



SYSTÈME DE GESTION DE BASE DE DONNEES RELATIONNELLE – AVANCEE

73

CHAP : PL\SQL : sous-programme et paquetage

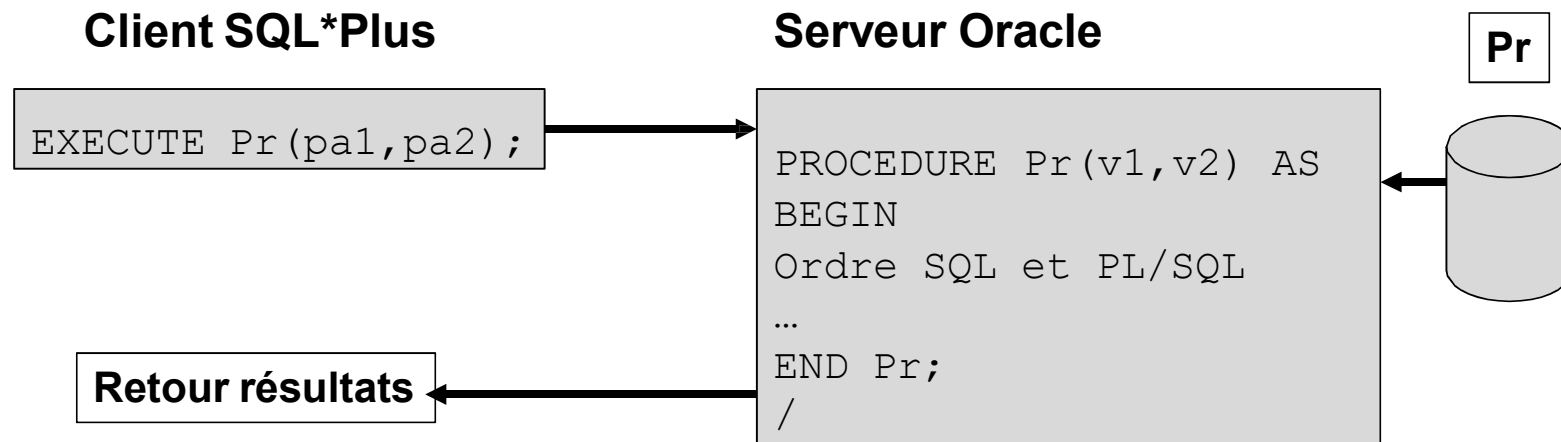
SOUS-PROGRAMME

- Blocs PL/SQL nommés, avec paramètres en **entrée** et/ou **sortie**
- Deux types de sous-programmes
 - **Fonction** retourne un résultat unique
 - **Procédure** stockée retourne un ou plusieurs résultats
- Sous-programmes compilés et stockés dans la BD
 - Code source valide stocké dans USER_SOURCE
 - Code recompilé
 - automatiquement lors des modifications directs des objets
 - manuellement

```
ALTER PROCEDURE <nom_procedure> COMPILE;
```

- **Sous-programmes partagés** par plusieurs utilisateurs

SOUS-PROGRAMME



SOUS-PROGRAMME

➤ *Sécurité*

- Droit d'accès sur les sous-programmes

➤ *Performance*

- Programme compilé et optimisé
- Réduction du nombre d'appels à la base
- Partage de code

➤ *Productivité*

- Simplicité de la maintenance des programmes : modularité, lisibilité, réutilisabilité, etc.

SYNTAXE DE CRÉATION DE PROCÉDURE

```
CREATE [OR REPLACE] PROCEDURE [(liste des parametres)] IS  
<Déclarations>  
BEGIN  
<instructions>  
[EXCEPTION  
<Gestion des exceptions> ]  
END ;
```

- La clause optionnelle OR REPLACE permet de recréer une procédure existante.
- Les clauses optionnelles IN (par défaut), OUT (une valeur peut être assignée au paramètre) et IN OUT permettent de spécifier la manière dont sont utilisés les paramètres.

EXEMPLE DE CRÉATION ET EXÉCUTION DE PROCÉDURE

Creation :

```
SQL> create or replace procedure bonjour (n number)
IS
begin
dbms_output.put_line('bonjour ' || to_char(n));
end ;
/
```

Execution possible depuis SQL*PLUS ou depuis un bloc PL/SQL

```
SQL> SET serveroutput ON;
SQL> execute bonjour(3);
```

Suppression :

```
SQL> DROP PROCEDURE bonjour;
```

EXEMPLE DE CRÉATION ET EXÉCUTION DE PROCÉDURE AVEC PARAMÈTRE IN

Creation :

```
create or replace procedure carre (a in number)
IS
begin
dbms_output.put_line('carre= ' || a*a);
end ;
/
```

Execution possible depuis SQL*PLUS ou depuis un bloc PL/SQL

```
execute carre(3);
Carre= 9
```

Un paramètre in est pris comme une donnée et ne doit pas être modifié dans la procédure.

EXEMPLE DE CRÉATION ET EXÉCUTION DE PROCÉDURE AVEC PARAMÈTRE OUT

Creation : Procédure qui calcule le maximum de deux nombres et le renvoie en résultat

```
declare
  a number;
  procedure max (a in number, b in number, x out number)
  is
    begin
      if a>b then x := a;
      else x:=b;
      end if;
    end;
  begin
    max(2,5,a);
    dbms_output.put_line('max = ' || a);
  end;
/
```

La procédure **max** a été déclarée à l'intérieur d'un bloc PLSQL

EXEMPLE DE CRÉATION ET EXÉCUTION DE PROCÉDURE AVEC PARAMÈTRE IN OUT

Les paramètres **in out** sont modifiables dans la procédure. Ce sont des paramètres données (en entrée) et résultat (en sortie).

Creation : Procédure qui permute deux nombres

```
create or replace procedure
permute (a in out number, b in out number) is
    w number;
begin
    w := a;
    a := b;
    b := w;
end;
/
```

Programme test

```
declare
    x number := 1;
    y number := 2;
begin
    permute (x,y);
    dbms_output.put_line(x || ' ' || y);
end;
/
```

RÉCAP PROCÉDURES : STRUCTURE GÉNÉRALE

```
CREATE [OR REPLACE] PROCEDURE <nom_procedure> [(paramètre1 [ IN |OUT | IN OUT ] TypeSQL
    [(:=|DEFAULT} expression]
[, paramètre2 [IN |OUT | IN OUT ] TypeSQL [(:=|DEFAULT} expression]

...)] {IS|AS}
-- déclarations des variables et
-- curseurs utilisés dans le corps de la procédure BEGIN
...
-- instructions SQL ou PL/SQL EXCEPTION
... END;
/
```

RÉCAP APPEL DE PROCÉDURE

Sous SQL/Plus

```
EXECUTE <nomPROCEDURE> (list_param);
```

```
--exemple SQL PLUS  
set serveroutput on  
EXECUTE Insérer_LignComm('F45','Estrade', 'C',50,100, 50);
```

```
--exemple SQL PLUS  
VARIABLE ret NUMBER  
.....  
EXECUTE Supprimer_Prod(123, :ret);  
IF ret=9 THEN.....
```

RÉCAP APPEL DE PROCÉDURE

Dans un programme PL/SQL

```
<nomPROCEDURE> (list_param);
```

```
ACCEPT vRefProd PROMPT 'entrez la référence du produit';
ACCEPT vDesign PROMPT 'entrez sa désigantion';
ACCEPT vCat PROMPT 'entrez sa catégorie';
ACCEPT vPrix PROMPT 'entrez son prix';
ACCEPT vStock PROMPT 'entrez son stock';
--Le prix à l'achat est le premier prix connu pour le produit
Ret NUMBER;
BEGIN
Inserer_Produit(&vRefProd, '&vDesign',&vPrix, &vCat,&vPrix,&vStock,
&vPrix);

.....
END;
/
```

FONCTIONS

Principes

- Structure générale analogue à celle d'une procédure mais **retourne un résultat unique**
- Retour de résultat avec la **clause RETURN**
- Appels à partir :
 - d'une requête SQL
 - d'une procédure, fonction, programme PL/SQL
 - d'un programme externe

EXEMPLE DE CRÉATION ET EXÉCUTION DE FONCTION

Creation : Convertir un montant en Dinar vers un montant en Euro

```
create function toEuro(montant in number) return number  
is  
begin  
    return montant / 3;  
end;  
/
```

Execution

```
SQL> select toEuro(salary) from employee;
```

EXEMPLE DE CRÉATION ET EXÉCUTION DE FONCTION

Creation : Convertir un montant en Dinar vers un montant en Euro avec test

```
create or replace function toEuro (montant in varchar2) return number is
begin
    return to_number (montant) / 3;
exception
    when others then          -- si erreur SQL
        return null;
end;
/
```

Execution

```
SQL> select toEuro(salary) from employee;
```

RÉCAP FONCTIONS : STRUCTURE GÉNÉRALE

```
CREATE [OR REPLACE] FUNCTION <nom_fonction>
[(variable1 type1,...,variablen typen )]
RETURN type_resultat IS
-- déclarations des variables, curseurs, exceptions
BEGIN
-- instructions SQL ou PL/SQL

RETURN(variable);
EXCEPTION
...
END;
/
```

The diagram illustrates the general structure of a PL/SQL function. It shows the following components:

- CREATE [OR REPLACE] FUNCTION <nom_fonction> [(variable1 type1,...,variablen typen)] RETURN type_resultat IS**: The function declaration. A callout box labeled "Résultat à retourner typé" points to the `type_resultat` part.
- déclarations des variables, curseurs, exceptions**: Comments for variable declarations.
- BEGIN**: The start of the function body.
- instructions SQL ou PL/SQL**: Comments for SQL or PL/SQL instructions.
- RETURN(variable);**: The statement that returns the result. A callout box labeled "Retour du résultat" points to this line.
- EXCEPTION**: The start of the exception handling section.
- ...**: Ellipsis indicating more exception handling code.
- END;**: The end of the function definition.
- /**: The SQL prompt to execute the function.

APPEL DE FONCTION

A partir d'une requête SQL

Produits qui ne sont pas de la catégorie 'A'
ayant un prix supérieur au prix moyen
des produits de la catégorie 'A'

```
SELECT RefProd, Designation FROM Produit  
WHERE Cat != 'A' AND Prix > PrixMoyen ('A');
```

A partir d'une procédure ou fonction

```
BEGIN  
...  
CASE Prix_Moyen('A')  
WHEN Prix_Moyen('B') >= 5000 THEN...  
...  
END
```


SOUS-PROGRAMMES RÉCURSIFS

Même principe que dans les autres langages

```
CREATE FUNCTION Factorielle (-----)
-----

BEGIN
IF n=1 THEN RETURN 1;
ELSE
    -----;
END IF;

END Factorielle;
/
```



A callout box labeled "Appel récursif" points to the recursive call line in the SQL code.

SOUS-PROGRAMMES IMBRIQUÉS

Principes

- Décrit dans la partie déclarative d'un autre sous-programme
- Durée de vie limitée au temps d'exécution du sous-programme qui l'imbrique
- Dernier élément déclaré dans la partie déclarative du sous-programme qui l'imbrique

SOUS-PROGRAMMES IMBRIQUÉS : STRUCTURE

```
CREATE [OR REPLACE] PROCEDURE <nom_procedure>
[(paramètre1 [ IN |OUT | IN OUT ] TypeSQL
    [{:=|DEFAULT} expression]
[, .....

...)] IS
-- déclarations des variables et
-- curseurs utilisés dans le corps de la procédure
PROCEDURE <nom_procedure_imb> [Parametre....] IS
BEGIN
.....

END <nom_procedure_imb>;
BEGIN
...
    Execute nom_procedure_imb (...);
EXCEPTION
...
END;
/
```

Déclaration de la
procédure imbriquée

Appel de la procédure
imbriquée

SOUS-PROGRAMMES IMBRIQUÉS : EXEMPLE

```
CREATE OR REPLACE FUNCTION Puissance (a INTEGER, b INTEGER)
RETURN INTEGER IS
  Ps INTEGER:=1; i INTEGER
-----
```

```
  FUNCTION Produit (n1 INTEGER, n2 INTEGER)
  RETURN INTEGER IS
  P INTEGER:=1;
  BEGIN
  P:=n1*n2;
  RETURN P;
  END Produit;
-----
```

Définition de
la fonction imbriquée

```
  BEGIN
  ...
  FOR i IN 1..b-1 LOOP
  Ps:=Produit(Ps,a);
  END LOOP;
  END Puissance;
```

Appel de
la fonction imbriquée

/

PAQUETAGE

C'est quoi ?

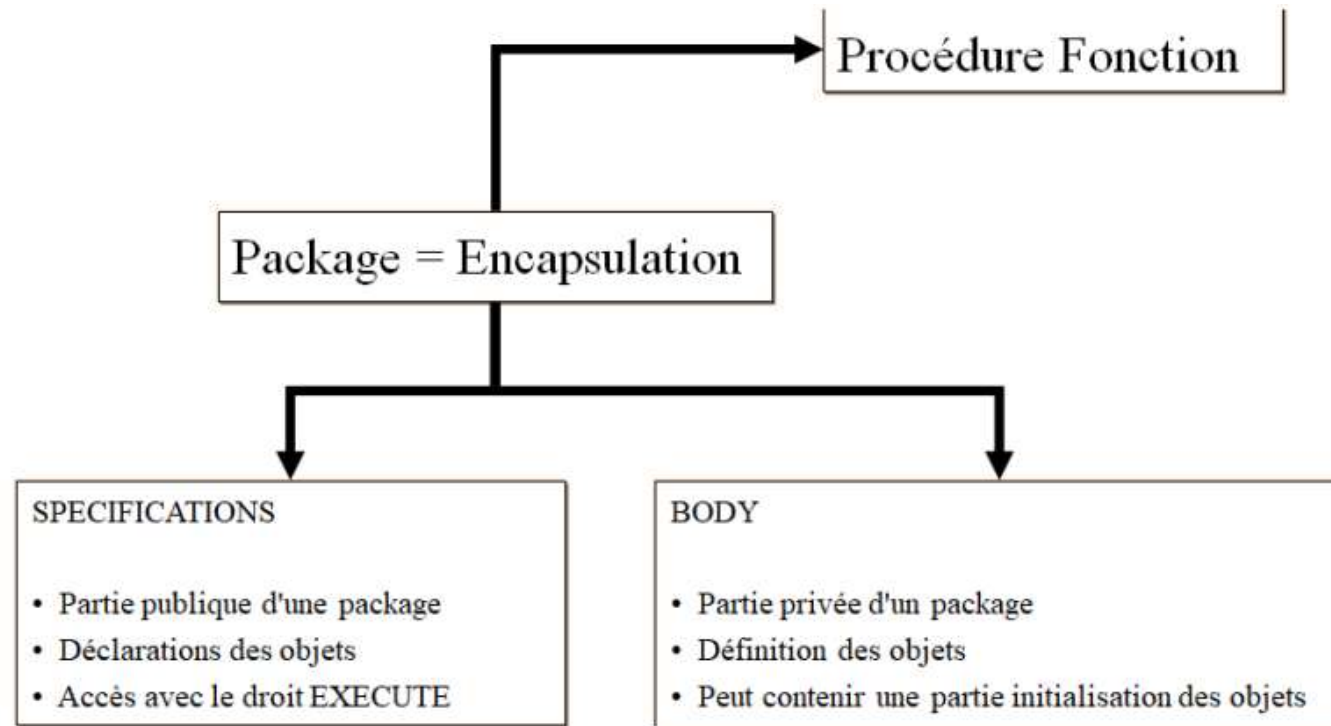
- Ensemble de différents objets (variables, exceptions, sous-programmes etc.) formant un ensemble homogène

Structure

- Spécification ou **partie visible**
 - Comprend les signatures des sous-programmes, la déclaration des variables, curseurs, exceptions etc.
 - Accessible au programme appelant, **PUBLIC**
- Corps ou **partie cachée ou implémentation ou BODY**
 - Corps des procédures ou des fonctions citées dans la partie spécification
 - Nouvelles procédures ou fonctions **privées** accessibles uniquement par des procédures ou fonctions du paquetage

PAQUETAGE

Les packages regrouperont des fonctions et des procédures fortement liées conçus pour répondre à des besoins spécifiques et permettre une utilisation autonome



PAQUETAGE : *PARTIE SPECIFICATION*

```
-- Partie spécification

CREATE [OR REPLACE] PACKAGE nom_package AS
Procedure Procedure1(liste des paramètres);
...
Function Fonction1(liste des paramètres)
                        return TYPE;
...
Variable_globale1 type1;
...
CURSOR Curseur_global1 IS...
...
END nom_package;
/
```


PAQUETAGE : *PARTIE BODY*

```
-- Partie body

CREATE [OR REPLACE] PACKAGE BODY nom_package AS
PROCEDURE Procedure1(liste des paramètres) IS
...
BEGIN
...
END Procedure1;
FUNCTION Fonction1(liste des paramètres)
                                RETURN type IS
...
BEGIN
...
RETURN (...);
END Fonction2;
END nom_package;
/
```

PAQUETAGE : STRUCTURE GÉNÉRALE

- Un package est composé d'un en tête et d'un corps :
 - L'en tête comporte les types de données définis et les prototypes de toutes les procédures (fonctions) du package.
 - Le corps correspond à l'implémentation (définition) des procédures (fonctions).
- Le premier END marque la fin de l'en tête du package. Cette partie doit se terminer par / pour que l'en tête soit compilé.
- Le corps (body) du package consiste à implémenter (définir) l'ensemble des procédures ou fonctions qui le constitue. Chaque procédure est définie normalement avec ses clauses BEGIN ... END.
- Le package se termine par / sur la dernière ligne.

PAQUETAGE : EXEMPLE

```
CREATE PACKAGE Produit AS

-- Procédure publique Insertion
PROCEDURE Insérer_Produit (vRefProd Produit.RefProd%TYPE,
vDesign Produit.Designation%TYPE, vCat Produit.Cat%TYPE,
vPrix Produit.Prix%TYPE, vStock Produit.Stock%TYPE,
vPrixAch Produit.PrixAch%TYPE);

-- Procédure publique Commande
PROCEDURE Commande(vRefComm Produit.RefComm%TYPE,
vRefProd Produit.RefProd%TYPE, vQte
LigneCommande.QtComm);

-- Fonction publique Prix

FUNCTION Prix_Moyen (vcat Produit.Cat%TYPE) RETURN NUMBER

END Produit;
/
```

PAQUETAGE : EXEMPLE

```
--CREATE PACKAGE BODY Employe AS

PROCEDURE Insérer_Produit (vRefProd Produit.RefProd%TYPE,
vDesign Produit.Designation%TYPE, vCat Produit.Cat%TYPE,
vPrix Produit.Prix%TYPE, vStock Produit.Stock%TYPE,
vPrixAch Produit.PrixAch%TYPE) IS
--déclarations
BEGIN
...
END Insere_Produit;

PROCEDURE Commande(vRefComm Produit.RefComm%TYPE,
vRefProd Produit.RefProd%TYPE, vQte LigneCommande.QtComm) IS
--déclarations
BEGIN
...
END Commande;

Prix_Moyen (vcat Produit.Cat%TYPE) RETURN NUMBER IS
--déclarations
BEGIN
...
END Prix_Moyen;
```

UTILISATION DE PAQUETAGE

- Création du package :
 - <SQL> @ chemin du fichier sql
 - Exécution de procedure dans un package
 - <SQL> exec nom-pack.nom procedure(param);
-
- Utilisation du package dans un bloc PLSQL