Procédures et fonctions stockées (sous-programmes)

1. Généralités
2. Développement d'un sous-programme
3. Utilisation d'un sous-programme
4. Gestion des erreurs

1. Généralités

- Sous-programmes
  - Blocks PL/SQL nommés
  - Deux type de sous-programmes : Procédures, Fonctions
  - Passage de paramètres entre blocks afin d ’améliorer la lisibilité, la modularité …
  - Peuvent être programmées soit en PL/SQL soit en Java
  - Récursivité permise

- Procédure et Fonction :
  - Procédure : effectue une action (en général)
  - Fonction : renvoie une valeur (calcul une valeur)

Sous-programmes stockés

- Sous-programmes stockés (programmes mémorisées) :
  - permettent de stocker et d'exécuter des traitements fréquemment utilisée au niveau du noyau du SGBDR plutôt que dans chaque application.
  - Le code est stocké une seule fois dans la base et peut être exploité par plusieurs applications.
  - Peuvent être chargées d'une manière permanente dans la BD, prêts à être exécutés.

- Avantages de l'utilisation de code stocké dans la base :
  - Le traitement complexes peut être réalisé au niveau de la base et donc par le serveur. Le fait de déplacer le traitement des applications vers la base peut considérablement améliorer les performances.
  - Réutilisation de requêtes dans la base : le code est stocké dans les zone SQL partagées stockant des représentations analysées des instructions SQL. La deuxième fois qu'une procédure est exécutée, elle peut tirer parti de l'analyse réalisée auparavant, améliorant ainsi le performances d'exécution.
  - Les traitements n'ont pas besoin d'être écrites dans chaque application.

2. Développement d'un sous-programme

2.1. Création d'une procédure

- Syntaxe de la commande create procedure :

Create [or replace] procedure *[schéma.]nom_procédure*
[(paramètre [, paramètre] …)]
[IS|AS]
[*déclarations locales*]
Begin
…
[Exception *manipulation des exceptions*]
End [*nom_procédure*] ;

bloc PL/SQL : corps de procédure

- Syntaxe d'un paramètre :
  - *Var-name* [IN | OUT | IN OUT] *type_données* [{:= | DEFAULT} *value*]
    - IN : paramètre formel en entrée, OUT : en sortie, IN OUT : entrée-sortie
    - Le mode par défaut : IN
  - Type_données : tous les types de PL/SQL sont utilisables, mais sans spécification de traille pour les types explicites ;
    - ex) create procedure (a char (20)) IS … : non permis

Exemple d'une procédure

- *Création d'un nouveau pilote :*

```
Create or replace procedure nv_pilote
 (x_nopilot  IN  pilote.nopilote%type, x_nom  IN  pilote.nom%type,
   x_sal  IN  pilote.sal%type, x_comm  IN  pilote.comm%type)
IS
Begin
     Insert  Into Pilote Values (x_nopilot, x_nom, x_sal, x_comm) ;
     Commit ;
End nv_pilote ;
```

- *Suppression d'un pilote à partir de son numéro :*

```
Create procedure del_pilote (x_nopilot  IN  pilote.nopilote%type)
IS
Begin
     Delete from Pilote where nopilot = x_nopilot ;
End del_pilote;
```

- Lorsqu'une procédure existe déjà, on peut la remplacer au moyen de la commande create or replace procedure. L'intérêt de cette commande est que les octrois du privilèges EXECUTE sont maintenus.

- Pour créer un objet procédural (procédure, fonction, package) dans son propre schéma, il faut disposer le privilège CREATE PROCEDURE. Si cet objet appartient à un autre utilisateur, il faut avoir reçu le privilège CREATE ANY PROCEDURE.

- note) Les blocs PL/SQL de sous-programmes peuvent inclure des instructions LMD, mais pas LDD (comme create , alter, …)

Compilation d'un sous-programme

- Compiler un sous-programme :
  - Compilation du code sources avec génération d'un pseudocode si aucune erreur de syntaxe n'est pas détectée.
  - Stockage du code source dans la base, même si des erreurs sont détectées.
  - Stockage du pseudocode (P code) dans la base → éviter la recompilation du sous-programme à chaque appel.
  - SQL> @nom_fichier ou SQL> start nom_fichier

- Exécuter un sous-programme :
  - SQL>

- Vois les erreurs :
  - SQL>

2.2. Création d'une fonction

- Syntaxe de la commande create function :
  - Idem qu'une procédure SAUF la clause **RETURN** qui spécifie le type de la valeur retournée par la fonction.

Create [or replace] function *[schéma.]nom_fonction*
[(paramètre [, paramètre] …)]
RETURN type_données
[IS|AS]
[*déclarations locales*]
… ; *- - bloc PL/SQL , le corps de la fonction*

- L'instruction *RETURN (valeur_résultat)* dans le corps finit l'exécution de la fonction et retourne sous le contrôle du programme appelant.
- Généralement placé vers la fin de la fonction pour "renvoyer" le résultat.
- Si pas de *RETURN*, l'exception prédéfinie *PROGRAMME-ERROR* est levée.

Exemple d'une fonction

– *Calcul du nombre moyen d'heure de vol des avions dont le type est transmis en paramètre ;*

```
Create or replace function moy_h_vol
            (x_codetype  IN  avion.type%type)
RETURN number
IS
nbhvol_avg NUMBER(8,2) := 0  /* sert à transmettre la valeur
                                résultat */
Begin
   Select Avg(nbhvol) Into nbhvol_avg From Avion
            Where type = x_codetype;
   Return (nbhvol_avg);
End moy_h_vol ;
```

Les paramètres

- Trois modes applicables à tout sous-programme :
  - IN (par défaut), OUT, IN OUT
- *Fortement conseillé de ne pas les utiliser dans les fonctions.* => le mode des paramètres est en général en entrée.

- IN : passage par valeur
  - Dans le sous-programme, un paramètre marqué IN se comporte comme une constante. Impossible d'avoir un paramètre IN en partie gauche d'une affection.
- OUT : passage par adresse
  - Permet à une procédure de retourner une valeur au programme appelant.
- IN OUT : combine les deux
  - Dans le programme appelant, passe les valeurs aux sous-programmes et récupère les valeurs mises à jours.
  - Dans le programme appelé, le paramètre IN OUT se comporte comme une variable normale initialisée (assignement de valeur possible, assignation à d'autre variables aussi).

2.3. Compilation et dictionnaire de données

- Diagnostics d'erreur correspondant en utilisant les vues suivantes du dictionnaire de données :
  - USER_ERROR : pour les objets procéduraux appartenant à l'utilisateur
  - ALL_ERRORS : pour les objets procéduraux appartenant à l'utilisateur ou auxquels il peut accéder
  - DBA_ERROR : pour les objets procéduraux de la base.
- Visualiser les erreurs associées à des objets procéduraux qui ont été créés ;

```
Ex) select    line ,          /* n° de ligne de l'erreur */
              position,       /* n° de colonne de l'erreur */
              text            /* texte de message d'erreur */
    From USER_ERRORS
    Where name = 'del_pilote' and type = 'PROCEDURE'  Order by 1
```

  - Note) type : PROCEDURE, FUNCTION, PAKAGE, PACKAGE BODY
- Note) Les erreurs à la compilation d'un objet procédural sont remplacées à chaque nouvelle compilation et disparaissent lors de la suppression de l'objet.

- La commande SQL*PLUS show errors
  - affiche les erreurs de la compilation la plus récente, en extrayant les erreurs de la vue de dictionnaire de données USER_ERRORS.

- Visualiser le code source d'objets procéduraux : Le code source stocké dans la BD peut être récupéré à partir des vues du dictionnaire de données suivantes :

  Ex)

- Des informations générales sur les objets procéduraux sont stockées dans les vues suivantes du dictionnaire de données :
  - USER_OBJECTS , ALL_OBJECTS ,DBA_OBJECTS

2.4. Suppression d'un sous-programme

- Pour supprimer une procédure ;

  Drop procedure nom_procédure ;

- Pour supprimer une fonction ;

  Drop function nom_fontion ;

- Note)
  - Pour supprimer un objet procédural (procédure, fonction, package, le corps d'un package), il faut soit le posséder soit disposer du privilège DROP ANY PROCEDURE ;
  - Pour supprimer un package ;

    Drop package nom_package ;
  - Pour supprimer le cops d'un package ;

    Drop package body nom_package ;

3. Utilisation d'un sous-programme stocké

Environnement d'appel		Commande
Interactif	SQL*PLUS	EXECUTE
Inclus dans une application cliente	SQL*FORMS	Appel direct
	PRO*XX	Order EXEC SQL
	Procédure Java	#sql {call }
	Bloc PL/SQL	Appel direct
	Autre procédure ou fonction	Appel direct

Modes d'appel à une procédure stockée.

3.1. Transmission des paramètres

- La transmission de valeurs des paramètres en entrée (IN) peut se faire de trois façons différentes :
  - Transmission par positionnement ;

```
SQL> EXECUTE nv_pilote ('3751', 'MARTIN', 25000.00,
        6000.00) ;
```

  - Transmission par nom ;

```
SQL> EXECUTE nv_pilote (x_nopilote => '3751', x_np=>
        'MARTIN', x_comm => 6000.00 , x_sal => 25000.00,) ;
```

  - Transmission mixte ;

```
SQL> EXECUTE nv_pilote ('3751', 'MARTIN', x_comm =>
        6000.00 , x_sal => 25000.00,) ;
```

Les paramètres par défaut

- Rendre optionnel l'appel de certains paramètres d'une procédure

- Ex)

```
create or replace procedure create_dep
(new_dep CHAR DEFAULT 'TEMP',
new_loc CHAR DEFAULT 'TEMP')  IS
..
```

```
SQL> Execute Create_dep ;
SQL> Execute  Create_dep('VOIRIE');
SQL> Execute  Create_dep('VOIRIE', 'NEW-YORK');
```

3.2. Appel aux procédures et aux fonctions

1. En mode interactif

– L'appel à une procédure stockée se fait par la commande :

```
Execute nom_procédure [(liste de paramètres)];
```

– L'appel à une fonction stockée se fait par la commande :

```
Execute :variable_locale := nom_fonction [(liste de paramètres)];
```

Ex) Appel à la procédure nv_pilote à partir d'un script SQL*PLUS ;

```
EXECUTE nv_pilote (10, 'toto',2000) ;
```

Ex) Appel à la fonction moy_h_vol ;

```
VARIABLE moyenne NUMBER
EXECUTE :moyenne := moy_h_vol('AB3');
PRINT moyenne ;
```

2. A partir de Developer (Oracle)

– Appel à la procédure *del_pilote* depuis une application SQL*Forms ;

```
Begin
   …
   del_pilote (numero);
   …
End ;
```

3. A partir d'un programme hôte

Ex) suppression d'un pilote par une procédure hôte écrite en C faisant appel à la procédure stockée *del_pilote* ;

```
Void sup_emp()  {
   EXEC SQL BEGIN DECLARE SECTION;
   Char numero [4];
   EXEC SQL END DECLARE  SECTION;
   Printf("\n Entrez le numéro d'employé : ") ;
   Scanf ("%d", &numéro);
   EXEC SQL del_pilote (:numero) ;
}
```

4. A partir d'un bloc PL/SQL

Ex) appel à la procédure del_pilote à partir d'un bloc PL/SQL

```
DECLARE
Numero pilote.nopilot%type;
BEGIN
   del_pilote(numero);
END;
```

5. A partir d'un autre schéma

– exécuter une procédure appartenant à un autre utilisateur

Ex) appel à la procédure del_pilote, créée par *martin* depuis le schéma *durand*

```
SQL > EXECUTE  martin.del_pilote('3761');
```

Note) Pour autoriser un autre utilisateur à exécuter un objet procédural vous appartenant, octroyez-lui le privilège EXECUTE sur cet objet :

```
Grant Execute on My_PROCEDURE to durand ;
```

=> Si un utilisateur ne dispose pas du privilège *EXECUTE*, il dot avoir reçu le privilège *EXECUTE ANY PROCEDURE* pour pouvoir exécuter la procédure.

6. A partir d'une base distante

– Pour exécuter une procédure distante, le nom du lien de BD doit être spécifié (DATABASE LINK).

Ex) appel de la procédure *del_pilote*, créée par *martin*, sur la base distante *airbase*, depuis *durand* ;

```
SQL>  Execute martin.del_pilote@airbase('3761');
```

Note) Pour simplifier la référence à un objet procédural, on peut utiliser un synomyme :

```
SQL> Execute martin.del_pilote('3671');
=  SQL> Create synonym del_pilote for martin.del_pilote ;
   SQL> Execute del_pilote('3671');

SQL> Execute  del_pilote@airbase('3671');
=  SQL> Create synonym  del_pilote for del_pilote@airbase
   SQL> Execute del_pilote('3671');
```

7. En tant que fonction utilisateur dans les expressions SQL

- Une fonction utilisateur peut être utilisée dans des expressions SQL (ou une fonction standard SQL peut l'être) avec les restrictions suivantes :
  - Elle doit être une fonction stockée et non procédure stockée.
  - Les types de données de ses arguments sont limitées à CHAR, DATE et NUMBER ;
  - Au moment de l'appel, les paramètres doivent être transmis par position et non par nom ;
  - La fonction ne doit pas comporter d'ordres INSERT, UPDATE, DELETE . Elle ne ne doit pas mettre de données à jour dans la base ;
  - Elle ne peut pas être utilisé dans des contraintes CHECK .

  - Ex) Select type, moy_h_vol (type) from avion ;

4. Gestion des erreurs

- Deux types d'erreurs
  - Erreur détectée par le SGBD, Erreur générée par l'utilisateur

- Solutions : chaque type peut être géré par
  - Section EXCEPTION :
    - idem que le traitement des erreurs dans un bloc PL/SQL
    - L'exécution de la procédure (fonction) est considérée comme *réussie*, par l'environnement appelant.
  - Environnement appelant
    - La procédure (fonction) génère un diagnostic d'erreur et communique les erreurs à l'environnement. L'environnement appelant gère l'erreur.

Gestion des erreurs par l'environnement appelant

1. Erreur émise par le SGBD
   - Le code erreur, sous la forme ORA-xxxxx, et le message associé sont transmis à l'appelant.
   - Ex) ZERO_DIVIDE : ORA-01476, NO_DATA_FOOUND : ORA-01403, …

2. Erreur générée par l'utilisateur
   - On peut définir le numéro de l'erreur et le message qui apparaîtra lorsque l'erreur se produira.
   -
     - Numéro_erreur : numéro fourni pour l'erreur utilisateur. Il doit être compris entre -20 000 et -20 999.

Ex) Vérifier que le montant de la commission d'un pilote est toujours inférieur à son salaire :

```
Create or replace Procedure verifi_nom
(x_nopilot pilote.nopilot%type )
IS
   dif  pilote.sal%type := 0 ;
Begin
   Select sal – NVL(comm, 0)  Into dif From Pilote
   Where nopilot = x_nopilot;

   If dif < 0 Then
            RAISE_APPLICATION_ERROR(-20001, 'commission
   supérieure au salaire') ;
   End If;
End;
```

- Ex 1: gestion des erreurs par une section EXCEPTION

  En cas de suppression d'un pilote, vérifier que le pilote n'est affecté à aucun vol. Lorsque le pilotes est affectés à des vols, afficher le message d'erreur 'le pilote est déjà affecté' sans le supprimer de la table.

- Ex 2 : gestion des erreurs par l'environnement appelant

  En cas de suppression d'un pilote, vérifier que le pilote existe dans la table pilote. Dans la procédure del_pilote, si le pilote n'existe pas dans la table, remplacer le message standard (qui va apparaître dans l'envenimement appelant), ORA_01403 : no data found, par le message, 'le pilote nom_pilote n'existe pas'.

Nommage de procédures et de fonctions

- Les procédures (fonctions) devraient être nommées en accord avec la fonction de gestions qu'elles exécutent.
  - Ex) ADD_WORKER : elle ajoute (*ADD*) un tuple dans la table *WORKER*.

- Le nom de la procédure devrait inclure celui de la table(ou des tables) qu'elle affecte.

SQL dynamique

SQL dynamique

- composer et exécuter les requêtes SQL en temps (run time).
- rend l'application plus flexible.
- dans le PL/SQL, on peut exécuter n'import quelle type de SQL (même la définition des données et des contrôles de données)
- The `EXECUTE IMMEDIATE` statement prepares (parses) and immediately executes a dynamic SQL statement

```
EXECUTE IMMEDIATE dynamic_string
[INTO {define_variable[, define_variable]... |
  record}]
[USING [IN | OUT | IN OUT] bind_argument
[, [IN | OUT | IN OUT] bind_argument]...]
[{RETURNING | RETURN} INTO bind_argument[,
  bind_argument]...];
```

```
DECLARE
   sql_stmt VARCHAR2(200);
   emp_id NUMBER(4) := 7566;
   salary NUMBER(7,2);
   dept_id NUMBER(2) := 50;
   dept_name VARCHAR2(14) := 'PERSONNEL';
   location VARCHAR2(13) := 'DALLAS';
   emp_rec emp%ROWTYPE;
BEGIN
   EXECUTE IMMEDIATE 'CREATE TABLE bonus (id NUMBER, amt
     NUMBER)';
   sql_stmt := 'INSERT INTO dept VALUES (:1, :2, :3)';
   EXECUTE IMMEDIATE sql_stmt USING dept_id, dept_name,
     location;
   sql_stmt := 'SELECT * FROM emp WHERE empno = :id';
   EXECUTE IMMEDIATE sql_stmt INTO emp_rec USING emp_id;
END;
```