Programmation en PL/SQL Oracle

Faculté Polydisciplinaire de Ouarzazate (IGE 2012/2013)

Mohamed NEMICHE

Table de matières

Introduction 5

I) Chapitre 1 : Développer un Bloc simple PL/SQL 8

I.1 - Structure d'un Block PL/SQL 8

I.1.1) Règles Syntaxiques d'un Bloc PL/SQL 8

I.1.2) Déclaration de Variables et Constantes – Syntaxe 9

I.1.3) PARTIE DECLARATIVE BLOC PL/SQL 9

I.1.4) Opérateurs en PL/SQL 12

I.1.5) Fonctions en PL/SQL - Exemples 12

I.1.6) Blocs Imbriqués et Portée d'une Variable - Exemple 12

I.1.7) Conventions de Casse pour le Code 12

I.2 - Interaction avec Oracle 13

I.2.1) Retrouver des Données (Extraire de données) - Syntaxe 13

I.2.2) Retrouver des Données - Exemple 13

II) Chapitre 2 : Traitements Conditionnels et Traitements Répétitifs 16

II.1 - Contrôler le Déroulement d'une Exécution PL/SQL 16

II.2 - PARTIE EXECUTABLE 16

II.3 - L'Instruction IF et CASE - Syntaxe 16

II.4 - Instructions LOOP 17

II.5 - Boucle FOR - Exemple 18

II.6 - Boucle WHILE - Exemple 18

III) Chapitre 3 : Curseurs 20

III.1 - Qu'est ce qu'un Curseur ? 20

III.2 - La déclaration d'un curseur 20

III.2.1) L'ouverture du curseur 21

III.3 - Traitement des lignes 22

III.4 - La fermeture du curseur 23

IV) Chapitre 4 : Gestion des Erreurs (EXCEPTION) 26

IV.1 - Exception 26

IV.2 - Intercepter les Exceptions - Syntaxe 26

IV.3 - Règles pour intercepter les Exceptions 27

IV.4 - Exceptions Oracle Non Prédéfinies 29
IV.5 - Exceptions Utilisateur (externes) 30
IV.6 - Fonctions d'interception des erreurs 31
IV.7 - Fonctions d'interception des erreurs- Exemple 31
V) Chapitre 5 : PROCEDURES, FONCTIONS ET PACKAGES 34
V.1 - Généralité 34
V.2 - Procédures 35
V.2.1) PROCEDURES / PARAMETRES 36
V.2.2) Correction des erreurs 37
V.3 - Fonctions 38
V.4 - LES PACKAGES 39
V.4.1) La structure Générale d'un package 39
V.4.2) Opérations sur les packages 41
VI) Chapitre 6 : Déclencheurs (TRIGGERs) 43
VI.1 - Définition 43
VI.2 - Syntaxe 43
VI.3 - Types de déclencheurs 43
VI.4 - Option BEFOR/AFTER 43
VI.5 - Le corps du trigger 44
VI.6 - Les noms de corrélation (OLD/New) 45
VI.7 - Les prédicats conditionnels INSERTING, DELETING et UPDATING 46

Introduction

L'utilisateur saisi une requête (en SQL) et Oracle fourni une réponse. Cette façon de travailler ne fonctionne pas dans un environnement de production, car tous les utilisateurs ne connaissent pas ou n'utilisent pas SQL, et il y a souvent des erreurs.

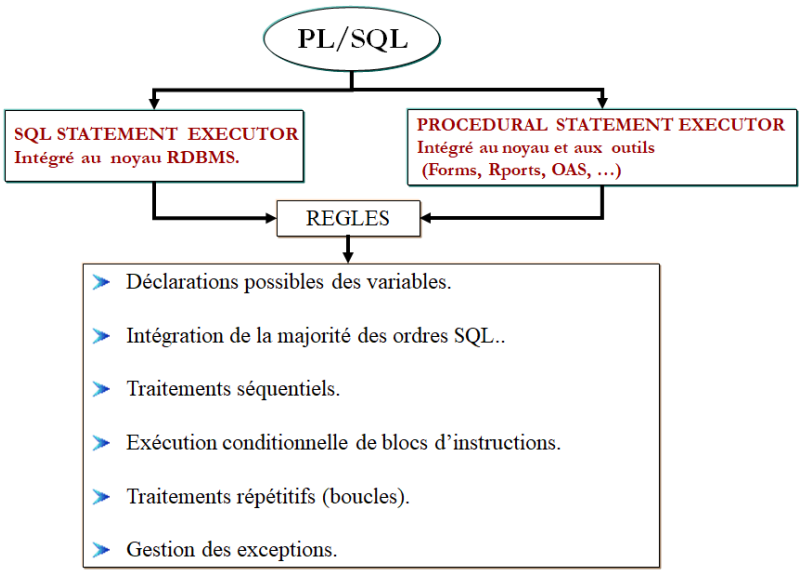
Pour surmonter ces limitations, Oracle intègre un gestionnaire PL / SQL au serveur de base de données et à certains de ses outils (formulaires, rapports, graphiques, etc.). Ce langage intègre toutes les caractéristiques des langages de troisième génération: gestion des variables, structure modulaire (procédures et fonctions), structures de contrôle (boucles et autres structures), contrôle des exceptions et intégration totale dans l'environnement Oracle. .

Les programmes créés avec PL / SQL peuvent être stockés dans la base de données en tant qu'objet supplémentaire; De cette manière, tous les utilisateurs autorisés ont accès à ces programmes et, par conséquent, à la distribution, à l'installation et à la maintenance du logiciel. De plus, les programmes sont exécutés sur le serveur, ce qui suppose une économie importante de ressources sur les clients et une réduction du trafic réseau.

L'utilisation du langage PL / SQL est également essentielle pour créer des déclencheurs de base de données, qui permettent l'implémentation de règles de gestion complexes et d'audits dans la base de données.

PL / SQL supporte toutes les commandes de consultation et de manipulation des données, fournissant sur SQL les structures de contrôle et autres éléments des langages procéduraux de troisième génération. Son unité de travail est le bloc, constitué d'un ensemble d'énoncés, d'instructions et de mécanismes de gestion des erreurs et des exceptions.

Présentation du langage PL/SQL



Avantages de PL/SQL

- Intégration
- Amélioration des performances
- Portabilité
- Développement modulaire

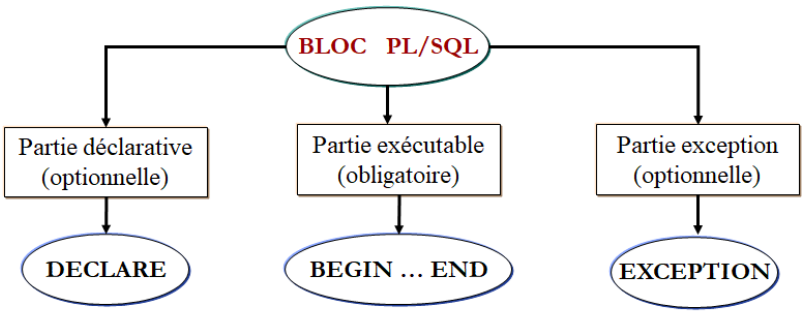
Chapitre 1

Développer un Bloc simple PL/SQL

I) Développer un Bloc simple PL/SQL

```
[ DECLARE ]
        - Variables, constantes, curseurs,
          exceptions utilisateurs
BEGIN
        - Ordres SQL
        - Instructions de Contrôle PL/SQL
[ EXCEPTION ]
        - Traitements à effectuer  lors d'erreurs
END ;
```

I.1 - Structure d'un Block PL/SQL



- Chaque instruction se termine par un point virgule.
- Les commentaires sont possibles /* */.
- Possibilité d'imbrication des blocs.

I.1.1) Règles Syntaxiques d'un Bloc PL/SQL

Identifiants :

- Peuvent contenir jusqu'à 30 caractères.
- Ne peuvent pas contenir de mots réservés à moins qu'ils soient encadrés de guillemets.
- Doivent commencer par une lettre.

- Doivent avoir un nom distinct de celui d'une table de la base ou d'une colonne.

- Utiliser un slash (/) pour exécuter un boc PL/SQL anonyme dans PL/SQL.
- Placer un point virgule (;) à la fin d'une instruction SQL ou SQL*PLUS
- Les chaînes de caractères et les dates doivent être entourées de simples quotes (' ').
- Les commentaires peuvent être
  - sur plusieurs lignes avec :

- **/*** début et
  fin de commentaire***/**
  - sur une ligne précédée de :
  -- début et fin de commentaire

I.1.2) Déclaration de Variables et Constantes – Syntaxe

identifier [CONSTANT] datatype [NOT NULL] [:= | DEFAULT expr];

Règles :

- Adopter les conventions de dénomination des objets.
- Initialiser les constantes et les variables déclarées NOT NULL.
- Initialiser les identifiants en utilisant l'opérateur d'affectation (:=) ou le mot réservé DEFAULT.
- Déclarer au plus un identifiant par ligne.

I.1.3) PARTIE DECLARATIVE BLOC PL/SQL

Types classiques

TYPE	VALEURS
BINARY-INTEGER	entiers allant de –2**31 à 2**31)
POSITIVE / NATURAL	entiers positifs allant jusqu'à 2**31 -1
NUMBER	Numérique (entre –2**418 à 2**418)
INTEGER	Entier stocké en binaire (entre –2**126 à 2**126)
CHAR (n)	Chaîne fixe de 1 à 32767 caractères (différent pour une colonne de table)
VARCHAR2 (n)	Chaîne variable (1 à 32767 caractères)
LONG	idem VARCHAR2 (maximum 2 gigaoctets)
DATE	Date (ex. 01/01/1996 ou 01-01-1996 ou 01-JAN-96 ...)
RAW	Permet de stocker des types de données binaire relativement faibles(<= 32767 octets) idem VARCHAR2. Les données RAW ne subissent jamais de conversion de caractères lors de leur transfert entre le programme et la base de données.
LONG RAW	Idem LONG mais avec du binaire

Déclaration de Variables Scalaires - Exemples

```
v_gender        CHAR( 1 );
v_count         BINARY_INTEGER := 0;
v_total_sal         NUMBER( 9, 2 ) := 0;
v_order_date        DATE := SYSDATE;
c_tax_rate          CONSTANT NUMBER ( 3, 2 ) := 8.25;
v_valid             BOOLEAN NOT NULL := TRUE;
```

L'Attribut %TYPE

- Déclarer une variable à partir :
  - D'une autre variable déclarée précédemment
  - De la définition d'une colonne de la base de données
- Préfixer %TYPE avec :
  - La table et la colonne de la base de données
  - Le nom de la variable déclarée précédemment
- PL/SQL détermine le type de donnée et la taille de la variable.

L'Attribut %TYPE - Exemple

```
DECLARE
```

```
v_last_name          s_emp.last_name%TYPE;
v_first_name         s_emp.first_name%TYPE;
v_balance            NUMBER( 7, 2 );
v_minimum_balance    v_balance%TYPE := 10;
```

- Le type de données de la colonne peut être inconnu.
- Le type de données de la colonne peut changer en exécution.

L'Attribut %ROWTYPE - Avantages

- Le nombre de colonnes, ainsi que les types de données des colonnes de la table de référence peuvent être inconnus.
- Le nombre de colonnes, ainsi que le type des colonnes de la table de référence peuvent changer en exécution
- Utile lorsqu'on recherche
  - Une ligne avec l'ordre SELECT.
  - Plusieurs lignes avec un curseur explicite.

Exemple

```
DECLARE
    dept_record    s_dept%ROWTYPE;
    emp_record     s_emp%ROWTYPE;
```

Les variables référencées à une table de la base

Elles sont liées à des tables au niveau de la base.On les déclare par l'attribut : %ROWTYPE

Exemples

```
        DECLARE
agent   employe%ROWTYPE   -- employe est la table employe --- de la base.
        Au niveau traitement, on pourra écrire :
        BEGIN
                SELECT *                    -- Sélection de tous les -- champs
                INTO  agent
                FROM  employe
                WHERE nom='DUMAS';
        END;
Ou

        BEGIN
                SELECT      nom,dt_entree       -- Sélection de certains champs
                INTO  agent.nom, agent.dt_entree
                FROM  employe
                WHERE nom='DUMAS';
```

END;

I.1.4) Opérateurs en PL/SQL

- Logiques
- Arithmétiques
- Concaténation
- Opérateur exponentiel (**)
- Parenthèses pour contrôler l'ordre des opérations

I.1.5) Fonctions en PL/SQL - Exemples

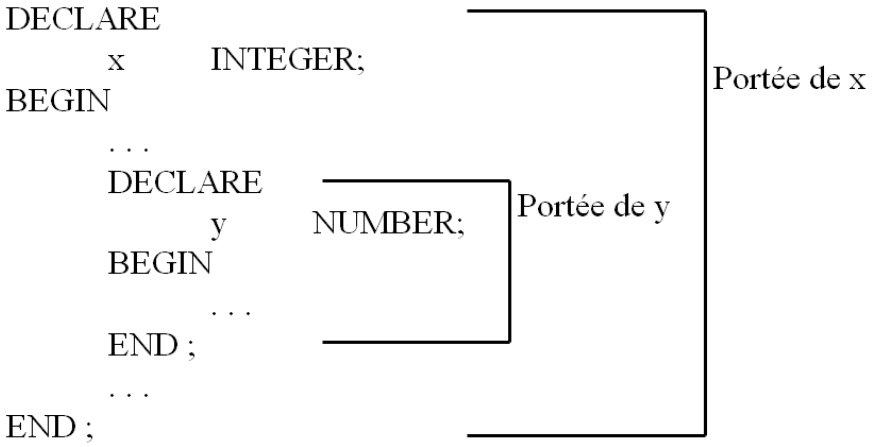
Construire une liste d'adresses pour une société :

v_mailing_address := v_name || CHR(10) ||v_address || CHR(10) || v_country || CHR (10) ||v_zip_code

Convertir le nom de famille en majuscule :

v_last_name := UPPER(v_last_name) ;

I.1.6) Blocs Imbriqués et Portée d'une Variable - Exemple



I.1.7) Conventions de Casse pour le Code

Conventions	Exemples et Casse
Commandes SQL	SELECT, INSERT
Mots Clés PL/SQL	DECLARE, BEGIN, IF
Types de Données	VARCHAR2, BOOLEAN
Identifiants et Paramètres	v_sal, emp_cursor, g_sal
Tables et Colonnes de la base de données	s_emp, order_date, id

I.2 - Interaction avec Oracle

I.2.1) Retrouver des Données (Extraire de données) - Syntaxe

Retrouver des lignes de la base de données avec le SELECT

```
SELECT      select_list
INTO        variable_name | record_name
FROM            table
WHERE       condition ;
```

- La clause INTO est obligatoire.
- Une seule ligne doit être retournée.
- Toute la syntaxe du SELECT est disponible.

I.2.2) Retrouver des Données - Exemple

Retrouver toutes les informations d'un département donné.

```
DECLARE
        v_nom emp.nome%TYPE;
        v_emp emp%ROWTYPE;
BEGIN
        select nome into v_nom from emp where matr = 500;
      select * into v_emp from emp     where matr = 500;
END
/
```

Exceptions SELECT

- Les ordres SELECT en PL/SQL doivent ramener une et une seule ligne.
- Si aucune ou plusieurs lignes sont retrouvées une exception est déclenchée.
- Exceptions du SELECT :
  - **TOO_MANY_ROWS**
  - **NO_DATA_FOUND**
- 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

- 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

Insertion de Données - Exemple

Ajouter des nouveaux employées à la base de donné :

```
DECLARE
v_emp emp%ROWTYPE;
v_nom emp.nome%TYPE;
BEGIN
v_nom := ‘ALAMI';
insert into emp (matr, nome) values(600, v_nom);
v_emp.matr := 610;
v_emp.nome := ‘TOTO';
insert into emp (matr, nome) values(v_emp.matr, v_emp.nome);
commit;
END; --Fin du bloc PL --
/
```

Chapitre 2

Traitements Conditionnels et Traitements Répétitifs

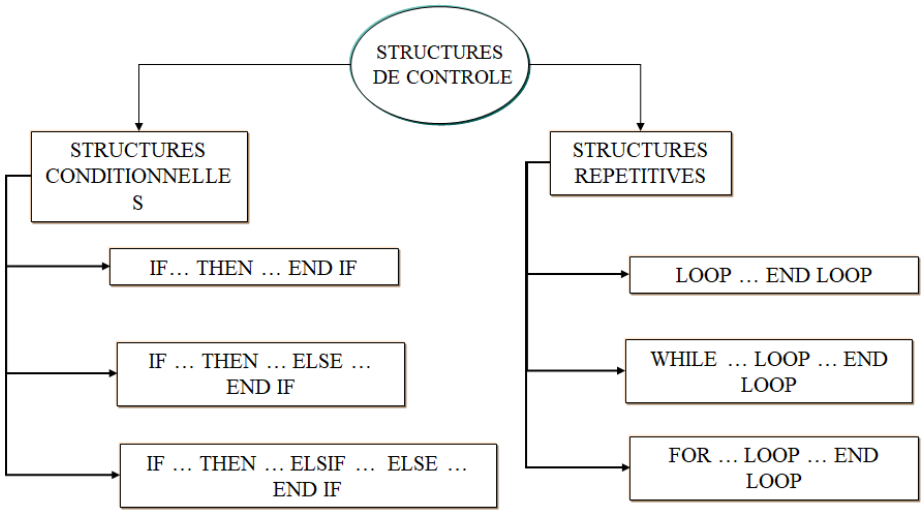
II)Traitements Conditionnels et Traitements Répétitifs

II.1 - Contrôler le Déroulement d'une Exécution PL/SQL

Modifier le déroulement logique des instructions en utilisant desstructures de contrôle

- Structures de contrôle conditionnel (Instruction IF)
- Structures de Contrôle Itératif
  - Boucle de base
  - Boucle FOR
  - Boucle WHILE
  - Instruction EXIT

II.2 - PARTIE EXECUTABLE



II.3 - L'Instruction IF et CASE - Syntaxe

On peut déclencher des actions en fonction du résultat de conditions

IF condition THEN
instructions ;
[ELSIF conditions THEN
instructions ;]
[ELSE
instructions ;]
END IF;

- ELSIF en un mot
- END IF en deux mots

- une seule clause ELSE est permise

Exemple

```
IF salaire < =1000 THEN
    nouveau_salaire := ancien_salaire + 100;
ELSEIF salaire > 1000 AND emp_id_emp = 1 THEN
    nouveau_salaire := ancien_salaire + 500;
ELSE    nouveau_salaire := ancien_salaire + 300;
END IF;
```

Choix

```
CASE expression
WHEN expr1 THEN instructions1;
WHEN expr2 THEN instructions2;
…
ELSE instructionsN;
END CASE;
```

Expression de type simple

II.4 - Instructions LOOP

- Les boucles répètent une instruction ou un ensemble d'instructions plusieurs fois.
- Trois types de boucles
  - Boucle de base
  - Boucle FOR
  - Boucle WHILE
- L'instruction EXIT permet de sortir de la boucle

Boucle de Base

```
LOOP                                    -- Début de boucle
    instruction1 ;                  -- Instructions
    . . .
    EXIT [ WHEN condition ];        -- Sortie de boucle
END LOOP ;                          -- Fin de boucle
```

Exercice

Ecrire un programme PL/SQL qui affiche les multiples de 3, 4 et 5 qui sont entre 4 et 32.

Solution

```
SET SERVEROUTPUT  ON  -- sous SQL pLUS
DECLARE
    i NUMBER(2) := 4;
BEGIN
```

```
        LOOP
                IF (MOD(i,3)=0) THEN
        DBMS_OUTPUT.PUT_LINE (i || ' est un multiple de 3');
                END IF;
                IF (MOD(i,4)=0) THEN
        DBMS_OUTPUT.PUT_LINE (i || ' est un multiple de 4');
                END IF;
                IF (MOD(i,5)=0) THEN
        DBMS_OUTPUT.PUT_LINE (i || ' est un multiple de 5');
                END IF;
                i := i+1;
                EXIT WHEN i>32;
        END LOOP;
    END;
/
```

II.5 - Boucle FOR - Exemple

Afficher le nombre de fois où la boucle est exécutée et la dernière valeur de l'index.

```
for i IN 1..100 LOOP

   somme := somme + i;

end loop;

/
```

II.6 - Boucle WHILE - Exemple

```
SET SERVEROUTPUT  ON  -- sous SQL pLUS
DECLARE
   i NUMBER(2) := 4;
BEGIN
   WHILE i<33   LOOP
           IF (MOD(i,3)=0) THEN
DBMS_OUTPUT.PUT_LINE (i || ' est un multiple de 3');
           END IF;
           IF (MOD(i,4)=0) THEN
DBMS_OUTPUT.PUT_LINE (i || ' est un multiple de 4');
           END IF;
           IF (MOD(i,5)=0) THEN
DBMS_OUTPUT.PUT_LINE (i || ' est un multiple de 5');
           END IF;
           i := i+1;
   END LOOP;
END;
/
```

Chapitre 3

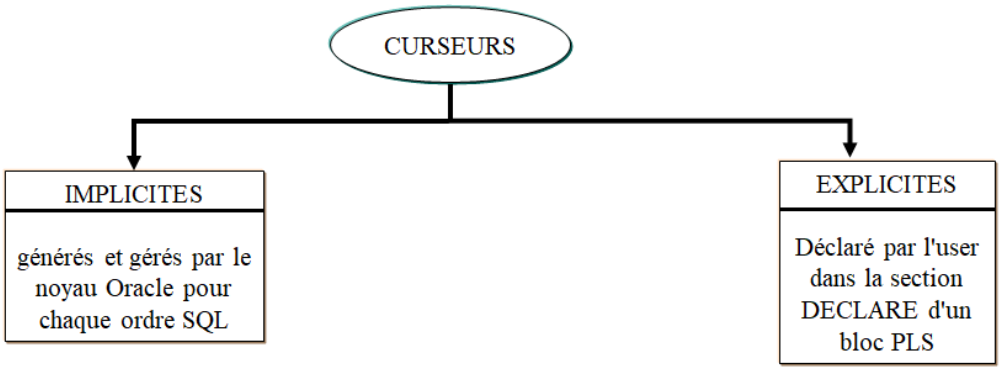
Curseurs

III) Curseurs

III.1 - Qu'est ce qu'un Curseur ?

- Le serveur Oracle utilises des zone de travail appelées *Zone Sql Privées* pour exécuter les instructions SQL et pour stocker les informations en cours de traitement.
- Vous pouvez utilisez des curseurs PL/SQL pour nommer une zone SQL privée et accéder aux données qu'elle contient.

- Une zone mémoire de taille fixe contenant le résultat d'une requête.
- Utilisée pour interpréter et analyser les ordres SQL.
- Le nombre de curseurs ouverts simultanément est défini par le paramètre OPEN_CURSORS. dans le PFILE de la base.



L'utilisation d'un curseur nécessite 4 étapes :

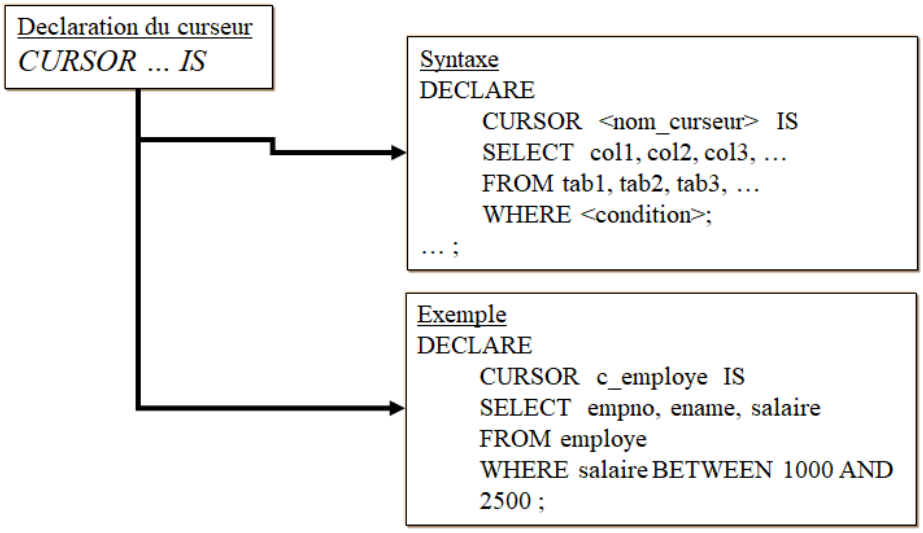
1. Déclaration du curseur : Section DECLARE
2. Ouverture du curseur : Section BEGIN
3. Traitement des lignes : Section BEGIN
4. Fermeture du curseur : Section BEGIN OU EXCEPTION

III.2 - La déclaration d'un curseur

- La déclaration du curseur permet de stocker l'ordre Select dans le curseur.
- Le curseur se définit dans la partie Declare d'un bloc PL/Sql.

Cursor nomcurseur IS Requete_SELECT ;

LES CURSEURS EXPLICITES



La déclaration d'un curseur

Declare
Cursor DEPT10 is
select ename, sal from emp where deptno=10 order by sal ;
Begin
... ...;
End ;

III.2.1) L'ouverture du curseur

- L'ouverture du curseur réalise :
    1. l'allocation mémoire du curseur
    2. l'analyse sémantique et syntaxique de l'ordre
    3. le positionnement de verrous éventuels (si select for update...)
- C'est seulement à l'ouverture du curseur que la requête SQL s'éxécute.
- L'ouverture du curseur se fait dans la section Begin du Bloc.

OPEN nomcurseur ;

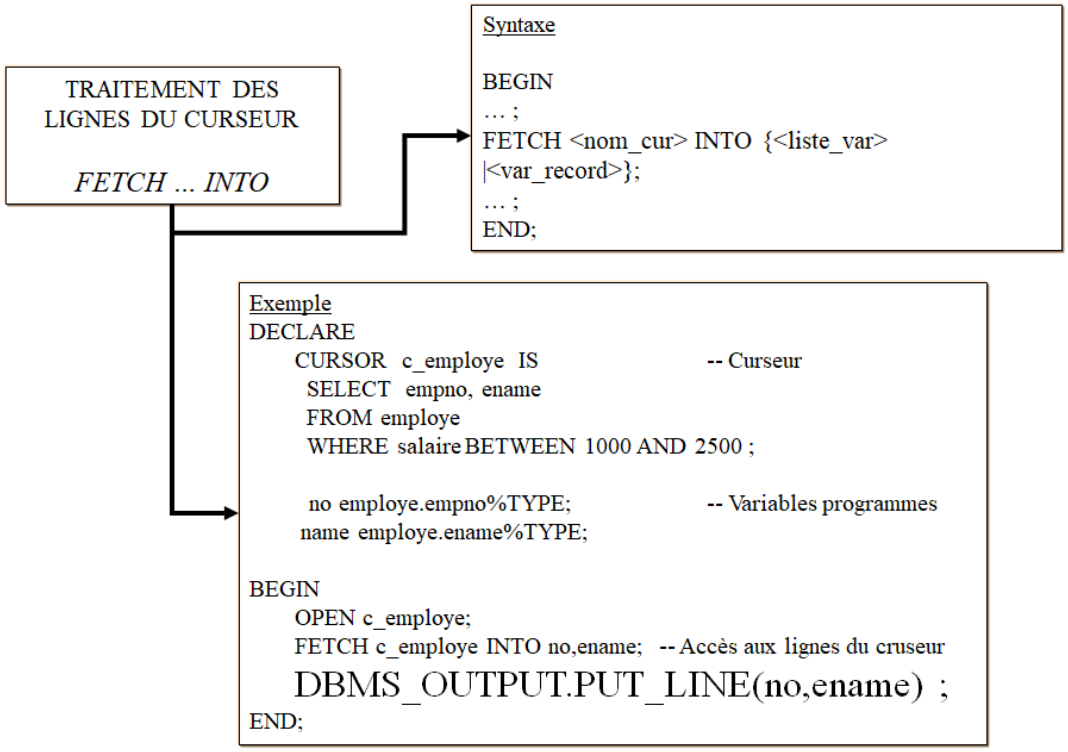
Exemple:

Declare
Cursor DEPT10 is
select ename, sal from emp where deptno=10 order by sal ;
Begin
Open DEPT10;
.....
End ;

III.3 - Traitement des 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 définie dans la partie Declare du bloc.
  - Le fetch ramène une seule ligne à la fois,
  - pour traiter n lignes il faut une boucle.

FETCH nomcurseur INTO liste_variables ou Nom_enregistrement;



Exemple :

create table resultat (nom1 char(10), sal1 number(7,2))
/
Declare
Cursor DEPT10 is select ename, sal from emp where deptno=20 order by sal ;
-- variables réceptrices
nom emp.ename%TYPE; -- Variable locale de même type que le champ ename
salaire emp.sal%TYPE;
Begin
Open DEPT10;
Fetch DEPT10 into nom, salaire ; -- Lecture 1° tuple
WHILE DEPT10%found loop -- Tant qu'on trouve une ligne

If salaire > 2500 then
insert into resultat values (nom,salaire);
end if;
Fetch DEPT10 into nom,salaire ; -- Lecture tuple suivant
end loop;
Close DEPT10;
End ;
/

Attributs d'un Curseur Explicite

Obtenir des informations sur le curseur en utilisant les attributs de curseur.

Attribut	Type	Description
%ISOPEN	Booléen	Evalué à TRUE si le curseur est ouvert.
%NOTFOUND	Booléen	Evalué à TRUE si le dernier fetch n'a ramené aucune ligne
%FOUND	Booléen	Evalué à TRUE tant que le fetch ramène des lignes
%ROWCOUNT	Numérique	Evalué au nombre total de lignes ramenées jusqu'à présent.

- Les attributs d'un curseur sont des indicateurs sur l'état d'un curseur. Ils nous fournissent des informations quant à l'exécution de l'ordre.
- Elles sont conservées par Pl/Sql après l'exécution du curseur.

III.4 - La fermeture du curseur

- Après le traitement des lignes, l'étape de fermeture permet d'effectuer la libération de la place mémoire.

CLOSE nomcurseur; *Close dept10 ;*

Exemple :

create table resultat (nom1 char(35), sal1 number(8,2))
/
Declare
*Cursor C1 is select * from pilote where adresse='Paris';*
-- variable réceptrice
unpilot pilote%rowtype;
Begin

```
Open C1;
Fetch c1 into unpilot ; -- Lecture 1° tuple
WHILE C1%found
loop
If unpilot.comm is not null then
insert into resultat values (unpilot.nompilot, unpilot.salpilot);
end if;
Fetch c1 into unpilot ; -- Lecture tuple suivant
end loop;
Close c1;
End ;
/
```

Boucle LOOP pour un curseur

```
BEGIN
open salaires;
loop
    fetch salaires into salaire;
    exit when salaires%notfound;
    if salaire is not null then
        total := total + salaire;
        DBMS_OUTPUT.put_line(total);
    end if;
end loop;
close salaires; -- Ne pas oublier
DBMS_OUTPUT.put_line(total);
END;
```

Boucle FOR pour un curseur

```
declare
cursor c is
select dept, nome from empwhere dept = 10;
employe c%rowtype;
begin
    FOR employe IN c LOOP
        dbms_output.put_line(employe.nome);
    END LOOP;
end;
```

Chapitre 4

Gestion des Erreurs

IV) Gestion des Erreurs (EXCEPTION)

IV.1 - Exception

- Une exception est une erreur qui survient durant une exécution
- 2 types d'exception :
  - **Interne**
    - **exception oracle pré-définie**
    - **exception oracle non pré-définie**
  - **Externe (exception définie par l'utilisateur)**
- **Les exceptions internes** sont générées par le moteur du système (division par zéro, connexion non établie, table inexistante, privilèges insuffisants, mémoire saturée, espace disque insuffisant, ...).
  - Une erreur interne est produite quand un bloc PL/Sql viole une règle d'Oracle ou dépasse une limite dépendant du système d'exploitation.
  - Chaque erreur ORACLE correspond un code SQL (SQLCODE)
- **Les exceptions externes** sont générées par l'utilisateur (stock à zéro, ...).

EXCEPTION	DESCRIPTION	TRAITEMENT
Erreur pré-définie du oracle	Une des 20 erreurs qui arrivent le plus fréquemment en langage P/SQL	Ne pas la déclarer et laisser oracle serveur l'émettre implicitement
Erreur non pré-définie du oracle	Toutes les autres erreur standard d'oracle	La déclarer à l'intérieur de la section declare et laisser oracle l'émettre implicitement
Erreur définie par l'utilisateur	Une condition que le programmeur définit comme anomalie	La déclarer à l'intérieur de la section declare et son émission est explicite

IV.2 - Intercepter les Exceptions - Syntaxe

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
statement1;
   statement2;
   . . .
```

[WHEN *exception3* [OR *exception4* . . .] THEN
statement1;
statement2;
. . .]
[WHEN OTHERS THEN
statement1;
statement2;
. . .]

IV.3 - Règles pour intercepter les Exceptions

- Le mot clé EXCEPTION débute la section de la gestion des exceptions
- Plusieurs exceptions sont permises (définie et pré-définie)
- Une seule exception est exécutée avant de sortir d'un bloc
- WHEN OTHERS est la dernière clause
  - Intercepte toutes les exceptions non gérées dans la même section d'exception
  - Utilisez le gestionnaire d'erreurs OTHERS et placez le en dernier lieu après tous les autres gestionnaire d'erreurs, sinon il interceptera toutes les exceptions mêmes celle qui sont prédéfinies.

Les erreurs internes d'Oracle pré-définies

- Utiliser le nom standard à l'intérieur de la section Exception
- Les noms des exceptions pré-définies oracle sont regroupées dans ce tableau :

Nom d'exception	Erreur ORACLE	SQLCODE
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
STORAGE_ERROR	ORA-06500	-6500
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

- CURSOR_ALREADY_OPEN : tentative d'ouverture d'un curseur déjà ouvert..
- DUP_VAL_ON_INDEX: insertion d'une ligne en doublon
- INVALID_CURSOR : opération incorrecte sur un curseur, comme par exemple la fermeture d'un curseur qui n'a pas été ouvert.
- LOGIN_DENIED : connexion à la base échouée car le nom utilisateur ou le mot de passe est invalide.
- NO_DATA_FOUND : déclenché si la commande SELECT INTO ne retourne aucune ligne ou si on fait référence à un enregistrement non initialise d'un tableau PL/SQL.
- NOT_LOGGED_ON : tentative d'accès à la base sans être connecté.
- PROGRAM_ERROR : problème général dû au PL/SQL.
- ROWTYPE_MISMATCH : survient lorsque une variable curseur d'un programme hôte retourne une valeur dans une variable curseur d'un bloc PL/SQL qui n'a pas le même type.
- STORAGE_ERROR : problème de ressources mémoire dû à PL/SQL.
- TIMEOUT_ON_RESOURCE : dépassement du temps dans l'attente de libération des ressources (lié aux paramètres de la base).
- TOO_MANY_ROWS : la commande SELECT INTO retourne plus d'une ligne.
- VALUE_ERROR : erreur arithmétique, de conversion, ou de contrainte de taille.
- ZERO_DIVIDE : tentative de division par zéro.

Exemple

```
DECLARE
    v_sal emp.sal%type;
BEGIN
    SELECT sal  INTO v_sal  from emp;
EXCEPTION
    WHEN TOO_MANY_ROWS then ... ;
    -- gérer erreur trop de lignes
    WHEN NO_DATA_FOUND then ... ;
    -- gérer erreur pas de ligne
    WHEN OTHERS then ... ;
    -- gérer toutes les autres erreurs
END ;
```

IV.4 - Exceptions Oracle Non Prédéfinies

Vous pouvez intercepter une erreur oracle non pré-définie en la déclarant en préalable, ou en utilisant la commande OTHERS, L'exception déclarée et implicitement déclenchée

1. Déclarer le nom de l'exception oracle non-prédéfinie

   Syntaxe exception_nom EXCEPTION;

2. Associer l'exception déclarée au code standard de l'erreur oracle en utilisant l'instruction

   Syntaxe PROGMA EXCEPTION_INIT (exception, erreur_number)

3. Traiter l'exception ainsi déclarée dans la section EXCEPTION

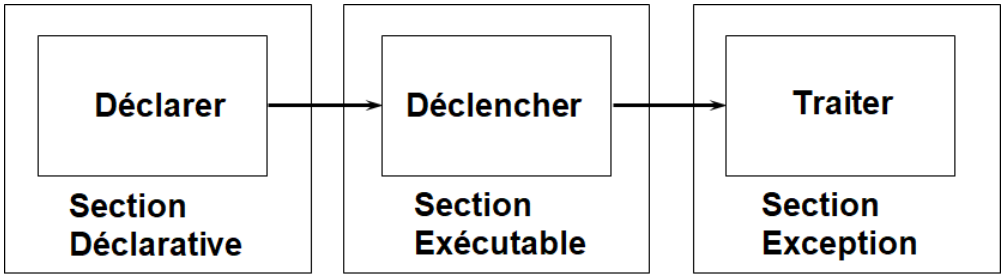
Exemple

Capturer l'erreur du Serveur Oracle numéro -2292 correspondant à la violation d'une contrainte d'intégrité.

```
DECLARE
e_emps      EXCEPTION;
PRAGMA EXCEPTION_INIT (e_products_remaining, -2292);
v_deptno dept.deptno%type:=&p_deptno
BEGIN
Delete from dept
Where deptno=v_deptno;
Commit;
EXCEPTION
WHEN e_emps THEN
dbms_output.pu_line('Suppression impossible du dept : ' || to_char(v_deptno) ||
'employés existant ');
END;
```

IV.5 - Exceptions Utilisateur (externes)

- PL/SQL permet à l'utilisateur de définir ses propres exceptions.
- La gestion des anomalies utilisateur peut se faire dans un bloc PL/SQL en effectuant les opérations suivantes :
- Sont définies dans la section **DECLARE**
- Sont déclenchées explicitement dans la section **BEGIN** par l'instruction **RAISE**
- Dans la section **EXCEPTION**, référencer le nom défini dans la section DECLARE.



1. Nommer l'erreur (type exception) dans la partie Declare du bloc.

```
DECLARE
        Nom_ano Exception;
```

2. Déterminer l'erreur et passer la main au traitement approprié par la commande **Raise.**

```
BEGIN
If (condition_anomalie) then raise Nom_ano ;
```

3. Effectuer le traitement défini dans la partie EXCEPTION du Bloc.

```
EXCEPTION
WHEN (Nom_ano) then (traitement);
```

```
DECLARE
        ...
        Nom_ano EXCEPTION;
BEGIN
        instructions ;
        IF (condition_anomalie) THEN RAISE Nom_ano;
        ...
EXCEPTION
        WHEN Nom_ano THEN (traitement);
END ;
```

On sort du bloc après l'exécution du traitement d'erreur.

Exemple :

DECLARE
Erreur_comm exception ;

```
v_pilot pilote%rowtype ;
BEGIN
Select * into v_pilot From PiloteWhere nopilot = '7100' ;
If  v_pilot.comm > v.pilot.sal Then
Raise erreur_comm ;
.....
EXECPTION
When erreur_comm then
Insert into erreur values(v_pilot.nom, ' Commission > salaire') ;
When NO_DATA_FOUND Then
Insert into erreur values(v_pilot.nopilot, ' non trouvé') ;
END ;
```

IV.6 - Fonctions d'interception des erreurs

- SQLCODE
  - Renvoie la valeur numérique associé au code de l'erreur.
  - Vous pouverz l'assigneer à une variable de type number
- SQLERRM
  - Renvoie le message associé au code de l'erreur.

Exemple de SQLCODE

Valeur de SQLCODE	Description
0	Pas d'exception enregistrée
1	Exception définie par l'utilisateur
+100	Exception NO_DATA_FOUND
Négative number	Autre code d'erreur oracle

IV.7 - Fonctions d'interception des erreurs- Exemple

Lorsqu'une exception est intercetée par la clause WHEN OTHERS, vous pouvez utiliser un ensemble de fonctions standard pour identifier l'erreur.

```
DECLARE
v_error_code     NUMBER;
v_error_message   VARCHAR2(255);
BEGIN
        . . .
```

```
        EXCEPTION
        . . .
        WHEN OTHERS THEN
        ROLLBACK;
          v_error_code := SQLCODE;
        v_error_message := SQLERRM;
        insert into erreur values(v_error_code, v_error_message);
END;
```

Chapitre 5

PROCEDURES, FONCTIONS ET PACKAGES

V) PROCEDURES, FONCTIONS ET PACKAGES

V.1 - Généralité

- Une **procédure est un bloc PL/SQL nommé**.
- Une **fonction est une procédure qui retourne une valeur.**
- Un **package est un agrégat de procédures et de fonctions.**
- Les packages, procédures, ou fonctions peuvent être appelés depuis toutes les applications qui possèdent une interface avec ORACLE (SQL*PLUS, Pro*C, SQL*Forms, ou un outil client particulier comme NSDK par exemple).
- Les procédures (fonctions) permettent de :
    - Réduire le trafic sur le réseau (les procédures sont locales sur le serveur)
    - Mettre en œuvre une architecture client/serveur de procédures et rendre indépendant le code client de celui des procédures (à l'API près)
    - Masquer la complexité du code SQL (simple appel de procédure avec passage d'arguments)
    - Sécuriser l'accès aux données (accès à certaines tables seulement à travers les procédures)
    - Optimiser le code (les procédures sont compilées avant l'exécution du programme et elles sont exécutées immédiatement si elles se trouvent dans la SGA (zone mémoire gérée par ORACLE). De plus une procédure peut être exécutée par plusieurs utilisateurs.
- Les packages permettent :
    - de regrouper des procédures ou des fonctions (ou les deux). On évite ainsi d'avoir autant de sources que de procédures.
    - De travailler en équipes et l'architecture applicative peuvent donc plus facilement s'organiser du côté serveur, où les packages regrouperont des procédures à forte cohésion intra (Sélection de tous les articles, Sélection d'un article, Mise à jour d'un article, Suppression d'un article, Ajout d'un article).
- Les packages sont utilisés comme de simples librairies par les programmes clients (librairies distantes « sur le serveur »)
- Les équipes de développement doivent prendre garde à ne pas travailler chacune dans « leur coin ».

- Les développeurs ne doivent pas perdre de vue la logique globale de l'application et les scénarios d'activité des opérateurs de saisie.
    - A l'extrême, on peut finir par coder une procédure extrêmement sophistiquée qui n'est sollicitée qu'une fois par an pendant une seconde. Ou encore, une gestion complexe de verrous pour des accès concurrent qui n'ont quasiment jamais lieu.

V.2 - Procédures

Les procédures ont un ensemble de paramètres modifiables en entrée et en sortie.

CREATE [OR REPLACE] PROCEDURE [<user>].<nom_proc>

(arg1 IN type1 [DEFAULT val_initiale [, arg2 OUT type2 [, arg3 IN OUT type3, …])

AS

[Déclarations des variables locales]

EXCEPTION

BEGIN

Argument	Signification
IN	Valeur par défaut. Argument en entrée. Elle ne être modifiée par la procédure.
OUT	Argument en sortie (modifié par la procédure).
IN OUT	Argument en entrée sortie. Elle peut être lue et modifiée par la procédure.
TYPE	Type de l'argument sans spécification de la taille
DEFAULT	Affecte une valeur par défaut à l'argument.
IS	Permet la définition de la procédure.

Procédure Exemple

Compter le nombre d'employés pour un département donné.

```
CREATE OR REPLACE PROCEDURE proc_dept (p_no IN dept.deptno%TYPE)
IS
v_no NUMBER;
BEGIN
```

```
        SELECT  COUNT(deptno)
        INTO v_no
        FROM  emp
        WHERE deptno=p_no;
        DBMS_OUTPUT.PUT_LINE('Nombre d'employés : '||'  '||v_no);
END;
/
```

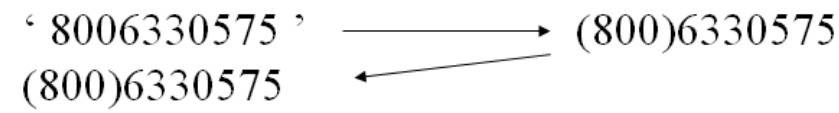
- Exemple de procédure qui modifie le salaire d'un employé.
  - Arguments : Identifiant de l'employée, Taux

modifie_salaire.sql

```
create procedure modifie_salaire (id in number, taux in number)
is
begin
    update employe set salaire=salaire*(1+taux)
    where Id_emp= id;
    exception
    when no_data_found then
    DBMS_OUTPUT.PUT_LINE('Employé inconnu : ' ||to_char(id));
End;
/
```

V.2.1) PROCEDURES / PARAMETRES

EXEMPLE IN OUT




```
CREATE OR REPLACE PROCEDURE format_phone(v_phone_no IN OUT VARCHAR2 (12))
IS
BEGIN
  v_phone_no := ' (' ||substr(v_phone_no,1,3)|| ') '||substr(v_phone,4,7)
END format_phone ;
/
```

Compilation de la procédure modifie_salaire

Il faut compiler le fichier sql qui s'appelle ici modifie_salaire.sql (attention dans cet exemple le nom du script correspond à celui de la procédure, c'est bien le nom du script sql qu'il faut passer en argument à la commande start).

```
SQL>start modifie_salaire
        Procedure created.
```

Appel de la procédure modifie_salaire

```
begin
```

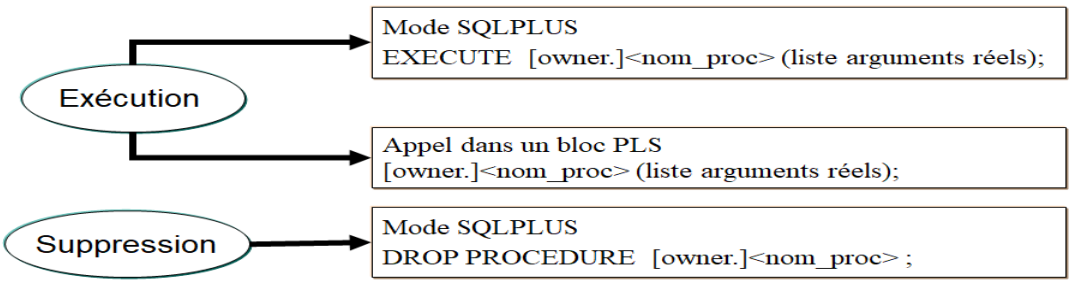
```
        modifie_salaire (15,-0.5);
end;
/
```

- L'utilisation d'un script qui contient les 4 lignes précédentes est bien sûr également possible : demarre.sql

```
begin
modifie_salaire (15,-0.5);
end;
/
```

- Lancement du script demarre.sql :

```
SQL> start demarre
```



V.2.2) Correction des erreurs

- Si le script contient des erreurs, la commande ***show err permet de visualiser les erreurs.***

- Pour visualiser le script global :
  - commande l (lettre l)
  - pour visualiser la ligne 4 : commande l4

```
SQL> start modifie_salaire

Warning: Procedure created with compilation errors.

SQL> show err
Errors for PROCEDURE MODIFIE_SALAIRE:

LINE/COL ERROR
-------- -----------------------------------------------------------
4/8      PLS-00103: Encountered the symbol "VOYAGE" when expecting one of the
following:
         := . ( @ % ;
         Resuming parse at line 5, column 21.

SQL> l4

  4*    update employe set salaire=salaire*(1+taux)
```

V.3 - Fonctions

- Une fonction est une procédure qui retourne une valeur. La seule différence syntaxique par rapport à une procédure se traduit par la présence du mot clé RETURN.
- Une fonction précise le type de donnée qu'elle retourne dans son prototype (signature de la fonction).
- Le retour d'une valeur se traduit par l'instruction RETURN (valeur).

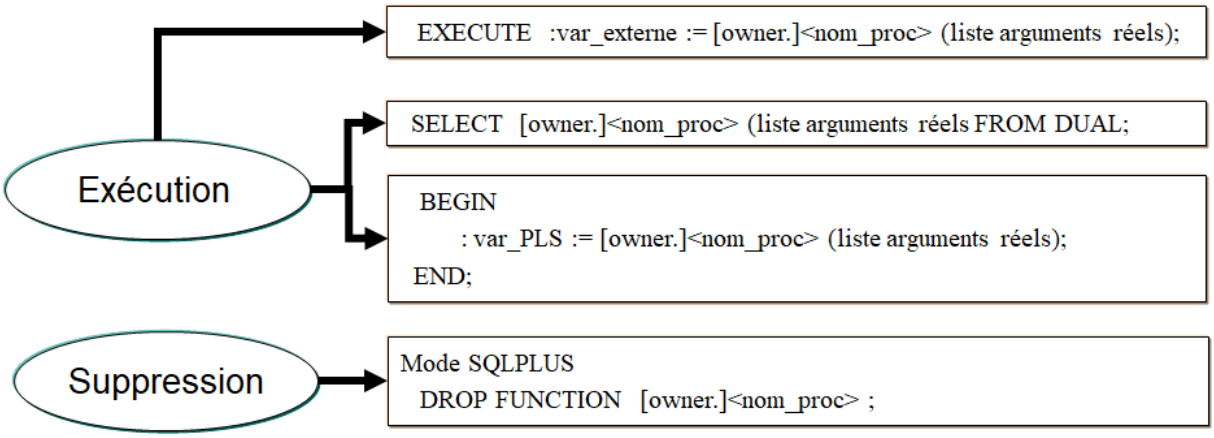
```
CREATE  [ OR  REPLACE ] FUNCTION  [<user>].<nom_proc>
    (arg1 IN type1 [DEFAULT val_initiale [, arg2 IN type2, …])
    RETURN  type_retour AS
            [ Déclarations des variables locales ]
BEGIN
            Contenu du bloc PLS
            RETURN  (var_retour );
Exception
END  [<nom_proc>];
/
```

Fonction -Exemple

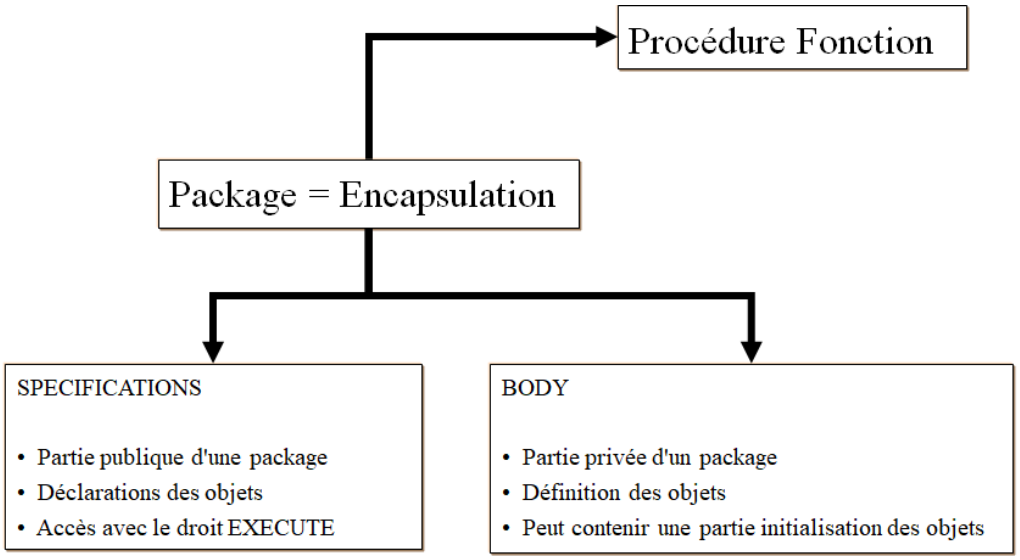
Compter le nombre d'employés pour un département donné.

```
CREATE  OR  REPLACE  FUNCTION   proc_dept (p_no IN dept.deptno%TYPE)
RETURN NUMBER AS
            v_no NUMBER;
    BEGIN
            SELECT  COUNT(deptno)
            INTO v_no
            FROM  emp
            WHERE deptno=p_no;
            RETRUN (v_no);
    END;
    /
```

LES FONCTIONS - Opérations de base



V.4 - LES PACKAGES



V.4.1) La structure Générale d'un package

package_general.sql

CREATE OR REPLACE PACKAGE ***nom_package IS***
définitions des types utilisés dans le package;
prototypes de toutes les procédures et fonctions du package;
END ***nom_package;***
/

CREATE OR REPLACE PACKAGE BODY nom_package IS
déclaration de variables globales;
définition de la première procédure;
définition de la deuxième procédure;
etc. ...
END nom_package;
/

Un package est composé d'un en tête et d'un corps :

- ✓ L'en tête comporte les types de données définis et les prototypes de toutes les procédures (fonctions) du package.
- ✓ Le corps correspond à l'implémentation (définition) des procédures (fonctions).

- Le premier END marque la fin de l'en tête du package. Cette partie doit se terminer par / pour que l'en tête soit compilé.
- Le corps (body) du package consiste à implémenter (définir) l'ensemble des procédures ou fonctions qui le constitue. Chaque procédure est définie normalement avec ses clauses BEGIN ... END.

Le package se termine par / sur la dernière ligne.

exemple

- Package *paquet1*comportant deux procédures (augmentation de salaire et suppression de vendeur) :

```
create or replace package ges emp is

procedure augmente salaire (v Id emp in number,
                            v taux salaire in number);

function moyenne_salaire (v_Id_employe IN NUMBER)
return NUMBER;

end ges emp;
/

create or replace package body ges_emp is

procedure augmente salaire (v Id emp in number,
                            v taux salaire in number)
is

begin

      update employe set salaire= salaire * v taux salaire
      where Id emp= v Id emp;
      commit;

end augmente_salaire;


function moyenne salaire (v Id employe IN NUMBER)
return NUMBER
is
valeur NUMBER;

begin

      select avg(salaire)
      into valeur
      from employe
      groupe by emp id emp;
      return (valeur);
end moyenne_salaire;


end ges_emp;
/
```

Compilation du package paquet1

SQL> @paquet1
Package created.
Package body created

V.4.2) Opérations sur les packages

Exécution de la procédure augmente_salaire du package ges_emp

SQL> begin
2 ges_emp.augmente_salaire(4,50);
3 end;
4 /
PL/SQL procedure successfully completed.

- Exécution de la procédure augmente_salaire du package ges_emp

SQL> begin
2 ges_emp.augmente_salaire(4,50);
3 end;
4 /
PL/SQL procedure successfully completed.

Chapitre 6

Triggers

VI) Déclencheurs (TRIGGERs)

VI.1 - Définition

Les triggers sont des simples procédures stockées qui s'exécutent implicitement lorsqu'une instruction INSERT, DELETE ou UPDATE porte sur la table (ou dans certains, cas sur la Vue associée).

VI.2 - Syntaxe

CREATE TRIGGER nom
BEFORE DELETE OR INSERT OR UPDATE ON tablename
[FOR EACH ROW WHEN (condition)]
DECLARE ..... <<<<déclarations>>>>
BEGIN
..... <<<< bloc d'instructions PL/SQL>>>>
END;

VI.3 - Types de déclencheurs

- ORACLE propose deux types de triggers:
    1. **les triggers de table (STATEMENT)**
        - sont déclenchées une seule fois.
    2. **les triggers de ligne (ROW).**
        - se déclenchent individuellement pour chaque ligne de la table affectée par le trigger,
- Si l'option FOR EACH ROW est spécifiée, c'est un trigger ligne, sinon c'est un trigger de table.
- doit être unique dans un même schéma
    1. peut être le nom d'un autre objet (table, vue, procédure) mais à éviter.

VI.4 - Option BEFOR/AFTER

- elle précise le moment quand ORACLE déclenche le trigger,
- les triggers AFTER row sont plus efficaces que les BEFORE row parce qu'ils ne nécessitent pas une double lecture des données.

- Elle comprend le type d'instruction SQL qui déclenche le trigger :
    - DELETE, INSERT, UPDATE
    - On peut en avoir une, deux ou les trois.
- Pour UPDATE, on peut spécifier une liste de colonnes. Dans ce cas, le trigger ne se déclenchera que si l'instruction UPDATE porte sur l'une au moins des colonnes précisée dans la liste.
    - S'il n'y a pas de liste, le trigger est déclenché pour toute instruction UPDATE portant sur la table.

Les triggers lignes

- Pour les triggers lignes, on peut introduire une restriction sur les lignes à l'aide d'une expression logique SQL : c'est la clause WHEN :
    - Cette expression est évaluée pour chaque ligne affectée par le trigger.
    - Le trigger n'est déclenché sur une ligne que si l'expression WHEN est vérifiée pour cette ligne.
    - L'expression logique ne peut pas contenir une sous-question.
    - Par exemple, WHEN (:new.empno>0) empêchera l'exécution du trigger si la nouvelle valeur de EMPNO est 0, négative ou NULL.

VI.5 - Le corps du trigger

- Le corps du trigger est un bloc PL/SQL :
    - Il peut contenir du SQL et du PL/SQL.
    - Il est exécuté si l'instruction de déclenchement se produit et si la clause de restriction WHEN, le cas échéant, est évaluée à vrai.

Exemple1 de trigger table

```
CREATE TRIGGER log AFTER INSERT OR UPDATE
ON Emp
BEGIN
    INSERT INTO log (table, date, username, action)
    VALUES ('Emp', sysdate, sys_context ('USERENV','CURRENT_USER'),
    'INSERT/UPDATE on Emp') ;
END ;
```

Exemple2 de trigger table

```
CREATE OR REPLACETRIGGER PERSON_UPDATE_SALAIREBEFORE UPDATE
ON Employe
BEGIN
```

```
        DBMS_OUTPUT.PUT_LINE(' Avant la mise à jour de quelque employé');
    END;
```

Maintenant, exécutant update...

UPDATE Employe SET sal= sal= sal+(sal*0.1);

SQL> UPDATE Employe SET sal= sal+(sal*0.1);

Avant la mise à jour de quelque employé

2 rows updated.

VI.6 - Les noms de corrélation (OLD/New)

- Lors de la création de triggers lignes, il est possible d'avoir accès à la valeur ancienne et la valeur nouvelle grâce aux mots clés OLD et NEW.
  - Il n'est pas possible d'avoir accès à ces valeurs dans les triggers de table.
- Si l'instruction de déclenchement du trigger est INSERT, seule la nouvelle valeur a un sens.
- Si l'instruction de déclenchement du trigger est DELETE, seule l'ancienne valeur a un sens.
- La nouvelle valeur est appelée :new.colonne
- L'ancienne valeur est appelée :old.colonne
  - Exemple : IF :new.salaire < :old.salaire then

<u>Exemple1 de trigger row</u>

```
CREATE OR REPLACETRIGGER  Employe_UPDATE_Salaire
BEFORE UPDATE
ON Employe
FOR EACH ROW
BEGIN
DBMS_OUTPUT.PUT_LINE('Avant la mise à jour ' ||
TO_CHAR(:OLD.sal) || ' vers ' || TO_CHAR(:NEW.sal));
END;
```

Maintenant, exécutant update...

```
SQL> UPDATE Employe SET sal= sal+(sal*0.5);
Avant la mise à jour  1000 vers 1500
Avant la mise à jour  2000 vers 3000
Avant la mise à jour  4000 vers 6000
   3 rows updated.
```

<u>Exemple2 de trigger ligne</u>

```
CREATE OR REPLACE TRIGGER difference_salaire
BEFORE UPDATE ON Emp
FOR EACH ROW
WHEN (:new.Empno > 0)
DECLARE
sal_diff number;
BEGIN
sal_diff := :new.sal - :old.sal;
dbms_output.put(' Old : ' || :old.sal || 'New : ' || :new.sal || 'Difference : ' || sal_diff);
END ;


drop SEQUENCE Employe$S1;
CREATE SEQUENCE Employe$S1  START WITH 1 MINVALUE 1 MAXVALUE 9999999999
NOCACHE;
CREATE OR REPLACE TRIGGER Employe$T1
 BEFORE INSERT ON Employe
 FOR EACH ROW
BEGIN
        If DBMS_REPUTIL.FROM_REMOTE = False And DBMS_SNAPSHOT.I_AM_A_REFRESH =
False Then
                IF :New.ID IS  NULL THEN
                        SELECT Employe$S1.NEXTVAL INTO :New.ID
                    FROM DUAL;
                END IF;
        END IF;
END;
/
```

VI.7 - Les prédicats conditionnels INSERTING, DELETING et UPDATING

Quand un trigger comporte plusieurs instructions de déclenchement (par exemple INSERT OR DELETE OR UPDATE), on peut utiliser des prédicats conditionnels (INSERTING, DELETING et UPDATING) pour exécuter des blocs de code spécifiques pour chaque instruction de déclenchement.

Exemple :

```
CREATE TRIGGER ...
BEFORE INSERT OR UPDATE ON employe
.....
BEGIN
```

.....
IF INSERTING THEN ..... END IF;
IF UPDATING THEN ..... END IF;
.....
END;

UPDATING peut être suivi d'un nom de colonne :

CREATE TRIGGER ...
BEFORE UPDATE OF salaire, commission ON employe
.....
BEGIN
.....
IF UPDATING ('salaire') THEN ..... END IF;
.....
END;

On peut avoir au maximum un trigger de chacun des types suivants pour chaque table :

BEFORE UPDATE row
BEFORE DELETE row
BEFORE INSERT row
BEFORE INSERT statement
BEFORE UPDATE statement
BEFORE DELETE statement
AFTER UPDATE row
AFTER DELETE row
AFTER INSERT row
AFTER INSERT statement
AFTER UPDATE statement
AFTER DELETE statement.

Même pour UPDATE, on ne peut pas en avoir plusieurs avec des noms de colonnes différents.

Références Bibliographiques

Ian ABRAMSON, Michael ABBEY, Michael COREY, Oracle 10g: Notions Fondamentales, Oracle Press.

Olivier Heurtel, Oracle 10g Installation du serveur sous Windows, Linux, Oracle, Eni Editions.