1 Fonctions

1.1 Définir une fonction

```
REPLACE OR CREATE FUNCTION schema.name(arguments) RETURNS type AS
$$
DECLARE
-- variables que vous utiliserez, par exemple:
BEGIN
-- code de la fonction
END;
$$
LANGUAGE plpgsql;
```

1.2 Astuces utiles

Tester si une requête renvoie un résultat vide (PERFORM remplace SELECT) :

```
PERFORM requête;
IF FOUND THEN
  -- ce qu'il faut faire si la requête n'est pas vide
ELSE
  -- ce qu'il faut faire si la requête est vide
END IF;
```

Mettre le résultat d'une requête dans une variable :

```
SELECT col1 INTO variable FROM ... WHERE ...
```

Par exemple :

```
SELECT COUNT(pid) INTO nb_franc FROM atp.player_big WHERE code='FRA';
```

Pour renvoyer une valeur particulière, on utilisera le mot-clé `RETURN`.

2 Pull the trigger

Les triggers (gâchettes en français) permettent d'associer une action à certains événements comme l'insertion dans une table, la mise à jour d'une table ou la suppression de données. On commence par définir une fonction qui retourne un type `trigger` ainsi :

```
REPLACE OR CREATE FUNCTION name(arguments) RETURNS trigger AS ...
```

Ensuite, on définira un trigger qui exécutera notre fonction à chaque occurence d'un événement. Cela permet d'assurer la cohérence de la base de données. Par exemple :

- on peut vouloir s'assurer qu'à chaque insertion dans la table `played_in`, le joueur en question n'est pas déjà inscrit dans un tournoi qui a lieu en même temps. On aura donc un trigger qui se déclenchera lorsqu'on détecte une insertion dans `played_in` (voir le cours de Stéphane).
- on peut vouloir nettoyer la base de données APT lors de la suppression d'un tournoi en supprimant aussi tous les matchs qui ont eu lieu dans ce tournoi ainsi que les informations concernant ces matchs. On aura alors un trigger qui se déclenchera lorsqu'on détecte une suppression dans la table `tournament`.

2.1 Définir un trigger

On suppose qu'on a déjà défini une fonction `f_trig` qui retourne un trigger. On veut désormais la lier à un événement particulier. Pour cela on utilise :

```
CREATE TRIGGER schema.name {BEFORE, AFTER} {INSERT, DELETE, UPDATE}
ON table
{FOR EACH ROW} EXECUTE f_trig;
```

Par exemple, si on veut exécuter la fonction `clean_matches` après un `DELETE` sur la table `tournament`, on écrira :

```
CREATE TRIGGER atp.tournament_clean AFTER DELETE
ON atp.tournament
FOR EACH ROW EXECUTE clean_matches();
```

Le `FOR EACH ROW` permet d'exécuter le trigger autant de fois qu'il y a de ligne concernée par l'événement. Par exemple, si vous voulez supprimez tous les tournois du French Open, vous allez écrire

```
DELETE FROM atp.tournament WHERE name='French Open';
```

Cette requête va supprimer plusieurs tournois. Si vous ne spécifiez pas FOR EACH ROW, le trigger ne sera exécuté qu'une seule fois. Sinon il est exécuté pour chaque tournoi supprimé.

2.2 Écrire une fonction pour les triggers

Quand un trigger se déclenche, cela concerne toujours une ligne en particulier. Par exemple, une ligne est supprimée ou ajoutée, ou modifiée. Vous pouvez accéder aux informations de cette ligne avec le mot clé NEW. Par exemple, lors de la suppression d'un tournoi, si vous voulez récupérer l'identifiant du tournoi qui est en train d'être supprimé, vous écrirez `NEW.tid`. Pour supprimer tous les matchs liés à ce tournoi, on écrira :

```
DELETE FROM atp.game_big WHERE tid=NEW.tid;
```

Votre fonction doit enfin retourner une ligne ou NULL. Par exemple, si vous déclenchez votre trigger avant une insertion et que vous voulez empêcher cette insertion car elle ne remplit pas une certaine condition, vous renverrez `NULL`. Sinon, vous renverrez `NEW`.

```
IF condition THEN
  RETURN NULL;
ELSE
  RETURN NEW;
ENDIF;
```

Vous voudrez peut-être aussi modifier une des colonnes de la ligne qui doit être insérez. Par exemple, vous voulez réécrire les noms des joueurs en majuscule lors d'une insertion. Vous pourrez alors modifier `NEW`.

```
NEW.first_name = first_name_majuscule;
RETURN NEW;
```

3 Un petit exercice

Pour s'entraîner, on va créer des triggers pour une table très simple : `Membres(id INT PRIMARY KEY, pseudo CHAR(30))` qui liste les pseudos des membres d'un forum.

1. Commencez par créer cette table dans votre schéma.
2. Créez une fonction `entid.insert_member_fun` (où entid est votre identifiant) de type trigger qui vérifie que le pseudo n'est pas utilisé. Si c'est le cas, votre procédure doit renvoyer NULL. Sinon, elle doit renvoyer NEW (la requête qui est à l'origine du déclenchement du trigger).
3. Créez un trigger `entid.insert_member` qui déclenche `insert_member_fun` avant chaque insertion dans la table `Membres`. Est-ce une exécution simple ou un exécution `FOR EACH ROW` ?
4. On souhaite garantir que la colonne `id` est bien une clé pour `Membres`. Modifiez la fonction `insert_member_fun` pour que à chaque insertion dans `Membres`, si la valeur de l'`id` proposée est déjà utilisée ou NULL, elle soit remplacée par la plus grande valeur d'`id` dans la table plus 1.

   Par exemple, si la table est `(1, 'bob') ; (3, 'jean')` et que l'utilisateur veut insérer la ligne `(NULL, 'alice')`, alors la ligne `(4, 'alice')` sera insérée.
5. Modifiez encore la fonction `insert_member_fun` pour que à chaque insertion dans `Membres`, si la valeur de l'`id` proposée est déjà utilisée ou NULL, elle soit remplacée par la plus petite valeur positive d'`id` disponible.

   Par exemple, si la table est `(1, 'bob') ; (3, 'jean')` et que l'utilisateur veut insérer la ligne `(NULL, 'alice')`, alors la ligne `(2, 'alice')` sera insérée.
6. Implémentez toutes ces contraintes pour les requêtes `UPDATE`.