Procédures trigger

Définition :

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 ;
- 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);

2. créer un trigger (CREATE TRIGGER) qui exécute cette procédure avant ou après exécution d'une requête SQL.

   CREATE TRIGGER nom_de_trigger { BEFORE | AFTER } { event [OR ...] }

   ON nom_table [FOR EACH { ROW | STATEMENT }]

   EXECUTE PROCEDURE fonc (arguments)

   où event = UPDATE, DELETE, INSERT

   FOR EACH ROW : trigger exécuté pour chaque ligne sur laquelle agit l'opération. Par exemple si 15 insertions alors le trigger sera appelé 15 fois, une fois par insertion.

   FOR EACH STATEMENT : trigger appelé une seule fois, indépendamment du nombre de lignes affectées par l'opération (même si 0 lignes modifiées).

Variables predefinies dans les triggers

Nom	Type	Description
NEW	RECORD	Nouvelle valeurs (pour UPDATE et INSERT)
OLD	RECORD	Anciennes valeurs (pour UPDATE et DELETE)
TG_NAME	name	Nom de trigger
TG_WHEN	text	BEFORE ou AFTER
TG_LEVEL	text	ROW ou STATEMENT
TG_OP	text	INSERT, UPDATE, ou DELETE
TG_RELNAME	name	Nom de la table associée au trigger
TG_NARGS	integer	Le nombre d'arguments optionnels de la commande CREATE TRIGGER
TG_ARGV[]	text[]	Les arguments optionnels de la commande CREATE TRIGGER

Remarque :

Pour supprimer un trigger :

DROP TRIGGER nom_de_trigger ON nom_table;

Exemple 1 :

Cet exemple de trigger assure qu'à chaque moment où une ligne est insérée ou mise à jour dans la table **emp**, le nom de l'utilisateur courant et l'heure sont insérés dans la ligne. Et cela assure qu'un nom d'employé est donné et que le salaire est une valeur positive supérieure à la moyenne des salaires.

```
CREATE TABLE emp (
    nom_employe             VARCHAR(15),
    salaire                 INTEGER,
    date_dermodif           TIMESTAMP,
    user_dermodif           VARCHAR(15)
);
```

-- a) créer la fonction associée au trigger sur la table emp

```
CREATE FUNCTION emp_stamp() RETURNS trigger AS $$

      DECLARE
      moy_salaire FLOAT;

      BEGIN

        -- Verifie que nom_employe et salaires sont donnés
        IF NEW.nom_employe IS NULL THEN
           RAISE EXCEPTION 'nom_employe ne peut pas être NULL';
        END IF;

        IF NEW.salaire IS NULL THEN
           RAISE EXCEPTION '% ne peut pas avoir un salaire NULL', NEW.nom_employe;
        END IF;

        -- Verifie que le nouveau salaire est >=0 et >=moyenne
        IF NEW.salaire < 0 THEN
           RAISE EXCEPTION '% ne peut pas avoir un salaire négatif', NEW.nom_employe;
        END IF;

        SELECT AVG(salaire) INTO moy_salaire FROM emp ;
        IF NEW.salaire < moy_salaire THEN
           RAISE EXCEPTION '% ne peut pas avoir un salaire < moyenne', NEW.nom_employe;
        END IF;

        -- Mémorise qui a changé le salaire et quand
        NEW.user_dermodif := user;
        NEW.date_dermodif := now();

        RETURN NEW;

      END;
$$ LANGUAGE plpgsql;
```

-- b) déclarer le trigger sur la table emp

```
CREATE TRIGGER t1 BEFORE INSERT OR UPDATE ON emp
    FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

Exemple 2 :

Cet exemple de trigger nous assure que toute insertion, modification ou suppression d'une ligne dans la table **emp** est enregistrée dans la table **emp_audit**. L'heure et le nom de l'utilisateur sont conservées dans la ligne avec le type d'opération réalisé.

```
CREATE TABLE emp (
    nom_employe             VARCHAR(15),
    salaire                 INTEGER
);

CREATE TABLE emp_audit(
    operation               CHAR(1)          NOT NULL,
    date                    TIMESTAMP,
    user                    VARCHAR(15)
    nom_employe             VARCHAR(15)      NOT NULL,
    salaire                 INTEGER
);
```

-- a) créer la fonction associée au trigger sur la table emp

```
CREATE OR REPLACE FUNCTION audit_employe() RETURNS TRIGGER AS $$
BEGIN

    -- Ajoute une ligne dans emp_audit pour refléter l'opération réalisée sur emp,
    -- utilise la variable spéciale TG_OP pour cette opération.

    IF (TG_OP = 'DELETE') THEN
        INSERT INTO emp_audit SELECT 'D', now(), user, OLD.*;
        RETURN OLD;

    ELSIF (TG_OP = 'UPDATE') THEN
        INSERT INTO emp_audit SELECT 'U', now(), user, NEW.*;
        RETURN NEW;

    ELSIF (TG_OP = 'INSERT') THEN
        INSERT INTO emp_audit SELECT 'I', now(), user, NEW.*;
        RETURN NEW;

    END IF;

END;
$$ language plpgsql;
```

-- b) déclarer le trigger sur la table emp

```
CREATE TRIGGER t2
   AFTER INSERT OR UPDATE OR DELETE ON emp
   FOR EACH ROW EXECUTE PROCEDURE audit_employe();
```


Exemple 3 :

Soient 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,
        statut          varchar
);

CREATE TABLE archive(
        id              integer,
        date_modif      date    not null,
        texte           text


);
```

-- a) créer la fonction associée au trigger sur la table article

-- cette fonction sera appelée lors d'une modification d'un article afin d'archiver la version précédente de l'article. De plus, la modification ne pourra se faire si l'article est déjà publié (statut ='publié').

```
CREATE OR REPLACE FUNCTION arch_art() RETURNS TRIGGER AS $$
BEGIN
        IF NEW.texte != OLD.texte THEN
                IF OLD.statut = 'publié'THEN
                    RAISE EXCEPTION 'Article % déjà publié',NEW.id;
                END IF;
                INSERT INTO archive(id, date_modif, texte) VALUES (OLD.id,OLD.date_modif,OLD.texte);
      END IF;

      RETURN NEW;

END;
$$ LANGUAGE 'plpgsql';
```

-- b) déclarer le trigger sur la table article

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

Exemple 4 :

I. On veut créer un trigger sur la table **Pieces**. Si le nombre de pièces en stock est insuffisant, alors il faut passer la commande auprès d'un fournisseur pour compléter le stock.

Les 4 tables de la base de données sont les suivantes :

```
---------------------- exemple.sql --------------------------------
DROP TABLE Commandes;
DROP TABLE Prix;
DROP TABLE Pieces;
DROP TABLE Fournisseurs;;

CREATE TABLE Pieces(
  piece_id INT PRIMARY KEY,
  description  VARCHAR(70) NOT NULL,
  en_stock INT NOT NULL,
  stock_max INT NOT NULL,
  stock_min INT NOT NULL,
  CHECK(stock_min < stock_max)
);

CREATE TABLE Fournisseurs(
   fournisseur_id INT PRIMARY KEY,
   nom VARCHAR(40) NOT NULL,
   adresse  VARCHAR(30)
);

CREATE TABLE Prix(
   fournisseur_id INT REFERENCES Fournisseurs(fournisseur_id),
   piece_id  INT REFERENCES Pieces(piece_id),
   prix DECIMAL(10,2)  NOT NULL CHECK(prix > 0),
   PRIMARY KEY(fournisseur_id,piece_id)
);

CREATE TABLE Commandes(
   fournisseur_id INT REFERENCES Fournisseurs,
   piece_id  INT NOT NULL REFERENCES Pieces,
   quantite  INT NOT NULL CHECK(quantite > 0),
   date_commande DATE NOT NULL,
   date_reception DATE DEFAULT NULL,
   PRIMARY KEY(fournisseur_id,piece_id,date_commande),
   CHECK( date_reception IS NULL OR date_reception >= date_commande)
```



```
-- a) créer la fonction associée au trigger sur la table Pieces

CREATE OR REPLACE FUNCTION verif_pieces() RETURNS TRIGGER AS $$

 DECLARE

   toto INT;
   quantite_deja_commandee    INT:=0;
   p     Prix.prix%TYPE;
   f     Fournisseurs.fournisseur_id%TYPE;

 BEGIN

   -- Si la quantité en stock augmente, on ne fait rien
   IF NEW.en_stock > OLD.en_stock THEN RETURN NEW;
   END IF;

   -- Si la quantité en stock donnée est négative, on la remet à 0
   IF NEW.en_stock < 0       THEN NEW.en_stock = 0;
   END IF;

   -- Combien de pièces de ce type est déjà commandé?
   FOR i IN SELECT quantite FROM Commandes WHERE piece_id = OLD.piece_id
   LOOP
      quantite_deja_commandee := quantite_deja_commande + i.quantite;
   END LOOP;

   -- Si (quantité commandée+quantité) en stock dépasse stock mini alors pas de commandes à passer
   IF quantite_deja_commandee + NEW.en_stock > OLD.stock_min THEN RETURN NEW;
   END IF;

   -- Stock insuffisant passer la commande auprès des fournisseurs

   -- Quel est le prix min pour cette pièce?
    SELECT min(prix) INTO p  FROM Prix WHERE OLD.piece_id = Prix.piece_id;

   -- Pas de fournisseur rien à faire
    IF NOT FOUND THEN RETURN NULL;
    END IF;

   -- Sélectionner le 1er fournisseur proposant un prix minimal
    SELECT fournisseur_id INTO f
    FROM Prix WHERE prix = p LIMIT 1;

   --Commander chez lui
    INSERT INTO Commandes VALUES
    (f,OLD.piece_id,OLD.stock_max - quantité_deja_commande - NEW.en_stock, current_date);
    RETURN NEW;
 END;
$$ LANGUAGE 'plpgsql';

-- b) Déclarer le trigger sur la table Pieces.

CREATE TRIGGER trg_verif_piece BEFORE UPDATE OR INSERT
ON Pieces FOR EACH ROW
EXECUTE PROCEDURE verif_pieces();
```

II. Si une commande est réalisée, on ajoute les pièces correspondant dans le stock (table **Pieces**).

-- a) la fonction associée au trigger sur la table Commande

```
CREATE OR  REPLACE FUNCTION deliver() RETURNS TRIGGER AS $$

 DECLARE

    toto  INT;
    i     INT;
    p     Prix.prix%TYPE;
    f     Fournisseurs.fournisseur_id%TYPE;

 BEGIN
    -- vérifier si effectivement il s'agit de la livraison d'une commande
    IF TG_OP = 'UPDATE' AND
         (NEW.date_reception IS NULL OR OLD.date_reception IS NOT NULL) THEN
        RETURN NEW;
    END IF;

    IF TG_OP = 'INSERT' AND NEW.date_reception IS   NULL THEN
        RETURN NEW;
    END IF;

    -- commande est délivrée, mettre a jour le stock
    UPDATE pieces SET en_stock = en_stock + NEW.quantite
    WHERE piece_id = NEW.piece_id;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'il y a un problème';
        RETURN NULL;
    END IF;

    RETURN NEW;
  END;
$$ LANGUAGE 'plpgsql';
```

-- b) Déclarer le trigger sur la table Commandes.

```
CREATE TRIGGER verif_commandes BEFORE UPDATE OR INSERT
ON Commande FOR EACH ROW
EXECUTE PROCEDURE deliver();
```