esprit
Ecole Supérieure Privée
d'Ingénierie et de Technologies

Cours SGBD

CHAPITRE 6.1 STRUCTURE D'UN BLOC PL/SQL

INTRODUCTION

- PL/SQL est une extension procédurale à SQL:
- C'est le langage procédural d'Oracle capable d'envoyer au noyau Oracle tous les ordres SQL
- Offre des structures algorithmiques classiques :

  Affectation, expression conditionnelle, boucle, etc…
- Offre des structures procédurales :

  Sous-programmes locaux, procédures stockées, fonctions et des déclencheurs
- Langage utilisé pour les produits Oracle

BLOC PL/SQL(1)

Un bloc PLSQL est composé de trois parties:

DECLARATION

DECLARE - Optionnelle

déclaration variables, constantes, types, curseurs,...

EXECUTABLE

BEGIN - Obligatoire

contient le code PL/SQL

.....

EXCEPTION - Optionnelle

END - Obligatoire;

/

BLOC PL/SQL(2)

Dans le code **PL/SQL** (partie procédural) nous pouvons avoir :

- des structures conditionnelles (IF-THEN-ELSE / IF-THEN-ELSIF / CASE)
- des structures répétitives (WHILE, LOOP, FOR,...)
- des ordres LMD (SELECT, UPDATE, DELETE, INSERT,...)
- des appels de fonctions, procédures, packages,...

BLOC PL/ SQL(3)

EXEMPLE

```
DECLARE
   v_ename varchar(20);
BEGIN
   Select last_name into v_ename  from
   employees where employee_id =7839;
END
;
/
```

- Placer un point virgule (;) à la fin de d'une instruction SQL ou PL/SQL
- Utiliser un slash (/) pour exécuter un bloc PL/SQL anonyme

SECTION DÉCLARATION

Débute par le mot **DECLARE** et permet la déclaration des variables et constantes.

EXEMPLE

```
V_No_Produit      Number ;
V_Designation     VARCHAR2(20) ;
V_PU              NUMBER(6,2) := 100.00;
```

SECTION EXÉCUTABLE

Intègre différents types d'instructions :

- Instruction d'affectation
- Instruction de contrôle de flux
- Instruction SQL
- Instruction curseur

SECTION EXCEPTION

- Traite les erreurs qui se produisent pendant l'exécution d'un programme PL/SQL
- Peut être prédéfinie :
  - DUP_VAL_ON_INDEX : insertion d'une clé dupliquée
  - NO_DATA_FOUND : instruction select ne retournant aucune ligne
  - ...
- Peut être définie : traiter par le programmeur.

TYPES DE BLOCS PL/ SQL(1)

Blocs anonymes

- Blocs PL/SQL non nommés, imbriqués dans une application ou créés de façon interactive.
- logés dans le serveur d'application (poste devlopp).
- Ils sont déclarés dans une application à l'endroit de leur exécution.
- Ils sont disponibles dans tous les environnements PL/SQL.

```
[DECLARE
<Déclarations>]
BEGIN
<Instructions>
[EXCEPTION
<Traitements des erreurs>]
END ;
```

TYPES DE BLOCS PL/ SQL (2)

Procédures et fonctions stockées

- Blocs PL/SQL nommés et stockées sous forme d'objet de base de données.
- Résident sur le serveur de base de données Oracle.
- Peuvent utiliser des paramètres d'entrée et de sortie.
- Sont invoquées d'une manière répétitive.

PROCEDURE nom (paramètres) IS
<déclaration>
BEGIN
<instructions>
[EXCEPTION]
<traitement des exceptions>
END ;
	FUNCTION nom(paramètres)
	RETURN type_donnée IS
	<déclaration>
	BEGIN
	<instructions>
	[EXCEPTION]
	<traitement des exceptions>
	END

GESTION DES VARIABLES

- Les variables sont déclarées et initialisées dans la section déclaratives
- De nouvelles valeurs sont affectées aux variables dans la section exécutable
- Passage des valeurs à des programmes par l'intermédiaire de paramètres

RÈGLES DE DÉNOMINATIONS DES VARIABLES

- Deux variables peuvent porter le même nom si elles sont dans des blocs distincts
- Les noms des variables doivent être différents des noms des colonnes et des tables utilisés dans le bloc.
- L'identifiant ne doit pas dépasser 30 caractères. Le premier caractère doit être une lettre, les autres peuvent être des lettres, des nombres ou des caractères spéciaux.

VARIABLES PL/ SQL

- INTEGER -- max 38chiffres
- NUMBER -- max 125 chiffres
- CHAR;-- Max 32767 caractères
- VARCHAR2; --
- DATE; -- 4712 avant AVJC à 9999 APJC
- BOOLEAN :TRUE,FALSE,NULL
- %TYPE : type de variable équivalent au type de colonne d'une table ou d'une autre variable
- %ROWTYPE : type de variable équivalent à une ligne d'une table

DÉCLARATION DES VARIABLES(EXEMPLE 1)

```
Declare
v_remise CONSTANT  real := 0.10;
v_hiredate          date;
g_deptno            number(2) NOT NULL := 10;
v_integer  integer; -- max 38chiffres
v_number 1 number; -- max 125 chiffres
v_number2  number(38,3);
v_bool boolean; -- valeurs possibles TRUE, FALSE, NULL et NOT NULL
v_varchar2 varchar2(20); -- Max 32767 caractères
v_date date; -- Les dates peuvent aller de -4712 avant AVJC à 9999 APJC
```

DÉCLARATION DES VARIABLES(EXEMPLE 2)

```
Declare
v_ename        emp.ename%type;
v_emp    emp%rowtype;
v_n1     number(5,3);
V_n2     v_n1%type;
```

Les enregistrements PL/SQL sont des types composites définis par l'utilisateur. Pour les utiliser :

1. Vous devez déclarer l'enregistrement dans la section déclarative d'un bloc PL/SQL. La syntaxe de définition de l'enregistrement est illustrée dans la diapositive ci-dessus.
2. Vous devez déclarer et éventuellement initialiser les composants internes de ce type d'enregistrement.

LE TYPE RECORD

- Les enregistrements PL/SQL sont des types composites définis par l'utilisateur. Pour les utiliser :
  1. Vous devez déclarer l'enregistrement dans la section déclarative d'un bloc PL/SQL.
  2. Vous devez déclarer et éventuellement initialiser les composants internes de ce type d'enregistrement

1

```
TYPE type_name IS RECORD
      (field_declaration[, field_declaration]…);
```

2

```
identifier    type_name;
```

LE TYPE RECORD : EXEMPLE

Déclarez des variables pour le stockage du nom, du poste et du salaire d'un nouvel employé.

Exemple :

```
...
  TYPE emp_record_type IS RECORD
    (last_name   VARCHAR2(25),
     job_id      VARCHAR2(10),
     salary      NUMBER(8,2));
  emp_record     emp_record_type;
...
```

INITIALISATION DES VARIABLES

- Opérateur d'affectation (:=)
- Mot clé DEFAULT
- Contrainte NOT NULL

EXEMPLES

```
v_integer number  := 12345;
v_bool  Boolean :=TRUE;
v_char  varchar(30) NOT NULL := 'SGBD';
v_date  date DEFAULT  '01-Janv-2009';
```

LES STRUCTURES DE CONTRÔLE(1): ALTERNATIVES

- IF condition THEN instructions; END IF;
- IF condition THEN instructions1;

  ELSE instructions2;

  END IF;
- IF condition1 THEN instructions1;

  ELSIF condition2;

THEN instructions2;

  ELSIF ... ELSE instructions N;

END IF;

LES STRUCTURES DE CONTRÔLE(2)

EXEMPLES

```
DECLARE
v1 integer  := 1100;
v2 integer  := 200;
BEGIN
   IF v1 < v2 THEN
      dbms_output.put_line('v1 < v2');
   ELSE
      dbms_output.put_line('v2 <= v1');
   END IF;
END;
/
```

LES STRUCTURES DE CONTRÔLE(3)

```
DECLARE
v1 number:= 685;
v2 number := 125;
V3 number :=870;
BEGIN
   IF v1 < v2 THEN
        IF  v2<v3 then
                dbms_output.put_line('v1 < v2 < v3');
        ELSIF v3 < v1 then
                dbms_output.put_line('v3 < v1 < v2');
        ELSE
                 dbms_output.put_line('v1 <=v3 < v2');
        END IF;
   ELSIF v1 < v3 then
                 dbms_output.put_line('v2 < v1 < v3');
   ELSIF v3<v2 then
                 dbms_output.put_line('v3 < v2 <= v1');
   ELSE
                 dbms_output.put_line('v2 <= v3 <= v1');
  END IF;
END;
/
```

LES STRUCTURES DE CONTRÔLE :CHOIX

```
CASE expression
WHEN valeur1 THEN instructions1;
WHEN valeur2 THEN instructions2;
…
ELSE instructionsN;
END CASE;
```

```
CASE
WHEN expr1 THEN instructions1;
WHEN expr2 THEN instructions2;
…
ELSE instructionsN;
END CASE;
```

LES STRUCTURES DE CONTRÔLE ITÉRATIVES(1) : WHILE ...LOOP

```
WHILE condition LOOP
   instructions;
END LOOP;
```

EXEMPLES

```
DECLARE
v1 integer  :=1;
BEGIN
        WHILE v1 <10  LOOP
          dbms_output.put_line(v1);
          v1 := v1+1;
        END LOOP;
END;
/
```

LES STRUCTURES DE CONTRÔLE ITÉRATIVES(2) :LOOP

```
LOOP
instructions;
EXIT [WHEN condition];
instructions;
END LOOP;
```

EXEMPLES

```
DECLARE
v1 integer  :=1;
BEGIN
        LOOP
          dbms_output.put_line(v1);
          EXIT WHEN  v1 =10;
          v1 := v1+1;
        END LOOP;
END;/
```

LES STRUCTURES DE CONTRÔLE ITÉRATIVES(3) :FOR … LOOP

```
FOR compteur IN inf..sup LOOP
instructions;
END LOOP;
```

EXEMPLES

```
DECLARE
v1 integer  :=1;
BEGIN
  FOR v1 IN 1..10 LOOP
    dbms_output.put_line(v1);
  END LOOP;
END;
/
```

INSTRUCTION SELECT DANS UN BLOC PL/SQL

Utilisation de la clause INTO pour identifier les variables PL/SQL qui doivent recevoir des valeurs des colonnes des tables d'une base de données.

```
DECLARE
  v_col1 …
  v_col2 …
   …
  v_coln …

BEGIN
  SELECT {*|col1,col2, ….coln}  INTO v_col1, v_col2, …
  v_coln
  FROM  table1, table2 , …
  WHERE condition
```

INSTRUCTION SELECT DANS UN BLOC PL/SQL : EXEMPLE (1)

```
DECLARE
v_emp  employees%ROWTYPE;
BEGIN
    dbms_output.enable;
    Select  * into v_emp From employees where
employee_id = '100';
    dbms_output.put_line('Nom employé :' ||
v_emp.last_name || chr(10)||
    'Fonction : ' || v_emp.job_id || chr(10)||
    'Departement : '|| v_emp.department_id ||
chr(10)||
    'Date recrutement  : '||  to_char(v_emp.hire_date,
'dd/mm/yyyy') || chr(10) ||
     'Salaire : ' || v_emp.salary );
END;
/
```

INSTRUCTION SELECT DANS UN BLOC PL/ SQL : EXEMPLE (2)

```
Declare
salaire_moy  employees.salary%type;
Begin
Select avg(salary) into salaire_moy From
employees
  Where department_id=10;
  dbms_output.put_line('Le salaire moyen des
employés du   département 10 est  : ' ||
to_char(salaire_moy));
End ;
/
```