

SGBD : EXPLOITATION D'UNE BASE DE DONNÉES [R206]

TD(TP) N^o5 – 6 - PROGRAMMATION PLPG/SQL

OBJECTIFS

- Programmation en PL/pgSQL
- Mise en oeuvre des curseurs
- introduction aux triggers

ENONCÉS

Exercice I : curseurs, move, fetch

On considère le schéma de base de données EMPLOYE suivant et son instance :

```
1 create table EMPLOYE
2 (
3     Empid      int,
4     Empnom     varchar(30),
5     Empgrade   int,
6     Empsalaire money,
7     primary key (empid)
8 );
```

```
=> SELECT * FROM employe;
empid | empnom  | empgrade | empsalaire
-----+-----+-----+-----
      1 | Paul    |          | €1 000,00
      2 | Linda   |          | €1 100,00
      3 | Natasha |          | €1 400,00
      4 | Julia   |          | €1 500,00
      5 | Emanuel |          | €2 000,00
      6 | Vector  |          | €2 100,00
      7 | Peter   |          | €21 500,00
      8 | Sandra  |          | €1 150,00
      9 | Pedro   |          | €1 600,00
     10 | Norman  |          | €2 200,00
     11 | Alvaro  |          | €2 500,00
     12 | Jemma   |          | €2 600,00
     13 | Bushy   |          | €2 300,00
     14 | Amir   |          | €1 150,00
```

(14 rows)

Question 1.1. *En utilisant le mécanisme de curseur (nommé), Créer une fonction `somme_salaire` dont le résultat est équivalent à la requête SQL suivante (en utilisant uniquement `FETCH`) :*

Date: March 11, 2022.

Hocine ABIR - IUT Villetaneuse .

```

1 CREATE FUNCTION somme_salaire
2   (out sm money) AS
3   $$
4     -- corps fonction
5   $$ LANGUAGE plpgsql;

```

```

=> SELECT sum(empsalaire) as ms
-> FROM employe;
      ms
-----
€44 100,00
(1 row)

```

Question 1.2. *A partir de la fonction somme_salaire, Créer une fonction sgrade_salaire dont le résultat est équivalent à la requête SQL suivante (en utilisant uniquement FETCH, et MOVE) :*

```

1 CREATE FUNCTION sgrade_salaire
2   (out grade int, out msg money)
3   RETURNS SETOF RECORD AS
4   $$
5     -- corps fonction
6   $$ LANGUAGE plpgsql;

```

```

=> SELECT empgrade as grade,
->       sum(empsalaire) as msg
-> FROM employe group by 1;
grade |      msg
-----+-----
1 | €4 400,00
2 | €4 500,00
3 | €4 100,00
4 | €26 000,00
5 | €5 100,00
(5 rows)

```

Question 1.3. *A partir de la fonction sgrade_salaire, Créer une fonction part_salaire dont le résultat est équivalent à la requête SQL suivante (en utilisant uniquement FETCH, et MOVE) :*

```

1 CREATE FUNCTION partition_salaire
2   (out emp employe, out msg money)
3   RETURNS SETOF RECORD AS
4   $$
5     -- corps fonction
6   $$ LANGUAGE plpgsql;

```

```

=> SELECT *, sum(empsalaire)
      OVER (PARTITION BY empgrade) as msg
FROM employe;

```

empid	empnom	empgrade	empsalaire	msg
14	Amir	1	€1 150,00	€4 400,00
2	Linda	1	€1 100,00	€4 400,00
8	Sandra	1	€1 150,00	€4 400,00
1	Paul	1	€1 000,00	€4 400,00

3		Natasha		2		€1 400,00		€4 500,00
9		Pedro		2		€1 600,00		€4 500,00
4		Julia		2		€1 500,00		€4 500,00
5		Emanuel		3		€2 000,00		€4 100,00
6		Vector		3		€2 100,00		€4 100,00
10		Norman		4		€2 200,00		€26 000,00
13		Bushy		4		€2 300,00		€26 000,00
7		Peter		4		€21 500,00		€26 000,00
11		Alvaro		5		€2 500,00		€5 100,00
12		Jemma		5		€2 600,00		€5 100,00

(14 rows)

Exercice II : Logging Audit Changes

On considère le schéma de relation `controle` suivant :

```

1 CREATE TABLE controle (
2     co_numero    integer PRIMARY KEY,
3     co_nom       text,
4     co_coef      decimal(4,2)
5 );

```

Pour cette entité, on souhaite enregistrer des informations concernant les différentes opérations de mise à jour (INSERT, DELETE, UPDATE) qui sont effectuées sur cette entité. Pour cela on propose une entité (Non-Normalisée) `controle_audit` décrite comme suit :

```

1 CREATE TABLE controle_audit (
2     action        text, -- INSERT/UPDATE/DELETE
3     action_timestamp timestamp -- estampille
4         default current_timestamp,
5     old_controle  controle, -- tuple supprime ou avant maj
6     new_controle  controle -- tuple insere ou apres maj
7 );

```

Question 2.1. Décrire un trigger `controle_audit_trig` pour assurer cette tâche comme dans l'exemple ci-dessous :

```

=> insert into controle values (1,'Controle Long',3);
=> insert into controle values (2,'Controle Court',1);
=> insert into controle values (3,'Controle Moyen',2);
=> update controle set co_coef=1.5 where co_numero=3;
=> delete from controle where co_numero=3;

SELECT * FROM controle_audit;
-[ RECORD 1 ]-----+-----
action          | INSERT

```

```

action_timestamp | 17/02/2014 19:14:51.814028
old_controle    | NULL
new_controle     | (1,"Controle Long",3.00)
-[ RECORD 2 ]----+-----
action          | INSERT
action_timestamp | 17/02/2014 19:15:03.129788
old_controle     | NULL
new_controle     | (2,"Controle Court",1.00)
-[ RECORD 3 ]----+-----
action          | INSERT
action_timestamp | 17/02/2014 19:15:10.785765
old_controle     | NULL
new_controle     | (3,"Controle Moyen",2.00)
-[ RECORD 4 ]----+-----
action          | UPDATE
action_timestamp | 17/02/2014 19:15:19.001938
old_controle     | (3,"Controle Moyen",2.00)
new_controle     | (3,"Controle Moyen",1.50)
-[ RECORD 5 ]----+-----
action          | DELETE
action_timestamp | 17/02/2014 19:15:27.093787
old_controle     | (3,"Controle Moyen",1.50)
new_controle     | NULL
=> select * from controle;
   co_numero |   co_nom   | co_coef
-----+-----+-----
           1 | Controle Long |    3.00
           2 | Controle Court |    1.00
(2 rows)

```

Exercice III : contraintes

Soit le schéma simplifié suivant :

```

1 CREATE TABLE enseignant (
2     e_num         integer PRIMARY KEY,
3     -- numero enseignant
4     e_nom         text,
5     e_bureau      text
6 );
7
8 CREATE TABLE cours (
9     c_num         integer PRIMARY KEY,
10    c_ens         integer,
11    -- numero enseignant qui assure ce cours

```

```

12     c_mat    text,
13     c_coef   decimal(4,2)
14 );

```

Et l'instance suivante :

```
=> select * from enseignant;
e_num | e_nom  | e_bureau
```

```

-----+-----+-----
      10 | Martin | T104
      20 | Maxime | T103
      30 | Paule  | T102

```

(3 rows)

```
=> select * from cours;
c_num | c_ens | c_mat  | c_coef
-----+-----+-----+-----
      1 |    10 | Java   |    3.50
      2 |    20 | Adm Syst |    1.50
      3 |    10 | Prog C  |    5.50
      4 |    10 | SGBD    |    2.50

```

(4 rows)

On considère le script suivant :

```

1 BEGIN;
2     INSERT INTO cours values(5,40,'UML',1.0);
3     DELETE FROM enseignant where e_num=20;
4     -- Rcheck
5     SELECT NOT EXISTS
6     (
7         SELECT c_ens FROM cours
8         WHERE c_ens IS NOT NULL
9         AND NOT EXISTS
10        (SELECT e_num
11         FROM enseignant
12         WHERE e_num=c_ens
13        )
14    ) as valide;
15 ROLLBACK;

```

Question 3.1. Exécuter ce script et donner le résultat affiché par la requête Rcheck.

Question 3.2. La requête `Rcheck` permet de vérifier une relation sémantique (ou contrainte) entre les tables `COURS` et `ENSEIGNANT` et produit deux résultats possibles (`true` ou `false`) selon que cette contrainte est satisfaite ou pas.

3.2.1. Quelle est la contrainte SQL que la requête `Rcheck` désigne.

3.2.2. Expliquer "le pourquoi" de la clause `c_ens IS NOT NULL` de la requête `Rcheck`.

Pour mettre en oeuvre la contrainte `Rcheck`, on propose la solution suivante :

```

1 CREATE FUNCTION Rcheck ()
2     RETURNS TRIGGER AS
3 $$
4     DECLARE
5         ens int;
6     BEGIN
7         SELECT c_ens into ens FROM cours
8             WHERE c_ens IS NOT NULL
9             AND NOT EXISTS
10                (SELECT e_num
11                 FROM enseignant
12                 WHERE e_num=c_ens
13                );
14         IF FOUND THEN
15             RAISE EXCEPTION 'Echec Rcheck !';
16         END IF;
17         RETURN NULL;
18     END;
19 $$ LANGUAGE plpgsql;
20
21
22 CREATE TRIGGER Rcheck_cours
23     AFTER INSERT OR UPDATE ON cours
24     FOR EACH statement
25     EXECUTE PROCEDURE Rcheck();
26
27 CREATE TRIGGER Rcheck_enseignant
28     AFTER DELETE OR UPDATE ON enseignant
29     FOR EACH statement
30     EXECUTE PROCEDURE Rcheck();

```

Question 3.3. En utilisant l'instance initiale, exécuter les commandes suivantes et donner le résultat affiché.

```

delete from enseignant where e_num=10;

update enseignant set e_num=21 where e_num=20;

insert into cours values(5,40,'Maths',3);

update cours set c_ens=40 where c_ens=20;

```

Question 3.4. *Décrire une nouvelle version de la fonction trigger Rcheck en complétant le schéma de fonction trigger ci-dessous :*

```

1 CREATE FUNCTION Rcheck ()
2     RETURNS TRIGGER AS
3 $$
4     DECLARE
5         ens int;
6     BEGIN
7         IF TG_RELNAME='enseignant' THEN
8
9             -- a completer
10
11         ELSIF TG_RELNAME='cours' THEN
12
13             -- a completer
14
15         END IF;
16         RETURN NULL;
17     END;
18 $$ LANGUAGE plpgsql;

```

Exercice IV :

Soit le schéma simplifié suivant :

```

1 create table emprunter(
2     num_abonne int ,
3     num_exemplaire int ,
4     date_retour date,
5     primary key(num_abonne, num_exemplaire)
6 );

```

et son instance :

```

-- Instance initiale
=> select * from emprunter;
  num_abonne | num_exemplaire | date_retour
-----+-----+-----

```

1	9 25/01/2014
1	7 25/01/2014
1	12 25/01/2014
3	8 10/02/2014
3	11 10/02/2014

(5 rows)

On souhaite mettre en oeuvre une règle de gestion qui consiste à :

ne pas autoriser un abonné à emprunter plus de 3 exemplaires par un trigger dont la fonction a pour prototype :

```

1 create or replace function testNbLivresEmp()
2   returns TRIGGER as
3   $$
4
5       -- a completer
6
7   $$ language plpgsql;
```

Question 4.1. Décrire cette fonction et son trigger de sorte à obtenir le résultat suivant :

```
=> insert into emprunter
> values (3,115,'15/1/2014'),
>        (3,117,'15/1/2014');
ERROR:  Echec - plafond atteint 3
```

```
=> select * from emprunter;
 num_abonne | num_exemplaire | date_retour
-----+-----+-----
          1 |              9 | 2014-01-25
          1 |              7 | 2014-01-25
          1 |             12 | 2014-01-25
          3 |              8 | 2014-02-10
          3 |             11 | 2014-02-10
(5 rows)
```

(tout est refusé)

Question 4.2. Décrire cette fonction et son trigger de sorte à obtenir le résultat suivant :

```
=> insert into emprunter
> values (3,115,'15/1/2014'),
>        (3,117,'15/1/2014');
```


NOTICE: Attention - exemplaire 117 refuse
INSERT 0 1

=> select * from emprunter;

num_abonne	num_exemplaire	date_retour
1	9	25/01/2014
1	7	25/01/2014
1	12	25/01/2014
3	8	10/02/2014
3	11	10/02/2014
3	115	15/01/2014

(6 rows)

(seuls les exemplaires en trop sont refusés)