ORACLE

Cours de PL/SQL

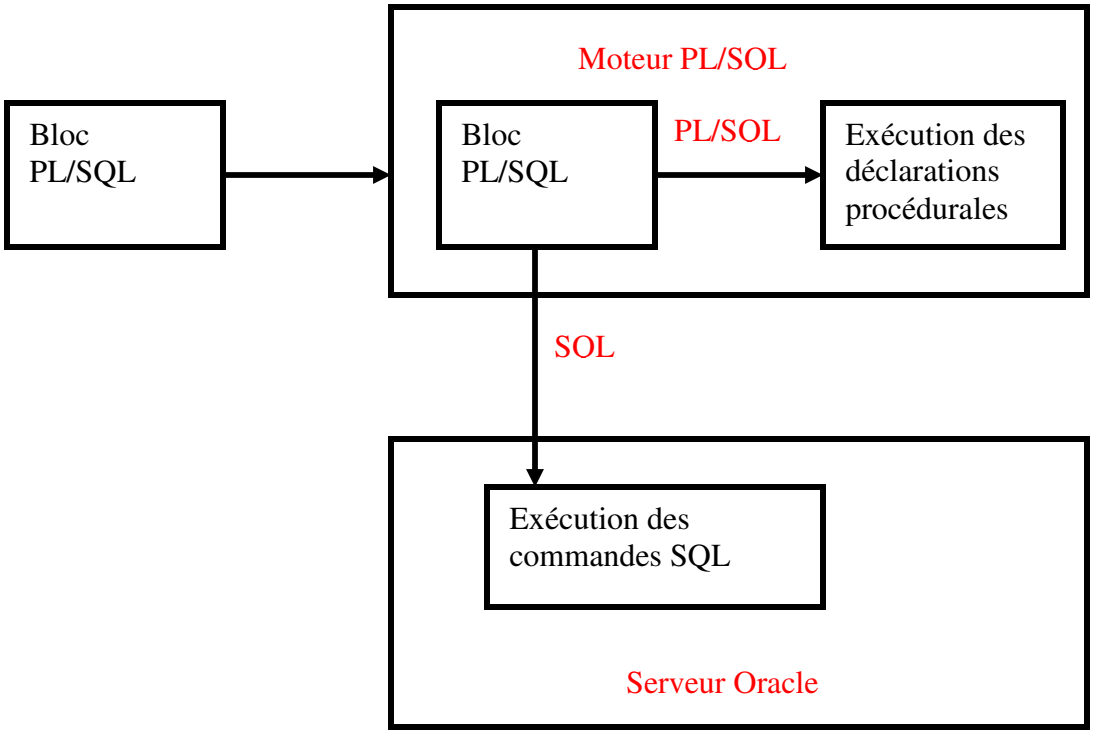
TABLE DES MATIÈRES

1. Introduction

PL/SQL :

- Procedural Language / SQL
- C'est un langage procédural
- C'est une extension de SQL

Environnement :



Avantages de PL/SQL :

- PL/SQL joue un rôle central entre le serveur Oracle (à travers les procédures, les fonctions, les packages, les déclencheurs de la base de données, etc.) et les différents outils de développement d'Oracle.
- PL/SQL peut regrouper un ensemble de requêtes SQL en un seul bloc et les envoie au serveur en une seule fois, ce qui réduit le trafic au niveau du serveur et augmente la performance.
- PL/SQL peut être utilisé par les outils d'Oracle (Forms, Reports, Designer2000,etc.)

PL/SQL = SQL +

- Variables et constantes
- Traitements conditionnels
- Traitements itératifs
- Curseurs en PL/SQL
- Les enregistrements et les tables PL/SQL
- Procédures
- Fonctions
- Les packages en Oracle
- Le traitement des exceptions (erreurs)

Bloc PL/SQL :

DECLARE
    Variables;
    Constantes;
    Curseurs;
    Tables PL/SQL;
    Enregistrements PL/SQL;
    Exceptions;
BEGIN
    Instructions SQL et PL/SQL
EXCEPTION
    Traitement des exceptions (gestion des erreurs)
END

N.B. :

- Chaque instruction se termine par ';'
- Les parties 'DECLARE' et 'EXCEPTION' sont facultatives
- On peut inclure des commentaires dans un bloc PL/SQL
    -- commentaires sur une seule ligne
    /* ……*/ commentaires sur plusieurs lignes

2. Les variables

Elles servent à :

- stocker temporairement des données
- manipuler des valeurs stockées
- elles sont réutilisables
- elles sont faciles à maintenir

Types de variables :

a. variables de type Oracle
b. variables booléennes
c. variables faisant référence au dictionnaire de données

N.B. :

- Le nom des variables ne doit pas être le même que celui d'une colonne ou d'une table de la base de données.
- Pour affecter une valeur à une variable, on utilise :
  :=
  Default
  Not null

  Exemples : v_sal := 2500;
  v_Mgr number(4) Default 7839;
  v_loc varchar(13) not null := 'MONTRÉAL';
- Il est preferable de précéder le nom des variables par 'v_'.
  Ex. : v_sal, v_nom, etc.

a. Variables de type Oracle :

Char, Varchar2, Number, Date, Long, Long Row

b. Variables booléennes :

Exemple : v_valide BOOLEAN := true;

c. Variables faisant référence au dictionnaire de données.

Exo #1 (a):

Pour la table Dept, créer des variables qui auront le même type et la même dimension que les colonnes de Dept. Leur affecter respectivement les valeurs : 60, 'RHU', 'MONTRÉAL'

NB. : On utilise sous SQL*Plus la commande : **SET SERVEROUTPUT ON** pour pouvoir afficher les résultats de la commande Oracle (Package) : **DBMS_OUTPUT.PUT_LINE(chaîne)**

Réponse (a)

```
DECLARE
        v_deptno number(2) := 60;
        v_dname varchar2(14) := 'RHU';
        v_loc varchar2(13) := 'MONTRÉAL';
BEGIN
        DBMS_OUTPUT.PUT_LINE('No dept : '||v_deptno);
        DBMS_OUTPUT.PUT_LINE('Nom dept : '||v_dname);
        DBMS_OUTPUT.PUT_LINE('Loc : '||v_loc);
END;
```

N.B. :

Pour affecter les colonnes d'une ligne d'une table :

SELECT colonne **INTO** v_col
FROM nom de table
WHERE condition(s);

Exo #2 :

Déclarer les variables v_deptno, v_dname, v_loc correspondantes aux colonnes de Dept.
Affecter à ces variables le no, le nom et la localisation du département no 20.
Afficher les valeurs de ces variables.

Réponse :

```
DECLARE
        v_deptno number(2);
        v_dname varchar2(14);
        v_loc varchar2(13);
BEGIN
        SELECT deptno, dname, loc INTO v_deptno, v_dname, v_loc
        FROM DEPT
        WHERE deptno=20;

        DBMS_OUTPUT.PUT_LINE('No dept : '||v_deptno);
        DBMS_OUTPUT.PUT_LINE('Nom dept : '||v_dname);
        DBMS_OUTPUT.PUT_LINE('Loc : '||v_loc);
END;
```

Exo #3 :
Insérer les valeurs 60, 'RHU', 'MONTRÉAL' dans la table DEPT.

Réponse :

```
DECLARE
        v_deptno number(2) := 60;
        v_dname varchar2(14) := 'RHU';
        v_loc varchar2(13) := 'MONTRÉAL';
BEGIN
        INSERT INTO DEPT('deptno, dname, loc)
        VALUES (v_deptno, v_dname, v_loc);
END;
```

- **Exo #4 :** Utilisation de variables SQL*Plus.
- Créer des variables sous SQL*Plus dans lequel on affecte les valeurs 70, 'Finance', 'Québec'
- Affecter ces variables à d,autres variables qu'on va créer dans un bloc PL/SQL.
- Afficher ces variables.

Réponse :

a. Sous SQL*Plus, on crée les variables suivantes :

```
ACCEPT p_dept 70
ACCEPT p_dname Finance
ACCEPT p_loc Québec
```

b. Bloc PL/SQL

```
DECLARE
        v_deptno number(2) := &p_deptno;
        v_dname varchar2(14) :=&p_dname;
        v_loc varchar2(13) := &p_loc;
BEGIN
        DBMS_OUTPUT.PUT_LINE('No dept : '|| v_deptno);
        DBMS_OUTPUT.PUT_LINE('Nom dept : '|| v_dname);
        DBMS_OUTPUT.PUT_LINE('Loc dept : '|| v_loc);
END;
```

c. Variables faisant référence au dictionnaire de données.

c1. Variables de même type qu'une colonne d'une table de BD :

Syntaxe :

Nom_var Table.colonne%type;
Exemple : v_dname DEPT.dname%type;

Exo #1 : Refaire l'ex #1 de la partie précédente avec '%type'

Réponse :

```
DECLARE
        v_deptno Dept.deptno%type := 60;
        v_dname Dept.dname%type := 'RHU';
        v_loc Dept.loc%type := 'MONTRÉAL';
BEGIN
        DBMS_OUTPUT.PUT_LINE('No dept : '||v_deptno);
        DBMS_OUTPUT.PUT_LINE('Nom dept : '||v_dname);
        DBMS_OUTPUT.PUT_LINE('Loc : '||v_loc);
END;
```

Exo #2 :

- Affecter les valeurs des colonnes de la table DEPT dont le no Dept=30 aux variables v_deptno, v_dname, v_loc.
- Afficher les valeurs de ces variables .

Réponse :

```
DECLARE
        v_deptno Dept.deptno%type;
        v_dname Dept.dname%type;
        v_loc Dept.loc%type;
BEGIN
        SELECT * INTO v_deptno, v_dname, v_loc
        FROM DEPT
        WHERE deptno=30;
        DBMS_OUTPUT.PUT_LINE('No dept : '||v_deptno);
        DBMS_OUTPUT.PUT_LINE('Nom dept : '||v_dname);
        DBMS_OUTPUT.PUT_LINE('Loc : '||v_loc);
END;
```

c2. Variables de même structure qu'une ligne d'une table de BD :

Syntaxe :

v_ligne Table%ROWTYPE

exemple : v_employe EMP%ROWTYPE

N.B. :

- La structure ligne contient autant de variables que de colonnes de la table.
- Ces variables portent le même nom et sont de même type que les colonnes de la table.
- Pour y accéder :

v_ligne.nom_colonne

Exemple : v_ligne.ename, v_ligne_sal

Exo :

Affecter la ligne de la table DEPT dont deptno=30 à la variable v_dept. Afficher le contenu de la variable.

Réponse :

```
DECLARE
        v_dept Dept%rowtype;
BEGIN
        SELECT * INTO v_dept
        FROM DEPT
        WHERE deptno=30;

        DBMS_OUTPUT.PUT_LINE('No dept : '||v_dept.deptno);
        DBMS_OUTPUT.PUT_LINE('Nom dept :
        '||v_dept.dname);
        DBMS_OUTPUT.PUT_LINE('Loc : '||v_dept.loc);
END;
```

3. Les traitements inconditionnels

```
IF condition1 THEN
    traitement1;
ELSIF condition2 THEN
    traitement2;
[ELSE
    traitement3;]
ENDIF;
```

- Le opérateurs utilisés dans les conditions sont les mêmes que dans SQL (=, <, >, <=, >=, IS NULL, LIKE, …)
- Dès que l'une des conditions est vraie, le traitement qui suit le THEN est exécuté.
- Si aucune condition n'est vraie, c'est le traitement qui suit le ELSE qui est exécuté.

Exo #1 :

Affecter le nom et le salaire de l'employé 'SMITH' aux variables n_nom, v_sal.
Afficher ces variables, pour v_sal si :

- v_sal < 1000 alors afficher aussi 'Bas salaire'
- v_sal >= 1000 alors afficher aussi 'Haut salaire'

Réponse :

```
DECLARE
    v_nom EMP.ename%type;
    v_sal EMP.sal%type;
BEGIN
    SELECT ename, sal INTO v_nom, v_sal
    FROM EMP
    WHERE ename = 'SMITH';
    DBMS_OUTPUT.PUT_LINE('Nom : 'll v_nom);
    IF v_sal < 1000 THEN
            DBMS_OUTPUT.PUT_LINE('Salaire : 'll v_sal ll' 'll'Bas salaire');
    ELSE
            DBMS_OUTPUT.PUT_LINE('Salaire : 'll v_sal ll' 'll'Haut salaire');
    ENDIF;
END;
```

Exo #2 :
Affecter la commission de l'employé 'FORD' à la variable v_comm.
Si v_comm est > 0 alors afficher 'l'employé a une commission'
Si v_comm est = 0 alors afficher 'Commission égale à zéro'
Si v_comm n'est pas renseignée (IS NULL) alors afficher 'Pas de commission'

Réponse :

```
DECLARE
    v_comm EMP.comm%type;
BEGIN
    SELECT comm INTO v_comm.
    FROM EMP
    WHERE ename = 'FORD';
    IF v_comm. > 0 THEN
            DBMS_OUTPUT.PUT_LINE('L"employé a une commission');
    ELSIF v_comm. = 0 THEN
            DBMS_OUTPUT.PUT_LINE('Commission égale à zéro');
    ELSE
            DBMS_OUTPUT.PUT_LINE('Pas de commission');
    ENDIF;
END;
```

4. Les traitements itératifs

Il existe trois types de boucles :

1. La boucle LOOP (de base)
2. La boucle FOR
3. La boucle WHILE

1. la boucle LOOP

Syntaxe :

```
LOOP
    Expression1;
    Expression2;
        .
        .
        .
EXIT [WHEN condition];
END LOOP;
```

Exo #1:
Initialiser la variable v_nb à 10.
Afficher la valeur de cette variable, puis incrémenter sa valeur de 1 jusqu'à la valeur 15.

Réponse :
```
DECLARE
    v_nb NUMBER :=10;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE(v_nb);
        v_nb := v_nb + 1;
    EXIT WHEN v_nb > 15;
    END LOOP;
END;
```

Exo #2:
Créer une table de multiplication Mult(op char(5), Res char(3)).
Écrire un programme qui fait le calcul et l'affichage de la table Mult.
On demandera à l'utilisateur quelle table de multiplication il souhaite.

Réponse :
```
CREATE TABLE Mult(Op char(5), res char(3));
PROMPT Quelle table voulez-vous?
ACCEPT TABLE
DECLARE
    Compteur NUMBER := 0;
BEGIN
    LOOP
        Compteur := compteur + 1;
        INSERT INTO Mult VALUES(
                                TO_CHAR(compteur)||'*'||TO_CHAR(&TABLE),
                                TO_CHAR(compteur*&TABLE));
    EXIT WHEN compteur = 10;
    END LOOP;
END;
SELECT * FROM Mult;
ROLLBACK;
```

2. la boucle FOR

Exécution d'un traitement un certain nombre de fois, le nombre étant connu.

Syntaxe :

```
BEGIN
    …
    FOR indice IN [REVERSE] exp1..exp2
        LOOP
            Instructions;
    END LOOP;
END;
```

N.B. :

- Inutile de déclarer la variable 'indice'
- Indice varie de exp1 à exp2 avec 1 pas de 1
- Si 'REVERSE' est précisé alors 'indice' varie de exp1 à exp2 avec un pas de –1

Exo #1 : Refaire l'exo précédent avec la boucle FOR

Réponse :

```
BEGIN
    FOR compteur IN 1..10
        LOOP
            INSERT INTO Mult VALUES(
                    TO_CHAR(compteur)||'*'||TO_CHAR(&TABLE),
                    TO_CHAR(compteur*&TABLE);
    END LOOP;
END;
```

Exo #2 : Faites le calcul de factoriel de 5. Afficher le résultat. 5!=5*4*3*2*1

Réponse :

```
DECLARE
    v_fact NUMBER := 1;
BEGIN
    FOR I IN 1..5
        LOOP
            v_fact = v_fact * i;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Factoriel de 5 : '||v_fact);
END;
```

2. la boucle WHILE

Exécution d'un traitement tant qu'une condition est vraie.

Syntaxe :

```
WHILE condition
        LOOP
            Instructions;
END LOOP;
```

Exo : Refaire l'exo #1 (Multiplication) avec la boucle WHILE.

Réponse :

```
DECLARE
    Compteur NUMBER :=0;
BEGIN
    WHILE compteur < 10
        LOOP
            Compteur := compteur + 1;
            INSERT INTO MULT VALUES (
                    TO_CHAR(compteur)||'*'||TO_CHAR(&TABLE),
                    TO_CHAR(compteur*&TABLE);
    END LOOP;
END;
SELECT * FROM MULT;
```

5. Les curseurs en PL/SQL

Il existe deux types de curseurs :

- **Curseurs implicite :** c'est un curseur SQL généré et géré par le noyau d'Oracle pour chaque ordre SQL d'un bloc.
- **Curseur explicite :** c'est un curseur SQL généré et géré par l'utilisateur pour traiter un ordre SELECT qui ramène plus d'une ligne.

Curseur explicite : 4 étapes

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

1. **Déclaration du curseur :** se fait dans la partie DECLARE.

Syntaxe :

```
DECLARE
    CURSOR cur_emp IS
        SELECT empno, ename, sal
        FROM EMP
        WHERE deptno = 10;
        …
BEGIN
    ….
END;
```

2. **Ouverture du curseur :** l'ouverture du curseur lance l'exécution de l'ordre SELECT associé au curseur. L'ouverture se fait dans la section BEGIN du bloc.

Syntaxe :

```
OPEN nom_curseur
```

Exemple :

```
DECLARE
    CURSOR cur_emp IS
        SELECT empno, ename, sal
        FROM EMP
        WHERE deptno = 10;
        …
BEGIN
    OPEN cur_emp;
    ….
END;
```

3. **Traitement de lignes :** Après l'exécution du SELECT, les lignes ramenées sont traitées une par une, la valeur de chaque colonne du SELECT doit être stockée dans une variable réceptrice.

Syntaxe :

```
FETCH cur_emp INTO liste_variables
```

Exemple:

```
DECLARE
    CURSOR cur_emp IS
        SELECT empno,ename, sal
        FROM EMP
        WHERE deptno = 10;
    v_empno EMP.emp%type;
    v_ename EMP.ename%type;
    v_sal EMP.sal%type;
BEGIN
    OPEN cur_emp;
    LOOP
        FETCH cur_emp INTO v_empno, v_ename, v_sal;
```

```
            DBMS_OUTPUT.PUT_LINE('No Emp : '||v_empno);
            DBMS_OUTPUT.PUT_LINE('Nom Emp : '||v_ename);
            DBMS_OUTPUT.PUT_LINE('Sal Emp : '||v_sal);
        END LOOP;
        CLOSE cur_emp; /*partie 4*/
    END;
```

4. **Fermeture du curseur :** Il est important de fermer le curseur une fois qu'on en a plus besoin, cela permet de libérer la mémoire occupée par ce dernier.

Syntaxe :

```
CLOSE nom_curseur; /* voir l'exemple précédent*/
```

Exo #1 :

Afficher le No d'employé, le nom, hiredate de l'employé 'SCOTT'. Utiliser un curseur.

Réponse :

```
DECLARE
    v_empno EMP.empno%type;
    v_nom EMP.ename%type;
    v_date EMP.hiredate%type;
    CURSOR cur_scott IS
        SELECT empno, ename, hiredate
        FROM EMP
        WHERE ename = 'SCOTT';
BEGIN
    OPEN cur_scott;
    FETCH cur_scott INTO v_empno, v_nom, v_date;
    CLOSE cur_scott;
    DBMS_OUTPUT.PUT_LINE(v_empno, v_nom, v_date);
END;
```

Exo #2 :

Afficher le nom, le salaire et le nom du département de l'employé 'SMITH'.

Réponse :

```
DECLARE
    v_nom EMP.ename%type;
    v_sal EMP.sal %type;
    v_dept DEPT.dname%type;
    CURSOR cur_emp IS
        SELECT e.ename, e.sal, d.dname
        FROM EMP e, DEPT d
        WHERE e.deptno = d.deptno
        AND e.ename = 'SMITH';
BEGIN
    OPEN cur_emp;
    FETCH cur_emp INTO v_nom, v_sal, v_dept;
    DBMS_OUTPUT.PUT_LINE(v_nom, v_sal, v_dept);
    CLOSE cur_emp;
END;
```

Exo #3 :

Afficher le nom, le salaire des cinq premiers employés de la table EMP.

Réponse :

```
DECLARE
    v_nom EMP.ename%type;
    v_sal EMP.sal %type;
    CURSOR cur_sal IS
        SELECT e.ename, e.sal
        FROM EMP
BEGIN
    OPEN cur_sal;
    FOR i IN 1..5
        LOOP
            FETCH cur_sal INTO v_nom, v_sal;
            DBMS_OUTPUT.PUT_LINE('Nom : '||v_nom);
            DBMS_OUTPUT_PUT.LINE('Salaire : '||v_sal);
    END LOOP;
    CLOSE cur_sal;
END;
```

Les attributs d'un curseur :

%FOUND
%NOTFOUND } Dernière ligne traitée
%ISOPEN : ouverture d'un curseur
%ROWCOUNT : nombre de lignes déjà traitées

- nom_curseur%FOUND = true : si le dernier FETCH a ramené une ligne.
- nom_curseur%NOTFOUND = true : si le dernier FETCH n'a pas ramené de ligne
- nom_curseur%ISOPEN = true : si le curseur est ouvert

```
ex. :  IF NOT (nom_curseur%ISOPEN) THEN
           OPEN nom_curseur;
       ENDIF;
```

- nom_curseur%ROWCOUNT :traduit la nième ligne ramenée par le FETCH.

Exo #1 :
Afficher toutes les lignes de la table DEPT en utilisant un curseur, l'attribut '%FOUND' et la boucle WHILE.

Réponse :

```
DECLARE
    v_dept DEPT%ROWTYPE;
    CURSOR cur_dept IS
        SELECT * FROM DEPT;
BEGIN
    OPEN cur_dept;
    FETCH cur_dept INTO v_dept;
    WHILE cur_dept%FOUND
        LOOP
            DBMS_OUTPUT.PUT_LINE(v_dept.deptno||''||v_dept.dname||''||v_dept.loc);
            FETCH cur_dept INTO v_dept;
        END LOOP;
        CLOSE cur_dept;
END;
```

Exo #2 :
Afficher le nom et le salaire des 6 premiers employés de EMP. Utiliser %ROWCOUNT, %NOTFOUND et la boucle LOOP.

Réponse :

```
DECLARE
    v_nom EMP.ename%TYPE;
    v_sal EMP.sal%TYPE;
    CURSOR cur_sal IS
        SELECT ename, sal FROM EMP;
BEGIN
    OPEN cur_sal;
    LOOP
        FETCH cur_sal INTO v_nom, v_sal;
        DBMS_OUTPUT.PUT_LINE('Nom : '|| v_nom  ||' '|| 'Sal : '||v_sal);
    EXIT WHEN cur_sal%ROWCOUNT > 6 OR cur_sal%NOTFOUND;
        /* affichage de la dernière ligne*/
        DBMS_OUTPUT.PUT_LINE('Nom : '|| v_nom  ||' '|| 'Sal : '||v_sal);
    END LOOP;
    CLOSE cur_sal;
END;
```

Exo #3 :
Afficher les 3 premières lignes de la table DEPT
Réponse :

```
DECLARE
    v_dept DEPT%ROWTYPE;
    CURSOR cur_dept IS
        SELECT * FROM DEPT;
BEGIN
    OPEN cur_dept;
    LOOP
        FETCH cur_dept INTO v_dept;
        EXIT WHEN cur_dept%ROWCOUNT > 3 OR cur_dept%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(v_dept.deptnol||''||v_dept.dname||''||v_dept.loc);
    END LOOP;
    CLOSE cur_dept;
END;
```

Curseur FOR LOOP :

Syntaxe :

```
FOR nom_enregist IN nom_curseur LOOP
    Instruction1;
    Instruction2;
    …
END LOOP;
```

N.B. :

- l'ouverture du curseur, son FETCH et sa fermeture sont implicite
- l'enregistrement (nom_enregist) est déclaré implicitement.

Exo #1:
Afficher le nom et le salaire de tous les employés de EMP.
Réponse :

```
DECLARE
    CURSOR cur_emp IS SELECT ename, sal, FROM EMP;
BEGIN
    FOR v_emp IN cur_emp LOOP
        DBMS_OUTPUT.PUT_LINE('Nom : '|| v_emp.ename ||' '|| 'Sal : '||v_emp.sal);
    END LOOP;
END;
```

Exo #2:
Même exercice, sauf afficher uniquement les 3 premiers employés.
Réponse :

```
DECLARE
    CURSOR cur_emp IS SELECT ename, sal, FROM EMP;
BEGIN
    FOR v_emp IN cur_emp LOOP
        EXIT WHEN cur_emp%ROWCOUNT > 3;
        DBMS_OUTPUT.PUT_LINE('Nom : '|| v_emp.ename ||' '|| 'Sal : '||v_emp.sal);
    END LOOP;
END;
```

Curseurs avec paramètres

Syntaxe :

```
CURSOR nom_curseur [(param1 type1, param2 type2, …)] IS ordre select;
…
OPEN nom_curseur(contenu de param1, contenu de param2, …);
```

N.B. :

type = CHAR, VARCHAR2, NUMBER, %TYPE,…

Exo #1 :

Créer un curseur avec paramètre p_deptno qui ramène le No, le nom et la localisation du département no 10

Réponse :

```
DECLARE
    v_dept DEPT%ROWTYPE;
    CURSOR cur_dept(p_num DEPT.deptno%type) IS
        SELECT * FROM DEPT WHERE deptno = p_num;
BEGIN
    OPEN cur_dept(10); /* département 10 */
    FETCH cur_dept INTO v_dept;
    DBMS_OUTPUT.PUT_LINE(v_dept.deptno||''||v_dept.dname||''||v_dept.loc);
    CLOSE cur_dept;
END;
```

Exo #2 :

Même exercice sauf qu'il faut ramener les informations des dept 10 et 20.

Réponse :

```
DECLARE
    v_dept DEPT%ROWTYPE;
    CURSOR cur_dept(p_num DEPT.deptno%type) IS
        SELECT * FROM DEPT WHERE deptno = p_num;
BEGIN
    OPEN cur_dept(10); /* département 10 */
    FETCH cur_dept INTO v_dept;
    DBMS_OUTPUT.PUT_LINE(v_dept.deptno||''||v_dept.dname||''||v_dept.loc);
    CLOSE cur_dept;

    OPEN cur_dept(20); /* département 20 */
    FETCH cur_dept INTO v_dept;
    DBMS_OUTPUT.PUT_LINE(v_dept.deptno||''||v_dept.dname||''||v_dept.loc);
    CLOSE cur_dept;
END;
```

Exo #3 :
Afficher le no d'employé, le nom d'employé et le salaire des employés qui travaillent dans le département no 10.
Réponse :

```
DECLARE
    v_emp EMP%ROWTYPE;
    CURSOR cur_emp(p_num DEPT.deptno%TYPE) IS
        SELECT * FROM EMP WHERE deptno = p_num;
BEGIN
    OPEN cur_emp(10);
    LOOP
        FETCH cur_emp INTO v_emp;
        EXIT WHEN cur_emp%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(v_emp.empno||''||v_emp.ename||''||v_emp.sal);
    END LOOP;
    CLOSE cur_emp;
END;
```

6. Les enregistrements et les tables PL/SQL

De même que les variables, les variables composées ont aussi des types de données. Les types de données composées (collection) sont soit des enregistrements (Record), soit des tables (Matrice).

Un enregistrement est un groupe d'items stockés dans des champs chacun d'eux (item) a un nom et un type de données.

Une table contient une colonne et une clé primaire qui permet l'accès aux lignes.

Une fois définis, les enregistrements et les tables PL/SQL peuvent être réutilisés.

6.1. Les enregistrements PL/SQL

Chaque enregistrement peut avoir autant de champs (colonnes) que nécessaire.

On peut associer à un enregistrement des valeurs initiales et peuvent être Not Null.

Les champs sans valeurs initiales sont initialisées à Null

La valeur 'Default' peut être utilisée lors de la définition d'un enregistrement.

On peut définir un enregistrement dans la partie déclaration d'un bloc, d'une procédure, d'une fonction ou d'un package.

Un enregistrement peut être composé d'un autre enregistrement.

Création d'un enregistrement

Syntaxe :

```
TYPE nom_type IS RECORD
        Décl_champ1, décl_champ2, …);
Identificateur nom_type;
```

Où

nom_type : nom de l'enregistrement

déclar_champ : nom_champ {type_champ | variable%type | table.col%type | table%rowtype}[[not null] { := | Default} expr]

N.B. :

Les champs déclarés Not Null doivent être initialisés.

Exo #1 :

Créer un enregistrement qui contient 3 champs : le nom, le job et le salaire.
Lui affecter le nom, le job et le salaire de l'employé 'SMITH' de la EMP.

Réponse :

```
DECLARE
    TYPE type_enregist IS RECORD
        (nom EMP.ename%type, job EMP.job%type, sal EMP.sal%type);
    enregist type_enregist;
BEGIN
    SELECT ename, job, sal INTO enregist
    FROM EMP
    WHERE ename='SMITH';
    DBMS_OUTPUT.PUT_LINE(enregist.nom||enregist.job||enregist.sal);
END;
```

Exemple :

```
DECLARE
    TYPE type_rec IS RECORD
        (num EMP.empno%type Not Null :=100,
        nom EMP.ename%type,
        job EMP.job%type,
        sal EMP.sal%type);
    rec type_rec;
```

Exo #2 :
Écrire un programme pour afficher les informations de la table DEPT en utilisant les enregistrements, et SQL*Plus pour faire entrer le no dept.
Réponse :

SQL*Plus	Set server output on	(Paramètres SQL*Plus)
	Set verify off	affiche la valeur de p_deptno avant et après
	Accept p_deptno	exécution

```
DECLARE
    v_dept DEPT%ROWTYPE;
BEGIN
    SELECT * INTO v_dept
    FROM DEPT
    WHERE deptno = &p_deptno
    DBMS_OUTPUT.PUT_LINE(v_dept.deptno||''||v_dept.dname||''||v_dept.loc);
END;
```

Curseur et enregistrement
Exo #3 :
Créer un curseur cur_emp qui ramène le no et le sal de l'employé 'SMITH'.
Créer un enregistrement enr_emp de même type que cur_emp, lui affecter le résultat du curseur et afficher le tout.
Réponse :

```
DECLARE
    CURSOR cur_emp IS
        SELECT empno, sal
        FROM EMP
        WHERE ename = 'SMITH';
    Enr_emp cur_emp%ROWTYPE;
BEGIN
    OPEN cur_emp;
    FETCH cur_emp INTO enr_emp;
    DBMS_OUTPUT.PUT_LINE(enr_emp.empno||enr_emp.sal);
    CLOSE cur_emp;
END;
```

6.2. Les tables PL/SQL

Une table PL/SQL est similaire à un tableau (Matrice).
Une table PL/SQL doit contenir deux (02) composants.
C'est une matrice dynamique (dimension variable)

1. Une clé primaire de type "BINARY-INTEGER" (entier binaire) qui indexe la table PL/SQL
2. Une colonne de type scalaire (char, date, number,….) ou de type enregistrement pour les données dans la table PL/SQL

Syntaxe :

```
TYPE nom_type IS TABLE OF
    {type_col | variable%type | table.col%type} [Not Null]
    index by binary-interger;
ident nom_type;
```

où :
nom_type : c'est la déclaration du nom du type de la table.
Type_col : char, number, varchar2, date,…
Ident : c'est le nom de l'identificateur qui représente la tablePL/SQL.

Exo #1 :
Créer la table PL/SQL de type date dans laquelle on affecte le champ hiredate de la table EMP pour les employés 'SMITH' et 'SCOTT'.
Réponse :

```
DECLARE
    v_ind number := 1;
    TYPE d_table_type IS TABLE OF date
        Index by binary-integer;
    d_table d_table_type;
BEGIN
    SELECT hiredate INTO d_table(v_ind)
    FROM EMP
    WHERE ename = 'SMITH';
    DBMS_OUTPUT.PUT_LINE(d_table(v_ind);
    v_ind := v_ind + 1;

    SELECT hiredate INTO d_table(v_ind)
    FROM EMP
    WHERE ename = 'SCOTT';
    DBMS_OUTPUT.PUT_LINE(d_table(v_ind);
END;
```

Exo #2 :
Créer 2 tables PL/SQL contenant les champs ename et hiredate de la table EMP.
Réponse :

```
DECLARE
    v_nb number;
    TYPE n_table_type IS TABLE OF EMP.ename%type
        Index by binary-integer;
```

```
    n_table d_table_type;
    TYPE d_table_type IS TABLE OF EMP.hiredate%type
        Index by binary-integer;
    d_table d_table_type;
    CURSOR cur_emp IS SELECT ename, hiredate FROM EMP;
BEGIN
    SELECT COUNT (*) INTO v_nb
    FROM EMP;
    OPEN cur_emp;
    FOR v_ind IN 1..v_nb LOOP
        FETCH cur_emp INTO n_table(v_ind), d_table(v_ind);
        DBMS_OUTPUT.PUT_LINE(n_table(v_ind) ||''|| d_table(v_ind));
    END LOOP;
    CLOSE cur_emp;
END;
```

Exo #3 :

Créer une table PL/SQL qui contient toute la ligne de DEPT

Réponse :

```
DECLARE
    v_nb number(2);
    TYPE dept_table_type IS TABLE OF DEPT%type
        Index by binary-integer;
    dept_table dept_table_type;
BEGIN
    SELECT COUNT (*) INTO v_nb
    FROM DEPT;
    FOR i IN 1..v_nb LOOP
        SELECT * INTO dept_table(i)
        FROM DEPT
        WHERE deptno = i*10;
    END LOOP;
    FOR i IN 1..v_nb LOOP
        DBMS_OUTPUT.PUT_LINE(dept_table(i).deptno ||''||
                                    dept_table(i).dname || ''||
                                    dept_table(i).loc);
    END LOOP;
END;
```

Utilisation des méthodes pour les tables PL/SQL

- **Exists(n) :** retourne vrai si nième élément de la table PL/SQL existe.
- **Count :** retourne le nombre d'éléments que la table contient.
- **First :** retourne le plus petit indice de la table PL.
- **Last :** retourne le grand indice de la table PL. Retourne NULL si la table est vide.
- **Prior(n) :** retourne l'indice qui précède l'indice n.
- **Next(n) :** retourne l'indice qui succède l'indice n.
- **Extend(n,i) :** augmente la taille d'une table PL.
  Extend : ajoute un élément null à la table
  Extend(n) : ajoute n éléments null à la table.
  Extend(n,i) : ajoute n copies du $i^{ième}$ élément à la table.
- **Trim :**
  Trim : supprime un élément à partir de la fin de la table.
  Trim(n) : supprime n éléments à partir de la fin de la table.
- **Delete :**
  Delete : supprime tous les éléments de la table PL.
  Delete(n) : supprime n éléments de la table PL.
  Delete(m,n) : supprime tous les éléments dans l'intervalle m..n de la table PL.

Syntaxe :

nom table.nom_méthode[(paramètres)]

Ex :

Ajouter à l'exercice précédent ce qui suit :

```
DBMS_OUTPUT.PUT_LINE('Le nombre d'éléments de la table est : '||d_table.count);
DBMS_OUTPUT.PUT_LINE('Le plus petit indice est : '||d_table.first);
DBMS_OUTPUT.PUT_LINE('Le plus grand indice est : '||d_table.last);
DBMS_OUTPUT.PUT_LINE('L'indice qui précède l'indice 5 est : '||d_table.prior(5));
DBMS_OUTPUT.PUT_LINE('L'indice qui succède l'indice 5 est : '||d_table.next(5));
IF d_table.exists(5) THEN
    DBMS_OUTPUT.PUT_LINE('Le 5ième élément existe);
ENDIF;
```

7. Les procédures

Une procédure est un bloc PL/SQL nommé qui exécute une ou plusieurs actions.
Une procédure peut être stockée dans la base de données, comme tout autre objet, et peut être réutilisée à son souhait.

Syntaxe :

CREATE [OR REPLACE] PROCEDURE nom_proc
(param1 [mode] type_donnée,
param2 [mode] type_donnée,
….
)
IS
Bloc PL/SQL

Où :
Mode : IN (par défaut) | out | IN OUT
Type_donnée : char, date, varchar2, number, %type,….

N.B. :

- Le bloc PL/SQL ne contient pas 'DECLARE'
- Une procédure peut être créée soit sous SQL*Plus, soit sous 'Procedure Builder' qui est un outil d'Oracle, dont le rôle est de créer des procédures, fonctions, packages et triggers de la base de données.

Création d'une procédure sous SQL*Plus

- Écrire la procédure dans l'éditeur et sauvegarder avec extension .sql
- Créer la procédure avec RUN.
- Utiliser 'SHOW ERRORS' pour voir les erreurs de la compilation.

Création sous Procedure Builder
C'est plus simple dans l'outil. Voir l'outil.

N.B. :
Sous SQL*Plus, ne pas confondre le fichier qui contient la procédure et la procédure elle-même.
Get c :/../fichier.sql
Run : création de la procédure.
Pour exécuter la procédure : execute nom_proc(paramètres)

N.B. :
Écrire le programme PL/SQL qui crée la procédure
On peut sauvegarder la procédure
SQL*Plus : fichier → save as → c:\Travail\nom_fichier.sql
L'appeler à partir de l'éditeur
GET c:\Travail\nom_fichier.sql
RUN (exécuter le fichier .sql et non la procédure
Création de la procédure
Sinon SHOW ERRORS
Exécuter la procédure → EXECUTE nom_proc(param)

Exo #1 :

```
CREATE OR REPLACE PROCEDURE proc_sal(p_empno EMP.empno%type) IS
    v_ename EMP.ename%type;
    v_sal EMP.sal%type;
BEGIN
    SELECT ename, sal INTO v_ename, v_sal
    FROM EMP
    WHERE empno = pempno;
    DBMS_OUTPUT.PUT_LINE(p_empno ||''||v_ename||''||v_sal);
END;
```

N.B. :

La meilleur façon de tester une procédure, c'est de créer un bloc PL/SQL dans lequel on fait appel à la procédure.

Pour l'exo #1 :

```
DECLARE
    v_num EMP.empno%type;
BEGIN
    v_num := 7369;
    proc_sal(v_num);
END;
```

Ou bien avec SQL*PLUS

```
SQL>EXECUTE proc_sal(7369);
```

Exo #2:

Créer une procédure qui donne le nom, le salaire et la commission d'un employé donné.

Réponse :

```
CREATE OR REPLACE PROCEDURE emp_sal
            (v_num IN EMP.empno%type,
             v_nom OUT EMP.ename%type,
             v_sal OUT EMP.sal%type,
             v_comm OUT EMP.comm) IS
BEGIN
    SELECT ename, sal, comm
    FROM EMP
    WHERE empno = v_num;
END;
```

Test sur SQL*PLUS: une fois la procedure créée

```
Variable g_nom char(15)
Variable g_sal number
Variable g_comm number
EXECUTE emp_sal(7369, :g_nom, :g_sal, :g_comm)
Print g_nom
Print g_sal
Print g_comm
```

Test avec un bloc PL/SQL

```
DECLARE
    v_nom char(15);
    v_sal number;
    v_comm number;
BEGIN
    Emp_sal(7654, v_nom, v_sal, v_comm);
    DBMS_OUTPUT.PUT_LINE(v_nom||''||v_sal||''||v_comm);
END;
```

Exo #3:

Écrire une procédure qui transforme un No tél de numérique (5141234567) au mode caractère (514) 123-4567.

```
CREATE PROCEDURE tel_format(v_tel IN OUT varchar2) IS
BEGIN
    v_tel := '('||substr(v_tel, 1, 3) || ')'||substr(v_tel, 4, 3)||'-'||substr(v_tel, 7, 4);
                                                    ou substr(v_tel, 7);
END;
```

Suppression d'une procédure : DROP PROCEDURE nom_proc

Dictionnaire de données : USER_SOURCE, USER_OBJECTS

8. Les fonctions

C'est aussi des blocs PL/SQL nommés qui retournent une valeur.
Une fonction peut être stockée dans la base de données comme les autres objets de la base de données.

Syntaxe :

```
CREATE [OR REPLACE] FUNCTION nom_fonction
      (param1 [mode] type_donnée,
       param2 [mode] type_donnée,
      ….
      )
      RETURN type_donnée IS
          Bloc PL/SQL;
```

Où :
Mode : IN seulement
Return type_donnée : ne doit pas avoir de dimension.

Exo #1 :
Écrire une fonction avec un seul paramètre en entrée (No employé qui retourne le sal d'un employé.

Réponse :

```
CREATE OR REPLACE FUNCTION sal_emp(v_num IN EMP.empno%type)
RETURN number IS
    v_sal EMP.sal%type := 0;
BEGIN
    SELECT sal INTO v_sal FROM EMP WHERE empno = v_num;
    Return (v_sal);
END;
```

Test avec le bloc PL/SQL:

```
DECLARE
    v_sal number := 0;
BEGIN
    v_sal := sal_emp(7369);
    DBMS_OUTPUT.PUT_LINE('Salaire : '||v_sal);
END;
```

Test sous SQL*PLUS:

```
Variable g_sal number;
EXECUTE :g_sal := sal_emp(7369)
Print g_sal
       800
```

Exo #2:
Écrire une fonction qui retourne la taxe d'une valeur

Réponse :

```
CREATE OR REPLACE f_tax(v_valeur IN number) IS
BEGIN
    RETURN (v_valeur * 0.15);
END;
```

Exo #3 :
Écrire une fonction qui permet de convertir les chartimes en HH24.
Réponse :

```
CREATE OR REPLACE FUNCTION translate_chartime_to_HH24
(p_time varchar2) return number IS
DECLARE
    V_return INT := -1;
BEGIN
    IF CHAR_LENGTH(p_time) = 10 THEN
        V_return = CONVERT(INT, substr(p_time, 5, 1));
        IF (substr(p_time, 17,1) = 'P') AND (v_return < 12) THEN
            V_return = v_return + 12;
    ELSE
        V_return = CONVERT(INT, substr(p_time, 5, 2));
        IF (substr(p_time, 18,1) = 'P') AND (v_return < 12) THEN
            V_return = v_return + 12;
    ENDIF;
    Return v_return;
END;
END translate_chartime_to_HH24;
```

Supprression d'une fonction : DROP FUNCTION nom_fonction

9. Les packages

Un package est un ensemble de procédures, de fonctions, de variables, de constantes, de curseurs et d'exceptions stockés dans la base de données Oracle.
Pour concevoir un package, on doit créer une spécification et un corps.
Ne peut être appelé ni paramétré (package).

9.1. Spécification

C'est la partie déclaration de tous les composants du package.
Syntaxe :

```
CREATE [OR REPLACE] PACKAGE nom_package IS
    Section declaration;
END;
```

Où :
Section déclaration : consiste à déclarer les variables, les constantes, les fonctions, les procédures, les curseurs et les exceptions.

9.2. Corps du package

Définit les procédures, les fonctions, les variables, les constantes, les curseurs et les exceptions du package.

Syntaxe :

```
CREATE PACKAGE BODY nom_package IS
    Corps des procédures;
    Corps des fonctions;
    Déclaration des variables;
    Déclaration des constantes;
    Corps des curseurs;
    Déclaration des exceptions;
END;
```

Exemple :

```
CREATE OR REPLACE PACKAGE comm_pack IS
    G_comm number := 10;
    Procedure calc_comm(v_comm IN number);
END;
SQL> EXECUTE comm_pack.g_comm := 5
       EXECUTE comm_pack.calc_comm(8)
```

Exo #1:

Créer un package (pack1) contenant :

- une variable global initialisée à 10
- une fonction (fonct1) avec 1 paramètre en entrée (no employé) et retourne le salaire de l'employé
- un curseur (cur_max) qui ramène le salaire maximal de tous les employés
- une fonction (fonct2) qui retourne le salaire maximal de tous les employés en utilisant le curseur.
- une procédure (pro1) avec 1 paramètre en entrée (le no employé) et 2 paramètres en sortie le nom et le salaire de l'employé.

Réponse :

Spécification :

```
CREATE OR REPLACE PACKAGE pack1 IS
    p_global number := 10;
    FUNCTION fonct1(p_num number) return number;
    FUNCTION fonct2 return number;
    CURSOR cur_max IS SELECT max(sal) FROM EMP;
    PROCEDURE proc1(p_num number, p_nom OUT char, p_sal OUT number);
END;
```

Body (corps)

```
CREATE PACKAGE BODY pack1 IS
    FUNCTION fonct1(p_num number) return number IS
        v_sal number;
    BEGIN
        SELECT sal INTO v_sal FROM EMP WHERE empno = p_num;
        Return (v_sal);
    END;
```

```
    FUNCTION fonct2 return number IS
        v_max number;
    BEGIN
        OPEN cur_max;
        FETCH cur_max INTO v_max;
        CLOSE cur_max;
        Return (v_max);
    END;
    PROCEDURE proc1(p_num number, p_nom OUT char, p_sal OUT number) IS
    BEGIN
        SELECT ename, sal INTO p_nom, p_sal
        FROM EMP
        WHERE empno = p_num;
    END;
END;
```

Test sous SQL*PLUS

```
SQL> variable g_var number
        Execute :g_var := pack1.p_global
        Print g_var          (10)

        Variable g_sal number
         Execute :g_sal := pack1.fonct1(7934)
         Print g_sal          (1300)

         Variable g_nom char(15)
          Execute pack1.proc1(7369, :g_nom, :g_sal)
          Print g_nom              ('SMITH ')
          Print g_sal              (800)

         Variable g_max number
         Execute :g_max := pack1.fonct2
         Print g_max         (5000)
```

Test avec un bloc PL/SQL

```
DECLARE
    p_nb number;
    p_char char(15);
BEGIN
    p_nb := pack1.fonct1(7369);
    DBMS_OUTPUT.PUT_LINE('Le salaire de l''employé 7369 est : '|| p_nb);
    DBMS_OUTPUT.PUT_LINE('Le salaire de l''employé 7369 est : '||
    pack1.fonct1(7369));
    pack1.proc1(7698, p_char, p_nb);
    DBMS_OUTPUT.PUT_LINE('Le nom et le sal de l''employé 7698 est : '||p_char
    ||''||p_nb));
    DBMS_OUTPUT.PUT_LINE('Le sal max des employés est : '||pack1.fonct2);
END;
```

Suppression des packages :

- Pour supprimer le corps et la spécification d'un package :
  DROP PACKAGE nom_package
- Pour supprimer le corps d'un package :
  DROP PACKAGE BODY nom_package

```
SELECT line, substr(text,1,70)
FROM USER_SOURCE
WHERE name = pack1 and type = 'PACKAGE';
                             = 'PACKAGE BODY';
```

10. Les exceptions (la gestion des erreurs)

La section EXCEPTION permet de gérer les erreurs survenues lors de l'exécution d'un bloc PL/SQL.
Il existe deux (2) types d'erreurs :

1. Erreurs utilisateur
2. Erreurs Oracle

10.1. Erreurs utilisateur

Syntaxe :

```
DECLARE
    nom_erreur EXCEPTION;
    …
BEGIN
    …
    IF condition THEN
        RAISE nom_erreur; /* On déclenche l'erreur */
        …
    EXCEPTION
        WHEN nom_erreur THEN traitement de l'erreur;
        [WHEN OTHERS THEN instr1;
                                  …..]
END;
```

N.B. :

Quand une erreur se déclenche, on peut identifier le code et le message de l'erreur, en utilisant les deux fonctions SQLCODE et SQLERRM.
SQLCODE : retourne le no de l'erreur.
SQLERRM : retourne le message associé à l'erreur.

Exemple :

SQLCODE	SQLERRM (Description)
0	No exception encountered
1	User_defined exception
+100	No data found
Nombre négatif	D'autres erreurs du serveur Oracle

Exo #1 :

Écrire un programme qui affiche le no, le nom et la localisation d'un département donné (le no dept servira d'entrée avec ACCEPT de SQL*PLUS.
Si le no dept n'existe pas dans la table DEPT alors erreur.

Réponse :

```
    ACCEPT p_dept
PL/SQL :
    DECLARE
        v_deptno DEPT.deptno%type;
        v_dname DEPT.dname%type;
        v_loc DEPT.loc%type;
        CURSOR cur_dept IS
            select deptno, dname, loc from DEPT where deptno = &p_dept;
        v_except EXCEPTION;
    BEGIN
        OPEN cur_dept;
        FETCH cur_dept INTO v_deptno, v_dname, v_loc;
        IF cur_dept%NOTFOUND THEN
            RAISE v_except;
        ENDIF;
        DBMS_OUTPUT.PUT_LINE(v_deptno||''||v_dname||''||v_loc);
        CLOSE cur_dept;
        EXCEPTION
            WHEN v_except THEN
              DBMS_OUTPUT.PUT_LINE('Le no dept n''existe pas …');
              DBMS_OUTPUT.PUT_LINE('Le no de l''erreur : '||SQLCODE);
              DBMS_OUTPUT.PUT_LINE('Le nom de l''erreur: '||SQLERRM);
    END;
```

Exo #2:

Utilisation de **SQL%NOTFOUND**

```
DECLARE
    v_except EXCEPTION;
BEGIN
    UPDATE DEPT
    SET loc = &p_loc
    WHERE deptno = &p_deptno
    IF SQL%NOTFOUND THEN
        RAISE v_except;
    ENDIF;
    COMMIT;
    EXCEPTION
        WHEN v_except THEN
          DBMS_OUTPUT.PUT_LINE('Le no dept n"existe pas …');
          DBMS_OUTPUT.PUT_LINE('Le no de l"erreur : '||SQLCODE);
          DBMS_OUTPUT.PUT_LINE('Le nom de l"erreur: '||SQLERRM);
END;
```

10.2. Erreurs d'Oracle

Les plus courantes sont :

1. **NO_DATA_FOUND** : lorsqu'un SELECT ne ramène pas d'information.

Exp :

```
DECLARE
    v_ename EMP.ename%type;
BEGIN
    SELECT ename INTO v_ename FROM EMP WHERE empno = &v_empno;
    DBMS_OUTPUT.PUT_LINE('L"employé est : '||v_ename);
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Le no d"employé n"existe pas');
            DBMS_OUTPUT.PUT_LINE('Le no de l"erreur : '||SQLCODE);  ⇨ 100
            DBMS_OUTPUT.PUT_LINE('Le nom de l"erreur: '||SQLERRM);
END;
```

⇩

ORA-01403 : No data found

2. **TOO_MANY_ROWS** : lorsqu'un SELECT ramène plus d'une ligne.

Exp :

```
DECLARE
    v_ename EMP.ename%type;
BEGIN
    SELECT ename INTO v_ename FROM EMP WHERE deptno = &v_deptno;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Le no de dept n"existe pas');
        WHEN TOO_MANY_ROWS THEN
            DBMS_OUTPUT.PUT_LINE('Le no de l"erreur:'||SQLCODE);   ⇨ -1422
            DBMS_OUTPUT.PUT_LINE('Le nom de l"erreur:'||SQLERRM);
    END;
```

⇩

ORA-1422 : exact fetch returns more than …

3. **ZERO_DEVIDE :** division par zéro.

Exp :

```
DECLARE
    v_val number := 20;
BEGIN
    v_val := v_val / &v_val1;
    DBMS_OUTPUT.PUT_LINE('v_val = '||v_val);
    EXCEPTION
        WHEN ZERO_DIVIDE THEN
            DBMS_OUTPUT.PUT_LINE('Division par zéro …);
            DBMS_OUTPUT.PUT_LINE('Le no de l"erreur:'||SQLCODE);   ⇨ -1476
            DBMS_OUTPUT.PUT_LINE('Le nom de l"erreur:'||SQLERRM);
    END;
```

⇩

ORA-01476 : Division is equal to zero

10.3. Autres erreurs d'Oracle

DUP_VAL_ON_INDEX : violation des contraintes de la clé primaire ou unique. Insertion de valeurs dupliquées.

VALUE_ERROR : erreur arithmétique.

INVALID_NUMBER : problème dans la conversion de caractères en nombres.

CURSOR_ALREADY_OPEN : le curseur est déjà ouvert.

INVALID_CURSOR : l'erreur surgit quand on essaie de fermer un curseur qui est déjà fermé.

ROWTYPE_MISMATCH : les types de données de l'enregistrement auxquels des données du curseur sont assignées sont incompatibles.

Exp :

```
DECLARE
    V_ename EMP.ename%type;
    CURSOR cur_emp IS select ename from EMP where deptno = &p_deptno;
BEGIN
    OPEN cur_emp;
    FETCH cur_emp INTO v_ename;
    DBMS_OUTPUT.PUT_LINE('Le nom de l''employé est : '||v_ename);
    OPEN cur_emp;
    FETCH cur_emp INTO v_ename;
    DBMS_OUTPUT.PUT_LINE('Le nom de l''employé est : '||v_ename);
    CLOSE cur_emp;
    EXCEPTION
        WHEN CURSOR_ALREADY_OPEN THEN
            DBMS_OUTPUT.PUT_LINE('Le curseur est déjà ouvert …');
            DBMS_OUTPUT.PUT_LINE('Le no de l''erreur:'||SQLCODE);  -6511
            DBMS_OUTPUT.PUT_LINE('Le nom de l''erreur:'||SQLERRM);
    END;
```

⇩

ORA-06511 : PL/SQL : cursor already open

11. Les gâchettes (TRIGGERs)

⇨ Types
- ➔ BEFORE / AFTER
- ➔ ROW / TABLE
- ➔ INSERT, DELETE, UPDATE {OF col1, col2, …}
- ➔ pas de COMMIT TRIGGER…

⇨ :OLD.colonne
- ➔ valeur avant mise à jour (NULL si INSERT)

⇨ :NEW.colonne
- ➔ valeur après mise à jour (NULL si DELETE)

⇨ Ne peut inclure d'instructions de contrôle de transaction
- ➔ pas de COMMIT, ROLLBACK, SAVEPOINT

⇨ Raise_application_error(numéro_erreur, chaîne)
- ➔ pas de ROLLBACK par défaut
- ➔ peut outrepasser défaut en définissant un
  EXCEPTION HANDLER pour numéro_erreur
- ➔ numéro_erreur ∈ [-20000, -20999] – autres numéros sont réservés

⇨ Compilation dynamique
- ➔ compilé au premier appel et conservé dans Shared Pool
- ➔ peut être évacué si pool plein
- ➔ appeler procédure pour éviter recompilation

Exemples de TRIGGERs

Exp 1 :

```
CREATE OR REPLACE TRIGGER af_detail_livraison
AFTER INSERT
ON DETAILSS_DE_LIVRAISON
FOR EACH ROW
DECLARE
BEGIN
    UPDATE lignes_de_commande
    SET
        Quantite_en_attente = quantite_en_attente - :NEW.quantite_livree
    WHERE
        Lignes_de_commande.commande_no_commande
            = :NEW.lignecomm_commande_no_commande
    AND lignes_de_commande.produit_no_produit
            = :NEW.lignecomm_produit_no_produit;
    UPDATE produits
    SET
        Quantite_en_stock - :NEW.quantite_livree
    WHERE
        no_produit = :NEW.lignecomm_produit_no_produit;
END;
```

Exp 2:

```
REM
REM                    Vérifier si la quantité en stock en suffisante
REM
PROMPT
PROMPT Creating Trigger BE_DET_LIV
CREATE OR REPLACE TRIGGER be_det_liv
BEFORE INSERT
ON DETAILS_DE_LIVRAISON
FOR EACH ROW
BEGIN
    DECLARE
        quantite NUMBER;
    BEGIN
        SELECT quantite_en_stock INTO quantite
        FROM produits
        WHERE no_produit = :NEW.lignecomm_produit_no_produit;
        IF :NEW.quantite_livree > quantite THEN
            RAISE_APPLICATION_ERROR(-20100, 'Quantité en stock insuffisante');
        END;
    END;
END;
```

Exp 3:

```
CREATE TRIGGER total_salaire
AFTER DELETE OR INSERT OR UPDATE OF deptno, sal ON EMP
FOR EACH ROW
BEGIN
    /* assume que deptno et sal sont des champs non null */
    IF DELETING OR (UPDATE AND :OLD.deptno != :NEW.deptno) THEN
        UPDATE DEPT
        SET total_sal = total_sal - :OLD.sal
        WHERE deptno = :OLD.deptno;
    ENDIF;
    IF INSERTING OR(UPDATING AND :OLD.deptno != :NEW.deptno) THEN
        UPDATE DEPT
        SET total_sal = total_sal + :NEW.sal
        WHERE deptno = :NEW.deptno;
    ENDIF;
    IF (UPDATING AND :OLD.deptno != :NEW.deptno AND :OLD.sal != :NEW.sal) THEN
        UPDATE DEPT
        SET total_sal = total_sal - :OLD.sal + :NEW.sal
        WHERE deptno = :NEW.deptno;
    ENDIF;
END;
```

12. Quelques commandes de SQL et de SQL*PLUS

12.1. Commandes de SQL

SELECT	Permet de récupérer des données de la B.D.
INSERT	DML(Langage de manipulation de données)
UPDATE	C'est respectivement insertion, la mise à jour et la suppression
DELETE	de lignes d'une table de la BD.
CREATE	DDL(Langage de définition de données)
AFTER	C'est la création, la mise à jour et la suppression des structures de
DROP	données des tables d'une base de données.
RENAME	
TRUNCATE	
COMMIT	C'est des commandes qui servent à gérer les changements
ROLLBACK	faits par le DML
SAVEPOINT	
GRANT	DCL(Langage de contrôle de données)
REVOKE	Donnent et enlèvent le droit d'accès à la BD et aux structures de la BD.

12.2. Commandes SQL*PLUS

DESC[RIBE] nom_table	Décrit la structure d'une table
A[PPEND] texte	Ajoute du texte à la fin de la ligne courante (L1 → a[ppend] texte
C[HANGE] / old/new	Change l'ancien texte
C[HANGE] /texte	Supprime le texte de la ligne courante
CL[EAR] BUFF[ER]	Supprime les lignes du buffer SQL
DEL	Supprime la ligne courante
I[NPUT]	Insérer un nombre indéfini de lignes
I[NPUT] texte	Insérer une ligne [texte]
L[ist]	Liste toutes les lignes du buffer
L[ist] n	Liste la $n^{ième}$ ligne
L[ist] n m	Liste de la ligne no n à m
R[UN]	Affiche et exécute la requête SQL se trouvant dans le buffer
n	Affiche la ligne no n
n texte	Remplace la $n^{ième}$ ligne par "texte"
O texte	Insérer une ligne avant la ligne no 1.

12.3. Fichiers de commandes SQL*PLUS

SAVE nom_fichier [.ext] [REPLACE] [APPEND] : Sauvegarde le contenu du buffer SQL dans nom_fichier

[REPLACE] : remplace le nom déjà existant
[APPEND] : ajoute au fichier existant

GET fichier[.ext] : obtenir le fichier

START fichier[.ext] : exécuter le fichier, équivalent à @

EDIT fichier.ext : ramène à l'éditeur le contenu du fichier .ext

EDIT : invoke l'éditeur de SQL*PLUS et sauvegardee le contenu du buffer dans un fichier par défaut (afiedt.buf).

SPOOL nom fichier.ext |OFF|OUT : stocke le résultat d'une requête dans un fichier.
OFF : ferme le fichier SPOOL
OUT : ferme le fichier SPOOL et envoie le fichier résultat à l'imprimante.

EXIT : quitter SQL*PLUS

DEFINE nom_variable [= chaîne de caractère seulement]: définir une variable utilisateur et lui affecte une valeur. C'est l'équivalent de ACCEPT.

```
DEFINE employe
DEFINE employe = SMITH
```

UNDEFINE nom_variable : supprime la variable utilisateur.

13. QUIZ

1. Developer receives an error due to the following statement in the declaration section

pi constant number;

Which of the following caused this problem?

A. There is not enough memory in the program for the constant
B. There is no value associated with the constant
C. There is no datatype associated with the constant
D. Pi is a reserved word

2. You are designing your PL/SQL exception handler. Which statement most accuratelydescribes the result of not creating an exception handler for a raised exception?
A. The program will continue without raising the exception
B. There will be a memory leak
C. Control will pass to thePL/SQL block caller's exception handler
D. The program will return a %NOTFOUND error

3. You are determining which types of cursors to use in your PL/SQL code. Which of the following statements is true about implicit cursors?
A. Implicit cursors are used for SQL statements that are not named
B. Developers should use implicit cursors with great care
C. Implicit cursors are used in cursor for loops to handle data
D. Implicit cursors are no longer a feature in Oracle

4. You are constructing PL/SQL process flow for your program. Which of the following is not a feature of a cursor for loop?
A. Record-type declaration
B. Opening and parsing of SQL statements
C. Fetches records from cursor
D. Requires exit condition to be defined

5. A developer would like to use a referential datatype declaration on a variable. The variable name is EMPLOYEE_LASTNAME, and the corresponding table and column is EMPLOYEE and LASTNAME, respectively. How would the developer define this variable using referential datatype?
A. Use employee.lname%type
B. Use employee.lname%rowtype
C. Look up datatype for EMPLOYEE column on LASTNAME table and use that
D. Declare it to be typeLONG

6. After executing an update statement, the developer codes a PL/SQL block to perform an operation based on SQL%ROWCOUNT. What data is returned by the SQL%ROWCOUNT operation?
A. A Boolean value representing the success on failure of the update
B. A numeric value representing the number of rows updated
C. A Varchar2 value identifying the name of the table updated
D. A Long value containing all data from the table

7. You are defining a check following a SQL statement to verify that the statement returned appropriate data. Which three of the following are implicit cursor attributes?(choose three)
A. %found
B. %too_many_rows
C. %notfound
D. %rowcount
E. %rowtype

8. You are constructing PL/SQL process flow into your program. If left out, which of the following would cause an infinite loop to occur in a simple loop?
A. loop
B. end loop
C. if-then
D. exit

9. You are coding your exception handler. The others exception handler is used to handle all of the following exceptions, except one. Which exception does the others exception handler not cover?
A. no_data_found
B. others
C. rowtype_mismatch
D. too_many_rows

10. You are defining a cursor in your PL/SQL block. Which line in the following statement will produce an error?
A. cursor action_cursor is
B. select name, rate, action
C. into action_record
D. from action_table
E. There are no errors in this statement

11. You are developing PL/SQL process flow into your program. Which of the following keywords is used to open a cursor for loop?
A. open
B. fetch
C. parse
D. None, cursor for loops handle cursor opening implicitly

12. You are determining the appropriate program flow for your PL/SQL application. Which one of the following statement about while loops is true?

A. Explicit exit statement are required in while loops

B. Counter variables are required in while loops

C. An if-then statement is needed to signal when a while loop should end

D. All exit conditions for while loops are handled in the while conditional clause

14. Réponse aux questions du QUIZ

1. **B.** There is no value associated with the constant

   Explanation: A value must be associated with a constant in the declaration section. If no value is given for the constant, an error will result.

2. **C.** Control will pass to PL/SQL block caller's exception handler

   Explanation: if the exception raised is not handled locally, then PL/SQL will attempt to handle it at the level of the process that called the PL/SQL block. If the exception is not handled there, then PL/SQL will attempt to keep finding an exception handler that will resolve the exception. If none is found, then the error will be returned to the user.

3. **A.** Implicit cursors are used for SQL statements that are not named

   Explanation: implicit cursors are used for all SQL statements exception for those statements that are named. They are never incorporated into cursor for loops, nor is much given to using them more or less, which eliminates choices B and C. they are definitely a feature of Oracle, eliminating choice D.

4. **D.** Requires exit condition to be defined

   Explanation: A cursor for loop handles just about every feature of cursor processing automatically, including exit conditions.

5. **A.** Use employee.lname%type

   Explanation: The only option in this question that allows the developer to use referential type declaration for column is choice A. Choice B uses the %rowtype referential datatype, which defines a record variable and is not what the developer is after.

6. **B.** A numeric value representing the number of rows updated

   Explanation: %rowcount returns the numeric value representing the number of rows that were manipulated by SQL statement.

7. **A, C, D.** %found, %notfound, %rowcount

   Explanation: These three are the only choices that are valid cursor attributes. The %too_many_rows attribute does not existe in PL/SQL. The %rowtype is a keyword that can be used to declare a record variable that can hold all column values from a particular table.

8. **D.** exit

   Explanation: Without an exit statement, a simple loop will not stop. Although the loop and end loop keyword are needed to define the loop, you should assume these are in place, and you are only trying to figure out how to end the loop. The if-then syntax might be used to determine a test condition for when the loop execution should terminate, but is not required in and of itself to end the loop process execution.

9. **B.** others

   Explanation: There is no others exception. The others exception handler handles all exceptions that may be raised in a PL/SQL block that do not have exception handlers explicitly defined for them. All others choices identify Oracle predefined exceptions that are caught by the others keyword then used in an exception handler. If there is no specific handler for another named exception, the others exception handler will handle that exception.

10. **C.** into action_record

    Explanation: The into clause is not permitted in cursors, nor is it required. Your fetch operation will obtain the value in thee current cursor record from the cursor.

11. **D.** None, cursor for loops handle cursor opening implicitly

    Explanation: The cursor for loops handle, among others things, the opening, parsing and executing of named cursors.

12. **D.** All exit conditions for while loops are handled in the while conditional clause

    Explanation: There is no needed for an exit statement in a while loop, since the exiting condition is defined in the while statement, eliminating choice A. Choice B is also wrong because you don't specifically need to use a counter in a while loop the way you do in a for loop. Finally, choice C is incorrect because even though the exit condition for a while loop evaluates to a Boolean value (for example, exit when (this_condition_is_true), the mecanism to handle the exit does not require an explicit if-then statement.