Triggers (chap 8 in Database Systems book)

Triggers are important parts of database systems; they define actions the database should take when some specific database related (UPDATE, INSERT,DELETE) events occur. Triggers are similar to procedures in that they are named PL/SQL blocks with declarative, executable, and exception handling sections. However, triggers are **executed implicitly** whenever a triggering event happens and a trigger **does NOT** accept arguments.

Used for:

- Maintaining/supplementing complex integrity constraints
- Enforcing complex business rules
- **Auditing** information by recording changes to tables
- Signalling other programs that actions need to take place when changes are made to a table

Syntax:

CREATE [OR REPLACE] TRIGGER *trigger_name*
BEFORE | AFTER | INSTEAD OF
INSERT | UPDATE | DELETE | INSERT OR UPDATE | *triggering_event*
ON *table_name*
[FOR EACH ROW | STATEMENT [WHEN *trigger_condition*]]
trigger_body;

Oracle has 14 types of Triggers:
INSERT
UPDATE
DELETE
INSTEAD OF

- Each trigger can be fired BEFORE or AFTER or INSTEAD OF (introduced in Oracle 8) the event that triggers it.

- Each Trigger can be fired once for each ROW (row-level) affected by the triggering statement,. or once for each STATEMENT (statement-level).

NOTE: to refer to old and new values inside the trigger, ORACLE uses two prefixes: **:new** and **:old**.
Prefixes within the trigger:
:new - refers to the newly updated column value
:old - refers to the original value of the column

Use the following tables from a Veterinary database

PETS_COUNTS (BREED_NAME, BREED_COUNT, LAST_UPDATE_DATETIME)

PETS (PET_ID, PET_NAME, BREED_NAME, YEAR_OF_BIRTH)

Write a CREATE Statement for a **trigger** that will update table PETS_COUNTS (change the count and set the last_update_datetime to SYSDATE) , whenever a pet is **added** or **deleted** from the PETS table. **Note: the PETS_COUNTS table has all possible breeds. ☺**

```
CREATE OR REPLACE TRIGGER Pets_Idr AFTER
  INSERT OR DELETE
  ON E1_Pets
  FOR EACH Row

BEGIN IF inserting THEN
  UPDATE E1_Pets_Counts
  SET Last_Update_Date      = Sysdate,
    Breed_Count             = Breed_Count + 1
  WHERE Upper (Breed_Name)  = Upper( :New.Breed_Name);
ELSE
  UPDATE E1_Pets_Counts
  SET Last_Update_Date      = Sysdate,
    Breed_Count             = Breed_Count - 1
  WHERE Upper (Breed_Name)  = Upper( :Old.Breed_Name);
END IF;
END;
```

More challenging trigger: Write a trigger to update the counts as in the above example, but instead of the assumptions about pre-existing data for all the breeds, add a new row to the PETS_COUNTS whenever a pet with a new breed is added to the database. Also, keep the breeds information in the PETS_COUNTS when the dogs are deleted from PETS table. The breed_count should be always ≥ 0. For the sake of maintainability, you can create two separate triggers: one for INSERT and one for DELETE. Please create tables, add data, and test.

```
CREATE TABLE E1_Pets_Counts
  (
    Breed_Name       VARCHAR2 (20) PRIMARY KEY,
    Breed_Count      INTEGER,
    last_update_date DATE
  );

CREATE TABLE e1_pets
  (
    pet_name      VARCHAR2(20),
    breed_name    VARCHAR2(20),
    year_of_birth NUMBER(4)
  );
Insert Into E1_Pets_Counts Values ('German shepherd',0, Sysdate);
Insert Into E1_Pets_Counts Values ('Golden Retriever',0, Sysdate);
Insert Into E1_Pets_Counts Values ('Boxer',0, Sysdate);
```