Tp Trigger

TP TRIGGER échauffement:

1-Objectifs du TP

L'objectif de ce premier exercice était de comprendre le fonctionnement de base des **déclencheurs (triggers)** dans un environnement PostgreSQL. Un trigger est un script qui se déclenche automatiquement lors d'événements spécifiques (INSERT, UPDATE, DELETE) afin d'assurer l'intégrité parfaite des données.

2-Étape préalable : Activer le langage

Avant de commencer, vous devez indiquer à PostgreSQL que vous allez utiliser le langage **plpgsql**.

```
CREATE LANGUAGE PLPGSQL;
```

3-Mise en pratique : Le trigger "Employee"

Étape A : Création de la table

Commencez par créer la table qui va recevoir les données.

```
CREATE TABLE Employee (
    nom_employe text,
    salaire integer,
    utilisateur_dermodif text
);
```

Étape B : Définition de la fonction trigger

Le trigger a besoin d'une fonction à exécuter. Ici, nous créons une fonction nommée fct() qui vérifie que le nom de l'employé n'est pas vide.

- Variable NEW : Cette variable spéciale contient la nouvelle ligne que l'on tente d'insérer ou de modifier.
- Logique : Si NEW.nom_employe est NULL, le système bloque l'opération et affiche une erreur.

```
CREATE OR REPLACE FUNCTION fct() RETURNS trigger AS $fct$
BEGIN
    IF NEW.nom_employe IS NULL THEN
        RAISE EXCEPTION 'nom_employe ne peut pas être NULL';
    END IF;
    RETURN NEW;
END;
$fct$ LANGUAGE plpgsql;
```

Étape C : Création du trigger

Il faut maintenant lier la fonction à la table Employee pour qu'elle s'active lors d'un ajout ou d'une modification.

```
CREATE TRIGGER trig
BEFORE INSERT OR UPDATE
ON Employee
FOR EACH ROW EXECUTE PROCEDURE fct();
```

```
tpTrigger=> \dt
          List of relations
 Schema |   Name   | Type  |  Owner
--------+----------+-------+---------
 public | Employe  | table | rayanew
 public | employee | table | rayanew
(2 rows)

tpTrigger=> CREATE OR REPLACE FUNCTION fct() RETURNS trigger AS $fct$
BEGIN
    IF NEW.nom_employe IS NULL THEN
        RAISE EXCEPTION 'nom_employe ne peut pas être NULL';
    END IF;
    RETURN NEW;
END;
$fct$ LANGUAGE plpgsql;
CREATE FUNCTION
tpTrigger=> CREATE TRIGGER trig
BEFORE INSERT OR UPDATE
ON Employee
FOR EACH ROW EXECUTE PROCEDURE fct();
CREATE TRIGGER
tpTrigger=>
```

4. Test et Observation

```
tpTrigger=> INSERT INTO Employee (nom_employe, salaire, utilisateur_dermodif)
VALUES ('Toto', 154, 'Tata');
INSERT 0 1
tpTrigger=> INSERT INTO Employee (nom_employe, salaire, utilisateur_dermodif)
VALUES (NULL, 154, 'Titi');
ERROR:  nom_employe ne peut pas être NULL
CONTEXT:  PL/pgSQL function fct() line 4 at RAISE
tpTrigger=>
```

```
INSERT INTO Employee VALUES ('Toto', 154, 'Tata');
```

Résultat attendu : La ligne est ajoutée normalement.

```
INSERT INTO Employee VALUES (NULL, 154, 'Titi');
```

Résultat attendu : PostgreSQL doit bloquer l'insertion et renvoyer l'erreur : `"nom_employe ne peut pas être NULL"`

```
tpTrigger=> INSERT INTO Employee VALUES ('Stagiaire', 0, 'System');
INSERT 0 1
tpTrigger=>
```

```
INSERT INTO Employee VALUES ('Stagiaire', 0, 'System');
```

Résultat attendu : La ligne est ajoutée normalement car le trigger vérifie que le nom.

TP Triggers (Parc Immobilier):

1. Objectifs du TP

L'objectif de cet exercice était de mettre en place une règle de gestion métier complexe entre deux colonnes d'une même table. Il s'agissait de garantir que le prix d'une place de parking ne soit renseigné que si l'appartement possède effectivement une place.

2-Mise en pratique : Le trigger "Immeuble":

Étape A : Création de la table

Commencez par créer la table qui va recevoir les données.

Étape B : Définition de la fonction trigger

Le trigger a besoin d'une fonction à exécuter. Ici, nous créons une fonction nommée fn_check_parking() qui vérifie que le prix du parking est NULL lorsque l'appartement ne possède pas de place de parking.

- Variable NEW : Cette variable spéciale contient la nouvelle ligne que l'on tente d'insérer ou de modifier.
- Logique : Si NEW.placeParking est NULL, le système bloque l'opération et affiche une erreur.

```
CREATE OR REPLACE FUNCTION fn_check_parking()
RETURNS trigger AS $$
BEGIN
    IF NEW.placeParking = 0 AND NEW.prixParking IS NOT NULL THEN
        RAISE EXCEPTION 'Le prix du parking doit être NULL si l
appartement n a pas de place de parking.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Étape C : Création du trigger

Il faut maintenant lier la fonction à la table Appartement pour qu'elle s'active lors d'un ajout ou d'une modification.

```
CREATE TRIGGER trg_parking_null
BEFORE INSERT OR UPDATE ON Appartement
FOR EACH ROW
EXECUTE FUNCTION fn_check_parking();
```

```
tpTrigger=> CREATE OR REPLACE FUNCTION fn_check_parking()
RETURNS trigger AS $$
BEGIN
    IF NEW.placeParking = 0 AND NEW.prixParking IS NOT NULL THEN
        RAISE EXCEPTION 'Le prix du parking doit être NULL si l appartement n a pas de place de parking.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
tpTrigger=>
CREATE TRIGGER trg_parking_null
BEFORE INSERT OR UPDATE ON Appartement
FOR EACH ROW
EXECUTE FUNCTION fn_check_parking();
CREATE TRIGGER
tpTrigger=>
```

3-Compte 3-rendu des tests

Les tests réalisés permettent de valider le bon fonctionnement du trigger `fn_check_parking`.

Le premier test, correspondant à l'insertion d'un appartement sans place de parking avec un prix de parking NULL, est accepté conformément à la règle définie.

Le second test, où un appartement sans place de parking est inséré avec un prix de parking renseigné, est correctement bloqué par le trigger, qui génère une erreur.

Ces résultats confirment que le trigger garantit la cohérence des données relatives au parking.

```
-- Création d'un immeuble de test
INSERT INTO Immeuble (adrNum, adrVoie, adrCodePostal, adrVille,
fibreOptique, parkingPrivatif)
VALUES ('10', 'Rue de la Paix', '75000', 'Paris', 1, 1);

-- TEST 1 : Insertion valide (Pas de parking, prix NULL) -> OK
INSERT INTO Appartement (immeuble, num, loyer, superficie, terrasse,
classeConso, chauffage, placeParking, prixParking)
VALUES (1, 101, 750, 35, 0, 'B', 'E', 0, NULL);
```

```sql
-- TEST 2 : Insertion invalide (Pas de parking, mais un prix de 60) -> DOIT BLOQUER
INSERT INTO Appartement (immeuble, num, loyer, superficie, terrasse, classeConso, chauffage, placeParking, prixParking)
VALUES (1, 102, 900, 40, 1, 'A', 'G', 0, 60.0);
```

```
tpTrigger=> -- Création d'un immeuble de test
INSERT INTO Immeuble (adrNum, adrVoie, adrCodePostal, adrVille, fibreOptique, parkingPrivatif)
VALUES ('10', 'Rue de la Paix', '75000', 'Paris', 1, 1);

-- TEST 1 : Insertion valide (Pas de parking, prix NULL) -> OK
INSERT INTO Appartement (immeuble, num, loyer, superficie, terrasse, classeConso, chauffage, placeParking,
VALUES (1, 101, 750, 35, 0, 'B', 'E', 0, NULL);

-- TEST 2 : Insertion invalide (Pas de parking, mais un prix de 60) -> DOIT BLOQUER
INSERT INTO Appartement (immeuble, num, loyer, superficie, terrasse, classeConso, chauffage, placeParking,
VALUES (1, 102, 900, 40, 1, 'A', 'G', 0, 60.0);
INSERT 0 1
INSERT 0 1
ERROR:  Le prix du parking doit être NULL si l appartement n a pas de place de parking.
CONTEXT:  PL/pgSQL function fn_check_parking() line 4 at RAISE
tpTrigger=>
```

I. Exercice : Pour aller plus loin (Produits et Commandes)

Question 1 : Script de création des tables

Ce script crée la structure nécessaire en respectant les clés primaires et étrangères définies.

```
-- Création des tables [cite: 229]
CREATE TABLE Mode_Livraison (
    num SERIAL PRIMARY KEY,
    libelle VARCHAR(100) NOT NULL
);

CREATE TABLE Frais_Transport (
    livraison INT,
    num INT NOT NULL,
    prix DOUBLE PRECISION,
    poids_min INT,
    poids_max INT,
    CONSTRAINT pk_frais_transport PRIMARY KEY (livraison, num),
    CONSTRAINT fk_livraison FOREIGN KEY (livraison) REFERENCES
Mode_Livraison(num)
);

CREATE TABLE Produit (
    code SERIAL PRIMARY KEY,
    libelle VARCHAR(100) NOT NULL,
    prix DOUBLE PRECISION NOT NULL,
    formation SMALLINT NOT NULL, -- 1 = formation, 0 = livre [cite: 216]
    poids INT -- NULL pour une formation [cite: 204]
);

CREATE TABLE Commande (
    num SERIAL PRIMARY KEY,
    destinataire VARCHAR(100),
    prixTotal DOUBLE PRECISION DEFAULT 0,
    livraison INT,
    transport INT,
    CONSTRAINT fk_f_transport FOREIGN KEY (livraison, transport)
REFERENCES Frais_Transport(livraison, num)
);

CREATE TABLE Ligne_Commande (
```

```
    commande INT,
    produit INT,
    prixu DOUBLE PRECISION,
    quantite INT,
    prixt DOUBLE PRECISION,
    CONSTRAINT pk_ligne_commande PRIMARY KEY (commande, produit),
    CONSTRAINT fk_lc_commande FOREIGN KEY (commande) REFERENCES
Commande(num),
    CONSTRAINT fk_lc_produit FOREIGN KEY (produit) REFERENCES
Produit(code)
);
```

Questions 2 à 4 : Triggers de contrôle (Exclusivité et Transport)

Ce trigger unique vérifie l'exclusivité entre formations et livres, ainsi que l'interdiction des frais de port pour les formations .

```
CREATE OR REPLACE FUNCTION fn_verif_ligne_commande()
RETURNS trigger AS $$
DECLARE
    v_est_formation INT;
    v_deja_formation INT;
    v_deja_livre INT;
    v_a_transport INT;
BEGIN
    -- 1. On détermine si le produit inséré est une formation
    SELECT formation INTO v_est_formation FROM Produit WHERE code =
NEW.produit;

    -- 2. On vérifie ce que contient déjà la commande
    SELECT COUNT(*) INTO v_deja_formation FROM Ligne_Commande lc
    JOIN Produit p ON lc.produit = p.code
    WHERE lc.commande = NEW.commande AND p.formation = 1;

    SELECT COUNT(*) INTO v_deja_livre FROM Ligne_Commande lc
    JOIN Produit p ON lc.produit = p.code
    WHERE lc.commande = NEW.commande AND p.formation = 0;

    -- 3. On vérifie s'il y a des frais de transport sur la commande
    SELECT transport INTO v_a_transport FROM Commande WHERE num =
NEW.commande;

    -- Règle Q2 : Si nouveau = formation, interdit si déjà livres [cite:
```

```
230-231]
    IF v_est_formation = 1 AND v_deja_livre > 0 THEN
        RAISE EXCEPTION 'Impossible : une formation ne peut être ajoutée
à une commande de livres.';
    END IF;

    -- Règle Q3 : Si nouveau = livre, interdit si déjà formation [cite:
232]
    IF v_est_formation = 0 AND v_deja_formation > 0 THEN
        RAISE EXCEPTION 'Impossible : un livre ne peut être ajouté à une
commande de formation.';
    END IF;

    -- Règle Q4 : Si formation, pas de frais de transport [cite: 233]
    IF v_est_formation = 1 AND v_a_transport IS NOT NULL THEN
        RAISE EXCEPTION 'Une formation ne peut pas avoir de frais de
transport.';
    END IF;

    -- Calcul automatique du prix total de la ligne (utile pour Q5/Q6)
    SELECT prix INTO NEW.prixu FROM Produit WHERE code = NEW.produit;
    NEW.prixt := NEW.prixu * NEW.quantite;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_verif_ligne
BEFORE INSERT ON Ligne_Commande
FOR EACH ROW EXECUTE FUNCTION fn_verif_ligne_commande();
```

Questions 5 et 6 : Mise à jour automatique du Prix Total

Ce trigger met à jour le montant global de la commande dès qu'une ligne est modifiée .

```
CREATE OR REPLACE FUNCTION fn_maj_prix_total_commande()
RETURNS trigger AS $$
BEGIN
    -- Met à jour le prix total de la commande parente
    UPDATE Commande
    SET prixTotal = (SELECT SUM(prixt) FROM Ligne_Commande WHERE
commande = NEW.commande)
    WHERE num = NEW.commande;

    RETURN NEW;
```

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

```
CREATE TRIGGER trg_maj_prix_total
AFTER INSERT OR UPDATE ON Ligne_Commande
FOR EACH ROW EXECUTE FUNCTION fn_maj_prix_total_commande();
```

Test :

```
-- Immeuble 1 : Équipé de tout (Fibre et Parking)
INSERT INTO Immeuble (adrNum, adrVoie, adrCodePostal, adrVille,
fibreOptique, parkingPrivatif)
VALUES ('12', 'Rue de la Paix', '75002', 'Paris', 1, 1);

-- Immeuble 2 : Uniquement avec parking privatif
INSERT INTO Immeuble (adrNum, adrVoie, adrCodePostal, adrVille,
fibreOptique, parkingPrivatif)
VALUES ('45B', 'Avenue Jean Jaurès', '69007', 'Lyon', 0, 1);

-- Immeuble 3 : Sans équipements particuliers
INSERT INTO Immeuble (adrNum, adrVoie, adrCodePostal, adrVille,
fibreOptique, parkingPrivatif)
VALUES ('7', 'Boulevard du Prado', '13008', 'Marseille', 0, 0);
```

```
tpTrigger=> CREATE TRIGGER trg_maj_prix_total
AFTER INSERT OR UPDATE ON Ligne_Commande
FOR EACH ROW EXECUTE FUNCTION fn_maj_prix_total_commande();
CREATE TRIGGER
tpTrigger=> -- Immeuble 1 : Équipé de tout (Fibre et Parking)
INSERT INTO Immeuble (adrNum, adrVoie, adrCodePostal, adrVille, fibreOptique, parkingPrivatif)
VALUES ('12', 'Rue de la Paix', '75002', 'Paris', 1, 1);

-- Immeuble 2 : Uniquement avec parking privatif
INSERT INTO Immeuble (adrNum, adrVoie, adrCodePostal, adrVille, fibreOptique, parkingPrivatif)
VALUES ('45B', 'Avenue Jean Jaurès', '69007', 'Lyon', 0, 1);

-- Immeuble 3 : Sans équipements particuliers
INSERT INTO Immeuble (adrNum, adrVoie, adrCodePostal, adrVille, fibreOptique, parkingPrivatif)
VALUES ('7', 'Boulevard du Prado', '13008', 'Marseille', 0, 0);
INSERT 0 1
INSERT 0 1
INSERT 0 1
tpTrigger=>
```

```
--Ajout d'un immeuble sans parking avec un prix de place de parking
INSERT INTO Appartement (
    immeuble, num, description, loyer, superficie,
    terrasse, classeConso, chauffage, placeParking, prixParking
)
VALUES (
    1, 201, 'Studio sans parking mais avec prix', 600.0, 30.0,
```

```
    0, 'B', 'E', 0, 45.0
);
```

```
INSERT INTO Appartement (
    immeuble, num, description, loyer, superficie,
    terrasse, classeConso, chauffage, placeParking, prixParking
)
VALUES (
    1, 201, 'Studio sans parking mais avec prix', 600.0, 30.0,
    0, 'B', 'E', 0, 45.0
);
ERROR:  Le prix du parking doit être NULL si l appartement n a pas de place de parking.
CONTEXT:  PL/pgSQL function fn_check_parking() line 4 at RAISE
tpTrigger=>
```

II. Exercice : Location de véhicules (MySQL)

Question 1 : Script de création (MySQL)

Ce script crée la base de données selon le schéma de location.

```
CREATE TABLE Permis (
    num INT PRIMARY KEY AUTO_INCREMENT,
    type VARCHAR(5) NOT NULL -- A, B, C... [cite: 245]
);

CREATE TABLE Conducteur (
    num INT PRIMARY KEY AUTO_INCREMENT,
    civilite VARCHAR(15),
    prenom VARCHAR(50),
    nom VARCHAR(50) [cite: 251-253]
);

CREATE TABLE titulaire (
    conducteur INT,
    permis INT,
    PRIMARY KEY (conducteur, permis),
    FOREIGN KEY (conducteur) REFERENCES Conducteur(num),
    FOREIGN KEY (permis) REFERENCES Permis(num) [cite: 246]
);

CREATE TABLE Vehicule (
    num INT PRIMARY KEY AUTO_INCREMENT,
    immatriculation VARCHAR(15) NOT NULL,
    permis_requis INT,
    FOREIGN KEY (permis_requis) REFERENCES Permis(num) [cite: 258-259]
);
```

```
CREATE TABLE Location (
    num INT PRIMARY KEY AUTO_INCREMENT,
    conducteur INT,
    vehicule INT,
    date_debut DATETIME,
    date_fin DATETIME,
    FOREIGN KEY (conducteur) REFERENCES Conducteur(num),
    FOREIGN KEY (vehicule) REFERENCES Vehicule(num) [cite: 263-268]
);
```

Question 2 : Trigger de vérification du permis

Ce trigger vérifie que le conducteur possède bien le type de permis requis pour le véhicule sélectionné lors d'une location.

```
CREATE OR REPLACE FUNCTION fn_verif_permis_location()
RETURNS trigger AS $$
DECLARE
    v_permis_requis INT;
    v_possede_permis INT;
BEGIN
    -- 1. On récupère l'ID du permis requis pour ce véhicule [cite: 258, 261]
    SELECT permis_requis INTO v_permis_requis
    FROM Vehicule WHERE num = NEW.vehicule;

    -- 2. On vérifie si le conducteur possède ce permis [cite: 246, 248]
    SELECT COUNT(*) INTO v_possede_permis
    FROM titulaire
    WHERE conducteur = NEW.conducteur AND permis = v_permis_requis;

    -- 3. Si le conducteur ne l'a pas, on bloque l'insertion
    IF v_possede_permis = 0 THEN
        RAISE EXCEPTION 'Erreur : Le conducteur n est pas titulaire du permis requis pour ce véhicule.';
    END IF;

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

Test:

```sql
-- Création d'un permis A (ID 2) et d'une moto (ID 2)
INSERT INTO Permis (type_permis) VALUES ('A'); -- [cite: 245]
INSERT INTO Vehicule (immatriculation, permis_requis) VALUES
('MOTO-999', 2); -- [cite: 261]

-- Tentative de location de la moto par Jean (Il n'a que le permis B)
INSERT INTO Location (conducteur, vehicule, date_debut)
VALUES (1, 2, NOW());
```

```
tpTrigger=> INSERT INTO Location (conducteur, vehicule, date_debut, date_fin)
VALUES (1, 1, NOW(), NOW() + interval '1 day');
INSERT 0 1
tpTrigger=>
```

```
tpTrigger=> INSERT INTO Location (conducteur, vehicule, date_debut, date_fin)
VALUES (1, 2, NOW(), NOW() + interval '1 day');
ERROR:  Erreur : Le conducteur n est pas titulaire du permis requis pour ce véhicule.
CONTEXT:  PL/pgSQL function fn_verif_permis_location() line 17 at RAISE
tpTrigger=>
```