PL SQL

Séance 5

Pr. M’barek ELHALOUI

Révision/ Questions

- Curseur implicite
- Curseur explicite
- Etapes d'utilisation d'un curseur (Declare, Open, Fetch, Close)
- Gestion des exceptions
- Questions ?

Les Procédures et Fonctions

- **Définition** :
  - Une procédure ou procédure stockée est un bloc PL/SQL nommé, défini par l'utilisateur et stocké dans la base de données.
  - Une fonction est identique à une procédure à la différence qu'elle retourne une valeur.

- **Avantages des procédures / Fonctions :**
  - Stockées coté serveur : Appel par API, Gains en trafic
  - Code précompilé : Meilleures performances
  - Partage et optimisation du code : Plusieurs utilisateurs, Plusieurs applications
  - Sécurité : accès aux données peut être retreint via des procédures
  - Accessibilité depuis les applications et outils ayant interface avec Oracle SQL*PLUS, Pro*C, SQL*Forms, Pro*Cobol,…

Les Procédures

- **Syntaxe :**

  CREATE [OR REPLACE] PROCEDURE *NOM_PROC* [(PARAMETRES [Mode] type, ...)]

  [IS | AS]

  BGIN … END; (Corps de la procedure)

- **CREATE :** indique que l'on veut créer une procédure stockée dans la base
- **OR REPLACE :** facultative. Permet d'écraser une procédure existante portant le même nom
- **NOM_PROC :** le nom donné par l'utilisateur à la procédure
- **Nom_PARAM :** Nom donné par l'utilisateur au paramètre transmis
- **Mode :** définit si le paramètre formel est en entrée (IN), en sortie (OUT) ou en entré-sortie (IN OUT). Par défaut : IN
- **Type :** Type(s) SQL ou PL SQL du (des) paramètre(s) de la procédure
- **Corps de la procedure :** Bloc PL SQL de la procedure

Les Procédures

- **Exemple 1 : Procédure Bonjour**

```
-- Procedure bonjour

create or replace procedure bonjour
is
begin
    dbms_output.put_line('Bonjour');
end;
/
```

- **Exécution de la procédure**

```
-- Execution sur un client SQL Plus
SQL> execute bonjour;
Bonjour
```

```
-- Appel dans un programme
SQL> begin
  2       bonjour;
  3  end;
  4  /
Bonjour
```

Les Procédures

- **Exemple 2 : Procédure Augmenter Salaire**

```
-- Procedure Augmentation de salaire avec un taux

create or replace procedure augmenterSalaire(
    numE in Employe.Empl_id%type,
    taux in number
) is
begin
   -- Mise à jour salaire employé
   update Employe
      set salary = salary + salary * taux / 100
      where Empl_id = numE;
end;
```

Exécution

```
--Appel dans un programe

DECLARE
code  Employe.Id_Employe%Type  ;

begin
     code := '705';
    augmenterSalaire (code, 10) ;

end;
```

Les Procédures

- **Exemple 3 :**

Employés affectés à des départements :

```
DEPT(DEPT_ID, DEPT_NOM)
EMPLOYE(EMP_ID, EMP_NOM, EMP_PRENOM, EMP_DEPT_ID)
```

- Insertion dans la table employé :

```
create or replace
procedure ajouter_emp(leNom employe.emp_nom%type,
                      lePrenom employe.emp_prenom%type) is
begin
  -- on utilise la séquence emp_seq pour générer la clé
  insert into employe(emp_id, emp_nom, emp_prenom)
  values(emp_seq.nextval, upper(leNom), upper(lePrenom));
end ajouter_emp;
```

- Exécution dans un client SQLPlus :

```
execute ajouter_emp('caron','anne');
--> a inséré (1,'CARON','ANNE')
```

Les Fonctions

- **Syntaxe :**

```
CREATE [OR REPLACE] FUNCTION NOM_FCT [(PARAMETRES [Mode] type, ...)] RETURN TYPE_RETOUR_FCT ]
[IS | AS]
BEGIN
……..
RETURN RESULTAT;
END [NOM_FCT];
/
```

- **RETURN TYPE_RETOUR_FCT** : indique le type de résultat de la fonction (signature de la fonction).
- Les autres paramètres sont identiques à ceux d'une procédure
- Une fonction diffère d'une procédure par le fait qu'on peut l'utiliser dans une expression.
- Les paramètres sont toujours passés avec le type IN.

Les Fonctions

- **Exemple 1 : Fonction carré**

```
-- Fonction Carré

create or replace fuction carre (n in number) return number
is
begin
     return n*n;
end;
```

```
DECLARE
x number  ;

begin
    x:=&nbre; /* la variable nbre doit etre lu auparavant via la commande
                "accept nbre prompt ('saisissez un nombre:');" */
     DBMS_OUTPUT.put_line('Le carré de :'|| x 'est' || carre(x));

end;
```

Les Fonctions

- **Exemple 2 : Surface d'un Cercle**

```
-- Fonction Carré

create or replace fuction carre (n in number) return number
is
begin
     return n*n;
end;
```

```
-- Programme surface cercle
DECLARE
R number  ;
S number  ;
begin
    R:=&Ray; /* la variable Ray doit etre lu auparavant via la commande
                "accept Ray prompt ('saisissez le rayon du cercle:');" */
    if R> 0 then
    S:= 3,14*carre(R);
    DBMS_OUTPUT.put_line('La surface du cercle est :' || S);
     end if;
end;
```

Les procédures et fonctions

- **Recompilation d'une procédure ou d'une fonction :**

- ALTER PROCEDURE nom_procédure COMPILE;
- ALTER FUNCTION nom_fonction COMPILE;

- **Exécution d'une procédure ou d'une fonction à l'aide de "execute" :**

- EXECUTE nom-procédure (Paramètres);
- EXECUTE : nom-variable := nom-fonction (Paramètres);
- EXECUTE DBMS_OUTPUT.PUT_LINE(nom-fonction(Paramètres));

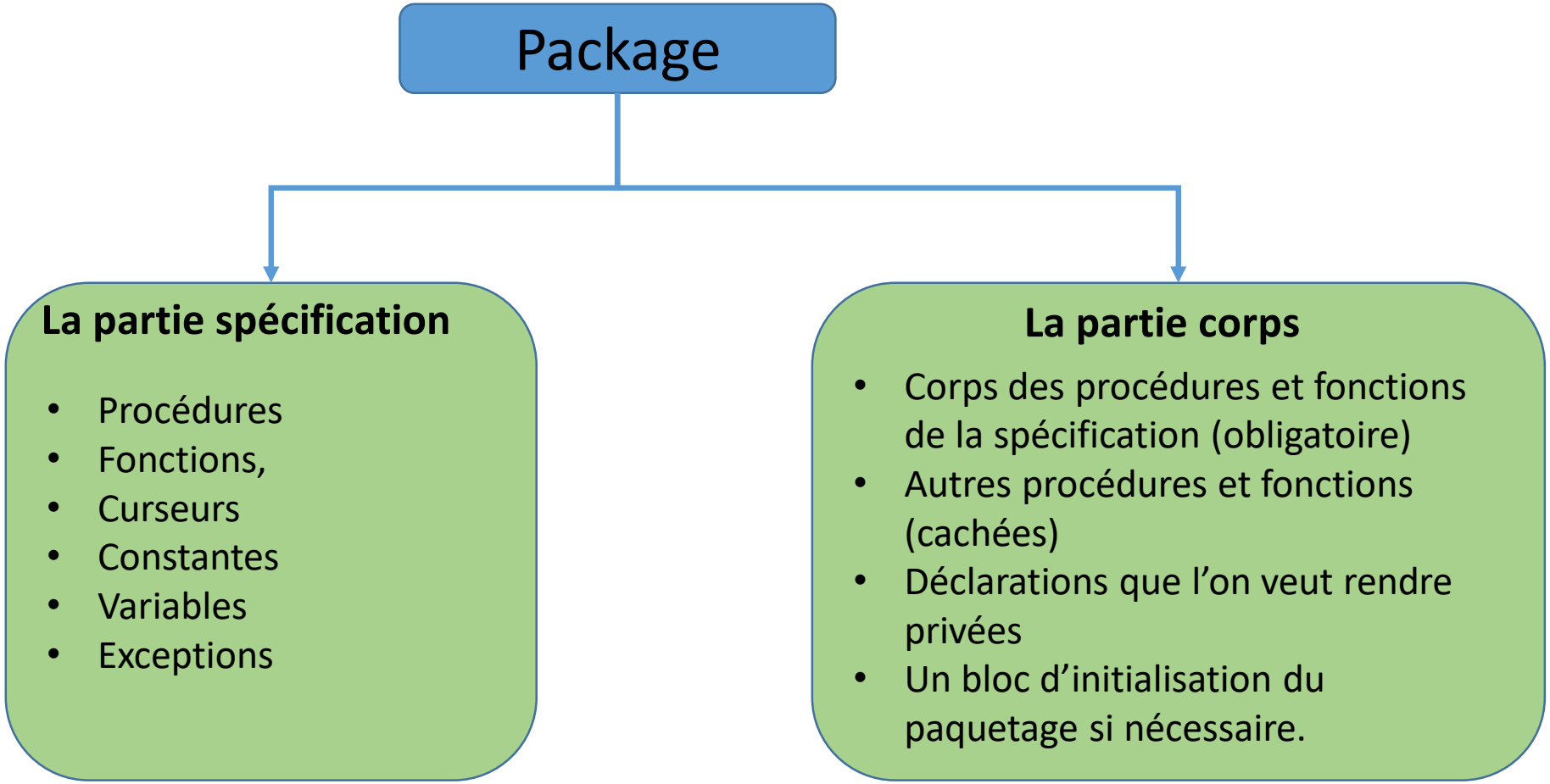
- **Suppression d'une procédure ou d'une fonction :**

- DROP PROCEDURE nom_procédure ;
- DROP FUNCTION nom_fonction ;

Les Packages

- Un package ou paquetage est un regroupement de procédures et de fonctions.
- Un package est constituée d'une déclaration ou spécification (la partie visible) et d'un corps (l'implémentation)
- La spécification du package contient des éléments que l'on rend accessibles à tous les utilisateurs du package .
- Le corps du package contient l'implémentation et ce que l'on veut cacher.
- Au niveau des droits, on donne le droit d'exécuter un package : on a alors accès à toute la spécification, pas au corps.
- Il faut mettre le minimum de choses dans la spécification (penser à encapsulation)
- La surcharge est autorisée, i.e. on peut avoir plusieurs procédures ou fonctions de même nom, avec des signatures différentes.
- Chaque session qui utilise le package possède 1 instance du package (il n'y a donc pas partage des variables)

Les Packages



Les Packages

- **Syntaxe**

CREATE OR REPLACE PACKAGE ***nom_package***

IS définitions des types utilisés dans le package;

prototypes de toutes les procédures et fonctions du package;

END ***nom_package;***

/

CREATE OR REPLACE PACKAGE BODY nom_package

IS *déclaration de variables globales;*

définition de la procédure 1 ;

définition de la procédure 2;

…

END nom_package;

/

Les Packages : Exemple

- **Spécification**

```
SQL> CREATE OR REPLACE PACKAGE GEST_EMP AS
 2            PROCEDURE EMP_LE_PLUS_PAYE(N OUT E_EMPLOYE.NOM%TYPE, S OUT
                                                E_EMPLOYE.SALAIRE%TYPE);
 3            FUNCTION  MASSE_SALAIRE RETURN NUMBER;
 4  END GEST_EMP;
 5  /

Package créé.

SQL>
```

Les Packages : Exemple

- **Corps**

```
SQL> CREATE OR REPLACE PACKAGE BODY GEST_EMP AS
 2
 3              PROCEDURE EMP_LE_PLUS_PAYE(N OUT E_EMPLOYE.NOM%TYPE, S OUT E_EMPLOYE.SALAIRE%TYPE)
 4              IS
 5              BEGIN
 6                              SELECT E.NOM, E.SALAIRE INTO N, S
 7                              FROM E_EMPLOYE E
 8                              WHERE E.SALAIRE >= ALL (SELECT SALAIRE FROM E_EMPLOYE);
 9              END EMP_LE_PLUS_PAYE;
10
11              FUNCTION MASSE_SALAIRE RETURN NUMBER
12              IS
13                              MAS    NUMBER:=0;
14              BEGIN
15                              SELECT SUM(SALAIRE) INTO MAS
16                              FROM E_EMPLOYE;
17
18                              RETURN MAS;
19              END MASSE_SALAIRE;
20  END GEST_EMP;
21  /

Corps de package créé.
```

Exercices

- Créez une fonction qui retourne l'âge d’un employé dont le matricule est donné comme paramètre.
- Créez une procédure stockée qui affiche les départements qui ont plus de 5 employés âgés de plus 56 ans.