CHAPITRE 5

PL/SQL

Dr. Nassira ACHICH

achichnassira@gmail.com

INTRODUCTION AU PL/SQL

- PL/SQL (Procedural Language / SQL) est un langage procédural intégré à Oracle.
- Il combine :
  - La puissance du SQL (manipulation de données),
  - Les structures de contrôle d'un langage de programmation (conditions, boucles, procédures…).
- Pourquoi utiliser PL/SQL ?
  - Automatiser des traitements (batchs, calculs…)
  - Créer des programmes stockés dans la base (procédures, fonctions)
  - Gérer les exceptions (erreurs)
  - Améliorer la performance en regroupant plusieurs requêtes dans un bloc PL/SQL
  - Renforcer la sécurité

ARCHITECTURE D'UN PROGRAMME PL/SQL

- Un bloc PL/SQL contient 3 parties :

```
DECLARE       -- Partie optionnelle : on déclare des variables

BEGIN         -- Partie obligatoire : instructions à exécuter

EXCEPTION     -- Partie optionnelle : gestion des erreurs

END;
/
```

- **DECLARE** : On crée des variables, des constantes,…
- **BEGIN** : On met le code exécuté (affichages, calculs, requêtes SQL…)
- **EXCEPTION** : On capture les erreurs (ex : aucune ligne trouvée)
- **/** : Indique à Oracle d'exécuter le bloc

ARCHITECTURE D'UN PROGRAMME PL/SQL

➢ Exemple

```
BEGIN
   DBMS_OUTPUT.PUT_LINE('Bonjour !');
END;
/
```

- **DBMS_OUTPUT.PUT_LINE** permet d'afficher du texte dans la console Oracle.

TYPES DES VARIABLES

- Les types habituels correspondants aux types SQL2 ou Oracle : integer, varchar,...
- Les types composites adaptés à la récupération des colonnes et des lignes des tables SQL : %TYPE, %ROWTYPE
- Type référence : REF

DéCLARATION DES VARIABLES

- Syntaxe générale

```
DECLARE
nom_variable type [:= valeur_initiale];
```

- **nom_variable** → le nom que vous choisissez
- **type** → VARCHAR2, NUMBER, DATE, BOOLEAN…
- **:=** → permet d'attribuer une valeur au moment de la déclaration
- Déclarations multiples interdites !!!! (I, j integer ; interdit !)

DéCLARATION DES VARIABLES

- Exemple

```
DECLARE
   nom VARCHAR2(20) := ‘Fatma';
   age NUMBER := 21;

BEGIN
   DBMS_OUTPUT.PUT_LINE(nom || ' a ' || age || ' ans');

END;
/
```

DéCLARATION DES VARIABLES

➢ Déclarer une variable basée sur une colonne

➢Syntaxe

```
variable nom_table.colonne%TYPE;
```

➢ Exemple:

➢Si la table employe contient : salaire NUMBER(8,2)

```
DECLARE
   v_sal employe.salaire%TYPE;
BEGIN
   ...
END;
/
```

DéCLARATION DES VARIABLES

➢ Déclarer une variable basée sur une ligne entière

➢Syntaxe

```
nom_variable nom_table%ROWTYPE;
```

➢ Exemple:

```
DECLARE
   v_emp employe%ROWTYPE;
BEGIN
   SELECT * INTO v_emp
   FROM employe
   WHERE id = 101;

   DBMS_OUTPUT.PUT_LINE(v_emp.nom || ' : ' || v_emp.salaire);
END;
/
```

SELECT… INTO (OBLIGATOIRE)

- En PL/SQL, un SELECT doit obligatoirement aller dans une variable.

- **Syntaxe**

```
SELECT liste_colonnes
INTO variables
FROM table
WHERE condition;
```

- **Exemple:**

```
DECLARE
   v_nom employe.nom%TYPE;
BEGIN
   SELECT nom INTO v_nom
   FROM employe
   WHERE id = 10;

   DBMS_OUTPUT.PUT_LINE('Nom : ' || v_nom);
END;
/
```

MANIPULATION DES DONNéES

➢ **Les instructions SQL sont identiques à SQL normal.**

```
BEGIN
   INSERT INTO employe(id, nom, salaire)
   VALUES (1, 'Karim', 1200);
END;
/
```

```
BEGIN
   UPDATE employe
   SET salaire = salaire + 100
   WHERE id = 1;
END;
/
```

```
BEGIN
   DELETE FROM employe
   WHERE id = 1;
END;
/
```

CONDITIONS IF… THEN… ELSE

➢ **Syntaxe**

```
IF condition THEN
   instructions;
ELSIF autre_condition THEN
   instructions;
ELSE
   instructions;
END IF;
```

Exemple:

```
DECLARE
   age NUMBER := 20;
BEGIN
   IF age < 18 THEN
      DBMS_OUTPUT.PUT_LINE('Mineur');
   ELSIF age = 18 THEN
      DBMS_OUTPUT.PUT_LINE('Juste 18 ans');
   ELSE
      DBMS_OUTPUT.PUT_LINE('Adulte');
   END IF;
END;
/
```

STRUCTURE CONDITIONNELLE : CASE

- Test de plusieurs conditions
- Différences avec IF-THEN-ELSE: Test de plusieurs valeurs en une seule construction
- **Syntaxe**

```
CASE <variable>
WHEN <expression 1> THEN <valeur 1>
WHEN <expression 2> THEN <valeur 2>
...                      THEN <valeur n>
END;
```

- **Exemple:** Affichage du club de football en fonction du nom de la ville (Grenoble – GF38 , Marseille – OM, ..., autre – pas d'équipe) :

```
val:=CASE ville
WHEN 'Grenoble' THEN 'GF38'
...
ELSE 'Pas d'équipe'
END;
dbms_output.put_line(val);
```

STRUCTURE ITéRATIVE : LOOP

- Exécution à plusieurs reprises d'un groupe d'instructions

- **Syntaxe**

```
LOOP
  <instruction1>;
    ...
END LOOP;
```

- **Exemple:** Incrémenter jusqu'à la valeur 10 un nombre initialisé à 0

```
val:=0;
LOOP
val:=val+1;
  IF (val=10) THEN
EXIT;
 END IF;
END LOOP;
```

LES BOUCLES - BOUCLE FOR

➢ **Syntaxe**

```
FOR variable IN debut..fin LOOP
   instructions;
END LOOP;
```

Exemple:

```
BEGIN
   FOR i IN 1..5 LOOP
      DBMS_OUTPUT.PUT_LINE('i = ' || i);
   END LOOP;
END;
/
```

LES BOUCLES - BOUCLE WHILE

➢ **Syntaxe**

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

Exemple:

```
DECLARE
   x NUMBER := 1;
BEGIN
   WHILE x <= 5 LOOP
      DBMS_OUTPUT.PUT_LINE(x);
      x := x + 1;
   END LOOP;
END;
/
```

LES CURSEURS

- Un curseur est un mécanisme PL/SQL qui permet de traiter plusieurs lignes renvoyées par une requête SQL.
  - Le SELECT… INTO ne fonctionne **que si une seule ligne** est retournée.
  - Un curseur permet de **lire ligne par ligne**, comme si on parcourait une table avec un doigt.
- Deux types de curseurs:
  - Curseur implicite
  - Curseur explicite

LES CURSEURS : CURSEUR IMPLICITE

- C'est Oracle qui le crée automatiquement pour toute commande SQL (SELECT, INSERT, UPDATE…).
- Il fournit des informations utiles via les attributs :
  - SQL%ROWCOUNT : nombre de lignes touchées (Combien de lignes Oracle a modifiées ?)
  - SQL%FOUND : C'est VRAI si au moins 1 ligne a été trouvée/modifiée (Est-ce que l'opération a trouvé quelque chose ?)
  - SQL%NOTFOUND : C'est le contraire de FOUND (Est-ce que l'opération n'a rien trouvé ?)
  - SQL%ISOPEN : toujours FALSE pour implicite
- Exemple

```
BEGIN
   UPDATE employe SET salaire = salaire + 100;

   DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' lignes modifiées');
END;
/
```

LES CURSEURS : CURSEUR EXPLICITE

- Créé par le programmeur → utilisé pour parcourir plusieurs lignes.

- Un curseur explicite s'utilise en **4 étapes** :

1. Déclaration du curseur

```
CURSOR nom_curseur IS
   SELECT ... FROM ... WHERE ...;
```

2. Ouverture

```
OPEN nom_curseur;
```

3. Lecture ligne par ligne

```
FETCH nom_curseur INTO variable1, variable2, ...;
```

4. Fermeture

```
CLOSE nom_curseur;
```

LES CURSEURS : CURSEUR EXPLICITE

- Exemple : afficher les noms et salaires de tous les employés

```
DECLARE
   -- 1. Déclaration du curseur
   CURSOR c_emp IS
      SELECT nom, salaire FROM employe;

   -- Variables pour stocker chaque ligne lue
   v_nom employe.nom%TYPE;
   v_sal employe.salaire%TYPE;

BEGIN
   -- 2. Ouvrir le curseur
   OPEN c_emp;

   LOOP
      -- 3. Lire une ligne dans les variables
      FETCH c_emp INTO v_nom, v_sal;

      -- Sortir si plus de lignes
      EXIT WHEN c_emp%NOTFOUND;

      -- 4. Utiliser les valeurs
      DBMS_OUTPUT.PUT_LINE(v_nom || ' : ' || v_sal);
   END LOOP;

   -- 5. Fermer le curseur
   CLOSE c_emp;
END;
/
```

GESTION DES EXCEPTIONS

- Une exception en PL/SQL, c'est une erreur (ou un cas particulier) qui survient à l'exécution : par exemple SELECT INTO qui ne retourne rien, une division par zéro, ou une violation d'intégrité.
- Quand une exception survient, PL/SQL interrompt le flot normal et cherche un gestionnaire d'exception (EXCEPTION).
- **Structure**

```
BEGIN
  -- instructions normales
EXCEPTION
  WHEN exception1 THEN
    -- gestion de exception1
  WHEN exception2 THEN
    -- gestion de exception2
  WHEN OTHERS THEN
    -- gestion pour toutes les autres exceptions
END;
/
```

GESTION DES EXCEPTIONS

➢ **Exemple**

```
DECLARE
   v_nom employe.nom%TYPE;
BEGIN
   SELECT nom INTO v_nom
   FROM employe
   WHERE id = 999;        -- n'existe pas

   DBMS_OUTPUT.PUT_LINE(v_nom);

EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Aucun employé trouvé');
   WHEN TOO_MANY_ROWS THEN
      DBMS_OUTPUT.PUT_LINE('Plus d'une ligne retournée');
END;
/
```

GESTION DES EXCEPTIONS

➢ **Exemple 1 :** SELECT INTO + NO_DATA_FOUND

```
DECLARE
  v_nom employe.nom%TYPE;
BEGIN
  SELECT nom INTO v_nom FROM employe WHERE id = 999; -- peut ne rien
retourner
  DBMS_OUTPUT.PUT_LINE('Nom: ' || v_nom);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Aucun employé avec cet ID (NO_DATA_FOUND).');
END;
/
```

➢ si la requête ne retourne rien, on tombe dans WHEN NO_DATA_FOUND et on gère proprement au lieu de laisser l'erreur remonter.

GESTION DES EXCEPTIONS

➢ **Exemple 2 :** Bloc imbriqué et re-raising

```
BEGIN
  BEGIN
    -- partie qui peut échouer
    SELECT salaire INTO v_sal FROM employe WHERE id = 10;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Bloc interne: pas d'employé 10, on ré-élève');
      RAISE; -- remonter l'exception au bloc externe
  END;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Bloc externe: traitement alternatif lorsque employé
manquant.');
END;
/
```

EXERCICE 1

- On dispose d'une table appelée EMP_TEMP :

```
CREATE TABLE emp_temp (
   id       NUMBER,
   nom      VARCHAR2(30),
   salaire  NUMBER
);
```

- Écrire un bloc PL/SQL qui :Insère dans EMP_TEMP 10 employés avec :
  - id allant de 1 à 10
  - nom = 'Employe' + numéro
  - salaire = 1000 + (numéro * 100)
- Ensuite, mettre à jour les employés dont l'ID est pair en augmentant leur salaire de 10%.

EXERCICE 1 - CORRECTION

```
DECLARE
   v_salaire NUMBER;
BEGIN
   ----------------------------------------------------------------
   1) Insérer 10 employés grâce à une boucle FOR
   ----------------------------------------------------------------
   FOR i IN 1..10 LOOP

      -- Calculer le salaire selon la règle
      v_salaire := 1000 + (i * 100);

      INSERT INTO emp_temp (id, nom, salaire)
      VALUES (i, 'Employe' || i, v_salaire);

   END LOOP;
```

```
   ----------------------------------------------------------------
2) Mise à jour des employés ayant un ID pair
   ----------------------------------------------------------------
FOR i IN 1..10 LOOP

      -- Tester si l'ID est pair (modulo = reste)
      IF MOD(i, 2) = 0 THEN
         UPDATE emp_temp
         SET salaire = salaire * 1.10      -- +10%
         WHERE id = i;
      END IF;

   END LOOP;

   COMMIT;  -- Valider les modifications
END;
/
```

EXERCICE 2

➢On a une table EMPLOYE(id, nom, salaire).

Écrire un bloc PL/SQL qui :

1) Lit tous les employés grâce à un curseur

2) Affiche pour chaque employé : Nom — Salaire

3) Gère l'exception si la table est vide.

EXERCICE 3

➢On a une table EMPLOYE(id, nom, salaire).

Écrire un bloc PL/SQL qui :

1. Parcourt tous les employés du département 'Ventes' en utilisant :
   - un curseur explicite
   - une boucle WHILE
2. Pour chaque employé :
   - Si son salaire est < 1500 → l'augmenter de 20%
   - Sinon → l'augmenter de 10%
3. Insérer dans une table LOG_SALAIRES un enregistrement décrivant la modification.
4. Gérer toutes les exceptions:NO_DATA_FOUND,TOO_MANY_ROWS, OTHERS

```
CREATE TABLE employes (
   id        NUMBER PRIMARY KEY,
   nom       VARCHAR2(30),
   salaire   NUMBER,
   departement VARCHAR2(20)
);
```

```
CREATE TABLE log_salaires (
   emp_id   NUMBER,
   ancien_salaire NUMBER,
   nouveau_salaire NUMBER,
   date_modif DATE
);
```

EXERCICE 4

➢un bloc PL/SQL qui :

1. Demande un ID d'employé (par variable)
2. Récupère son nom et salaire avec SELECT… INTO
3. Affiche :Employé : xxx — Salaire : yyy
4. Si l'ID n'existe pas → afficher un message personnalisé

EXERCICE 4 - CORRECTION

```
DECLARE
   v_id NUMBER := 20;
   v_nom employe.nom%TYPE;
   v_sal employe.salaire%TYPE;

BEGIN
   SELECT nom, salaire INTO v_nom, v_sal
   FROM employe
   WHERE id = v_id;

   DBMS_OUTPUT.PUT_LINE('Employé : ' ||
v_nom || ' — Salaire : ' || v_sal);
```

```
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Aucun employé
trouvé avec cet ID.');
   WHEN TOO_MANY_ROWS THEN
      DBMS_OUTPUT.PUT_LINE('Plus d'un
employé possède cet ID.');
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Erreur : ' ||
SQLERRM);
END;
```