- un bloc pl/sql doit se terminer par un '/' (pour qu'il soit exécuter)
- **'chaine caractère' et double quotes pour les identifiant (alias de table colonne..)**
- il y a trois types de variable :
    - Une variable bind (déclarer hors le bloc PLSQL) est une variable globale , qu'on déclare comme suit : VARIABLE nom type et qu'on doit précéder son nom par un ' :' pour l'utiliser dans un bloc plsql .
    - variable local déclarer dans le bloc DECLARE
    - variable de substitution (appelé aussi paramètre) qui est lu au clavier grâce à l'opérateur & ou &&
- dans DECLARE , on declare une variable PLSQL comme suit :
    - variable [CONSTANT] type [NOT NULL] [(:= ou DEFAULT) value/expr]
        - *expr* peut être une constante, une variable, un appel de fonction mais *pas* une colonne de base de données
    - pour le type on peut aussi faire :
        - au lieu de type on utilise tab.column%type
        - ou bien variable_déjà_défini%type
- Les variables BIND doivent être précédés du mot VARIABLE (nom type), ceux-ci sont déclarés hors le bloc PL-SQL
    - pour utiliser une variable bind il faut précéder son nom par :

Affiche de variables (deux méthodes) :

1. Afficher une variable locale (déclarer dans le bloc DECLARE ou bien un variable de substitution) : DBMS_OUTPUT.PUT_LINE (Variable) (on peut aussi afficher les variables de type bind de cette façon en précédant le nom de la variable avec ' : ') il ne faut pas oublier d'ajouter **SET SERVEROUTPUT ON** au début du script
    a. elle est une bonne habitude de programmation de faire les conversions explicites (TO_CHAR()) dans le PUT_LINE ;
2. Affiche une variable BIND : avec PRINT nom_variable (sans ' :') : **PRINT est une fonction SQL PLUS il faut l'utiliser hors le bloc plsql** (apres le ENDS ; /)

- le type NUMBER(n,s) : n représente le nombre de total des chiffres et s représente combien des chiffres parmi les n chiffres seront décimaux (Si on donne NUMBER(n) seulement le contenu de de NUMBER sera un nombre entier , si on essaie d'y mettre un nombre flottant ce dernier sera arrondi)
- Les fonctions suivantes ne sont pas disponibles dans les instructions PL/SQL : (celles-ci ne peuvent être utilisée que dans des instruction SQL du bloc PLSQL
    - DECODE.
    - Fonctions de groupe : AVG, MIN, MAX, COUNT, SUM, STDDEV, et VARIANCE. Les fonctions de groupe agissent sur des ensembles de lignes d'une table et ne sont, par conséquent, disponibles que dans les instructions SQL des blocs PL/SQL .
- ACC[EPT] variable [NUM[BER] | CHAR | DATE | BINARY_FLOAT | BINARY_DOUBLE] [FOR[MAT] format] [DEF[AULT] default] [PROMPT text|NOPR[OMPT]] [HIDE]
    - accept ne marche qu'avec les types ci-dessus (pas de varchar2) en plus on ne précise pas la taille du type)
- l'operateur ** est l'opérateur exponentiel (n**m = n à la puissance m)

- règle sur la programmation plsql : tous s'écrit en majuscule sauf les identifiants, les paramètres et les tables et les colonnes des basses
- for :
    - for i in [REVERSE]prem .. dernier LOOP – pas besoin de declarer i , celui-ci est valable --seulement dans la boucle
    - end loop ;
- SQL%FOUND SQL%NOTFOUND SQL%ROWCOUNT
- IF THEN ELSE END IF
- IF THEN ELSIF END
- loop basique : (En analogie avec les autres langages , c'est comme un while(true) qui contient un if(condition) Break ;
    - LOOP
        - instructions
        - EXIT [WHEN CONDITION] ;
    - END LOOP
- il est utile de donner des étiquettes à ses boucles aux cas ou on a des boucles imbriqué et on veut quitter la boucle mère (on met <<loop_name>> avant la boucle et on fait EXIT loop_name WHEN condition pour quitter une loop précise quand la condition est vraie)
- n'oublie pas COMMIT ; après des commandes LMD
- **ON NE PEUT EXECUTER QUE DES COMMANDES LMD DANS UN BLOC PLSQL**
- SQL PLUS variable d'environement :
    - If verify is on, SQL*Plus prints two lines for each substituted variable
- TRUE AND NULL => NULL , TRUE OR NULL => TRUE , FALSE AND NULL => FALSE , FALSE OR NULL => NULL
- INSERT INTO EMP(col) VALUES (DECODE(….))
- Il n'existe pas de type prédéfini pour les records ni pour les tables PL/SQL, Vous devez d'abord créer le type puis déclarer l'identifiant utilisant ce type.
- LES RECORDS :
    - déclaration :
        - façon 1 : on crée le type record : TYPE rec_type_name IS RECORD (champ1 type1,champ2 type2…) ;
        - puis on crée un record comme suit : rec_name rec_type_name ;
        - façon 2 : on définit un record qui a la structure d'un tableau déjà existant :
            - rec_name table_name%ROWTYPE
- LES TABLES plsql :
    - définition d'un type de table :
        - **TYPE** nom_table_type IS TABLE OF type (resp record ou nom variable déjà définit) INDEX BY BINARY_INTEGER
    - déclaration d'une table :
        - tab_name tab_type
    - autre :

- on utilise nom_table_name.EXISTS(i) pour verifier que l'index i exist dans la table pl sql .
- l'index d'une table est compris entre -2147483647 et 2147483647

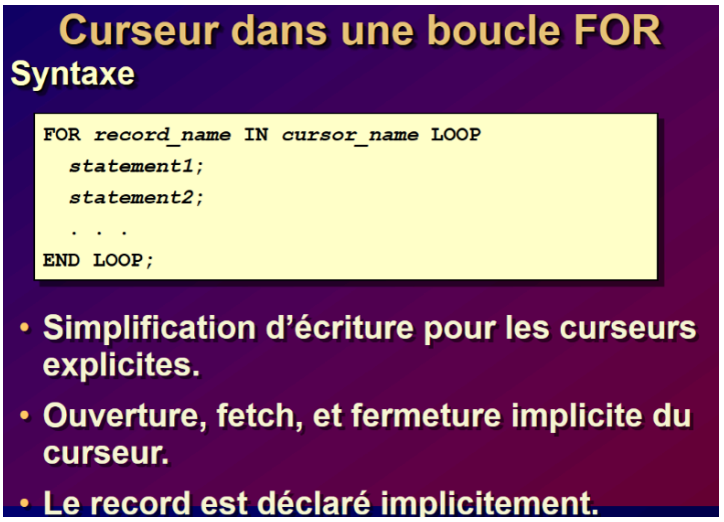
Fonction	Description
EXISTS(*n*)	Renvoie TRUE si le nième élément de la table PL/SQL existe.
COUNT	Renvoie le nombre d'éléments contenus dans la table PL/SQL
FIRST LAST	Renvoie le premier et le dernier (le plus petit et le plus grand) rang de la table PL/SQL . Renvoie NULL si la table est vide.
PRIOR(*n*)	Renvoie le nombre qui précède n dans la table.
NEXT(*n*)	Renvoie le nombre qui succède à n dans la table.
EXTEND(*n*, *i*)*	Augmente la taille de la table PL/SQL. EXTEND rajoute un élément nul à la table PL/SQL. EXTEND(*n*) rajoute n éléments nuls à la table PL/SQL. EXTEND(*n*, *i*) rajoute n copies de l'élément i à la table PL/SQL.
TRIM*	TRIM supprime le dernier élément de la table PL/SQL. TRIM(*n*) supprime les n derniers éléments de la table PL/SQL.
DELETE	DELETE supprime tous les éléments de la table PL/SQL. DELETE(*n*) supprime l'élément n de la table PL/SQL. DELETE(*m*, *n*) supprime tous les éléments de m à n de la table PL/SQL.

- Une table de records :
  - définition du type : TYPE name_table_type IS TABLE OF une_table_sql_existante%rowtype INDEX BY BINARY_INTEGER
- si un select into ne retourne rien c'est une erreur (exception)
- tab(i) et non pas tab[i]
- on ne peut pas select into une table entière
- curseur :
  - cursor cursor_name IS SELECT * FROM EMP ; (pas d'into)
  - OPEN cursor
  - FETCH cursor_name INTO var1,var2… ou bien record.
  - CLOSE cursor

Attribut	Type	Description
%ISOPEN	Boolean	Evalué à TRUE si le curseur est ouvert.
%NOTFOUND	Boolean	Evalué à TRUE si le dernier fetch n'a pas retourné de ligne.
%FOUND	Boolean	Evalué à TRUE si le dernier fetch a retourné une ligne ; complément de %NOTFOUND

Note : Avant le premier fetch, %NOTFOUND est initialisé à NULL. Ainsi, si l'affectation ne s'effectue pas avec succès, la boucle n'est jamais stoppée. C'est pourquoi l'instruction EXIT WHEN s'exécute seulement si sa condition WHEN est vraie. Pour être en sécurité, vous voudrez peut-être utiliser l'instruction EXIT ci-dessous:

EXIT WHEN emp_cursor%NOTFOUND OR emp_cursor%NOTFOUND IS NULL;



*le record_name n'est définit que dans la boucle (**on ne peut utiliser qu'un record et non pas des variables même si on veut récupérer une seule variable !!!**)

- On peut même utiliser un curseur sans le déclarer : FOR rec_name IN (SELECT … FROM … [ORDER BY]) LOOP … END LOOP ;
- les curseurs parametrés :
- CURSOR curs_name(par1 VARCHAR2,par2 NUMBER) (on ne precise pas de taille)
- IS SELECT * FROM … WHERE … = par1 && …. = par2
- OPEN CURSOR ('par1',par2)
- ou LOOP rec_name IN curs_name('par1',par2)
- WHERE CURRENT OF curs_name : pour référencer la ligne courante du curseur utilisée avec un curseur déclarer avec FOR UPDATE
- LES EXCEPTIONS