Chapitre I : Introduction au langage PL/SQL sous Oracle

Introduction : La plupart des SGBD relationnels implémente le langage SQL pour l'implémentation, la manipulation et le contrôle des bases de données relationnelles. Cependant, SQL ne permettant pas une programmation avancée. C'est ainsi que des fonctionnalités supplémentaires y sont ajoutées pour permettre de faire de la programmation procédurale dans les SGBD relationnels. Plusieurs langages sont issus de l'amélioration de SQL : PL/SQL (Procedural Language SQL) sous Oracle et MySQL, Transact SQL sous SQL Server etc. Ce chapitre donne les bases du le PL/SQL tel que implémenté sous Oracle.

I. La structure d'un code PL/SQL

Un programme PL/SQL est composé généralement de trois parties : L'entête, les déclarations et le corps du programme.

I. 1. Les déclarations

La première partie contient les déclarations : variable, constante, curseur, enregistrement etc.

I. 2. Le corps du programme

Le corps d'un programme PL/SQL est ouvert par le mot-clé **Begin** et fermé par le mot-clé **End**. Toutes les instructions à exécuter dans le programme doivent être entre ces deux mots-clés.

Remarque : Pour exécuter un programme PL/SQL on ferme le code par le caractère "/".

Exemple

```
Declare
    var_1          Type1 ;
    var_2          Type2 ;
    enreg          NomObjet%Rowtype ;
Begin
    Instruction_1 ;
    Instruction_2 ;
End ;
/
```

II. Les bases du langage PL/SQL Oracle

II. 1. Les variables et les constantes

En PL/SQL sous Oracle toute variable ou constante doit être déclarée avant son utilisation.

II. 1. 1. Les variables

La déclaration d'une variable se fait à l'aide du mot-clé ***Declare***.

La syntaxe de la déclaration d'une variable est la suivante :

Declare variable Type ;

Si plusieurs variables doivent être déclarées, on utilise une seule fois le mot-clé ***Declare***

```
Declare
        variable_1      Type_1 ;
        variable_2      Type_2 ;
```

II. 1. 2. Les constantes

La déclaration d'une constante se fait également avec le mot-clé ***Declare***. De plus entre l'identificateur de la constante et son type on utilise le mot-clé ***Constant***. La syntaxe de la déclaration d'une constante est la suivante :

```
Declare
        id_constante    Constant        Type := valeur ;
```

II. 1. 3. L'affectation d'une valeur à une variable

L'affectation d'une valeur à une variable peut se faire de différentes manières en PL/SQL :

1. Initialisation lors de la déclaration :

   Declare id_variable type **Default** Valeur ;
2. Affectation d'une valeur à une variable déjà déclarée :

   id_variable **:=** valeur ;
3. Affectation du résultat d'une requête à une variable :

   Select Attribut **Into** id_variable **From** Table [**Where** condition(s)] ;

Remarque : Il faut veiller à ce que le résultat de la requête puise être affecté à la variable.

Exemple

```
Declare     x     integer Default 78 ;
            y     char(6) ;
            z     date ;
Begin
      y := 'IA2451' ;
      Select DateNaiss Into z From Etudiant Where Matricule = y ;
End ;
/
```

II. 2. Les opérateurs

De la même manière que les autres langages de programmation le PL/SQL permet l'utilisation d'opérateurs pour effectuer des calculs ou des tests :

- Arithmétiques : + ; - ; * ; / ;
- Comparaison : = ; <> ; <= ; >= ; < ; >
- Booléens : AND ; OR ; NOT
- Ensemblistes : IS NULL ; LIKE ; BETWEEN ; IN ; ANY

II. 3. Les commentaires

Pour donner des explications dans un code PL/SQL, il est possible d'y intégrer des commentaires pour faciliter la compréhension du code. Il existe deux manières de commenter un code PL/SQL :

- -- Commentaires sur une seule ligne
- /* Commentaire sur
  plusieurs lignes */

II. 4. L'affichage à l'écran

L'affichage à l'écran se fait avec ***dbms_Output.put_Line*** selon la syntaxe :

dbms_Output.Put_Line (id_variable) ;
dbms_Output.Put_Line ('Texte à afficher') ;

Remarque : Pour afficher plusieurs valeurs : le contenu d'une variable et une chaine littérale on utilise l'opérateur de concaténation ||.

Exemple

```
Declare
    x       Char(8) Default ‘Alphabet’ ;
    y       Varchar(25) Default 'Une chaine de caractères' ;
Begin
    dbms_Output.Put_Line (x) ; -- Affiche : Alphabet
    dbms_Output.Put_Line (y) ; -- Affiche : 'Une chaine de caractères
    dbms_Output.Put_Line (‘Affichage d’’une chaine’) ; -- Affiche : Affichage d’une chaine
    dbms_Output.Put_Line (‘La valeur de x est ’ || x) ; -- Affiche : La valeur de x est Alphabet
End ;
/
```

III. Les structures

Pour programmer des instructions qui s'exécutent suivant une condition ou pour répéter l'exécution d'une instruction ou d’un groupe d’instructions, PL/SQL implémente des structures de choix et des boucles.

III. 1. Les instructions de choix

En PL/SQL (sous Oracle) nous avons les instructions If et Case dont les syntaxes sont données ci-dessous.

III. 1. 1. L'instruction IF

La syntaxe de l'instruction If est :

```
If  condition  Then
    Instruction(s) ;
[ElsIf  condition Then
    Instruction(s) ;
Else
    Instruction(s) ;]
End If ;
```

Remarque : A noter que si **Else** doit être suivi de **If**, il faut l’écrire **ElsIF** (sans e à **Else** et sans espace entre **Els** et **If**)

III. 1. 2. L'instruction Case

```
Case  id_variable
    When Valeur_1 Then Instruction(s) ;
    - - - - - - - - -
    When Valeur_n Then Instruction(s) ;
    [Else     Instruction(s) ;]
End Case ;
```

```
Case
    When Condition_1 Then Instruction(s) ;
    - - - - - - - - - - -
    When Condition_n Then Instruction(s) ;
    [Else       Instruction(s) ;]
End Case ;
```

Exemple

```
Declare
    x       Varchar(26) ;
    y       Varchar(40) ;
    z       Integer Default 0 ;
Begin
    Select Filiere Into x From Etudiant Where Matricule = 'IA0514' ;
    If x = 'Lettre moderne' then
            y := 'Vous êtes un étudiant de LM (UFR LASHU)' ;
    ElsIf x = 'Langue étrangère appliquée' Then
            y := 'Vous êtes un étudiant de LEA (UFR LASHU)' ;
    Else
            y := 'Vous n"êtes pas étudiant de l"UFR LASHU' ;
    End If ;
    dbms_Output.Put_Line(y) ;
End ;
/
```

III. 2. Les instructions iteratives

```
While Condition Loop
    Instruction(s) ;
End Loop ;
```

```
Loop
        Instruction(s) ;
        Exit [When Condition] ;
End Loop ;
```

For compteur **In [Reverse]** Val_Min .. Val_Max **Loop**
    Instruction (s) ;
End Loop ;

Exemple

```
Declare
    a       Integer Default 0 ;
Begin
    While a < 5 Loop
            a := a + 1 ;
    End Loop ;
    dbms_Output.Put_Line(a) ;
End ;
/
```

```
Declare
        c       Integer Default 0 ;
Begin
        Loop
                c := c + 5 ;
                If c >= 35 Then
                        Exit ;
                End If ;
        End Loop ;
        dbms_Output.Put_Line(c) ;
End ;
/
```

```
Declare
    i       Integer ;
    j       Integer ;
Begin
    For i In 15 .. 20 Loop
            Dbms_Output.Put_Line ('i = ' || i) ;
    End Loop ;
    For j In Reverse 20 .. 15 Loop
            Dbms_Output.Put_Line ('j = ' || j) ;
    End Loop ;
End ;
/
```

Remarque : La commande **Exit When** permet d'arrêter l'exécution de la boucle dans laquelle elle est placée ;

IV. Les Triggers ou déclencheurs

IV. 1. Qu'est-ce qu'un trigger ?

Un trigger est un objet de base de données associé à une table. Son exécution est déclenchée par la survenance d'une action bien déterminée.

Les triggers doivent respecter les conditions suivantes :

- Un trigger ne peut pas être placé sur une vue ou sur une table temporaire ;
- Deux triggers de même instant et même événement ne peuvent porter sur la même table ;
- Il est nécessaire d'être super-utilisateur pour créer, modifier ou supprimer un trigger ;
- Si l'exécution d'un trigger dont l'instant est BEFORE échoue, l'exécution de la requête qui l'a appelé échoue également ;
- La seule manière d'interrompre l'exécution d'un trigger est de provoquer une erreur en son sein.

IV. 2. Création de triggers

IV. 2. 1. Syntaxe de création

La syntaxe de la création d'un trigger est :

Create TRIGGER Nom Instant Événement
On Nom_Table **For Each Row**
Declare
    V1 Type ;
    V2 Type ;
Begin
    Instruction (s) ;
End ;

Un trigger est appelé de manière automatique selon son événement et son instant.

Remarques

i. **Instant** indique le moment où le trigger doit être appelé. Il prend les valeurs **Before** (avant l'exécution de la requête qui appelle le trigger) ou **After** (après l'exécution de la requête qui appelle le trigger) ;
ii. **Événement** donne l'opération qui déclenche l'exécution du trigger. Les opérations possibles sont **Insert**, **Update** et **Delete**.

IV. 2. 2. Les mots clés Old et New

Dans le corps d'un trigger, on peut faire référence aux colonnes de la table associée au trigger en utilisant les mots clés **Old** et **New**.

- **:Old.*Col_name*** fait référence à une colonne d'un enregistrement déjà inséré dans la base avant sa modification ou sa suppression ;
- **:New.*Col_name*** fait référence à une colonne d'un enregistrement qui sera inséré après l'exécution du trigger.

Remarques

- L'utilisation de **:New.*Col_name* = *value*** requiert le droit **UPDATE** sur la colonne ;
- L'utilisation de ***value* = :New.*Col_name*** requiert le droit **SELECT** sur la colonne ;
- La commande **CREATE TRIGGER** requiert le droit **SUPER**.

Exemple

```
Create Trigger ControleAge Before Insert
On Etudiant For Each Row
Declare
        n integer ;
Begin
        n := :New.Age ;
        If n > 50 or n < 15 then
            :New.Age := Null ;
        End If ;
End ;
/
```

V. Les routines

Une routine est un ensemble d'instructions regroupées dans un bloc pouvant des arguments en entrée et retourner un résultat. Elle sert essentiellement à créer de petits programmes dont on a souvent besoin dans les tâches d'administration. Ces programmes sont stockés dans la base et permettent d'automatiser l'exécution de tâches dont on a souvent besoin. Ils permettent également de diminuer l'envoi de messages entre les utilisateurs et le serveur. Ils améliorent alors les

performances du serveur. Nous allons donner dans ce qui suit les fonctions et les procédures stockées. La différence entre une fonction et une procédure est que la première renvoie un résultat à la fin de son exécution, alors la dernière ne renvoie pas de valeur.

V. 1. Les procédures stockées

V. 1. 1. Qu'est-ce qu'une procédure stockée ?

Une **procédure stockée** (***stored procedure***) est un ensemble d'instructions SQL précompilées, enregistrées sur le serveur, directement dans la base de données. Elle peut être exécutée sur demande : lancées par un utilisateur, un administrateur DBA ou encore de façon automatisée par un trigger.

Les requêtes envoyées à un serveur SQL font l'objet d'une *analyse syntaxique* puis d'une *interprétation* avant d'être *exécutées*. Ces étapes sont très lourdes si l'on envoie plusieurs requêtes complexes très souvent. Les procédures stockées permettent de diminuer le temps d'exécution car elles sont envoyées une seule fois sur le réseau puis analysées, interprétées et stockées sur le serveur **sous forme exécutable (pré-compilée)**. A chaque fois qu'un utilisateur envoie une requête avec le nom de la procédure, son code est exécuté.

On peut aussi passer des paramètres à une procédure stockée lors de son appel, et recevoir le résultat de ses opérations comme celui de toute requête SQL.

V. 1. 2. Syntaxe de création de procédures stockées

La syntaxe de création d'une procédure stockée est :

```
Create Procedure Nom_Procedure(id_variable IN|OUT|INOUT Type)
As
        v1      Type ;
        v2      Type ;
Begin
        Instruction(s) ;
End ; /
```

Une procédure est appelée avec la commande **Exec**. La syntaxe de l'appel d'une procédure est :

```
Begin
        Exec Nom_Procedure(Valeur_Arguments) ;
End ; /
```

Exemple :

```
Create Procedure Affichage(a In integer, b Out varchar)
As
    x integer default 0 ;
Begin
    x := 5 * a ;
    If x < 5 Then
        b := 'Vous avez donné une valeur plus petite que 1' ;
    ElsIf x > 5 Then
        b := 'Vous avez donné une valeur plus grande que 1' ;
    End If ;
End ;
/

Declare
    y integer Default 6 ;
    z varchar(45) ;
Begin
    Exec Affichage(y, z) ;
    dbms_Output.Put_Line (z) ; -- Affiche "Vous avez donné une valeur plus grande que 1"
End ;
/
```

Un argument peut être en entrée (In), en sortie (Out) ou en entrée et sortie (InOut).

- Un argument en entrée permet de donner une valeur qui sera utilisée lors de l'exécution des instructions de la procédure ;
- Un argument en sortie permet de garder une valeur obtenue lors de l'exécution de la procédure.

V. 2. Les fonctions

V. 2. 1. Syntaxe de création de Fonctions

La syntaxe de création d'une fonction est :

```
Create Function Nom_Fonction(Liste arguments avec leur type) Return Type_Retour
Is
        v1      Type ;
        v2      Type ;
Begin
        Instruction(s) ;
        Return Variable_Retour ;
End ;
```

Une fonction est appelée comme suit :

```
Declare
        Id_Var          Type ;
Begin
        id_Var := Nom_Fonction(Valeur_Arguments) ;
        dbms_Output.Put_Line (id_Var) ;
End ;
/
```

V. 2. 2. Quelques fonctions utiles

Il existe des fonctions très utiles permettant de résoudre des problèmes rencontrés dans l'administration d'une base de données :

a. Concat(liste de chaines à concaténer) : Elle permet de concaténer deux ou plusieurs chaines de caractères. Elle permet aussi de concaténer une chaine et un entier.

b. SUBSTR(var, debut, nb_caracteres) : Elle permet d'extraire une chaine d'une chaine de caractères. Dans cette syntaxe, **var** est la variable contenant la chaine dans laquelle on veut extraire la sous-chaine, **debut** et la position du premier caractère à extraire et **nb_caracteres** est le nombre de caractères à extraire.

c. Current_Date / SysDate : Ces fonctions renvoie la date actuelle sous le format : JJ/MM/AA. **SysDate** donne la date du système sous le même format que **Current_Date**.

Exemple

```
Begin
    dbms_Output.Put_Line (Current_Date) ; /* Affiche 13/03-17 */
End ; /
```

VI. Les enregistrements et les curseurs

VI. 1. Les cureurs

Un curseur est une zone mémoire utilisée par le moteur de base de données Oracle pour analyser et interpréter tout ordre SQL. Un curseur est une structure permettant de stocker des enregistrements. Les curseurs sont très utiles pour garder le résultat des requêtes qui ramènent plusieurs enregistrements.

Il existe deux types de curseur :

- Les curseurs implicites créés, utilisés et gérés par Oracle pour traiter les ordres SQL ;
- Les curseurs explicites créés et utilisés par l'utilisateur pour exploiter le résultat d'un ordre SQL.

Tous les curseurs doivent utilisés dans le corps d'une routine doivent être déclarés dans la partie des déclarations.

La syntaxe de création d'un curseur :

```
Cursor Nom_Cursuer Is Requete ;
```

Pour exploiter un curseur, on doit l'ouvrir, le parcourir et le fermer à la fin.

```
Open Nom_Cursor ;
    FETCH Nom_Curseur Into Liste_Variables ;
Close Nom_Cursor ;
```

Remarque : FETCH ramène une seule ligne à la fois, pour traiter plusieurs lignes, il faut une boucle

Il est possible d'utiliser la boucle **For** pour exploiter un curseur. Cette manière de faire est plus simple dans la mesure où, cette boucle permet de :

- Déclarer implicitement la variable de parcours ;

- Ouvrir le curseur ;
- Réaliser les FETCH successifs ;
- Fermer le curseur.

Exemple

```
Declare
    Cursor Test Is Select * From Salle where Batiment = 'BP' ;
Begin
    For i in Test
    Loop
        Dbms_Output.Put_Line(i.Numero || ' ' || i.Capacite || ' ' || i.Type || ' ' || i.Climatise) ;
    End Loop ;
End ;
/
```

VI. 2. Les enregistrements

Un enregistrement est une structure correspondant à une ligne d'une table ou d'un curseur.

La syntaxe de la déclaration d'un enregistrement est :

```
Nom_Record        NomObjet%Rowtype        ;
```

Exemple

```
Declare
    Lig_E1    Etudiant%Rowtype ;
    Cursor    Liste_Etu Is Select Numero, Nom, Prenom From Etudiant Where Age >= 27 ;
    Lig_E2    Liste_Etu%Rowtype ;
Begin
    Open Liste_Etu ;
    Loop
        Fetch Liste_Etu Into Lig_E2 ;
        Exit When Liste_Etu%NotFound ;
        Dbms_Output.Put_Line (Lig_E2.Numero || ' ' || Lig_E2.Nom || ' ' || Lig_E2.Prenom) ;
    End Loop ;
    Close Liste_Etu ;
```

End ;

/