

Aufgabe 1 (Anlegen und auslösen eines Triggers)

Erzeugen Sie einen Trigger und lösen Sie ihn aus. Sie können dafür die Beispiel-Sourcen im Übungsverzeichnis verwenden. Führen Sie ein SQL-Skript erst dann aus, wenn Sie den Inhalt verstanden und nachvollzogen haben. Überprüfen Sie nach dem Ausführen, ob die Ergebnisse Ihren Erwartungen entsprechen.

a) Erzeugen Sie die Tabellen „inventory“ und „pending_orders“ und tragen Sie einige Beispieldaten ein.

Tabellen erzeugen:

```
CREATE TABLE inventory (part_no NUMBER PRIMARY KEY, parts_on_hand NUMBER, reorder_point NUMBER, reorder_quantity NUMBER);
```

```
CREATE TABLE pending_orders (part_no NUMBER, order_quantity NUMBER, order_date DATE);
```

Testdaten erfassen:

```
INSERT INTO inventory (part_no, parts_on_hand, reorder_point, reorder_quantity) VALUES (1, 20, 15, 30);
```

b) Speichern Sie den Trigger „reorderTrigger“ in der Datenbank.

Trigger:

```
CREATE OR REPLACE TRIGGER reorderTrigger
AFTER UPDATE OF parts_on_hand ON inventory
FOR EACH ROW
WHEN (new.parts_on_hand < new.reorder_point)
DECLARE
    x NUMBER;
BEGIN
    SELECT COUNT(*) INTO x
    FROM pending_orders
    WHERE part_no = :new.part_no;
    IF x = 0 THEN
        INSERT INTO pending_orders
        VALUES (:new.part_no, :new.reorder_quantity, sysdate);
    END IF;
END reorderTrigger;
/
```

c) Überprüfen Sie den Inhalt der Tabelle „pending_orders“.

```
SELECT * FROM pending_orders;
```

Es gibt keinen Inhalt

d) Lösen Sie den Trigger aus.

```
UPDATE inventory SET parts_on_hand = 3 WHERE part_no = 1;
```

e) Überprüfen Sie, ob der Trigger wirklich ausgelöst wurde.

```
SELECT * FROM pending_orders;
```

ergibt jetzt:

```
PART_NO ORDER_QUANTITY ORDER_DA
-----
1          30 09.10.06
```

Aufgabe 2 (Schreiben eines Triggers)

Schreiben und Testen einen von Ihnen selbst erstellten Trigger.

a) Legen Sie die Tabelle „projekt“ mit Projektnummer (PRIMARY KEY), Budget und Abteilungsnummer an und füllen Sie diese mit Daten. Sie können das Skript auf dem Skripte-Server verwenden.

```
drop table projekt;
```

```
CREATE TABLE Projekt
```

```

( ProjNr    DECIMAL(4) NOT NULL,
  Budget    DECIMAL(7),
  AbtNr     DECIMAL(4) NOT NULL,
  PRIMARY KEY (ProjNr)
);

insert into Projekt values (1, 5000, 1);
insert into Projekt values (2, 1000, 1);
insert into Projekt values (3, 2200, 10);
insert into Projekt values (4, 100, 3);
insert into Projekt values (5, 500, 4);
insert into Projekt values (6, 1000, 1);
insert into Projekt values (7, 3000, 4);
insert into Projekt values (8, 1000, 8);
insert into Projekt values (9, 1000, 4);
insert into Projekt values (10, 1000, 7);
insert into Projekt values (23, 5500, 3);

commit;

```

b) Legen Sie eine abgeleitete Tabelle (mit redundanten Daten) „Budget_Uebersicht“ an, welche die Summe der Budgets je Abteilung enthält.

```
CREATE TABLE Budget_Uebersicht AS SELECT AbtNr, SUM(Budget) as Budget FROM Projekt GROUP BY AbtNr;
```

c) Schreiben Sie einen DB-Trigger, der die redundanten Summeninformationen in ihrer obigen Tabelle „Budget_Uebersicht“ bei Änderungen (UPDATE) in der Projekt-Tabelle fortschreibt.

```

CREATE OR REPLACE TRIGGER budgetupdate_Trigger
AFTER UPDATE OF Budget ON Projekt
BEGIN
DROP Tmp_Uebersicht
CREATE TABLE Tmp_Uebersicht AS SELECT AbtNr, SUM(Budget) as Budget FROM Projekt GROUP BY AbtNr

FOR EACH ROW
WHEN (SELECT Budget FROM Tmp_Uebersicht WHERE Tmp_Uebersicht.AbtNr = Budget_Uebersicht.AbtNr !=
Budget_Uebersicht.Budget)

BEGIN
UPDATE Budget_Uebersicht SET Budget_Uebersicht.Budget = Tmp_Uebersicht.Budget WHERE
Budget_Uebersicht.AbtNr = Tmp_Uebersicht.AbtNr;
END;
DROP Tmp_Uebersicht
END budgetupdate_Trigger;
/

```

d) Testen Sie den Trigger, indem Sie das Budget einiger Projekte verringern und ein neues Projekt einfügen.

```
UPDATE Projekt SET Budget = 4000 WHERE ProjNr = 1;
```

e) Vergleichen Sie zur Kontrolle mit der Sicht.