masquer=1

TD 11 : TRIGGERS

1. RAPPELS TRIGGERS

La syntaxe d'une expression de création de trigger en SQL3 est la suivante :

1. CREATE TRIGGER <nom-trigger>
2. BEFORE | AFTER | INSTEAD OF
3. INSERT | DELETE | UPDATE OF <liste_attributs>
4. ON <nom-table>
5. [ORDER <valeur de priorité>]
6. [REFERENCING NEW | OLD AS <nom-variable>]
7. FOR EACH ROW | STATEMENT
8. [WHEN (<conditionSQL>)]
9. [DECLARE]
10. <variables locales>
11. BEGIN <actionSQL> END ;

Les expressions entre […] sont optionnelles. Le symbol '|' sépare les options :

Ligne 1: <nom-trigger> indique le nom du trigger.
Ligne 2: indique si le trigger est déclenché avant (BEFORE), après (AFTER) ou à la place (INSTEAD OF) d'un événement
Ligne 3: indique le type de l'événement
Ligne 4: indique la table concernée par l'événement
Ligne 5: la clause ORDER est optionnelle et sert à gérer les priorités entre des triggers en conflit
Ligne 6: la clause REFERENCING sert à nommer une ou plusieurs variables temporaires utilisables dans la partie condition et dans la partie action. NEW et OLD désignent respectivement,
   1. la valeur du dernier n-uplet modifié par l'événement si la granularité de déclenchement est ROW (ligne 7)
   2. l'ensemble des nuplets touchés par l'événement si la granularité est STATEMENT (ligne 7).

   Le nom de la variable est `:nom_variable` (avec un ':' ajouté au début dans le reste du trigger. Si la clause REFERENCING est omise, alors les variables s'appellent par défaut `:new` et `:old` dans le reste du trigger.
Ligne 7: la clause FOR EACH détermine la granularité de déclenchement:
   - ROW : la règle est déclenchée à chaque n-uplet touché par l'événement.
   - STATEMENT : la règle est déclenchée qu'une seule fois pour l'événement (**Oracle** : cette ligne est vide pour FOR EACH STATEMENT).

Ligne 8: permet d'indiquer une condition SQL qui doit être vraie pour déclencher le trigger.
Ligne 9: lause optionnelles pour la déclaration de variables locales
Ligne 10: déclaration de variables locales (optionnelles)
Ligne 11: indique les actions à effectuer (en PLSQL) – voir TME7 pour plus d'explications.

Il est évident qu'à un événement de type INSERT (resp. DELETE) ne peut pas correspondre une delta-structure OLD (resp. NEW). Une règle AFTER ne devrait en principe pas modifier la valeur

de la variable temporaire déclarée par NEW (cette valeur est déjà écrite dans la base), une règle BEFORE ne devrait en principe pas modifier la base par une commande INSERT, UPDATE ou DELETE sur les relations de la base (puisque l'événement n'a pas encore eu lieu réellement), elle ne peut modifier que les variables temporaires.

2. BASE DE DONNÉES « ENTREPRISE »

La base de données d'une entreprise contient les trois tables suivantes :

```
CREATE TABLE  EMPLOYE (
        ID_EMP NUMBER(8) PRIMARY KEY,
        NOM VARCHAR(32) NOT NULL,
        PRENOM VARCHAR(32) NOT NULL,
        FONCTION VARCHAR(32),
        SALAIRE NUMBER(7,2) NOT NULL);
CREATE TABLE PROJET (
        ID_PROJ NUMBER(8) PRIMARY KEY,
        NOM VARCHAR(32) NOT NULL,
        ID_CHEF_PROJET NUMBER(8) REFERENCES EMPLOYE
                ON DELETE SET NULL
                ON UPDATE CASCADE);
CREATE TABLE PARTICIPE (
        ID_EMP NUMBER(8) REFERENCES EMPLOYE
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        ID_PROJET  NUMBER(8) REFERENCES PROJET
                ON DELETE RESTRICT
                ON UPDATE RESTRICT,
        PRIMARY KEY (ID_EMP, ID_PROJET));
```

La table EMPLOYE contient pour chaque employé son identifiant, son nom et prénom, sa fonction (optionnelle), et son salaire. La table PROJET contient les identifiants et noms des projets et une référence vers l'employé qui dirige le projet. La table PARTICIPE stocke les employés avec les projets auxquels ils participent.

Rappel **Oracle** :

1. ON UPDATE CASCADE n'est pas implanté.
2. ON DELETE RESTRICT est utilisé par défaut.

2.1 Écrire un trigger BEFORE qui évite qu'un salaire ne puisse diminuer (l'ancien salaire est maintenu en cas de diminution par une mise-à-jour).

2.2 Écrire un trigger AFTER qui évite qu'un salaire ne puisse diminuer (l'ancien salaire est maintenu en cas de diminution par une mise-à-jour).

2.3 Écrire deux triggers `nb_empl_avant` et `nb_empl_après` qui affichent pour chaque opération de suppression dans la table EMPLOYE, le nombre d'employés avant et après la suppression. Vous pouvez utiliser la fonction DBMS_OUTPUT.PUT_LINE.

2.4 Écrire un trigger qui annule les transaction avec des opérations qui suppriment plus de 50 n-uplets dans la relation EMPLOYE.

2.5 Écrire un trigger AFTER qui *simule* les actions ON DELETE | UPDATE associées aux clés étrangères de la table PROJET (on enlève les actions définies dans le schéma pour les remplacer par un trigger) :

- quand un EMPLOYE est effacé, l'attribut ID_CHEF_PROJET de la table PROJET est mis à NULL pour les projets concernés.
- quand l'identifant d'un EMPLOYE est mis-à-jour, l'attribut ID_CHEF_PROJET de la table PROJET est mis à jour pour les projets concernés.

2.6 Écrire un trigger AFTER qui *simule* les actions ON DELETE | UPDATE associées à la clé étrangère ID_EMP de la table PARTICIPE :