ED PL/SQL

(Énoncé)

Par la suite on considère que les tables utilisées par les exercices ont été déjà crées et remplies avec les données nécessaires.

Déclarations, itérations, boucles, instructions conditionnelles

Exercice 1. Soit la table suivante :
VOL(Numvol, Heure_départ, Heure_arrivée, Ville_départ, Ville_arrivée)
Écrivez un programme PL/SQL qui insère le vol AF110 partant de Paris à 21h40 et arrivant à Dublin à 23h10 (hypothèse : le vol n'est pas déjà présent dans la table).

Exercice 2. Soit la table RES(NO). Écrivez un bloc PL/SQL qui inséré les chiffres de 1 à 100 dans cette table.

Exercice 3. Écrivez un bloc PL/SQL qui affiche la somme des nombres entre 1000 et 10000.

Exercice 4. Écrivez un programme PL/SQL qui affiche le reste de la division de 17664 par 171. Ne pas utilisez la fonction MOD.

Exercice 5. Créez une type tableau pouvant contenir jusqu'à 50 entiers.
1. Créez une variable de ce type, faites une allocation dynamique et dimensionnez ce tableau à 20 emplacements.
2. Placez dans ce tableau la liste des 20 premiers carrés parfaits : 1, 4, 9, 16, 25, …
3. Affichez ce tableau.

Exercice 6. Écrire une fonction PL/SQL qui prends en entrée un nombre entier n et retourne le factoriel de ce nombre $n!$. Implémenter deux versions : itérative et récursive. La version récursive est basée sur la relation de récurrence : $n! = n \cdot [(n-1)!]$

Curseurs, déclencheurs, relations

Exercice 7. On considère la table suivante:
PILOTE(Matricule, Nom, Ville, Age, Salaire).
Écrivez un programme PL/SQL qui calcule la moyenne des salaires des pilotes dont l'âge est entre 30 et 40 ans.

Exercice 8. Soit la base de données suivante (simplifiée) de gestion de la mémoire d'un ordinateur :
DISQUE(nom, capacité, vitesse, fabricant);
PARTITION(nomDisque, nomPartition, taille);
Écrivez en PL/SQL le déclencheur qui lors de l'insertion d'une nouvelle ligne dans la table PARTITION vérifie que la taille totale des partitions sur le disque concerné (y compris la partition qui est en cours d'être ajoutée) ne dépasse pas la capacité du disque. Si tel n'est pas le cas, l'enregistrement de la nouvelle cage ne doit pas être fait et un message doit être affiché pour indiquer cette anomalie.

Exercice 9. Soit la relation :
EMPLOYE(ID, NOM, DEPARTEMENT, AGE, SALAIRE).
Écrivez un bloc PLSQL qui effectue une augmentation de 200 euros du salaire des employés du département 'Commercial' et qui utilise le dernier curseur implicite pour afficher le nombre d'employés affectés par ce changement.

Exercice 10. Soit la relation EMPLOYE de l'exercice précédent. crivez un bloc PL/SQL qui affiche les noms des employØs du dØpartement 'Commercial' qui sont gØs de plus de 40 ans. Utilisez un curseur implicite dans une boucle FOR.

Exercice 11. Soit la relation EMPLOYE de l'exercice précédent. Écrivez une procédure PLSQL qui prends en paramètre un NUMBER (age limite) et qui affiche pour chaque département le nombre des employés qui dépassent l'age limite. Utilisez un curseur avec paramètre l'age limite.

Exercice 12. Soit la table suivante :
METEO(NOM_VILLE, TempØrature, HumiditØ
crire une fonction PL/SQL qui prends en entrØe le nom d'une ville et retourne la tempØrature et l'humiditØde cette ville. GØrer aussi par une exception le cas ou la ville n'existe pas.

Exercice 13. Soit la table METEO de l'exercice prØcØdent.
crire un dØclencheur qui avant l'insertion d'une nouvelle ville dans la table vØrifie :
a. Si la tempØrature est la plus grande de toutes les villes, afficher un message d'avertissement.
b. Si la ville existe dØ dans la table, ne pas l'insØrer une nouvelle fois mais faire la mis a jour seulement.

Exercice 14. On considère la base de données suivante :
COMPETITION(CODE_COMP, NOM_COMPETITION)
PARTICIPANT(NO_PART, NOM_PART, DATENAISSANCE, ADRESSE, EMAIL)
SCORE(NO_PAR, CODE_COMP, NO_JUGE, NOTE)
Écrire un bloc PLSQL qui lit à la console le nom d'une compétition et qui affiche les participants avec leur score total (la somme de tous les scores par tous les juges). Utilisez un curseur avec paramètre.

Exercice 15. On considère la table COMPETITION donné dans l'exercice précédent. Écrire un dØclencheur qui vØrifie que le code d'une compØition commence par les lettres 'CMP' avant son l'insertion dans la table COMPETITION.

Exercice 16. On considère la base de données suivante :
CLIENT(CL_ID, CL_NOM, CL_ADDR, CL_VILLE, EMAILID, CONTACT_NO)
MAGAZINE(MAG_ID, MAG_NOM, PRIX_UNITE, TYPE_ABONNEMENT)
ABONNEMENT(CL_ID, MAG_ID, START_DATE, END_DATE)
Écrire une fonction PL/SQL qui retourne le nombre de clients de Dijon qui se sont abonnés au magazine « Vogue » après août 2010. S'il n'y a pas de clients qui remplissent la condition, lancer une exception utilisateur avec un message d'erreur.

Exercice 17. Créer un déclencheur qui est lancé après chaque nouvelle commande INSERT dans la table ABONNEMENT de l'exercice précédent. Le déclencheur fait la mis a jour du nombre d'abonnements dans la table suivante :
TRACK_ABONNEMENTS(MAG_NOM MAGAZINE.MAG_NOM%TYPE,
NB_ABONN INTEGER)
On considère que la table TRACK_ABONNEMENTS contient déjà tous les magazines (ceux qui

n'ont pas d'abonnement on NB_ABONN à 0).

Exercice 18. Soit les tables suivantes :

VOL(Numvol, Heure_départ, Heure_arrivée, Ville_départ, Ville_arrivée)

ESCALE(Numescale, Ville_Escale, Durée_Escale)

Écrivez un programme PL/SQL qui affiche les vols pour un tour du monde au départ de Paris avec des escales et des durées d'escale prédéfinies dans la table ESCALE. Le nombre d'escales à faire doit être demandé à l'utilisateur. Le numéro de chaque escales est donné par Numescale. Hypothèse de travail : pour chaque escale il existe un vol et un seul satisfaisant les contraintes. Par exemple, si l'utilisateur souhaite un tour du monde de quatre escales, le tour du monde proposé sera le suivant :

Paris → Escale no 1 → Escale no 2 → Escale no 3 → Escale no 4 → Paris

Exercice 19. Modifiez le programme PL/SQL de l'exercice précédent pour qu'il fonctionne même si plusieurs vols satisfont les contraintes.

Exercice 20. On considéré la table Vol de l'exercice précédent :

VOL(Numvol, Heure_départ, Heure_arrivée, Ville_départ, Ville_arrivée)

Écrivez une procédure PL/SQL capable de faire des propositions de tours du monde, prenant en entrée la ville de départ (qui est aussi la destination finale) et deux bornes (supérieure et inférieure) pour le nombre d'escales. Dans ce cas il n'y a pas de liste prédéfinie d'escales et on ne s'intéresse pas à la durée des escales. La procédure doit afficher les vols pour chaque tour du monde proposé. Cherchez à utiliser une procédure ou fonction récursive.

Références bibliographiques

Anne-Sophie LACROIX, Jérôme GABILLAUD, Programmez avec SQL et PL/SQL, Editions ENI 2015
Joan Casteel, *Oracle 11g: PL/SQL Programming, Cengage Learning*, 2012
Michael McLaughlin, *Oracle Database 12c PL/SQL Programming*, McGraw-Hill 2014

Ressources web :

http://gtuplsql.blogspot.fr
http://alexandre-mesle.com/index.php/enseignement/bases-de-donnees/
http://docs.oracle.com/

Rappel SQL

INSERT INTO Table (...) VALUES (...)	UPDATE Table SET col1 = valeur1, col2 = valeur2 WHERE condition	SELECT attributs FROM Table WHERE condition	SELECT attributs INTO variables FROM Table WHERE condition

Rappel syntaxe PL/SQL

Bloc PLSQL :

```
DECLARE
    – Déclarations variables, constantes, records, curseurs, exceptions etc.
BEGIN
    – Code
EXCEPTION
    – Traitement exceptions
END
```

Instructions conditionnelles :

```
IF cond1 THEN
    – code
ELSIF cond2 THEN
    – code
…
ELSE
    – code
END IF
```

```
CASE expr
WHEN val1 THEN
    – code
WHEN val2 THEN resultat2
    – code
ELSE
    – code
END CASE
```

Boucles :

```
LOOP
  – code
  EXIT WHEN condition
END LOOP
```

```
WHILE condition
LOOP
  – code
END LOOP
```

```
FOR i IN val1 .. val2 LOOP
  – code
END LOOP

FOR item IN curseur LOOP
  – code
END LOOP
```

Curseurs :

```
CURSOR C IS SELECT_STATEMENT ;
FETCH C INTO variable_record ;
FETCH C INTO liste_variables ;
CLOSE C ;
```

Procédures et fonctions :

```
CREATE OR REPLACE PROCEDURE [FUNCTION] nomProcedure(liste_paramètres)
[RETURN TYPE]
IS
  – Déclarations variables
BEGIN
    – code
    [RETURN val]
EXCEPTION
    – code
    [RETURN val]
END
```

Les paramètres peuvent être IN, OUT ou IN OUT. Exemple :

```
FONCTION moyenneSalaire (id_departement IN NUMBER) RETURN NUMBER IS ...
```

Déclencheurs :

```
CREATE OR REPLACE TRIGGER nomTrigger
[BEFORE | AFTER]
[INSERT [OR] UPDATE [OR] DELETE]
[OF  nom_attribut]
ON table
[REFERENCING OLD as O NEW as N]
[FOR EACH ROW]
[WHEN condition]
DECLARE
  – declarations
BEGIN
  – code
END
```

Attention : éviter les boucles infinies. Exemple déclencheurs A et B :

INSERT déclenche A
A exécute UPDATE
UPDATE déclenche B
B exécute INSERT
INSERT déclenche A → boucle !

Traitement exceptions :

```
EXCEPTION
WHEN exception1 THEN
  – code
WHEN exception1 THEN
  – code
WHEN exception1 THEN
  – code
WHEN OTHERS
  – code
END
```