Corrigé du TP BDA (Séance n° 3)
Création de triggers sur les relations

Première étape : Contrôle de l'intégrité de référence

Q1 : Définissez le trigger permettant d'interdire la suppression d'une matière qui est enseignée.

```
CREATE TRIGGER t_sup_matiere
BEFORE DELETE ON MODULE
FOR EACH ROW
DECLARE
     un_enseignt  ENSEIGNT%ROWTYPE ;
     CURSOR c_enseignt IS
          SELECT * FROM ENSEIGNT WHERE CODE = :old.CODE ;
BEGIN
     OPEN c_enseignt ;
     FETCH c_enseignt INTO un_enseignt ;
     IF c_enseignt%FOUND THEN
          RAISE_APPLICATION_ERROR
          (-20001, 'Matiere enseignee impossible a supprimer') ;
     END IF ;
     CLOSE c_enseignt ;
END ;
```

Le curseur c_enseignt permet de sélectionner les éventuels enseignements concernant la matière que l'utilisateur veut détruire. Il suffit ensuite de tester l'existence d'au moins un enregistrement dans le curseur. Avec cette solution, ORACLE ne déclenche aucune exception si la matière considérée n'est pas enseignée.

Autre possibilité pour spécifier le corps du trigger :
Une requête permet de tester l'existence d'un enseignement pour la matière considérée. Cette requête retourne au plus une valeur (grâce au **DISTINCT**). Cependant, s'il n'y a aucun résultat, Oracle déclenche l'exception **NO_DATA_FOUND** et si elle n'est pas explicitement gérée, il y a échec du trigger. L'instruction **NULL** est alors utilisée.

```
CREATE TRIGGER t_sup_matiere
BEFORE DELETE ON MODULE
FOR EACH ROW
DECLARE
     vcode  module.code%TYPE ;
     mat_enseignee EXCEPTION ;

BEGIN
     SELECT DISTINCT CODE INTO vcode
     FROM ENSEIGNT WHERE CODE = :old.CODE ;

     IF vcode IS NOT NULL THEN RAISE mat_enseignee ; END IF;

EXCEPTION
     WHEN NO_DATA_FOUND THEN NULL ;
     WHEN mat_enseignee THEN
          RAISE_APPLICATION_ERROR
          (-20001,'Matière enseignée non supprimable');
END ;
```

Il serait aussi possible de compter le nombre de valeurs pour l'attribut CODE et de déclencher une exception si ce nombre est strictement supérieur à zéro.

```
CREATE TRIGGER t_sup_matiere
BEFORE DELETE ON MODULE
FOR EACH ROW
DECLARE
NB NUMBER(2, 0);
BEGIN
      SELECT Count(*) INTO NB
      FROM ENSEIGNT WHERE CODE = :old.CODE ;
      IF NB > 0 THEN
      RAISE_APPLICATION_ERROR
      (-20001,'Matière enseignée non supprimable');
      END IF;
END ;
```

Q2 : Définissez le trigger contrôlant la même contrainte d'intégrité que précédemment mais cette fois ci en insertion.

La contrainte en question est " toute matière enseignée doit exister dans MODULE ". En insertion, la question est donc de définir un trigger contrôlant l'existence d'une matière lors de l'ajout de tuples dans ENSEIGNT.

```
CREATE OR REPLACE TRIGGER t_ins_enseignt BEFORE INSERT ON ENSEIGNT
FOR EACH ROW
DECLARE
     vcode  MODULE.CODE%TYPE ;

BEGIN
     SELECT CODE INTO vcode FROM MODULE WHERE CODE = :new.CODE ;

EXCEPTION
     WHEN NO_DATA_FOUND THEN
          RAISE_APPLICATION_ERROR (-20002, 'Matière inexistante') ;
END ;

Solution avec comptage :

CREATE OR REPLACE TRIGGER t_ins_enseignt BEFORE INSERT ON ENSEIGNT
FOR EACH ROW
DECLARE
     NB  NUMBER(2, 0) ;

BEGIN
     SELECT Count(*) INTO NB FROM MODULE WHERE CODE = :new.CODE ;
     IF NB = 0 THEN
     RAISE_APPLICATION_ERROR (-20002, 'Matière inexistante') ;
     END IF ;
END ;
```

Deuxième étape : Mise à jour automatique

Q3 : Définissez le trigger permettant une affectation automatique de la valeur 0, dès qu'une valeur nulle est utilisée pour les attributs MOY_CC et MOY_TEST dans la relation NOTATION.

```
CREATE TRIGGER t_maj_notation
BEFORE INSERT OR UPDATE OF MOY_CC, MOY_TEST ON NOTATION
FOR EACH ROW

BEGIN
     IF :new.MOY_CC IS NULL THEN :new.MOY_CC := 0 ; END IF;
     IF :new.MOY_TEST IS NULL THEN :new.MOY_TEST := 0 ; END IF;
END ;
```

Troisième étape : Contrôle de contraintes dynamiques

Q4 : Définissez le trigger permettant de réaliser le contrôle suivant : il doit être impossible d'attribuer des notes à un étudiant pour une matière qu'il ne suit pas.

```
CREATE TRIGGER t_ins_notation
BEFORE INSERT ON NOTATION
FOR EACH ROW
DECLARE
     NB NUMBER(2, 0);
BEGIN
     SELECT count(*) INTO NB FROM ENSEIGNT
     WHERE NUM_ET = :new.NUM_ET AND CODE = :new.CODE ;
     IF NB = 0 THEN
          RAISE_APPLICATION_ERROR (-20003, 'Matière non suivie') ;
     END IF ;
END ;
```

Q5 : Définissez le trigger permettant de s'assurer que la somme des coefficients de contrôle continu et de test est toujours égale à 100.

```
CREATE TRIGGER t_ins_matiere
BEFORE INSERT OR UPDATE OF COEFF_CC, COEFF_TEST ON MODULE
FOR EACH ROW
BEGIN
     IF :new.COEFF_CC + :new.COEFF_TEST != 100 THEN
          RAISE_APPLICATION_ERROR (-20004, 'Problème de coefficients') ;
     END IF ;
END ;
```

Quatrième étape : gestion de données calculées

Q6 : Création de la relation et programme d'insertion automatique de tuples dans GROUPES.

```
CREATE TABLE GROUPES(   ANNEE NUMBER(2,0),
                        NUMERO NUMBER(1,0),
                        EFFECTIF NUMBER(3,0),
                        CONSTRAINT PK_GROUPE PRIMARY KEY (ANNEE, NUMERO))

                          /* Parcours automatique */
DECLARE
      CURSOR C_GROUPE IS
            SELECT ANNEE, GROUPE, COUNT(*) effectif
            FROM ETUDIANT
            GROUP BY ANNEE, GROUPE ;
BEGIN
      FOR c IN C_GROUPE LOOP
            INSERT INTO GROUPES VALUES (c.ANNEE, c.GROUPE, c.effectif) ;
      END LOOP ;
END ;

                              /* Parcours manuel */
DECLARE
      CURSOR C_GROUPE IS
            SELECT ANNEE, GROUPE, COUNT(*) effectif
            FROM ETUDIANT
            GROUP BY ANNEE, GROUPE ;
      grp C_GROUPE%ROWTYPE ;
BEGIN
      OPEN C_GROUPE ;
LOOP
    FETCH C_GROUPE INTO grp ;
    DBMS_OUTPUT.PUT_LINE(grp.ANNEE||' '|| grp.GROUPE||' '|| grp.effectif) ;
    EXIT WHEN C_GROUPE%NOTFOUND ;
    INSERT INTO GROUPES VALUES (grp.ANNEE, grp.GROUPE, grp.effectif);
END LOOP ;
      CLOSE C_GROUPE ;
END ;
```

Q7 : Définissez le trigger permettant le re-calcul automatique, **chaque fois que nécessaire**, des effectifs de chaque groupe et la mise à jour de la relation GROUPE.

```
CREATE OR REPLACE TRIGGER t_maj_effectif
BEFORE INSERT OR DELETE OR UPDATE OF GROUPE ON ETUDIANT
FOR EACH ROW
DECLARE
     CURSOR c_groupe IS
          SELECT NUMERO FROM GROUPE
          WHERE ANNEE = :new.ANNEE AND EFFECTIF =
             (SELECT MIN(EFFECTIF) FROM GROUPE
             WHERE ANNEE = :new.ANNEE) ;
     num_groupe GROUPE.NUMERO%TYPE ;
     maj BOOLEAN := TRUE ;
     gr NUMBER ;
BEGIN
     IF :new.ANNEE IS NULL THEN
          RAISE_APPLICATION_ERROR (-20005, 'Spécifier l''annee') ;
     ELSE
            IF :new.GROUPE IS NULL THEN
                  OPEN c_groupe ;
                  FETCH c_groupe INTO num_groupe ;
                  IF c_groupe%NOTFOUND THEN
                        /* La relation GROUPE est vide. Il s'agit donc
                        de l'insertion du premier etudiant de la
                        promotion */
                        INSERT INTO groupe VALUES (:new.ANNEE, 1,1);
                        maj := FALSE ;
                  ELSE
                        :new.GROUPE := num_groupe ;
                  END IF ;
                  CLOSE c_groupe ;
            ELSE
                  SELECT COUNT(*) INTO gr FROM GROUPE
                  WHERE NUMERO = :new.GROUPE AND ANNEE = :new.ANNEE;
                  IF gr = 0 THEN
                        /*  La valeur du groupe spécifiée pour
                        l'étudiant est inexistante dans la relation
                        GROUPE */
                        INSERT INTO GROUPE
                              VALUES(:new.ANNEE, :new.GROUPE,1) ;
                        maj := FALSE ;
                  END IF ;
            END IF ;
            IF maj THEN
                  UPDATE GROUPE SET EFFECTIF = EFFECTIF + 1
                  WHERE NUMERO = :new.GROUPE AND ANNEE = :new.ANNEE ;
            END IF ;
            IF DELETING OR (UPDATING AND :old.groupe IS NOT NULL) THEN
                  UPDATE GROUPE SET EFFECTIF = EFFECTIF - 1
            WHERE NUMERO = :old.GROUPE AND ANNEE = :old.ANNEE ;
            END IF ;
     END IF ;
END ;
```