

PL/SQL

Seul avec sql, on ne peut pas faire des traitements

D'où l'apparition du **PL/SQL** qui représente :

- Un langage procédural d'Oracle
- depuis la version 6
- étend SQL :
 - Cohabite (boucles, conditions...) avec des instructions SQL (select, update, delete ...)
 - but : stocker **des traitements complexes** autant qu'objets sur le serveur de base de données (ex procédures , déclencheurs..).

Avantages :

- ☐ Intégration parfaite du SQL
- ☐ Très bonnes performances ;
- ☐ Portabilité ;
- ☐ Facilité de programmation ;

Déclaration, initialisation des variables

- **Déclaration et initialisation :**
`nom_variabl et type_variable := valeur;`

~~Déclarations multiples interdites :~~
~~i, j integer;~~

▪ **Les types de données**

-Types SQL :Number, Varchar2, varchar, char, date

-le type colonne : nom_table.champ%**type**

-le type ligne (tuples): nom_table%**rowtype**

Ex

var2 vnom%TYPE;

ligne_etudiant Etudiant%rowtype ;

Pour affecter les données :

□ - ligne_etudiant.nom := 'Ali' ;

□ - ligne_etudiant.année := 1 ;

Affectation des valeurs aux variables PL/SQL

SELECT col1, col2,... INTO var_recep1, var_recep2 FROM ma_table

SELECT * INTO var_rowtype FROM ma_table;

NB : SELECT doit obligatoirement ramener **une seule ligne et une seule** : sinon erreur

Pas aucune (NO_DATA_FOUND) et pas plusieurs (TOO_MANY_ROWS)

Variable tableau (type TABLE)

Syntaxe :

- 1- TYPE nomTypeTableau IS TABLE OF
{ TypeScalaire | variable%TYPE | table%ROWTYPE | Table.colonne%type }
[INDEX BY BINARY_INTEGER]; (facultative depuis la V8)
- 2- - nomTable nomTypeTableau;

Initiation

```
BEGIN
    tab_brevets(-1)      := 'PL-1';
    tab_brevets(-2)      := 'PL-2';
    tab_nomPilotes(7800) := 'Bidal';
    tab_pilotes(0).brevet := 'PL-0';
END;
```

Les Fonctions

Fonction	Description
EXISTS (x)	Retourne TRUE si le x ^e élément du tableau existe.
COUNT	Retourne le nombre d'éléments du tableau.
FIRST / LAST	Retourne le premier/dernier indice du tableau (NULL si tableau vide).
PRIOR (x) / NEXT (x)	Retourne l'élément avant/après le x ^e élément du tableau.
DELETE	Supprime un ou plusieurs éléments au tableau.
DELETE (x)	
DELETE (x, y)	

Insertion d'enregistrement

INSERT INTO **Table** (column_2, column_4) **VALUES** ('Explicit value', 'Explicit value');

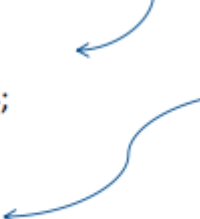
Ex :

INSERT INTO emp values r_EenregEmp || INSERT INTO emp values(r_EenregEmp.numemp,...)

Modification

```
UPDATE emp  
SET sal = sal+ v_sal  
WHERE numemp= 124;
```

```
UPDATE emp  
SET sal = sal +140  
WHERE numemp=123;
```



Suppressions

```
DECLARE  
v_salmin NUMBER(4) := 1000;  
BEGIN  
  DELETE FROM emp WHERE sal<v_salmin ;  
END;
```

Conflits de noms

Si une variable porte le même nom qu'une colonne d'une table, c'est la colonne qui l'emporte

```
Exemple ;  
  DECLARE  
    name varchar(30) := 'toto';  
  BEGIN  
    DELETE FROM emp where name = name;  
  ...
```

➡ ⚠ ici DELETE entraine la suppression de tous les employés et nom seulement l'employé 'toto'

Pour éviter ça, le plus simple est de ne pas donner de nom de colonne à une variable

Variables de substitution

Des variables définies sous **SQL*PLUS** qui sont passées en paramètre d'entrée à un **bloc PL/SQL**

Il faut préfixer le nom de la variable par le symbole **(&)** pour y accéder dans le bloc PL/SQL

```

SET SERVEROUTPUT ON
ACCEPT vnum PROMPT 'entrer le code : '
ACCEPT vaugm PROMPT 'Entrer l\'augmentation : '
declare
    vnom emp.ename%type;
    vsal emp.sal%type;
BEGIN
    SELECT ename,sal into vnom,vsal FROM emp where empno=&vnum;
    vsal := vsal+&vaugm;

```

Variables de session

Globales : accessible depuis tous les sous pgm du package, seul dans la **session courante**

- ☐ On utilise la directive **VARIABLE**
- ☐ Au moment de l'affectation préfixer le nom de la variable par (:)
- ☐ L'affichage sous SQL*PLUS est réalisé par la directive **PRINT**

Application Exemple :

```

VARIABLE g_compteur NUMBER;
DECLARE
    vincr number := 100;
BEGIN

    :g_compteur := vincr+1;
END;
/

=====apres execution, Affichage sous SQL*PLUS
SQL> PRINT g_compeur ;

```

Structures conditionnelles

Instruction NULL : aucune action n'est exécutée.

IF -THEN

```

IF condition THEN
    instructions;
END IF;

```

IF .. THEN-ELSE

```

IF condition THEN
    instructions1;
ELSE
    instructions2;
END IF;

```

IF -THEN-ELSEIF

```

IF condition1 THEN
    instructions1;
ELSIF condition2 THEN
    instructions2;
ELSIF ...
...
ELSE
    instructionsN;
END IF;

```

- If i<10 then
- i :=i+1;
- Else
- NULL;
- End if;

Structure CASE

Syntaxe :

```
CASE expression
  WHEN expr1 THEN instruction1;
  WHEN expr2 THEN instruction2;
  ...
  ELSE instructionN;
END CASE;
```

Ou :

```
CASE
  WHEN condition1 THEN
    instructions1;
  WHEN condition2 THEN
    instructions2;
  ...
  ELSE instructionsN;
END CASE;
```

Exemple :

```
DECLARE
  vmention char(2);
  vnote NUMBER(4,2);
BEGIN
  ...
  CASE
    WHEN vnote >=16 THEN
      vmention:='TB';
    WHEN vnote >=14 THEN
      vmention:='B';
    WHEN vnote >=12 THEN
      vmention:='AB';
    WHEN vnote >=10 THEN
      vmention:='P';
    ELSE vmention:='R';
  END CASE;
  ...
```

50

Structures répétitives LOOP -> EXIT WHEN

Syntaxe :

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

- Obligation d'utiliser la commande EXIT

Exemple :

```
DECLARE
  Vsomme NUMBER(4):=0;
  Ventier NUMBER(3):=1;
BEGIN
  LOOP
    vsomme:=vsomme+ventier;
    ventier:=ventier+1;
    EXIT WHEN ventier>100;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('SOMME=|| vsomme);
END;
/
```

WHILE LOOP -> END LOOP

Syntaxe :

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

FOR LOOP -> END LOOP

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