

### TP3 — Déclencheurs et contraintes d'intégrité

Les déclencheurs (triggers) permettent l'exécution automatique d'actions lors d'événements sur les données (INSERT, UPDATE, DELETE).

Ils offrent de nombreux avantages, parmi lesquels :

1. **Mise en œuvre automatique de contraintes** de tout type, indépendamment des applications qui manipulent les données.
2. **Réduction de la maintenance applicative** : une modification dans un déclencheur est automatiquement appliquée à toutes les applications utilisant la table, sans recompilation ni changements côté client.
3. **Journalisation automatique** des opérations réalisées sur les tables (historisation, audit, suivi).
4. **Notification automatique** des modifications grâce aux mécanismes d'alerte ou de signaux supportés par le SGBD.

Ces mécanismes permettent de centraliser les règles métier directement dans la base, garantissant une meilleure intégrité et cohérence des données.

On considère le schéma relationnel suivant :

**EMPLOYE**(ID, EMPLOYE, QUALIFICATION, SALAIRE)

**PROJET**(ID, PROJET, ID\_RESPONSABLE, DATE\_DEBUT, DATE\_FIN)

**PARTICIPE**(ID\_EMPLOYE, ID\_PROJET, ROLE)

En utilisant la base créée par le script en <https://bdda.al-moualime.com/sql/tp3-projets-postgresql-fr.sql>, dont voici la partie relative à la définition des tables :

-- EMPLOYE

CREATE TABLE IF NOT EXISTS employe (

id	did	NOT NULL PRIMARY KEY,	
employe	dnom	NOT NULL UNIQUE,	-- MAJUSCULE SANS BLANCS
qualification	dqualification	NOT NULL,	-- MAJUSCULE SANS BLANCS
salaire	dsalaire	NOT NULL	-- > 0.00 AND VALUE <= 100000.00

);

-- PROJET

CREATE TABLE IF NOT EXISTS projet (

id	did	NOT NULL PRIMARY KEY,
projet	dnom	NOT NULL UNIQUE,
id_responsable	did	REFERENCES employe(id),
date_debut	ddate	NOT NULL,
date_fin	DATE,	

CONSTRAINT chk\_projet\_dates

CHECK (date\_fin IS NULL OR date\_fin >= date\_debut)

);

-- PARTICIPE

CREATE TABLE IF NOT EXISTS participe (

id_employe	did	NOT NULL,	
id_projet	did	NOT NULL,	
role	drole	NOT NULL DEFAULT 'MEMBRE',	-- RESPONSABLE / MEMBRE

PRIMARY KEY (id\_employe, id\_projet),

FOREIGN KEY (id\_employe) REFERENCES employe(id),

FOREIGN KEY (id\_projet) REFERENCES projet(id)

);

## Travail demandé

Chaque règle ci-dessous doit être implémentée au moyen d'un trigger approprié (BEFORE/AFTER INSERT, UPDATE ou DELETE).

### 1. Contrôle des rôles et du nombre de projets par employé

L'objectif est de faire respecter automatiquement des règles métier lors de l'insertion ou la modification des participations.

Vous devez garantir que :

- Le rôle d'un employé dans un projet doit être **obligatoirement** MEMBRE ou RESPONSABLE. Toute autre valeur doit être refusée.
- Un employé ne peut être **responsable que dans 2 projets maximum**. Toute tentative d'ajout ou de modification dépassant cette limite doit être refusée.
- Un employé ne peut participer comme **membre à plus de 3 projets**. Toute tentative d'ajout ou de modification violant cette règle doit être refusée.

### 2. Synchronisation responsable ↔ participation

Dès qu'un employé est désigné comme responsable d'un projet, il doit automatiquement apparaître comme participant à ce projet.

Vous devez donc mettre en place un mécanisme qui :

- détecte toute nomination ou changement de responsable dans PROJET,
- vérifie s'il apparaît déjà dans PARTICIPE,
- l'ajoute automatiquement avec le rôle RESPONSABLE dans le cas contraire.

### 3. Interdiction de supprimer la participation du responsable

La participation du responsable ne doit jamais pouvoir être supprimée.

Vous devez :

- bloquer toute tentative de suppression de la ligne PARTICIPE correspondant au responsable du projet,
- déclencher une erreur explicite (RAISE EXCEPTION) lorsque cela se produit.

### 4. Rôle imposé pour le responsable

Le rôle du responsable doit être impossible à modifier.

Vous devez empêcher que :

- un responsable devienne membre,
- son rôle soit modifié vers une autre valeur.

Le trigger doit, au choix :

- **bloquer l'opération,**
- ou **corriger automatiquement** le rôle pour le remettre à RESPONSABLE.