Memento - PL/SQL

1. Généralités

PL/SQL veut dire « Procedural Language extensions to SQL ». C'est une extension de SQL : des requêtes SQL (SELECT, DELETE, INSERT et UPDATE) cohabitent avec les structures de contrôle habituelles de la programmation structurée (blocs, alternatives, boucles)
PL/SQL est un langage propriétaire de Oracle. Ce n'est un langage autonome ; il est utilisé à l'intérieur d'autres produits Oracle.

PL/SQL peut être utilisé pour l'écriture des procédures stockées et des triggers. Il convient aussi pour écrire des fonctions utilisateurs qui peuvent être utilisées dans les requêtes SQL (en plus des fonctions prédéfinies)
Les blocs ou procédures PL/SQL sont compilés et exécutés par le moteur PL/SQL. Ce moteur est intégré au moteur de la base de données et dans un certain nombre d'outils (Forms, Report).

2. Structure d'un programme

Un programme est structuré en blocs d'instructions de 3 types :

- Blocs anonymes
- procédures
- fonctions

Les blocs anonymes (ou externe) ne peuvent être appelés.
Les procédures et les fonctions sont stockées dans la base de données .

2.1 Structure d'un bloc

Il y a trois sections distinctes dans un bloc : DECLARE, BEGIN et EXCEPTION. Un bloc se termine par END ;
Un bloc PL/SQL peut contenir un autre dans la section BEGIN. Chaque variable a une portée limitée par le bloc dans lequel elle est déclarée.

```
DECLARE
-- déclarations de variables, de constantes, de types
BEGIN
-- Les instructions à exécuter
EXCEPTION
-- Traitements des exceptions ou erreurs
END;
```

La section DECLARE ne peut pas contenir d'instructions exécutables. Toutefois, il est possible de définir dans cette section des procédures ou des fonctions contenant une section exécutable.
Toute variable doit être déclarée avant d'être utilisée dans la section exécutable.
La section de gestion des erreurs (facultative) débute par le mot clé EXCEPTION. Elle contient le code exécutable mis en place pour la gestion des erreurs. Lorsqu'une erreur intervient dans l'exécution, le programme est stoppé et le code erreur est transmis à cette section.

Seule la section BEGIN est obligatoire. Chaque instruction se termine par **;**
-- : commentaire sur une seule ligne
/* */ : commentaire sur plusieurs lignes

3. Types de données

3.1 Type prédéfinis:

- NUMBER : entiers et réels
- BINARY_INTEGER, PLS_INTEGER : entiers
- CHAR, LONG, VARCHAR2 : chaînes de caractères
- BOOLEAN :
- DATE, TIMESTAMP : date, date et heure
- LOB : objet de grande taille
- BLOB : objet de grande taille codé en binaire
- CLOB : objet de grande taille composé de caractères
- BFILE : pointeur sur un objet de grande taille codé en binaire ou en caractères

Ex : compte Number ;
Societe Constant char(12) := 'LOGICIEL SGBD' ;
Nom Char(30) NOT NULL := 'KAWTHAR' ;
AnnivDate Date :=ToDate('12/JAN/1970', 'DD/MON/YYYY') ;
DateJour Date :=SYSDATE ;

3.2 Attributs %TYPE et %ROWTYPE

%TYPE : permet de déclarer une variable du même type qu'une variable déjà définie ou qu'une colonne d'une table de la base de données

Ex : prix Number(5,2) ;
PrixSolde prix%TYPE ;

%ROWTYPE : permet de déclarer un enregistrement dont les champs ont le même type qu'un tuple de la BD

Ex : employe emp%ROWTYPE ;

3.3 Types composés

TABLE : ensemble de valeurs de même type, similaire à un vecteur

Ex : Declare
Type Equipe_type Table of Varchar2(35) index by Binary_Integer ;
groupe Equipe_type ;
Begin
groupe(1) := 'lion' ;
groupe(10) :='tigre' ;

```
End ;
```

RECORD : enregistrement avec des types hétérogènes, similaire à struct de C

```
Ex : Declare
     Type adresse_type is Record of (
                    No number(4) ;
                    Rue varchar2(45) :=  null ;
                    Ville varchar2(40)  ) ;
     Adresse adresse_type ;
     Type Client_type  is record of (
          NomC varchar2(50) ;
          Id  Client.NumCli%TYPE ;
          Adresse adresse_type ) ;
      client Client_type ;
      Begin
         Select Nom, Ville into client.NomC, client.Adresse.Ville from client where
         ClientNum = 415 ;
       End ;
```

Variables hôtes

Ce sont des variables définies en dehors du bloc PL/SQL (par exemple dans SQL Plus). Elles sont alors préfixées par **:** ou par **&** dans le bloc PL/SQL.

Ex :

```
Variable g_mensuel Number
Define p_annuel = 5000
Declare
Sal_mens Number (7,2) := &p_annuel ;
Begin
:g_mensuel := sal_mens /12 ;
end ;

/
print g_mensuel
```

p_annuel est une variable de type chaîne de caractères (dans ce cas elle contient les caractères 5, 0, 0 ,0)

4 Structures de Contrôle

4.1 Sélection

if then else

```
IF condition1 THEN
-- Instructions
ELSIF condition2 THEN
-- Instructions
ELSE
-- Instructions
END IF;
```

Le ELSE final permet de traiter les cas non traités

```
Ex :  Declare
        Qte_vendue Number(5) :=0 ;
        NoArticle produit.NumProd%TYPE := 59;
        Begin
        Select sum(quantite) into Qte_vendue from commande where NumProd = NoArticle ;
        If  Qte_vendue >10 Then
           Update produit Set PrixUni = 1.1*PrixUni  where NumProd = NoArticle ;
        Elsif Qte_vendue > 20 Then
           Update produit Set PrixUni = 1.2*PrixUni  where NumProd = NoArticle ;
        End if ;
        Commit;
        END ;
```

CASE

```
Case selecteur
   When exp1 Then instruction1
   When exp2 Then instruction2
   …
   When expN Then instructionN
   [else instructions]
   End ;
```

Ou bien

```
Case
   When condition1 Then instruction1
   When condition 2 Then instruction2
   …
   When condition N Then instructionN
   [else instructions]
   End ;
```

```
Ex : Declare
       Grade Char(1) := 'a' ;
       Appreciation char(15) ;
       Begin
```

```
Appreciation :=
   Case grade
      When ‘a’ then ‘Très bien’ ;
       When ‘b’ then ‘bien’ ;
       When ‘c’ then ‘Moyen’ ;
       When ‘d’ then ‘Mediocre’ ;
       Else ‘Grade inexistant’ ;
   End ;
End ;
```

4.2 Itérations

Il y a trois type de boucles : Loop, For et while

LOOP:

```
Loop
Instructions ;
[Exit when condition ;]
End Loop ;
```

```
Ex : Declare indice Number(3) :=0;
     Begin
     Loop
     Dbms_output.put_line(‘ok’|| To_char(indice) ;
     Indice := indice +1 ;
     Exit when indice = 10 ;
     End loop ;
     End ;
```

FOR :

```
For compteur in [reverse] min.. max Loop
    Instructions ;
End loop ;
```

La variable compteur n’a pas besoin d’être déclarée (entier implicite). Sa valeur initiale est min et sa valeur finale est max

```
Ex : declare
     max_id number(2) ;
     begin
     select max(NumClient) into max_id from Client
     for compt in 1..3 Loop
         max_id := max_id +compt ;
         insert into temp (NumClient, ville) values (max_id, ‘Rabat’) ;
     end loop ;
     end ;
```

WHILE :

```
While condition loop
    Instructions ;
End loop ;
```

```
Ex : declare
       k number(3) :=0 ;
       begin
        while k<25 loop
          delete from temp where NumClient = k ;
           k :=k+5 ;
         end loop ;
        end ;
```

4.3 Branchements

Permet de se brancher au moyen d'une étiquette. A utiliser avec modération

```
Ex :
Begin
<<test>> a :=a+ 5 ;
             if a> b then
                b := b+c ;
             GOTO test ;
             End if ;
End ;
```

5 Curseurs

A l'exécution d'une requête SQL, le serveur Crée un espace en mémoire qui contient les résultats de la requête. Cet espace est appelé curseur. Ce dernier permet d'accéder à l'ensemble des tuples résultats. Il y a deux types de curseurs : implicite et explicite.

Curseur implicite : associé automatiquement à toute requête Select into. Ce dernier n'a pas besoin d'être déclaré car il est généré automatiquement (et fermé automatiquement). Ce curseur implicite ne peut contenir qu'un seul tuple

Curseur explicite : utilisé pour exécuter une requête SQL qui retourne plusieurs lignes. Le curseur doit alors être explicitement déclaré.

5.1 Etapes

Les étapes à suivre pour un curseur explicite sont les suivantes :

1. Déclarer le curseur ;
2. Ouvrir le curseur ;

3. Obtenir un tuple et le réaliser le traitement
4. Fermer le curseur

Déclaration du curseur : Declare Cursor nom_curseur IS clause_select

La clause_select ne doit pas contenir into

Ouverture du curseur : OPEN nom_curseur ;

L'ouverture du curseur lance l'exécution de la requête associée. Les tuples sont alors stockés en mémoire. Le curseur pointe alors sur le premier tuple.

Accès aux tuples : FETCH nom_curseur into [var1, var2,…] | [nom_enregistrement]

Fermeture du curseur : CLOSE nom_curseur ;

L'espace alloué est alors libéré

5.2 Attributs d'un curseur

Ce sont des propriétés liées au curseur :

- %NOTFOUND : propriété false si un tuple est obtenu
- %FOUND : propriété false si aucun tuple n'est obtenu
- %ROWCOUNT : fournit le nombre de tuples impliqués dans la requête
- %ISOPEN : permet de vérifier si le curseur est disponible

Ex :
```
declare
  cursor curs1 IS select sum(Quantite), NumProd from commandes group by NumProd;
  qte commandes.Quantite%TYPE ;
  nop commandes.NumProduit%TYPE ;
  Type liste_type Table of commandes.NumProduit%TYPE;
  Liste liste_type ;
  i binary_integer ;
  begin
  open curs1 ;
  loop
  fetch curs1 into qte, nop ;
  exit when curs1%NOTFOUND ;
  if qte > 10 then
     liste(i) = nop ;
  end if ;
  end loop ;
  end ;
```

Les mêmes attributs sont disponibles pour les curseurs implicites. Ils sont préfixés par SQL.

- %NOTFOUND : propriété vraie si la clause SQL n'affecte aucun tuple
- %FOUND : propriété vraie si la clause SQL affecte au mois un tuple
- %ROWCOUNT : fournit le nombre de tuples impliqués dans la requête
- %ISOPEN : toujours fausse car curseur implicite est immédiatement fermé

Ex :

```
Delete from Produit where NumFour = 120 ;
If SQL%NOTFOUND then
   Dbms_output.put_line('aucun produit pour ce fournisseur') ;
End if ;
```

Boucle FOR pour curseur

La boucle FOR pour curseur peut être utilisée pour simplifier l'usage du curseur

```
FOR nom_tuple IN nom_curseur LOOP
      Instructions ;
END LOOP ;
```

Cette boucle réalise implicitement l'ouverture du curseur avant l'itération et le Fetch à chaque itération. La boucle se termine automatiquement s'il n'y a plus de tuple (Close implicite).
Nom_tuple est une variable enregistrement de même type que la table. Elle n'a pas besoin d'être déclarée.

Il est également possible de spécifier une requête au lieu du curseur :

```
FOR nom_tuple IN (clause_select) LOOP
      Instructions ;
END LOOP ;
```

Curseur Paramétré

Déclaration : CURSOR nom_curseur(param1 Type,..) IS …

Type : type sans la taille

Ouverture : OPEN nom_curseur(val1, ..) ;

Curseur pour Mise à jour et CURRENT OF

Permet de verrouiller des enregistrements pour réaliser une clause DML (delete ou update).

Select ….
From ….
For UPDATE [OF nom_col] [NOWAIT] ;

Avec l'option NOWAIT, le système retourne une erreur si les enregistrements sont déjà verrouillés par une autre session. Sinon, il y a blocage jusqu'à déverrouillage.

Ex : Declare
curs1 IS select NumProd, PrixUni from Produit FOR UPDATE ;
no Produit.NumProduit%ROWTYPE ;
prix Produit.PrixUni%ROWTYPE ;
begin
open curs1 ;
loop
fetch curs1 into no, prix ;
exit when curs1%NOTFOUND ;
if NumProd > 500 then
delete from Produit where CURRENT of curs1 ;
end if ;
end loop ;
close curs1 ;
end ;

6 Gestion des erreurs

Si un bloc PL/SQL contient une routine de gestion des erreurs, alors il y aura branchement à cette routine en cas d'erreur. Deux types d'exceptions existent :

- Implicites : prédéfinies par le système ;
- Explicites : définies par l'utilisateur.

Les exceptions prédéfinies sont automatiquement levées quand l'erreur correspondante se produit.
Les exceptions utilisateurs doivent être déclarées et levées explicitement (RAISE).
Les exceptions doivent être traitées dans la section exception du blocPL/SQL

Exception
WHEN excep1 [or except3] then
Instructions ;
[WHEN except2 then
Instructions ;]
[WHEN OTHERS then
Instructions ;]

Quelques exceptions prédéfinies :

Exception	No	Description
CURSOR_ALREADY_OPEN	ORA-06511	Ouverture d'un curseur déjà ouvert
INVALID_CURSOR	ORA-011001	Opération invalide sur un curseur
NO_DATA_FOUND	ORA-01403	Select.. ne retourne aucun tuple
TOO_MANY_ROWQ	ORA-01422	Select ..into clause retourne plus d'un tuple
ZERO_DIVIDE	ORA-01476	Divivsion par 0
DUP_VAL_ON_INDEX	ORA-00001	Tentative d'insertion de valeurs dupliquées

Ex :

```
Declare
     Prix Produit.PrixUni%TYPE ;
    trop_bas  EXCEPTION ;
    val_nulle EXCEPTION ;
Begin
   Select PrixUni Into Prix from Produit where NumProd = 110 ;
   If PrixUni < 10 then
      Raise trop_bas ;
   Elsif PrixUni is NULL then
      Raise val_nulle ;
   End if ;
 Exception
  When NO_DATA_FOUND Then
    Dbms_output.put_line('Pas de produit portant le no 10') ;
 When trop_bas Then
   Update produit set PrixUni = 2* PrixUni where NumProd = 110 ;
  When val_nulle Then
    Dbms_output.put_line('Valeur nulle') ;

End ;
```

7 Procédures et fonctions

Sous-programmes sont des blocs PL/SQL nommés qui peuvent être appelés à partir de n'importe quel bloc ou sous-programme PL/SQL. Le sous-programme peut être paramétré.
Un sous-programme peut être :

- une fonction quand il retourne un résultat et un seul
- une procédure quand il retourne zéro ou plusieurs résultats

La procédure ou la fonction est stockée dans la base de données et peut être réutilisée
L'appel dans SQLPlus est réalisé avec execute :

7.1 Procédure

Les paramètres peuvent être de mode IN ou OUT ou IN OUT.

```
Create [or Replace] Procedure  nom_proc [(param1 mode type, param2 mode type,..)]  IS |AS
[déclarations
…]
Begin
…
[Exception
…]
End [nom_proc];
```

Le type de paramètre ne doit pas comporter la taille.
Dans le mode IN, les paramètres sont passés par valeur et dans le mode OUT ou IN OUT, les paramètres sont passés par adresse.

Ex :

```
Create or replace query_prod ( p_no IN number,
                                p_des OUT varchar2) IS
Begin
Select desi into p_des from produit where NumProd= p_no ;
End query_prod ;
```

7.2 Fonction

Les paramètres sont de mode IN.

```
Create [or Replace]Function  nom_fonc [(param1 type, param2  type,..)]  return type IS |AS
[déclarations
…]
Begin
…
[Exception
…]
End [nom_proc];
```

Ex :

```
Create function max_qte(p_noclnumber) Return number is
    Qte number(4) ;
    Begin
       Select max(quantite) into qte from commande where NumCli = p_nocl ;
Return qte ;
End ;
```

Ex2 : Appel de fonction

```
Declare
 Cursor curs_client is select NumCli, Nom, Ville from Client ;
 Qte_cli number(4) ;
Begin
For enr_client in curs_client loop
 Qte_cli :=max_qte(enr_client.NumCli) ;
End loop ;
End ;
```

Suppresion :

Drop nom_fonc | nomproc

8 Packages

Le langage PL/SQL offre l'organisation en package qui permet de regrouper un ensemble de procédures, de fonctions et d'items (curseurs, exceptions, constantes, variables) connexes. Le package correspond à un module fonctionnel et offre une interface pour l'utilisation des fonctionnalités implémentées.
Le package permet de spécifier séparément plusieurs procédures et le corps de leur code dans une même unité qui est stockée sous un seul nom. Le package comprend deux parties :

- La spécification : définit l'interface visible ;
- Le corps : implémente la spécification

Spécification :

```
Create [or replace] Package nom_pack IS | AS
   Déclaration de type et d'items publics
   Déclaration de procédures et de fonctions
End nom_pack ;
```

Corps :

```
Create [or replace] Package Body nom_pack IS | AS
   Déclaration de type et d'items privés
  Corps  de procédures et de fonctions
End nom_pack ;
```

```
Ex : create Package Stock AS
Procedure query_prod ( p_no  number , p_des OUT  varchar2) ;
function max_qte(p_nocl number) Return number ;
end Stock ;
```

```
Create package Body Stock as
query_prod ( p_no IN number, p_des OUT varchar2) IS
Begin
Select desi into p_des from produit where NumProd= p_no ;
```

```
Exception
When NO_DATA_FOUND then
   Dbms_output.put_line('Aucun produit de ce no ' || p_no) ;
End query_prod ;

function max_qte(p_nocl number) Return number is
    Qte number(4) ;
    Begin
       Select max(quantite) into qte from commande where NumCli = p_nocl ;
Return qte ;
End max_qte;

End Stock ;
```

L'appel du package se fait avec nom_package.nomprocure(paramètres)

Ex : Appel à partir de SQLPlus

```
Variable g_des varchar2(30)
Execute Stock.query_prod(10, :g_des)
Print g_des
```

9 Packages Oracle

Oracle offre plusieurs packages prédéfinis qui peuvent être utilisées parmi lesquel le package **DBMS_output**. Ce dernier offre la possibilité d'afficher les données dans une session SQLPlus. Les principales procédures de ce package sont :

Nom procédure	desciption
DBMS_OUTPUT.ENABLE	active l'affichage
DBMS_OUTPUT.DISABLE	Désactive l'afffichage
DBMS_OUTPUT.PUT(chaine)	Affiche chaine à la suite de la ligne courante
DBMS_OUTPUT.PUT_LINE(chaine)	Affiche chaine suivie de caractère de fin de ligne
DBMS_OUTPUT.NEWLINE	Permet de passer à la ligne suivante
DBMS_OUTPUT.GET_LINE (ligne OUT VARCHAR2, statut OUT INTEGER)	Permet de lire une ligne
DBMS_OUTPUT.GET_LINES (ligne OUT VARCHAR2, nbl IN OUT INTEGER)	Permet de lire nbl lignes

Pour réaliser l'affichage, la procédure dbms_output.enable doit être appelée en premier. Elle est équivalente à la commande SQLPlus : set serverouput on .
D'autres packages sont disponibles comme le package UTL_FILe permet la lecture et l'écriture dans des fichiers alors que DBMS_JOB permet de planifier des travaux et DBMSS_SQL donne la possibilité de générer des clauses SQL dynamiquement.