جامعة عبد المالك السعدي
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵄⴰⴱⴷⵍⵎⴰⵍⵉⴽ ⵙⵙⴰⵄⴷⵉ
Université Abdelmalek Essaadi

Ecole Nationale des sciences appliquées Al-Hoceima
Université Abdelmalek Essaadi

ENSAH

PL/SQL

ID1-S2

2021/2022

Mohamed CHERRADI

ORACLE

Plan

- Introduction
- Bloc PL/SQL
- Déclaration des variables
- Structure de contrôle
- Curseurs
- Les exceptions
- Les fonctions et procédures
- Les packages
- Les triggers

Introduction (1/5)

- **Nécessite de PL/SQL:**

✓ PL/SQL: Procedural Language SQL

✓ SQL est un langage non procédural

✓ Parfois les traitements complexes sont difficiles a écrire, si l'on peut pas utiliser des variables et les structures de programmation comme les boucles, les conditions, etc.

✓ Pour cette raison, on doit avoir un langage procédural pour combiner des requêtes SQL avec des variables et des structures de programmation habituelles.

Introduction (2/5)

- **Quelques caractéristique de PL/SQL:**

✓ Extension du langage SQL : des requêtes SQL se combinent avec les structures de contrôles habituelles de la programmation structurée (blocs, boucles, …)

✓ Sa syntaxe ressemble a celle du langage Pascal et Ada

✓ Un programme est constitué de procédures, de fonction, …

✓ En générale, l'échange d'information entre les requêtes SQL et le reste du programme est effectué par des variables

✓ PL/SQL est un langage propriétaire d'Oracle. Il est crée par Oracle et utilisé dans le cadre de bases de données relationnellles.

Introduction (3/5)

- **Quelques caractéristique de PL/SQL (Suite):**

✓ PL/SQL: langage procédural d'Oracle étend SQL

- Instruction spécifique a PL/SQL
- Instructions SQL intégrées dans PL/SQL
  - Instructions du **LRD**: SELECT
  - Instructions de **LMD**: INSERT, UPDATE, DELETE
  - Instructions de **LDD**: CREATE, ALTER, DROP, RENAME, TRUNCATE, ..
  - Instructions du langage de contrôle des transactions (**LCT**): COMMIT, ROLLBACK, SAVEPOINT
  - Fonctions: TO_CHAR, TO_DATE, UPPER, SUBSTR, …
  - ….

Introduction (4/5)

- **Instructions spécifiques a PL/SQL**

- ✓ Définition des variables
- ✓ Traitements conditionnels
- ✓ Traitement répétitifs (ou boucles)
- ✓ Traitement des curseurs
- ✓ Traitement des erreurs
- ✓ …

Introduction (5/5)

- **Base et utilisation de PL/SQL:**

✓ PL/SQL : Langage basé sur les paradigmes de programmation procédurale et structuré

✓ Il est utilisé pour l'écriture des procédures stockées et des déclencheurs (triggers)

✓ On l'utilise aussi pour écrire des fonctions utilisateurs qui peuvent être exploitées dans les requêtes SQL, ainsi que pour des fonctions prédéfinies

✓ On l'utilise aussi dans plusieurs outils d'oracle: Forms, Report, …

Structure d'un programme (1/2)

- **Unité de base : blocs**

- ✓ PL/SQL: n'interprète pas une commande mais un ensemble de commande contenues dans un bloc PL/SQL
- ✓ En général, un programme est organisé en blocs d'instructions de types:
  - ✓ Procédures anonymes
  - ✓ Procédures nommées
  - ✓ Fonction nommées
- ✓ Un bloc peut contenir plusieurs autres blocs

Structure d'un programme (2/2)

- **Structure d'un bloc**

```
DECLARE  -- Section optionnelle
        -- Définition des variables
BEGIN – Section obligatoire
        -- Implémentation : Les instructions a exécuter
Exception – Section optionnelle
        -- Code de gestion des exceptions
END;
/ -- Obligatoire pour que vous puissiez exécuter le script
```

Seuls BEGIN et END sont obligatoires

Les blocs, comme les instructions, se terminent par un « ; »

- **NB: Le programme peut être:**
  - ✓ Directement tapé sur une ligne de commande
  - ✓ Ecrit dans un fichier puis chargé

Variables (1/12)

- **Variables en PL/SQL**

- Identificateurs oracle:
  - ✓ Comporte 30 caractères au plus
  - ✓ Commence par une lettre
  - ✓ Peut contenir des lettres, des chiffres, _, $, #, etc
- Pas sensible a la casse
- Porté habituelle des langages a blocs
- Doit être déclaré avant d'être utilisé

Variables (2/12)

- **Possibilités de placer les commentaires**
  - ✓ **- -** Commentaire sur une seule ligne
  - ✓ **/*** Commentaire sur plusieurs lignes ***/**

- **Types de variables:**
  - ✓ **Type habituels** qui correspondent aux types SQL ou Oracle : Intéger, Number, Char, Varchar2, …
  - ✓ **Type composites** qui s'adaptent a la récupération des lignes, des colonnes et des tables SQL : %TYPE, %ROWTYPE
  - ✓ **Type référence** : REF

Variables (3/12)

- **Déclaration d'une variable**
  - ✓ **Syntaxe:** Identificateur (CONSTANT) type := valeur;

- **Exemple:**
  - ✓ Age integer;
  - ✓ Nom varchar(20);
  - ✓ dateNaissance date;
  - ✓ Ok boolean := true;
  - ✓ …
- **NB:** Déclarations multiples interdites
  - ✓ I, h integer ;

Variables (4/12)

- **Déclaration %TYPE**
  - ✓ Possible de déclarer une variable de même type qu'une colonne de table ou une vue
  - ✓ **Exemple:** nom emp.name%TYPE;

- **Déclaration %ROWTYPE**
  - ✓ Une variable peut contenir toutes les colonnes d'une ligne d'une table
  - ✓ **Exemple:** employe emp%ROWTYPE;

Variables (5/12)

- **Exemple**
- Considérant la table:

  EMP(empno, ename, fonction, mgr, dateEmbauche, sal, comm, deptno)

- **Manipulation des variables composites:**

```
employe emp%ROWTYPE;
nom emp.ename%TYPE;
SELECT INTO employe FROM emp WHERE ename='Mcherradi';
nom := employe.ename;
Employe.deptno := 20;
…
INSERT INTO emp VALUES employe;
```

Variables (6/12)

- **Type RECORD**
- Equivalent a STRUCT du langage C.

- **Syntaxe:**

```
TYPE nomRecord IS RECORD (
        Champ1 type1,
        Champ2 type2,
        Champ3 type3,
        …
);
```

Variables (7/12)

- **Exemple (Type RECORD)**

```
DECLARE
      TYPE enreg IS RECORD (
            Num  emp.empno%TYPE,
            Name  emp.ename%TYPE,
             Job  emp.job%TYPE
       );
      R_EMP  enreg; -- variable record de type enreg

BEGIN
      R_EMP.num :=1;
      R_EMP.nom := 'CHERRADI';
      R_EMP.job := 'Ingénieur';
END;
```

Variables (8/12)

- **Affectation**
- Plusieurs façons de donner une valeur a une variable:
  - ✓ Opérateur d'affectation (**:=**)
  - ✓ Directive **INTO** de la requête SELECT.

- **Exemple**
  - ✓ Date_embauche := '24/12/2019';
  - ✓ SELECT ename INTO nom FROM emp WHERE empno=3;

- **NB:**
  - ✓ SELECT ne renvoie qu'une seule ligne,
  - ✓ Dans Oracle, il n'est pas possible d'inclure la clause SELECT sans INTO dans une procédure
  - ✓ Pour renvoyer plusieurs lignes, on va utiliser les curseurs.

Variables (9/12)

- **Problème des conflits de noms**
- Si une variable porte le même nom qu'une colonne d'une table c'est la colonne qui l'emporte

- **Exemple**

```
DECLARE
        ename varchar(30) := 'Mohamed';
BEGIN
        -- emp contient un champ ename
        DELETE FROM emp WHERE ename='Ali';
END;
/
```

- **NB:** Pour éviter les conflits de nommages, préfixer les variables PL/SQL par v_.

Variables (10/12)

- **Affichage**

- ✓ Pour plus de clarté, il est utile d'afficher les valeurs des variables
- ✓ Activer le retour écran: **SET SERVEROUTPUT ON;**
- ✓ Sortie standard, le paquetage : **DBMS_OUTPUT('…' || …);**
- ✓ Un paquetage est un regroupement de procédures et de fonctions
- ✓ Concaténation de chaines : Opérateur ||

Variables (11/12)

- **Exemple 1:**

```
SET SERVEROUTPUT ON;
DECLARE
        a number;
BEGIN
        a := 10;
        DBMS_OUTPUT.PUT_LINE(‘ la valeur de a est : ‘ || a);
END;
/
```

Variables (12/12)

- **Exemple 2:**

```
SET SERVEROUTPUT ON;
DECLARE
        nb number;
BEGIN
        DELETE FROM emp WHERE ename='mohamed';
        nb := sql%rowcount; -- curseur sql explicite
        DBMS_OUTPUT.PUT_LINE(' nb= ' || nb);
END;
/
```

- **Résultat:** retourne le nombre d'enregistrement supprimé dans le champ nom est 'mohamed'.

Structure de controle (1/9)

```
IF condition THEN
        instructions1
END IF;
```

```
IF condition THEN
        instructions1
ELSE
        instruction2
END IF;
```

```
IF condition1 THEN
        instruction 1;
ELSIF condition2 THEN
        instruction 2;
ELSIF …
        …
ELSE
        instruction n;
END IF;
```

Structure de controle (2/9)

- **Lecture d'une variable**
  - ✓ ACCEPT n number PROMPT 'Veuillez saisir la valeur de n'
- **Exemple:**

```
ACCEPT n number PROMPT 'Veuillez saisir la valeur de n'
SET SERVEROUTPUT ON;
DECLARE
        n number;
BEGIN
        n := 3;
        IF (n>0) THEN
                DBMS_OUTPUT.PUT_LINE(' n est strictement positif');
        ELSIF(n=0) THEN
                DBMS_OUTPUT.PUT_LINE(' n est null');
        ELSE
                DBMS_OUTPUT.PUT_LINE(' n est null');
        END IF;
END;
/
```

Structure de controle (3/9)

- **Choix**

```
CASE expression
        WHEN expr1 THEN instruction1;
        WHEN expr2 THEN instruction2;
        …
        ELSE instruction;
END CASE;
```

- **NB**: expression peut avoir n'importe quel type simple (ne peut pas par exemple être ***record***)

Structure de controle (4/9)

- **Exemple:**

```
SET SERVEROUTPUT ON
DECLARE
        n integer;
BEGIN
        n := 3;
CASE n
        WHEN 1 THEN DBMS_OUTPUT.PUT_LINE('Lundi');
        WHEN 2 THEN DBMS_OUTPUT.PUT_LINE('Mardi');
        WHEN 3 THEN DBMS_OUTPUT.PUT_LINE('Mercredi');
        WHEN 4 THEN DBMS_OUTPUT.PUT_LINE('Jeudi');
        WHEN 5 THEN DBMS_OUTPUT.PUT_LINE('Vendredi');
        WHEN 6 THEN DBMS_OUTPUT.PUT_LINE('Samedi');
        ELSE DBMS_OUTPUT.PUT_LINE('Dimanche');
END CASE;
END;
/
```

Structure de controle (5/9)

- **Boucle « Tant que »:**

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

Structure de controle (6/9)

- **Exemple (Calcule de la moyenne de 10 entiers):**

```
SET SERVEROUTPUT ON
DECLARE
        compteur number(2);
        somme number(2) := 0;
        moyenne number(3, 1);
BEGIN
        compteur := 1;
        WHILE compteur <= 10 LOOP
                somme := somme + compteur;
                compteur := compteur + 1;
        END LOOP;
        moyenne := somme / 10;
        DBMS_OUTPUT.PUT_LINE('La moyenne est : ' || moyenne);
END;
/
```

Structure de controle (7/9)

- **Boucle « Faire … Tant que »:**

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

Structure de controle (8/9)

- **Exemple (Calcule de la moyenne de 10 entiers):**

```
SET SERVEROUTPUT ON
DECLARE
        compteur number(2);
        somme number(2) := 0;
        moyenne number(3, 1);
BEGIN
        compteur := 1;
        LOOP
                somme := somme + compteur;
                compteur := compteur + 1;
        EXIT WHEN compteur > 10;
        END LOOP;
        moyenne := somme / 10;
        DBMS_OUTPUT.PUT_LINE('La moyenne est : ' || moyenne);
END;
/
```

Structure de controle (9/9)

- **Boucle « Pour »:**

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

- **Exemple:**

```
For i IN 1..100 LOOP
        somme := somme + i;
END LOOP;
DBMS_OUTPUT.PUT_LINE('La somme de 1 a 100 est : ' || somme);
```

Interactions simples avec la base (1/4)

- **Extraire des données – erreurs :**

✓ Si la clause **SELECT** renvoie plus d'une ligne, une exception « **TOO_MANY_ROWS » (ORA-01422)** est levée. Voir la suite du cours sur les exceptions.

✓ Si la clause SELECT ne renvoie aucune ligne, une exception « **NO_DATA_FOUND » (ORA-01403)** est levée.

Interactions simples avec la base (2/4)

- **Exemple :**

```
SET SERVEROUTPUT ON
DECLARE
        v_nom  emp.ename%TYPE;
        v_emp  emp%ROWTYPE;
BEGIN
        SELECT nom INTO v_nom FROM emp WHERE matr = 7844;
        SELECT * INTO v_emp FROM emp WHERE matr = 7844;
        DBMS_OUTPUT.PUT_LINE('l employee trouve s appel:'|| v_nom);
END;
/
```

Interactions simples avec la base (3/4)

- **Modification de données :**

✓ Les requêtes SQL (insert, update, delete,…) peuvent utiliser les variables PL/SQL

✓ Les commit et rollback doivent être explicites; aucun n'est effectué automatiquement à la sortie d'un bloc

✓ Voyons plus de détails pour l'insertion de données;

Interactions simples avec la base (4/4)

- **Exemple d'insertion:**

```
SET SERVEROUTPUT ON
DECLARE
        v_emp  emp%ROWTYPE;
        v_nom  emp.ename%TYPE;
BEGIN
        v_nom := 'MOHAMED';
        INSERT INTO emp (empno, ename) VALUES (600, v_nom);
        v_emp.empno := 610;
        v_emp.ename := 'AHMED';
        INSERT INTO emp (empno, ename) VALUES (v_emp.empno, v_emp.ename);
        commit;

END;
/
```

Curseurs (1/21)

- **Définition**

✓ **Curseur:** C’est une variable spéciale qui pointe sur le résultat d’une requête SQL. La déclaration d’un curseur est liée au texte de la requête.

✓ **Curseur:** zone de mémoire de taille fixe, utilisée par le noyau d'Oracle pour analyser et interpréter tout ordre SQL.

Curseurs (2/21)

- **Types de curseur:**

✓ Il existe deux types de curseurs: **Implicite** & **Explicite**

✓ **Implicite** : créés et gérés par Oracle à chaque ordre SQL (lorsque la close INTO accompagne le SELECT).

✓ **Explicite** : créés et gérés par le programmeur afin de pouvoir traiter un SELECT qui retourne plusieurs tuples.

Curseurs (3/21)

- **NB:**

✓ Pour utiliser un curseur explicite, on doit passer par les étapes suivantes :

1. Déclaration du curseur
2. Ouverture du curseur
3. Traitement des lignes du résultat
4. Fermeture du curseur

Curseurs (4/21)

- **Déclaration du curseur:**

✓ Association d'un nom de curseur à une requête SELECT;
✓ Se fait dans la section DECLARE d'un bloc PL/SQL;

```
DECLARE
        CURSOR nom_curseur IS Requête_Select ;
```

- **Exemple:**

```
DECLARE
        CURSOR checkValidation IS
        SELECT nom, note FROM etds WHERE note >12;
```

Curseurs (5/21)

- **Ouverture du curseur:**

✓ Alloue un espace mémoire au curseur et positionne les éventuels verrous

✓ Après avoir déclaré le curseur, il faut l'ouvrir dans la section exécutable **BEGIN**.

✓ L'ouverture du curseur amorce l'analyse de SELECT et son exécution.

✓ La réponse est calculée et rangée dans un espace temporaire:

OPEN nom_curseur ;

- **Exemple:**

```
DECLARE
        CURSOR checkValidation IS
        SELECT nom, note FROM etds WHERE note >12;
BEGIN
        OPEN checkValidation;
END;
/
```

Curseurs (6/21)

- **Traitement des lignes:**

✓ L'accès aux données se fait par la clause : **FETCH INTO**

✓ **FETCH** nom_curseur **INTO** var1, var2, …

✓ **FETCH** permettant de récupérer une ligne de l'ensemble des lignes associés au curseur et stocker les valeurs dans des variables réceptrices

✓ Pour traiter plusieurs tuples, il faut utiliser une boucle.

Fermeture du curseur:

- Après le traitement des lignes pour libérer la place mémoire, il faut fermer le curseur: **CLOSE** nom_curseur ;

Curseurs (7/21)

- **Exemple 1:**

```
DECLARE
        CURSOR checkValidation IS
                SELECT nom, note FROM etds WHERE note >12;
        nomEtd  etds.nom%TYPE;
        noteEtd etds.note%TYPE;
BEGIN
        OPEN checkValidation;
                - - On récupère le premier enregistrement
                FETCH checkValidation INTO nomEtd, noteEtd ;
                - - S'il y a un enregistrement récupéré
                WHILE checkValidation %found LOOP
                        - - Traitement de l'enregistrement récupéré
                        FETCH checkValidation INTO nomEtd, noteEtd ; - - On récupère l'enregistrement suivant
                END LOOP;
        CLOSE checkValidation;
END;
/
```

Curseurs (8/21)

- **Exemple 2:** (création et remplissage d'une table 'mention' à partir de la table etudiant)

```
- - Création de la table
CREATE TABLE mention(nom varchar2(10), notegenerale number(7,2));

DECLARE
        CURSOR checkValidation IS SELECT nom, note FROM etds;
        nom  etds.nom%TYPE; noteGlob etds.note%TYPE;
BEGIN
        OPEN checkValidation;
                FETCH checkValidation INTO nom, noteGlob;
                WHILE checkValidation %found LOOP
                        IF noteGlob > 12 THEN
                                INSERT INTO mention VALUES (nom, notegenerale) ;
                         END IF ;
                        FETCH checkValidation INTO nom, noteGlob;
                END LOOP;
        CLOSE checkValidation;
END;
/
```

Curseurs (9/21)

- **Statut d'un curseur (Attribut):**

✓ Les attributs d'un curseur sont des indicateurs sur son état.

✓ Quatre attributs permettent d'évaluer l'état du curseur:

- **%Found** : vrai si le dernier **FETCH** a ramené un tuple.
- **%NotFound** : vrai si le dernier FETCH n'a ramené aucun tuple.
- **%RowCount** : compte le nombre de FETCH exécutés sur un curseur;
- **%IsOpen** : vrai si le curseur est ouvert;

Curseurs (10/21)

- **Remarque:**

✓ Avec un curseur implicite créé par le SGBD Oracle, les mêmes attributs aux colonnes sont disponibles à condition de les préfixer par SQL. Ces attributs référent au dernier curseur implicite utilisé par l'application.

Curseur Implicite	Curseur Explicite
SQL**%FOUND**	nom-curseur**%FOUND**
SQL**%NOTFOUND**	nom-curseur**%NOTFOUND**
SQL**%ISOPEN**	nom-curseur**%ISOPEN**
SQL**%ROWCOUNT**	nom-curseur**%ROWCOUNT**

Curseurs (11/21)

- **L'attribut de curseur %ROWTYPE:**

✓ L'attribut %ROWTYPE permet de déclarer une variable de même type que l'enregistrement de la table

✓ **Syntaxe** : Nom_de_variable nom_table%ROWTYPE ;

Exemple 1:

✓ **DECLARE** enrg_emp emp**%ROWTYPE;**

✓ Avec un curseur :

CURSOR nom_curseur **IS** Requete_SELECT ;

nom_variable nom_curseur**%ROWTYPE** ;

✓ Les éléments de la structure (nom_variable) sont identifiés par :

nom_variable.nom_colonne

✓ La structure est renseignée par le FETCH :

FETCH nom_curseur **INTO** nom_variable

Curseurs (12/21)

- **Utilisation simplifiée des curseurs :**

✓ L'utilisation FOR LOOP **remplace** OPEN, FETCH et CLOSE.

✓ Lorsque le curseur est invoqué, un enregistrement est automatiquement créé avec les mêmes éléments de données que ceux définies dans SELECT.

Exemple : (création et remplissage d'une table 'mention' à partir de la table etudiant)

```
CREATE TABLE mention(nom varchar2(10), notegenerale number(7,2));
DECLARE
        CURSOR checkValidation IS SELECT nom, note FROM etds;
BEGIN
        FOR enreg_e IN checkValidation LOOP
                IF enreg_e.noteGlob > 12 THEN
                        INSERT INTO resultat VALUES (enreg_e.nom, enreg_e.notegenerale) ;
                 END IF;
        END LOOP;
END;
```

Curseurs (13/21)

- **Exercice :**

✓ Ecrire un bloc PL/SQL permettent de:

- Créer la tables Emp2 à partir de la table Emp.
- Utiliser un curseur pour sélectionner toutes les colonnes de la table Emp.
- Parcourir ce curseur afin d'insérer dans Emp2 les employés gagnant plus que 2000$.
- Afficher le nombre des employés dans la table Emp2.

Curseurs (14/21)

```
CREATE TABLE emp2 AS select * FROM emp;
TRUNCATE TABLE emp2;
SET SERVEROUTPUT ON
DECLARE
        CURSOR emp_cur IS SELECT * FROM EMP ;
        ligne emp_cur%rowtype; etudiant_count NUMBER(5) := 0;
BEGIN
        OPEN emp_cur;
        FETCH emp_cur INTO ligne ;
        WHILE emp_cur%FOUND LOOP
                IF ligne.sal>2000 THEN
                    etudiant_count := etudiant_count+1;
                    INSERT INTO emp2 VALUES
                    (ligne.empno,ligne.ename,ligne.job,ligne.mgr,
                    ligne.hiredate,ligne.sal,ligne.comm,ligne.deptno);;
                END IF;
                FETCH emp_cur INTO ligne ;
         END LOOP ;
DBMS_OUTPUT.PUT_LINE ('Le nombre total des employes est '|| emp_cur%ROWCOUNT);
DBMS_OUTPUT.PUT_LINE ('le nombre des employes ayant le salaire plus que 2000:' || etudiant_count );
CLOSE emp_cur;
END;/
SELECT ename,sal FROM emp2;
```

Curseurs (15/21)

- **Utilisation d'une variable de type *Record* :**

✓ Un record permet de définir des types composites.

```
DECLARE
      TYPE enreg IS RECORD (
            v1 TYPE1,
            v2 TYPE2
       );
```

✓ **Exemple:** Déclaration d'une variable de ce type : nom_variable nom_record

```
DECLARE
      TYPE enrg_emp IS RECORD (
            nom emp.ename %TYPE,
            salaire emp.sal %TYPE
       );
e enrg_empl
```

Curseurs (16/21)

- **Exemple:**

```
DECLARE
      TYPE etudiant IS RECORD (
            cne NUMBER,
            nom VARCHAR2(10),
            prenom VARCHAR2(10),
            date_naiss DATE,
            sexe VARCHAR2(10)
       );
et  etudiant;
BEGIN
            et.cne := 12124512;
            et.nom := CHERRADI';
            et.prenom := ‘Mohamed';
            et.date_naiss := '12/12/1984';
            et.sexe := 'MASCULUN';
            DBMS_OUTPUT.PUT_LINE ( 'L etudiant de nom:'||et.nom ||'et
            prenom:'||et.prenom||'de CNE:'|| et.cne||'Ne le:'||et.date_naiss||'de sexe:'||et.sexe);
END;
/
```

Curseurs (17/21)

- **Modification des données:**

✓ La modification des données se fait habituellement avec **INSERT**, **UPDATE** ou **DELETE**, mais on peut utiliser: **FOR UPDATE** dans la déclaration du curseur.

- **Objectif du curseur modifiable:**

✓ Modification via SQL sur le n-uplet courant

✓ Permet d'accéder directement en modification ou en suppression du nuplet récupéré par **FETCH**.

Curseurs (18/21)

- **Remarque:**

✓ Un curseur qui comprend **plus** d'une table dans sa définition **ne permet pas la modification des tables de BD.**

✓ Seuls les curseurs définis sur **une seule table sans fonction d'agrégation et de regroupement** peuvent utilisés dans une MAJ : delete, update, insert avec le **CURRENT OF CURSOR**.

- **Syntaxe:**

```
CURSOR nomCurseur IS ….. FOR UPDATE;
    - - déclarations des variables et opérations
WHERE CURRENT OF nomCurseur;
```

Curseurs (19/21)

- **Exercice :**

✓ Créer un curseur qui permet de supprimer tous les champ NULL de la colonne COMM dans la table EMP, et mettre le résultat dans une table EMP_COMM,

Curseurs (20/21)

```
CREATE TABLE EMP_COMM AS SELECT ename,sal,comm FROM EMP;
SET SERVEROUTPUT ON
DECLARE
        CURSOR mce IS SELECT ename, sal, comm FROM EMP FOR UPDATE;
        nom EMP_COMM.ename%TYPE ; salaire EMP_COMM.sal%TYPE ;
        commission EMP_COMM.comm%TYPE ;
BEGIN
        OPEN mce;
        FETCH mce INTO nom, salaire, commision;
        WHILE mce%FOUND LOOP
                IF commision IS NULL THEN
                    DELETE FROM EMP_COMM WHERE CURRENT OF mce ;
                END IF;
                FETCH mce INTO nom, salaire, commision;
         END LOOP ;
         CLOSE emp_cur;
END;
/
```

Curseurs (21/21)

- **Exercice :**

✓ Créer une table emp_manager_2000 a partir de la table Emp,

✓ Ecrire un bloc PL/SQL permettant de :

- De calculer le nombre des employés ayant un salaire moins de 2000$,
- Utiliser un curseur pour sélectionner toutes les colonnes des employés de la table Emp qui ont un salaire entre <2000$.
- Afficher le nombre de tous les employés ayant le salaire <2000$,