TD-TP SEANCE 5 : PROGR. SQL – PL/SQL – OCILIB

Objectif. Se familiariser avec la programmation SGBD avec PL/SQL et OCILIB. Pour cela, vous allez créer des scripts PL/SQL (extension « .sql ») à déclencher avec la commande @nom_fichier dans SQL*Plus. Ces blocs PL/SQL peuvent à terme être intégrés au serveur sous forme de procédures stockées.

Q1 : Phase préliminaire et prise en main

Créer la table RESULTAT (CODE number, MESSAGE char(50)). Compléter le bloc PL/SQL ci-dessous pour qu'il génère le résultat escompté.

***PROGRAMME PL/SQL* :**

```
DELETE FROM RESULTAT ;
PROMPT Nombre de lignes a produire
ACCEPT n
DECLARE
  x NUMBER:= ???;
BEGIN
  FOR i IN 1..&n LOOP
    IF MOD(x, ???) = 0 THEN   --   x   is
even
     INSERT INTO RESULTAT VALUES (???);
    ELSE
     INSERT INTO RESULTAT VALUES (???);
    END IF;
    x := x ??? ;
  END LOOP;
  COMMIT;
END; /
SELECT * FROM RESULTAT ;
```

***Table RESULTAT obtenue*:**

```
CODE    MESSAGE
----    ------
1   100 is even
2   101 is odd
3   102 is even
        … etc
```

Q2 : Bloc PL/SQL, expression conditionnelle

Lancer les scripts CREATE-bis.sql et CREATE-bis-data.sql.

Ecrire un programme PL/SQL qui : a) demande un numéro de client ; puis b) insère un tuple dans la table RESULTAT et le visualise. Ce tuple doit être tel que: (i) s'il n'y a pas de commande pour ce client, il a comme valeur 'pas de commande', (ii) sinon, il a comme valeur 'n commandes pour le client X' avec n le nombre de commandes pour et X le numéro du client.

Q3 : Bloc PL/SQL, curseurs et boucles

Ecrire un programme qui permette d'insérer dans la table RESULTAT les nième et n+1ième dernières commandes (leur numéro de commande, date, et prix total) en partant de la plus récente. Afficher les pour vérifier le bon fonctionnement. Le prix total de la commande sera calculé à partir des détails et des prix des produits commandés. Le nombre *n* est un paramètre saisi par l'utilisateur.

Q4 : Programmation OCI vs. PL/SQL vs. SQL

Soit le pseudo code suivant :

```
Stocker les clients dans une variable CURSOR nommée CLI_RES ;
Stocker les commandes dans une variable CURSOR nommée COM_RES ;
Stocker les détails  dans une variable CURSOR nommée DET_RES ;
Pour chaque ligne CLI_RES_i de CLI_RES FAIRE :
  NB = 0;
  Pour chaque ligne COM_RES_j de COM_RES FAIRE :
    Pour chaque ligne DET_RES_k de DET_RES FAIRE :
      Si (   CLI_RES_i.Numcli=COM_RES_j.Numcli et
             COM_RES_j.Numcom=DET_RES_k.numcom )
        NB = NB + DET_RES_k.Qte;
      Fin Si;
    Fin Pour;
  Fin Pour;
RES = "nombre des articles du client" + CLI_RES_i.ID + "=" + NB ;
Insérer (CLI_RES_i.ID, RES) dans la table RESUTLAT ;
Fin Pour;
```

Implémenter ce traitement en C avec OCILIB (à partir du code C fournit dans le répertoire du cours) et mesurer le temps d'exécution du programme obtenu.
Implémenter le même traitement en PLSQL (à partir du script PL/SQL fourni), puis mesurer le temps d'exécution du script obtenu.
Implémenter le même traitement en SQL pur et mesurer le temps d'exécution.

Q5 : Bloc PL/SQL, curseurs paramétrés

Ecrire un programme qui utilisera un curseur paramétré et qui permette de vérifier que le montant payé (COM.Payement) pour une commande donnée (renseignée par l'utilisateur) est égal à la somme des lignes de commandes correspondantes. Pour la commande demandée, une ligne sera insérée dans la table résultat. Cette ligne aura la forme suivante : « NumCom : 100 - Payement : 5000 - Prix : 5000 ».

Q6 : Bloc PL/SQL, exceptions

Compléter le programme de la question 2 pour gérer l'erreur survenant dans le cas où le nombre N serait strictement supérieur au nombre de commandes dans la table Commandes. Dans ce cas, un message d'erreur sera inséré dans la table résultat.

Q7 : Bloc PL/SQL, packages

Ecrire le package « client » comportant une procédure insérant dans la table RESULTAT les nième et n+1ième commandes d'un client donné en argument et gérant l'exception, et une fonction renvoyant le nombre de commandes pour un client donné passé en argument. Tester l'utilisation du package et des procédures.