Oracle PL/SQL

Module 1

INTRODUCTION

Qu'est ce que le PL/SQL ?

- SQL :
  - Est un langage ensembliste et non procédural. Les traitements complexes sont parfois difficiles à écrire si on ne peut utiliser des variables et les structures de programmation comme les boucles et les alternatives
- PL/SQL :
  - Est un langage procédural, qui intègre des ordres SQL de gestion de la base de données

Fonctionnalités de PL/SQL

- PL/SQL est un langage algorithmique complet
- Instructions SQL intégrées dans PL/SQL :
  - Select
  - insert, update, delete
  - La partie gestion de transactions
  - Les fonctions (to_char, to_date, upper, substr, round, ...)
- Une partie procédurale (IF, WHILE, …)
- Instructions spécifiques à PL/SQL :
  - Défintion de variables
  - Traitements conditionnels
  - Traitements répétitifs
  - Traitement des curseurs
  - Traitement des erreurs

Module 2

LE BLOC PL/SQL

La structure d'un bloc PL/SQL

- Les blocs, comme les instructions se terminent par un « ; »
- Seuls BEGIN et END sont obligatoires

```
DECLARE
-- Déclaration de variables, constantes, exceptions
BEGIN
-- Les instructions à exécuter (commandes exécutables,
-- instructions SQL et
-- PL/SQL, Possibilité de blocs fils (imbrication de blocs))
EXCEPTION
-- Traitement des exceptions (gestion des erreurs)
END
```

Module 2

LES VARIABLES

Les variables

- Identificateurs Oracle :
  - 30 caractères au plus
  - commence par une lettre
  - peut contenir lettres, chiffres, _, $ et #
- Pas sensible à la casse
- Portée habituelle des langages à blocs
- Doivent être déclarées avant d'être utilisées

Types de variables

- Les types habituels correspondants aux types SQL2 ou Oracle :
  - integer, varchar,…
- Types composites adaptés à la récupération des colonnes et lignes des tables SQL :
  - %TYPE, %ROWTYPE

Déclaration des variables

- Les variables locales se définissent dans la partie DECLARE d'un bloc PL/SQL.
  - identificateur [CONSTANT] type [:= valeur];

```
DECLARE
   nom CHAR(15);
   numero NUMBER;
   date_jour DATE;
   salaire NUMBER(7,2);
```

Déclaration des variables

+ VARIABLES PL/SQL de type booléen :

```
DECLARE
        reponse BOOLEAN := true ;
BEGIN
…..
END;
```

Déclaration des variables

- VARIABLES faisant référence au dictionnaire de données
  - Variables reprenant le même type qu'une colonne dans la table

Syntaxe :

```
Nom_variable table.colonne%TYPE ;
```

Exemple :

```
DECLARE
        nom emp.ename%TYPE ;
BEGIN
.....
END;
```

Déclaration des variables

+ Variables de même type qu'une variable précédemment définie

Syntaxe :

nom_variable2 nom_variable1%TYPE ;

Exemple :

```
DECLARE
        commi NUMBER(7,2) ;
        salaire commi%TYPE ;
BEGIN
.....
END;
```

Déclaration des variables

- Variables reprenant la même structure qu'une ligne d'une table
  - Chaque variable de la structure Enreg a le même nom et le même type que la colonne associée

Syntaxe :

```
nom_variable table%ROWTYPE ;
```

Exemple :

```
DECLARE
        enreg emp%ROWTYPE ;
BEGIN
.....
END;
```

Initialisation des variables

- L'initialisation d'une variable peut se faire par :
  - L'opérateur := dans les sections DECLARE, BEGIN et EXCEPTION
  - L'ordre SELECT … INTO …. dans la section BEGIN
  - Le traitement d'un curseur dans la section BEGIN
- Une variable est visible dans le bloc où elle a été déclarée et dans les blocs imbriqués si elle n'a pas été redéfinie.

Initialisation des variables

- Avec l'opérateur :=

- Figer l'affectation de valeur à une variable avec la clause CONSTANT

```
DECLARE
        nom CHAR(10) := 'Ben Amor' ;
        salaire NUMBER(7,2) := 1500 ;
        reponse BOOLEAN := TRUE ;
BEGIN
....
END;
```

```
DECLARE
        pi CONSTANT NUMBER(7,2) := 3.14 ;
BEGIN
....
END ;
```

Initialisation des variables

- Interdire les valeurs non renseignées avec la clause NOT NULL

```
DECLARE
        i NUMBER NOT NULL := 1000 ;
BEGIN
....
END;
```

Initialisation des variables

- L'ordre SELECT
  - La clause INTO est OBLIGATOIRE.
  - Le SELECT doit obligatoirement ramener une ligne et une seule sinon erreur
  - Pour traiter un ordre SELECT qui permet de ramener plusieurs lignes, on utilise un curseur.

Syntaxe :

```
SELECT col1, col2 INTO var1, var2
FROM table [WHERE condition ] ;
```

Initialisation des variables

- L'ordre SELECT
  - Exemple

```
DECLARE
        nom_emp CHAR(15) ;
        salaire emp.sal%TYPE ;
        commision emp.comm%TYPE ;
        nom_depart CHAR(15) ;
BEGIN
        SELECT ename, sal, comm, dname
        INTO nom_emp, salaire, commision, nom_depart
        FROM emp, dept
        WHERE ename = 'Hammami'
            AND emp.deptno=dept.deptno ;
…..
END ;
```

Module 3

LES TRAITEMENTS CONDITIONNELS

Définition et syntaxe

```
IF condition THEN
        instructions;
END IF;
```

```
IF condition THEN
        instructions 1;
ELSE
        instructions 2;
END IF;
```

```
IF condition 1 THEN
        instructions 1;
ELSEIF condition 2 THEN
        instructions 2;
ELSEIF …
…
ELSE
        instructions N;
END IF;
```

Définition et syntaxe - Exemple

```
DECLARE
        emploi CHAR(10)
        nom CHAR(15) := 'Hammami' ;
        mes CHAR(30) ;
BEGIN
        SELECT job INTO emploi FROM emp WHERE ename=nom;
        IF emploi IS NULL THEN
                mes := nom||'n'a pas d''emploi';
        ELSIF emploi = 'Commercial' THEN
                UPDATE emp SET com = 1000 WHERE ename = nom;
                mes := nom || 'commision modifiée' ;
        ELSE
                UPDATE emp SET com = 0 WHERE ename = nom;
                mes := nom || 'pas de commision' ;
        END IF ;
        INSERT INTO resultat VALUES (mes) ;
        COMMIT;
END;
```

Module 4

LES TRAITEMENTS RÉPÉTITIFS

Les boucles

- Il existe 3 type de boucles :
  - La boucle de base (LOOP)
  - La boucle FOR
  - La boucle WHILE

La boucle LOOP

```
LOOP
        instructions;
EXIT [WHEN condition];
        instructions;
END LOOP;
```

Exemple : Insérer les 10 premiers chiffres dans la table resultat.

```
DECLARE
        nbre NUMBER := 1 ;
BEGIN
        LOOP
                INSERT INTO resultat VALUES (nbre) ;
                nbre := nbre + 1
        EXIT WHEN nbre > 10 ;
        END LOOP ;
END ;
```

La boucle FOR

```
FOR compteur IN [REVERSE] inf .. Sup LOOP
        instructions;
END LOOP;
```

Exemple : Calcul de factorielle 9.

```
DECLARE
        Fact NUMBER := 1 ;
BEGIN
        FOR i IN 1..9
        LOOP
                fact := fact * i ;
        END LOOP ;
        INSERT INTO resultat
        VALUES (fact, 'FACTORIELLE9');
END ;
```

La boucle WHILE

```
WHILE condition LOOP
        instructions;
END LOOP;
```

Exemple : Reste de la division de 7324 par 9.

```
DECLARE
        reste NUMBER := 7324 ;
BEGIN
        WHILE reste >= 9
        LOOP
        reste := reste - 9 ;
        END LOOP ;
        INSERT INTO resultat
        VALUES (reste,'reste division de 7324 par 9 ');
END;
```

Module 5

LES CURSEURS

Définition

- Pour traiter une commande SQL, PL/SQL ouvre une zone de contexte pour exécuter la commande et stocker les informations.
- Le curseur permet de nommer cette zone de contexte, d'accéder aux informations et éventuellement de contrôler le traitement.
- Cette zone de contexte est une mémoire de taille fixe, utilisée par le noyau pour analyser et interpréter tout ordre SQL.
- Les statuts d'exécution de l'ordre se trouvent dans le curseur.

Les types de curseur

- Le curseur explicite
  - Il est créé et géré par l'utilisateur pour traiter un ordre Select qui ramène plusieurs lignes. Le traitement du select se fera ligne par ligne.
- Le curseur implicite
  - Il est généré et géré par le noyau pour les autres commandes SQL.

Les curseurs explicites

- Pour traiter les select qui renvoient plusieurs lignes
- Ils doivent être déclarés
- Le code doit les utiliser explicitement avec les ordres OPEN, FETCH et CLOSE
- Le plus souvent on les utilise dans une boucle dont on sort quand l'attribut NOTFOUND du curseur est vrai

Les curseurs explicites

- L'utilisation d'un curseur pour traiter un ordre Select ramenant plusieurs lignes, nécessite 4 étapes :
    1. Déclaration du curseur
    2. Ouverture du curseur
    3. Traitement des lignes
    4. Fermeture du curseur

Déclaration d'un curseur explicite

- Tout curseur explicite utilisé dans un bloc PL/SQL doit être déclaré dans la section DECLARE du bloc en donnant :
  - Son nom
  - L'ordre select associé

Syntaxe :

```
CURSOR nom_curseur IS ordre_select;
```

Exemple :

```
DECLARE
        CURSOR dept_10 IS
        SELECT ename,Sal FROM emp
        WHERE deptno = 10
        ORDER BY sal;
BEGIN
… ;
… ;
END;
```

Ouverture d'un curseur explicite

- Après avoir déclaré le curseur , il faut l'ouvrir pour faire exécuter l'ordre SELECT.
- L'ouverture du curseur se fait dans la section BEGIN du bloc

Syntaxe :

```
OPEN nom_curseur;
```

Exemple :

```
DECLARE
        CURSOR dept_10 IS
        SELECT ename,Sal FROM emp
        WHERE deptno = 10
        ORDER BY sal;
BEGIN
… ;
        OPEN dept_10;
… ;
END;
```

Traitement des lignes

- Après l'exécution du SELECT, les lignes ramenées sont traitées une par une.
- La valeur de chaque colonne du SELECT doit être stockée dans une variable réceptrice
- Le Fetch ramène une seule ligne à la fois. Pour traiter n lignes, il faut prévoir une boucle

Syntaxe :

```
FETCH nom_curseur INTO liste_variables;
```

Exemple :

```
DECLARE
        CURSOR dept_10 IS Select ename,sal FROM emp
                WHERE deptno = 10 ORDER BY sal;
        nom emp.ename%TYPE;
        salaire emp.sal%TYPE;
BEGIN
        OPEN dept_10;
        LOOP FETCH dept_10 INTO nom, salaire;
                IF salaire >2500 THEN
                INSERT INTO resultat VALUES
(nom,salaire);
        END IF;
        EXIT WHEN salaire = 5000;
END LOOP;
END;
```

Fermeture du curseur

- Après le traitement des lignes, on ferme le curseur pour libérer la place mémoire

Syntaxe :

```
CLOSE nom_curseur;
```

Exemple :

```
DECLARE
 ...
BEGIN
 ....
        END LOOP;
        CLOSE dept_10;
END;
```

Les attributs d'un curseur

- Les attributs d'un curseur sont des indicateurs sur l'état d'un curseur
  - %FOUND → Dernière ligne traitée
  - %NOTFOUND → Dernière ligne traitée
  - %ISOPEN → Ouverture d'un curseur
  - %ROWCOUNT → nombre de lignes

Attribut %FOUND

- Type: booléen
- Syntaxe:
  - Curseur implicite : SQL%FOUND
  - Curseur explicite : nom_curseur%FOUND
- Si sa valeur est vrai donc le dernier FETCH a ramené une ligne

Attribut %FOUND

```
DECLARE
        CURSOR dept_10 IS SELECT  ename,sal FROM emp
        WHERE deptno = 10 ORDER BY sal;
        nom emp.ename%TYPE;
        salaire emp.sal%TYPE;
BEGIN
        OPEN dept_10;
        FETCH dept_10 INTO nom, salaire;
        WHILE dept_10%FOUND
        LOOP
                IF salaire >2500 THEN
                        INSERT INTO resultat VALUES (nom,salaire);
                END IF;
                FETCH dept_10 INTO nom, salaire;
        END LOOP;
        CLOSE dept_10;
END;
```