LIFBDW2 : PL/pgSQL – Exceptions et triggers

Ce TP est dans la continuité du précédent. Nous allons capturer des exceptions dans les fonctions et/ou travailler sur des fonctions particulières qui sont appelées après certaines opérations : les triggers.

1. **Consulter la documentation en ligne sur PL/pgSQL :**
    a. https://docs.postgresql.fr/13/plpgsql.html
    b. https://w3resource.com/PostgreSQL/postgresql-triggers.php
2. **Exceptions : Ré-écrire la fonction de division pour gérer le problème de division par zéro en levant une exception.**
3. **Créer les tables Company, table1 et table2 via le script suivant :**

```
DROP TABLE IF EXISTS company;
DROP TABLE IF EXISTS table1;
DROP TABLE IF EXISTS table2;

CREATE TABLE company (
 id SERIAL PRIMARY KEY NOT NULL,
 name TEXT NOT NULL,
 created_at TIMESTAMP,
 modified_at TIMESTAMP DEFAULT NOW()
)

CREATE TABLE table1 (
 id SERIAL PRIMARY KEY NOT NULL,
 name TEXT NOT NULL,
 created_at TIMESTAMP,
 modified_at TIMESTAMP DEFAULT NOW()
)

CREATE TABLE table2 (
 id SERIAL PRIMARY KEY NOT NULL,
 name TEXT NOT NULL,
 created_at TIMESTAMP,
 modified_at TIMESTAMP DEFAULT NOW()
)
```

4. **Créer un trigger qui remplit le champ created_at sur la table Company à l'insertion.**
5. **Similairement, créer un trigger qui met à jour l'attribut modified_at lors d'une mise à jour. Tester avec les instructions suivantes :**

```
INSERT INTO company (name) VALUES ('Company 2');
INSERT INTO company (name) VALUES ('Company 3');
UPDATE company SET name='Company new 3' WHERE name='Company 3';
select * from company;
```

6. **Ajouter ces triggers à table1 et table2.**
7. **On va maintenant créer une nouvelle relation :**

```
DROP TABLE IF EXISTS logs;
CREATE TABLE log (
 id SERIAL PRIMARY KEY NOT NULL,
 table_name TEXT NOT NULL,
 table_id TEXT NOT NULL,
 description TEXT NOT NULL,
 created_at TIMESTAMP DEFAULT NOW()
)
```

8. **Créer un trigger qui ajoute un tuple dans la table log après chaque modification (INSERT, DELETE, UPDATE) de la table company en instanciant les champs avec les valeurs pertinentes.**

Ex :

```
INSERT INTO company (name) VALUES ('Company 5');
INSERT INTO company (name) VALUES ('Company 6');
UPDATE company SET name='Company new 5' WHERE name='Company 5';
select * from log ;
```

58	company	19	company added. Id: 19	2020-11-11 15:56:15
59	company	20	company added. Id: 20	2020-11-11 15:56:18
60	company	9	company updated. Id: 9	2020-11-11 15:56:48
61	company	17	company updated. Id: 17	2020-11-11 15:56:48
62	company	19	company updated. Id: 19	2020-11-11 15:56:48

9. **Définir des triggers s'appuyant sur la même fonction sur les relations table1 et table2.**
10. **Modifier la fonction afin qu'elle puisse être utilisée pour un trigger similaire sur la table employes (TP précédent).**
11. **Ajouter un attribut salaire sur la table employé.**
12. **Mettre à jour les salaires de la table employé. On supposera que chaque employé a un salaire de départ qui augmente par année d'ancienneté (salaire= salaire_départ + années_dans_entreprise * coefficient). Le salaire de départ et le coefficient étant différent en fonction de la fonction occupée :**

```
-- directeur : 3000 ; 499
-- Consultant ou développeurs  : 2000 , 199
-- Responsable 2500, 150
-- comptable 2400, 100
-- commercial 2000, 100
```

13. **Créer un trigger sur Employé afin d'empêcher toute baisse de salaire.**
14. **Créer un trigger qui a chaque insertion ou modification vérifie qu'une personne ne peut pas avoir un salaire plus important qu'un de ses collègues (même département) occupant la même fonction.**