afpa
La formation professionnelle

Secteur Tertiaire Informatique
Filière étude - développement

Activité « Développer la persistance des données »

Langage PL/SQL

Accueil

Apprentissage

Période en entreprise

Evaluation

Code barre

SOMMAIRE

I INTRODUCTION

Le PL/SQL est le langage procédural d'ORACLE. Il constitue une extension au SQL qui est le langage de requêtes.

L'objectif du PL/SQL est de pouvoir mélanger la puissance des instructions SQL avec la souplesse d'un langage procédural dans un même traitement.

Ces traitements peuvent être exécutés, soit directement par les outils ORACLE (bloc anonyme), soit à partir d'objets de la base de données (Procédures stockées, fonctions et triggers).

II INSTRUCTIONS PL/SQL

II.1 INSTRUCTIONS SQL INTEGREES DANS PL/SQL

Ces instructions sont utilisables avec pratiquement la même syntaxe qu'en SQL :

- La partie interrogation : SELECT.
- La partie manipulation : INSERT, UPDATE, DELETE.
- La partie gestion des transactions : COMMIT, ROLLBACK, SAVEPOINT…
- Les fonctions comme TO_CHAR, TO_DATE, UPPER, SUBSTR, ROUND…

Les principaux éléments complémentaires de PL/SQL sont les commentaires, les variables, les fonctions, et les instructions de contrôle du déroulement.

II.2 INSTRUCTIONS SPECIFIQUES AU PL/SQL

Les caractéristiques procédurales de PL/SQL apportent les possibilités suivantes :

- La gestion des variables (déclaration, affectation, utilisation).
- Les structures de contrôle (séquences, tests, boucles).

Des fonctionnalités supplémentaires sont disponibles :

- La gestion des curseurs (traitements du résultat d'une requête ligne par ligne)
- Les traitements d'erreurs (déclaration, action à effectuer).

II.3 LE BLOC PL/SQL

PL/SQL n'interprète pas une commande, mais un ensemble de commandes contenues dans un « bloc » PL/SQL.
Ce bloc est compilé et exécuté par le moteur de la base de données ou de l'outil utilisé.

Un bloc est organisé en trois sous-ensembles de code.

```
DECLARE
```

```
/* Déclaration des variables, des constantes, des exceptions et des curseurs */
BEGIN [nom du bloc]
/* Instruction SQL, PL/SQL, structures de contrôle */
../..
EXCEPTION
/* Traitement des erreurs */
END [nom du bloc] ;
```

Les **commentaires en ligne** sont situés après deux tirets -- ;
Un commentaire en ligne peut être placé sur la même ligne qu'une instruction, à la suite de l'instruction ou bien en début de ligne, toute la ligne constituant alors le commentaire. Si le commentaire nécessite plusieurs lignes, il faut répéter les tirets sur chaque ligne.
Il est également possible de créer un bloc de commentaires en plaçant /* en début et */ fin de bloc

II.4 GESTION DES VARIABLES

Les variables sont des zones mémoires nommées permettant de stocker une information (donnée).
En PL/SQL, elles permettent de stocker des valeurs issues de la base ou de calculs, afin de pouvoir effectuer des tests, des calculs ou des valorisations d'autres variables ou de données de la base.
Les variables sont définies de la manière suivante :

- Le nom : Composé de lettre, chiffres, $, _ ou #, avec un maximum de 30 caractères. Ce ne doit pas être un mot réservé ORACLE.
- Le type : Détermine le format de stockage de l'information et de sont utilisation.

Toute variable doit être déclarée avant utilisation. Comme en SQL, PL/SQL n'est pas sensible à la casse. Les noms des variables peuvent être saisis en majuscule ou en minuscule.
Les instructions se terminent par « ; »

II.5 LES TYPES

Il existe plusieurs catégories de types de variables :

- simples (INTEGER, NUMBER, DATE, CHAR, BOOLEAN, VARCHAR, ...)
- composés (RECORD, TABLE, VARRAY , NESTED TABLE)
- référence
- pointeur de LOB (CLOB, BLOB, BFILES, NCLOB)

II.6 LES TYPES SIMPLES

Les types simples dans PL/SQL sont nombreux. Les principaux sont :

- Pour les **types numériques** : REAL, INTEGER, NUMBER (précision de 38 chiffres par défaut), NUMBER(x) (nombres avec x chiffres de précision), NUMBER(x,y) (nombres avec x chiffres de précision dont y après la virgule).
- Pour les **types alphanumériques** : CHAR(x) (chaine de caractère de longueur fixe x), VARCHAR(x) (chaine de caractère de longueur variable jusqu'à x), VARCHAR2 (idem que précédent excepté que ce type supporte de plus longues chaines et que l'on n'est pas obligé de spécifier sa longueur maximale). PL/SQL permet aussi de manipuler des dates (type DATE) sous différents formats.

Une autre spécificité du PL/SQL est qu'il permet d'assigner comme type à une variable celui d'un champ d'une table (par l'opérateur %TYPE) ou d'une ligne entière (Opérateur %ROWTYPE). Dans la déclaration suivante :

Exemple :

```
DECLARE
nom emp.name%TYPE;
employe emp%ROWTYPE;
```

La variable nom est définie comme étant du type de la colonne « name » de la table emp (qui doit exister au préalable). De même, employe est un vecteur du type d'une ligne de la table emp. A supposer que cette dernière ait trois champs numero, name, age de type respectifs NUMBER, VARCHAR, INTEGER, la variable employe disposera de trois composantes : employe.numero, employe.name, employe.age, de même types que ceux de la table.

II.7 LES SOUS-TYPES

PL/SQL propose des sous-types synonymes des types simples. Ces sous-types permettent d'assurer la compatibilité avec les types standards ANSI/ISO et IBM.

Type ORACLE	Sous-type
NUMBER	DEC, DECIMAL, NUMERIC, DOUBLE PRECISION, FLOAT, REAL INTEGER, INT, SMALLINT
BINARY_INTEGER	NATURAL, NATURALN, POSITIVE, POSITIVEN, SIGNTYPE
VARCHAR2	STRING, VARCHAR
CHAR	CHARACTER

II.8 LES TYPES COMPOSES

Il existe 3 modèles de types de données composées :

- RECORD
- TABLE

- VARRAY

Ils vont servir à définir, dans une première phase, des types structurés.
On se référera, dans une seconde phase, à ces types lors de la déclaration des variables.

II.8.1 RECORD

Le type RECORD permet de définir des types structurés destinés à contenir une seule ligne à plusieurs colonnes.

Syntaxe :

```
TYPE nom_type IS RECORD (champl {type_scalaire | identifiant%TYPE} [champ2 {type_scalaire | identifiant%TYPE}...]);
```

Déclaration de la variable :

```
nom_variable nom_type ;
```

Exemple :

```
TYPE enreg IS RECORD (nom client.raisonsociale%TYPE, commande commande.numcom%TYPE, livraison DATE);
comcli enreg ;
```

comcli : variable de type composé 'enreg' et contient les champs nom, commande, livraison.

II.8.2 TABLE

Le type TABLE permet de définir des types tableau à 2 colonnes, dont l'une des colonnes sera obligatoirement une clé primaire. Cette clé permettra d'accéder aux lignes du tableau comme un indice.

Syntaxe :

```
TYPE nom_type IS TABLE OF {type_colonne | table.colonne%TYPE} INDEX BY BINARY_INTEGER;
```

Déclaration de la variable :

```
nom_variable nom_type;
```

Exemple :

```
TYPE typeraisoc IS TABLE OF Client.raisonsociale%TYPE INDEX BY BINARY_INTEGER;
nom_societe typeraisoc ;
```

nom-societe est une variable de type composé typeraisoc contenant une colonne de type identique à la colonne raisonsociale de la table client.

```
Nom_societe(l) :='DARTY';
nom_societe(2) :='CARREFOUR';
```

On ne peut définir qu'une seule colonne. Par contre, cette colonne peut être de type record.

Exemple :

```
TYPE typeraisoc IS TABLE OF Client%ROWTYPE INDEX BY BINARY_INTEGER;
nom_societe typeraisoc ;
nom_societe(l).raisonsociale := 'DARTY';
nom_societe(l).ville :='PARIS';
nom_societe(2).raisonsociale :='CARREFOUR';
nom_societe(2).ville :='MARSEILLE';
```

II.8.3 VARRAY

Une collection de type VARRAY possède une dimension maximale qui doit être précisée lors de la déclaration de la collection. Ces collections possèdent une longueur fixe et donc la suppression d'éléments ne permet pas de gagner de la place en mémoire. Les éléments sont numérotés à partir de la valeur 1.

Syntaxe :

```
TYPE nom_type IS VARRAY (taille_maxi) OF type_element [NOT_NULL]
```

nom_type : représente le nom de la collection
taille_maxi : nombre d'éléments maximum présents dans la collection
type_element : représente le type de données des éléments de la collection.

Exemple : Déclaration et utilisation de collections

```
DECLARE
        TYPE calendrier is VARRAY(366) OF DATE ;
        Annee calendrier := calendrier (tp_date('01/01/2007','DD/MM/YYYY'));
BEGIN
        -- changement de valeur
        Annee(1) := calendrier (tp_date('01/01/2008','DD/MM/YYYY'));
END ;
```

II.9 LES VARIABLES LOCALES

PL/SQL dispose de l'ensemble des types utilisables dans la définition des colonnes des tables. Ce qui permet un format d'échange cohérent entre la base de données et les blocs PL/SQL. Cependant, les étendues des valeurs possibles pour chacun de ces types peuvent être différentes de celle du SQL.
Il dispose également un certain nombre de types propres, principalement pour gérer les données numériques.

Enfin, PL/SQL permet de définir des types complexes basés soit sur des structures issues des tables, soit sur des descriptions propres à l'utilisateur.

Syntaxe : Dans le bloc DECLARE

Nom-de-variable [CONSTANT] **type** [[NOT NULL] :=**expression**] ;

CONSTANT : La valeur de la variable n'est pas modifiable dans le code de la section BEGIN
NOT NULL : empêche l'affectation d'une valeur NULL à la variable, la variable doit être initialisée avec une valeur.
Expression : valeur initiale affectée à la variable dans le bloc.

Exemple : Liste de tous les employés dont le nom commence par une chaîne de caractères, donnée (B).

```
DECLARE
vRech varchar(24) := 'B%';
BEGIN
SELECT noemp, prenom, nom, dept FROM Employes
WHERE nom like vRech;
END;
```

Exemples : renvoi du numéro d'employé le plus élevé.

```
DECLARE
vempId int;
BEGIN
SELECT vempId = max(noemp) FROM Employes;
END
```

Si l'instruction SELECT renvoie plusieurs lignes, la valeur affectée à la variable est celle correspondant à la dernière ligne renvoyée.

```
DECLARE
vempId int;
BEGIN
SELECT vempId = noemp FROM Employes;
END ;
```

II.10 LES ELEMENTS DE CONTROLE DE STRUCTURE

AFFECTATION

X:= 2;

CONCATENATION
chaine := chaine1 || chaine2 || chaine3;

Exemple:
chaine := 'La valeur est ' || to_char(x) || ' unités'

OPERATEURS RELATIONNELS
= , != ou <>
> , < , >= , <=
in, between, like

OPERATEURS LOGIQUES
not, and, or

VALEURS BOOLEENNES
true, false, null

COMMENTAIRES
-- commentaire sur une ligne
/* commentaire sur plusieurs lignes */

II.11 LES ENTREES/SORTIES

II.11.1 AFFICHAGES DE RESULTATS

On utilise le package (ensemble de procédures et fonctions) **DBMS_OUTPUT**.
Tout d'abord, avant le bloc PL/SQL, utiliser l'instruction qui autorise l'utilisation des sorties écran (une seule fois pas session):

```
set serveroutput on
```

Puis pour chaque affichage :

```
dbms_output.put_line('Nom du client : ' || vraisonsociale);
```

|| est l'opérateur de concaténation de chaînes de caractère.

II.11.2 SAISIE DE DONNEES AU CLAVIER

&nom : saisie d'un mot au clavier ;

Un mot est un nombre ou une chaîne de caractères alphanumériques (éventuellement entre apostrophes).
Oracle substituera la suite des caractères saisis à &nom.

Exemple

```
DECLARE
 enrclient client%ROWTYPE;
BEGIN
 SELECT * into enrclient FROM CLIENT
 WHERE idclient = &code;
 SELECT * into enrclient FROM CLIENT
 WHERE raisonsociale LIKE '%&nom%';
END ;
```

II.12 TRAITEMENTS CONDITIONNELS

Les instructions sont exécutées si la condition est évaluée à TRUE. Si elle est évaluée à FALSE ou NULL, elle est éventuellement traitée par le ELSE.
L'instruction ELSIF permet d'imbriquer plusieurs traitements conditionnels.

Syntaxe :

```
IF condition THEN
commandes PLSQL;
[ELSIF condition THEN
commandes PLSQL;]
[ELSIF condition THEN
commandes PLSQL;]
[ELSE
commandes PLSQL;]
END IF;
```

Exemple :

```
IF stock < 5 THEN
                alertstock :='Alerte grave - commande immédiate';
ELSIF stock < 20 THEN
/* 5 <= stock < 20 */
                alertstock :='Alerte moyenne - commande à prévoir';
```

```
ELSIF stock > 50 THEN
/* stock > 50 */
                alertstock :='Stock à bon niveau';
ELSE
/***** 20 <= stock <= 50 */
                alertstock :='Attention - stock en baisse';
END IF;
```

Remarque:
Pour tester les valeurs NULL,

```
IF adresse != NULL  ne fonctionne pas;
```

Il faut écrire:

```
IF adresse IS NULL  ou IF adresse IS NOT NULL
```

II.13 TRAITEMENTS ITERATIFS

On dispose de 3 instructions d'itération dans PLSQL:

- LOOP ... END LOOP
- FOR ... LOOP ... END LOOP
- WHILE ... LOOP ... END LOOP

II.13.1 LOOP ... END LOOP

Permet d'exécuter plusieurs fois un même ensemble d'instructions.
On doit sortir de l'itération à l'aide d'une instruction conditionnelle (EXIT WHEN condition) ou inconditionnelle contenue dans un bloc IF .... END IF (EXIT ou GOTO).

Syntaxe :

```
LOOP
        Exit WHEN condition;
        ensemble d'instructions;
END LOOP;
```

II.13.2 FOR ... LOOP ... END LOOP

Permet d'exécuter un nombre précis d'itérations.

Syntaxe :

```
FOR compteur IN [REVERSE] borne-inf..borne-Sup
```

```
LOOP
    ensemble d'instructions;
END LOOP;
```

Remarques :

- La variable compteur n'a pas besoin d'être préalablement déclarée, FOR crée implicitement une variable locale.
- Avec l'option REVERSE, on va de borne-sup à borne-inf
- L'incrément est de 1 et de - 1 avec REVERSE.
- Les .. font office de séparateur entre borne-inf et borne - sup.
- On peut aussi forcer la sortie avec un exit [when condition].

Exemple :

```
SET SERVEROUTPUT ON; -- autorise la sortie écran
BEGIN
FOR i IN 1..5 LOOP
        dbms_output.put_line('ligne ' || i); -- sortie écran
END LOOP;
END;
```

II.13.3 WHILE ... LOOP

Permet d'exécuter une itération avec une condition d'entrée dans la boucle, condition vérifiée à chaque boucle.

Syntaxe :

```
WHILE condition LOOP
    ensemble d'instructions
END LOOP;
```

Exemple :

```
DECLARE x integer;
BEGIN x:= &donne;  -- lire au clavier
WHILE abs(x) < 100
LOOP
    x := x*x;
END LOOP;
```

```
dbms_output.put_line('x= ' || to_char(x));
END;
```

II.14 LES CURSEURS

Un curseur est une zone de contexte en mémoire, ouverte par Oracle pour exécuter une requête et stocker les résultats.
Il existe 2 sortes de curseurs : **implicites** et **explicites**.

II.14.1 CURSEUR IMPLICITE

Un curseur implicite nommé 'SQL' est créé par Oracle pour nommer la zone de contexte ouverte pour le traitement d'un ordre SQL. La requête ne doit pas retourner plus d'une ligne.

Exemple :

```
DECLARE
art article%ROWTYPE;
BEGIN
SELECT * INTO art
FROM article
WHERE idarticle = 3;
dbms_output.put_line(art.designation)
--impression du nom du produit
END;
```

art.idarticle contiendra 3, art.designation le nom de l'article, etc.

II.14.2 CURSEUR EXPLICITE

Si on utilise une requête devant retourner plus d'une ligne, on doit déclarer un curseur, dit explicite, auquel on associe cette requête.
L'exploitation d'un curseur se fait en trois étapes : ouverture, lecture, fermeture.

II.14.3 DECLARATION DE CURSEUR

La déclaration d'un curseur a pour effet d'associer un nom de variable de type curseur à une requête SQL.
Quand la requête sera exécutée (ouverture du curseur), on pourra accéder aux lignes du résultat stockées en mémoire par le biais du nom associé (nom du curseur) et transférer chaque ligne du résultat dans des variables.
On peut associer des paramètres formels à un curseur qui seront référencés dans la clause

WHERE de la requête.

Syntaxe :

```
CURSOR nom [(paraml type [,param2 type] ...)]
IS requête
[FOR UPDATE OF nom_col1 [, nom_col2] ... ];
```

Le type du paramètre peut être : char, number, boolean, date.

Exemple :

```
CURSOR curart
IS SELECT idarticle, désignation
FROM article;
```

Déclaration de curseur avec paramètres :

```
CURSOR curart (prixunit number(5,2), qte number)
IS SELECT idarticle, désignation
FROM article
WHERE prixunit > curart.prixunit
AND qtestock < qte ;
```

On remarque que lorsqu'il peut y avoir confusion de noms (ici prixunit), c'est le paramètre du curseur qu'il faut préfixer et non pas la colonne de la table.

II.14.4 OUVERTURE DU CURSEUR

L'ouverture du curseur provoque l'exécution de la requête. On peut passer des paramètres au curseur à l'ouverture.

Syntaxe :

```
OPEN nom_du_curseur [(param1 [,parami] ...)];
```

Exemple :

```
DECLARE
CURSOR curart (prixunit number(5,2), qte number) IS
SELECT idarticle, désignation FROM article
WHERE prixunit > curart.prixunit AND qtestock < curart.qte ;
```

```
TYPE recart IS RECORD
(ident article.idarticle%TYPE,
desig article.designation%TYPE);
produit recart;
BEGIN
OPEN curart(1300,250);
-- équivaut à SELECT idarticle, designation
-- FROM article
-- WHERE prixunit > 1300 AND qtestock < 250 ;
```

II.14.5 LECTURE DU CURSEUR

La commande FETCH permet de copier une ligne de résultats, dans une ou plusieurs variables.

Syntaxe :

```
FETCH nom_du_curseur INTO {liste_de_variables | record};
```

Exemple : affiche tous les articles stockés dans le curseur.

```
LOOP
FETCH curart INTO produit;
EXIT WHEN curart%NOTFOUND;
dbms_output.put_line(produit.ident,produit.desig);
END LOOP;
```

II.14.6 FERMETURE DU CURSEUR

Libère les ressources associées au curseur.

Syntaxe :

```
CLOSE nom_du_curseur;
```

II.14.7 MODIFICATION ET SUPPRESSION DE DONNEES

Pour pouvoir supprimer la ligne de la table correspondant à la ligne courante du curseur (ou y modifier des données), il faut impérativement avoir déclaré le curseur avec l'option "FOR UPDATE OF col1, col2,..."

Dans l'instruction SQL de modification ou de suppression, il faut utiliser le mot clé CURRENT OF nom_curseur, dans la clause WHERE.

Exemple :

```
IF adresse IS NULL THEN
UPDATE client SET adresse='??????'
WHERE CURRENT OF nom_curseur;
END IF;
```

II.14.8 ATTRIBUT DES CURSEURS

Pour les curseurs explicites :

- %NOTFOUND : booléen. Vrai quand la commande FETCH échoue.
- %FOUND: inverse de %NOTFOUND.
- %ROWCOUNT : nombre de lignes lues dans le curseur.
- %ISOPEN : booléen. Vrai si le curseur est ouvert.

Dans le cas d'un curseur implicite, on peut utiliser SQL%NOTFOUND par exemple

Exemple :

```
FETCH nom_curseur INTO nom_variable ;
IF nom_curseur%NOTFOUND THEN
Dbms_output.put_line ('Fetch a échoué') ;
END IF;
```

Exemple complet :

```
DECLARE
CURSOR curart
                (prix article.prixunit%TYPE,
                qte article.qtestock%TYPE)
IS
SELECT * FROM article
        WHERE prixunit > prix
AND Qtestock > qte;
art article%ROWTYPE
BEGIN
OPEN curart(100, 10)
FETCH curart INTO art;
WHILE curart%FOUND LOOP
        dbms_output.put_line(art.designation)
```

```
        FETCH curart INTO art;
END LOOP;
CLOSE curart;
END;
```

II.14.9 UTILISATION DE FOR...LOOP

L'utilisation de FOR ... LOOP avec un curseur permet d'ouvrir, parcourir et fermer un curseur sans utiliser OPEN, FETCH INTO et CLOSE.

Syntaxe :

```
FOR nom_variable IN nom_curseur (param...) LOOP
        ensemble d'instructions;
END LOOP;
```

II.15 LES EXCEPTIONS

Lorsque ORACLE rencontre une erreur dans l'exécution d'un programme (overflow, violation de contraintes d'intégrité ...), il sort immédiatement du bloc, en affichant le code et le message d'erreur.

Si le développeur veut poursuivre l'exécution du programme, il doit rediriger et traiter ces cas dans la partie EXCEPTION du bloc PL/SQL(sauf les exceptions internes déjà prénommées par Oracle).

Dans cette partie EXCEPTION, il pourra pour chaque cas d'erreur ou d'exception, à condition de les préciser et de les avoir déclarées (pour les exceptions externes), exécuter un traitement particulier.

II.15.1 DECLARATION

On peut gérer 2 types d'erreurs ou d'exceptions :

- les exceptions internes (prédéfinies par Oracle)
- les exceptions externes (déclarées par le développeur).

Syntaxe :

```
DECLARE
 déclarations
 nom_exception_externe EXCEPTION;
BEGIN
```

```
 instructions
 IF condition THEN
RAISE nom_exception_externe;
 END IF;
 instructions
EXCEPTION
 WHEN nom_exception_externe THEN instructions ;
 WHEN nom_exception_interne THEN instructions ;
 WHEN OTHERS THEN instructions
END;
```

Remarques :

- Les exceptions internes sont redirigées par Oracle sans que l'on doive utiliser la commande RAISE.
- Le fait de traiter une exception fait sortir du bloc (BEGIN ... END) auquel appartient l'exception. Le fait de traiter des exceptions dans le bloc le plus externe d'un programme entraîne donc l'arrêt du programme. Il faudra intégrer une exception dans un bloc imbriqué si l'on désire continuer le traitement après la gestion de cette exception.
- Dans la partie EXCEPTION, la condition WHEN OTHERS THEN permet de gérer les erreurs internes qui n'ont pas de nom spécifique ou qu'on ne désire pas traiter en particulier.

II.15.2 EXCEPTIONS INTERNES

Une erreur interne se produit quand un bloc PL/SQL viole une règle d'Oracle ou dépasse une limite dépendant du système d'exploitation.

Oracle a prévu un ensemble d'erreurs prédéfinies qui sont les suivantes

Nom de l'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
PROGRAMM_ERROR	ORA-06501	-6501
STORAGE_ERROR	ORA-06500	-6500
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
TRANSACTION _ BACKED_OUT	ORA-00061	-61
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476
OTHERS (autres erreurs)		

Remarque : L'erreur NO_DATA_FOUND peut être générée par un SELECT INTO, un UPDATE ou un DELETE.

Exemple :

```
DECLARE
  code number;
  msg varchar2(200);
  recart article%ROWTYPE;
  CURSOR curcli IS
SELECT * FROM client
FOR UPDATE OF adresse;
  reccli client%ROWTYPE;
BEGIN
  OPEN CURCLI;
  LOOP
FETCH curcli INTO reccli;
EXIT WHEN curcli%NOTFOUND;
SELECT * INTO recart FROM article;
BEGIN
  INSERT INTO client VALUES (reccli.idclient,reccli.raisonsociale, NULL, NULL, NULL, NULL, reccli.idrep);
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
dbms_output.put_line('Attention doublon');
  WHEN TOO_MANY_ROWS THEN
dbms_output.put_line('Passage trop de lignes');
```

```
 WHEN OTHERS THEN
code:= SQLCODE;
msg:= SQLERRM(code);
dbms_output.put_line(to_char(code));
dbms_output.put_line(msg);
END;
 END LOOP;
CLOSE curcli;
END;
```

II.15.3 EXCEPTIONS EXTERNES

Si le programmeur veut gérer d'autres types d'erreurs que celles prédéfinies, généralement des erreurs liées à l'application, il doit les définir et aussi assurer le branchement sur la partie EXCEPTION à l'aide de la commande RAISE, utilisée dans un bloc de traitement conditionnel.

Exemple :

```
DECLARE
 code number;
 msg varchar2(200);
 recart article%ROWTYPE;
 pas_de_prix EXCEPTION;
BEGIN
 SELECT * INTO recart FROM article
 WHERE idarticle = &numéro;
 IF recart.prixunit = 0 THEN
        RAISE pas_de_prix;
 END IF;
EXCEPTION
 WHEN pas_de_prix THEN
        dbms_output.put_line('Cet article n'a pas de prix');
 WHEN OTHERS THEN
        code:= SQLCODE;
        msg:= SQLERRM(code);
        dbms_output.put_line(to - char(code»;
        dbms_output.put_line(msg);
```

```
END;
```

II.15.4 LA PROCEDURE RAISE_APPLICATION_ERROR

La procédure RAISE_APPLICATION_ERROR(n, 'message'), où n est un numéro choisi par le développeur entre -20000 et -20999 a la particularité de stopper immédiatement le programme et de renvoyer un numéro et un message d'erreur au programme appelant.

Exemple :

```
DECLARE
 code number;
 msg varchar2(200);
CURSOR curart IS
SELECT * FROM article
FOR UPDATE OF qtestock;
 recart article%ROWTYPE;
 mise_a_jour EXCEPTION;
BEGIN
 OPEN curart
 LOOP
FETCH curart INTO recart;
EXIT WHEN curart%NOTFOUND;
BEGIN
 IF recart.qtestock < 30 THEN
        RAISE_APPLICATION_ERROR (-20500,'Programme stoppé');
 END IF;
 IF recart.qtestock < 100 THEN
        RAISE mise_a_jour;
 END IF;
EXCEPTION
 WHEN mise_a_jour THEN
UPDATE article SET qtestock = qtestock + 100
WHERE CURRENT OF curart;
END;
 END LOOP;
 CLOSE curart;
```

```
EXCEPTION
  WHEN OTHERS THEN
        code:= SQLCODE;
        msg:= SQLERRM(code);
        dbms_output.put_line(to_char(code));
        dbms_output.put_line(msg);
END;
```

Etablissement référent
Direction de l'ingénierie Neuilly

Equipe de conception
Groupe d'étude de la filière étude - développement

Remerciements :

Date de mise à jour 05/12/2014
afpa © Date de dépôt légal décembre 14

afpa / Direction de l'Ingénierie13 place du Générale de Gaulle / 93108 Montreuil Cedex
association nationale pour la formation professionnelle des adultes
Ministère des Affaires sociales du Travail et de la Solidarité