420-C42

Langages d'exploitation des bases de données

Partie 17

PL/pgSQL I

Introduction

PL/pgSQL I

Introduction

- Le langage SQL est un langage déclaratif et, même si les avantages sont nombreux, il peut présenter certaines limitations lorsqu'on le compare à un langage procédural.
- Les grands SGBDR supportent non seulement le langage SQL mais offrent aussi un langage procédural d'appoint :
  - Oracle : PL/SQL (*Procedural Langage*/SQL)
  - MySQL : *aucun nom formel mais ressemble à PL/SQL tout en étant plus limité*
  - MS SQL : T-SQL (Transact-SQL)
  - PostgreSQL : PL/pgSQL (*Procedural Langage*/pgSQL)
  - SQLite : n'offre pas cette possibilité

PL/pgSQL I

Introduction

- Il faut comprendre que ces ajouts sont tous différents les uns des autres. Toutefois, la plupart ressemblent au PL/SQL d'Oracle dont ils s'inspirent (sauf T-SQL).
- Par conséquent, les scripts écrits pour un SGBD ne sont pas directement compatibles les uns avec les autres. Il existe des traducteurs qui permettent de faire le passage d'un à l'autre (souvent imparfaits).

PL/pgSQL I

Sommaire

- PL/pgSQL permet :
  - L’utilisation de constantes, de variables et de structures de contrôle.
  - L’utilisation combinée du SQL et PL/pgSQL.
  - Comme le permettent les procédures et les fonctions de SQL, le PL/pgSQL permettra d’augmenter encore plus :
    - **l’intégrité**
    - la sécurité
    - la performance
    - la facilité d’utilisation de la BD
    - la centralisation

PL/pgSQL I

Avantages et inconvénients

- Avantages :
  - Puisqu'il est possible d'effectuer des tâches complexes du côté serveur, la centralisation de ces opérations permet l'accroissement de :
    - l'intégrité : permet d'ajouter des contraintes impossibles par les 6 contraintes existantes
    - la performance :
      - les requêtes SQL sont préparées
      - pas besoin de transférer des informations du côté client pour effectuer des analyses complexes
    - la sécurité : possibilité de définir des accès limités aux objets de la BD pour les utilisateurs
    - la productivité : facilite énormément l'utilisation de la BD
  - Permet d'automatiser des tâches.
- Désavantages :
  - Plus complexe à mettre en place, à tester et à maintenir.
  - En centralisant plus d'opérations du côté serveur, ce dernier peut devenir sur-utilisé.

PL/pgSQL I

Syntaxe générale

- On peut déclarer du code PL/pgSQL dans plusieurs contextes :
  - bloc anonyme
  - procédure et fonction
  - fonction dédiée aux déclencheurs (*trigger*).
- Dans tous les cas, il existe trois parties (toujours dans une chaîne de carcatères) :
  - DECLARE : zone déclarative
    - elle sert à déclarer **toutes** les variables locales
    - zone optionnelle, on ne l'utilise pas si aucune variable n'est requise
  - BEGIN : corps des opérations
    - zone obligatoire
    - présente le corps des opérations du bloc/procédure/fonction
  - EXCEPTION : zone de gestion des exceptions
    - zone optionnelle, on ne l'utilise pas si aucune gestion d'exception n'est nécessaire
  - END : fin de la zone déclarative

PL/pgSQL I

Syntaxe générale

- Insensible à la casse.
- Les commentaires sont les mêmes :
  - -- commentaire de fin de ligne
  - /* commentaire multilignes */
- On utilise généralement les « *dollar-quoted* » $$ pour faciliter l'écriture du code qui est **toujours** sous forme de chaîne de caractères.

PL/pgSQL I

Outils de débogage – Débogueur

- Il existe un outil de débogage qui peut être installé à titre d'extension avancée à PostgreSQL.
- Toutefois, cet outil n'est pas toujours disponible et d'autres SGBDR n'offrent aucun outil du genre. La seule alternative reste souvent l'affichage de messages pertinents.

PL/pgSQL I Outils de débogage – Affichage

- Il est possible d'afficher simplement un message avec l'instruction RAISE NOTICE (nous y reviendrons plus en détail) :

```
RAISE NOTICE 'Le message ici';
RAISE NOTICE 'Un message avec deux variables % et %', var1, var2;
```

- Il est important de comprendre qu'un message exclusivement destiné au débogage ne devrait jamais rester en production.
- Toutefois, cette approche peut être très pratique dans certaines situations de débogage.
- Présente une certaine similarité à des outils d'autres langages comme :
  - print(…)
  - std::cout << …

PL/pgSQL I

Outils de débogage – ASSERT

- PostgreSQL implémente la stratégie de validation conditionnelle « *assert* », ce qui peut être vraiment pratique (selon les pratiques de l'entreprise).
- Cette stratégie permet la validation de conditions selon deux modes d'exécution spéciaux. L'approche va comme suit :
  - À n'importe quel endroit du code, on utilise ASSERT pour valider une expression conditionnelle.
  - Souvent, on valide qu'une ou plusieurs variables aient ou n'aient pas certaines valeurs. Souvent la validation des conditions de l'exécution normale d'une fonction.
  - Lorsque le code s'exécute en mode ASSERT-ON, les validations ASSERT sont effectuées :
    - si ASSERT retourne vrai, l'exécution continue sans conséquence
    - si ASSERT retourne faux, l'exécution est interrompue et, optionnellement, un message personnalisé est affiché
  - Lorsque le code s'exécute en mode ASSERT-OFF, les validations ASSERT ne sont pas effectuées.

PL/pgSQL I

Outils de débogage – ASSERT

- Voici un exemple :

…

ASSERT *condition*, 'Un message simple ici';

ASSERT *condition*, 'Un message avec 2 variables % et %', var1, var2;

…

- La variable d’environnement plpgsql.check_asserts détermine si le mode ASSERT est actif ou non. Voir :
  - SHOW plpgsql.check_asserts;
  - SET plpgsql.check_asserts TO on;
  - SET plpgsql.check_asserts TO off;

PL/pgSQL I

Bloc anonyme

- Un bloc anonyme est un bloc d'instructions exécuté une seule fois et immédiatement après sa déclaration.
- Le synopsis est simple :

```
DO [ LANGUAGE nom_du_langage ] '_le_bloc_d'exécution_';
DO '_le_bloc_d'exécution_' [ LANGUAGE nom_du_langage ];
```

- Exemple :

```
DO $$                                   -- on utilise parfois un nom représentatif $bloc_anonyme$
LANGUAGE plpgsql
BEGIN
    RAISE NOTICE 'Sur le point d'ajouter...';
    INSERT INTO employe VALUES('Bob');
    RAISE NOTICE '% a été ajouté!', 'Bob';
END
$$;
```