HAI502I – CORRECTION TRAVAUX PRATIQUES 4
DEFINITION DE TRAITEMENTS EN PL/SQL ET CREATION DE TRIGGERS

Objectifs du TP :
L'objet de cette quatrième séance de TP est la définition de traitements sur les données d'une base, en utilisant le langage procédural d'ORACLE PL/SQL.

Première étape : Gestion de valeurs

L'objectif de cette étape est d'écrire un programme PL/SQL permettant d'extraire de la base des données calculées et de les stocker dans une relation de travail pour les consulter ultérieurement. Plus précisément, on désire pour un abonné, dont le numéro sera donné par l'utilisateur à l'exécution du programme, déterminer le nombre total d'emprunts qu'il a effectués.

Rappel : Il est possible de faire saisir une valeur par l'utilisateur à l'aide du symbole &.
Par exemple : **SELECT** * **FROM** PILOTE **WHERE** Plnum=&Plnum ;
Affichera : Enter value for Plnum:

Pour cela, réalisez les opérations suivantes :

- ◆ créer une relation AB_NB qui doit comporter seulement deux attributs : NUMERO dont la définition doit être compatible avec NUM_AB de ABONNE et NB défini comme un entier sur 3 positions.

- ◆ A l'aide de votre éditeur préféré, définissez le bloc PL/SQL permettant :
  - de stocker dans une variable le nombre d'emprunt d'un abonné ;
  - d'insérer le couple formé du numéro d'abonné et du nombre d'emprunts de cet abonné dans la relation de travail AB_NB ;

  Après exécution du programme, contrôlez les résultats obtenus en consultant l'extension de la relation AB_NB.

En phase de test, effectuez des **ROLLBACK**, pour défaire les opérations d'insertion réalisées.
Les Rollback seront vus dans le cours sur les transactions. Le principe est, en faisant un rollback, de pouvoir revenir dans l'état du dernier enregistrement, i.e. annulation de toutes les modifications depuis le dernier point de sauvegarde (dans notre cas, il s'agit du début de session).

- ◆ Terminez le programme en tenant compte de la possibilité de n'avoir aucun abonné pour le numéro spécifié (insertion dans AB_NB d'une valeur -1 comme nombre de prêt), de n'avoir aucun prêt pour l'abonné (insertion dans AB_NB de NULL comme nombre de prêt).

Enfin, après l'avoir testé, définissez le programme comme une transaction, i.e. en insérant comme première et dernière instruction du bloc, l'ordre **COMMIT**. Testez l'exécution et assurez-vous de l'impossibilité de défaire la transaction.

```
-- Bloc Ins_ab_nb.sql
Declare
nb_emp number(3,0);
numero  number(6,0);
numero2 abonne.num_ab%type;
pasemprunt  exception;
begin
 numero := &numabonne ;
select num_ab into numero2 from abonne
where num_ab = numero;
 select count(num_ex) into nb_emp
 from emprunt
 where num_ab = numero and d_retour is not null;
 if nb_emp = 0 then raise pasemprunt; end if;
 insert into ab_nb values (numero,nb_emp);
exception
when  no_data_found  then
      insert into erreurs  values ('PAS D ABONNE DE NUMERO  ' || numero);
when pasemprunt  then
    begin
      insert into erreurs  values ('PAS D EMPRUNT EN  COURS POUR ' ||
numero);
      insert into ab_nb values (numero,NULL);
    end;
end;
/
```

Deuxième étape : Fonction stockée

Il s'agit, en s'inspirant du programme précédent, de définir une fonction `nbprets` qui accepte en paramètre d'entrée un numéro d'abonné et retourne le nombre de prêt de celui-ci.

- Créez la fonction `nbprets` puis testez la pour tous les abonnés de la base à l'aide d'une requête SQL.
- Modifiez le programme défini lors de la première étape afin qu 'il utilise la fonction `nbprets`.

```
create or replace function nbprets (num integer) RETURN Integer is nb
integer;
begin
select count(*) into nb
from emprunt where num_ab = num; return(nb);
end;
/
```

il est possible d'utiliser :

```
select num_ab, nbprets(num_ab) from emprunt;
```

ou bien dans un programme PL/SQL :

```
nb_emp := nbprets(numero);
```

Troisième étape : Gestion de curseurs

Il vous est demandé d'établir pour chaque abonné, le nombre d'emprunts réalisés pour chacune des catégories existantes.
Vous devez gérer tous les cas possibles : un abonné sans emprunt, un abonné sans emprunt pour une catégorie existante, une catégorie jamais empruntée.
Vous devez également traiter toutes les exceptions possibles.

- Nombre d'emprunts par catégorie par abonné

```
declare
-- le curseur est obtenu par produit cartésien assurant ainsi la
-- prise en compte de toutes les exceptions
cursor abonne_cat is
select distinct num_ab, categorie
from abonne, livre;
begin
delete from resultat;
commit;
for un_abonne_cat  in abonne_cat  loop
      select count(num_ex) into nb_emprunt
      from emprunt, exemplaire, livre
      where numero=num_ex and exemplaire.isbn = livre.isbn and
      categorie = un_abonne_cat.categorie
      and emprunt.num_ab =un_abonne_cat.num_ab;
      insert into RESULTAT values
(un_abonne_cat.num_ab,un_abonne_cat.categorie,nb_emprunt)
   end loop;
end loop;
end;
/
```

Quatrième étape : Trigger de contrôle d'une CI simple

Pour un trigger, il ne faut pas définir des traitements pouvant être pris en charge par une contrainte explicite d'Oracle. Il ne s'agit ici que de tester les possibilités offertes par les triggers et de bien comprendre le modèle d'exécution des triggers par rapport aux contraintes statiques.

- A l'aide d'un trigger ***before***, il s'agit de traiter la contrainte sur la relation EXEMPLAIRE permettant d'interdire la suppression d'un exemplaire si son état est BON. Testez le trigger défini en tentant la suppression de l'exemplaire de n° 7001.

Rappel : Il est possible d'afficher un message sur la console avec :

DBMS_OUTPUT.PUT_LINE (Texte à afficher)

Il faut cependant, au préalable, spécifier : **SET SERVEROUTPUT ON**

Trigger interdisant la suppression d'un exemplaire dont son état est bon

```
create or replace
```

```
trigger supp_exemplaire before delete on exemplaire
for each row
declare
etat_ex exemplaire.etat%type;
pb exception;
begin
if :old.etat  = 'BON' then raise_application_error (-20001, 'Suppression
impossible, Exemplaire en bon état');  end if;
end;

/* version avec exception utilisateur */
/* pour afficher a la console executer -> set serveroutput on  */
create or replace trigger supp_exemplaire before delete on exemplaire
for each row
declare
pb exception;
begin
if :old.etat  = 'BON'  then raise pb; end if;
exception
when pb then
    begin
       DBMS_OUTPUT.PUT_LINE ('Attention – Exemplaire en bon état ');
       raise_application_error (-20001, 'Suppression impossible');
    end;
end;
/



SQL> delete from exemplaire where numero = '7001';
Attention - – Exemplaire en bon état
delete from exemplaire where numero = '7001'
*
ERROR at line 1:
ORA-20001: Suppression impossible
ORA-06512: at "OPS$TOTO.SUPP_EXEMPLAIRE", line 10
ORA-04088: error during execution of trigger 'OPS$TOTO.SUPP_EXEMPLAIRE'
```

Cinquième étape : Trigger de mises à jour automatique

- Définissez le trigger permettant une affectation automatique de la date d'emprunt (en utilisant la fonction **SYSDATE** qui retourne la date du système) lors d'une insertion dans la relation EMPRUNT. De même, la date de retour sera calculée en ajoutant 21 jours à la date d'emprunt.

Insertion dans la relation EMPRUNT : affectation automatique des dates d'emprunt et de retour prévue.

```
create or replace trigger maj_auto_emprunt before insert on emprunt
for each row
begin
:new.d_emprunt := sysdate;
:new.d_retour := sysdate + 21;
end;
/
```