Compléments de PL/SQL

Cours 3*

26 mars 2019

Note globale : DS Promo (2/3) + DS de groupe (1/3)
Isabelle Gonçalves - isabelle.goncalves@univ-lyon1.fr

* contient des extraits des cours de Guillaume Cabanac, Fouad Dahak, Yacine Bouhabel, Amrouche Karima, Richard Grin, Adnene Belfodil, Anes Bendimerad, Alain Pillot, Mooneswar Ramburrun + de la documentation Oracle

Package

- Regroupement logique de :
  - types PL/SQL,
  - constantes,
  - variables,
  - exceptions,
  - procédures,
  - fonctions
- Ne peut pas être appelé mais permet au serveur Oracle de charger plusieurs objets simultanément en mémoire.

Quand on fait appel à un élément du package, celui-ci est entièrement chargé en mémoire, ce qui limite les accès disques par la suite.

Packages Oracle

- Ce sont des packages intégrés
- Exemple : DBMS_OUTPUT
  - permet d'envoyer des messages
  - peut être exécuté par n'importe quel utilisateur Oracle
  - exemples de procédures dans ce package : PUT, PUT_LINE

Package : Structure

- Un fichier de spécifications :
  - Contient les éléments publics : Prototypes des procédures et fonctions, déclaration de variables ou constantes globales ...
  - Peut exister sans la partie corps
- Un fichier pour le corps du package
  - L'implémentation des procédures et fonctions publiques
  - Contient en plus les éléments privés
  - Ne peut pas exister sans la partie spécifications

Package : Spécifications

```
CREATE OR REPLACE PACKAGE rh_package IS
- - variable globale
        g_comm number :=10;
- - fonction publique
        FUNCTION get_emp
              (pn IN number) RETURN emp%ROWTYPE;
- - procédure publique
        PROCEDURE reset_comm
              (v_comm IN number);
END rh_package;
```

Premier fichier

rh_package_sp.sql

Package : Corps (1/3)

```
CREATE OR REPLACE PACKAGE BODY rh_package IS
- - fonction publique
        FUNCTION get_emp (pn IN number)
        RETURN emp%ROWTYPE IS
                v_emp emp%ROWTYPE;
        BEGIN
                SELECT *
                INTO v_emp FROM emp WHERE eno = pn;
                RETURN v_emp;
        END get_emp;
...
```

Deuxième fichier

rh_package.sql

Package : Corps (2/3)

```
...
- - fonction privée
        FUNCTION validate_comm (v_comm IN NUMBER)
        RETURN BOOLEAN IS
                v_max_comm NUMBER;
       BEGIN
              SELECT MAX(comm)
              INTO v_max_comm
              FROM emp;
              IF v_comm > v_max_comm THEN RETURN FALSE;
              ELSE RETURN TRUE;
              END IF
       END validate_comm;
...
```

Package : Corps (3/3)

```
...
- - procédure publique
        PROCEDURE reset_comm( v_comm IN NUMBER ) IS
                v_valid BOOLEAN;
        BEGIN
                v_valid := validate_comm(v_comm);
                IF v_valid = TRUE THEN
                        g_comm := v_comm;
                ELSE
                        g_comm := 0;
                END IF;
        END reset_comm;
END rh_package;
```

Package : Utilisation

- Exemple 1 : utilisation d'une variable globale à l'extérieur du package
  -> nom_package.nom_variable

```
rh_package.g_comm := 15 ;
```

- Exemple 2 : appel d'une procédure à l'extérieur du package
  -> nom_package.nom_procédure

```
rh_package.reset_comm(1500) ;
```

- Exemple 3 : appel d'une procédure d'un autre schéma
  ->nom_schema.nom_package.nom_procédure

```
scott.rh_package.reset_comm(1500) ;
```

Package : Suppression

- Suppression de la spécification et du corps

```
DROP PACKAGE package_name;
```

- Suppression du corps seul

```
DROP PACKAGE BODY package_name;
```

Package : Avantages

- Modularité
- Simplification du développement
  Seule la spécification est nécessaire pour la compilation. Le corps sera nécessaire pour l'exécution.
- Informations cachées
- Amélioration des performances

Package : Exercice

Faire l'exercice 1 du TD3

Collection : Introduction

- Un ensemble ordonné d'éléments de même type
- Une seule dimension mais on peut créer des collections de collections, des collections d'enregistrements ...
- On distingue les types :
  - VARRAY : taille maximale et éléments groupés. Entiers comme indices de case commençant à 1.
  - nested TABLE : taille variable et éléments groupés au départ. Entiers comme indices de case commençant à 1.
  - index-by TABLE : tableaux associatifs avec comme clés des entiers ou des chaînes de caractères.

Collection : Déclaration de type et variable

VARRAY

```
TYPE nom_type IS VARRAY(taille_max) OF type_element ;
```

nested TABLE

```
TYPE nom_type IS TABLE OF type_element ;
```

index-by TABLE

```
TYPE nom_type IS TABLE OF type_element
INDEX BY type_cle ;
```

type_cle sera PLS_INTEGER ou VARCHAR2

variable

```
nom_variable nom_type ;
```

Collection : Initialisation, accès à un élément

- Initialisation nécessaire avant utilisation pour les types VARRAY et nested TABLE

dans la déclaration

nom_variable nom_type := nom_type (elem1, elem2...) ;
nom_variable nom_type := nom_type ()

dans la section BEGIN

nom_variable := nom_type (elem1, elem2...) ;
nom_variable := nom_type ()

- Accéder à un élément

nom_variable(indice)

Collection : Exemple

```
DECLARE
        TYPE ename_table_type IS TABLE OF emp.ename%TYPE
                INDEX BY BINARY_INTEGER;
        TYPE hiredate_table_type IS TABLE OF DATE
                INDEX BY BINARY_INTEGER;
        ename_table ename_table_type;
        hiredate_table hiredate_table_type;
BEGIN
        ename_table(1) := 'CAMERON';
        hiredate_table(8) := SYSDATE + 7;
        IF ename_table.EXISTS(1) THEN
                INSERT INTO ...
        ...
END;
/
```

Collection : Méthodes

- EXTEND ou EXTEND(n) pour ajouter un élément ou n éléments null
- FIRST/LAST retourne le premier/dernier indice (null si vide)
- PRIOR(i)/NEXT(i) retourne l'indice de l'élément qui précède/suit l'élément d'indice i
- COUNT permet de connaître le nombre d'éléments présents
- LIMIT permet de connaître le nombre maximal d'éléments
- TRIM ou TRIM(n) pour supprimer le dernier ou les n derniers éléments
- DELETE ou DELETE(i) pour supprimer tous les éléments ou l'élément d'indice i

Appel

nom_variable.nom_méthode[(paramètres)]

EXTEND et TRIM ne peuvent pas être utilisées avec un type index-by

Collection : Boucle

```
i := variable_table.first;
WHILE i IS NOT NULL LOOP
        DBMS_OUTPUT.PUT_LINE(variable_table(i));
        i := variable_table.NEXT(i); - - get subscript of next element
END LOOP;
```

Collection : Exceptions prédéfinies

- COLLECTION_IS_NULL : la collection n'est pas initialisée
- NO_DATA_FOUND : l'élément accédé n'existe plus
- SUBSCRIPT_BEYOND_COUNT : l'indice de l'élément accédé n'existe plus
- SUBSCRIPT_OUTSIDE_LIMIT : l'indice est en dehors des valeurs autorisées
- VALUE_ERROR : l'indice est null ou n'est pas convertible en entier

Package : Exercice

Terminer le TD3