

Cours de Bases de données

PL/SQL (Part 1)

Master : Traitement intelligent des systèmes

Préparé par: Mme. Khawla Elansari

Année Universitaire: 2021/ 2022

1. SQL vs. PL/SQL

SQL vs. PL/SQL

- **SQL** : langage de requête structuré qui ajoute, supprime, modifie ou manipule les données d'une base de données.

- **PL/SQL** : langage procédural qui est une extension de SQL et qui contient des instructions SQL dans sa syntaxe.

SQL	PL/SQL
C'est un Langage de requête structurée de base de données.	C'est un langage de programmation de base de données utilisant SQL.
Les variables de données ne sont pas autorisées	Les variables de données sont autorisées.
Aucune structure de contrôle prise en charge.	Les structures de contrôle sont prise en charges e.g la boucle for, while, etc.
La requête effectue une seule opération.	Le bloc PL/SQL exécute le groupe d'opérations en tant que bloc unique.
SQL est un langage déclaratif.	PL/SQL est un langage procédural.
SQL peut être intégré à PL/SQL.	PL/SQL peut être incorporé dans SQL.
Il est directement en interaction avec le serveur de base de données	N'interagit pas avec le serveur de base de données.
C'est un langage orienté données.	C'est un langage orienté application.
Il est utilisé pour écrire des requêtes, des instructions DDL et DML.	Il s'agit de blocs de programme, de fonctions, de déclencheurs de procédures et de packages.

PL / SQL

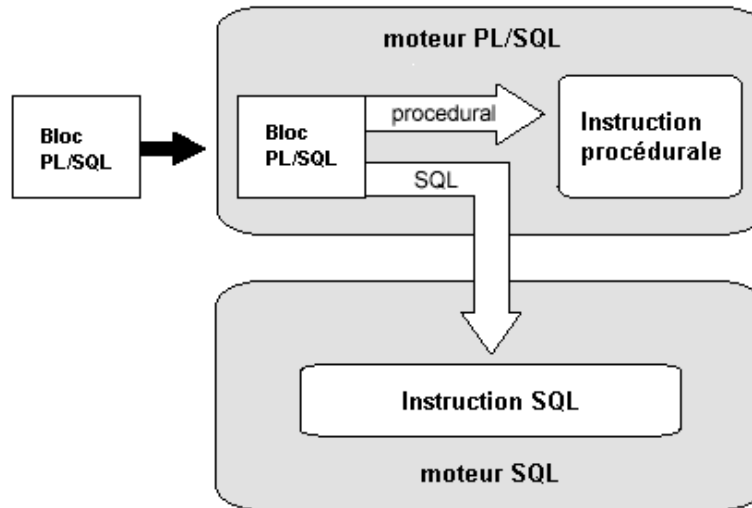
- PL/SQL est un langage procédural qui interagit avec la base de données Oracle.
- Il peut accéder à toutes les capacités de SQL.
- De plus, il permet aux développeurs d'utiliser les fonctionnalités généralement associées aux langages de programmation. Par exemple, PL/SQL prend en charge les **variables**, les **tableaux**, les **exceptions** et le **flux d'instructions de contrôle**. Il permet aux développeurs de structurer leur code en sous-programmes, qui incluent des **packages**, des **déclencheurs** (Triggers), des **fonctions** et des **procédures**.

PL/SQL avantages

- Faire cohabiter des structures de contrôle (si, pour et tant que) avec des instructions SQL (principalement SELECT, INSERT, UPDATE et DELETE).
- La modularité (un bloc d'instruction peut être composé d'un autre, etc.) : un bloc peut être nommé pour devenir une procédure ou une fonction cataloguée, donc réutilisable. Une procédure, ou fonction, cataloguée peut être incluse dans un paquetage (package) pour mieux contrôler et réutiliser ces composants logiciels.
- La portabilité : un programme PL/SQL est indépendant du système d'exploitation qui héberge le serveur Oracle. En changeant de système, les applicatifs n'ont pas à être modifiés.
- L'intégration avec les données des tables : on retrouvera avec PL/SQL tous les types de données et instructions disponibles sous SQL, et des mécanismes pour parcourir des résultats de requêtes (curseurs), pour traiter des erreurs (exceptions), pour manipuler des données complexes (paquetages DBMS_xxx) et pour programmer des transactions (COMMIT, ROLLBACK, SAVEPOINT).

Structure

- PL/SQL est un langage structuré en blocs, constitués d'un ensemble d'instructions.
- Un bloc PL/SQL est intégralement envoyé au moteur PL/SQL, qui traite chaque instruction PL/SQL et sous-traite les instructions purement SQL au moteur SQL, afin de réduire le trafic réseau.



2. PL/SQL - Notions de base

Blocs Anonymes

- Un bloc anonyme est un bloc de code qui possède sa propre structure DECLARE/BEGIN/END. Les blocs anonymes peuvent être autonomes (comme illustré ici) ou être intégrés à n'importe quel autre programme PL/SQL.

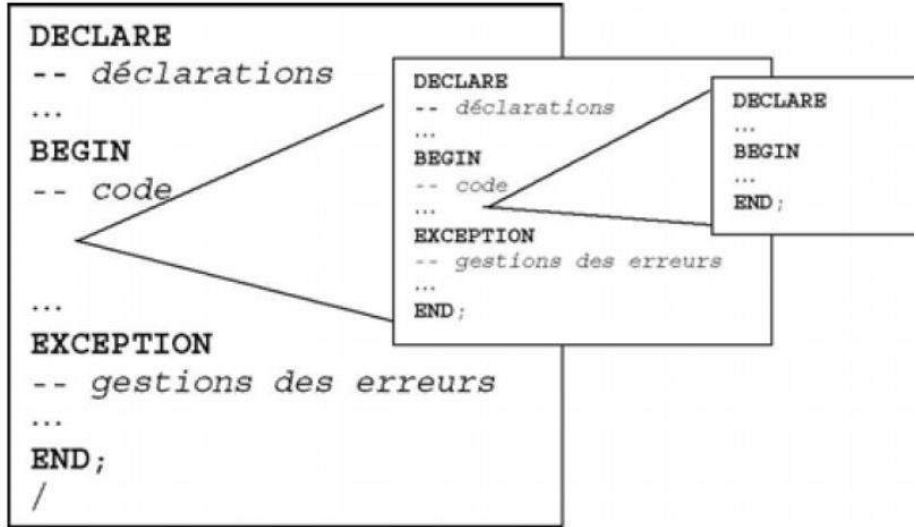
<pre>-- Simple code ----- set serveroutput on; Declare -- The declaration section defines all variables, cursors, subprograms, and other elements to be used in the code. msg varchar2(40) := 'Cours PL/SQL - Master IPS'; BEGIN -- The procedural section contains the main body of the routine. dbms_output.put(msg); dbms_output.put_line(msg); END;</pre>	
Script Output x	
Task completed in 0.031 seconds	
Cours PL/SQL - Master IPS	Cours PL/SQL - Master IPS
PL/SQL procedure successfully completed.	

Blocs Anonymes

1. **Déclaration:** Cette section commence par le mot-clé DECLARE. Il n'est pas considéré comme obligatoire et comporte des variables, des sous-programmes, etc.
2. **Commandes exécutables:** Cette section commence par les mots-clés BEGIN et END respectivement. Il est considéré comme obligatoire et contient des instructions PL / SQL. Il se compose d'au moins une ligne de code exécutable.
3. **Gestion des exceptions:** Cette section commence par le mot-clé EXCEPTION. Elle comprend les types d'exceptions que le code gérera.
4. **BEGIN:** C'est le mot-clé utilisé pour pointer vers le bloc d'exécution. Il est requis dans un code PL / SQL où la logique métier réelle est décrite.
5. **END:** C'est le mot-clé utilisé pour déterminer la fin du bloc de code

Blocs Anonymes

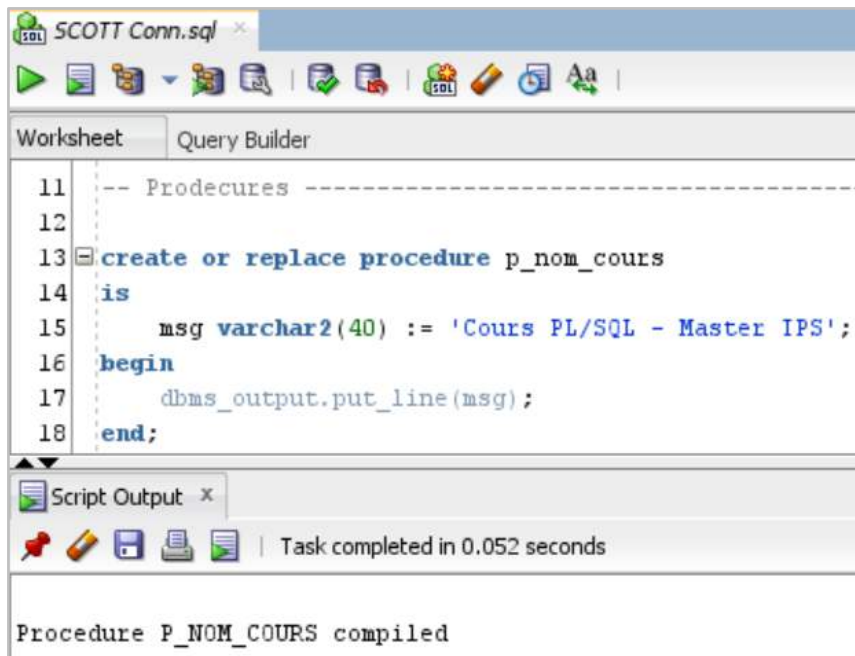
- Chaque bloc PL/SQL peut être constitué de 3 sections :
 - Une section facultative de **déclaration et initialisation** de types, variables, Triggers et constantes
 - Une section obligatoire contenant les **instructions d'exécution**
 - Une section facultative de **gestion des erreurs**



Chaque ligne complète du code PL/SQL doit se terminer par un point-virgule (;)

Procédures PL/SQL

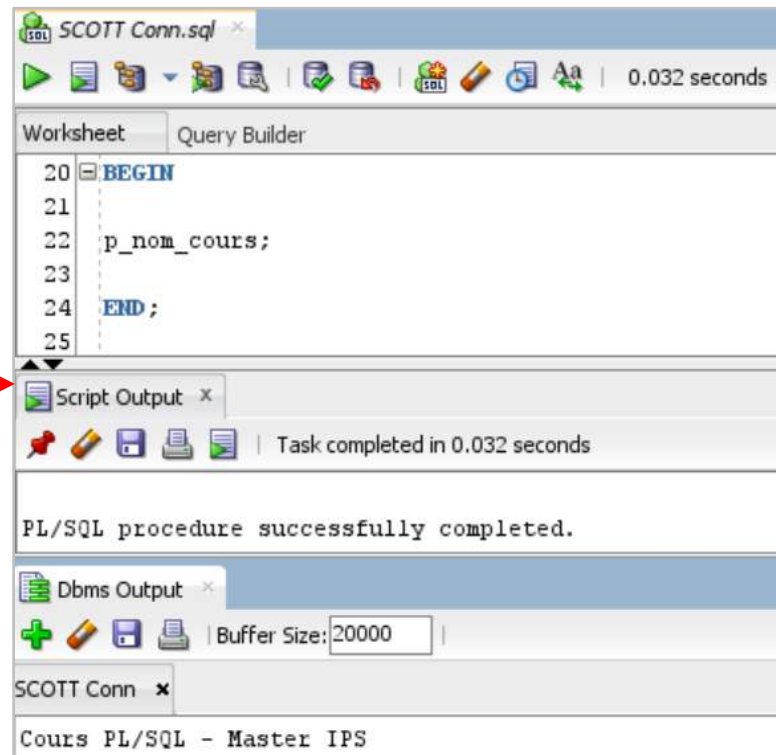
- Vous pouvez stocker du code PL/SQL dans la base de données.



The screenshot shows the SQL Developer interface with a worksheet titled 'SCOTT Conn.sql'. The code in the worksheet is as follows:

```
11  -- Prodecures -----
12
13  create or replace procedure p_nom_cours
14  is
15      msg varchar2(40) := 'Cours PL/SQL - Master IPS';
16  begin
17      dbms_output.put_line(msg);
18  end;
```

Below the code editor, the 'Script Output' window shows the message: 'Procedure P_NOM_COURS compiled'. A red arrow points from the 'msg' variable in the code to the 'Script Output' window in the second screenshot.



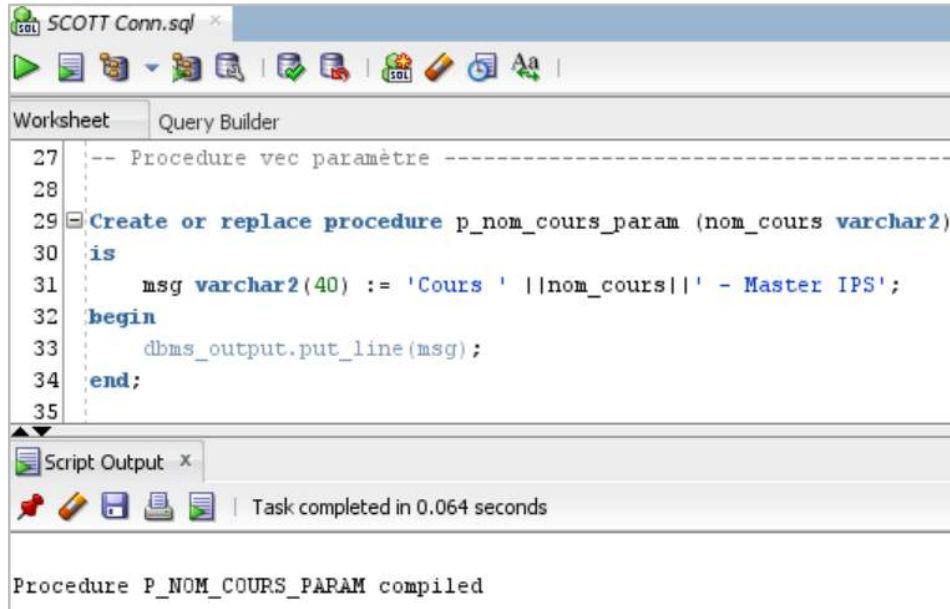
The screenshot shows the SQL Developer interface with a worksheet titled 'SCOTT Conn.sql'. The code in the worksheet is as follows:

```
20  BEGIN
21
22      p_nom_cours;
23
24  END;
```

Below the code editor, the 'Script Output' window shows the message: 'PL/SQL procedure successfully completed.' Below that, the 'Dbms Output' window shows the message: 'Cours PL/SQL - Master IPS'.

Procédures PL/SQL - Paramètres

- PL/SQL permet de passer des paramètres aux procédures Stockées

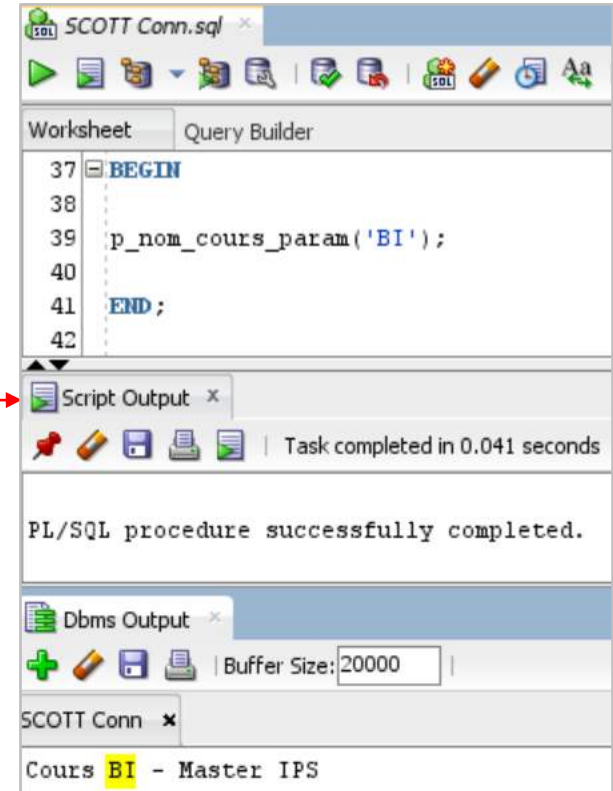


The screenshot shows the SQL Developer interface with a worksheet titled 'SCOTT Conn.sql'. The 'Worksheet' tab is active, displaying a PL/SQL script. The script defines a procedure named 'p_nom_cours_param' that takes a 'nom_cours' parameter of type 'varchar2'. Inside the procedure, a message is constructed by concatenating 'Cours ', the parameter value, and ' - Master IPS'. The procedure is then executed, and the 'Script Output' window shows the message 'Procedure P_NOM_COURS_PARAM compiled'.

```
27 -- Procedure vec paramètre -----
28
29 Create or replace procedure p_nom_cours_param (nom_cours varchar2)
30 is
31     msg varchar2(40) := 'Cours ' || nom_cours || ' - Master IPS';
32 begin
33     dbms_output.put_line(msg);
34 end;
35
```

Script Output x
Task completed in 0.064 seconds

Procedure P_NOM_COURS_PARAM compiled



The screenshot shows the SQL Developer interface with a worksheet titled 'SCOTT Conn.sql'. The 'Query Builder' tab is active, displaying the same PL/SQL script. The 'Script Output' window shows the message 'PL/SQL procedure successfully completed.' The 'Dbms Output' window shows the output of the procedure: 'Cours BI - Master IPS'.

```
37 BEGIN
38
39     p_nom_cours_param('BI');
40
41 END;
42
```

Script Output x
Task completed in 0.041 seconds

PL/SQL procedure successfully completed.

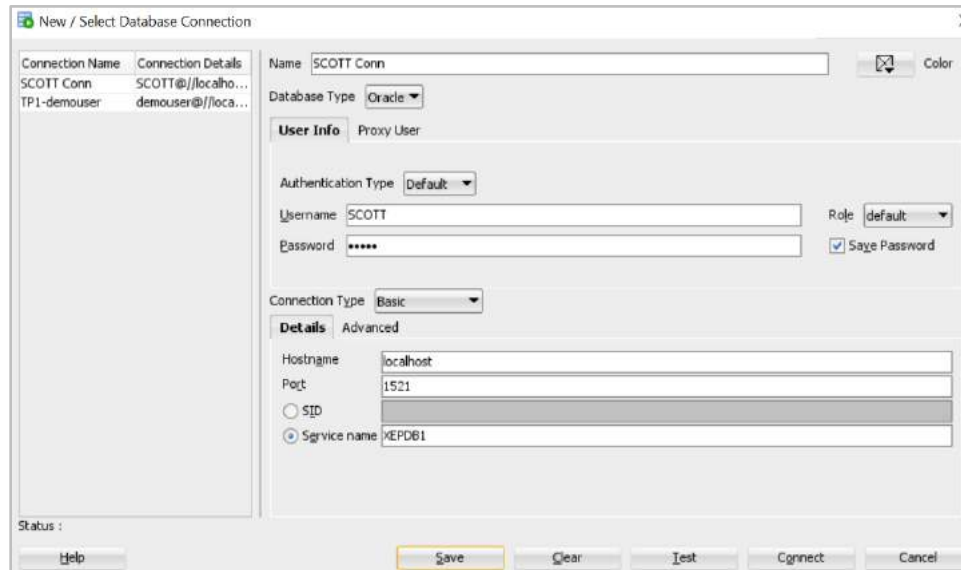
Dbms Output x
Buffer Size: 20000

SCOTT Conn x

Cours BI - Master IPS

Schéma SCOTT

- Aller sur : C:\app\LENOVO\product\21c\dbhomeXE\rdbms\admin
- Exécuter le script **scott.sql**
- Créer une nouvelle connexion au Schéma **SCOTT / TIGER**



3. Principes fondamentaux du PL/SQL

Identifiants PL/SQL

- Les identifiants sont des noms d'éléments et d'unités de programme PL/SQL. Ces éléments et unités peuvent être de différentes sortes : constantes, variables, exceptions, curseurs, variables de curseur, sous-programmes et packages.
 - Un identifiant ne peut dépasser 30 caractères.
 - Un identifiant se compose d'une lettre éventuellement suivie de plusieurs lettres, chiffres, signes dollar, traits de soulignement (_) et signes numériques
 - Par défaut, les identifiants ne sont pas sensibles à la casse, donc v_index_nr et V_Index_NR sont les mêmes.
 - Les identifiants doivent être différents des mots réservés

Exemples:

- v_index_nr (valide)
- v\$index_nr (valide)
- v index_nr (invalide : contient un espace)
- 5_index_nr (invalide : commence par un nombre)
- v-index_nr (invalide : contient -, un trait d'union)

Variables PL / SQL

Déclaration

- En PL/SQL, les variables doivent être incluses dans le bloc de déclaration avant de pouvoir être utilisées. Il existe plusieurs manières de déclarer une variable. La méthode la plus courante consiste à utiliser une déclaration directe:

```
declare
    variable_name [constant] DATATYPE
    [DEFAULT value | DEFAULT NULL];
begin
```

- Le mot-clé **constant** signifie que la valeur de la variable ne peut pas être modifiée dans le corps du programme. Si vous déclarez une variable en tant que constante, vous devez lui affecter une valeur par défaut à l'aide de la clause facultative de valeur DEFAULT.

- Ex:

```
declare

    v_sal_nr NUMBER;
    v_name_tx VARCHAR2(10) DEFAULT 'KING';
    v_start_dt DATE := SYSDATE; -- same as DEFAULT SYSDATE
begin
```


Variables PL / SQL

Déclaration par référence

- Vous pouvez également déclarer une variable par référence (%TYPE pour les variables simples et %ROWTYPE pour les variables pouvant stocker toute la ligne)

```
declare

variable_name table.column%TYPE; -- La variable a le meme type que la colonne
variable_name2 variable_name%TYPE; -- la variable a le meme type que la variable_name
variable_row table%ROWTYPE; /* La variable va stocker les valeurs de ttes les
colonnes d'un enregistrement de la table */
begin
```

- Ex:

```
declare
v_empno1 emp.empNo%TYPE;
v_empno2 v_empno1%TYPE;
v_dept_rec dept%ROWTYPE;
begin
...
```



- La définition de types de données par référence est extrêmement utile et réduit la maintenance du programme requise, car la modification du type de données d'une colonne dans la base de données ne nécessite pas de rechercher tous les emplacements où cette colonne est référencée.

=> Les modifications sont héritées automatiquement et à la volée.

N.B: Les déclarations multiples prises en charge dans certaines langues ne sont pas autorisées en PL/SQL

```
declare
    v1_nr, v2_nr NUMBER; -- INVALID
    -- VALID
    v1_nr NUMBER;
    v2_nr NUMBER;
begin
    ...
```

PL/SQL peut gérer les types de records suivants :

- Basés sur une table/vue
- Définis par l'utilisateur.

Record basé sur une table | vue

- L'attribut **%ROWTYPE** permet de déclarer un enregistrement PL/SQL qui représente une ligne dans une table de base de données, sans lister toutes les colonnes.
- Le code continue de fonctionner même après l'ajout de colonnes à la table.
- Si vous souhaitez représenter un sous-ensemble de colonnes dans une table ou des colonnes de différentes tables, vous pouvez définir une vue pour sélectionner les colonnes souhaitées et effectuer les jointures nécessaires, puis appliquer **%ROWTYPE** à la vue ou au curseur.
- Syntaxe : `nom_record nom_cols%ROWTYPE;` -- Nom de la table ou la vue

Variables PL / SQL

Records

Table Emp:

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	17-DEC-80	800	(null)	20
2	7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
3	7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30

```
DECLARE
    empl Emp%ROWTYPE;
    -- record basé sur la structure d'une ligne de la table Emp;
BEGIN
    SELECT * INTO empl FROM Emp WHERE EMPNO=7499;

    -- afficher les données du record
    dbms_output.put_line('Nom : ' || empl.ENAME);
    dbms_output.put_line('Salaire : ' || empl.SAL);
    dbms_output.put_line('Age : ' || empl.HIREDATE);
END;
```

Script Output x

Task completed in 0.034 seconds

Nom : ALLEN
Salaire : 1600
Age : 20-FEB-81

Record défini par l'utilisateur

- Le type RECORD est défini comme suit :

```
TYPE nom_record IS RECORD (  
    champ1  typedonnees1  [NOT NULL]  [:= expression_par_defaut],  
    champ2  typedonnees2  [NOT NULL]  [:= expression_par_defaut],  
    ...  
    champN  typedonneesN  [NOT NULL]  [:= expression_par_defaut]);  
);
```

- Pour référencer des champs individuels dans un record, vous utilisez la notation nom_rec.nom_champ. Par exemple, pour référencer le champ Nom dans le record rec_emp, vous utiliseriez rec_emp.Nom

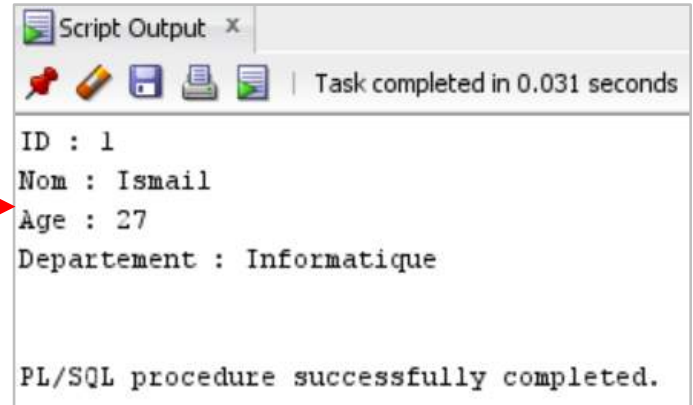
Variables PL / SQL

Records

```
DECLARE
    -- autres déclarations
    TYPE Personne IS RECORD (
        Id NUMBER NOT NULL := 0,
        Nom VARCHAR(30),
        Age NUMBER,
        Departement VARCHAR(30)
    );

    p1 Personne;

BEGIN
    p1.Id := 1;
    p1.Nom := 'Ismail';
    p1.Age := 27;
    p1.Departement := 'Informatique';
    dbms_output.put_line('ID : ' || p1.Id);
    dbms_output.put_line('Nom : ' || p1.Nom);
    dbms_output.put_line('Age : ' || p1.Age);
    dbms_output.put_line('Departement : ' || p1.Departement);
END;
```



Script Output x

Task completed in 0.031 seconds

ID : 1
Nom : Ismail
Age : 27
Departement : Informatique

PL/SQL procedure successfully completed.

Variables PL / SQL

vArray

- Un tableau PL/SQL:
 - est une collection ordonnée d'élément du même type
 - accessible uniquement en PL/SQL
 - stockés en mémoire, ils peuvent grandir dynamiquement

`TYPE nom_tableau IS VARRAY(max_elem) of type_element [NOT NULL];` -- nom_tableau : nom du tableau

- Pour initialiser une variable VARRAY à une collection vide (zéro élément), vous utilisez la syntaxe suivante

```
varray_nom nom_type := nom_type();
```

- Si vous souhaitez spécifier des éléments pour la variable VARRAY lors de son initialisation, vous pouvez utiliser cette syntaxe:

```
varray_nom nom_type := nom_type(element1, element2, ...);
```

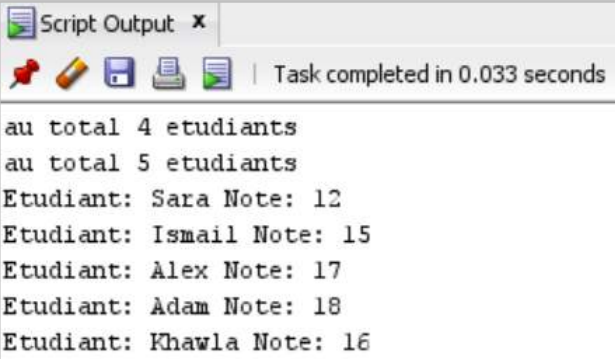
N.B: Un varray est automatiquement NULL lorsqu'il est déclaré et doit être initialisé avant que ses éléments puissent être référencés.

Variables PL / SQL

vArray

```
DECLARE
    Type tabnotes IS VARRAY(10) OF INTEGER;
    Type tabnoms IS VARRAY(10) OF VARCHAR2(20);
    nl tabnotes; -- Declaration
    cl tabnoms; -- Declaration
    total integer;

BEGIN
    nl:= tabnotes();
    nl.extend(4);
    --nl:= tabnotes(12,15,17,18); -- Initialisation
    cl:= tabnoms('Sara', 'Ismail','Alex','Adam');-- Initialisation
    total := cl.count;
    dbms_output.put_line('au total ' || total || ' etudiants');
    cl.extend; --
    nl.extend;
    cl(total +1):= 'Khawla';
    nl(1) := 12; nl(2) := 15; nl(3) := 17; nl(4) := 18;
    nl(total +1) := 16;
    total := cl.count;
    dbms_output.put_line('au total ' || total || ' etudiants');
    FOR i in 1 .. total LOOP
        dbms_output.put_line('Etudiant: ' || cl(i) || ' Note: ' || nl(i));
    END LOOP;
END;
```



Script Output x


Task completed in 0.033 seconds

```
au total 4 etudiants
au total 5 etudiants
Etudiant: Sara Note: 12
Etudiant: Ismail Note: 15
Etudiant: Alex Note: 17
Etudiant: Adam Note: 18
Etudiant: Khawla Note: 16
```


Variables PL / SQL

vArray de records

```
set serveroutput on;
declare
    type tablemul is record ( par8 number, par9 number);
    type tableentiers is varray(50) of tablemul;
    ti tableentiers;
    i number;
begin
    ti := tableentiers();
    Ti.extend(10);
    for i in 1..10 loop
        ti(i).par9 := i*9 ;
        ti(i).par8:= i*8;
        dbms_output.put_line (i||'*8='||ti(i).par8||' '||i||'*9='||ti(i).par9 );
    end loop;
end;
```

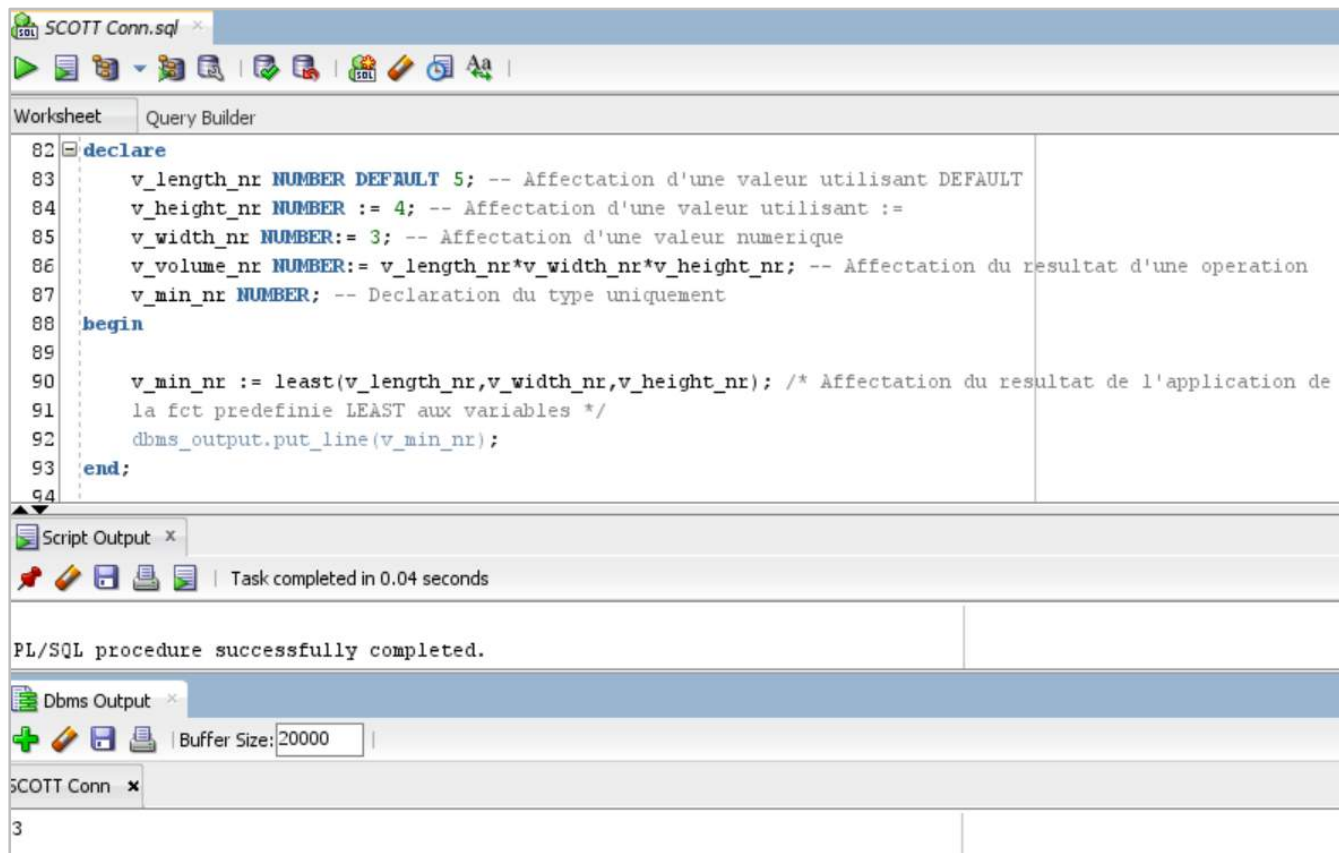


Script Output x

1*8=8 1*9=9
2*8=16 2*9=18
3*8=24 3*9=27
4*8=32 4*9=36
5*8=40 5*9=45
6*8=48 6*9=54
7*8=56 7*9=63
8*8=64 8*9=72
9*8=72 9*9=81
10*8=80 10*9=90

Variables PL / SQL

Affectation d'une valeur à une variable



The screenshot displays the SQL Developer interface with a PL/SQL script in the 'Query Builder' tab. The script defines several variables and performs an assignment. Below the script, the 'Script Output' and 'Dbms Output' tabs show the successful execution of the procedure.

```
82 declare
83     v_length_nr NUMBER DEFAULT 5; -- Affectation d'une valeur utilisant DEFAULT
84     v_height_nr  NUMBER := 4; -- Affectation d'une valeur utilisant :=
85     v_width_nr   NUMBER:= 3; -- Affectation d'une valeur numerique
86     v_volume_nr  NUMBER:= v_length_nr*v_width_nr*v_height_nr; -- Affectation du resultat d'une operation
87     v_min_nr     NUMBER; -- Declaration du type uniquement
88 begin
89
90     v_min_nr := least(v_length_nr,v_width_nr,v_height_nr); /* Affectation du resultat de l'application de
91     la fct predefinie LEAST aux variables */
92     dbms_output.put_line(v_min_nr);
93 end;
94
```

Script Output
Task completed in 0.04 seconds

PL/SQL procedure successfully completed.

Dbms Output
Buffer Size: 20000

SCOTT Conn x

3

Variables PL / SQL

Affectation d'une valeur à une variable

```
declare
  v_char_tx CHAR(1):='H';
  v_text1_tx VARCHAR2(10) := 'Hello';
  v_text1_tx VARCHAR2(50) := 'It's Misha's text.';
  v_text2_tx VARCHAR2(50) := 'It's Misha's text.'; /* You can use other delimiters (not only !, but </>, [/], {/}, and (/))
  to declare the start and end of a quoted string */
  v_int1_nr BINARY_INTEGER :=5; -- integer
  v_int2_nr BINARY_INTEGER :=-5; -- integer
  v_int4_nr BINARY_INTEGER :=+5; -- integer
  v_real1_nr NUMBER :=5.0;
  v_real2_nr NUMBER :=5.;
  v_real3_nr NUMBER :=-7.123;
  v_real5_nr NUMBER :=.5;
begin
  ...
```

Structures de contrôle - Alternatives

if - then - else

```
IF condition1 THEN
    instructions;
ELSIF condition2 THEN
    instructions;
    .
    .
    .

ELSE
    instructions;
END IF;
```

case – when - else

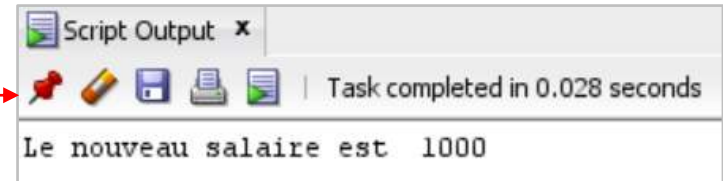
```
CASE variable
    WHEN expr1 THEN instructions1;
    WHEN expr2 THEN instructions2;
    ...
    WHEN exprN THEN instructionsN;
    [ELSE instructionsN+1;]
END CASE [étiquette];
```

```
CASE
    WHEN condition1 THEN instructions1;
    WHEN condition2 THEN instructions2;
    ...
    WHEN conditionN THEN instructionsN;
    [ELSE instructionsN+1;]
END CASE [étiquette];
```

Structures de contrôle - Alternatives

if - then - else

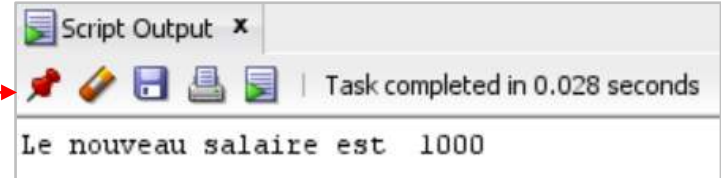
```
-- Augmenter les salaires de 25% pour les employes dont le salaire est inferieur ou egale à 800
Declare
    v_employe EMP%ROWTYPE;
Begin
    Select * into v_employe From Emp Where EMPNO = 7369;    /*doit retourner 1 tuple seulement */
    IF v_employe.sal <= 800
    THEN Begin
        v_employe.sal := v_employe.sal * 1.25;
        DBMS_OUTPUT.put_line ('Le nouveau salaire est' || ' ' || v_employe.sal);
    End;
    ELSE
        DBMS_OUTPUT.put_line ('Le salaire est' || ' ' || v_employe.sal);
    END IF;
End;
```



Structures de contrôle - Alternatives

case - when - else

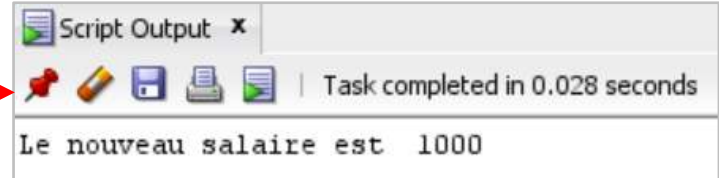
```
-- Augmenter les salaires de 25% pour les employes dont le salaire est inferieur ou egale à 800
Declare
    v_employe EMP%ROWTYPE;
Begin
    Select * into v_employe From Emp Where EMPNO = 7369;    /*doit retourner 1 tuple seulement */
    case
        when v_employe.sal <= 800 THEN
            Begin
                v_employe.sal := v_employe.sal * 1.25;
                DBMS_OUTPUT.put_line ('Le nouveau salaire est' || ' ' || v_employe.sal);
            End;
        ELSE
            DBMS_OUTPUT.put_line ('Le salaire est' || ' ' || v_employe.sal);
        END Case;
End;
```



Structures de contrôle - Alternatives

case - when - else

```
-- Augmenter les salaires de 25% pour les employes dont le salaire est egale à 800
Declare
    v_employe EMP%ROWTYPE;
Begin
    Select * into v_employe From Emp Where EMPNO = 7369;    /*doit retourner 1 tuple seulement */
    case v_employe.sal
        when 800 THEN
            Begin
                v_employe.sal := v_employe.sal * 1.25;
                DBMS_OUTPUT.put_line ('Le nouveau salaire est' || ' ' || v_employe.sal);
            End;
        ELSE
            DBMS_OUTPUT.put_line ('Le salaire est' || ' ' || v_employe.sal);
        END Case;
End;
```



Structures de contrôle - Itératives

while Loop

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

loop – exit when

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

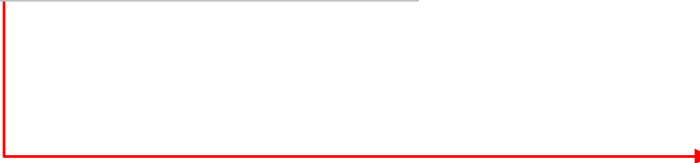
for Loop

```
FOR compteur IN [REVERSE] valeurInf..valeurSup LOOP  
    instructions;  
END LOOP;
```


Structures de contrôle - Itératives

while - loop

```
--Retourner les chiffres de 1 à 6  
Declare  
    v_compteur1 number (6,0) := 0;  
Begin  
    <<B1>> WHILE v_compteur1 <= 5 Loop  
        v_compteur1 := v_compteur1 +1;  
        DBMS_OUTPUT.put_line ( v_compteur1);  
    End Loop B1;  
End;
```



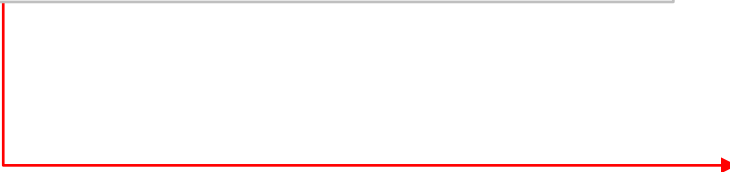
Script Output x

1
2
3
4
5
6

Structures de contrôle - Itératives

loop—exit when

```
--Retourner les chiffres de 1 à 6
Declare
    v_compteur1 number (6,0) := 0;
Begin
    LOOP
        v_compteur1 := v_compteur1 +1; DBMS_OUTPUT.put_line ( v_compteur1);
        EXIT WHEN v_compteur1 > 5;
    End Loop;
End;
```



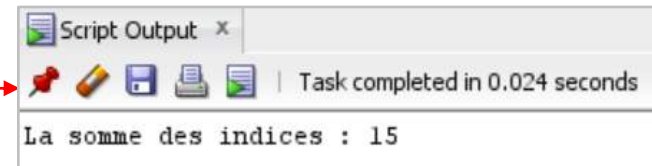
Script Output x

1
2
3
4
5
6

Structures de contrôle - Itératives

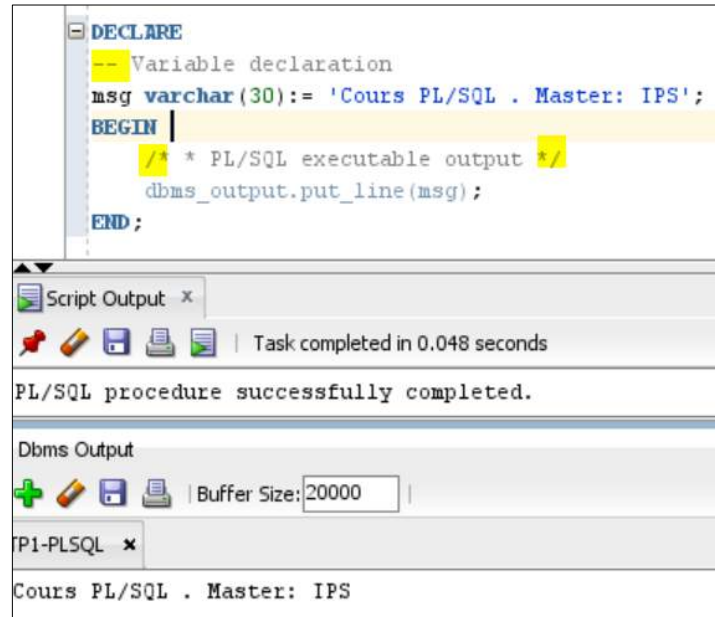
for - loop

```
--Calculer la somme des entiers de 1 à 5  
DECLARE  
    Somme int :=0;  
Begin  
    FOR J IN 1..5 LOOP  
        Somme := Somme + J;  
    END LOOP;  
    DBMS_OUTPUT.put_line ('La somme des indices : ' || Somme);  
END;
```



Commentaires PL / SQL

- Le code PLSQL comprend des commentaires qui expliquent l'intention du code. PL / SQL a à la fois plusieurs lignes et des commentaires sur une seule ligne. Les commentaires sur une seule ligne commencent par un double trait d'union délimiteur -- et les commentaires sur deux lignes commencent par /* et terminent par */



```
DECLARE
-- Variable declaration
msg varchar(30) := 'Cours PL/SQL . Master: IPS';
BEGIN
/* * PL/SQL executable output */
  dbms_output.put_line(msg);
END;
```

Script Output x

Task completed in 0.048 seconds

PL/SQL procedure successfully completed.

Dbms Output

Buffer Size: 20000

P1-PLSQL x

Cours PL/SQL . Master: IPS

Thank you!