PostgreSQL Avancé !

Michel Meynard

22 mai 2007

1 Règles PostgreSQL

Les règles permettent de réécrire les commandes select, insert, update, delete réalisées sur une table ou sur **une vue**. Par défaut les vues ne sont pas modifiables dans PostgreSQL (contrairement à Access ou Oracle). La syntaxe est :

```
CREATE [ OR REPLACE ] RULE nom AS ON événement
    TO table [ WHERE condition ]
    DO [ ALSO | INSTEAD ] { NOTHING | commande | ( commande ; commande ... ) }
```

Par exemple, avec le schéma de bd suivant :

```
ANIMAL(N_A, NOM, AGE, RACE, ORIGINE, N_MENU) avec
   ANIMAL(N_MENU) -> MENU(N_M)
MENU(N_M, QTE_VIANDE, QTE_LEGUME)
```

Si l'on crée la vue `animal_menu` de la façon suivante :

```
CREATE OR REPLACE VIEW animal_menu AS
 SELECT a.n_a, a.nom, a.age, a.race, a.origine, a.n_menu,
        m.qte_viande, m.qte_legume
   FROM animal a, menu m
  WHERE a.n_menu = m.n_m;
```

L'insertion dans la vue d'un nouvel animal **et** d'un nouveau menu sera possible après écriture de la règle :

```
create or replace rule insert_ani_menu AS ON insert to animal_menu
do instead (
insert into menu(n_m, qte_viande, qte_legume) values
(new.n_menu, new.qte_viande, new.qte_legume);
insert into animal(n_a, nom, age, race, origine, n_menu) values
(new.n_a, new.nom, new.age, new.race, new.origine, new.n_menu);
);
```

Après exécution de la requête suivante :

```
insert into animal_menu (n_a, nom, age, race, origine, n_menu,
  qte_viande, qte_legume) values
  (10, 'toto',23,'LION', 'AFRIQUE', 12,100,0);
```

Un nouvel animal et un nouveau menu ont été créés ! Attention, si l'une des deux insertions ne peut avoir lieu (violation de contrainte), aucun des deux ne sera exécuté : la suite de commandes est exécuté dans une transaction.

2 PL/pgSQL

On peut créer des fonctions ou procédures dans un langage de programmation avec la commande SQL :

```
CREATE [ OR REPLACE ] FUNCTION
    nom ( [ [ modearg ] [ nomarg ] typearg [, ...] ] )
    [ RETURNS type_ret ]
  { LANGUAGE nomlang
    | AS 'definition'
  } ...
```

Le mode d'un argument : soit IN, soit OUT, soit INOUT. En cas d'omission, la valeur par défaut est IN. Si on veut utiliser le langage PL/pgSQL pour la 'definition' :

```
CREATE OR REPLACE FUNCTION mult(i integer, j integer)
  RETURNS integer AS
$BODY$
begin
return i * j;
end;
$BODY$
  LANGUAGE 'plpgsql' VOLATILE;
```

Notez les apostrophes `$BODY$` afin d'éviter le double apostrophage à l'intérieur de la fonction. A l'utilisation :

```
SELECT mult(2,3);
6
```

PL/pgSQL est un langage structuré en blocs. Le texte complet de la définition d'une fonction doit être un bloc. Un bloc est défini comme :

```
[ DECLARE
    déclarations ]
BEGIN
    instructions
END;
```

2.1 Déclencheurs

Pour créer un déclencheur (trigger) qui s'exécute avant ou après une commande sql sur une table, il faut :

1. créer une **fonction déclencheur** ou procédure déclencheur (`CREATE FUNCTION`) ayant les propriétés suivantes :
   - aucun argument ;
   - un type de retour `trigger` ;
   - Une fonction déclencheur doit renvoyer soit NULL soit une valeur record/ligne ayant exactement la structure de la table pour laquelle le déclencheur a été lancé (souvent NEW) ;
   - plusieurs variables spéciales sont créées automatiquement :

     NEW Type de données RECORD ; variable contenant la nouvelle ligne de base de données pour les opérations INSERT/UPDATE ;

     OLD ancienne ligne de base de données pour les opérations UPDATE/DELETE ;

     TG_ARGV [] arguments de l'instruction CREATE TRIGGER ;
2. créer un trigger (`CREATE TRIGGER`) qui exécute cette procédure avant ou après exécution d'une requête SQL.

2.1.1 Exemple

On veut créer un déclencheur qui vérifie avant insertion d'un nouveau menu qu'un menu ayant les mêmes caractéristiques n'existe pas déjà dans la table. On commence par créer la fonction déclencheur :

```
CREATE OR REPLACE FUNCTION verif_menu_existant()
  RETURNS "trigger" AS
$BODY$
DECLARE
nm menu.n_m%type;
BEGIN
select n_m into nm
from menu
where qte_viande=NEW.qte_viande and
  qte_legume=NEW.qte_legume;
IF FOUND THEN
  RAISE EXCEPTION 'La table menu contient déjà le menu % avec les mêmes quantités !', nm;
else
  return NEW;
END IF;
END;
$BODY$
  LANGUAGE 'plpgsql' VOLATILE;
```

Quelques remarques :
- pas de paramètres, retourne "trigger" ;
- déclaration de la variable nm du même type que le numéro de menu dans la table menu ;
- exécution d'une requête select ... into nm ;
- s'il existe un menu ayant les mêmes quantités, alors lever une exception qui va afficher un message et avorter la transaction : aucune insertion !
- sinon on retourne la nouvelle ligne (NEW) ;

Remarquons que cette fonction déclencheur ne peut être appelée avec select comme la fonction mult précédente : il faut créer un trigger qui l'exécute !

```
CREATE TRIGGER avant_insert
  BEFORE INSERT
  ON menu
  FOR EACH ROW
  EXECUTE PROCEDURE verif_menu_existant();
```

Remarquons que la procédure déclencheur pourrait être écrite dans un autre langage que plpgsql! A l'exécution, on obtient :

```
insert into menu values (10,20,0);
ERROR: La table menu contient déjà le menu 4 avec les mêmes quantités !
État SQL :P0001

insert into menu values (13,20,50);
La requête a été exécutée avec succés : 1 lignes modifiées.
```