**PL\_SQL**

Table des matières

[Chapitre 1 : Définition, Objectifs, avantages & différents Programmes PLSQL : 2](#_Toc93593508)

[Chapitre 2 : INSTRUCTIONS PLP/GSQL : 3](#_Toc93593509)

[2.1 : LES INSTRUCTION ALTERNATIVES : 3](#_Toc93593510)

[2.2: INSTRUCTIONS CASE : 3](#_Toc93593511)

[2.3 LES BOUCLES : 3](#_Toc93593512)

[Chapitre 3 : AFFECTATION : 4](#_Toc93593513)

[Chapitre 4 : REFERENCE DES TYPES DE COLONNES : 4](#_Toc93593514)

[Chapitre 5 : Référence à un ligne entière d’une table : 5](#_Toc93593515)

[Chapitre 6 : Déclaration d’une fonction 5](#_Toc93593516)

[Chapitre 7 : LES CURSEURS 6](#_Toc93593517)

[Chapitre 8 : DECLENCHEUR (TRIGGER) : 6](#_Toc93593518)

# 

# : Définition, Objectifs, avantages & différents Programmes PLSQL :

PLSQL c’est l’outil qui permet de faire de la programmation (programmes = algorithmes = code) dans les bases de données dans le but de faire des traitements complexes (combinaison de plusieurs commandes sql) qu’on ne pouvait pas faire avec le simple langage SQL et les stocker sur base données.

Il permet aussi de faire plusieurs requêtes en même temps (langage ensembliste) et les exécuter sur

**Exemples** :

Sur SQL on ne pouvait pas changer l’affichage de la commande SELECT et on ne pouvait pas stocker l’affichage dans la base données, mais grâce à PL Pg SQL on peut le faire.

On ne peut pas faire (c’est fastidieux) tous les jours select de tous les nouveaux clients qui s’inscrivent dans un site donc on peut demander ceci avec un programme plsql.

Il existe différents types de programmes qu’on peut réaliser avec PLSQL :

1. **Un Bloc Anonyme** : C’est un programme qui ressemble au langage sql sauf qu’iil contient au moins un point d’entrée, un point de sortie (c’est ce qu’il le différencie du langage sql habituel) - On le fait pour tester nos programmes.
2. **Une fonction :** c’est comme dans le cour Algo, il faut définir les paramètres, le type de retour etc ..
3. **Les procédures :** c’est des fonctions qui retournent un vide.
4. **Un déclencheur :** c’est l’évènement qui permet de déclencher un programme (une fonction, procédure)

Dans les chapitres suivants on va détailler chacun de ces programmes.

**Déclaration de la variable**

*nom* [ CONSTANT ] *type* [ COLLATE *nom\_collationnement* ] [ NOT NULL ] [ { DEFAULT | := | = } *expression* ];

**Déclaration de la variable :**

* Toutes les variables utilisées dans un programme PLSQL doivent être déclarées en amont de celui-ci
* Dans la déclaration de la variable le minimum qu’on peut trouver c’est le nom de la variable (doit être différent des noms des tables et colonnes de la base de données déjà existantes pour distinguer entre les différentes valeurs dans notre base de données), et son type (comme en algorithme)
* Selon le besoin on peut aussi rajouter des précisions comme l’initialisation de la variable (à partir de quelle valeur on veut qu’elle démarre dans notre programme), est ce qu’elle est constante ou pas dans tous le programme, est ce qu’elle peut prendre les valeurs nulles ou pas.

Voici quelques exemples de de type de variables qu’on peut trouver dans PLPGSQL :

* Integer ; (entier) (pour les identifiants des colonnes)
* numeric (5, 2) ; (pour afficher les virgules) 🡺 ici c’est 2 chiffres après la virgule.
* Varchar ;
* Myrow nomtable %ROWTYPE ; (type ligne de table) et dans ce cas type ligne on va avoir les types de chaque colonne de la ligne donc différents types possibles
* Myfield nomtable.nomcolonne %TYPE ; (type de la colonne de la table).

# : INSTRUCTIONS PLP/GSQL :

Comme dans les algorithmes dans PL\_PGSQL on peut trouver toutes différentes structures de contrôles à savoir : SEQUENCES, CONDITIONS (SI et SI..SINON), LES SWITCHS, LES BOUCLES, LES PROCEDURES ET FONCTIONS

## 2.1 : LES INSTRUCTION ALTERNATIVES :

**IF** condition 1

**THEN** bloc instructions1

**ELSE IF** condition 2

**THEN** bloc instructions2

**ELSE** bloc instructions3

**END IF;**

## 2.2: INSTRUCTIONS CASE :

**CASE** (variable) variable = variable qu’on veut traiter

**WHEN** valeur1 **THEN** bloc instructions1

**WHEN** valeur2 **THEN** bloc instructions2

**ELSE** bloc instructions par défaut

**END CASE ;**

## 2.3 LES BOUCLES :

* **Boucle FOR (POUR)** : on sait à l’avance le nombre d’itérations.

**FOR** variable **IN born\_inf** …**Borne Sup**

**LOOP** Bloc d’Instructions

**END LOOP;**

* **Boucle WHILE (La boucle TANQUE… FAIRE…FTNQUE) :** La condition avant de rentrer dans la boucle, il se peut donc que la boucle ne sera pas réalisée si la condition n’est pas affirmée.

**WHILE** condition

**LOOP** Bloc instructions

**END LOOP;**

* **Boucle LOOP (la boucle FAIRE…TANQUE) :** La condition est dans la boucle donc on exécute au moins une fois la boucle (c’est l’itération de préparation).

**LOOP** instructions

**EXIT WHEN** conditions;

**END LOOP;**

# : AFFECTATION :

Le symbole d’affectation est **:=** pour une seule valeur à une seule variable

Ou

**SELECT** col1, col2, col3…coln **INTO** valeur col1, …,valeur coln

# : REFERENCE DES TYPES DE COLONNES :

Myfield nom\_table.nomcolonne**%**TYPE 🡺 la colonne myfield prendra comme type le même type que la colonne nom\_table.nomcolonne.

**Exemple :**

Dans l’exo 2 de SQL :

On crée une colonne « nom\_client » dans la table client et on lui donne le même type que la colonne cli\_societe

**Declare**

Nom\_client client.cli\_societe**%**type ; – nom\_client est de même type que client.cli\_societe

**BEGIN**

SELECT cli\_societe INTO nom\_client FROM client where ncli = 003;

RAISE NOTICE ‘le nom du client est : nom\_client’

**End**

**On a affiché la valeur de nom\_client**

# : Référence à un ligne entière d’une table :

Ma\_ligne nom\_table%ROWTYPE ;

**Exemple :**

Dans l’exo 2 de SQL :

On crée une ligne **var** du même type que les lignes de la table **client** on la remplit avec les mêmes valeurs de la ligne 5 de la table **client**

**Declare**

Var client%ROWTYPE ; – var est de même type que les lignes de la table client

**BEGIN**

SELECT \* INTO var FROM client where ncli = 005;

RAISE NOTICE var

**End**

# : Déclaration d’une fonction

**CREATE** **OR** **REPLACE** **FUNCTION** nom\_fonction(arg1, arg2, …) RETURNS TYPE

**AS** $$

**DECLARE**

Variables locales TYPES;

**BEGIN**

**Instructions**

**END**

$function$;

Dans le PLpgsql on a deux types de fonctions en fonction de l’existence ou pas de retour :

Si le retour est vide 🡺 Procédure

Si le retour est une valeur quelconque 🡺 Fonction

# : LES CURSEURS

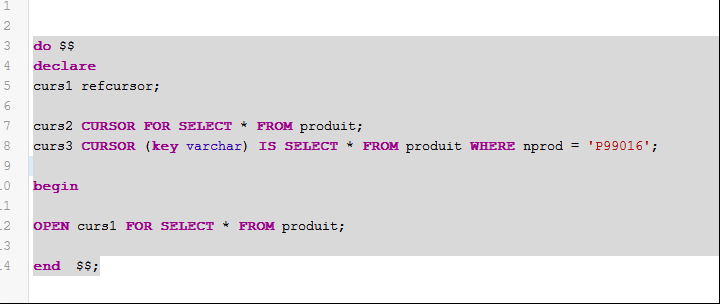
Les curseurs sont un type de donné utilisés pour le traitement par lot. (Pour éviter les bugs dans l’unité centrale)

Souvent c’est quand on ignore le nombre de lignes qui existent à traiter.

**Déclaration de curseurs :**

*Nom\_du\_curseur* **CURSOR** ( *arguments* ) **FOR** *requête*;

**Exemples**:



**Utilisation des curseurs :**

A VOIR LA SSEMAINE PROCHAINE

# : DECLENCHEUR (TRIGGER) :

Le trigger est un programme (dans une BDD) associé à un évènement et à **une** **seule** table et qui s’exécute dès que l’évènement auquel il est associé se produit.

Il peut être actif ou non (enable/desable)

**Create TRIGGER** “nom\_du\_trigger)

**BEFORE/AFTER** 🡺 Avant ou après

**DELETE OR INSERT OR UPDATE** 🡺 évènement déclencheur

**ON** «nom\_TABLE »

**EXECUTE PROCEDURE** (nom\_de\_la\_fonction)