TP5 SQL R307

Exercice 1

c) Tester la procédure directement dans PostgreSQL. Cas normaux et erreur :

```
banque_db=# Select * FROM Compte;
-- Test de la procédure :
CALL virement(1, 2, 100);
SELECT numCompte, solde
FROM Compte
WHERE numCompte IN (1, 2);

CALL virement(2, 3, 500);
SELECT numCompte, solde
FROM Compte
WHERE numCompte IN (2, 3);
 numcompte | solde  | typecompte
-----------+--------+------------
         4 |    500 | COURANT
         5 | 100000 | EPARGNE
         6 |     50 | EPARGNE
         7 |    500 | COURANT
         8 |   4000 | COURANT
         9 |    500 | COURANT
        10 |  14000 | EPARGNE
         1 |     50 | COURANT
         2 |  12500 | EPARGNE
         3 |   1500 | COURANT
(10 rows)

ERROR:  Solde insuffisant sur le compte 1
CONTEXT:  PL/pgSQL function virement(integer,integer,real) line 16 at RAISE
 numcompte | solde
-----------+-------
         1 |    50
         2 | 12500
(2 rows)
```

```
CALL
 numcompte | solde
-----------+-------
         2 | 12000
         3 |  2000
(2 rows)

banque_db=# CALL virement(999, 2, 100);
ERROR:  Compte source inexistant : 999
CONTEXT:  PL/pgSQL function virement(integer,integer,real) line 12 at RAISE
banque_db=# CALL virement(1, 3, 200);
ERROR:  Solde insuffisant sur le compte 1
CONTEXT:  PL/pgSQL function virement(integer,integer,real) line 16 at RAISE
banque_db=# SELECT numCompte, solde
FROM Compte
WHERE numCompte IN (1, 3);
 numcompte | solde
-----------+-------
         1 |    50
         3 |  2000
(2 rows)

banque_db=#
```

d) Expliquer l'intérêt d'implémenter cette logique sous forme de procédure stockée plutôt que côté application Java.

1. Garantie de l'atomicité
La procédure stockée s'exécute dans une transaction gérée par le SGBD.
En cas d'erreur (par exemple solde insuffisant), une exception est levée et toutes les modifications sont automatiquement annulées.
Cela garantit qu'un virement est exécuté entièrement ou pas du tout.

2. Centralisation de la logique métier critique
Le virement est une règle métier sensible.
En la plaçant dans la base de données, toutes les applications clientes (Java, scripts SQL, outils d'administration) utilisent exactement la même logique, sans duplication de code.

3. Sécurité et cohérence des données
La procédure empêche toute modification incohérente des soldes, même en cas d'appel direct à la base de données.
La règle ne peut pas être contournée par un simple UPDATE SQL.

4. Réduction des échanges application / base
Le virement regroupe plusieurs opérations SQL en un seul appel, ce qui réduit les allers-retours entre l'application et la base de données.

Exercice 2

d) Tester le trigger à l'aide :

- d'une mise à jour SQL directe
- de la procédure stockée du premier exercice.

```
banque_db=# UPDATE Compte
SET solde = solde + 100
WHERE numCompte = 3;
UPDATE 1
banque_db=# SELECT *
FROM HistoriqueCompte
WHERE numCompte = 3
ORDER BY dateModification DESC;
 id | numcompte | anciensolde | nouveausolde |      datemodification
----+-----------+-------------+--------------+----------------------------
 11 |         3 |        2200 |         2300 | 2026-01-09 14:54:18.113819
```

Si le solde ne change pas on n'ajoute pas de ligne inutilement

```
banque_db=# SELECT COUNT(*)
FROM HistoriqueCompte
WHERE numCompte = 3;
 count
-------
     5
(1 row)

banque_db=# UPDATE Compte
SET solde = solde
WHERE numCompte = 3;
UPDATE 1
banque_db=# SELECT COUNT(*)
FROM HistoriqueCompte
WHERE numCompte = 3;
 count
-------
     5
(1 row)
```

Test avec la procédure :

```
banque_db=# CALL virement(2, 4, 500);
CALL
banque_db=# SELECT *
FROM HistoriqueCompte
WHERE numCompte IN (2, 4)
ORDER BY dateModification DESC;
 id | numcompte | anciensolde | nouveausolde |      datemodification
----+-----------+-------------+--------------+----------------------------
 12 |         2 |       12000 |        11500 | 2026-01-09 14:59:26.44548
 13 |         4 |         500 |         1000 | 2026-01-09 14:59:26.44548
```

Test virement refusé :

```
banque_db=# SELECT COUNT(*)
FROM HistoriqueCompte
WHERE numCompte IN (1,2);
 count
-------
     7
(1 row)

banque_db=# CALL virement(1, 2, 1000);
ERROR:  Solde insuffisant sur le compte 1
CONTEXT:  PL/pgSQL function virement(integer,integer,real) line 16 at RAISE
banque_db=# SELECT COUNT(*)
FROM HistoriqueCompte
WHERE numCompte IN (1,2);
 count
-------
     7
(1 row)
```

e) Expliquer l'intérêt d'un trigger par rapport à une journalisation réalisée côté application.

1. Règle incontournable
Le trigger est exécuté automatiquement par le SGBD lors d'une modification du solde, quel que soit le moyen utilisé (application Java, script SQL, outil d'administration).
La journalisation ne peut donc pas être contournée.

2. Indépendance vis-à-vis de l'application
La logique de journalisation est totalement indépendante de l'application.
Même si plusieurs applications accèdent à la base de données, toutes les modifications sont tracées de manière uniforme.

3. Cohérence et fiabilité
Le trigger s'exécute dans la même transaction que la modification du solde.
Si la transaction est annulée, aucune trace n'est enregistrée, ce qui garantit la cohérence entre les données et l'historique.

4. Centralisation de la règle transversale
La journalisation est une règle transverse qui concerne l'ensemble du système.
Le trigger permet de centraliser cette règle directement au niveau de la base de données, sans duplication de code.