

R3.07 - SQL dans un langage de programmation

Abdelbadie Belmouhcine, Mohammed Yasser Khayata

Institut Universitaire de Technologie de Vannes - Université Bretagne Sud
abdelbadie.belmouhcine@univ-ubs.fr

27 novembre 2023



1 Introduction au PL/SQL

2 Bloc PL/SQL

3 Structures de contrôles

4 Curseurs

5 Exceptions

6 Déclencheurs

Ressources

- SAAD, M. (2016). *PL/SQL sous Oracle 12c : Guide du développeur*. Éditions ENI
- SOUTOU, C. (2020). *Programmer avec Oracle : SQL, PL-SQL, XML, JSON, PHP, Java*. Éditions Eyrolles

Introduction au PL/SQL

Système de Gestion de Bases de Données (SGBD)

- ❖ **SGBD** est un **ensemble de programmes** permettant de **créer** et **maintenir** une **base de données**
- ❖ Activités prises en charge :
 - ☞ **Définition** d'une base de données (spécification des **types de données** à stocker)
 - ☞ **Construction** d'une base de données (**stockage des données** proprement dites)
 - ☞ **Manipulation** des données (**ajouter**, **supprimer** et **retrouver des données**)

Exemples de SGBD

Oracle, MySQL, SQL Server, PostgreSQL, DB2, SyBase, Microsoft Access, ...

Base de données relationnelle

- ❖ Collection **bien structurée** de **données opérationnelles interreliées**
- ❖ Entité **cohérente logiquement**, véhiculant une certaine **sémantique**
- ❖ Mise à disposition de **plusieurs utilisateurs**
- ❖ Utilisée pour des **traitements par lots** et à **réponse immédiate**
- ❖ Constituée d'un **ensemble de tables (relations)**

Table

- ❖ Collection de **données structurées** relative à un **domaine bien défini**



Exemples de tables

étudiants d'un établissement universitaire, produits d'une entreprise commerciale

- ❖ Chaque **ligne** de la table correspond à un **enregistrement**
- ❖ Chaque **ligne** est composée de **colonnes**, appelées **champs de données** ou **attributs** de la table

SQL (Structured Query Language)

SQL est un **langage** de **requêtes** ensembliste et assertionnel proposé par le **SGBDR**. Il sert à la **création**, à l'**administration**, à l'**interrogation** et aux **manipulations** des **bases de données relationnelles**



| La norme **SQL 2011** est la plus répandue aujourd'hui

Sous-Langages de SQL I

❖ Le langage SQL compte trois sous-langages :


➡ Langage de Définition des Données (LDD) :

- 👉 CREATE TABLE : Créer une table
- 👉 ALTER TABLE : Ajouter ou modifier une colonne
- 👉 DROP TABLE : Supprimer une table
- 👉 CREATE INDEX : Créer un index
- 👉 DROP INDEX : Supprimer un index
- 👉 TRUNCATE : Supprimer tous les enregistrements d'une table

Sous-Langages de SQL II

- ➡ Langage de Manipulation des Données (LMD) :
 - 👉 INSERT : Insérer des enregistrements dans une table
 - 👉 UPDATE : Modifier des données dans une table
 - 👉 DELETE : Supprimer des enregistrements dans une table
 - 👉 SELECT : Extraire des données à partir d'une base
 - 👉 COMMIT : Valider les opérations de mise à jour pour la transaction en cours
 - 👉 ROLLBACK : Annuler les opérations de mise à jour pour la transaction en cours
- ➡ Langage de Contrôle des Données (LCD) :
 - 👉 GRANT : Attribuer des privilèges à un utilisateur
 - 👉 REVOKE : Supprimer des privilèges d'un utilisateur

Caractéristiques de SQL

 SQL permet d'exprimer des contraintes (d'attribut ou de table)

- ☞ de clé primaire (PRIMARY KEY)
- ☞ de clé étrangère (REFERENCES)
- ☞ d'existence (NOT NULL)
- ☞ d'unicité (UNIQUE)
- ☞ de vérification (CHECK)

 Mais il ne permet pas

- ☞ d'exprimer toutes les contraintes ➡ *Déclencheurs*
- ☞ d'appliquer des traitements « complexes » sur les données ➡ *Langage non procédural*

Présentation et historique I

PL/SQL

- 🔗 Acronyme : **Procedural Language / Structured Query Language**
- 🔗 Extension de **SQL** : les manipulations de données optimisées par le **SGBDR** cohabitent avec les éléments habituels de la programmation structurée
- 🔗 Disponible dans **Oracle** Database depuis la version 7
- 🔗 Combinaison des avantages d'un langage de programmation classique avec les structures algorithmiques et les possibilités de manipulation de données offertes par **SQL**

Présentation et historique II



Évolution de PL/SQL

- Développé par **Oracle** à la fin des années **1980**
- Capacités élargies au début des années **1990**
- Évolution vers un langage de programmation **procédural** et **orienté objet** avec l'avènement des concepts orientés objet dans Oracle 8i et 9i
- À partir de la version 11g, **PL/SQL** est passé d'un langage interprété à un **langage compilé**. Le code **PL/SQL** est stocké dans le **tablespace** système après une compilation directe

Avantages de PL/SQL



Les manipulations de données dans le développement d'une application

- ☞ Interface d'accès aux données : JDBC / PDO ➡ **R3.01**
- ☞ Patron de conception Data Access Object (DAO) ➡ **R4.01**
- ☞ Interface Object-Relational Mapping (ORM) : JPA / Doctrine



Le coût de ces approches peuvent limiter les performances



Le PL/SQL permet d'automatiser de manière efficace et transparente certains traitements grâce aux SGBDR, c'est un complément essentiel

Moteur PL/SQL

- ❖ Lorsque le moteur **PL/SQL** reçoit un **bloc** pour exécution, il effectue les opérations suivantes :
 - 🔊 séparation des commandes **SQL** et **PL/SQL**
 - 🔊 passage des commandes **SQL** au **processeur SQL** (SQL statement executor)
 - 🔊 passage des **instructions procédurales** au **processeur d'instructions procédurales** (procedural statement executor)

Unités PL/SQL

❖ Les **unités** de PL/SQL sont :

- ➡ bloc anonyme PL/SQL
- ➡ fonction
- ➡ procédure
- ➡ déclencheur
- ➡ bibliothèque
- ➡ paquetage (package)
- ➡ corps de paquetage
- ➡ type
- ➡ corps de type

Bloc PL/SQL

Structure d'un programme PL/SQL I

❖ Un programme est structuré en **blocs d'instructions** de 3 types :

- 👉 les **procédures anonymes**
- 👉 les **procédures nommées**
- 👉 les **fonctions nommées**



Structure d'un bloc anonyme

```
DECLARE
    -- déclaration de toutes les variables et tous les types, constantes,
    -- curseurs et exceptions utilisateur référencés dans la section exécutable
BEGIN
    -- instructions PL/SQL (affectations, instructions conditionnelles,
    -- boucles, appels de procédure, etc.) ainsi que les commandes SQL
EXCEPTION
    -- récupération des erreurs
END;
```

Structure d'un programme PL/SQL II



- ☞ Les blocs, comme les instructions, se terminent par un
« ; »
- ☞ Seuls **BEGIN** et **END** sont obligatoires

Identificateur

- ☞ **30** caractères au plus
- ☞ **commence** par une **lettre**
- ☞ peut contenir **lettres**, **chiffres**, **_**, **\$** et **#**
- ☞ **pas** sensible à la **casse**
- ☞ ne correspond **pas** à un **mot réservé** (SELECT, INSERT, UPDATE, COMMIT, ROLLBACK, SUM, MAX, ...)
- ☞ **doit** être distinct des noms de **tables** et de **colonnes**

Littéraux

- ❖ Les **dates** et les **chaînes de caractères** sont délimitées par des **simples quotes** ('20-NOV-2023' ou 'texte')
- ❖ Les valeurs numériques :
 - ✎ Écriture standard : nombre réel avec partie entière et décimale (ex. -5.25)
 - ✎ Écriture scientifique : mantisse et exposant (ex. 2E5 = $2 \times 10^5 = 200000$)

Commentaires

- ❖ Il est évidemment possible (et recommandé) de commenter un programme



```
-- Commentaire  
  
-- Commentaire d'une (fin de) ligne  
  
/*  
Commentaire  
de plusieurs  
lignes  
*/
```

Types de données en PL/SQL

❖ Types scalaires

- ☞ Types de données SQL
- ☞ BOOLEAN
- ☞ PLS_INTEGER
- ☞ BINARY_INTEGER
- ☞ REF CURSOR (curseur de référence)
- ❖ Sous-types définis par l'utilisateur

❖ Types composés

- ☞ Tableaux
- ☞ Enregistrements
- ☞ ...

Types de données - Caractères

❖ CHAR[(n)]

- ☞ Chaîne de caractères de longueur fixe
- ☞ Longueur en PL/SQL : 1 à 32767 octets
- ☞ Longueur en SQL : 1 à 2000 octets
- ☞ Par défaut, *n* est égal à 1

❖ NCHAR[(n)]

- ☞ Identique à CHAR, mais avec représentation en unicode

❖ VARCHAR2(n)

- ☞ Chaîne de caractères de longueur variable
- ☞ Longueur en PL/SQL : 1 à 32767 octets
- ☞ Longueur en SQL : 1 à 4000 octets

❖ NVARCHAR2(n)

- ☞ Identique à VARCHAR2, avec représentation en unicode

❖ LONG

- ☞ Semblable à VARCHAR2
- ☞ Longueur en PL/SQL : max 32760 octets
- ☞ Longueur max en SQL : 2 Go

❖ RAW(n)

- ☞ Données binaires de longueur variable
- ☞ Longueur en PL/SQL : 1 à 32767 octets
- ☞ Longueur en SQL : 1 à 2000 octets
- ☞ Aucune conversion de caractères lors du transfert

❖ LONG RAW

- ☞ Semblable à LONG, avec données binaires en hexadécimal
- ☞ Longueur max en PL/SQL : 32760 octets
- ☞ Longueur max en SQL : 2Go

Types de données - Numériques I

❖ NUMBER[n[, m]]

- ☞ Valeur numérique décimale
- ☞ n chiffres significatifs (38 par défaut)
- ☞ m positions décimales ou entières (0 par défaut)
- ☞ Plage : -2^{418} à 2^{418}

❖ PLS_INTEGER et BINARY_INTEGER

- ☞ Entiers de -2^{31} à 2^{31}
- ☞ PLS_INTEGER plus rapide (utilise les registres du processeur)



Le type **PLS_INTEGER** et ses sous-types peuvent être convertis implicitement vers les types de données suivants : **CHAR**, **VARCHAR2**, **NUMBER** et **LONG**. Aussi, tous les types de données précédents, à l'exception du type **LONG**, et tous les sous-types de **PLS_INTEGER**, peuvent être implicitement convertis en **PLS_INTEGER**.

Types de données - Numériques II

❖ BINARY_FLOAT

- ☞ Nombre à virgule flottante simple précision
- ☞ 5 octets par valeur
- ☞ Valeurs entre -3.4×10^{38} et 3.4×10^{38}
- ☞ Plus petite valeur positive : 1.2×10^{-38}
- ☞ Plus grande valeur négative : -1.2×10^{-38}

❖ BINARY_DOUBLE

- ☞ Nombre à virgule flottante double précision
- ☞ 9 octets par valeur
- ☞ Valeurs entre -1.79×10^{308} et 1.79×10^{308}
- ☞ Plus petite valeur positive : 2.3×10^{-308}
- ☞ Plus grande valeur négative : -2.3×10^{-308}



SIMPLE_FLOAT et **SIMPLE_DOUBLE** sont des sous-types de **BINARY_FLOAT** et **BINARY_DOUBLE** avec la contrainte **NOT NULL**

Types de données - Grands objets

❖ CLOB

- Permet de stocker un flot de caractères, par exemple un texte
- Taille maximale de 128 To en PL/SQL et 4 Go en SQL

❖ NCLOB

- Identique à CLOB, mais les données de ce type sont représentées par le unicode

❖ BLOB

- Permet de stocker un objet binaire non structuré, comme le multimédia (images, sons, vidéo, etc.)
- Taille maximale de l'objet de 128 To en PL/SQL et 4 Go en SQL

❖ BFILE

- Permet de stocker des données binaires dans un fichier externe à la base

Types de données - Autres types

❖ DATE

- ☞ Permet de stocker des dates, constituées du siècle, de l'année, du mois... des secondes
- ☞ Plage de type DATE de 1/1/-4712 à 31/12/9999. Par défaut, heures, minutes et secondes valent 00:00:00

❖ TIMESTAMP[(n)]

- ☞ Permet de stocker des dates et heures avec la granularité des fractions de secondes
- ☞ Précision des fractions de secondes de 0 à 9 (par défaut n=6)

❖ TIMESTAMP WITH TIME ZONE

- ☞ Permet de stocker des dates et heures de type TIMESTAMP avec la prise en compte des fuseaux horaires

❖ TIMESTAMP WITH LOCAL TIME ZONE

- ☞ Permet de faire la distinction entre une heure de serveur et une heure du client

❖ INTERVAL YEAR TO MONTH

- ☞ Permet de stocker la différence de deux dates avec la précision mois/année

❖ INTERVAL DAY TO SECOND

- ☞ Permet d'enregistrer une période de temps en jours, heures, minutes et secondes

❖ BOOLEAN

- ☞ Stocke les valeurs booléennes logiques (VRAI, FAUX, NULL représente une valeur inconnue)
- ☞ **Existe seulement en PL/SQL**

Déclarations de variables



DECLARE

```
sexe CHAR(1);  
/* déclaration d'une variable pour stocker le sexe d'une personne (M ou F)  
*/  
i BINARY_INTEGER:= 0;  
/* déclaration d'un compteur initialisé à 0 */  
date_commande DATE:= SYSDATE;  
/* déclaration d'une variable pour stocker la date de création d'une  
commande. La variable est initialisée à la date du jour.*/  
message VARCHAR2(30):='Bonjour';  
/* déclaration d'une variable pour stocker un message, qui est initialisée  
à 'Bonjour' */  
note FLOAT := 10.5;  
/* déclaration d'une variable pour stocker la note, qui est initialisée à  
10,5 */
```

BEGIN

```
NULL;  
END;
```

Déclaration par un type de données



DECLARE

```
nom EMPLOYE.ENOM%TYPE;  
sal EMPLOYE.SALAIRE%TYPE;  
/* déclaration de deux variables nom et sal ayant les mêmes type de données  
   que les champs ENOM et SALAIRE de la table EMPLOYE */
```

```
min_salaire sal%type;  
/* déclaration d'une variable min_salaire ayant le même type de données que  
   la variable sal */
```

BEGIN

```
NULL;
```

END;

Déclarations de constantes



DECLARE

```
pi NUMBER(8,5) :=3.14159; -- type de données SQL  
min_salaire CONSTANT REAL := 1000.00; -- type de données SQL  
trouve CONSTANT BOOLEAN := FALSE; -- type de données PL/SQL
```

BEGIN

```
    NULL;  
END;
```

Paquetage DBMS_OUTPUT en Oracle I

- ❖ **DBMS_OUTPUT** assure la gestion des **entrées-sorties** dans les **blocs PL/SQL**. Les procédures clés sont :
 - ❖ **ENABLE** : active les entrées-sorties
 - ❖ **DISABLE** : désactive les entrées-sorties
 - ❖ **PUT (expression)** : affiche l'expression à l'écran
 - ❖ **NEW_LINE** : effectue un retour à la ligne
 - ❖ **PUT_LINE (expression)** : appelle PUT suivi d'un appel de NEW_LINE
 - ❖ **GET_LINE (out chaîne, out état)** : lit une ligne dans les paramètres chaîne, avec état prenant la valeur 0 si la valeur lue est valide
 - ❖ **GET_LINES (out chaîne, in-out nbre_lignes)** : permet la lecture de plusieurs lignes

Paquetage DBMS_OUTPUT en Oracle II



Avant d'utiliser ce paquetage, on doit l'activer au préalable avec la commande **SET SERVEROUTPUT ON**. L'appel de toute procédure d'un paquetage se réalise avec l'instruction **nom_paquetage.nom_procédure(paramètres)**

Exemple



```
-- activation du paquetage sous SQL*PLUS
SET SERVEROUTPUT ON

DECLARE
    nom VARCHAR2(25) := 'THOMAS';
    fonct VARCHAR2(25) := 'PRESIDENT';
    sal NUMBER := 5000;

BEGIN

    DBMS_OUTPUT.ENABLE; -- Activation du paquetage sous PL/SQL
    DBMS_OUTPUT.PUT_LINE('Votre nom est      || nom||','');
    DBMS_OUTPUT.PUT_LINE('votre fonction est  || fonct||','');
    DBMS_OUTPUT.PUT_LINE('et votre salaire est || sal||.');
```

END;

Exemple



```
-- activation du paquetage sous SQL*PLUS
SET SERVEROUTPUT ON

DECLARE
    nom VARCHAR2(25) := 'THOMAS';
    fonct VARCHAR2(25) := 'PRESIDENT';
    sal NUMBER := 5000;

BEGIN

    DBMS_OUTPUT.ENABLE; -- Activation du paquetage sous PL/SQL
    DBMS_OUTPUT.PUT_LINE('Votre nom est      || nom||','');
    DBMS_OUTPUT.PUT_LINE('votre fonction est  || fonct||','');
    DBMS_OUTPUT.PUT_LINE('et votre salaire est || sal||'.');

END;
```

Votre nom est THOMAS ,
votre fonction est PRESIDENT ,
et votre salaire est 5000.

Affectation

❖ L'**affectation** de variable s'effectue :

- ☞ directement avec « **:=** »
- ☞ via une requête **SELECT** avec la directive **INTO**



Conflits de noms

- ❖ Si une **variable** porte le **même nom** qu'une **colonne**, c'est la **colonne** qui l'emporte ce qui peut provoquer de graves erreurs
- ❖ Pour éviter cela, on peut préfixer par « **v__** » le nom d'une **variable**^a

a. et par « **p__** » le nom d'un **paramètre** (bonne pratique pour les procédures et fonctions)

Exemple



Affectation simple

```
SET SERVEROUTPUT ON
DECLARE
  a INTEGER;
  b INTEGER;
  c INTEGER;
BEGIN -- affectation des variables
  a := 1;
  DBMS_OUTPUT.PUT_LINE('A = ' || a);
  b := a+3;
  DBMS_OUTPUT.PUT_LINE('B = ' || b);
  a := 3;
  DBMS_OUTPUT.PUT_LINE('A = ' || a);
  b := 5;
  DBMS_OUTPUT.PUT_LINE('B = ' || b);
  c := a+b;
  DBMS_OUTPUT.PUT_LINE('C = ' || c);
  c := b-a;
  DBMS_OUTPUT.PUT_LINE('C = ' || c);
END;
```

Exemple



Affectation simple

```
SET SERVEROUTPUT ON
DECLARE
  a INTEGER;
  b INTEGER;
  c INTEGER;
BEGIN -- affectation des variables
  a := 1;
  DBMS_OUTPUT.PUT_LINE('A = ' || a);
  b := a+3;
  DBMS_OUTPUT.PUT_LINE('B = ' || b);
  a := 3;
  DBMS_OUTPUT.PUT_LINE('A = ' || a);
  b := 5;
  DBMS_OUTPUT.PUT_LINE('B = ' || b);
  c := a+b;
  DBMS_OUTPUT.PUT_LINE('C = ' || c);
  c := b-a;
  DBMS_OUTPUT.PUT_LINE('C = ' || c);
END;
```

```
A = 1
B = 4
A = 3
B = 5
C = 8
C = 2
```

Exemple



Affectation et initialisation

```
SET SERVEROUTPUT ON
DECLARE
    /* on peut faire l'affectation dans la partie déclarative : initialisation
       */
    salaire NUMBER;
    heure_travail NUMBER := 40;
    salaire_horaire NUMBER := 22.50;
    bonus NUMBER := 100;
    pays VARCHAR2(50);
    nom VARCHAR2(50) := 'thomas';
    max_sal NUMBER := 700;
    valide BOOLEAN;
BEGIN -- on peut faire l'affectation dans le corps du bloc
    salaire := (salaire_horaire * heure_travail) + bonus;
    nom := UPPER(nom);
    pays := 'France';
    valide := (salaire > max_sal);
    DBMS_OUTPUT.PUT_LINE(nom || ' habite en ' || pays);
    DBMS_OUTPUT.PUT_LINE(' Il gagne ' || salaire);
END;
```

Exemple



Affectation et initialisation

```
SET SERVEROUTPUT ON
DECLARE
    /* on peut faire l'affectation dans la partie déclarative : initialisation
       */
    salaire NUMBER;
    heure_travail NUMBER := 40;
    salaire_horaire NUMBER := 22.50;
    bonus NUMBER := 100;
    pays VARCHAR2(50);
    nom VARCHAR2(50) := 'thomas';
    max_sal NUMBER := 700;
    valide BOOLEAN;
BEGIN -- on peut faire l'affectation dans le corps du bloc
    salaire := (salaire_horaire * heure_travail) + bonus;
    nom := UPPER(nom);
    pays := 'France';
    valide := (salaire > max_sal);
    DBMS_OUTPUT.PUT_LINE(nom || ' habite en ' || pays);
    DBMS_OUTPUT.PUT_LINE(' Il gagne ' || salaire);
END;
```

L'employé THOMAS a pour matricule 7800

R3.07 - SQL dans un langage de programmation

Exemple



Affectation par select

```
SET SERVEROUTPUT ON
DECLARE
    num NUMBER;
    nom_emp VARCHAR2(30) := 'THOMAS';
BEGIN
    -- affectation des variables par l'instruction SELECT ... INTO
    SELECT MATRICULE INTO num
    FROM EMPLOYE
    WHERE ENOM= nom_emp;
    DBMS_OUTPUT.PUT_LINE('L'employé ' || nom_emp || ' a pour matricule ' || num);
END;
```

Exemple



Affectation par select

```
SET SERVEROUTPUT ON
DECLARE
    num NUMBER;
    nom_emp VARCHAR2(30) := 'THOMAS';
BEGIN
    -- affectation des variables par l'instruction SELECT ... INTO
    SELECT MATRICULE INTO num
    FROM EMPLOYE
    WHERE ENOM= nom_emp;
    DBMS_OUTPUT.PUT_LINE('L'employé ' || nom_emp || ' a pour matricule ' || num);
END;
```

THOMAS habite en France
Il gagne 1000

Priorité



```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
  a INTEGER := 1+3**3;  
  b INTEGER := (1+3)**3;  
  c INTEGER := (((1+3)*(4+5))/6);  
  d INTEGER := 2**2*3**2;  
  e INTEGER := (2**2)*(3**2);
```

```
BEGIN
```

```
  DBMS_OUTPUT.PUT_LINE('a = ' || a);  
  DBMS_OUTPUT.PUT_LINE('b = ' || b);  
  DBMS_OUTPUT.PUT_LINE('c = ' || c);  
  DBMS_OUTPUT.PUT_LINE('d = ' || d);  
  DBMS_OUTPUT.PUT_LINE('e = ' || e);
```

```
END;
```

Priorité



```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
a INTEGER := 1+3**3;  
b INTEGER := (1+3)**3;  
c INTEGER := (((1+3)*(4+5))/6);  
d INTEGER := 2**2*3**2;  
e INTEGER := (2**2)*(3**2);
```

```
a = 28  
b = 64  
c = 6  
d = 36  
e = 36
```

```
BEGIN
```

```
DBMS_OUTPUT.PUT_LINE('a = ' || a);  
DBMS_OUTPUT.PUT_LINE('b = ' || b);  
DBMS_OUTPUT.PUT_LINE('c = ' || c);  
DBMS_OUTPUT.PUT_LINE('d = ' || d);  
DBMS_OUTPUT.PUT_LINE('e = ' || e);
```

```
END;
```

Blocs imbriqués et portée d'un objet



```
SET SERVEROUTPUT ON
-- bloc parent
DECLARE
  x INTEGER :=10;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Bloc parent');
  DBMS_OUTPUT.PUT_LINE('x = ' || x);
-- sous bloc
  DECLARE
    y INTEGER;
  BEGIN
    x :=x+5;
    DBMS_OUTPUT.PUT_LINE('Sous Bloc');
    DBMS_OUTPUT.PUT_LINE('x = ' || x);

  END;
END;
```

Blocs imbriqués et portée d'un objet



```
SET SERVEROUTPUT ON
-- bloc parent
DECLARE
  x INTEGER :=10;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Bloc parent');
  DBMS_OUTPUT.PUT_LINE('x = ' || x);
-- sous bloc
DECLARE
  y INTEGER;
BEGIN
  x :=x+5;
  DBMS_OUTPUT.PUT_LINE('Sous Bloc');
  DBMS_OUTPUT.PUT_LINE('x = ' || x);

END;
END;
```

Bloc parent
x = 10
Sous Bloc
x = 15

Variables de substitution



```
SET SERVEROUTPUT ON
DECLARE
    mat NUMBER := &s_mat; -- variable de substitution
    -- On suppose que le matricule saisi est correcte
    nom EMPLOYE.ENOM%TYPE;
    fonct EMPLOYE.FONCTION%TYPE;
    sal EMPLOYE.SALAIRE%TYPE;
BEGIN
    SELECT ENOM, FONCTION, SALAIRE INTO nom, fonct, sal FROM EMPLOYE WHERE
        MATRICULE = mat;
    DBMS_OUTPUT.PUT_LINE ('L'employé de nom '||nom);
    DBMS_OUTPUT.PUT_LINE ('Sa fonction est '|| fonct);
    DBMS_OUTPUT.PUT_LINE ('Son salaire est '|| sal);
END;
```

Variables de substitution



```
SET SERVEROUTPUT ON
DECLARE
    mat NUMBER := &s_mat; -- variable de substitution
    -- On suppose que le matricule saisi est correcte
    nom EMPLOYE.ENOM%TYPE;
    fonct EMPLOYE.FONCTION%TYPE;
    sal EMPLOYE.SALAIRE%TYPE;
BEGIN
    SELECT ENOM, FONCTION, SALAIRE INTO nom, fonct, sal FROM EMPLOYE WHERE
        MATRICULE = mat;
    DBMS_OUTPUT.PUT_LINE ('L'employé de nom '||nom);
    DBMS_OUTPUT.PUT_LINE ('Sa fonction est '|| fonct);
    DBMS_OUTPUT.PUT_LINE ('Son salaire est '|| sal);
END;
```

Entrez une valeur pour s_mat : 7800
ancien :... mat NUMBER := &s_mat; ...
nouveau :... mat NUMBER := 7800; ...
L'employé de nom THOMAS
Sa fonction est PRESIDENT
Son salaire est 5000

Variables de substitution - Accept



```
ACCEPT s_mat PROMPT 'Entrer Matricule Employé : '  
SET SERVEROUTPUT ON  
DECLARE  
    nom EMPLOYE.ENOM%TYPE;  
    fonct EMPLOYE.FONCTION%TYPE;  
    sal EMPLOYE.SALAIRE%TYPE;  
BEGIN  
    SELECT ENOM, FONCTION, SALAIRE FROM EMPLOYE WHERE MATRICULE = &s_mat;  
    DBMS_OUTPUT.PUT_LINE ('L'employé de nom '||nom);  
    DBMS_OUTPUT.PUT_LINE ('Sa fonction est '|| fonct);  
    DBMS_OUTPUT.PUT_LINE ('Son salaire est '|| sal);  
END;
```

Variables de substitution - Accept



```
ACCEPT s_mat PROMPT 'Entrer Matricule Employé : '  
SET SERVEROUTPUT ON  
DECLARE  
    nom EMPLOYE.ENOM%TYPE;  
    fonct EMPLOYE.FONCTION%TYPE;  
    sal EMPLOYE.SALAIRE%TYPE;  
BEGIN  
    SELECT ENOM, FONCTION, SALAIRE FROM EMPLOYE WHERE MATRICULE = &s_mat;  
    DBMS_OUTPUT.PUT_LINE ('L'employé de nom '||nom);  
    DBMS_OUTPUT.PUT_LINE ('Sa fonction est '|| fonct);  
    DBMS_OUTPUT.PUT_LINE ('Son salaire est '|| sal);  
END;
```

Entrer Matricule Employé : 7800
ancien ... WHERE MATRICULE = s_mat; ...
nouveau : ... WHERE MATRICULE = 7800; ...
L'employé de nom THOMAS
Sa fonction est PRESIDENT
Son salaire est 5000

Variables de session (hôtes)



```
VARIABLE g_x NUMBER;  
VARIABLE g_y NUMBER;  
DECLARE  
    z NUMBER;  
BEGIN  
    :g_x := 10;  
    :g_y := 15;  
    z := :g_x;  
    :g_x := :g_y;  
    :g_y := z;  
END;  
/  
PRINT g_x;  
PRINT g_y;
```

Variables de session (hôtes)



```
VARIABLE g_x NUMBER;  
VARIABLE g_y NUMBER;  
DECLARE  
    z NUMBER;  
BEGIN  
    :g_x := 10;  
    :g_y := 15;  
    z := :g_x;  
    :g_x := :g_y;  
    :g_y := z;  
END;  
/  
PRINT g_x;  
PRINT g_y;
```

G_X
15
G_Y
10

Structures de contrôles

Structures de contrôle en PL/SQL

❖ Structures conditionnelles :

- ☞ L'instruction **IF** : Utilisée pour exécuter un ensemble d'instructions si une condition est vraie
- ☞ L'instruction **CASE** : Utilisée pour effectuer différentes actions en fonction de la valeur d'une expression

❖ Structures itératives :

- ☞ La boucle de base **LOOP** : Répète une série d'instructions sans condition globale
- ☞ La boucle **FOR** : Contrôle le nombre d'itérations grâce à un indice



La boucle **FOR** peut aussi être utilisée avec une **collection indexée** ou un **curseur** lors de l'itération à travers les enregistrements d'une requête SQL

- ☞ La boucle **WHILE** : Contrôle les itérations selon la vérification d'un prédicat

Conditions logiques en PL/SQL

- ❖ Une **condition** en PL/SQL est une expression booléenne évaluée comme **TRUE**, **FALSE** ou **NULL**

Opérateur	Description
<	Inférieur
>	Supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal
=	Égal
<>, !=, =, ^=	Différent
IS NULL	Vérifie si l'opérande est NULL
IS NOT NULL	Vérifie si l'opérande n'est pas NULL
LIKE	Comparaison partielle avec % et _
BETWEEN	Appartenance à un intervalle
IN	Appartenance à une liste de valeurs prédéfinies
AND	Et logique
OR	Ou logique
NOT	Négation logique

Traitements conditionnels I

❖ On dispose du :



IF

```
-- Branchement conditionnel
IF condition THEN
    instructions;
[ELSIF condition THEN
    instructions;]
[ELSE
    instructions;]
END IF;
```


Traitements conditionnels II

❖ ainsi que du :



CASE simple

```
-- Case simple
CASE selecteur
  WHEN expression_1 THEN instructions_1;
  WHEN expression_2 THEN instructions_2;
  ...
  WHEN expression_n THEN instructions_n;
[ELSE
  instructions_else;]
END CASE;
```



Si aucune condition n'est vraie et que l'instruction **ELSE** n'existe pas dans la commande, alors l'exception prédéfinie **CASE_NOT_FOUND** est soulevée

Traitements conditionnels III



CASE de recherche

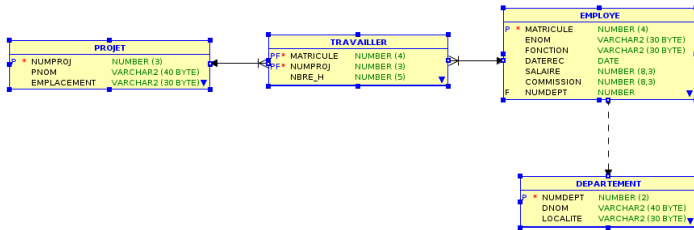
```
-- Case de recherche
CASE
  WHEN condition_1 THEN instructions_1;
  WHEN condition_2 THEN instructions_2;
  ...
  WHEN condition_n THEN instructions_n;
[ELSE
  instructions_else;]
END CASE;
```



Si aucune condition n'est vraie et que l'instruction **ELSE** n'existe pas dans la commande, alors l'exception prédéfinie **CASE_NOT_FOUND** est soulevée

Exemple - IF

❖ Soit le SR suivant :



❖ Écrire un bloc **PL/SQL** qui permet d'augmenter le salaire d'un employé particulier, selon le nombre des heures de travail (quota) dans des projets :

```
🔧 salaire = salaire + 0.1 * quota, si quota <= 100,  
🔧 salaire = salaire + 0.2 * quota, si 100 < quota <= 200,  
🔧 salaire = salaire + 0.3 * quota, si 200 < quota <= 300,  
🔧 salaire = salaire + 0.4 * quota, si quota >300
```

Exemple - IF



Solution

```
DECLARE
-- déclaration des variables
    quota TRAVAILLER.NBRE_H%TYPE;
    bonus EMPLOYE.SALAIRE%TYPE;
    sal EMPLOYE.SALAIRE%TYPE;
    emp_id EMPLOYE.MATRICULE%TYPE :=7566;
BEGIN
-- sélection de nombre des heures de travail
    SELECT SUM(NBRE_H) INTO quota FROM TRAVAILLER WHERE MATRICULE = emp_id;
-- sélection de salaire
    SELECT SALAIRE INTO sal FROM EMPLOYE WHERE MATRICULE = emp_id;
-- détermination de bonus
    IF quota <= 100 THEN bonus := 0.1*quota;
    ELSIF quota <= 200 THEN bonus := 0.2*quota;
    ELSIF quota <= 300 THEN bonus := 0.3*quota;
    ELSE bonus := 0.4*quota;
    END IF;
-- modification de salaire
    UPDATE EMPLOYE SET SALAIRE = SALAIRE + bonus WHERE MATRICULE = emp_id;
-- sélection de salaire après modification
    SELECT SALAIRE INTO sal FROM EMPLOYE WHERE MATRICULE = emp_id;
END;
```

Exemple - CASE simple

- ❖ Écrire un bloc **PL/SQL** qui, à partir d'une appréciation donnée, affiche la mention associée

Exemple - CASE simple

- ❖ Écrire un bloc **PL/SQL** qui, à partir d'une appréciation donnée, affiche la mention associée



Solution

```
SET SERVEROUTPUT ON
DECLARE
  app CHAR(1);

BEGIN
  app := 'B';
  -- affichage de la mention
  CASE app
    WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
    WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Très bien');
    WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Bien');
    WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('Passable');
    WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Faible');
  ELSE
    DBMS_OUTPUT.PUT_LINE(' appréciation inconnue !!!');
  END CASE;
END;
```

Exemple - CASE de recherche

- ❖ Écrire un bloc **PL/SQL** qui, à partir d'une moyenne donnée, affiche la mention associée

Exemple - CASE de recherche

- ❖ Écrire un bloc **PL/SQL** qui, à partir d'une moyenne donnée, affiche la mention associée



Solution

```
SET SERVEROUTPUT ON
DECLARE
    moyenne NUMBER(4,2) :=13.50;
BEGIN
    CASE
        WHEN moyenne between 18 and 20 THEN
            DBMS_OUTPUT.PUT_LINE('Excellent');
        WHEN moyenne between 16 and 17.99 THEN
            DBMS_OUTPUT.PUT_LINE('Très Bien');
        WHEN moyenne between 14 and 15.99 THEN
            DBMS_OUTPUT.PUT_LINE('Bien');
        WHEN moyenne between 12 and 13.99 THEN
            DBMS_OUTPUT.PUT_LINE('Assez bien');
        WHEN moyenne between 10 and 11.99 THEN
            DBMS_OUTPUT.PUT_LINE('Passable');
        WHEN moyenne between 0 and 9.99 THEN
            DBMS_OUTPUT.PUT_LINE('Redouble');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Moyenne inconnue !!!');
        END CASE;
END;
```


Traitements répétitifs I

❖ On dispose de la :



boucle « générale »

```
-- Boucle "générale"  
LOOP  
  instructions;  
  ...  
  [EXIT [WHEN condition_1;]]  
  ...  
  [CONTINUE [WHEN condition_2;]]  
  ...  
END LOOP;
```



! Une boucle **sans EXIT** est une boucle **infinie**

Traitements répétitifs II

❖ ainsi que de la :



boucle « tant que »

```
-- Boucle "tant que"  
WHILE condition LOOP  
    instructions;  
    ...  
END LOOP;
```

Traitements répétitifs III

❖ sans oublier la :



boucle « pour »

```
-- Boucle "pour"  
FOR compteur IN [REVERSE] inf..sup LOOP  
    instructions;  
    ...  
END LOOP;
```



Le compteur est déclaré **implicitement** comme un **entier**. Il n'est pas nécessaire de le déclarer

Traitements répétitifs IV



Parmi toutes ces boucles, on utilisera principalement la boucle « **pour** », mais avec un **curseur**...

Curseurs

Curseurs PL/SQL

- ❖ Une **zone mémoire privée** de SQL contenant le **résultat** d'une requête
- ❖ Deux types de **curseurs** :
 - ☞ **Curseurs implicites** : Déclarés automatiquement lors de l'exécution d'une instruction SQL. L'utilisateur n'a pas de contrôle sur ces curseurs
 - ☞ **Curseurs explicites** : Générés et gérés par le programmeur pour traiter une requête **SELECT** renvoyant plusieurs lignes (définis dans la section DECLARE)

Curseurs implicites

- ❖ Géré implicitement par PL/SQL lors de l'exécution d'instructions comme **SELECT**, **INSERT**, **DELETE** ou **UPDATE**
- ❖ Le **programmeur** ne peut pas le contrôler, mais peut obtenir des informations à partir de ses attributs
- ❖ Principaux attributs :
 - ❖ **SQL%ISOPEN** : Toujours **FALSE**, car le curseur implicite se ferme après l'exécution de l'instruction
 - ❖ **SQL%FOUND** : **TRUE** si l'instruction **SELECT** ou **LMD** affecte au moins une ligne
 - ❖ **SQL%NOTFOUND** : **TRUE** si l'instruction **SELECT** ou **LMD** ne renvoie aucune ligne
 - ❖ **SQL%ROWCOUNT** : Retourne le nombre de lignes renvoyées par **SELECT** ou affectées par la dernière **LMD**

Exemple



Attributs des curseurs implicites

```
SET SERVEROUTPUT ON
BEGIN
  -- modification des projets dont l'emplacement est PARIS
  UPDATE PROJET
  SET EMPLACEMENT='BORDEAUX'
  WHERE EMPLACEMENT='PARIS';
  -- vérification si des modifications sont effectuées
  IF SQL%FOUND THEN DBMS_OUTPUT.PUT_LINE ('Modification effectuée');
  ELSE DBMS_OUTPUT.PUT_LINE ('Pas de modification');
  END IF;
  -- vérification si des modifications sont effectuées
  IF SQL%NOTFOUND THEN DBMS_OUTPUT.PUT_LINE('Pas de modification');
  ELSE DBMS_OUTPUT.PUT_LINE('Modification effectuée');
  END IF;
  -- Suppression des lignes de la table TRAVAILLER
  DELETE FROM TRAVAILLER;
  -- le nombre des lignes supprimées
  DBMS_OUTPUT.PUT_LINE('Le nombre des lignes supprimées de la table
    TRAVAILLER est  ' || SQL%ROWCOUNT);
  ROLLBACK;
END;
```


Curseurs explicites

- ❖ Construit et géré par le **programmeur**
- ❖ Offre un contrôle plus programmatique par rapport au **curseur implicite**
- ❖ Permet de traiter un ensemble de lignes de manière efficace



Déclaration et définition

```
-- Curseur explicite
DECLARE
  -- Déclaration
  CURSOR nom_curseur RETURN return_type;
  -- Définition
  CURSOR nom_curseur IS requête_select;
```

Exemple



Déclaration et définition

```
DECLARE
-- déclaration c1
  CURSOR c1 RETURN PROJET%ROWTYPE;
-- déclaration et définition c2
  CURSOR c2 IS SELECT * FROM EMPLOYE WHERE SALAIRE > 4000;
-- définition c1
  CURSOR c1 RETURN PROJET%ROWTYPE IS SELECT * FROM PROJET;
-- déclaration c3
  CURSOR c3 RETURN TRAVAILLER%ROWTYPE;
-- définition c3
  CURSOR c3 IS SELECT * FROM TRAVAILLER WHERE MATRICULE = 780;
BEGIN
  NULL;
END;
```

Exemple - Curseurs (I)

- ❖ Écrire un bloc **PL/SQL** qui permet d'afficher les villes où a travaillé l'employé ayant le matricule 7600

Exemple - Curseurs (I)

- ❖ Écrire un bloc **PL/SQL** qui permet d'afficher les villes où a travaillé l'employé ayant le matricule 7600



Solution (Parmi d'autres)

```
SET SERVEROUTPUT ON
DECLARE
-- déclaration du curseur
    CURSOR c IS SELECT DISTINCT EMPLACEMENT FROM TRAVAILLER T, PROJET P WHERE T
        .NUMPROJ = P.NUMPROJ AND MATRICULE=7600 ORDER BY EMPLACEMENT;
BEGIN
    DBMS_OUTPUT.PUT_LINE('L'employé de matricule 7600 a travaillé dans les
        villes :');
    FOR rec IN c LOOP -- lecture du curseur
        DBMS_OUTPUT.PUT_LINE('-- ' || rec.EMPLACEMENT);
    END LOOP;
END;
```

Exemple - Curseurs (II)

- ❖ Écrire un bloc **PL/SQL** qui permet d'afficher les noms et les salaires des employés ayant la fonction 'MANAGER'

Exemple - Curseurs (II)

- ❖ Écrire un bloc **PL/SQL** qui permet d'afficher les noms et les salaires des employés ayant la fonction 'MANAGER'



Solution

```
SET SERVEROUTPUT ON
DECLARE
-- déclaration du curseur
    CURSOR emp_curseur IS SELECT ENOM, SALAIRE FROM EMPLOYE WHERE FONCTION = '
        MANAGER';
BEGIN
    -- chargement et accès aux éléments du curseur
    FOR rec1 in emp_curseur LOOP
        DBMS_OUTPUT.PUT_LINE( rec1.enom||' a le salaire '|| rec1.salaire);
    END LOOP;
END;
```

Exceptions

Déclencheurs

