Université de l'Unité Africaine (UUA)
Année universitaire : 2020 – 2021
Filière: Informatique de Gestion
Niveau: 2ème année de Licence (IG2)
Module : Gestion de base de données Oracle
Enseignant : M. SANA

Gestion de base de données Oracle

Chap 3: Le langage PL/SQL – Partie 1

Enseignant : M. SANA

Table des matières

I. Introduction

1. Pourquoi PL/SQL ?

- **SQL est un langage non procédural**
- Le développement d'application autour d'une BDR nécessite d'utiliser :
  - Des **variables**
  - Des **structures de contrôle** de la programmation (boucles et alternatives)

➔ Besoin d'un **langage procédural** pour lier plusieurs requêtes SQL avec des variables et dans les structures de contrôle habituelles =

L4G (langage de 4ième Génération) :

D'où **PL/SQL** (Acronyme : ProceduraL SQL) : langage de programmation procédural et structuré pour développer des applications autour de bases de données relationnelles (SQL)

2. Normalisation du langage

- **PL/SQL** est un langage propriétaire de **Oracle**

Pas de véritable standard, la plupart des SGBD relationnels propose des **L4G** (langage de 4ième génération) spécifiques, semblables à PL/SQL :

- **PostgreSQL** propose **PL/pgSQL** très proche de PL/SQL et **PL/pgPSM**
- **MySQL** et **Mimer SQL** proposent un langage analogue dans le principe mais plus limité : **SQL/PSM (**de la norme SQL2003),
- **IBM DB2** propose un dérivé de PL/SQL : **SQL-PL**
- **Microsoft/SQL server** et **Sybase** propose **Transact-SQL (T-SQL)** développé par à l'origine par Sybase
- **…**

3. Principales caractéristiques de PL/SQL

- **PL/SQL = Extension de SQL** : des requêtes SQL cohabitent avec les structures de contrôle habituelles de la programmation structurée (blocs, alternatives, boucles)
- La syntaxe ressemble au langage **Ada**
- Un programme est constitué de **variables**, **procédures** et **fonctions**

- PL/SQL permet :
  - L'utilisation de **variables permettant l'échange d'information** entre les requêtes SQL et le reste du programme : ces variables sont de type simple et structuré dynamique (%TYPE, %ROWTYPE, etc)
  - des traitements plus complexes, notamment pour la gestion des cas particuliers et des erreurs (traitement des **exceptions**),
  - l'utilisation de **librairies standards prédéfinies** (supplied PLSQL packages, comme les RDBMS_xxx)
  - un **paramétrage** et la création **d'ordres SQL dynamiques**

On a choisi ici de présenter dans ce cours le langage PL/SQL d'Oracle puisque c'est l'objet du module. Nous verrons à la fin du cours ses différences avec PL/pgSQL de PostgreSQL.

4. Utilisation de PL/SQL

- **Le PL/SQL peut être utilisé sous 3 formes :**
  - un **bloc de code** exécuté comme **une commande SQL**, via un interpréteur standard (SQL+ ou SQL*PLus)
  - un **fichier de commande PL/SQL**
  - un **programme stocké** (procédure, fonction, package ou trigger)

- **Ainsi PL/SQL peut être utilisé :**
  - pour l'écriture de **procédures stockées** et des **triggers** (Oracle accepte aussi le langage Java)
  - pour l'écriture de **fonctions utilisateurs** qui peuvent être utilisées dans les requêtes SQL (en plus des fonctions prédéfinies)
  - dans des **outils Oracle**, Forms et Report en particulier

II. Eléments de programmation, variable et type dans PL/SQL

1. Ordres SQL supportés dans PL/SQL

Ce sont les instructions du langage SQL :

- Pour la **manipulation de données** (LMD) :
  - INSERT
  - UPDATE
  - DELETE
  - SELECT
- Et certaines instructions de **gestion de transaction** :
  - COMMIT,
  - ROLLBACK
  - SAVEPOINT
  - LOCK TABLE
  - SET TRANSACTION READ ONLY.

2. Structure d'un programme PL/SQL : les blocs

- **Un programme est structuré en blocs d'instructions de 3 types :**
  - procédures anonymes
  - procédures nommées
  - fonctions nommées

- **Structure d'un bloc :**

[DECLARE]

- définitions de variables, constantes, exceptions, curseurs

BEGIN

- les instructions à exécuter (ordres SQL, instructions PL/SQL, structures de contrôles)

[EXCEPTIONS]

- la récupération des erreurs (traitement des exceptions)

END ;

- **Remarques :**
  - Un bloc peut contenir d'autres blocs
  - Seuls begin et end sont obligatoires
  - Les blocs comme les instructions se termine par un « ; »

3. Identification de variable dans PL/SQL

- **Identificateurs sous Oracle** (insensible à la case Majus/Minus):
  - 30 caractères au plus,
  - Commence par une lettre
  - Peut contenir : lettres, chiffres, _, $ et #
- **Portée habituelle des langages à blocs**
- Doivent être **déclarées avant d'être utilisées**
- **Commentaires :**
  - **--** Pour une fin de ligne
  - **/*** Pour plusieurs lignes ***/**

4. Les types de variables dans PL/SQL

- Les types habituels correspondants aux types SQL2 ou Oracle :

integer, **varchar**,...

- Les types composites adaptés à la récupération des colonnes et des lignes des tables SQL : **%TYPE**, **%ROWTYPE**
- Type référence : **REF**

5. Declaration de variable

- Identificateur [CONSTANT] type [:= valeur];
- Ex :
  - **age integer; nom varchar(30);**
  - **dateNaissance date; ok boolean := true;**
- Déclarations multiples **interdites !** (I, j integer ; interdit !)

6. Déclaration des variables **%TYPE**, **%ROWTYPE** %TYPE

- Déclaration qu'une variable est du même type qu'une colonne d'une table ou d'une vue (ou qu'une autre variable) :

- Ex : **nom emp.nome.%TYPE;**

Déclare que la variable « nom » contiendra une colonne de la table emp

%ROWTYPE

- Une variable peut contenir toutes les colonnes d'une ligne d'une table/vue
- Ex : **employe emp%ROWTYPE;**

Déclare que la variable « employe » contiendra une ligne de la table emp

7. Utilisation des variables %TYPE , %ROWTYPE

```
DECLARE
        employe emp%ROWTYPE;
        nom emp.nome.%TYPE;
        …
BEGIN
        SELECT * INTO employe
        FROM emp
        WHERE matr = 900;
        nom := employe.nome;
        employe.dept := 20;
        …
        INSERT INTO emp
        VALUES employe;
        …
END ;
```

8. Types de données dans PL/SQL

- Types de données standards SQL :
  - char, date, number, etc.
- Types intégrés dans PL/SQL :
  - boolean, binary, integer

- Types structurés PL/SQL :
  - record, table
- Structures de données pour la gestion de résultats de requêtes :
  - cursor
- Types de données définis par les utilisateurs

9. Type structure PL/SQL : Type RECORD

- Equivalent à **struct** du langage C

Exemple :

```
TYPE nomRecord IS RECORD (
    champ1 type1,
    champ2 type2,
    ...);
```

- Utilisation :

```
TYPE emp2 IS RECORD (
matr integer,
nom varchar(30));

employe emp2;
employe.matr := 500;
```

10. Affectation et contrôle de nom de variables

Plusieurs façons d'affecter une valeur à une variable :

- Par « := »
- par la directive **INTO** de la requête SELECT

Exemples :

```
dateNaissance := '09/09/2015';
SELECT nome INTO nom FROM emp
WHERE matr = 125;
```

Conflits de noms :

- Si une variable porte le même nom qu'une colonne d'une table, c'est la colonne qui l'emporte :

```
DECLARE nome varchar(30) := Isaac;
BEGIN
        DELETE FROM emp WHERE nome = nome;
        …
```

→ Pour éviter ça, le plus simple est de ne pas donner de nom de colonne à une variable !

III. Structure de contrôle dans PL/SQL

1. Structure conditionnelle IF

- **Objectif** : Test de condition simple
- **Syntaxe** :

```
IF <condition> THEN
<instruction(s)>
END IF;
```

Exemple 1: Augmentation du salaire de 15% si la date est postérieure au 09/09/2015

```
IF date > '09/09/2015' THEN
        Salaire := salaire*1,15;
END IF;
```

Exemple 2: Imbrication de plusieurs conditions - Augmentation du salaire de 15% de l'employé 'Isaac' si la date est postérieure au 17/11/2008

```
IF date> '17/11/2008' THEN
        IF employé=Isaac THEN
                salaire := salaire*1,15;
        END IF;
END IF;
```

2. Structure conditionnelle: IF THEN ELSE

- **Objectif** : Test de condition simple avec traitement de la condition opposée
- **Syntaxe** :

```
IF <condition> THEN
<instruction(s)>
ELSE
<instruction(s)>
END IF;
```

- **Remarque** : possibilité d'imbrication de plusieurs conditions

Exemple 1: Augmentation du salaire de 15% si la date est postérieure au 17/11/2008, de 5% sinon :

```
IF date> '17/11/2008' THEN
        salaire := salaire*1,15;
ELSE
        salaire := salaire*1,05;
END IF;
```

Exemple 2 :

- Augmentation du salaire de 15% de l'employé 'Isaac' si la date est postérieure au 09/09/2015 ou 10% sinon
- Augmentation pour les autres employés de 5%

```
IF employé = 'Isaac' THEN
    IF date > '09/09/2015 THEN
        salaire := salaire*1,15;
    ELSE
        salaire := salaire*1,10;
    END IF;
ELSE
    salaire := salaire*1,05;
END IF;
```

3. Structure conditionelles: IF THEN ELSEIF

- **Objectif** : Test de plusieurs conditions
- **Syntaxe** :

```
IF <condition> THEN
  <instruction(s)>
ELSIF <condition> THEN
  <instruction(s)>
ELSIF <condition> THEN
  <instruction(s)>
...
END IF;
```

Ex : Augmentation du salaire de 15% de l'employé 'Mohamed' de 10% de l'employé 'Isaac' et de 5% pour les autres.

```
IF employé='Mohamed' THEN
  salaire := salaire*1,15;
ELSIF employé=' Isaac' THEN
  salaire := salaire*1,10;
ELSE
  salaire := salaire*1,05;
END IF;
```

Structure conditionelles: CASE (1)

- **Objectif** : Test de plusieurs conditions
- **Différences avec IF-THEN-ELSE:**
  - Test de plusieurs valeurs en une seule construction
  - Définition de la valeur d'une variable
- **Syntaxe** :

```
CASE <variable>
  WHEN <expression 1> THEN <valeur 1>
  WHEN <expression 2> THEN <valeur 2>
  ...                 THEN <valeur n>
  END;
```

Ex : Affichage du club de football en fonction du nom de la ville (Ouagadougou – EFO, Bobo – RC Bobo, ..., autre – pas d'équipe) :

```
val:=CASE ville
    WHEN Ouagadougou THEN 'EFO'
    WHEN Bobo THEN 'RC Bobo'
   ...
   ELSE 'Pas d'équipe'
END;
dbms_output.put_line(val);
```

Remarque :

dbms_output.put_line(<chaîne de caractères>) = permet l'affichage en SQL du message en chaîne de caractères

4. Structure itérative : LOOP

- **Objectif** : Exécution à plusieurs reprises d'un groupe d'instructions
- **Syntaxe** :

```
LOOP
  <instruction1>;
  ...
END LOOP;
```

- **Exemple** : Incrémenter jusqu'à la valeur 10 un nombre initialisé à 0

```
val:=0;
LOOP
   val:=val+1;
IF (val=10) THEN
      EXIT;
 END IF;
END LOOP;
```

5. Structure itérative : WHILE

- **Objectif** : Exécution d'un groupe d'instructions jusqu'à vérification d'une condition
- **Syntaxe** :

```
WHILE <condition> LOOP
    <instruction1>;
    ...
END LOOP ;
```

Exemple : Incrémenter jusqu'à la valeur 10 un nombre initialisé à 0

```
val:=0;
WHILE var < 10 LOOP
  val:=val+1;
END LOOP;
```

6. Structure itérative FOR

- **Objectif** : Itérations d'un groupe d'instructions un certain nombre de fois
- **Syntaxe** :

```
FOR <variable d'itération> IN <borne inf>..<borne sup>
LOOP
  <instruction1>;
    ...
END LOOP;
```

Ex : Afficher les valeurs entières de 1 à 5

```
FOR compteur IN 1..5 LOOP
    dbms_output.put_line(compteur);
END LOOP;
```

Remarque : Possibilité de parcourir en ***sens inverse***

- **Syntaxe :**

```
FOR <variable d'itération> IN REVERSE <borne inf>..<borne sup> LOOP
    <instruction1>;
    ...
END LOOP;
```

Ex : Afficher les valeurs entières de 1 à 5 en sens inverse

```
FOR compteur IN REVERSE 1..5 LOOP
    dbms_output.put_line(compteur);
END LOOP;
```

7. Structure itérative: EXIT

- **Objectif** : Quitter une structure itérative
- **Syntaxe** :

```
<Structure itérative> LOOP
    <instruction1>;
    ...
    EXIT [WHEN <condition>]
    END LOOP;
```

Ex : Afficher les valeurs entières de 1 à 5 et interrompre la boucle si la valeur du compteur est égale à 3

```
FOR compteur IN REVERSE 1..5 LOOP
    dbms_output.put_line(compteur);
    EXIT WHEN compteur=3;
END LOOP;
```

IV. Curseurs

1. Objectif et définition d'un curseur

- **Objectif** : Récupération d'un résultat de requête sous forme d'une collection

Si une seule ligne :

select a,b into x,y from table where cle = 123

Récupération de l'unique ligne du résultat de la requête

Placement dans le couple de variables (x,y)

Si plusieurs lignes ???

Besoin d'un curseur …

- **Définition** : un **curseur** est une variable permettant d'accéder à un résultat de requête SQL représentant une collection (ensemble de nuplets).

- **curseur implicite** : créé et géré par le SGBD à chaque ordre SQL
- **curseur explicite** : créé et géré par l'utilisateur afin de pouvoir traiter un SELECT qui retourne plusieurs lignes.

2. Fonctionnement du curseur explicite

L'utilisation d'un curseur explicite nécessite 4 étapes :

1) Déclaration du curseur
2) Ouverture du curseur

   Remplissage en une seule fois par exécution de la requête

3) Traitements des lignes

   Récupération des lignes avec un parcours séquentiel du curseur par un pointeur logique

4) Fermeture du curseur

   Libération de la zone qui devient inaccessible

a. Déclaration du curseur

Dans la section DECLARE du bloc avec la syntaxe :

Syntaxe :

CURSOR nom_curseur **IS** instruction_select ;

Ex :

```
DECLARE
CURSOR dpt IS
SELECT ename, sal from emp where dptno = 10 ;
BEGIN
 ...
END ;
```

b. Ouverture du curseur

Après avoir déclaré le curseur, il faut l'ouvrir dans la section exécutable (BEGIN) afin de faire exécuter l'ordre SELECT :

OPEN nom_curseur ;

Conséquences :

- allocation mémoire du curseur
- analyse de l'instruction SELECT
- positionnement des verrous éventuels (dans le cas SELECT ... FOR **UPDATE**)

c. 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.

FETCH nom_curseur INTO liste_variables ;

- FETCH ramène une seule ligne
- pour traiter n lignes, prévoir une boucle

d. Fermeture du curseur

Après le traitement des lignes, pour libérer la place mémoire.

CLOSE nom_curseur ;

3. Exemple d'utilisation de curseur

Exemple 1:

```
DECLARE
    CURSOR dpt_10 IS
        SELECT ename, sal FROM emp WHERE deptno=10 ORDER BY sal ;
        nom emp.ename%TYPE ;
        salaire emp.sal%TYPE ;
        BEGIN
            OPEN dpt_10 ;
            LOOP
                FETCH dept_10 INTO nom, salaire ;
                IF salaire > 2500 THEN
                        INSERT INTO résultat VALUES (nom, salaire) ;
                 END IF;
                 EXIT WHEN salaire = 5000 ;
               END LOOP ;
              CLOSE dpt_10 ;
         END;
```

Ex 2 : Calculer la somme des salaires de tous les employés

```
   DECLARE
        CURSOR c IS
            SELECT * FROM Emp WHERE salaire > 1000;
        total INTEGER := 0;
BEGIN
   OPEN c ;
   FOR emp IN c LOOP
         total := total + emp.salaire;
    END LOOP;
    CLOSE c ;
END;
```

4. Attributs sur les curseurs

Indicateurs sur l'état d'un curseur :

- %FOUND :

- ✓ booléen VRAI si un n-uplet est trouvé

- %NOTFOUND :
  - ✓ booléen VRAI après la lecture du dernier n-uplet
- %ISOPEN :
  - ✓ booléen VRAI si le curseur est actuellement actif
- %ROWCOUNT :
  - ✓ numérique, retourne le nombre de n-uplets dans le curseur

Remarque :

Curseur implicite : SQL%FOUND, ….

Curseur explicite : nom_curseur%FOUND, …

Etats d'un curseur : Curseur implicite VS Curseur explicite

	SQL%…	nom_curseur%…
%FOUND	• **insert, update, delete : ≥ 1 ligne traitée** • **select …into : 1 seule ligne**	**dernier FETCH a ramené 1 ligne**
%NOTFOUND	**insert, update, delete , select …into… : 0 ligne**	**dernier FETCH n'a pas ramené de ligne**
%ISOPEN	**false (curseur toujours fermé après utilisation)**	**true si curseur ouvert**
%ROWCOUNT	• **insert, update, delete : nombre de lignes traitées** • **select… into … : valeur 0,1ou 2 si 0,1, ≥ 1 ligne(s) traitée(s)**	**nième ligne traitée par le FETCH**

- **Ecriture simplifiée : déclaration de variable (1)**

Déclaration implicite d'une structure dont les éléments sont d'un type identique aux colonnes ramenées par le curseur :

Syntaxe :

```
DECLARE
  CURSOR nom_curseur IS instruction_select ;
  nom_structure nom_curseur%ROWTYPE ;
BEGIN
  ...
  FETCH nom_curseur INTO nom_structure ;
```

- **Ecriture simplifiée: exemple**

Exemple :

```
DECLARE
      CURSOR c1 IS
          SELECT ename, sal+NVL(comm,0) saltot FROM emp ;
      c1_res c1%ROWTYPE ;
BEGIN
  OPEN c1 ;
  LOOP
     FETCH c1 INTO c1_rec ;
     EXIT WHEN c1%NOTFOUND ;
    IF c1_rec.saltot > 2000 THEN
       INSERT INTO temp VALUES (c1_rec.ename, c1_rec.saltot) ;
    END IF ;
END LOOP ;
  CLOSE c1 ;
END;
```

- **Ecriture simplifiée : curseurs et boucles**

```
DECLARE
CURSOR nom_c IS select …;
BEGIN
FOR nom_r IN nom_c
LOOP
/* traitement */
END LOOP ;
```

→

```
DECLARE
CURSOR nom_c IS select …;
nom_r nom_c%ROWTYPE ;
BEGIN
OPEN nom_c ;
LOOP
FETCH nom_c INTO nom_r ;
EXIT WHEN
nom_c%NOTFOUND ;
/* traitement */
END LOOP ;
CLOSE nom_c ;
```

Déclaration du curseur dans la boucle FOR :

```
FOR nom_rec IN (SELECT ...)
  LOOP
     traitement ;
  END LOOP ;
```

Syntaxe qui évite la déclaration du curseur dans un DECLARE.

- **Curseur paramétré**

Pour utiliser un même curseur avec des valeurs différentes, dans un même bloc PL/SQL :

```
DECLARE
      CURSOR nom_c (par1 type, par2 type,...) IS ordre_select;
BEGIN
   OPEN nom_c(val1, val2,...) ;
```

- Type : char, number, date, boolean SANS spécifier la longueur
- Passage des valeurs des paramètres à l'ouverture du curseu

Exemple :

```
DECLARE
        CURSOR grosal IS SELECT DISTINCT sal FROM emp ORDER BY sal desc ;
        CURSOR c1(psal number) IS
            SELECT ename, sal, empno FROM emp WHERE sal = psal;
BEGIN
   FOR vgrosal IN grosal LOOP
     EXIT WHEN grosal%rowcount > &Nombre ;
      -- Nombre est une variable SQL*Plus
     FOR employe IN c1(vgrosal.sal) LOOP
       INSERT INTO resultat VALUES ( employe.sal, employe.ename || ' de
          numero : ' || to_char(employe.empno) ) ;
   END LOOP;
  END LOOP;
END;
```

5. Déclaration "current of"

- **Objectif** :
  - Modification via SQL sur le n-uplet courant
  - Permet d'accéder directement en modification ou suppression du nuplet récupéré par FETCH
- **Syntaxe** (pour un curseur c) :

```
CURSOR nomCurseur ... FOR UPDATE
  ... <opération> WHERE CURRENT OF nomCurseur
```

Remarques :

- Ne pas oublier de déclarer « FOR UPDATE »
- Il faut au préalable réserver les lignes lors de la déclaration du curseur par un verrou d'intention :

(SELECT ... FOR UPDATE [OF nom_colonne,...]).

Exemple 1: Augmenter les employés dont le salaire est supérieur à 1500 €

```
DECLARE
    CURSOR c1 IS SELECT ename, sal FROM emp FOR UPDATE OF sal ;
BEGIN
    FOR c1_rec IN c1
   LOOP
     IF c1_rec.sal > 1500 THEN
        INSERT INTO resultat VALUES (c1_rec.ename, c1_rec.sal * 1.3);
        UPDATE emp SET sal = sal * 1.3 WHERE CURRENT OF c1;
     END IF;
  END LOOP;
END;
```

Exemple 2: Augmentation de salaire des employés peu rémunérés (employés dont le salaire est inférieur à 1500 €)

```
DECLARE
      CURSOR c IS SELECT * FROM emp FOR UPDATE;
BEGIN
    FOR e IN c LOOP
        IF e.salaire < 500.0 THEN
            UPDATE emp SET salaire = salaire*1.2 WHERE CURRENT OF c;
        END IF;
   END LOOP;
END;
```