Les triggers
(tiré de http://www.linuxfrench.net)

Fred Hémery

IUT Béthune
Département
Réseaux & Télécommunications

Base de Données (IC5) — 07/08

Introduction

- Les triggers (aussi appelés déclencheurs en français) sont, au même titre que les contraintes, un élément fondamental dans la structure des bases de données. Alors qu'une contrainte permet de définir une règle algébrique, le trigger permet de définir une règle algorithmique. Il peut également être très utile d'un point de vue purement fonctionnel.
- Le trigger est une fonction affectée à une table et qui est déclenchée sur certaines opérations.
- Un trigger peut être déclenché avant ou après une instruction de type `INSERT`, `DELETE` ou `UPDATE`. De plus, il peut être appelé pour chacune des lignes concernées par l'instruction, ou une seule fois pour l'instruction.

Introduction

- Contrainte logique.
  - Besoin de définir des contraintes de type « soit l'un, soit l'autre ». C'est-à-dire deux champs qui sont des clefs étrangères, mais pour un tuple donné il n'est pas possible d'avoir les deux champs renseignés. Lors de l'écriture d'une valeur dans ce champ, il y a vérification que cette condition sera valide avec la nouvelle valeur.
- Utilité fonctionnelle.
  - Besoin de garder une trace des versions successives d'un champ texte.
    - Gérer cela au sein de votre application,
    - le plus fiable et le plus simple sera de définir un trigger, déclenché sur le `UPDATE`, qui ira archiver l'ancienne version du texte avant de le mettre à jour. (Il sera aussi nécessaire d'effectuer un archivage sur le `DELETE`).

Création d'un trigger

- La première étape sera de créer une procédure stockée. Cette procédure (ou fonction) constitue le trigger proprement dit. Cette fonction n'admet aucun argument, et retourne le type TRIGGER.
- Lorsqu'une fonction est appelée en tant que trigger, elle hérite automatiquement lors de l'exécution d'un certain nombre de variables particulièrement utiles(`TriggerData`), parmi lesquelles on peut citer :
  - `NEW` : La ligne telle qu'elle sera après l'exécution de l'instruction qui a déclenché le trigger.
  - `OLD` : L'état de la ligne avant l'exécution de l'instruction qui a déclenché le trigger.

Création d'un trigger

- Comme on l'a vu, la fonction associée au trigger pourra être exécutée :
  - Pour chaque ligne '*touchée*' par la requête qui a déclenché le trigger (*row-level trigger*)
  - Une fois pour la requête qui a déclenché le trigger (*statement-level trigger*)
- De plus la fonction peut être exécutée
  - Avant la prise en compte de la requête ou
  - Après
- Un *statement-level trigger* renvoie toujours une valeur nulle
- Un *row-level trigger* exécuté renvoie un tuple (une ligne de la table affectée par la requête)
- Un *row-level trigger* exécuté après la requête peut renvoyer des données, mais elles ne sont pas prises en compte par le système. Il est préférable de renvoyer une valeur nulle.

Création d'un trigger

- Un *row-level trigger* exécuté avant la requête peut renvoyer une valeur nulle. Cela indique à l'exécuteur de la requête qu'il ne doit pas effectuer cette opération pour la ligne (tuple) courante.
- Pour les requêtes de type `UPDATE` ou `INSERT` le tuple renvoyé représente le tuple qui sera inséré (ou le nouveau tuple) après traitement de la requête.
- Une fonction associée au trigger qui ne veut pas modifier le comportement de la requête doit renvoyer sans aucune modification le tuple qu'elle a reçu en entrée. (la variable NEW pour les requêtes `INSERT` et `UPDATE` et `OLD` pour les requêtes `DELETE`)

Les procédures stockées

Une entité fonctionnelle : C'est un morceau de code fait une fois pour toutes qui va s'occuper d'effectuer une opération qui a une signification fonctionnelle (plusieurs requêtes).

- Par exemple

```
create table comptes(no integer primary key, solde decimal(10,2));
```

- En cas de virement on utilise la fonction suivante :

```
create function virement(integer,integer,decimal) returns integer
AS '
DECLARE crediteur ALIAS FOR $1;
DECLARE debiteur ALIAS FOR $2;
DECLARE montant ALIAS FOR $3;
BEGIN
  UPDATE comptes SET solde=solde+montant WHERE no=crediteur;
  UPDATE comptes SET solde=solde-montant WHERE no=debiteur;
  return 0;
END;
'   LANGUAGE 'plpgsql';
```


Les procédures stockées

- la structure générale de création d'une procédure (fonction) :

```
CREATE FUNCTION nom_fonction(parametres)
  RETURNS type AS 'code de la fontion'
LANGAGE 'nom du langage' ;
```

- Pour appeler la fonction

```
select virement(1,2,100);
```

ou encore (plus rigoureux)

```
BEGIN TRANSACTION; select virement(1,2,100); COMMIT;
```

- Le langage pour écrire une fonction stockée doit avoir été déclaré dans le SGBD.
  - Le langage C
  - La syntaxe complète du langage `pl/pgSQL` se trouve en ligne. Elle est très proche du `pl/SQL` d'Oracle.
- Une fonction associée à un trigger est une fonction stockée qui n'accepte aucun paramètre et qui renvoie une variable de type `TRIGGER`

Création d'un trigger

- Nous allons nous créer deux tables, une table pour y stocker des articles, et une autre table d'archivage afin de stocker les versions successives des articles :

```
create table article(id integer not null primary key, date_modif date not
  null default now(), texte text, status varchar);
create table archive(id
  integer, date_modif date not null, texte text);
```

- Maintenant, nous allons créer une fonction qui sera appelée lors d'une modification d'un article afin d'archiver la version précédente de l'article :

```
CREATE FUNCTION arch_art() RETURNS TRIGGER AS '
BEGIN
        IF NEW.texte != OLD.texte THEN
                INSERT INTO archive(id, date_modif, texte)
                VALUES (OLD.id,OLD.date_modif,OLD.texte);
        END IF;
        RETURN NEW;
END;
' LANGUAGE 'plpgsql';
```


Installation d'un trigger

- La dernière ligne (`RETURN NEW`), retourne la ligne telle qu'elle devra être écrite dans la base de données. Dans notre cas, elle reste inchangée par rapport à l'ordre `UPDATE` original.
- On remarque que le trigger sera appelé pour tout ordre `UPDATE`, et qu'il est donc nécessaire de vérifier que le texte de l'article a bel et bien été modifié avant de l'archiver.
- Maintenant, nous allons créer le trigger proprement dit. Pour cela, nous indiquons à la base de données la table concernée, les opérations concernées, et la procédure stockée contenant le code du trigger :

```
CREATE TRIGGER trg_arch_art BEFORE DELETE OR UPDATE ON article
  FOR EACH ROW EXECUTE PROCEDURE arch_art();
```

```
CREATE TRIGGER trigger [ BEFORE | AFTER ]
    [ INSERT | DELETE | UPDATE [ OR ... ] ]
    ON relation FOR EACH [ ROW | STATEMENT ]
    EXECUTE PROCEDURE procedure (args);
```


Exemple d'utilisation

```
insert into article(id,texte) values(1, 'Ceci est le premier article original');
```

```
Select * from article;
id | date_modif |                texte                | status
---+------------+-------------------------------------+--------
 1 | 2004-11-19 | Ceci est le premier article original |
(1 row)
```

```
Select * from archive;
id | date_modif | texte
---+------------+-------
(0 rows)
```

```
Select * from article;
Id | date_modif |                 texte                 | status
---+------------+---------------------------------------+--------
 1 | 2004-11-19 | Ceci est le premier article version 2 |
(1 row)
```

```
select * from archive;
id | date_modif |                texte
---+------------+--------------------------------------
 1 | 2004-11-19 | Ceci est le premier article original
(1 row)
```