Le langage PL/pgSQL

Fred Hémery

IUT Béthune
Département
Réseaux & Télécommunications

Base de Données (IC5) — 07/08

Introduction

- Un langage procédural permet au SGBD d'exécuter du code autre que des requêtes SQL.
- Le code est analysé par un interpréteur (qu'il faut ajouter à l'installation du SGBD).
- L'interpréteur dialogue ensuite avec le SGBD pour lui faire exécuter des requêtes SQL Le SGBD peut gérer de cette façon plusieurs langages procéduraux
- Pour installer un nouveau langage procédural avec postgresql, il faut être Administrateur de Base de Données et utiliser la commande suivante :

```
createlang plpgsql <nomBaseDonnees>
```


Introduction

- Le langage procédural PL/pgSQL permet :
  - La création de fonctions et de triggers
  - L'ajoute des structures de contrôles au langage SQL
  - Des calculs plus complexes
  - Une utilisation aisée
- Les avantages
  - Les performances sont améliorées
  - Intégration parfaite avec SQL (types de données, opérations, fonctions)
  - Les fonctions écrites sont portables dans tous les environnements postgresql

Introduction

- Le langage PL/pgSQL est structuré en bloc. Un bloc est défini de la manière suivante :

```
[ <<label>> ]
 [ DECLARE
      declarations ]
 BEGIN
      statements
 END;
```

- Les commentaires

```
--
-- un commentaire
--
/*
un autre commentaire
*/
```


La Déclaration

Déclarations :

- Toutes les variables utilisées dans le bloc doivent être déclarées dans la section déclaration.
- Les variables peuvent avoir le type des données SQL comme `INTEGER`, `VARCHAR`, `CHAR`

```
user_id INTEGER;
quantity NUMERIC(5);
url VARCHAR;
myrow tablename%ROWTYPE;
myfield tablename.fieldname%TYPE;
arow RECORD;
```

La syntaxe complète :

```
name [ CONSTANT ] type [ NOT NULL ] [ { DEFAULT | := } expr ];
```


La Déclaration

Les paramètres des fonctions et les alias

- Lorsque l'on utilise des paramètres pour exécuter une fonction, les paramètres sont nommés $1, $2, ... dans le bloc. Pour la lisibilité, il est important de renommer les paramètres en utilisant un alias.

```
name ALIAS FOR $n;
```

- Exemple

```
CREATE FUNCTION sales_tax(REAL) RETURNS REAL AS $$
 DECLARE
      subtotal ALIAS FOR $1;
 BEGIN
      return subtotal * 0.06;
 END;
 $$ LANGUAGE plpgsql;
```


La Déclaration

- Le type ligne
  - Une variable composée est appelée une variable `row`. Ce type de variable pourra contenir une ligne du résultat d'un `SELECT`. Un champ de la ligne pourra être accédé en utilisant la notation pointée, par exemple : `rowvar.field`.

```
name tablename%ROWTYPE;
```

- Le type `RECORD`
  - Une variable `RECORD` est une variable ligne sans avoir un type pré-défini
  - Il est utilisé avec les instructions `SELECT` et `FOR`

La Déclaration

Les attributs

- En utilisant les attributs `%TYPE` et `%ROWTYPE`, il est possible de déclarer une variable du type d'un attribut d'une table. Par exemple :

```
variable%TYPE
user_id    users.user_id%TYPE;

table%ROWTYPE
users_rec users%ROWTYPE;
```

Les instructions

L'assignation

```
identifier := expression;

user_id := 20; tax := subtotal * 0.06;
```

SELECT INTO

- Stocke dans une variable de type RECORD ou ROWTYPE le résultat d'une sélection. Si cette sélection renvoie plusieurs lignes (tuples), seule la première est conservée.
- ATTENTION : Il ne faut pas confondre cette instruction avec le SELECT INTO du SQL pur, qui elle stocke le résultat de la sélection dans une nouvelle table

Les instructions

Exemple d'utilisation de SELECT INTO

```
SELECT INTO myrec * FROM EMP WHERE empname = myname;
 IF NOT FOUND THEN
     RAISE EXCEPTION ''employee % not found'', myname;
 END IF;
```

Autre Exemple

```
DECLARE
     users_rec RECORD;
     full_name varchar;
 BEGIN
     SELECT INTO users_rec * FROM users WHERE user_id=3;

     IF users_rec.homepage IS NULL THEN
         -- pour de homepage, pour cet utilisateur
         -- return "http://"
         RETURN ''http://'';
     END IF;
 END;
```


Les instructions

- Exécuter une expression ou une requête sans utilisation du résultat

```
PERFORM query;
```

- Exemple d'utilisation de PERFORM

```
PERFORM create_mv('cs_session_page_requests_mv', my_query);
```

- Exécuter une requête dynamique

```
EXECUTE query-string;
```

- Exemple d'utilisation de EXECUTE

```
EXECUTE 'UPDATE tbl SET '
        || quote_ident(fieldname)
        || ' = '
        || quote_literal(newvalue)
        || ' WHERE ...';
```


Les instructions

- Obtenir le statut de l'exécution d'une instruction

```
GET DIAGNOSTICS variable = item [ , ... ] ;
```

- Actuellement l'item que l'on peut indiquer
  - ROW_COUNT : le nombre de tuples traités par la précédente requête
  - RESULT_OID : l'OID du dernier tuple inséré (uniquement après INSERT)
- Exemple d'utilisation

```
GET DIAGNOSTICS var_integer = ROW_COUNT;
```


Les structures de contrôle

- Permet d'ajouter une structure de contrôle au langage SQL qui n'en dispose pas
- RETURN
  - Permet de terminer un résultat en terminant l'exécution de la fonction
  - Si la fonction doit renvoyer un SETOF élément on utilisera : RETURN NEXT (qui ne termine pas la fonction) dans une boucle suivi d'un RETURN simple (qui termine la fonction)

```
RETURN NEXT expression;
RETURN expression;
```


Les structures de contrôle

- Les instructions conditionnelles

```
IF ... THEN
IF ... THEN ... ELSE
IF ... THEN ... ELSE IF
IF ... THEN ... ELSIF ... THEN ... ELSE
IF boolean-expression THEN
    statements
END IF;
```

- Un exemple

```
IF v_user_id <> 0 THEN
    UPDATE users SET email = v_email
     WHERE user_id = v_user_id;
 END IF;
IF v_count > 0 THEN
    INSERT INTO users_count(count) VALUES(v_count);
    return ''t'';
 ELSE
    return ''f'';
 END IF;
```


Les structures de contrôle

- Les boucles simples

```
[<<label>>]
 LOOP
     statements
 END LOOP;
```

```
LOOP
    -- some computations
    IF count > 0 THEN
        EXIT; -- exit loop
    END IF;
END LOOP;
```

- EXIT : Sort de la boucle englobante

```
EXIT [ label ] [ WHEN expression ];
```

```
LOOP
    -- some computations
    EXIT WHEN count > 0;
END LOOP;
```

```
BEGIN
    -- some computations
    IF stocks > 100000 THEN
        EXIT; -- illegal.
              -- Can't use EXIT outside
              -- of a LOOP
    END IF;
END;
```


Les structures de contrôle

- Les boucles WHILE

```
[<<label>>]
 WHILE expression LOOP
     statements
 END LOOP;
```

- un exemple

```
WHILE amount_owed > 0 AND gift_certificate_balance > 0 LOOP
    -- some computations here
 END LOOP;

 WHILE NOT boolean_expression LOOP
    -- some computations here
 END LOOP;
```

Les structures de contrôle

- La boucle FOR

```
[<<label>>]
FOR name IN [ REVERSE ] expression .. expression LOOP
    statements
END LOOP;
```

- un exemple

```
FOR i IN 1..10 LOOP
  -- some expressions here

    RAISE NOTICE ''i is %'',i;
END LOOP;

FOR i IN REVERSE 10..1 LOOP
    -- some expressions here
END LOOP;
```


Les structures de contrôle

- Boucle pour traiter le résultat d'une requête

```
[<<label>>]
FOR record | row IN select_query LOOP
    statements
END LOOP;
```

- un exemple

```
CREATE FUNCTION cs_refresh_mviews () RETURNS INTEGER AS '
DECLARE
    mv RECORD;
BEGIN
  PERFORM cs_log(''Refreshing materialized views...'');
  FOR mv IN SELECT * FROM cs_materialized_views ORDER BY sort_key LOOP
    -- Now "mv" has one record from cs_materialized_views
    PERFORM cs_log(''Refreshing materialized view ''
                || quote_ident(mv.mv_name)  || ''...'');
    EXECUTE ''TRUNCATE TABLE  '' || quote_ident(mv.mv_name);
    EXECUTE ''INSERT INTO''||quote_ident(mv.mv_name) ||'' ''
                || mv.mv_query;
  END LOOP;
  PERFORM cs_log(''Done refreshing materialized views.'');
  RETURN 1;
end;
' LANGUAGE 'plpgsql';
```


Les structures de contrôle : Les curseurs

- Un curseur permet de stocker la position dans le résultat d'une requête
- Déclaration

```
DECLARE
  curs1 refcursor;
  curs2 CURSOR FOR SELECT * from tenk1;
  curs3 CURSOR (key int) IS SELECT * from tenk1 where unique1 = key;
```

- L'ouverture : Avant l'utilisation d'un curseur, il faut l'ouvrir

```
OPEN unbound-cursor FOR SELECT ...;
OPEN unbound-cursor FOR EXECUTE query-string;

OPEN curs1 FOR SELECT * FROM foo WHERE key = mykey;
OPEN curs1 FOR EXECUTE ''SELECT * FROM '' || quote_ident($1);
```


Les structures de contrôle : Les curseurs

- Utilisation d'un curseur

```
FETCH cursor INTO target;

FETCH curs1 INTO rowvar;
FETCH curs2 INTO foo,bar,baz;
```

- Fermeture d'un curseur

```
CLOSE cursor;

CLOSE curs1;
```


Les structures de contrôle : Les curseurs

- Renvoyer un curseur

```
CREATE TABLE test (col text);
INSERT INTO test VALUES ('123');

CREATE FUNCTION reffunc(refcursor) RETURNS refcursor AS '
BEGIN
        OPEN $1 FOR SELECT col FROM test;
        RETURN $1;
END;
' LANGUAGE 'plpgsql';
```