Cours n°4 : Les sous-programmes et les packages PL/SQL

Auteur : Riadh ZAAFRANI

2ème année Licence Computer Science - GLSI

Octobre 2021

1

1

Plan du cours

- **Les sous-programmes**
- **Les types d'arguments**
- **Les packages**

2

Les sous-programmes

- Le langage **PL/SQL**, est un langage algorithmique complet ; il bénéficie de la possibilité de structuration de code, avec un procédé de décomposition de gros blocs de code en plus petits modules qui peuvent être appelés par d'autres modules.
- PL/SQL fournit les structures suivantes :
- **Procédure :** C'est un bloc PL/SQL nommé qui exécute une ou plusieurs actions et est appelé comme une commande PL/SQL. On peut passer et récupérer de l'information d'une procédure à travers sa liste d'arguments.

3

3

Les sous-programmes

- **Fonction :** C'est un bloc PL/SQL nommé qui renvoie une seule valeur et est utilisé comme une expression PL/SQL. On peut passer de l'information à une fonction à travers sa liste d'arguments.
- **Bloc anonyme :** C'est un bloc PL/SQL non nommé qui exécute une ou plusieurs actions. Un bloc anonyme permet au développeur de contrôler la portée des identifiants et la gestion des exceptions.
- **Package :** Un ensemble nommé de procédures, fonctions, types et variables. Un package n'est pas vraiment un module, c'est une application.

4

4

Les blocs nommés

- Tous les types de blocs PL/SQL nommés ont la structure suivante :
- **EN-TETE**
  IS/AS
  …
  BEGIN
  …
  [EXCEPTION]
  …
  END;
- L'entête indique comment un bloc nommé ou un programme doit être appelé.

5

5

Les procédures

- Une **procédure** est un sous-programme qui effectue un traitement particulier.
- Lorsqu'une procédure est créée, elle est d'abord compilée puis stockée dans la base de données sous sa forme compilée.
- Ce **code compilé** peut ensuite être exécuté à partir d'un autre bloc PL/SQL; le code source de la procédure est également stocké et n'a nul besoin d'être analysé une seconde fois à l'exécution.

6

6

Les procédures

- Le code, qui crée une **procédure** est :
- **CREATE [OR REPLACE] PROCEDURE <nom_procédure>**
  [(<nom_argument> [{IN | OUT | IN OUT]}] TYPE [,…])]
  {IS | AS}
  …
  BEGIN
  …
  EXCEPTION
  …
  END [<nom_procédure>];
- **IS | AS** Les deux mots clés sont équivalents; ils déterminent le début de la section des déclarations.

Les procédures

```
SQL> CREATE PROCEDURE AugmenterSalaire
 2  IS
 3 CURSOR c_employé IS SELECT fonction,salaire
 4        FROM employés
 5        FOR UPDATE OF salaire;
 6  begin
 7    for v_employé in c_employé Loop
 8      CASE v_employé.FONCTION
 9        When 'Chef des ventes' Then
10            UPDATE employés SET salaire = salaire * 1.25
11                WHERE CURRENT OF c_employé;
12         Else
13            UPDATE employés SET salaire = salaire * 1.1
14                WHERE CURRENT OF c_employé;
15      END CASE;
16    end loop; COMMIT;
17  END AugmenterSalaire;
18  /
```

- **Procédure créée.**

Les procédures

- La création de la procédure n'implique pas l'exécution du code; elle est d'abord compilée puis stockée dans la base de données sous sa forme compilée, le code compilé pouvant ensuite être exécuté à partir d'un autre bloc PL/SQL.
- La syntaxe pour exécuter une procédure est :
- **<nom_procédure> [(valeur_argument [,…])];**
- Si la procédure n'a aucun argument, il faut l'appeler sans parenthèse ou sans aucun argument entre les parenthèses.

Les procédures

- SQL> begin
  2 AugmenterSalaire;
  3 end;
  4 /
- **Procédure PL/SQL terminée avec succès.**
- SQL> begin
  2 AugmenterSalaire();
  3 end;
  4 /
- **Procédure PL/SQL terminée avec succès.**

Les procédures

- On peut aussi utiliser l'instruction **CALL** pour appeler une procédure stockée, à partir de SQL*PLUS.
- **SQL>** CALL AugmenterSalaire();
- **Appel terminé.**

11

11

Les fonctions

- Une fonction est très semblable à une procédure.
- La différence réside en ce qu'un appel de procédure est en soi une instruction PL/SQL, tandis qu'un appel de fonction fait partie d'une expression.
- La syntaxe de création d'une fonction est :

12

12

Les fonctions

- CREATE [OR REPLACE] FUNCTION <nom_fonction>
  [(<nom_argument> [{IN | OUT | IN OUT]}] TYPE [,...])] RETURN
  <Type_valeur>
  {IS | AS}
  ...
  BEGIN
  ...
    RETURN <expression>;
  EXCEPTION
  ...
  END [<nom_fonction>];
- Type_valeur C'est le type de la valeur de retour de la fonction.
- Une fonction doit contenir au moins une clause RETURN dans sa section d'exécution.

13

13

Les fonctions

```
SQL> CREATE FUNCTION NombreProduits
  2  RETURN number
  3  IS
  4     v_nombre_produits number;
  5  begin
  6     SELECT count(*) INTO v_nombre_produits
FROM produits;
  7     RETURN v_nombre_produits;
  8  end NombreProduits;
  9  /
```

- Fonction créée.

14

14

Les fonctions

```
SQL> begin
  2  dbms_output.put_line('Le nombre de
produits est :'||NombreProduits);
  3  end;
  4  /
```

- **Le nombre de produits est :4**
- **Procédure PL/SQL terminée avec succès.**

La suppression des blocs

- Comme les tables, les procédures et les fonctions peuvent faire l'objet d'une suppression, provoquant leur élimination du dictionnaire de données :
- **DROP PROCDEDURE <nom_procédure>;**
- **DROP FUNCTION <nom_fonction>;**

Plan du cours

- Les sous-programmes
- Les types d'arguments
- Les packages

17

Les arguments

- Comme dans tout autre langage, vous pouvez créer des procédures et des fonctions qui reçoivent des arguments ayant différents modes et dont le passage se fera par valeur ou par référence.
- Les arguments peuvent avoir trois modes : **IN**, **OUT** ou **IN OUT**, IN étant la valeur par défaut.
- La syntaxe de déclaration des arguments est :
- **NOM_ARGUMENT [{IN | OUT | IN OUT}] TYPE [,...]**

18

18

Les arguments

- SQL> CREATE PROCEDURE AugmenterSalaire(Montant IN number)
  2 IS
  3 begin
  4 UPDATE employés SET salaire = salaire + montant;
  5 END AugmenterSalaire;
  6 /
- Procédure créée.

19

19

Les arguments : IN

- La valeur de l'argument réel est passée à la procédure lors de son invocation.
- A l'intérieur de celle-ci, l'argument formel se comporte comme une constante PL/SQL, à savoir qu'il est en lecture seule et ne peut donc être modifié.
- Lorsque la procédure se termine et que le contrôle repasse à l'environnement appelant, l'argument réel n'est pas modifié.

20

20

Les arguments : IN

- SQL> CREATE PROCEDURE AugmenterSalaire(Montant IN number)
```
 2  IS
 3  begin
 4    UPDATE employés SET salaire = salaire+montant;
 5       montant := 2000;
 6  END AugmenterSalaire;
 7  /
```
- **Avertissement : Procédure créée avec erreurs de compilation.**
- Comme vous pouvez l'observer, il est impossible d'affecter une valeur a un argument de type **IN**.

21

21

Les arguments : OUT

- La valeur de l'argument réel est ignorée lors de son invocation de la procédure.
- A l'intérieur de celle-ci, l'argument formel se comporte comme une variable PL/SQL, n'ayant pas été initialisée, contenant donc la valeur NULL et supportant les opérations de lecture et d'écriture.
- Lorsque la procédure se termine et que le contrôle repasse à l'environnement appelant, le contenu de l'argument formel est affecté à l'argument réel.

22

22

Les arguments : OUT

```
SQL> CREATE PROCEDURE NombreProduits(nombre_produits OUT
number)
  2  IS
  3  begin
  4    dbms_output.put_line('La valeur est :'|| nombre_produits);
  5    SELECT count(*) INTO nombre_produits FROM produits;
  6  end NombreProduits;
  7  /
```

```
SQL> Declare
  2   v_nombre_produits number := 0;
  3  begin
  4   NombreProduits(v_nombre_produits);
  5   dbms_output.put_line ('La valeur est :'|| v_nombre_produits);
  6  end;
  7  /
```

```
La valeur est :
La valeur est : 4
```


Les arguments : IN OUT

- Ce mode est une combinaison de IN et OUT.
- La valeur de l'argument réel est passée à la procédure lors de son invocation.
- Lorsque la procédure se termine et que le contrôle repasse à l'environnement appelant, le contenu de l'argument formel est affecté à l'argument réel.

Les arguments : IN OUT

```
SQL> CREATE PROCEDURE NombreProduits(nombre_produits IN OUT
number)
  2  IS
  3  begin
  4    dbms_output.put_line('La valeur est :'|| nombre_produits);
  5    SELECT count(*) INTO nombre_produits FROM produits;
  6  end NombreProduits;
  7  /
```

```
SQL> Declare
  2   v_nombre_produits number := 0;
  3  begin
  4   NombreProduits(v_nombre_produits);
  5   dbms_output.put_line('La valeur est :'|| v_nombre_produits);
  6  end;
  7  /
```

La valeur est : 0
La valeur est : 4

25

25

Association des arguments par position / par nom

- Les arguments peuvent être indiqués :
- **Par position :** Dans ce cas, chaque argument est remplacé par la valeur occupant la même position dans la liste.
- **Par nom :** Dans ce cas, chaque argument est passé dans un ordre quelconque en faisant apparaître la correspondance :
- **(nom_argument => valeur_argument [,...]);**

26

26

Passage de paramètres

- Le passage d'un paramètre de sous-programme peut s'effectuer de deux manières : par **référence** ou par **valeur**.
- Dans le premier cas, un pointeur sur le paramètre réel est passé au paramètre formel correspondant, et dans le second, la valeur du paramètre réel est copiée dans le paramètre formel.
- Par défaut, PL/SQL passera les paramètres **IN** par valeur et les paramètres **IN OUT** et **OUT** par référence.

Les blocs locaux

- Les procédures et les fonctions peuvent être utilisées sans être stockées dans la base, elles sont déclarées dans la section déclarative d'un bloc PL/SQL.

La surcharge de blocs

- Deux ou plusieurs blocs peuvent avoir le même nom et une liste différente de paramètres. De tels modules sont dits **surchargés**.
- Le compilateur compare le paramètre réel aux paramètres des listes des deux modules et il exécute alors le code du programme dont l'entête correspond.

La surcharge de blocs

```
SQL> Declare
  2  FUNCTION ControlValeur(a_date DATE)
  3  RETURN BOOLEAN IS
  4  Begin
  5    RETURN a_date <= SYSDATE;
  6  end;
  7  FUNCTION ControlValeur(a_nombre NUMBER)
  8  RETURN BOOLEAN IS
  9  Begin
 10     RETURN a_nombre >= 0;
 11   end;
 12 Begin
 13   If ControlValeur(UID) then
 14      dbms_output.put_line(UID);
 15   end if;
 16 end;
 17 /
```

67

Plan du cours

- Les sous-programmes
- Les types d'arguments
- Les packages

31

Les packages

- Un package est une structure PL/SQL qui permet de stocker un ensemble d'objets logiquement associés et comprend deux parties distinctes :
  - La **spécification** qui contient la déclaration d'objets publics.
  - Le **corps** qui concerne l'implémentation des sous programmes publics et privés. Ces derniers sont invisibles de l'extérieur

32

32

Les packages

- la **spécification** et le **corps** sont stockés séparément dans le dictionnaire de données.
- La séparation de la déclaration de la **spécification** de celle du **corps** du package, permettra par exemple de compiler des modules qui appellent des sous_programmes spécifiés mais non encore implémentés.

33

33

La spécification de package

- La spécification d'un package contient des informations relatives au contenu du package.
- Les règles de déclarations sont :
  - L'ordre d'apparition des éléments contenus dans un package peut être quelconque.
  - Tous les types d'éléments ne doivent pas nécessairement être présents.
  - Toutes les déclarations de procédures et de fonctions doivent être des déclarations préalables. Une déclaration préalable décrit simplement le sous-programme et ses arguments sans en inclure le code.

34

34

La spécification de package

- La syntaxe de création d'un package :
- **CREATE [OR REPLACE] PACKAGE NOM_PACKAGE {IS | AS}**
  [Déclarations des variables et des types]
  [Déclarations d'exceptions]
  [Spécifications des curseurs]
  [Spécifications des modules]
  END [NOM_PACKAGE];

35

35

Exemple de spécification de package

- Le package **GererProduit** suivant regroupe essentiellement deux fonctions booléennes et une procédure.
- La procédure AddProduit reçoit les données d'un produit à créer, invoque les fonctions VerifieCatégorie et VerifieFournisseur pour valider les valeurs reçues puis insère un n-uplet à la table produit.

36

36

Exemple de spécification de package

```
SQL> CREATE OR REPLACE PACKAGE GererProduit
  2  AS
  3      CURSOR c_produit RETURN PRODUIT%ROWTYPE;
  4      v_produit c_produit%ROWTYPE;
  5      e_PasDeCatégorie   EXCEPTION;
  6      e_PasDeFournisseur EXCEPTION;
  7      TYPE t_Produits IS TABLE OF PRODUIT%ROWTYPE
  8          INDEX BY BINARY_INTEGER;
  9      FUNCTION VerifieCatégorie(a_code_catégorie
CATEGORIE.CODE_CATÉGORIE%TYPE)
 10              RETURN BOOLEAN;
 11      FUNCTION VerifieFournisseur(a_no_fournisseur
FOURNISSEUR.NO_FOURNISSEUR%TYPE)
 12              RETURN BOOLEAN;
 13      PROCEDURE AddProduit(a_produit PRODUIT%ROWTYPE);
 14  END GererProduit;
 15  /
```

Package créé.

37

37

Eléments publics et privés d'un package

- L'un des concepts centraux des packages est le niveau de confidentialité de ses éléments.
- Un élément de package, peut être en effet public ou privé.
- **Public :** Défini dans les spécifications. Un élément public peut être référencé dans d'autres programmes et blocs PL/SQL.
- **Privé :** Défini uniquement dans le corps du package. Un élément privé ne peut être référencé en dehors du package.
- La séparation des éléments privés et publics offre aux développeurs un contrôle sur leurs données et programmes.

38

38

Comment référencer les éléments d'un package

- Un package est propriétaire de ses objets.
- Pour référencer n'importe lequel de ces objets; on préfixe le nom de l'objet avec le nom du package :
- **NOM_PACKAGE.NOM_OBJET;**

39

39

Le corps de package

- Le corps d'un package ne peut pas précéder la spécification du package.
- Il peut inclure des déclarations additionnelles qui sont globales dans le corps du package, mais qui ne sont pas visibles dans la spécification.
- Toute déclaration préalable dans la spécification de package doit s'accompagner d'une définition dans le corps du package, la spécification du sous-programme devant être identique dans les deux emplacements.

40

40

Le corps de package

- La syntaxe de création est la suivante :
- **CREATE [OR REPLACE] PACKAGE BODY NOM_PACKAGE {IS | AS}**
  [Déclarations des variables et des types]
  [Spécifications et SELECT des curseurs]
  [Implémentations des modules]
  [BEGIN
  Ordres exécutables]
  [EXCEPTION
  Exceptions]
  END [NOM_PACKAGE];

41

41

Exemple du corps de package

```
CREATE OR REPLACE PACKAGE BODY GererProduit
IS
    v_utilisateur VARCHAR2(25);
    CURSOR c_produit RETURN PRODUIT%ROWTYPE
        IS SELECT NumProduit, NomProduit,No_fournisseur,
        Code_catégorie, quantité,prixunitaire FROM produit;
FUNCTION VerifieCatégorie(a_code_catégorie
catégories.code_catégorie%TYPE)
RETURN BOOLEAN AS
begin
   for v_code_catégorie in (Select code_catégorie from catégories
                    Where code_catégorie = a_code_catégorie) loop
        RETURN TRUE; -- Catégorie trouvée
   end loop;
   RETURN FALSE;
end VerifieCatégorie;
```

42

42

Exemple du corps de package

```
FUNCTION VerifieFournisseur(a_no_fournisseur
FOURNISSEUR.NO_FOURNISSEUR%TYPE)
            RETURN BOOLEAN As
begin
  for v_no_fournisseur in (Select no_fournisseur
        from fournisseur
        Where no_fournisseur = a_no_fournisseur) loop
      RETURN TRUE; -- Fournisseur trouvé
  end loop;
  RETURN FALSE;
end VerifieFournisseur;
```


Exemple du corps de package

```
PROCEDURE AddProduit(a_produit PRODUIT%ROWTYPE) AS
BEGIN
IF NOT VerifieCatégorie(a_produit.Code_catégorie) THEN
      RAISE e_PasDeCatégorie
END IF;
IF NOT VerifieFournisseur(a_produit.No_fournisseur) THEN
      RAISE e_PasDeFournisseur
END IF;
INSERT INTO PRODUIT VALUES
        (a_produit.NumProduit, a_produit.NomProduit,
         a_produit.No_fournisseur, a_produit.Code_catégorie,
         a_produit.quantité,a_produit.prixunitaire);
end AddProduit;
```

Exemple du corps de package

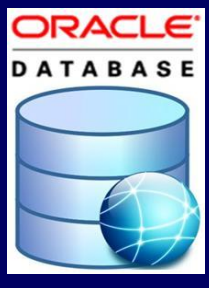
```
Begin
      SELECT USER INTO v_utilisateur FROM DUAL;
Exception
      When e_PasDeCatégorie then
         dbms_output.put_line('Erreur code_catégorie');
      When e_PasDeFournisseur then
         dbms_output.put_line('Erreur no_fournisseur');
      When others then
         dbms_output.put_line('Erreur Package GererProduit');
END GererProduit;
/
```

- Corps de package créé.

45

45

Cours n°4 : Les sous-programmes et les packages PL/SQL



Auteur : Riadh ZAAFRANI

Merci pour votre attention

46