Trigger & Functions in PL/SQL

Before & After Trigger

```
CREATE OR REPLACE TRIGGER change_Code
BEFORE UPDATE OF Code ON Country
FOR EACH ROW
BEGIN
UPDATE Province
SET Country = :NEW.Code
WHERE Country = :OLD.Code;
END;
/
```

```
INSERT INTO Country (Name,Code,Population)
VALUES ('Lummerland', 'LU', 4);
SELECT * FROM Politics WHERE country='LU';
```

Mit diesem Trigger könnte folgendes die erwarteten Ergebnisse bringen:

```
UPDATE Country
SET Code = 'UK'
WHERE Code = 'GB';
SELECT * FROM Province WHERE Country='UK';
```

```
CREATE OR REPLACE TRIGGER change_Code
BEFORE UPDATE OF Code ON Country
FOR EACH ROW
BEGIN
UPDATE Province
SET Country = :NEW.Code
WHERE Country = :OLD.Code;
END;
/
```

Instead of - Trigger

```
CREATE OR REPLACE TRIGGER InsAllCountry
INSTEAD OF INSERT ON AllCountry
FOR EACH ROW
BEGIN
        INSERT INTO Country (Name,Code,Population,Area)
        VALUES (:NEW.Name, :NEW.Code,:NEW.Population, :NEW.Area);
        INSERT INTO Economy (Country,Inflation)
        VALUES (:NEW.Code, :NEW.Inflation);
        INSERT INTO Population
                (Country, Population_Growth,infant_mortality)
        VALUES (:NEW.Code, :NEW.Population_Growth,:NEW.infant_mortality);
END;
/
```

Sysdate in Constraints und Triggern verwenden

Sysdate in Constraints verwenden

ALTER TABLE tableName ADD CONSTRAINT contraintName check (myDate < SYSDATE)

Funktioniert leider so nicht …

- Es ist nicht möglich undeterministische Funktionen zu verwenden
- SYSDATA ist nicht deterministisch für Oracle, da es nicht das selbe Ergebnis bei jeder Anfrage geliefert wird

Warum macht das Sinn?

- Ein Constraint muss IMMER für ALLE Einträge gelten
- Kann das nicht gewährleistet werden, wenn myData > SYSDATE wäre
- Irgendwann würde SYSDATE myDate „einholen“ und dann wäre eine Constraintverletztung erzeugt worden.

Sysdate in Functions verwenden

```
 CREATE OR REPLACE FUNCTION Date_in_Past(pastDate IN tableName.dateColumnName %
ColumnType
        return varchar2 deterministic
Is
Beginn
        return CASE WHEN pastDate >= SYSDATE THEN ‘N‘ ELSE ‘Y‘ end;
/
```

In diesem Fall ist Oracle gut gläubig und nimmt eine als „determinitic“ deklarierte Function auch als solche hin, egal was darin passiert.

Function an eine Table binden

ALTER TABLE tableName ADD (Date_in_Past_ing as
(cast(Date_in_Past(myDate) as VARCHAR2(1)))
/

- Hier wird eine neue Spalte definiert, die eine Function als Inhalt hat.
- Da die Function als „deterministic“ gilt kann sie hier verwendet werden
- Das fühlt sich nicht wirklich sauber an, ist aber wohl die „sauberste“ Lösung, wenn man es mit Constraints machen möchte.

Leider eher eine Saubär als eine saubere Lösung

ALTER TABLE tableName ADD CONSTRAINT constraintName check
(Date_in_Past_ing = ‘Y‘)
/

Trigger erstellen mit Sysdate in der Bedingung

```
Create Trigger myTrigger
        before insert or update on myTableName
        for each row
        beginn
                if inserting or updating('myDate')
                then
                        if new myDate >= SYSDATE
                        then
                                raise_application_error(ErrorNumber,'mydate ist wrong')
                        end if
                end if
        end;
/
```

Innerhalb eines Trigger kann Sysdate verwendet werden. Das scheint die einfachere Lösung ist aber Laufzeittechnisch die schlechtere Lösung

Zeitgesteuerte Trigger aufrufe

Einmalige zeitgesteuerte Jobs

```
create table jobtest (x DATE);
begin
        DBMS_SCHEDULER.CREATE_JOB
        (job_name => 'job1',
        job_type => 'PLSQL_BLOCK',
        job_action => 'begin insert into jobtest values (SYSDATE); end;',
        start_date => SYSDATE+1/1440,
        enabled => TRUE);
end;
/
```

Schreibt in einer Minute die aktuelle Zeit in die Tabelle JobTest

enabled: TRUE aktiviert sofort, FALSE hält einen Job deaktiviert

```
execute DBMS_SCHEDULER.ENABLE('job1');
execute DBMS_SCHEDULER.DISABLE('job1');
```

manuell aufrufen bzw löschen:

```
execute DBMS_SCHEDULER.RUN_JOB('job1');
execute DBMS_SCHEDULER.DROP_JOB('job1');
```

Wiederholende zeitgesteuerte Jobs

```
begin
DBMS_SCHEDULER.DROP_JOB('job2');
DBMS_SCHEDULER.CREATE_JOB
        (job_name => 'job2',
        job_type => 'PLSQL_BLOCK',
        job_action => 'begin
                      update country set population = population + 1
                      where code=''CN'';
                      end;',
        auto_drop => FALSE,
        repeat_interval => 'FREQ = MINUTELY; INTERVAL = 2;
        BYSECOND = 5, 18, 31, 45, 51',
        start_date => SYSDATE+1/2880, -- after 30 secs
        end_date => SYSDATE+11/1440, -- after 11 minutes
        enabled => TRUE);
end;
/
```

Erhöht alle zwei Minuten zu jeweils 5 angegeben Sekundenzeitpunkten die Bevölkerung um 1

Ein Wort der Warnung zu Triggern

Laufzeit	• Trigger sind in der Regel langsamer als Constraints in der Laufzeit
Multiuser	• Falls mehrere User Trigger implementieren können kann es bei verschieden User zu unterschiedlichen Ergebnissen bei der selben Eingabe kommt
Nachvollziehbarkeit	• Es gibt keine „einfache Übersicht“ über Trigger und ihr Verhalten • Es kann nicht einfach nachvollzogen werden, wann ein Trigger aktiv ist oder welche Trigger gerade reagiert haben