Implémentation des SGBDs

Gestion des Transactions

Auteur :
Nathanaël Bayle

15 Janvier 2021

Énoncé

Considérons le schéma de base de données suivant :

- Compte-cheque(num-cc, num-client, solde)
- Compte-epargne(num-ce, num-client, solde)
- Decision-credit(num-dossier, num-client, décision)

1. Implémenter cette base de données sous le SGBD Oracle et remplissez les tables Compte-cheque et Compte-epragne avec des données exemple.

2. Écrire les quatres procédures PL/SQL suivantes :
   (a) Transfert(num-client, from-num-cpt, to-num-cpt, m) qui permet de transférer un montant m du compte from-num-cpt vers le compte to-num-cpt.
   (b) Virement-cc(num-client, num-cc, m) qui permet de réaliser un virement d'un montant m vers le compte chèques num-cc.
   (c) Virement-ce(num-client, num-ce, m) qui permet de réaliser un virement d'un montant m vers le compte épargne num-ce.
   (d) Traitement-credit(num-client, m) qui permet de vérifier une demande de crédit du client num-client. Si le montant de crédit demandé m est 3 fois supérieur à la somme des soldes de ces comptes chèques et épargne, le crédit est accepté (insertion d'une ligne dans la table Decision-crédit avec une décision OK). Sinon, la demande de crédit est rejetée (insertion d'une ligne dans la table Decision-crédit avec une décision KO).

3. Simuler des problèmes de concurrence
   (a) Ajouter à vos procédures un délai artificiel (utiliser la fonction dbms_lock.sleep(nb_seconds IN NUMBER)). Ce délai artificiel permet de simuler un délai qui serait lié à des communications réseau, une attente de confirmation d'un utilisateur, etc.
   (b) Exécuter vos procédures à partir de sessions différentes pour simuler au moins trois problème de concurrence.

4. Modifier vos procédures en utilisant les verrous pour éviter les problèmes précédents.

5. Comment peut-on éviter les problèmes de concurrence que vous avez rencontrez sans utiliser les verrous ?

Solution

Question 1

```sql
CREATE TABLE Compte_cheque (
    num_cc varchar(10) NOT NULL,
    num_client INT NOT NULL,
    solde INT
);

CREATE TABLE Compte_epargne (
    num_ce varchar(10) NOT NULL,
    num_client INT NOT NULL,
    solde INT
  );

CREATE TABLE Decision_credit (
    num_dossier varchar(10) NOT NULL,
    num_client INT NOT NULL,
    decision varchar(2)
);

INSERT INTO Compte_cheque VALUES ('CCHQ001', 1, 1000);
INSERT INTO Compte_cheque VALUES ('CCHQ002', 2, 1500);
INSERT INTO Compte_cheque VALUES ('CCHQ003', 3, 2000);
INSERT INTO Compte_cheque VALUES ('CCHQ004', 4, 2500);

INSERT INTO Compte_epargne VALUES ('LIVA001', 1, 100);
INSERT INTO Compte_epargne VALUES ('LIVA002', 2, 200);
INSERT INTO Compte_epargne VALUES ('LIVA003', 3, 100);
INSERT INTO Compte_epargne VALUES ('LIVA004', 4, 100);

INSERT INTO Decision_credit VALUES ('DOS001', 1, null);
INSERT INTO Decision_credit VALUES ('DOS002', 2, null);
INSERT INTO Decision_credit VALUES ('DOS003', 3, null);
INSERT INTO Decision_credit VALUES ('DOS004', 4, null);
```

On a donc :

Compte Chèque

NUM_CC	NUM_CLIENT	SOLDE
CCHQ001	1	1000
CCHQ002	2	1500
CCHQ003	3	2000
CCHQ004	4	2500

Compte Épargne

NUM_CE	NUM_CLIENT	SOLDE
LIVA001	1	100
LIVA002	2	200
LIVA003	3	100
LIVA004	4	100

Question 2

Transfert

```
CREATE OR REPLACE PROCEDURE Transfert
(num INT, from_num_cpt VARCHAR, to_num_cpt VARCHAR, m INT)
AS
    x INT;
    y INT;
BEGIN
    IF (from_num_cpt LIKE 'CCHQ%') THEN
        SELECT solde INTO x FROM Compte_cheque
            WHERE num_cc = from_num_cpt;
        SELECT solde INTO y FROM Compte_epargne
            WHERE num_ce = to_num_cpt;
        UPDATE Compte_cheque SET solde = x - m
            WHERE num_client = num;
        UPDATE Compte_epargne SET solde = y + m
            WHERE num_client = num;
    ELSE
        SELECT solde INTO x FROM Compte_epargne
            WHERE num_ce = from_num_cpt;
        SELECT solde INTO y FROM Compte_cheque
            WHERE num_cc = to_num_cpt;
        UPDATE Compte_epargne SET solde = solde - m
            WHERE num_client = num;
        UPDATE Compte_cheque SET solde = solde + m
            WHERE num_client = num;
    END IF;
    COMMIT;
END;
/
```

Virement_cc

```
CREATE OR REPLACE PROCEDURE Virement_cc
(num INT, cc VARCHAR, m INT)
AS
    x INT;
    y INT;
BEGIN
    SELECT solde INTO x FROM Compte_cheque
        WHERE num_client = num;
    SELECT solde INTO y FROM
        (SELECT * FROM Compte_cheque WHERE num_cc = cc)
        WHERE num_cc = cc;
    IF ( cc LIKE 'CCHQ%') THEN
        UPDATE Compte_cheque SET solde = x - m
            WHERE num_client = num;
        UPDATE Compte_cheque SET solde = y + m
            WHERE num_cc = cc;
    END IF;
    COMMIT;
END;
/
```

Virement_ce

```
CREATE OR REPLACE PROCEDURE Virement_ce
(num INT, ce VARCHAR, m INT)
AS
    x INT;
    y INT;
BEGIN
    SELECT solde INTO x FROM Compte_epargne
        WHERE num_client = num;
    SELECT solde INTO y FROM
        (SELECT * FROM Compte_epargne WHERE num_ce = ce)
        WHERE num_ce = ce;
    IF ( ce LIKE 'LIVA%') THEN
        UPDATE Compte_epargne SET solde = solde - m
            WHERE num_client = num;
        UPDATE Compte_epargne SET solde = solde + m
            WHERE num_ce = ce;
    END IF;
    COMMIT;
END;
/
```

Traitement_credit

```
CREATE OR REPLACE PROCEDURE Traitement_credit
(num INT, m INT)
AS
    x int;
BEGIN
    SELECT SUM(solde) INTO x FROM
        (SELECT * FROM Compte_cheque UNION SELECT * FROM
            Compte_epargne)
        WHERE num_client = 1;
    IF ( m > 3*x ) THEN
        UPDATE Decision_credit SET decision = 'OK'
            WHERE num_client = num;
    ELSE
        UPDATE Decision_credit SET decision = 'KO'
            WHERE num_client = num;
    END IF;
    COMMIT;
END;
/
```

Question 3

Perte de mise à jour

On modifie les procédures Transfert et Virement_cc pour simuler une perte de mise à jour.

```
CREATE OR REPLACE PROCEDURE Transfert
(num INT, from_num_cpt VARCHAR, to_num_cpt VARCHAR, m INT)
AS
    x INT;
    y INT;
BEGIN
    IF (from_num_cpt LIKE 'CCHQ%') THEN
        SELECT solde INTO x FROM Compte_cheque
            WHERE num_cc = from_num_cpt;
        SELECT solde INTO y FROM Compte_epargne
            WHERE num_ce = to_num_cpt;
        DBMS_LOCK.sleep(10);
        UPDATE Compte_cheque SET solde = x - m
            WHERE num_client = num;
        UPDATE Compte_epargne SET solde = y + m
            WHERE num_client = num;
    ELSE
        SELECT solde INTO x FROM Compte_epargne
            WHERE num_ce = from_num_cpt;
        SELECT solde INTO y FROM Compte_cheque
            WHERE num_cc = to_num_cpt;
        DBMS_LOCK.sleep(10);
        UPDATE Compte_epargne SET solde = solde - m
            WHERE num_client = num;
        UPDATE Compte_cheque SET solde = solde + m
            WHERE num_client = num;
    END IF;
    COMMIT;
END;
/

CREATE OR REPLACE PROCEDURE Virement_cc
(num INT, cc VARCHAR, m INT)
AS
    x INT;
    y INT;
BEGIN
    SELECT solde INTO x FROM Compte_cheque
        WHERE num_client = num;
    SELECT solde INTO y FROM
        (SELECT * FROM Compte_cheque WHERE num_cc = cc)
        WHERE num_cc = cc;
    DBMS_LOCK.sleep(10);
    IF (cc LIKE 'CCHQ%') THEN
        UPDATE Compte_cheque SET solde = x - m
            WHERE num_client = num;
        UPDATE Compte_cheque SET solde = y + m
            WHERE num_cc = cc;
    END IF;
    COMMIT;
END;
/
```

Depuis la session 1 :

```
SQL> EXEC Virement_cc(2,'CCHQ001',500);
PL/SQL procedure successfully completed.
```

Depuis la session 2 :

```
SQL> EXEC Transfert(1,'CCHQ001','LIVA001',500);
PL/SQL procedure successfully completed.
```

On obtient pour Compte_cheque :

NUM_CC	NUM_CLIENT	SOLDE
CCHQ001	1	500
CCHQ002	2	1000
CCHQ003	3	2000
CCHQ004	4	2500

On obtient pour Compte_epargne :

NUM_CE	NUM_CLIENT	SOLDE
LIVA001	1	600
LIVA002	2	200
LIVA003	3	100
LIVA004	4	100

On a de la perte de mise à jour lorsque deux transaction T_1 et T_2 modifient simultanément la même valeur. Les modifications effectué par Virement_cc sont perdu pour le compte CCHQ001.

Virement_cc	Transfert	BD
read solde		solde = 1000
	read solde	
solde = solde + 500		
write solde		solde = 1500
	solde = solde - 500	
	write solde	solde = 500

Lecture Impropre

On modifie les procédures Traitement_credit et Virement_cc pour simuler une lecture impropre.

```
CREATE OR REPLACE PROCEDURE Virement_cc
(num INT, cc VARCHAR, m INT)
AS
    x INT;
    y INT;
BEGIN
    SELECT solde INTO x FROM Compte_cheque
        WHERE num_client = num;
    SELECT solde INTO y FROM
        (SELECT * FROM Compte_cheque WHERE num_cc = cc)
        WHERE num_cc = cc;
    IF ( cc LIKE 'CCHQ%') THEN
        UPDATE Compte_cheque SET solde = x - m
            WHERE num_client = num;
        UPDATE Compte_cheque SET solde = y + m
            WHERE num_cc = cc;
    END IF;
    COMMIT;
END;
/

CREATE OR REPLACE PROCEDURE Traitement_credit( num_cli INT, m
    INT)
AS
    x int;
BEGIN
    SELECT SUM(solde) INTO x FROM (select * from
        Compte_cheque UNION SELECT * FROM Compte_epargne)
        WHERE num_client = 1;
    DBMS_LOCK.sleep(10);
    IF ( m > 3*x ) THEN
        UPDATE Decision_credit SET decision = 'OK' WHERE
            num_client = num_cli;
    ELSE
        UPDATE Decision_credit SET decision = 'KO' WHERE
            num_client = num_cli;
    END IF;
    EXCEPTION WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE ('Probleme rencontre');
END;
/
```

Depuis la session 1 :

```
SQL> EXEC Traitement_credit(1,3000);
```

Depuis la session 2 :

```
SQL> EXEC Virement_cc(1,'CCHQ002',500);
PL/SQL procedure successfully completed.
```

Depuis la session 1 :

```
PL/SQL procedure successfully completed.
```

On obtient pour Compte_cheque :

NUM_CC	NUM_CLIENT	SOLDE
CCHQ001	1	500
CCHQ002	2	2000
CCHQ003	3	2000
CCHQ004	4	2500

On obtient pour Decision_credit :

NUM_DOSSIE	NUM_CLIENT	DECISION
DOS001	1	KO
DOS002	2	
DOS003	3	
DOS004	4	

On a de la lecture impropre lorsque T_2 lit une valeur modifié, non validée par T_1. Ce qui entraîne une incohérence. Si tout c'était bien passé, la décision aurait du être 'OK' et non 'KO'.

Virement$_c c$	Traitement$_c redit$	BD
	read SUM(solde)	SUM(solde) = 1100
solde = solde - 500		
write solde		SUM(solde) = 600
	3000 < 3*1100	
	write decision	decision = 'KO'

Lecture non reproductible

On utilise une nouvelle procédure Lecture_non_reproductible et Virement_cc pour simuler une lecture non reproductible.

```
CREATE OR REPLACE PROCEDURE Lecture_non_reproductible
AS
        x INT;
BEGIN
        DBMS_OUTPUT.enable;
        SELECT solde INTO x FROM Compte_cheque
            WHERE num_client = 1;
        DBMS_OUTPUT.put_line('Solde du client 1 : ' || x);
        DBMS_LOCK.sleep(10);
        SELECT solde INTO x FROM Compte_cheque
            WHERE num_client = 1;
        DBMS_OUTPUT.put_line('Solde du client 1 : ' || x);
END;
/

CREATE OR REPLACE PROCEDURE Virement_cc
(num INT, cc VARCHAR, m INT)
AS
    x INT;
    y INT;
BEGIN
    SELECT solde INTO x FROM Compte_cheque
        WHERE num_client = num;
    SELECT solde INTO y FROM
        (SELECT * FROM Compte_cheque WHERE num_cc = cc)
        WHERE num_cc = cc;
    IF ( cc LIKE 'CCHQ%') THEN
        UPDATE Compte_cheque SET solde = x - m
            WHERE num_client = num;
        UPDATE Compte_cheque SET solde = y + m
            WHERE num_cc = cc;
    END IF;
    COMMIT;
END;
/
```

Depuis la session 1 :

```
SQL> EXEC Lecture_non_reproductible;
```

Depuis la session 2 :

```
SQL> EXEC Virement_cc(2,'CCHQ001',500);
PL/SQL procedure successfully completed.
```

Depuis la session 1 :

```
Solde du client 1 : 1000
Solde du client 1 : 1500

PL/SQL procedure successfully completed.
```

On a une lecture non reproductible lorsque T2 lit deux valeurs de A différentes. On a donc bien un problème de lecture non reproductible ici.

Lecture_non_reproductible	Virement_cc	BD
read solde		solde = 1000
	solde = solde + 500	
	write solde	solde = 1500
read solde		solde = 1500

Question 4

Transfert avec Verrou

```
CREATE OR REPLACE PROCEDURE Transfert
(num INT, from_num_cpt VARCHAR, to_num_cpt VARCHAR, m INT)
AS
    x INT;
    y INT;
BEGIN
    LOCK TABLE Compte_cheque IN EXCLUSIVE MODE;
    LOCK TABLE Compte_epargne IN EXCLUSIVE MODE;
    IF (from_num_cpt LIKE 'CCHQ%') THEN
        SELECT solde INTO x FROM Compte_cheque
            WHERE num_cc = from_num_cpt;
        SELECT solde INTO y FROM Compte_epargne
            WHERE num_ce = to_num_cpt;
        UPDATE Compte_cheque SET solde = x - m
            WHERE num_client = num;
        UPDATE Compte_epargne SET solde = y + m
            WHERE num_client = num;
    ELSE
        SELECT solde INTO x FROM Compte_epargne
            WHERE num_ce = from_num_cpt;
        SELECT solde INTO y FROM Compte_cheque
            WHERE num_cc = to_num_cpt;
        UPDATE Compte_epargne SET solde = solde - m
            WHERE num_client = num;
        UPDATE Compte_cheque SET solde = solde + m
            WHERE num_client = num;
    END IF;
    COMMIT;
END;
/
```

Virement_cc avec Verrou

```
CREATE OR REPLACE PROCEDURE Virement_cc
(num INT, cc VARCHAR, m INT)
AS
    x INT;
    y INT;
BEGIN
    LOCK TABLE Compte_cheque IN EXCLUSIVE MODE;
    SELECT solde INTO x FROM Compte_cheque
        WHERE num_client = num;
    SELECT solde INTO y FROM
        (SELECT * FROM Compte_cheque WHERE num_cc = cc)
        WHERE num_cc = cc;
    IF ( cc LIKE 'CCHQ%') THEN
        UPDATE Compte_cheque SET solde = x - m
            WHERE num_client = num;
        UPDATE Compte_cheque SET solde = y + m
            WHERE num_cc = cc;
    END IF;
    COMMIT;
END;
/
```

Virement_ce avec Verrou

```
CREATE OR REPLACE PROCEDURE Virement_ce
(num INT, ce VARCHAR, m INT)
AS
    x INT;
    y INT;
BEGIN
    LOCK TABLE Compte_epargne IN EXCLUSIVE MODE;
    SELECT solde INTO x FROM Compte_epargne
        WHERE num_client = num;
    SELECT solde INTO y FROM
        (SELECT * FROM Compte_epargne WHERE num_ce = ce)
        WHERE num_ce = ce;
    IF ( ce LIKE 'LIVA%') THEN
        UPDATE Compte_epargne SET solde = solde - m
            WHERE num_client = num;
        UPDATE Compte_epargne SET solde = solde + m
            WHERE num_ce = ce;
    END IF;
    COMMIT;
END;
/
```

Traitement_credit avec Verrou

```
CREATE OR REPLACE PROCEDURE Traitement_credit
(num INT, m INT)
AS
    x int;
BEGIN
    LOCK TABLE Compte_cheque IN EXCLUSIVE MODE;
    LOCK TABLE Compte_epargne IN EXCLUSIVE MODE;
    LOCK TABLE Decision_credit IN EXCLUSIVE MODE;
    SELECT SUM(solde) INTO x FROM
        (SELECT * FROM Compte_cheque UNION SELECT * FROM
            Compte_epargne)
        WHERE num_client = 1;
    IF ( m > 3*x ) THEN
        UPDATE Decision_credit SET decision = 'OK'
            WHERE num_client = num;
    ELSE
        UPDATE Decision_credit SET decision = 'KO'
            WHERE num_client = num;
    END IF;
    COMMIT;
END;
/
```

Question 5

Afin d'éviter des problèmes de concurrences, on peut mettre en place des points de sauvegardes. La reprise sur panne garantit le retour au dernier état cohérent de la base précédant l'interruption, mais c'est au programmeur de définir ces points de sauvegarde dans le code des programmes. Il est donc possible de subdiviser une transaction en plusieurs étapes :

- en sauvant les informations modifiées à la fin de chaque étape,
- en gardant la possibilité de valider l'ensemble des mises a jour ou bien de tout annuler