

Exploitation d'une base de données

Département Informatique

IUT2 de Grenoble

BUT1 - Ressource 2.06

Introduction à l'exploration et l'administration des BD

1. Révisions : BD relationnelle et SQL simple
2. SQL pour explorer et analyser
3. SQL augmenté : *triggers*
4. Administration de 1^{er} niveau

3. SQL augmenté : *triggers*

3.1. Triggers et fonctions simples

- Introduction
- Fonction en PL/pgSQL
- Trigger sur table
- Pptés : déclenchement, var. prédéf., validation/annulation

3.2. Triggers sur vues

3.3. Fonctions avancées en PL/pgSQL

- Déclaration de variables
- Instructions

3. SQL augmenté : *triggers*

3.1. Triggers et fonctions simples

- Introduction
- Fonction en PL/pgSQL
- Trigger sur table
- Pptés : déclenchement, var. prédéf., validation/annulation

3.2. Triggers sur vues

3.3. Fonctions avancées en PL/pgSQL

- Déclaration de variables
- Instructions

Trigger

Définition d'un *trigger*

exécute **automatiquement** une **fonction** lorsque certains **événements** se produisent.

Dans ce cours

- fonction : PL/pgSQL
- événement : **INSERT**, **UPDATE**, **DELETE** sur table ou vue

Quel objectif ?

- tenir un historique des actions sur la base
- annuler ou propager des modifications
- vérifier des contraintes complexes
- exécuter des actions sur une vue complexe

PL/pgSQL

PostgreSQL Procedural Language

- langage procédural pour définir des traitements ...
- exécutés sur le serveur de BD ...
- mélangeant requêtes SQL et structures de contrôles ou instructions classiques

PL/pgSQL

Syntaxe spécifique pour une fonction déclenchée par un trigger

```
CREATE FUNCTION nom_fonc() RETURNS trigger  
AS $$  
    corps de la fonction  
$$ LANGUAGE 'plpgsql';
```

Corps d'une fonction simple

```
BEGIN  
    suite de requêtes SQL de modifications de données  
    RETURN rec;  
END;
```

Trigger défini sur une table

Syntaxe

```
CREATE TRIGGER nom_trigger  
{ BEFORE | AFTER }  
{ INSERT | UPDATE | DELETE [ OR ... ] }  
ON nom_table  
FOR EACH { ROW | STATEMENT }  
EXECUTE FUNCTION nom_fonc();
```

Identifié par *nom_trigger* + *nom_table*

```
DROP TRIGGER nom_trigger ON nom_table;
```

 Attention aux boucles infinies !

Ne pas exécuter dans la fonction, la même action sur même table

Cas d'utilisation : historique (audit) des actions

```
CREATE FUNCTION nom_fonc() RETURNS trigger
AS $$
BEGIN
    INSERT TG_OP + ppte d'état de la base dans la table d'audit.
    RETURN NULL;
END; $$ LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER nom_trigger
AFTER INSERT OR UPDATE OR DELETE
ON nom_table
FOR EACH STATEMENT
EXECUTE FUNCTION nom_fonc();
```

Cas d'utilisation : mise à jour d'un attribut calculé

```
CREATE FUNCTION nom_fonc() RETURNS trigger
AS $$
BEGIN
    UPDATE de l'attribut calculé, dans une autre table.
    RETURN new;
END; $$ LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER nom_trigger
BEFORE INSERT
ON nom_table
FOR EACH ROW
EXECUTE FUNCTION nom_fonc();
```

Cas d'utilisation : vérifier une contrainte complexe

```
CREATE FUNCTION nom_fonc() RETURNS trigger
AS $$
BEGIN
    Annuler la modification si elle contrevient à une contrainte.
    RETURN new;
END; $$ LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER nom_trigger
BEFORE INSERT
ON nom_table
FOR EACH ROW
EXECUTE FUNCTION nom_fonc();
```

Déclenchement

FOR EACH ROW

- fonction déclenchée autant de fois que de lignes concernées
- **BEFORE** : avant chaque opération
- **AFTER** : après l'ensemble des opérations

FOR EACH STATEMENT

- fonction déclenchée une seule fois
- **BEFORE** : avant l'ensemble des opérations
- **AFTER** : après l'ensemble des opérations

Variables prédéfinies

n-uplets **new** et **old**

- si **FOR EACH ROW**
- **old** : version du n-uplet avant la requête : `NULL` si `INSERT`
- **new** : version du n-uplet après la requête : `NULL` si `DELETE`

variables caractérisant le trigger

- **TG_WHEN** : `BEFORE`, `AFTER`
- **TG_LEVEL** : `ROW`, `STATEMENT`
- **TG_OP** : `INSERT`, `UPDATE`, `DELETE`
- **TG_TABLE_NAME** : nom de la table associée
- **TG_NAME** : nom du trigger

Utilisables par exemple pour afficher un message de *log* :

```
RAISE NOTICE '% ON % FIRES %', TG_OP, TG_TABLE_NAME, TG_NAME;
```

Validation/Annulation de la requête déclencheuse

RETURN

- **FOR EACH ROW + BEFORE : RETURN rec;**
avec **rec** =
 - un n-uplet non null si **DELETE** (**RETURN old;**),
 - le n-uplet à insérer si **INSERT** (**RETURN new;**)
 - le nouvel n-uplet si **UPDATE** (**RETURN new;**)
- **FOR EACH ROW + AFTER : RETURN NULL;**
- **FOR EACH STATEMENT : RETURN NULL;**

⚠ Attention

FOR EACH ROW + BEFORE : RETURN NULL;

⇒ pour la ligne associée à cette exécution de la fonction :

- la requête déclencheuse ne s'applique pas ;
- aucun autre trigger ne sera plus déclenché.

Commandes `psql` utiles

Pour retrouver les fonctions trigger

- liste des fonctions : `\dft`
- code d'une fonction : `\sf nom_fonc`

Pour retrouver les triggers

- liste des triggers :

```
SELECT trigger_name,event_object_table
FROM information_schema.triggers;
```
- définition des triggers sur une table : `\d nom_table`