M1 · Bases de données avancées

TD6 Les triggers

Partie 1 , Création des tables

SQL Shell (psql)

postgres=# CREATE DATABASE banque;

CREATE DATABASE

postgres=# \c banque

Vous êtes maintenant connecté à la base de données « banque » en tant qu'utilisateur « postgres ».

banque=# CREATE TABLE comptes (

banque(# numcompte integer PRIMARY KEY,

banque(# nomclient varchar(20) NOT NULL,

banque(# solde numeric(10,2) NOT NULL,

banque(# decouvert_autorise numeric(10,2) NOT NULL DEFAULT 0

banque(# CHECK (decouvert_autorise <=0)

banque(#);

CREATE TABLE

```
banque=# \d comptes
                          Table ¾ public.comptes ╗
      Colonne       |         Type          | Collationnement | NULL-able | Par dÚfaut
--------------------+-----------------------+-----------------+-----------+------------
 numcompte          | integer               |                 | not null  |
 nomclient          | character varying(20) |                 | not null  |
 solde              | numeric(10,2)         |                 | not null  |
 decouvert_autorise | numeric(10,2)         |                 | not null  | 0
Index :
    "comptes_pkey" PRIMARY KEY, btree (numcompte)
Contraintes de vÚrification :
    "comptes_decouvert_autorise_check" CHECK (decouvert_autorise <= 0::numeric)
```

Le but du TP est de mettre en place des contraintes d'intégrité sur les données et un système d'audit sur les transactions faites sur les comptes.

Créez la table ci-dessus et peuplez-la de quelques lignes de données.

```
INSERT INTO comptes VALUES (12345, 'David', 30, -1000);
INSERT INTO comptes VALUES (12346, 'Eric', 5000, -5000);
INSERT INTO comptes VALUES (12347, 'Emmanuel', 10000, -5000);
INSERT INTO comptes VALUES (12348, 'Nicolas', -900, -1000);
INSERT INTO comptes VALUES (12349, 'Francois', 500, -1000);
INSERT INTO comptes VALUES (12350, 'Christian', 0, 0);
```

```
banque=# SELECT * FROM comptes;
 numcompte | nomclient |  solde   | decouvert_autorise
-----------+-----------+----------+--------------------
     12345 | David     |    30.00 |           -1000.00
     12346 | Eric      |  5000.00 |           -5000.00
     12347 | Emmanuel  | 10000.00 |           -5000.00
     12348 | Nicolas   |  -900.00 |           -1000.00
     12349 | Francois  |   500.00 |           -1000.00
     12350 | Christian |     0.00 |               0.00
(6 lignes)
```

Syntaxe

```
CREATE TRIGGER nom
{BEFORE|AFTER} event
ON TABLE FOR EACH {ROW|STATEMENT}
WHEN (condition)
EXECUTE PROCEDURE func();

----- Variables :
-- NEW : pas de sens pour DELETE
-- OLD : pas de sens pour INSERT
-- TG_OP : l'operation qui a declanché le trigger  : INSERT UPDATE DELETE
---- event : INSERT UPDATE DELETE

CREATE OR REPLACE FONCTION func () RETURNS trigger AS $$
    BEGIN



    END;
$$ LANGUAGE plpgsql;
```

Partie 2 , *Mise en place des contraintes d'intégrité*

Contraintes génériques

D'autres contraintes sur les données ne peuvent pas être exprimées par les mécanismes vus jusqu'à maintenant

Exemple :

la note d'un examen est entre 0 et 20

le prix soldé d'un article est inférieur au prix entier

le salaire d'un manager est plus élevé que celui des ses subalternes

En SQL :

Contraintes sur une seule table : clause CHECK

Contraintes sur plusieurs tables : Assertions SQL

Triggers

- Objectif : surveiller l'état d'une BD et réagir quand une condition se présente
- Les *triggers* sont en général exprimés dans une syntaxe similaire aux assertions (voir plus loin) et incluent les parties suivantes:
  - **événement** (e.g., une opération de mise à jour de la BD)
  - **condition** (une condition qui déclenche l'exécution du trigger)
  - **action** (à réaliser quand la condition est satisfaite)

1. Le solde d'un compte ne doit jamais être inférieur au découvert autorisé.

```
-- Autre syntaxe :
ALTER TABLE comptes ADD CHECK (solde >= decouvert_autorise);
```

```
ALTER TABLE comptes
ADD CONSTRAINT jamais_inferieur_decouvert_autorise
CHECK (solde >= decouvert_autorise);
```

```
banque=# \d comptes
                              Table ¾ public.comptes ¶
      Colonne       |         Type          | Collationnement | NULL-able | Par dÚfaut
--------------------+-----------------------+-----------------+-----------+------------
 numcompte          | integer               |                 | not null  |
 nomclient          | character varying(20) |                 | not null  |
 solde              | numeric(10,2)         |                 | not null  |
 decouvert_autorise | numeric(10,2)         |                 | not null  | 0
Index :
    "comptes_pkey" PRIMARY KEY, btree (numcompte)
Contraintes de vÚrification :
    "comptes_decouvert_autorise_check" CHECK (decouvert_autorise <= 0::numeric)
    "jamais_inferieur_decouvert_autorise" CHECK (solde >= decouvert_autorise)
```

2. Si le solde d'un compte devient négatif, l'utilisateur doit être averti.

```
CREATE OR REPLACE FUNCTION compte_devient_negatif() RETURNS trigger AS $$
    BEGIN
        RAISE NOTICE 'Le solde du compte numero % devient négatif', NEW.numcompte;
        RETURN NULL;
    END;
    $$ LANGUAGE plpgsql;

CREATE TRIGGER devient_negatif
AFTER UPDATE ON comptes
FOR EACH ROW -- condition évaluée juste après la maj de chaque ligne affectée
WHEN (NEW.solde < 0) -- NEW : la ligne après la modification
EXECUTE PROCEDURE compte_devient_negatif();

-- Test :
UPDATE comptes
SET solde = solde - 600
WHERE numcompte = 12345 OR numcompte = 12349;
```

```
banque=# UPDATE comptes
banque-# SET solde = solde - 600
banque-# WHERE numcompte = 12345 OR numcompte = 12349;
NOTICE:  Le solde du compte numero 12345 devient n'gatif
NOTICE:  Le solde du compte numero 12349 devient n'gatif
UPDATE 2
```

3. Un compte ne peut être fermé (supprimé de la table) que si le solde est 0.

```
CREATE OR REPLACE FUNCTION compte_ne_peut_etre_ferme() RETURNS trigger AS $$
    BEGIN
        RAISE NOTICE 'Le compte % ne peut etre ferme (supprime) que si le solde est 0',
OLD.numcompte;
        -- Pour des triggers BEFORE de type FOR EACH ROW :
        -- si un trigger renvoie NULL, la mise à jour sur la ligne courante
        -- ainsi que tous les triggers suivants sur cette même ligne - sont annulés
        RETURN NULL;
    END;
    $$ LANGUAGE plpgsql;
```

```sql
CREATE TRIGGER supprimer_compte
BEFORE DELETE ON comptes
FOR EACH ROW
WHEN (OLD.solde <> 0)
EXECUTE PROCEDURE compte_ne_peut_etre_ferme();

-- Test :
DELETE FROM comptes;
```

```
banque=# DELETE FROM comptes;
NOTICE:  Le compte 12346 ne peut etre ferme (supprime) que si le solde est 0
NOTICE:  Le compte 12347 ne peut etre ferme (supprime) que si le solde est 0
NOTICE:  Le compte 12348 ne peut etre ferme (supprime) que si le solde est 0
NOTICE:  Le compte 12345 ne peut etre ferme (supprime) que si le solde est 0
NOTICE:  Le compte 12349 ne peut etre ferme (supprime) que si le solde est 0
DELETE 1
```

Partie 3 , Mise en place de règles de gestion

La banque a décidé d'appliquer des frais de 5% à toutes les transactions de retrait. Implémentez un trigger qui met en place cette nouvelle règle de gestion.

```sql
CREATE OR REPLACE FUNCTION frais_transactions_de_retrait() RETURNS trigger AS $$
    BEGIN
        NEW.solde := NEW.solde - (((OLD.solde - NEW.solde) * 5) / 100);
        RETURN NEW;
    END;
    $$ LANGUAGE plpgsql;

CREATE TRIGGER retrait
BEFORE UPDATE OF solde ON comptes
FOR EACH ROW
WHEN (OLD.solde > NEW.solde)
EXECUTE PROCEDURE frais_transactions_de_retrait();

-- Test :
UPDATE comptes
SET solde = solde - 100
WHERE numcompte = 12345;
```

Attention : *Un trigger « AFTER » avec une tel fonction :*

```
CREATE OR REPLACE FUNCTION frais_transactions_de_retrait() RETURNS trigger AS $$
    BEGIN
        UPDATE comptes SET solde = (NEW.solde - (((OLD.solde - NEW.solde) * 5) / 100))
WHERE numcompte = OLD.numcompte;
        RETURN NULL;
    END;
    $$ LANGUAGE plpgsql;
```

semble provoquer au moins une autre diminution du solde car le système interprète le retrait des 5% comme une nouvelle transaction de retrait.

Pour vérifier que le nouveau trigger et les triggers implémentés à la section précédente se déclenchent dans le bon ordre, testez les cas de figure suivants. Modifiez au besoin pour obtenir les bons comportements.

Les triggers pour le même évènement sont appelés dans l'ordre alphabétique.

1. Un utilisateur a un solde 100 Euros, un découvert autorisé de 100 Euros, et effectue un retrait de 100 Euros de son compte. Son compte doit être débité de 105 Euros, ce qui rend son solde négatif et doit déclencher un avertissement.

```
UPDATE comptes SET solde = 100, decouvert_autorise = -100 WHERE numcompte =
12345; (x 2), pour éliminer l'effet des frais sur un retrait
UPDATE comptes SET solde = solde - 100 WHERE numcompte = 12345;
```

```
banque=# UPDATE comptes SET solde = 100 , decouvert_autorise = -100 WHERE numcompte = 12345;
UPDATE 1
banque=# SELECT * FROM comptes;
 numcompte | nomclient | solde  | decouvert_autorise
-----------+-----------+--------+--------------------
     12346 | Eric      | 289.48 |           -5000.00
     12347 | Emmanuel  | 394.74 |           -5000.00
     12348 | Nicolas   | 394.74 |           -1000.00
     12349 | Francois  | 394.74 |           -1000.00
     12345 | David     | 100.00 |            -100.00
(5 lignes)


banque=# UPDATE comptes SET solde = solde - 100 WHERE numcompte = 12345;
NOTICE:  Le solde du compte numero 12345 devient n'gatif
UPDATE 1
banque=# SELECT * FROM comptes;
 numcompte | nomclient | solde  | decouvert_autorise
-----------+-----------+--------+--------------------
     12346 | Eric      | 289.48 |           -5000.00
     12347 | Emmanuel  | 394.74 |           -5000.00
     12348 | Nicolas   | 394.74 |           -1000.00
     12349 | Francois  | 394.74 |           -1000.00
     12345 | David     |  -5.00 |            -100.00
(5 lignes)
```

2. Un utilisateur a un solde de 2 Euros, un découvert autorisé de 100 Euros, et effectue un retrait de 100 Euros. Son compte doit être débité de 105 Euros, ce qui ferait passer son solde sous le découvert autorisé. Le retrait doit être refusé.

```
UPDATE comptes SET solde = 2, decouvert_autorise = -100 WHERE numcompte =
12348; (x 2), pour éliminer l'effet des frais sur un retrait
UPDATE comptes SET solde = solde - 100 WHERE numcompte = 12348;
```

```
banque=# UPDATE comptes SET solde = solde - 100 WHERE numcompte = 12348;
ERREUR:  la nouvelle ligne de la relation « comptes » viole la contrainte de vérification « jamais_inferieur_decouvert_autorise »
DÉTAIL : La ligne en échec contient (12348, Nicolas, -103.00, -100.00).
banque=# SELECT * FROM comptes;
 numcompte | nomclient | solde  | decouvert_autorise
-----------+-----------+--------+--------------------
     12349 | Francois  | 394.74 |           -1000.00
     12345 | David     |  -5.00 |            -100.00
     12347 | Emmanuel  | -17.64 |            -100.00
     12346 | Eric      |   2.00 |            -100.00
     12348 | Nicolas   |   2.00 |            -100.00
(5 lignes)
```

Partie 4 , Mise en place d'une table audit

La banque doit maintenir une trace de toutes les opérations portées sur les comptes bancaires. La table ci-dessous maintient l'historique des transactions.

```
CREATE TABLE audit (
      numoperation SERIAL PRIMARY KEY,
      numcompte INTEGER NOT NULL REFERENCES comptes,
      date_operation DATE NOT NULL,
      operation VARCHAR(10) NOT NULL
            CHECK (operation in ('RETRAIT', 'OUVERTURE', 'DEPOT', 'FERMETURE')),
      montant numeric(10,2)
);
```

```
banque=# \d audit
                                    Table ¼ public.audit ¶
    Colonne     |         Type          | Collationnement | NULL-able |                Par dÚfaut
----------------+-----------------------+-----------------+-----------+-------------------------------------------
 numoperation   | integer               |                 | not null  | nextval('audit_numoperation_seq'::regclass)
 numcompte      | integer               |                 | not null  |
 date_operation | date                  |                 | not null  |
 operation      | character varying(10) |                 | not null  |
 montant        | numeric(10,2)         |                 |           |
Index :
    "audit_pkey" PRIMARY KEY, btree (numoperation)
Contraintes de vÚrification :
    "audit_operation_check" CHECK (operation::text = ANY (ARRAY['RETRAIT'::character varying, 'OUVERTURE'::character varying, 'DEPOT'
::character varying, 'FERMETURE'::character varying]::text[]))
Contraintes de clÚs Útrangþres :
    "audit_numcompte_fkey" FOREIGN KEY (numcompte) REFERENCES comptes(numcompte)
```

Écrivez un trigger qui peuple la table `audit` à chaque transaction.
Prenez soin de faire en sorte que seules les opérations qui ont été complétées apparaissent dans la table d'audit. Par exemple, si un utilisateur tente de retirer un montant et que le solde devient inférieur au découvert autorisé, l'opération est refusée et elle ne devra donc pas apparaître dans la table `audit`.

```
CREATE OR REPLACE FUNCTION log_solde() RETURNS trigger AS $$
    DECLARE
    nom_operation VARCHAR(10);
    valeur_montant numeric(10,2);
    BEGIN
        IF (OLD.solde > NEW.solde) THEN
           nom_operation := 'RETRAIT';
           valeur_montant :=  (OLD.solde - NEW.solde);
        ELSE
           nom_operation := 'DEPOT';
           valeur_montant :=  (NEW.solde - OLD.solde);
        END IF;

        INSERT INTO audit VALUES (DEFAULT, NEW.numcompte, current_date ,
nom_operation , valeur_montant);
        RETURN NULL;
    END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER comptes_audit_sold
AFTER UPDATE OF solde ON comptes
FOR EACH ROW
EXECUTE PROCEDURE log_solde();
```

```
banque=# SELECT * FROM comptes;
 numcompte | nomclient | solde  | decouvert_autorise
-----------+-----------+--------+--------------------
     12346 | Eric      | 500.00 |            -100.00
     12348 | Nicolas   | 500.00 |            -100.00
     12349 | Francois  | 500.00 |            -100.00
     12347 | Emmanuel  | 500.00 |            -100.00
     12345 | David     | 500.00 |            -100.00
(5 lignes)


banque=# UPDATE comptes SET solde = solde - 100 WHERE numcompte = 12346;
UPDATE 1
banque=# UPDATE comptes SET solde = solde + 100 WHERE numcompte = 12347;
UPDATE 1
banque=# SELECT * FROM audit;
 numoperation | numcompte | date_operation | operation | montant
--------------+-----------+----------------+-----------+---------
            6 |     12346 | 2022-03-01     | RETRAIT   |  105.00
            7 |     12347 | 2022-03-01     | DEPOT     |  100.00
```

```
banque=# UPDATE comptes SET solde = solde - 600 WHERE numcompte = 12345;
ERREUR:  la nouvelle ligne de la relation « comptes » viole la contrainte de vérification « jamais_inferieur_decouvert_autorise »
DÉTAIL : La ligne en échec contient (12345, David, -130.00, -100.00).
banque=# SELECT * FROM audit;
 numoperation | numcompte | date_operation | operation | montant
--------------+-----------+----------------+-----------+---------
            6 |     12346 | 2022-03-01     | RETRAIT   |  105.00
            7 |     12347 | 2022-03-01     | DEPOT     |  100.00
(2 lignes)
```

Ouverture de compte

```
CREATE OR REPLACE FUNCTION log_insert() RETURNS trigger AS $$
    BEGIN
        INSERT INTO audit VALUES (DEFAULT, NEW.numcompte, current_date ,
'OUVERTURE' , NEW.solde);
        RETURN NULL;
    END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER comptes_audit_insert
AFTER INSERT ON comptes
FOR EACH ROW
EXECUTE PROCEDURE log_insert();
```

```
banque=# INSERT INTO comptes VALUES (96741, 'Xavier', 500, -600);
INSERT 0 1
banque=# SELECT * FROM comptes;
 numcompte | nomclient | solde  | decouvert_autorise
-----------+-----------+--------+--------------------
     96741 | Xavier    | 500.00 |            -600.00
     12345 | David     | 100.00 |            -100.00
     12346 | Eric      |  90.00 |            -100.00
     12347 | Emmanuel  |  90.00 |            -100.00
     12348 | Nicolas   |  90.00 |            -100.00
     12349 | Francois  |   0.00 |            -100.00
(6 lignes)


banque=# SELECT * FROM audit;
 numoperation | numcompte | date_operation | operation | montant
--------------+-----------+----------------+-----------+---------
           55 |     96741 | 2022-03-01     | OUVERTURE |  500.00
            6 |     12346 | 2022-03-01     | RETRAIT   |  105.00
            7 |     12347 | 2022-03-01     | DEPOT     |  100.00
           53 |     12349 | 2022-03-01     | FERMETURE |    0.00
(4 lignes)
```

Que se passe-t-il maintenant si on tente de fermer un compte ? Dans quel ordre les triggers sont-ils exécutés ? Si vous rencontrez d'autres problèmes, proposez des solutions et implémentez-les.

```
CREATE OR REPLACE FUNCTION log_delete() RETURNS trigger AS $$
    BEGIN
        INSERT INTO audit VALUES (DEFAULT, OLD.numcompte, current_date ,
'FERMETURE' , OLD.solde);
        RETURN OLD;
    END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER comptes_audit_delete
AFTER DELETE ON comptes
FOR EACH ROW
EXECUTE PROCEDURE log_delete();
```

```
banque=# SELECT * FROM comptes;
 numcompte | nomclient | solde  | decouvert_autorise
-----------+-----------+--------+--------------------
     12345 | David     | 100.00 |            -100.00
     12346 | Eric      |  90.00 |            -100.00
     12347 | Emmanuel  |  90.00 |            -100.00
     12348 | Nicolas   |  90.00 |            -100.00
     12349 | Francois  |   0.00 |            -100.00
(5 lignes)


banque=# DELETE FROM comptes;
NOTICE:  Le compte 12345 ne peut etre ferme (supprime) que si le solde est 0
NOTICE:  Le compte 12346 ne peut etre ferme (supprime) que si le solde est 0
NOTICE:  Le compte 12347 ne peut etre ferme (supprime) que si le solde est 0
NOTICE:  Le compte 12348 ne peut etre ferme (supprime) que si le solde est 0
ERREUR:  une instruction insert ou update sur la table « audit » viole la contrainte de clé
étrangère « audit_numcompte_fkey »
DÉTAIL : La clé (numcompte)=(12349) n'est pas présente dans la table « comptes ».
CONTEXTE : instruction SQL « INSERT INTO audit VALUES (DEFAULT, OLD.numcompte, current_date , 'FERMETURE' , OLD.solde) »
fonction PL/pgSQL log_delete(), ligne 3 à instruction SQL
banque=# SELECT * FROM audit;
 numoperation | numcompte | date_operation | operation | montant
--------------+-----------+----------------+-----------+---------
            6 |     12346 | 2022-03-01     | RETRAIT   |  105.00
            7 |     12347 | 2022-03-01     | DEPOT     |  100.00
(2 lignes)
```

➔ ***Problème !***

Autre méthode :

```
DROP TRIGGER comptes_audit_delete ON comptes;

--  Les triggers pour le même évènement sont appelés dans l'ordre alphabétique
--  et comme on a deja un ' TRIGGER supprimer_compte BEFORE DELETE ON comptes '
-- faut que notre nouveau trigger soit appelé APRES le trigger supprimer_compte
CREATE TRIGGER supprimer_le_compte
BEFORE DELETE ON comptes
FOR EACH ROW
EXECUTE PROCEDURE log_delete();
```

c'est mieux de faire un trigger AFTER car audit ça doit être pour les opérations qu'on a réussi a faire

```
banque=# SELECT * FROM comptes;
 numcompte | nomclient | solde  | decouvert_autorise
-----------+-----------+--------+--------------------
     96741 | Xavier    | 500.00 |            -600.00
     12345 | David     | 100.00 |            -100.00
     12346 | Eric      |  90.00 |            -100.00
     12347 | Emmanuel  |  90.00 |            -100.00
     12348 | Nicolas   |   0.00 |            -100.00
(5 lignes)


banque=# DELETE FROM comptes;
NOTICE:  Le compte 96741 ne peut etre ferme (supprime) que si le solde est 0
NOTICE:  Le compte 12345 ne peut etre ferme (supprime) que si le solde est 0
NOTICE:  Le compte 12346 ne peut etre ferme (supprime) que si le solde est 0
NOTICE:  Le compte 12347 ne peut etre ferme (supprime) que si le solde est 0
ERREUR:  UPDATE ou DELETE sur la table « comptes » viole la contrainte de clé étrangère « audit_numcompte_fkey » de la table « audit »
DÉTAIL : La clé (numcompte)=(12348) est toujours référencée à partir de la table « audit ».
```

Même problème !

On ne peut pas supprimer un compte car « numcompte » est une clé étrangère dans la table audit.

Proposition de solution 1 : Ajouter une colonne « statuts du compte » a la table comptes et ensuite le trigger qui gère ce qui se passe avant une requête « `DELETE FROM comptes ..` » va jamais supprimer une ligne de la table mais va juste changé le statuts du compte de « ACTIF » a « PAS ACTIF ».

Proposition de solution 2 : Compte ne soit plus une clé étrangère dans la table « audit ». (proposition validé par le prof).

```
ALTER TABLE audit DROP CONSTRAINT "audit_numcompte_fkey";
-- ALTER TABLE audit ADD CONSTRAINT "audit_numcompte_fkey" FOREIGN KEY (numcompte) REFERENCES comptes(numcompte);
```

Méthode du prof :

```
CREATE TRIGGER audit
AFTER UPDATE OR DELETE OR INSERT
ON comptes
FOR EACH ROW
EXECUTE PROCEDURE audit();

CREATE OR REPLACE FUNCTION audit()
RETURNS trigger AS $$
      DECLARE
        montant numeric(10,2);
      BEGIN
        CASE WHEN TG_OP = 'INSERT' THEN
             INSERT INTO audit (numcompte, date_operation, operation, montant)
             VALUES (NEW.numcompte, CURRENT_DATE, 'OUVERTURE',NEW.solde);

             WHEN TG_OP = 'DELETE' THEN
             INSERT INTO audit (numcompte, date_operation, operation, montant)
             VALUES (OLD.numcompte, CURRENT_DATE, 'FERMETURE',OLD.solde);

             WHEN TG_OP = 'UPDATE' THEN
             montant := NEW.solde - OLD.solde ;
             IF montant <> 0 THEN
             INSERT INTO audit (numcompte, date_operation, operation, montant)
             VALUES (NEW.numcompte, CURRENT_DATE,
                     CASE WHEN montant > 0
                          THEN 'DEPOT'
                          ELSE 'RETRAIT'
                     END ;
                     , montant) ;
             END IF;
         END CASE;
         RETURN NULL;
         END;
 $$ LANGUAGE plpgsql ;
```

Partie 5 , Une règle de gestion plus complexe

La banque met en place un plafond de 1000 Euro sur les retraits qui ont lieu sur une journée. Sans modifier le schéma de la base, écrivez un trigger qui interdit les retraits qui violeraient cette contrainte.

Les triggers pour le même évènement sont appelés dans l'ordre alphabétique. On a déjà :

```
CREATE TRIGGER retrait
BEFORE UPDATE OF solde ON comptes
FOR EACH ROW
WHEN (OLD.solde > NEW.solde)
EXECUTE PROCEDURE frais_transactions_de_retrait();
```

Mtn, on veut ajouter APRES ,un nouveau trigger :

```
CREATE OR REPLACE FUNCTION limite_par_jour() RETURNS trigger AS $$
    DECLARE
    nb_retraits_qui_ont_lieu_ajd  numeric(10,2);
    BEGIN
       SELECT SUM(montant) INTO nb_retraits_qui_ont_lieu_ajd
       FROM audit
       WHERE operation = 'RETRAIT' AND numcompte = OLD.numcompte AND
date_operation = current_date
       GROUP BY numcompte;

       IF NOT FOUND THEN
            RETURN NEW;
       ELSEIF (nb_retraits_qui_ont_lieu_ajd + (OLD.solde - NEW.solde)) >= 1000
THEN
            RETURN NULL; -- La mise à jour déclenchante ne sera pas exécutée
       END IF;
        RETURN NEW;
    END;
$$ LANGUAGE plpgsql;
```

Dans L'Alphabet français U > R (le trigger **retrait** *) :*

```
CREATE TRIGGER un_retrait_est_sous_le_plafond
BEFORE UPDATE OF solde ON comptes
FOR EACH ROW
EXECUTE PROCEDURE limite_par_jour();
```

Vérifiez que les triggers se déclenchent dans le bon ordre en testant les cas suivants.

1. Un client a un solde de 1000 Euros, un découvert autorisé de 100 Euros. Il effectue deux retraits de 500 Euros, donc il est débité deux fois de 525 Euros. La seconde transaction devrait être refusée pour dépassement du plafond quotidien de retrait.

```
banque=# SELECT * FROM comptes;
 numcompte | nomclient | solde  | decouvert_autorise
-----------+-----------+--------+--------------------
     96741 | Xavier    | 500.00 |            -600.00
     12345 | David     | 100.00 |            -100.00
     12346 | Eric      |  90.00 |            -100.00
     12347 | Emmanuel  |  90.00 |            -100.00
     12348 | Nicolas   |   0.00 |            -100.00
(5 lignes)


banque=# UPDATE comptes SET solde = 1000 WHERE numcompte = 12348;
UPDATE 1
banque=# SELECT * FROM comptes;
 numcompte | nomclient |  solde  | decouvert_autorise
-----------+-----------+---------+--------------------
     96741 | Xavier    |  500.00 |            -600.00
     12345 | David     |  100.00 |            -100.00
     12346 | Eric      |   90.00 |            -100.00
     12347 | Emmanuel  |   90.00 |            -100.00
     12348 | Nicolas   | 1000.00 |            -100.00
(5 lignes)


banque=# UPDATE comptes SET solde = solde - 500 WHERE numcompte = 12348;
UPDATE 1
banque=# SELECT * FROM audit;
 numoperation | numcompte | date_operation | operation | montant
--------------+-----------+----------------+-----------+---------
           64 |     12348 | 2022-03-01     | DEPOT     | 1000.00
           65 |     12348 | 2022-03-01     | RETRAIT   |  525.00
(2 lignes)
```

```
banque=# UPDATE comptes SET solde = solde - 500 WHERE numcompte = 12348;
UPDATE 0
banque=# SELECT * FROM comptes;
 numcompte | nomclient | solde  | decouvert_autorise
-----------+-----------+--------+--------------------
     96741 | Xavier    | 500.00 |            -600.00
     12345 | David     | 100.00 |            -100.00
     12346 | Eric      |  90.00 |            -100.00
     12347 | Emmanuel  |  90.00 |            -100.00
     12348 | Nicolas   | 475.00 |            -100.00
(5 lignes)
```

2. Un client a un solde de 1000 Euros, un découvert autorisé de 10 Euros. Il effectue deux retraits de 500 Euros, donc il est débité deux fois de 525 Euros. La transaction est-elle bloquée pour raison de découvert ou de plafond de retrait ? Expliquez en déterminant l'ordre dans lequel les triggers sont déclenchés.

```
banque=# UPDATE comptes SET solde = 1000, decouvert_autorise = -10 WHERE numcompte = 12347;
UPDATE 1
banque=# SELECT * FROM comptes;
 numcompte | nomclient |  solde  | decouvert_autorise
-----------+-----------+---------+--------------------
     96741 | Xavier    |  500.00 |            -600.00
     12347 | Emmanuel  | 1000.00 |             -10.00
     12345 | David     |  100.00 |            -100.00
     12346 | Eric      |   90.00 |            -100.00
     12348 | Nicolas   | 1000.00 |             -10.00
(5 lignes)


banque=# UPDATE comptes SET solde = solde - 500 WHERE numcompte = 12347;
UPDATE 1
banque=# SELECT * FROM comptes;
 numcompte | nomclient |  solde  | decouvert_autorise
-----------+-----------+---------+--------------------
     96741 | Xavier    |  500.00 |            -600.00
     12347 | Emmanuel  |  475.00 |             -10.00
     12345 | David     |  100.00 |            -100.00
     12346 | Eric      |   90.00 |            -100.00
     12348 | Nicolas   | 1000.00 |             -10.00
(5 lignes)


banque=# SELECT * FROM audit;
 numoperation | numcompte | date_operation | operation | montant
--------------+-----------+----------------+-----------+---------
           64 |     12348 | 2022-03-01     | DEPOT     | 1000.00
           65 |     12348 | 2022-03-01     | RETRAIT   |  525.00
           66 |     12348 | 2022-03-01     | DEPOT     |  525.00
           67 |     12347 | 2022-03-01     | DEPOT     |  910.00
           68 |     12347 | 2022-03-01     | RETRAIT   |  525.00
(5 lignes)
```

```
banque=# UPDATE comptes SET solde = solde - 500 WHERE numcompte = 12347;
UPDATE 0
banque=# SELECT * FROM comptes;
 numcompte | nomclient |  solde  | decouvert_autorise
-----------+-----------+---------+--------------------
     96741 | Xavier    |  500.00 |            -600.00
     12347 | Emmanuel  |  475.00 |             -10.00
     12345 | David     |  100.00 |            -100.00
     12346 | Eric      |   90.00 |            -100.00
     12348 | Nicolas   | 1000.00 |             -10.00
(5 lignes)
```

La transaction est-elle bloquée pour raison de découvert ou de plafond de retrait ?

Plafond de retrait·

La contraint sur le découvert est mise en place par une contrainte d'intégrité