



HAI502I - TRAVAUX PRATIQUES 4

DEFINITION DE TRAITEMENTS EN PL/SQL ET CREATION DE TRIGGERS

Objectifs du TP :

L'objet de ce TP est la définition de traitements sur les données d'une base ainsi que la définition de triggers, en utilisant le langage procédural d'ORACLE PL/SQL. Une attention particulière des gestions des exceptions, des commentaires sera faite.

Remarques :

Les étapes 1, 2 et 4 vous permettront de voir un exemple de création d'un procédure, d'une fonction et d'un trigger en PL/SQL et doivent être faite pendant les séances de TP.

Les étapes 3 et 5 vous permettront d'aller plus loin et peuvent être faite à la maison en fonction du temps qu'ils vous restent pendant les séances de TP.

Première étape : Gestion de valeurs [A faire en séance de TP]
--

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 table 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 Pnum=&Pnum ;***

Affichera : Enter value for Pnum:

Pour cela, réalisez les opérations suivantes :

- Créer une table 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 table de travail AB_NB ;

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

En phase de test, exécuter l'ordre SQL « DELETE FROM AB_NB » pour supprimer les insertions réalisées.

- 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).



Deuxième étape : Fonction stockée [A faire en séance de TP]

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`.

Troisième étape : Gestion de curseurs [Facultatif]

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.

Quatrième étape : Trigger de contrôle d'une CI simple [A faire en séance de TP]

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 table `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**

Cinquième étape : Trigger de mises à jour automatique [Facultatif]

- 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 table `EMPRUNT`. De même, la date de retour sera calculée en ajoutant 21 jours à la date d'emprunt.