# Utilisation des bases de données

## CM6 : PL/pgSQL et triggers

Mickaël Martin-Nevot

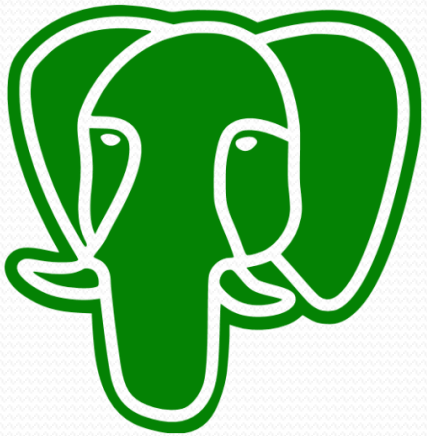
V1.0.0

# Utilisation des bases de données


I. Prés.
II. BD et SGBD
III. Merise
IV. LDD
V. LMD
VI. LCT
VII. Droits
VIII. LDSP
IX. SQL avancé
X. PL/pgSQL

# PL/pgSQL

- *Procedural language/PostgreSQL structured query language* : langage procédural de PostgreSQL
- Pourquoi PL/pgSQL (limites de SQL) ?
  - Combinaison de requêtes avec mécanismes procéduraux (variables, structures de contrôle, etc.)
  - Ajout de fonctions utilisateur
  - *Triggers*
  - Procédures stockées (hors cadre du cours)



SQL n'est pas procédural

# Trigger

- **Déclencheur** (procédure déclenchée, « gâchette »)
- Définition de **contraintes d'intégrité sémantique**
- Spécifie les actions à réaliser lors (avant ou après) **d'écritures** (`INSERT`, `UPDATE`, `DELETE`, etc.)
- Associé à **une seule relation**
- **Exécution automatique**
- *Trigger* de ligne ou de table
- Utilise une fonction (PL/pgSQL) *trigger*

Si écriture est effective
(même avec aucun tuple concerné)

Il faut définir la fonction trigger avant le trigger

Lorsqu'on supprime une relation, tous les triggers associés sont aussi supprimés

# Trigger

- `CREATE FUNCTION` : Création d'une fonction *trigger*

Syntaxe :

```
CREATE FUNCTION function_name() RETURNS trigger AS $$
BEGIN
    [IF condition
        [THEN RAISE EXCEPTION '...';]
    END IF;]
    [...]
    RETURN NEW;
END; $$ LANGUAGE plpgsql;
```

**Type de retour** : `$$` permet de délimiter une définition de fonction (sans cela, chaque `;` est considéré comme une fin de requête SQL)

- Variables spéciales (transitoires) : ← Uniquement *trigger* de ligne
  - `NEW` : tuple créé/modifié (uniquement `INSERT`, `UPDATE`)
  - `OLD` : tuple affecté (uniquement `UPDATE`, `DELETE`)
  - Etc.

# Trigger

- `CREATE TRIGGER` : création d'un *trigger*

**Syntaxe :**

```
CREATE [...] TRIGGER name {BEFORE|AFTER|...} {<EVENT> [OR ...]}
ON Tbl [...] [FOR [EACH] {ROW|STATEMENT}] [WHEN (condition)]
EXECUTE PROCEDURE function_name (arguments)
```

`<EVENT>` : `INSERT`, `UPDATE`, `DELETE` ou `TRUNCATE`

  - `BEFORE`, `AFTER` : fonction appelée avant ou après l'événement
  - `FOR EACH ROW`, `FOR EACH STATEMENT` : déclenché pour chaque tuple (*trigger* de ligne) ou une seule fois par demande d'écriture SQL (*trigger* de table)
  - `WHEN` : condition de déclenchement

- `DROP TRIGGER` : suppression d'un trigger

**Syntaxe :**

```
DROP TRIGGER [IF EXISTS] name ON Tbl [...]
```

# Exemple de trigger

```
CREATE OR REPLACE FUNCTION garde_fou() RETURNS Trigger AS $$
BEGIN
    IF (SELECT COUNT(*) FROM Convention WHERE Convention.idp = NEW.idp) > 2
        THEN RAISE EXCEPTION 'Employe trop occupe';
    END IF;
    RETURN NEW;
END; $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS Garde_Fou ON Convention;

CREATE TRIGGER Garde_Fou
AFTER INSERT ON Convention
FOR EACH ROW
EXECUTE PROCEDURE garde_fou();

-- Provoque une erreur.
INSERT INTO Convention VALUES (8, 13, 12, '2020-4-12', '2020-7-01', 5, '...', 11);
```

# Aller plus loin

- Procédures stockées
- Curseurs
- Gestion des erreurs

# Crédits

## Auteur

Mickaël Martin-Nevot
mmartin.nevot@gmail.com

- Laurent Carmignac

Carte de visite électronique

## Relecteurs

Cours en ligne sur : www.mickael-martin-nevot.com