DES3UE WS 2024 Übung 5

Abgabetermin: siehe e-Learning, Abgabeform elektronisch

☐ **DES3UEG1: Glock** **Name** _____ **Aufwand in h** _____
☐ **DES3UEG2: Werth** **Punkte** _____ **Kurzzeichen Tutor** _____

Ziel dieser Übung ist die Vertiefung des Trigger-Konzepts und die praktische Anwendung der unterschiedlichen Typen.

1. Trigger (Sakila) (4+3+2+3 = 12 Punkte)

Die Zeilen in der Tabelle PAYMENT speichern den Bezahlvorgang nachdem ein Film zurückgegeben wurde. Erstellen Sie in PL/SQL ein Programm, das sicherstellt, dass bei INSERT- und UPDATE-Operationen auf diese Tabelle der bezahlte Betrag nicht die vorgesehene Rate eines Films überschreitet. Beachten Sie NULL-Werte!

1. Erstellen Sie eine Prozedur mit dem Namen CHECK_AMOUNT mit folgenden Parametern:
   a. die Film-Id
   b. den bezahlten Betrag (wird eingelesen **und** ggf. geändert zurückgegeben)
   c. die Anzahl der verliehenen Tage

   Verwenden Sie die Film-Id, um die Verleihrate (rental_rate) für den angegebenen Film zu finden. Vergleichen Sie den bezahlten Betrag mit dem tatsächlich zu bezahlenden Betrag (Verleihrate pro verliehenem Tag, mal Anzahl der verliehenen Tage). Wenn der bezahlte Betrag höher ist als die insgesamt zu bezahlende Verleihrate, soll folgende Fehlermeldung **ausgelöst** werden:

   „Invalid amount for film <film>, maximum for <days> is <rate * days>."

   Ersetzen Sie die Elemente in der Mitteilung in der benutzerdefinierten Fehlermeldung durch die entsprechenden Werte.
   Ist der angegebene Betrag kleiner als 0 wird der Betrag auf 0 korrigiert und zurückgegeben. Testen Sie die Prozedur zumindest, indem Sie für den Film 200 überprüfen, ob ein Betrag von 20 für 3 Tage möglich ist. Rufen Sie dazu die Prozedur in einem anonymen Block auf.

   ***Hinweis*:**
   Die Prozedur sollte keinen Exception Handler aufweisen, da in der nachfolgenden Aufgabe die Prozedur von einem Trigger aufgerufen wird und ein Behandeln einer Exception den Trigger erfolgreich beenden würde und somit die mit dem Trigger verbundene Datenbankoperation durchgeführt wird. D.h. es sollte bspw. kein Handler für ... WHEN NO_DATA_FOUND THEN ... definiert sein, wenn im SELECT kein Satz gefunden wird, da die film_id nicht vorhanden ist. Grundsätzlich würde dieser Fehler schon vom deklarativen Foreign Key Constraint behandelt werden.

2. Erstellen Sie für die Tabelle PAYMENT einen Trigger mit dem Namen CHECK_AMOUNT_TRG, der bei einer INSERT- oder UPDATE-Operation in einer Zeile ausgelöst wird. Der Trigger soll die Prozedur CHECK_AMOUNT aufrufen, um die in der Aufgabe 1 definierte Regel auszuführen. Der Trigger soll die Film-Id und die Verleihdauer ermitteln (aus den Tabellen RENTAL und INVENTORY) und gemeinsam mit dem Betrag an die Prozedur übergeben. Stellen Sie sicher, dass der Trigger im Falle von UPDATE nur bei relevanten Attributen feuert.

3. Testen Sie CHECK_AMOUNT_TRG anhand der folgenden Fälle und notieren Sie, was passiert: Aktualisieren Sie die Tabelle payment und setzen Sie den Betrag für die Verleihvorgänge 6500 (payment_id) auf 25 (amount), 3000 auf 1 und 1200 auf -10. Überprüfen Sie Ihre Änderungen und nehmen Sie diese dann wieder zurück (ROLLBACK).

4. Erweitern Sie die Tabelle PAYMENT und fügen Sie ein Logging-Feld hinzu (user_modified). Erstellen Sie einen Trigger LOG_PAYMENT, der die beiden Felder last_update und user_modified bei einer Änderung des Betrags aktualisiert. Testen Sie den Trigger ausführlich (führen Sie nach den Tests ein ROLLBACK aus, um Ihre Datenbasis nicht zu verändern). Anschließend entfernen Sie die zusätzliche Spalte (user_modified) wieder aus Ihrer PAYMENT-Tabelle.

2. INSTEAD OF-Trigger (2+4+1 = 7 Punkte)

INSTEAD OF-Trigger werden ausschließlich für Sichten eingesetzt, um Daten zu ändern, bei denen eine DML-Anweisung für eine Sicht abgesetzt wird, die implizit nicht aktualisierbar ist. Diese Trigger werden INSTEAD OF-Trigger genannt, da der Datenbankserver im Gegensatz zu anderen Trigger-Typen nicht die auslösende Anweisung ausführt, sondern den Trigger auslöst. Mit diesem Trigger werden INSERT-, UPDATE- oder DELETE-Operationen direkt für die zu Grunde liegenden Basistabellen durchgeführt. Sie können INSERT-, UPDATE- oder DELETE-Anweisungen für eine Sicht erstellen, und der INSTEAD OF-Trigger arbeitet unsichtbar im Hintergrund und sorgt für die Ausführung der gewünschten Aktionen.

Führen Sie folgendes Skript aus, um die Basistabellen für die Aufgabe zu erstellen.

```
DROP TABLE new_rental;
DROP TABLE new_customer;

CREATE TABLE new_rental AS
SELECT *
FROM rental;
ALTER TABLE new_rental ADD CONSTRAINT new_rental_pk PRIMARY KEY  (rental_id);

CREATE TABLE new_customer (customer_id, store_id NULL, first_name, last_name,
email, address_id NULL, active, create_date, last_update) AS
SELECT *
FROM customer;

ALTER TABLE new_customer ADD CONSTRAINT new_customer_pk PRIMARY KEY
(customer_id);
```

Skript UE05_02.sql

1. Erstellen Sie eine Sicht premium_customer basierend auf den im obigen Skript erstellten Tabellen NEW_CUSTOMER und NEW_RENTAL mit folgenden Spalten: customer_id, first_name, last_name und numFilms (zählt die Anzahl der geliehenen Filme, d.h. Inventar). Es sollen nur Kunden in der View enthalten sein, die mehr als 30 Filme geliehen haben.

2. Legen Sie nun für die zuvor erstellte Sicht premium_customer einen INSTEAD OF-Trigger an, um DML-Operationen auf dieser Sicht zu kontrollieren. Folgende Funktionalitäten sind gefordert:
   a. Bei einem DELETE wird der Satz aus new_customer gelöscht und auch alle zugehörigen Verleihvorgänge aus new_rental.
   b. Bei einem INSERT wird ein neuer Kunde in new_customer angelegt und alle vorhandenen Daten ausgefüllt (inkl. Erstell- und Änderungsdatum). Wird kein Primärschlüssel beim INSERT angegeben, so stellen Sie einen automatisch bereit. Hierfür berechnen Sie vorher auf Basis der höchsten ID einen möglichen Schlüssel, den Sie verwenden.
   c. Bei einem UPDATE auf numFilms geben Sie eine entsprechende Fehlermeldung zurück, dass dieses berechnete Feld nicht aktualisiert werden kann. D.h. Sie lösen hierfür einen entsprechenden selbst definierten Fehler aus.
   d. Bei einem UPDATE auf customer_id geben Sie eine entsprechende Fehlermeldung zurück, dass ein Primärschlüssel nicht verändert werden darf.

e. Bei einem UPDATE auf die anderen Felder aktualisieren Sie diese in der Basistabelle new_customer.

3. Überprüfen Sie die implementierte Funktionalität des INSTEAD OF-Triggers mit mind. den folgenden Statements.

```
INSERT INTO premium_customer (customer_id, first_name, last_name)
       VALUES (3000, 'Mary', 'Poppins');
INSERT INTO premium_customer (first_name, last_name)
       VALUES ('Mary', 'Poppins');

SELECT *
FROM new_customer
WHERE LOWER(last_name) LIKE  'poppins';

DELETE FROM premium_customer
WHERE customer_id = 51;

SELECT *
FROM new_customer
WHERE customer_id = 51;

SELECT *
FROM new_rental
WHERE customer_id = 51;

UPDATE premium_customer
   SET numFilms = 40
 WHERE customer_id = 470;

UPDATE premium_customer
   SET customer_id = 40
 WHERE customer_id = 470;

UPDATE premium_customer
   SET first_name = 'John',
       last_name = 'Smith'
 WHERE customer_id = 470;

SELECT *
FROM new_customer
WHERE customer_id = 470;
```

3. Trigger für Systemereignisse (1+3+1 = 5 Punkte)

1. Erstellen Sie eine Tabelle USER_LOGGING mit den Feldern session_id, login_time, db_user, os_user, ip und host_name. Wählen Sie geeignete Datentypen.

2. Erstellen Sie für das Schema einen Systemtrigger, der pro Session beim Anmelden einen Datensatz einfügt. Verwenden Sie die Funktion SYS_CONTEXT mit den entsprechenden Parametern um die Session-ID, die IP-Adresse, den Betriebssystem-User und den Host-Namen (Bezeichnung des verbundenen Rechners) zu ermitteln. Schreiben Sie diese Werte gemeinsam mit dem angemeldeten Datenbank-User und eines aktuellen Zeitstempels in die Tabelle.

3. Melden Sie sich bei der Datenbank ab und wieder an. Wiederholen Sie diese Schritte (wenn möglich) auch von einem anderen Rechner aus. Zeigen Sie den Inhalt Ihrer Tabelle an.