PROJET SQL : PROCÉDURES STOCKÉES ET DÉCLENCHEURS

1. Introduction

Les procédures stockées permettent :

- d'ajouter des structures de contrôle (condition, boucle, etc.) au langage SQL;
- d'effectuer des traitements complexes;

Elles sont nécessaires pour la définition des déclencheurs (appelés *triggers* en anglais). Un déclencheur permet d'appeler une fonction en cas de survenu d'un événement (par exemple lors de la suppression d'une ligne dans une table).

Remarque : on parle ici de procédures stockées car c'est l'expression consacrée en SQL mais il faut comprendre que ces procédures peuvent être aussi des fonctions c'est-à-dire qu'elles peuvent renvoyer des valeurs.

Grâce aux procédures stockées on peut grouper un bloc de traitement et une série de requêtes *au sein* du serveur de bases de données. On évite ainsi la charge de la communication entre le client et le serveur.

En effet, chaque requête SQL émise par le client doit être exécutée individuellement par le serveur de bases de données. Cela signifie que l'application cliente doit envoyer chaque requête au serveur de bases de données, attendre que celui-ci la traite, recevoir les résultats, faire quelques traitements, et enfin envoyer d'autres requêtes au serveur. Tout ceci induit des communications entre les processus et induit une surcharge du réseau.

Avec PostegreSQL, il existe plusieurs langages pour définir les procédures stockées. Nous utiliserons PL/pgSQL.

Remarque : le thème des procédures stockées est un peu complexe. Il vous faut absolument consulter la documentation de PostgreSQL 9.3. Pour ce faite aller sur le site http://postgresqlfr.org/ et consulter dans la documentation la partie V – Programmation serveur (et en particulier la partie 40).

2. Les fonctions

2.1. Exemple avec une valeur de retour unique

Exemple :

```
create function carre(x integer) returns integer as '
 /* cette fonction renvoie le carré de x
    on aurait pu aussi écrire : create or replace function carre ... */
declare
  -- déclaration des variables locales
  res integer;

begin
  res = x * x;
  return res;
end;
' language plpgsql;
```

Remarques :

- Pour déclarer une fonction, il faut utiliser l'instruction : « `create` ». On peut aussi utiliser la commande « `create or replace` » ;
- Pour supprimer une fonction, utilisez la commande « `drop function nom_fct` » ;
- Commentaires : utilisez la syntaxe /* ... */ pour plusieurs lignes ou -- (deux tirets) pour une seule lignes ;
- Retour avec « `return` » : renvoie un seul résultat et quitte la fonction ;
- Pour tester la fonction, entrer la commande : `select carre(5) ;`
- La fonction « `carre` » pourra être utiliser dans une autre fonction dans une expression du genre : `z=carre(7)*(3+carre(4)*5)`.

2.2. Structures de contrôle

On retrouve les structures bien connues :

La condition :

```
if then
loop
```

La boucle « tant que » :

```
while expression loop
    instructions
end loop;
```

La boucle « pour » :

```
for nom in [ reverse ] expression .. expression loop
    instruction
end loop;
```

Exercice 1 : écrivez une fonction `estPaire` qui renvoie vrai si le nombre passé en paramètre est pair. L'opérateur modulo est représenté par le symbole %.

Exercice 2 : écrivez une fonction qui renvoie le cube d'un nombre passé en paramètre. Vous devez réutilisez dans cette fonction la fonction `carre`. Vous utiliserez la commande `create or replace` pour créer la fonction.

Exercice 3 : écrivez une fonction qui calcule la puissance d'un nombre (ex. : 2^5) en utilisant une structure `while`.

Exercice 4 : écrivez une fonction qui renvoie la factorielle d'un nombre en utilisant une structure `for`.

2.3. Exemple avec plusieurs valeurs de retour

```
CREATE OR REPLACE FUNCTION chercher(prenom varchar) RETURNS setof varchar AS '
declare
  ligne record;
  compteur integer;
```

```
begin
  for ligne in select * from client loop
    compteur := 0;

    if ligne.prenom = prenom
    then
     compteur := compteur + 1;
       return next ligne.nom;
    end if;
  end loop;

  return;
end;
' LANGUAGE plpgsql;
```

Remarque : le code de la fonction est encadré par des caractères `'`. Si on veut utiliser le caractère `'` à l'intérieur de la fonction alors il faudra le doubler (ce qui donne : `''`).

Pour essayer la fonction :
```
select * from chercher('Caroline');
```

3. Les déclencheurs

3.1. Définitions

Un déclencheur (ou *trigger*) permet de réaliser un traitement sur la base de données quand un événement de type insertion, suppression ou mise à jour survient.

Pour mettre en place le déclencheur, il faut procéder en deux temps :
- d'abord définir une fonction en respectant quelques règles,
- puis définir le déclencheur lui-même qui appellera la fonction précédemment définie.

Une fonction déclencheur peut être définie pour s'exécuter avant ou après une commande `INSERT`, `UPDATE` ou `DELETE`

Elle peut être définie pour être exécutée soit une fois par ligne modifiée (déclencheur de niveau ligne) soit une fois par expression SQL (déclencheur de niveau instruction).

La fonction déclencheur doit être définie avant que le déclencheur lui-même puisse être créé. La fonction déclencheur doit être déclarée comme une fonction ne prenant aucun argument et retournant le type `trigger`.

Quand une fonction PL/pgSQL est appelée en tant que *trigger*, plusieurs variables spéciales sont créées automatiquement. Ce sont entre autre :

- NEW : type de données RECORD; variable contenant la nouvelle ligne de base de données pour les opérations INSERT/UPDATE dans les déclencheurs de niveau ligne. Cette variable est *NULL* dans un *trigger* de niveau instruction.
- OLD : type de données `RECORD`; variable contenant l'ancienne ligne de base de données pour les opérations `UPDATE`/`DELETE` dans les *triggers* de niveau ligne. Cette variable est *`NULL`* dans les *triggers* de niveau instruction.

Une fois qu'une fonction déclencheur est créée, le déclencheur (*trigger*) est établi avec CREATE TRIGGER. La même fonction déclencheur est utilisable par plusieurs déclencheurs.

Il existe deux types de déclencheurs : les déclencheurs en mode ligne et les déclencheurs en mode instruction. Dans un déclencheur mode ligne, la fonction du déclencheur est appelée une fois pour chaque ligne affectée par l'instruction qui a lancé le déclencheur.

Un déclencheur en mode instruction n'est appelé qu'une seule fois lorsqu'une instruction appropriée est exécutée, quelque soit le nombre de lignes affectées par cette instruction. En particulier, une instruction n'affectant aucune ligne résultera toujours en l'exécution de tout déclencheur mode instruction applicable. Ces deux types sont quelque fois appelés respectivement des « déclencheurs niveau ligne » et des « déclencheurs niveaux instruction ».

Le moment où un déclencheur est exécuté a aussi son importance. En effet, pour une instruction donnée le déclencheur peut être exécuté avant ou après l'instruction. C'est au programmeur de spécifier ce moment.

Les déclencheurs « avant » en mode instruction se lancent naturellement avant que l'instruction ait fait une modification alors que le déclencheur « après » en mode instruction se lance à la fin de l'instruction. Les déclencheurs « avant » en mode ligne s'exécutent immédiatement avant qu'une ligne particulière ne soit traitée alors que les déclencheurs « après » en mode ligne se déclenchent à la fin de l'instruction (mais avant tout déclencheur « après » en mode instruction).

3.2. Déclencheur en cascade

Si une fonction déclencheur exécute des commandes SQL, alors ces commandes peuvent relancer des déclencheurs. On appelle ceci un déclencheur en cascade. Il n'y a pas de limitation directe du nombre de niveaux de cascade. Il est possible que les cascades causent une invocation récursive du même déclencheur . Par exemple, un déclencheur `INSERT` pourrait exécuter une commande qui insère une ligne supplémentaire dans la même table, entraînant un nouveau lancement du déclencheur `INSERT`.

Il est de la responsabilité du programmeur d'éviter les récursions infinies dans de tels scénarios.

3.3. Exemples

```
create function minEtMaj() returns trigger as
'
-- cette fonction force le nom en majuscule et le prénom en minuscule
begin
  new.nom = upper(new.nom);
  new.prenom = lower(new.prenom);
  return new;
end;
' language plpgsql;

create trigger decMinEtMaj before insert or update on client
```

```
for each row execute procedure minEtMaj();


--pour supprimer le trigger
drop trigger decMinEtMaj on joueur;
--pour supprimer la fonction (attention! il faut supprimer le trigger associé
avant)
drop function minEtMaj();


CREATE or replace FUNCTION essai() RETURNS trigger as $essai$
declare
  nb integer;

begin
  select into nb count(*)
  from adresse
  where idClient=new.id;
  raise NOTICE '%', nb; -- affiche le nombre d'adresse du client que l'on est en
train d'ajouter ou de mettre à jour

  if nb=0
  then
    return null; -- au cas le nombre d'adresse est nulle on ne fait pas
l'insertion ou la mise à jour
  end if;

  return new;
end;
$essai$ LANGUAGE plpgsql;
```

Remarque : dans cet exemple, le code de la fonction est délimité par la chaîne `$essai$` ce qui évite le problème du caractère '.

```
drop trigger dec_essai on client;
CREATE TRIGGER dec_essai
    before INSERT OR UPDATE on client
    FOR EACH ROW EXECUTE PROCEDURE essai();
```

Pour essayer :

```
update client set prenom='toto' where id=5;
select * from client where id=5;
```

Exercice 5 :

Dans la base de données de Reims Outillage, il est possible d'insérer (ou de modifier) une commande avec une adresse de livraison qui ne correspond pas à une adresse correspondant au client qui a passé la commande.

Concevez un déclencheur qui permet d'éviter cette situation.