

# Chapitre 3 : Introduction au PL/SQL





# Pourquoi PL/SQL

- › Le langage **SQL** est un langage **déclaratif non procédural** permettant d'exprimer des requêtes dans un langage relativement simple.
- › Il **n'intègre aucune structure de programmation** permettant par exemple d'exécuter une boucle itérative.
- › Les traitements complexes sont parfois difficiles à écrire si **on ne peut utiliser des variables et les structures de programmation comme les boucles et les alternatives**.
- › On ressent vite le **besoin** d'un **langage procédural** pour **lier plusieurs requêtes SQL** avec des variables et dans les structures de programmation habituelles.



# Pourquoi PL/SQL

› PL/SQL permet :

- l'utilisation de **variables permettant l'échange d'information** entre les requêtes SQL et le reste du programme
- la création **d'ordres SQL dynamiques**
- des **traitements plus complexes** en définissant **des procédures et des fonctions**
- la gestion **des cas particuliers et des erreurs** (traitement des **exceptions**)
- L'utilisation de **structures itératives et alternatives**
- l'utilisation de **librairies standards prédéfinies**
- Le déclenchement de traitements spécifiques suite à des instructions (insert, update, select) en utilisant **les déclencheurs**.



# C'est quoi PL/SQL ?

- › PL/SQL(Procedural Language / Structured Query Language)
- › PL/SQL est un langage de programmation procédural et structuré pour développer des applications autour de bases de données relationnelles
- › Il intègre parfaitement le langage SQL en lui apportant une dimension procédurale.
- › Des requêtes SQL cohabitent avec les structures de contrôle habituelles de la programmation structurée (blocs, alternatives, boucles)



# Normalisation du langage PL/SQL

- › **PL/SQL** est un langage propriétaire de **Oracle**
- › **Pas de véritable standard**, la plupart des SGBD relationnels propose des **L4G** (langage de 4ième génération) spécifiques, semblables à PL/SQL :
  - **Microsoft/SQL server** et **Sybase** propose **Transact-SQL (T-SQL)**
  - **PostgreSQL** propose **PL/pgSQL** très proche de PL/SQL
  - **MySQL** propose un langage plus limité : **SQL/PSM**
  - **IBM DB2** propose un dérivé de PL/SQL : **SQL-PL**



# Structure d'un Programme : La notion de bloc

- › Le langage PL/SQL permet de définir un ensemble de commandes contenues dans ce que l'on appelle un "**bloc**" PL/SQL.
- › On peut distinguer deux types de blocs:
  - Bloc «**anonyme**», il correspond à une séquence qui s'exécute à l'endroit où elle existe.
  - Bloc «**nommé**», qui peut être une procédure ou une fonction, pouvant être appelée autant de fois que nécessaire.
- › Les blocs PL/SQL peuvent être utilisés dans les outils de la famille Oracle (Sql\*Plus, Sql\*Developer, Sql\*Pro, etc.)



# Structure d'un Programme : La notion de bloc

- › Chaque bloc PL/SQL peut être constitué de 3 sections:
  - Une section ***facultative*** (*optionnelle*) de déclaration et initialisation de types, variables et constantes
  - Une section ***obligatoire*** contenant les instructions d'exécution
  - Une section ***facultative*** de gestion des erreurs



# Structure d'un Bloc PL/SQL

## [DECLARE]

- définitions de variables, constantes, exceptions, curseurs

## BEGIN

- les instructions à exécuter (ordres SQL, instructions PL/SQL, structures de contrôles)

## [EXCEPTION]

- la récupération des erreurs (traitement des exceptions)

END;

Seuls BEGIN et END  
sont obligatoires

Les blocs et les  
instructions se  
terminent par un « ; »



# Section de déclaration

- › Débute par le mot clé ***DECLARE***
- › Section déclarative optionnelle (facultative) d'un bloc
- › Elle contient toutes les déclarations et initialisation de types, variables et constantes.
- › Cette section ne doit pas contenir d'instructions exécutables.
- › Toute variable doit avoir été déclarée avant de pouvoir être utilisée dans la section exécutable.



# Section de déclaration

› Exemple:

**DECLARE**

Var1 INTEGER := 15;

**BEGIN**

Var1 := Var1 + 3;

**END;**



## Section exécutable

- › Débute par le mot clé ***BEGIN*** et se termine par ***END***
- › Section obligatoire d'un bloc
- › Elle contient les instructions de programme et la section de gestion des erreurs
- › Chaque instruction doit être suivi du terminateur d'instruction ;



# Section exécutable

› Exemple :

```
DECLARE
```

```
    Var1 INTEGER := 15;
```

```
BEGIN
```

```
    Var1 := Var1 + 3;
```

```
END;
```



# Section de gestion des exceptions

- › Débute par le mot clé ***EXCEPTION***
- › Section optionnelle (facultative)
- › Elle contient le code exécutable mis en place pour la gestion des erreurs
- › Lorsqu'une erreur intervient dans l'exécution, le programme est stoppé et le code erreur est transmis à cette section



# Section de gestion des exceptions

› Exemple :

```
DECLARE
```

```
    nb INTEGER := 20;
```

```
    excep EXCEPTION;
```

```
BEGIN
```

```
    IF nb > 10 THEN
```

```
        RAISE excep;
```

```
    END IF;
```

```
    EXCEPTION
```

```
        WHEN excep THEN (CODE_ERREUR, MESSAGE);
```

```
END;
```



# Déclaration des variables

- › **Nom\_variable type [NOT NULL] [ { := | DEFAULT } expression ] ;**
  - **Nom\_variable** : Commence par **une lettre** et peut contenir **des lettres, chiffres, \_, \$, ou #**.
  - **Type** : Simple (nombre, chaîne, etc.) ou Composé (tableau, enregistrement, etc.)
  - **[NOT NULL]** : interdit d'affecter une valeur nulle dans la variable et donc exige obligatoirement son initialisation à la déclaration.
  - **[ { := | DEFAULT } expression ]** : initialise une variable dès sa déclaration



# Déclaration des variables

› Les déclarations multiples sont **interdites** (une seule déclaration par ligne) : ~~i,j:=integer;~~

› Exemples :

- age integer;
- date\_Naissance date;
- date\_demain date DEFAULT SYSDATE+1;
- ok boolean:= true;



# Déclaration des variables

## Exemple :

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
    c varchar(15);
```

```
BEGIN
```

```
    c:= 'Hello World !' ;
```

```
    ...
```

```
END;
```

```
/
```

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
    c varchar(15) := 'Hello World !' ;
```

```
BEGIN
```

```
    ...
```

```
END;
```

```
/
```



# Déclaration des variables

› Les variables dans PL/SQL sont insensible à la case. En

PL/SQL, les expressions suivantes sont équivalentes :

- `NOM_VARIABLE NUMBER;`
- `Nom_Variable Number;`
- `nom_variable number;`



# Déclaration des constantes

- › Une constante est une variable dont l'initialisation est obligatoire et dont la valeur ne pourra pas être modifiée en cours d'exécution
- › Elle est déclarée avec le mot clé : `CONSTANT` qui doit précéder le type
- › Les identificateurs de variables et de constantes doivent être différents des noms de colonnes.



# Déclaration des constantes

› *Syntaxe:*

**Nom\_constant** **CONSTANT** nom\_de\_type { **:=** | **DEFAULT** }  
**expression ;**

› Exemple :

Pi **CONSTANT** **NUMBER** **:=** 3.14159 ;



# Déclaration des variables et constantes

- › Pour une meilleure lisibilité des programmes, il est recommandé de préfixer les identificateurs selon les conventions suivantes :

| Identifiant | Préfixe | Exemple             |
|-------------|---------|---------------------|
| Variables   | V_      | V_Salaire           |
| Constante   | V_      | V_PI                |
| Exception   | E_      | E_Stock_Insuffisant |
| Curseurs    | C_      | C_Etudiant          |



# Affectation de valeurs aux variables

- › L'affectation d'une expression dans une variable se fait généralement par l'opérateur " := "
- › L'affectation peut être soit à la déclaration d'une variable ou d'une constante, soit dans la section exécutable.

## › *Syntaxe*

Nom\_variable := expression;



# Affectation de valeurs aux variables

## *Exemple*

```
DECLARE
```

```
    V_compteur NUMBER := 15;
```

```
BEGIN
```

```
    V_Compteur := V_Compteur+1;
```

```
END;
```

```
/
```



# Les Instructions d'un bloc PL/SQL

- › Dans PL/SQL, on ne peut pas utiliser les instructions du LDD (ALTER, CREATE, DROP)
- › Les instructions SQL qu'on peut utiliser dans PL/SQL :
  - La partie LMD (insert, update, delete)
  - La partie LID (select)
  - La partie gestion de transactions (commit, rollback, ...)
  - Les fonctions (to\_char, to\_date, upper, substr, round, ...)



# Les Instructions d'un bloc PL/SQL

- › + Des instructions spécifiques à PL/SQL existent :
  - Définition de variables, constantes, expressions, affectations
  - Traitements conditionnels
  - Traitements répétitifs
  - Traitement des curseurs
  - Traitement des erreurs et d'exceptions



# Exemple

**DECLARE**

*-- déclaration de variables*

qty\_on\_hand NUMBER;

**BEGIN**

*-- ensemble d'instructions*

**SELECT** quantity **INTO** qty\_on\_hand **FROM** inventory  
    **WHERE** product = 'TENNIS RACKET';

**IF** qty\_on\_hand > 0 **THEN**

**UPDATE** inventory **SET** quantity = quantity - 1  
        **WHERE** product = 'TENNIS RACKET';

**INSERT INTO** purchase\_record  
        **VALUES** ('Tennis racket purchased', SYSDATE);

**END IF;**

**END;**

/



# Affichage dans PL/SQL

› Pour faire un affichage en PL/SQL, on utilise le paquetage

## ***DBMS\_OUTPUT***

- **PUT** (valeur); -- affichage : valeur chaîne de caractères/  
numérique/ date
- **PUT\_LINE** (valeur); -- même chose avec retour à la ligne
- **NEW\_LINE**; -- retour à la ligne

› Ces fonctions sont désactivées par défaut

› Activées par la commande : ***SET SERVEROUTPUT ON***



# Affichage dans PL/SQL

Exemple :

```
SET SERVEROUTPUT ON
```

```
BEGIN
```

```
    DBMS_OUTPUT.PUT_LINE('Hello World');
```

```
END;
```

```
/
```



# Lecture de valeur entrée par l'utilisateur

- › Lecture de la variable entrée par l'utilisateur hors du bloc :  
accept x number prompt 'Veuillez entrer un nombre: '
- › Utilisation de la variable dans le bloc :
  - *Section Déclaration* : Variable1 Integer;
  - *Section Exécutable* : Variable1 := &x;



# Lecture de valeur entrée par l'utilisateur

› Exemple :

```
SET SERVEROUTPUT ON
```

```
accept x number prompt 'Veuillez entrer un nombre: '
```

```
BEGIN
```

```
    DBMS_OUTPUT.PUT_LINE('La valeur entrée par l  
utilisateur est : ' || &x);
```

```
END;
```

```
/
```



# Exercice

Ecrire un bloc PL/SQL qui demande l'âge de l'utilisateur puis l'affiche par le message « Vous avez donc XX ans »



# Solution

```
accept x number prompt 'entrez_votre_age: '  
SET SERVEROUTPUT ON  
DECLARE  
    V_age NUMBER;  
BEGIN  
    V_age := &x;  
    DBMS_OUTPUT.PUT_LINE('Vous avez donc ' || V_age || '  
ans!');  
END;  
/
```



# Les Types PL/SQL

- › Les types PL/SQL se répartissent en trois classes :
  - Les types simples
  - Les types dérivés
  - Les types composés



# Les types PL/SQL Simples

- › PL/SQL supporte les types du langage SQL (CHAR, VARCHAR, NUMBER, DATE) et offre des types supplémentaires pour déclarer des variables et des constantes ainsi que des sous types.

| Catégorie       | Types                                                                                          |
|-----------------|------------------------------------------------------------------------------------------------|
| Numériques      | NUMBER[(e,d)], BINARY_INTEGER, BINARY_FLOAT, INTEGER (INT), SMALLINT, DECIMAL (DEC), REAL, ... |
| Alphanumériques | CHAR, VARCHAR, LONG, ...                                                                       |
| Booléen         | BOOLEAN                                                                                        |
| Date            | DATE, TIMESTAMP, ...                                                                           |



# Les Types dérivés (%TYPE)

## › L'attribut %TYPE

- référence à un type existant qui est soit le type d'une colonne de table, soit un type défini précédemment
- %TYPE se lit : de même type que

## › *Syntaxe:*

- **nom\_variable1 nom\_variable2%TYPE ;**



# Les Types dérivés (**%TYPE**)

## › **Exemple:**

-- déclaration de la variable V\_Dat2 de même type que V\_Dat1(DATE)

V\_Dat1 DATE;

V\_Dat2 V\_Dat1**%TYPE**;

-- déclaration de la variable V\_Nom de même type que la colonne

-- ENAME de la table EMP

V\_Nom EMP.ENAME**%TYPE**;



# Les Types dérivés (**%ROWTYPE**)

## › L'attribut **%ROWTYPE**

- permet de déclarer un enregistrement de même structure qu'une ligne d'une table de la BD

## › *Syntaxe:*

- **nom\_enregistrement nom\_table%ROWTYPE;**



# Les Types dérivés (**%ROWTYPE**)

## › ***Exemple:***

--variable de type enregistrement de même structure qu'une ligne de la table EMP

```
V_EMP EMP%ROWTYPE;
```



# Exemple

## › Schéma relationnel (les clés sont soulignées)

- immeuble (Adr, NbEtg, DateConstr, NomGerant)
- appart (Adr, Num, Type, Superficie, Etg, NbOccup)
- personne (Id, Nom, Age, CodeProf, Adr, Num)

## › Déclarations de variables

DECLARE

nomPersonne VARCHAR(10) := 'Guillaume';

monAdr CONSTANT VARCHAR(20) := '2 rue de la Paix';

**etage** appart.Etg%TYPE;

**monImmeuble** immeuble%ROWTYPE;

BEGIN

SELECT \* INTO monImmeuble FROM immeuble WHERE Adr=monAdr;

SELECT Etg INTO **etage** FROM appart

WHERE Adr=**monImmeuble.Adr** AND Num=10;

INSERT INTO personne VALUES (nomPersonne, 35, 'ING', **monImmeuble.Adr**, 10, **etage**);

END;



# Exercice

- › Soit les deux tables suivantes:
  - Employe(Matr, NomE, PrénomE, salaire, commision, DeptNo)
  - Departement (DeptNo, DeptNom)
  
- Créer un bloc PL/SQL permettant d'afficher le nom et le prénom de l'employé ayant comme matricule 900.
  
- Créer un bloc PL/SQL permettant d'afficher le nom, le salaire, la commission et le nom de département de l'employé avec le matricule 900.



# Solution

**DECLARE**

```
V_employe Employe%ROWTYPE;
```

**BEGIN**

```
SELECT * INTO V_employe FROM Employe
```

```
WHERE Matr = 900;
```

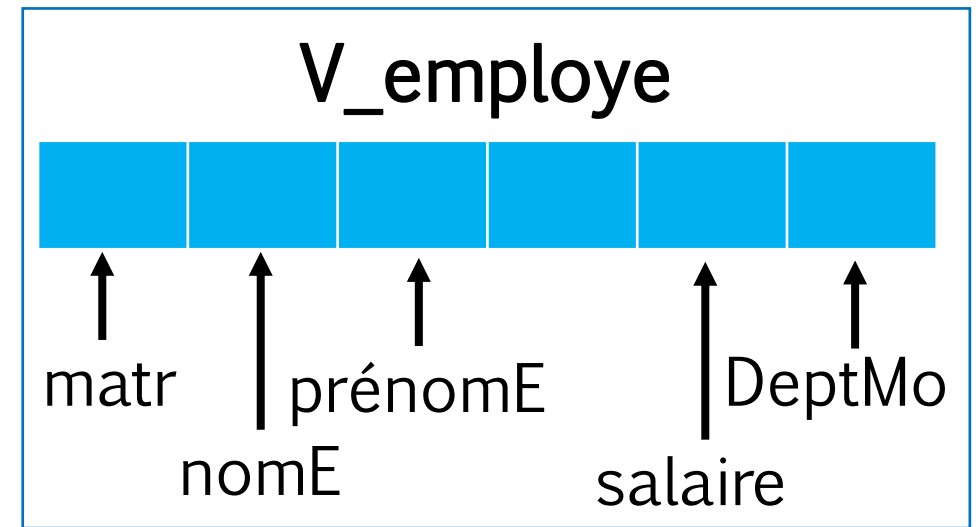
```
dbms_output.put_line ('L employé numéro 900 :');
```

```
dbms_output.put_line (' son nom est ' || V_employe.nomE);
```

```
dbms_output.put_line ('son prénom est ' || V_employe.prenomE);
```

**END;**

/





# Solution

## DECLARE

```
V_nomE Employe.NomE%TYPE;  
V_salaire Employe.salaire%TYPE;  
V_commission Employe.commission%TYPE;  
V_nomD Departement.DeptNom%TYPE;
```

## BEGIN

```
SELECT NomE, salaire, commission, DeptNom  
INTO V_nomE, V_salaire, V_commission, V_nomD  
FROM Employe, departement  
WHERE Matr = 900 AND Employe.DeptNo=Departement.DeptNo;  
dbms_output.put_line ('Nom ' || V_nomE || ' Salaire ' || V_salaire ||  
' Commission ' || V_commission || ' Nom de département ' || V_nomD);
```

END;

/



# Type PL/SQL Composé : Enregistrement

› La déclaration se fait en deux étapes :

a/ Déclaration du type enregistrement

*Syntaxe*

```
TYPE nom-type IS RECORD (  
    champ1 type1,  
    champ2 type2, ...);
```

b/ Déclaration d'une variable du type enregistrement

*Syntaxe*

```
nom_variable nom_type;
```



# Type PL/SQL Composé : Enregistrement

› *Exemple*

*DECLARE*

```
TYPE emp2 IS RECORD (  
    matr integer,  
    nom varchar(30));  
employee emp2;
```

*BEGIN*

```
employee.matr := 500;  
employee.nom := 'Mohamed';
```

*END;*

/



# Type PL/SQL Composé : Enregistrement

## › Type enregistrement par référence à une table :

PL/SQL permet de déclarer un enregistrement de même structure qu'une table de la BD en utilisant le type générique **%ROWTYPE** :

### › *Syntaxe*

Nom\_enregistrement nom\_table**%ROWTYPE**;

### › *Exemple*

**employe emp%ROWTYPE;**



# Type PL/SQL Composé : Table

› Une table PL/SQL est un vecteur d'éléments d'un même type scalaire. La déclaration se fait en deux étapes :

a/ Déclaration du type des éléments de la table

*Syntaxe :*

```
TYPE nom_type IS TABLE OF type_scalaire [NOT NULL]  
INDEX BY BINARY_INTEGER;
```

b/ Déclaration de la variable du type table

*Syntaxe :* nom\_table nom\_type;



# Type PL/SQL Composé : Table

*Exemple :*

**DECLARE**

**TYPE** TAB\_NOTE **IS TABLE OF** NUMBER

**INDEX BY** BINARY\_INTEGER; --déclaration d'un type

TNOTE TAB\_NOTE ; --déclaration d'une variable

**BEGIN**

TNOTE(5) := 13;

TNOTE(6) := TNOTE(5) + 2;

**END;**

/



# Type PL/SQL Composé : Table

*Exemple :*

**DECLARE**

```
TYPE tableau_Pilotes IS TABLE OF pilote%ROWTYPE  
INDEX BY BINARY_INTEGER;  
tabPilotes tableau_Pilotes;
```

**BEGIN**

```
...  
tabPilotes(4).Age := 37;  
tabPilotes(4).Salaire := 42000;  
tabPilotes.DELETE(5);
```

...

**END;**



# Type PL/SQL Composé : Table

› Les attributs d'une variable de type table :

| Attribut      | Explication                                                                                                                                                          | Exemple                                                                |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------|
| <b>EXISTS</b> | EXISTS(n) renvoie TRUE si le nième élément de la variable de type table existe. Sinon, renvoie FALSE                                                                 | IF sal_tab.EXISTS(i) THEN sal_tab(i) := sal_tab(i) + 500; END IF;      |
| <b>COUNT</b>  | renvoie le nombre d'éléments contenus dans la variable de type table                                                                                                 | IF ename_tab.COUNT = 50 THEN ...<br>END IF;                            |
| <b>DELETE</b> | DELETE supprime tous les éléments de la variable de type table.<br>DELETE(n) supprime le nième élément<br>DELETE(m, n) supprime tous les éléments de la plage m .. n | ename_tab.DELETE(3);<br>ename_tab.DELETE(20, 30);<br>ename_tab.DELETE; |



# Type PL/SQL Composé : Table

› Les attributs d'une variable de type table :

| Attribut                    | Explication                                                                                                                            | Exemple                                                                                                              |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| <b>FIRST</b><br><b>LAST</b> | FIRST et LAST renvoient le premier et le dernier (le plus petit et le plus grand) index de la variable de type table                   | FOR i IN emp_tab.FIRST ..<br>emp_tab.LAST LOOP ... END LOOP;                                                         |
| <b>PRIOR</b><br><b>NEXT</b> | PRIOR(n) renvoie le numéro d'index précédant l'index n dans la variable.<br>NEXT(n) renvoie le numéro d'index qui succède à l'index n. | i := any_tab.FIRST;<br>WHILE i IS NOT NULL LOOP<br>-- utiliser any_tab(i)<br>i := any_tab.NEXT(i);<br>END LOOP; END; |



# Structures de contrôle conditionnelles

## › Instruction Conditionnelle: IF

Cette instruction permet de faire des tests conditionnels

### › Syntaxe :

**IF** *condition1* **THEN** *traitement* ;

**[ELSEIF** *condition2* **THEN** *traitement* ; ]

ELSE et ELSEIF : optionnelles

**[ELSEIF** *condition3* **THEN** *traitement* ; ]

ELSEIF: plusieurs

**[ELSE** *traitement* ; ]

ELSE: une seule

**END IF;**



# Exemple

```
SET SERVEROUTPUT ON
accept x prompt 'entrer le premier nombre'
accept y prompt 'entrer le deuxième nombre'
DECLARE
    V_Num1 integer := &x;
    V_Num2 integer := &y;
BEGIN
    IF (V_Num1 > V_Num2) THEN
        DBMS_OUTPUT.PUT_LINE('le maximum est ' || V_Num1);
    ELSIF (V_Num1 < V_Num2) THEN
        DBMS_OUTPUT.PUT_LINE('le maximum est ' || V_Num2);
    ELSE
        DBMS_OUTPUT.PUT_LINE('les deux nombres sont égaux');
    END IF;
End;
/
```



# Structures de contrôle conditionnelles

- › Structure Case : 2 syntaxes sont possibles :

**CASE** var

**WHEN** contenu-var **THEN** instructions1;

**WHEN** contenu-var **THEN** instructions2;

...

**WHEN** contenu-var **THEN** instructionsN;

[**ELSE** instructionsN+1;]

**END CASE**;

**CASE**

**WHEN** condition1 **THEN** instructions1;

**WHEN** condition2 **THEN** instructions2;

...

**WHEN** conditionN **THEN** instructionsN;

[**ELSE** instructionsN+1;]

**END CASE**;



# Exemple 1 :

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
    V_Num integer := 0;
```

```
BEGIN
```

```
    LOOP
```

```
        V_Num := V_Num + 1;
```

```
        CASE V_Num
```

```
            WHEN 1 THEN dbms_output.put_line('1');
```

```
            WHEN 2 THEN dbms_output.put_line('2');
```

```
            WHEN 3 THEN dbms_output.put_line('3');
```

```
            ELSE EXIT;
```

```
        END CASE;
```

```
    End LOOP;
```

```
End;
```

```
/
```



## Exemple 2 :

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
    V_Num integer := 0;
```

```
BEGIN
```

```
    LOOP
```

```
        V_Num := V_Num + 1;
```

```
        CASE
```

```
            WHEN V_Num between 1 and 3
```

```
                THEN dbms_output.put_line(V_Num || ' -> 1-3' );
```

```
            WHEN V_Num < 5
```

```
                THEN dbms_output.put_line(V_Num || ' < 5' );
```

```
            ELSE
```

```
                dbms_output.put_line(V_Num || ' >= 5' );
```

```
        END CASE ;
```

```
        EXIT WHEN V_Num = 5 ;
```

```
    END LOOP;
```

```
END;
```

```
/
```



# Structures de contrôle de répétition

- › **La boucle de base : LOOP**

- › **Syntaxe :**

***LOOP***

*Séquence;*

***END LOOP;***

- › Cette syntaxe met en place une boucle simple ou aucune condition de sortie n'est indiquée,
- › Il faut donc une instruction ***EXIT*** pour sortir de ce type de boucle: **EXIT [ WHEN condition].**



# Exemple

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
    Compteur integer;
```

```
BEGIN
```

```
    Compteur := 0;
```

```
    LOOP
```

```
        Compteur := Compteur + 1;
```

```
        dbms_output.put_line(Compteur);
```

```
        EXIT WHEN Compteur = 10;
```

```
    END LOOP;
```

```
END;
```

```
/
```



# Structures de contrôle de répétition

## › La boucle WHILE

### › Syntaxe :

***WHILE** condition*

***LOOP***

*Séquence;*

***END LOOP;***

- › Cette syntaxe permet de mettre en place une boucle dont la condition de test est évaluée au début.
- › Si condition donne le résultat FALSE, les instructions suivantes jusqu'au mot clé **END LOOP;** ne seront pas exécutées



# Exemple

**SET SERVEROUTPUT ON**

**Declare**

I integer := 0;

**Begin**

**While** I < 3

**Loop**

I := I + 1;

dbms\_output.put\_line(I);

**End loop;**

**End;**

/



# Structures de contrôle de répétition

## › La boucle FOR

Cette syntaxe permet de mettre en place une boucle dont le nombre d'itérations est fixé dès l'entrée

### › Syntaxe :

**FOR** *indice* **IN [REVERSE]** *b\_inf .. b\_sup*

**LOOP**

*Séquence;*

**END LOOP;**

### › Où ***Indice***:

- variable entière déclarée implicitement par la boucle
- locale
- Référencé dans la boucle seulement
- S'incrémente automatiquement de 1 à la fin de chaque itération



# Exemple

1-**SET SERVEROUTPUT ON**  
**BEGIN**

**FOR** i **IN** 1..3  
**LOOP**

dbms\_output.put\_line(i);  
**END LOOP;**

**END;**

/

2-**SET SERVEROUTPUT ON**  
**BEGIN**

**FOR** i **IN REVERSE** 1..3  
**LOOP**

dbms\_output.put\_line(i);  
**END LOOP;**

**END;**

/



# Exercice

Ecrivez un programme plaçant la valeur 10 dans une variable a, puis affichant la factoriel de a.

$$10! = 10 * 9 * 8 * \dots * 1$$



# Solution

SET SERVEROUTPUT ON

DECLARE

    a NUMBER;

    res NUMBER;

BEGIN

    a := 10;

    res := 1;

    WHILE a > 0 LOOP

        res := res \* a;

        a := a - 1;

    END LOOP;

    DBMS\_OUTPUT.PUT\_LINE(10 || '! = ' || res);

END;

/



# Imbrication de blocs PL/SQL

- › Les blocs PL/SQL peuvent être imbriqués les uns dans les autres:

```
DECLARE  
#  
BEGIN  
#
```

```
DECLARE  
##..  
BEGIN  
##
```

```
BEGIN  
###  
END ;
```

```
##..  
END ;
```

```
#..  
END ;
```