TP03 - Fonctions

Minh-Hoang DANG

December 1, 2018

```
In [1]: -- connection: host='localhost' dbname='TP03' user='postgres' password='postgres'
```

1 Ecrire une fonction qui prend deux chaînes et retourne la longueur de la plus longue.

```
In [2]: CREATE OR REPLACE FUNCTION longer_string(text, text)
        RETURNS text AS $$
            BEGIN
                IF char_length($1) > char_length($2) THEN
                    RETURN $1;
                ELSE
                    RETURN $2;
                END IF;
            END;
        $$ LANGUAGE plpgsql;

        SELECT * FROM longer_string('court', 'long');
```

```
1 row(s) returned.
```

longer_string
court

2 Ecrire une fonction qui prend un texte et affiche (notice) tous les mots un par un.

```
In [3]: CREATE OR REPLACE FUNCTION afficher_mots(text)
        RETURNS void AS $$
            DECLARE
                iter INTEGER := 1;
                mot TEXT := '';
            BEGIN
                LOOP
                    mot := split_part($1, ' ', iter);
                    RAISE NOTICE 'Mot %: %', iter, mot;
                    iter := iter + 1;
                    EXIT WHEN mot = '';
                END LOOP;
            END;
        $$ LANGUAGE plpgsql;

        SELECT * FROM afficher_mots('Dies irae, dies illa! Solvet saeclum in favilla, teste
        David cum Sybilla!')
```

```
1 row(s) returned.
NOTICE:  Mot 1: Dies
NOTICE:  Mot 2: irae,
NOTICE:  Mot 3: dies
NOTICE:  Mot 4: illa!
NOTICE:  Mot 5: Solvet
NOTICE:  Mot 6: saeclum
NOTICE:  Mot 7: in
NOTICE:  Mot 8: favilla,
NOTICE:  Mot 9: teste
NOTICE:  Mot 10: David
NOTICE:  Mot 11: cum
NOTICE:  Mot 12: Sybilla!
NOTICE:  Mot 13:
```

afficher_mots

3 Ecrire une version itérative de la fonction factorielle.

```
In [4]: CREATE OR REPLACE FUNCTION factoriel(integer)
        RETURNS integer AS $$
            DECLARE
                res integer := 1;
                iter integer := 1;
            BEGIN
                LOOP
                    res := res * iter;
                    iter := iter + 1;
                    EXIT WHEN iter > $1;
                END LOOP;
                RETURN res;
            END;

        $$ LANGUAGE plpgsql;

        select * from factoriel(3);
```

```
1 row(s) returned.
```

factoriel
6

4 Calculer les deux valeurs suivantes :

4.1 Somme des durées de tous les films en écrivant une requête SQL puis une fonction parcourant tous les enregistrements un par un (for in). Assurez-vous que les deux méthodes retournent le même résultat.

- Requête SQL:

```
In [5]: SELECT SUM(duree) FROM films;
```

```
1 row(s) returned.
```

sum
10468

- Fonction SQL:

```
In [6]: CREATE OR REPLACE FUNCTION sum_duree_films()
        RETURNS integer AS $$
            DECLARE
                total_length integer := 0;
                unit_record RECORD;
            BEGIN
                FOR unit_record IN SELECT duree FROM films LOOP
                    total_length := total_length + unit_record.duree;
                END LOOP;
                RETURN total_length;
            END;
        $$ LANGUAGE plpgsql;

        SELECT * FROM sum_duree_films();

1 row(s) returned.
```

sum_duree_films
10468

4.2 Prix moyen d'achat des films en écrivant une requête SQL puis une fonction parcourant les enregistrements avec un curseur. La fonction devra lever une exception s'il n'y a aucun élément dans la table. Assurez-vous que les deux méthodes retournent le même résultat et testez l'exception en vidant la table (TRUNCATE TABLE).

- Requête SQL:

```
In [7]: SELECT AVG(prixachat) FROM dvds

1 row(s) returned.
```

avg
13.9561

- Function SQL:

```
In [8]: CREATE OR REPLACE FUNCTION avg_prix_achat()
        RETURNS float AS $$
            DECLARE
                total integer;
                unit_record RECORD;
                total_prix float := 0;
            BEGIN
                SELECT COUNT(nodvd) FROM dvds INTO total;
                IF total = 0 THEN
                    RAISE EXCEPTION 'Table dvds est vide!'
                    USING HINT = 'Charger les données!';
                END IF;
```

```
            FOR unit_record IN SELECT prixachat FROM dvds LOOP
                total_prix := total_prix + unit_record.prixachat;
            END LOOP;
            RETURN total_prix/total;
        END;
    $$ LANGUAGE plpgsql;

    SELECT * FROM avg_prix_achat();
```

```
1 row(s) returned.
```

avg_prix_achat
13.9561

5 Les numéros de dvds sont complètement chamboulés. Ecrire une fonction pour le renuméroter de 1 à n de manière consécutive. Faites ce qu'il faut pour que ne pas avoir de problèmes avec les clés étrangères.

```
In [9]: CREATE OR REPLACE FUNCTION corriger_nodvd()
        RETURNS void AS $$
            DECLARE
                unit_record RECORD;
                iter integer := 1;
                current_nodvd integer;
            BEGIN
                ALTER TABLE locations DROP CONSTRAINT IF EXISTS locations_nodvd_fkey;
                ALTER TABLE dvds DROP CONSTRAINT IF EXISTS dvds_pkey;

                FOR unit_record IN SELECT nodvd FROM dvds LOOP
                    current_nodvd := unit_record.nodvd;

                    IF iter IN (SELECT nodvd FROM dvds) THEN
                        iter := iter + 1;
                    END IF;

                    UPDATE dvds SET nodvd = iter WHERE nodvd = current_nodvd;
                    UPDATE locations SET nodvd = iter WHERE nodvd = current_nodvd;

                    iter := iter + 1;

                END LOOP;

                ALTER TABLE dvds ADD CONSTRAINT dvds_pkey PRIMARY KEY (nodvd);

                ALTER TABLE locations ADD CONSTRAINT locations_nodvd_fkey FOREIGN KEY (nodvd)
                REFERENCES public.dvds (nodvd) MATCH SIMPLE
                ON UPDATE CASCADE
                ON DELETE NO ACTION;
            END;
        $$ LANGUAGE plpgsql;

        select * from corriger_nodvd();
```

```
duplicate key value violates unique constraint "locations_pkey"
DETAIL:  Key (nodvd, datelocation)=(6, 2015-03-21) already exists.
CONTEXT:  SQL statement "UPDATE locations SET nodvd = iter WHERE nodvd =
current_nodvd"
PL/pgSQL function corriger_nodvd() line 18 at SQL statement
```

```
In [10]: select * from dvds LIMIT 10;

10 row(s) returned.
```

nodvd	prixachat	titre
1	13	Red Corner
3	13	There Goes the Neighborhood
4	20	Le dernier des Mohicans
5	9	The Man from Earth
6	15	Etre ou ne pas etre
7	17	L amour en equation
8	20	Flowers in the Attic
9	15	Chinatown
10	18	Memento
11	12	Warrior

6 Ecrire les triggers suivants :

6.1 Si on tente d'insérer ou de modifier un film en mettant une année inférieure à 1891 (pourquoi ?) une exception est levée.

En Mars 1891, William K. L. Dickson a développé le premier kinétoscope en se basant sur les travaux de Louis Le Prince. Ici, on ne considère que les films tourné avec un kinétoscope.

```
In [11]: CREATE OR REPLACE FUNCTION verifier_anneesortie_film()
         RETURNS TRIGGER AS $$
             BEGIN
                 RAISE EXCEPTION 'Année de la sortie de "%" ne peut être avant 1891!, (input %)',
         new.titre, new.anneesortie;
             END;
         $$ LANGUAGE plpgsql;

         DROP TRIGGER IF EXISTS trigger_films_anneesortie on films;
         CREATE TRIGGER trigger_films_anneesortie
             BEFORE INSERT OR UPDATE OF anneesortie ON films
             FOR EACH ROW
             WHEN (NEW.anneesortie < 1891)
                 EXECUTE PROCEDURE verifier_anneesortie_film();

In [12]: DELETE FROM films WHERE titre = 'This Race Horse';
         INSERT INTO films VALUES('This Race Horse', 0.25, 'Cursus Limited', 'E.Muybridge', 1878,
         'Drama');

Année de la sortie de "This Race Horse" ne peut être avant 1891!, (input 1878)
CONTEXT:  PL/pgSQL function verifier_anneesortie_film() line 3 at RAISE


In [13]: UPDATE films SET anneesortie = 1890 WHERE titre = 'Red Corner';
         SELECT * FROM films WHERE titre = 'Red Corner';

Année de la sortie de "Red Corner" ne peut être avant 1891!, (input 1890)
CONTEXT:  PL/pgSQL function verifier_anneesortie_film() line 3 at RAISE
```

6.2 Si un dvd est inséré dans la table sans prix d'achat indiqué (par exemple « INSERT INTO dvds (NoDVD, Titre) VALUES (1001, '8 mm'); ») alors le prix d'achat est fixé au prix d'achat moyen des dvds correspondant au même film. Par contre si le prix est indiqué il ne faut rien faire.

```
In [14]: CREATE OR REPLACE FUNCTION verifier_prixachat_dvds()
         RETURNS TRIGGER AS $$
             DECLARE
                 prixAvg float;
             BEGIN
                 SELECT AVG(prixachat) FROM dvds INTO NEW.prixachat ;
                 RETURN NEW;
             END;
         $$ LANGUAGE plpgsql;

         DROP TRIGGER IF EXISTS trigger_dvds_prixachat on dvds;
         CREATE TRIGGER trigger_dvds_prixachat
             BEFORE INSERT ON dvds
             FOR EACH ROW
             WHEN ( NEW.prixachat IS NULL )
                 EXECUTE PROCEDURE verifier_prixachat_dvds();

In [15]: DELETE FROM dvds WHERE nodvd = 1003;
         DELETE FROM dvds WHERE nodvd = 1002;

         INSERT INTO dvds (nodvd, titre) VALUES(1003, '8 mm');
         INSERT INTO dvds VALUES(1002, 7, '8 mm');

         SELECT * FROM dvds WHERE nodvd = 1003 or nodvd = 1002;

2 row(s) returned.
```

nodvd	prixachat	titre
1003	14	8 mm
1002	7	8 mm

7 Trouver au moins deux triggers pertinents supplémentaires sur cette base et implémentez-les.

7.1 Il n'y a pas de clé étranger pour les clients pour la table locations

```
In [16]: CREATE OR REPLACE FUNCTION verifier_clients_locations()
         RETURNS TRIGGER AS $$
             BEGIN
                 IF (NEW.noclient IN (SELECT noclient FROM clients)) THEN
                     RETURN NEW;
                 ELSE
                     RAISE EXCEPTION 'Vérifiez que le client % existe avant!', NEW.noclient;
                     --RETURN NULL;
                 END IF;
             END;
         $$ LANGUAGE plpgsql;

         DROP TRIGGER IF EXISTS trigger_locations_clients on locations;
         CREATE TRIGGER trigger_locations_clients
             BEFORE INSERT ON locations
             FOR EACH ROW
             EXECUTE PROCEDURE verifier_clients_locations();

In [17]: INSERT INTO locations VALUES (1003, current_date, 666, 3);
```

```
Vérifiez que le client 666 existe avant!
CONTEXT:  PL/pgSQL function verifier_clients_locations() line 6 at RAISE
```

7.2 Lors de l'insertion dans la table location, si la date n'est pas précisé, on met la date du jour

```
In [18]: CREATE OR REPLACE FUNCTION verifier_date_locations()
         RETURNS TRIGGER AS $$
             BEGIN
                 SELECT current_date INTO NEW.datelocation;
                 RETURN NEW;
             END;
         $$ LANGUAGE plpgsql;

         DROP TRIGGER IF EXISTS trigger_locations_date on locations;
         CREATE TRIGGER trigger_locations_date
             BEFORE INSERT ON locations
             FOR EACH ROW
             WHEN(NEW.datelocation IS NULL)
                 EXECUTE PROCEDURE verifier_date_locations();

In [19]: INSERT INTO locations(nodvd, noclient, dureelocation) VALUES (1003, 2, 3);
         SELECT * FROM locations WHERE noclient = 2 and nodvd = 1003;
```

```
1 row(s) returned.
```

nodvd	datelocation	noclient	dureelocation
1003	2018-12-01	2	3