**PL/pgSQL**

Table des matières

[1. Le langage PL/PGSQL 2](#_Toc93749351)

[2. Déclaration de variables et constantes 2](#_Toc93749352)

[Syntaxes 2](#_Toc93749353)

[Quelques exemples 2](#_Toc93749354)

[3. Instructions PL/PGSQL 3](#_Toc93749355)

[3.1 Les instructions alternatives 3](#_Toc93749356)

[Syntaxes 3](#_Toc93749357)

[Exemples 3](#_Toc93749358)

[3.2 Boucle LOOP 4](#_Toc93749359)

[3.3 Boucle FOR 4](#_Toc93749360)

[3.4 Boucle WHILE 4](#_Toc93749361)

[4. Affectation (assignation) 5](#_Toc93749362)

[5. Déclaration d’une fonction 5](#_Toc93749363)

[5.1 Type de retour « returns » void : Procédure 5](#_Toc93749364)

[6. Curseurs PL/pgSQL 6](#_Toc93749365)

[7.1 Déclaration d’un curseur 6](#_Toc93749366)

[7.2 Ouverture de curseurs 6](#_Toc93749367)

[7.3 Utilisation de curseurs 7](#_Toc93749368)

[7.4 Fermeture du curseur 7](#_Toc93749369)

[8. Déclencheur (Trigger) 7](#_Toc93749370)

[8.1 Caractéristiques d’un Trigger 7](#_Toc93749371)

[8.3 Activer/Désactiver un Trigger 8](#_Toc93749372)

[8.4 Supprimer un Trigger 8](#_Toc93749373)

# Le langage PL/PGSQL

* PL/pgSQL (Procedural Language/PostgreSQL) est un langage procédural du système de base de données PostgreSQL.
* Permet de créer des procédures ou fonctions ou des déclencheurs, et d’ajouter des structures de contrôle afin d’effectuer des traitements complexes.
* Le langage procédural a pour avantage de grouper des requêtes et traitements pour les envoyer au serveur afin qu’il le traite et ainsi gagner du temps.
* Un programme PLSQL peut être :
  + Un bloc anonyme (algorithme structuré : DECLARE – BEGIN – […] – END, et il n’y a pas d’enregistrement)
  + Une fonction (comme une procédure mais qui retourne un résultat avec RETURN)
  + Un déclencheur (TRIGGER programme évènementiel avec exécution automatique au moment de l’évènement INSERT ou UPDATE ou DELETE)

# Déclaration de variables et constantes

Les variables peuvent être de n’importe quel type cependant leurs noms doivent être différents des noms de colonnes et doit commencer par une lettre (< 30 caractères) avec PostgeSQL.

### Syntaxes

**CREATE**

**Nom\_variable** [CONSTANT] **type** [CONTRAINTE] {[DEFAULT | := } expression]

### Quelques exemples

id\_utilisateur **integer**;

quantité **numeric**(**5**);

url **varchar**;

ma\_ligne nom\_table**%ROWTYPE**; // fait référence aux types de toutes les colonnes de la table

mon\_champ nom\_table.nom\_colonne**%TYPE**; // fait référence au champ de la colonne de la table indiquée

# Instructions PL/PGSQL

## 3.1 Les instructions alternatives

Les instructions IF et CASE permettent d'exécuter des commandes basées sur certaines conditions.

### Syntaxes

* **3 formes IF**

**IF** condition1 **THEN** instruction1 **END IF**

**IF** condition1 **THEN** instruction1 **ELSE** instruction2 **END IF**

**IF** condition1 **THEN** instruction1 **ELSIF** condition2 **THEN** instruction2 **ELSE** instruction3 **END IF**

* **2 formes CASE**

**CASE** variable1 **WHEN** valeur1 **THEN** instruction1 **ELSE** instruction par défaut **END CASE**

**CASE WHEN** valeur1 **THEN** instruction1 **ELSE** instruction par défaut **END CASE**

### Exemples

**CASE** x

**WHEN** 1, 2 **THEN**

msg := 'un ou deux';

**ELSE**

msg := 'autre valeur que un ou deux';

**END CASE**;

**IF** expression-booleenne **THEN**

instructions

**END IF**;

IF expression-booleenne THEN

instructions

ELSE

instructions

END IF;

### 3.2 Boucle LOOP

**Exemple :**

**LOOP**

instructions

**IF** nombre > 0 **THEN**

**EXIT**; //sortie de boucle

**END IF**;

**END LOOP**;

LOOP crée boucle inconditionnelle répétée indéfiniment jusqu'à une instruction **EXIT** ou **RETURN.**

**Syntaxe**

**LOOP**

instructions

**END LOOP** ;

### 3.3 Boucle FOR

FOR crée une boucle qui effectue une itération sur une plage de valeurs entières.

**Syntaxe**

**FOR** nom\_variable **IN** borne\_inf .. borne\_sup

**LOOP** instruction

**END LOOP**;

**Exemple :**

**FOR** i **IN** 1..10 LOOP //prend les valeurs de 1 à 10 avec une indice d’itération 1 dans la boucle

**END LOOP**;

### 3.4 Boucle WHILE

WHILE répète une séquence d'instructions aussi longtemps que la condition est évaluée à vrai.

**Exemple :**

**WHILE** montant\_possede > 0 **AND** balance\_cadeau > 0

**LOOP** instructions

**END LOOP**;

**Syntaxe**

**WHILE** condition

**LOOP** instructions

**END LOOP**;

# Affectation (assignation)

L'affectation d'une valeur à une variable s'écrit :

Nom\_variable { := | = } expression;

**Exemple :**

taxe := sous\_total \* 0.06;

mon\_enregistrement.id\_utilisateur := 20;

# Déclaration d’une fonction

**CREATE FUNCTION** définit une nouvelle fonction. **CREATE OR REPLACE FUNCTION** crée une nouvelle fonction ou la remplace si elle existe déjà.

**Exemple :**

**CREATE FUNCTION** taxe\_ventes(sous\_total real) **RETURNS** real AS $$

BEGIN

**RETURN** sous\_total \* 0.06;

END;

$$ LANGUAGE plpgsql;

## 5.1 Type de retour « returns » void : Procédure

Une procédure est une fonction qui ne retourne rien. Il y a deux moyens de la déclarer :

**CREATE FUNCTION** stamp\_user(id int, comment text) **RETURNS void** AS $$

#variable\_conflict use\_variable

DECLARE

curtime timestamp := now();

BEGIN

UPDATE users SET last\_modified = curtime, comment = comment

WHERE users.id = id;

END;

$$ LANGUAGE plpgsql;

**create [or replace] procedure** procedure\_name(parameter\_list)

language plpgsql as $$

declare

-- variable declaration

begin

-- stored procedure body

end; $$

# Curseurs PL/pgSQL

Au lieu d'exécuter la totalité d'une requête à la fois, il est possible de créer un curseur qui encapsule la requête, puis lit le résultat de plusieurs lignes à la fois.

## 7.1 Déclaration d’un curseur

nom **CURSOR** [ ( arguments ) ] **FOR** requête

**Exemples**

DECLARE

curs1 refcursor;

curs2 CURSOR FOR SELECT \* FROM tenk1;

curs3 CURSOR (cle integer) FOR SELECT \* FROM tenk1 WHERE unique1 = cle;

## 7.2 Ouverture de curseurs

**OPEN** nom\_curseur **FOR** requête

**Exemples**

OPEN curs1 FOR SELECT \* FROM foo WHERE cle = ma\_cle;

## 7.3 Utilisation de curseurs

Une fois qu'un curseur a été ouvert, il peut être manipulé en utilisant FETCH : cela permet de parcourir ligne par ligne la table dans le curseur afin de pourvoir faire des traitements.

**FETCH** curseur **INTO** cible ;

**Exemple**

FETCH curs1 INTO rowvar;

FETCH curs2 INTO foo, bar, baz;

FETCH LAST FROM curs3 INTO x, y;

## 7.4 Fermeture du curseur

CLOSE est utilisé :

* Pour libérer des ressources avant la fin de la transaction
* Pour libérer la variable curseur pour pouvoir la réouvrir

**CLOSE** curseur ;

**Exemple :**

CLOSE curs1 ;

# 8. Déclencheur (Trigger)

## 8.1 Caractéristiques d’un Trigger

**CREATE TRIGGER** <nom\_trigger>

BEFORE | AFTER

DELETE OR INSERT OR UPDATE

ON <nom\_table>

[FOR EACH ROW]

EXECUTE PROCEDURE <nom\_func>

**CREATE TRIGGER** crée un nouveau déclencheur. Le déclencheur est associé à la table, à la vue ou à la table distante spécifiée et exécute la fonction *nom\_fonction* lorsque certains événements comme INSERT, UPDATE ou DELETE surviennent.

## 8.3 Activer/Désactiver un Trigger

DISABLE ALL TRIGGERS ;

AlTER TRIGGER nmo\_trigger ENABLE

## 8.4 Supprimer un Trigger

DROP TRIGGER nom\_trigger