Bases de données
PL/pgSQL – les bases

Silviu Maniu

2022/2023

Polytech App3

Contenu

PL/pgSQL

PL/pgSQL : Programming Language for (Postgre)SQL = SQL + langage procédural (impératif)

Rappel :

- **declaratif** (SQL) : dire **quoi**
- **procedural** : dire **comment**

PL/pgSQL : l'union entre SQL et un langage de programmation

PL/pgSQL – Pourquoi

Factorisation : faire fonction/procedure pour ne pas repeter des ordres SQL

Expressivité :

- tout n'est pas exprimable en SQL
- PL/pgSQL langage programmation = tout ce qui est calculable

Performance : PL/pgSQL envoyé au serveur une seule fois, compilé puis exécuté

PL/pgSQL – Pourquoi

Architecture en couches : sépare BD du reste de l'application

- traitements BD stockées dans la base (*procédure/fonction stockée*)
- **encapsulation** : chaque traitement est transparent pour l'utilisateur

Portabilité : toute la partie BD peut être changée sans conséquence sur l'interface (Web, programme)

PL/pgSQL – Comment

Etapes :

1. programmeur écrit des procédures et fonctions PL/pgSQL pour les traitements dans le cahier de charges
2. les procédures sont compilées et stockées dans la base
3. elles sont appelées quand besoin par les couches hautes (interface)

Depuis :

- mode interactif (ligne de commande, IDE, ...)
- mode programme (Java, PHP, etc.)

Contenu

Le langage PL/pgSQL

PL/pgSQL contient :

- partie **programmation** : les blocs, types, variables, operateurs, tests, boucles
- partie **BD** : ordres SQL, types

Exemple rapide :

```
create or replace procedure nombrelignes() as $$
declare
  nblig integer;
begin
  select count(*) into nblig from table;
  raise notice '%',nblig;
end;
$$ language plpgsql;
```

Le langage PL/pgSQL

Déroulement :

- chargement sur serveur = compilation + stockage
- **message résultat** : ok ou erreur de compilation
- **appel** : `execute nombrelignes`

Bloc PL/pgSQL

Possibilité d'exécuter directement du code PL/pgSQL en ligne commande, sous-bloc dans une procédure, etc.

Structure :

```
declare
  --variables, constantes, curseurs, etc.
begin
  -- ordres SQL, instructions PL/pgSQL, structures controle
end;
```

Bloc PL/pgSQL

Possibilité d'exécuter directement du code PL/SQL en ligne commande (**bloc anonyme**)

Exemple :

```
do $$ -- par defaut, plpgsql
  declare
    x varchar(10);
  begin
    x := 'Hello World';
    raise notice '%',x;
  end
$$;
```

PL/pgSQL – Variables

Déclarées après **`declare`** et avant **`begin`**

Variable = nom + type (+ contrainte BD)

Types de variable :

- tous types BD (**`integer`**, **`varchar`**, etc.)
- programmation : non-BD (booléen), dérivé

```
--programmation
adresse varchar(20);
x integer := 1;
--BD
nom varchar(10) not null;
```

PL/pgSQL – Variables

Types dérivés, référence à une entité existante

- programmation : variable
- BD : colonne, table, curseur

```
--programmation
x integer;
y x%type;
--BD
vnom client.nom%type;
cli client%rowtype;
```

PL/pgSQL – Variables

Affectation a une variable :

- programmation : résultat d'une **expression**
- BD : résultat **requête** (`select into` – seule ligne résultat, `fetch into` – curseur)

Exemple programmation :

```
x := 0;
vnom := 'Monsieur' || vnom; -- ou concat('Monsieur',vnom)
y := (x + 5) * y;
```

PL/pgSQL – Variables

Exemple BD :

```
declare
  vref varchar2(10);
  vprix articles.prixht%type;
  cli client%rowtype;
begin
  select refart, prixht
    into vref, vprix
    from articles
    where nom = 'cravate';
  select *
    into cli
    from client
    where nom = 'Toto';
end;
```

PL/pgSQL – Opérateurs, tests

Opérateurs :

- programmation : $=$, $<$, $>$, !, $>=$, $<=$, `between`, `like`, `and`, `or`, etc.
- BD : comme en SQL

Tests (`if-then-else`) :

```
if vnocli = 10 then
  update ... ;
  insert ... ;
[elsif ... then]
else delete ... ;
end if;
```

PL/pgSQL – Boucles

```
for n in 100 . . 110 loop
  insert into client(nocli) values (n);
end loop;

n := 100;
while n <= 110 loop
  insert into client(nocli) values (n);
  n := n + 1;
end loop;
```

PL/pgSQL – Procédures stockées (syntaxe)

```
create or replace procedure p
    (x1 in | out | inout t1, x2 ... , ...)
as $$
declare
   ... -- déclarations (variables)
begin
   ... -- corps (ordres SQL, tests, boucles, etc.)
end;
$$ language plpgsql;
```

Paramètres :

- `in` (par défaut) – variable entrée seulement
- `out` – variable sortie (référence)
- `inout` – variable entrée-sortie)

PL/pgSQL – Fonctions stockées (syntaxe)

```
create or replace function f (...)
    returns integer /* ou autre */
as $$
declare
  ...
begin
  ...
  return x;
end;
$$ language plpgsql;
```

Supprimer procedure/fonction (ordre SQL) :

- `drop procedure p`
- `drop function f`

PL/pgSQL – Appels

Depuis bloc anonyme :

```
do $$
declare
        x char;
begin
  call suppr(x);
  ...
end;
$$;
```

SQL*Plus :

```
call suppr('W');
```

Note : pour appeler une fonction il faut utiliser un bloc anonyme (voir feuilles exemples)

Contenu

Vues – principes

Vues – aspect clé d'une bonne conception BD

- peut **encapsuler** les requêtes sans paramètres
- peut cacher les détails des tables origine
- les vues sont **persistentes**

Definie par rapport a une requête

```
create view nom_view as requête;
select * from nom_view;
```

- requête executée chaque à chaque `SELECT` dans la Vues
- les modification dans les tables d'origine se propagent dans la vue
- meme principe qu'une fonction sans paramètres

Vues – exemple

Vue qui contient les références et prix des cravates

```
create view view_cravates as
  select refart, prixht from articles
  where nom = 'cravate';

select * from view_cravates;
```

Compétences à acquérir

Analyse/implementation : savoir écrire des procédures et fonctions stockées utilisant des ordres SQL et des éléments de programmation (test, boucles, etc.)

Étapes :

1. écrire les ordres BD nécessaires
2. choisir procédure, fonction ou vue
3. choix variables et types (paramètres, passage information entre ordres)
4. appel, debug, etc.

Remerciements

Le contenu de ce cours est grandement inspiré des cours d'Emmanuel Waller

- `https://www.lri.fr/~waller/`