
13              the application administrator or DBA for more information.

```

Es wurden keine Daten in die Tabelle DVDKopien eingefügt:

	DVDNR	KATALOGNR	FNR
1			
2			
3	10001	2028	F1
4	10002	2028	F2
5	199004	2028	F1
6	468123	2028	F2
7	269260	1245	F1
8	183669	1245	F3
9	329270	1245	F4
10	178643	2239	F2
11	389653	2239	F4
12	158234	1062	F3
13	139558	2468	F2
14	469118	2468	F2
15	310094	1062	F2

### 2.4.3 Test Update in Filme

Wir ändern die katalognr eines bestehenden Filmes zu welchem keine DVDKopie existiert, auf welchen also keine Referenz vorhanden ist:

```

1  UPDATE filme
2  SET katalognr = 1111
3  WHERE katalognr = 1672;

```

Es wird keine Exception geworfen. In der Tabelle Filme sind die geänderten Daten nun ersichtlich (Zeile 8):

	KATALOGNR	TITEL	MINDESTALTER	GEBUEHR
1				
2				
3	2028	Casablanca	9	8.5
4	1245	Ocean's Eleven	12	9.5
5	2239	A Space Odyssey	12	7.5
6	1062	Pulp Fiction	16	8.5
7	2588	The Pelican Brief	12	8.5
8	1111	Erin Brockovich	9	8.9
9	2468	Ratatouille	6	7.5

Für den zweiten Fall versuchen wir die katalognr eines bestehenden Filmes zu ändern, zu welchem eine DVDKopie existiert, auf welchen also eine Referenz vorhanden ist:

```

1  UPDATE filme
2  SET katalognr = 2222
3  WHERE katalognr = 2468 OR
4        katalognr = 1111;

```

Es wird eine Exception geworfen von unserem Trigger der die Meldung 'es gibt noch dvdkopien dieses filmes' ausgibt.

```

1  Fehler beim Start in Zeile 1 in Befehl:
2  UPDATE filme
3  SET katalognr = 2222
4  WHERE katalognr = 2468 OR
5        katalognr = 1111

```

```

6 Fehlerbericht:
7 SQL-Fehler: ORA-20000: es gibt noch dvdkopien dieses filmes
8 ORA-06512: in "MVDBS10.FILME.UPDATE.KATALOGNR", Zeile 11
9 ORA-04088: Fehler bei der Ausfuehrung von Trigger 'MVDBS10.FILME.UPDATE.KATALOGNR'
10 ORA-02063: vorherige 3 lines von ORION.HELIOS.FHNW.CH
11 20000. 00000 - "%s"
12 *Cause:      The stored procedure 'raise_application_error'
13              was called which causes this error to be generated.
14 *Action:     Correct the problem as described in the error message or contact
15              the application administrator or DBA for more information.

```

Es wurden keine Daten in die Tabelle Filme geändert:

	KATALOGNR	TITEL	MINDESTALTER	GEBUEHR
1				
2				
3	2028	Casablanca	9	8.5
4	1245	Ocean's Eleven	12	9.5
5	2239	A Space Odyssee	12	7.5
6	1062	Pulp Fiction	16	8.5
7	2588	The Pelican Brief	12	8.5
8	1111	Erin Brockovich	9	8.9
9	2468	Ratatouille	6	7.5

Auch hier wird ein Update auf mehrer Datensätze durchgeführt. Da es sich um eine Transaktion handelt, wird keiner der beiden Datensätze angepasst, obwohl das Update beim Datensatz '1111' nicht gegen die referentielle Integrität verstösst.

#### 2.4.4 Test Delete in Filme

Wir löschen einen Film, auf welchen keine Referenz verweist:

```

1 DELETE FROM filme
2 WHERE katalognr = 2588;

```

Es wird keine Exception geworfen. In der Tabelle Filme ist der Datensatz mit der katalognr 2588 nicht mehr vorhanden:

	KATALOGNR	TITEL	MINDESTALTER	GEBUEHR
1				
2				
3	2028	Casablanca	9	8.5
4	1245	Ocean's Eleven	12	9.5
5	2239	A Space Odyssee	12	7.5
6	1062	Pulp Fiction	16	8.5
7	1111	Erin Brockovich	9	8.9
8	2468	Ratatouille	6	7.5

Für den zweiten Fall versuchen wir einen Film zu löschen, zu welchem DVDKopien existieren, auf welchen also eine Referenz verweist:

```

1 DELETE FROM filme
2 WHERE katalognr = 2468;

```

Es wird eine Exception geworfen von unserem Trigger der die Meldung 'es gibt noch dvdkopien dieses filmes' ausgibt.

```

1 Fehler beim Start in Zeile 1 in Befehl:
2 DELETE FROM filme
3 WHERE katalognr = 2468
4 Fehlerbericht:
5 SQL-Fehler: ORA-20000: es gibt noch dvdkopien dieses filmes
6 ORA-06512: in "MVDBS10.FILME.DELETE", Zeile 11
7 ORA-04088: Fehler bei der Ausfuehrung von Trigger 'MVDBS10.FILME.DELETE'

```

```

8  ORA-02063: vorherige 3 lines von ORION.HELIOS.FHNW.CH
9  20000. 00000 - "%s"
10 *Cause:      The stored procedure 'raise_application_error'
11              was called which causes this error to be generated.
12 *Action:     Correct the problem as described in the error message or contact
13              the application administrator or DBA for more information.

```

Es wurden keine Daten in der Tabelle Filme gelöscht:

	KATALOGNR	TITEL	MINDESTALTER	GEBUEHR
3	2028	Casablanca	9	8.5
4	1245	Ocean's Eleven	12	9.5
5	2239	A Space Odyssey	12	7.5
6	1062	Pulp Fiction	16	8.5
7	1111	Erin Brockovich	9	8.9
8	2468	Ratatouille	6	7.5