TD9 : Les transactions, pannes et concurrence d'accès

Corrigé

Partie I - Gestion des transactions

Question I.1

a) Update COMPTE set SOLDE = SOLDE + 1000 where COMPTE.NOM='A';

Insert into OPERATION values (date_heure, 'DEPOS', 'A', 100);

Update COMPTE set SOLDE = SOLDE - 1000 where COMPTE.NOM='B';

Insert into OPERATION values (date_heure, 'RETRAIT', 'B', 100);

b) La contrainte d'intégrité peut être par exemple que la somme des dépôts soit égale à la somme des retraits. Les deux opérations doivent donc être exécutées ou aucune.

Question I.2 :

a) A la demande de validation, les contraintes peuvent être vérifiées par le serveur BD et les opérations sont éventuellement toutes annulées en cas de transgression de règle d'intégrité.

Begin, commit, abort (rollback), savepoint.

b) Les problèmes de concurrence et de pannes des transactions. Des protocoles doivent etre mis en œuvre pour répondre à ces deux problèmes.

Partie II - Reprise à chaud

Question II.1 :

On utilise la technique dite des pages ombres. La mise à jour n'est plus écrite à sa place mais à une nouvelle adresse, si bien que l'ancienne version n'est pas écrasée.

La technique consiste à reporter toutes les pages modifiées restant en cache au moment de la validation sur une partie libre du disque dans un premier temps. Ensuite, il reste à écraser l'ancienne table des pages par la nouvelle en une entrée sortie.

Avantage : rien à faire à la reprise.

Problème : Il faut plusieurs instructions assembleur pour écraser la table des pages, ce qui crée une fenêtre de vulnérabilité (validation pas entièrement atomique).

Le nombre d'I/O du procédé quand on modifie N pages :

- Pendant la transaction : M ≤ N pages écrites sur disques dans une partie libre
- A la demande de validation : N-M I/O pour écrire les pages modifiées non encore présentes sur disque dans une partie libre, 1 I/O pour écraser l'ancienne table des pages, éventuellement des I/O pour libérer les pages ombres (selon la gestion de la mémoire).

Le granule de verrouillage est forcément la page.

Question II.2 :

a) Atomicité : il faut stocker les images avant modification des valeurs mises à jour par la transaction pour pouvoir défaire les transactions non validées en cas de panne.

Durabilité : forcer le flush de toutes pages modifiées restant en cache par les transactions validées quand elles commettent.

b) L'avantage évident de cette méthode est de rendre le commit complètement atomique. De plus, les I/O disque au commit sont reduites, seules les pages restantes en cache sont reportées.

Le nombre d'I/O du procédé quand on modifie N pages :

- Pendant la transaction : 1 I/O partielle (peut contenir les infos de plusieurs transactions) dans le journal avant, M ≤ N pages écrites sur disques dans une partie libre
- A la demande de validation : N-M I/O pour écrire les pages modifiées non encore présentes sur disque dans une partie libre.

c) Toute modification de page doit être reportée dans le journal avec la valeur avant modification de la page.

```
Mise_a_jour(Ti, Page, offset, taille, val_après[]) {
 For(i=0 ;i<taille ;i++)
Val_avant[i] = Page[offset+i] ;
 Ajout_journal(Ti, Page, offset, taille, val_avant[]);
 For(i=0 ;i<taille ;i++)
Page[offset+i] = val_après[i] ;
}
```

```
Insert(Ti, Page, offset, taille, val[]) {
 Ajout_journal(Ti, Page, offset, taille, NULL);
 For(i=0 ;i<taille ;i++)
Page[offset+i] = val[i] ;
}
```

Question II.3 :

En sauvegardant les modifications dans un journal après, on autorise le système à ne pas forcément flusher le buffer de chaque transaction à sa validation. On peut ainsi repartir les I/O disques au cours du temps, et les rendre complètement désynchroniser des opérations effectuées par les transactions. Les ressources sont ainsi utilisées de façon optimale.

Question II.4 :

On peut regrouper les opérations effectuées par une même transaction sur un même objet, voire compresser les journaux. Toutefois, le journal avant a très peu de chance d'être saturé vu qu'il ne contient que les informations des transactions actives. Quant au journal après, il est régulièrement vidé sur bande.

Question II.5 :

Les pages du journal avant concernant des transactions inactives (terminées) peuvent être détruites. La partie du journal après antérieure à la dernière sauvegarde de la BD peut être détruit.
Oui… Le journal avant permet de défaire les transactions non validées et le journal après permet de refaire les transactions validées.

Question II.6 :

Temps	T_1	T_2	*BD*	*Journal*	*Gestionnaire de récupération*
t_1	*DébutT.*		A=*0* B=*10* C=*20*		
t_2				*(*Début*, 1)*	
t_3	*Écrire(50,* A*)*				
t_4				*(*Défaire*, 1,* A*: 0)*	
t_5		*DébutT.*			
t_6				*(*Refaire*, 1,* A*: 50)*	
t_7				*(*Début*, 2)*	
t_8		*Écrire(100,* B*)*			
t_9				*(*Défaire*, 2,* B*: 10)*	
t_{10}	*Écrire(60,* C*)*				
t_{11}				*(*Défaire*, 1,* C*: 20)*	
t_{12}				*(*Refaire*, 2,* B*: 100)*	
t_{12}				*(*Refaire*, 1,* C*: 60)*	
t_{14}	*ConfirmerT.*				
t_{15}			B = *100*		
t_{16}				*(*Confirmer*, 1)*	
t_{17}		*Écrire(200,* A*)*			
t_{18}				*(*Défaire*, 2,* A*: 50)*	
t_{19}			C = *60*		
t_{20}		*ConfirmerT.*			
t_{21}					*Panne*

Question II.7 :

t_{22}			*A = 50*		*(*Défaire*,2,*A*: 50)*
t_{23}			*C = 60*		*(*Refaire*,1,*C*: 60)*
t_{24}			*B = 10*		*(*Défaire*,2,*B*: 10)*

Partie III - Reprise à froid

Question III.1 :

On refait à partir du journal d'images après, et on défait avec le journal avant… Si le journal avant est sur le même disque que la BD, il faut assurer sa sécurité face aux pannes en enregistrant ses modifications dans le journal après…

Question III.2 :

a) Id_transaction, adresse logique ou physique de l'image, image après

b) On part de la fin du journal après et on refait toutes les transactions validées depuis la dernière sauvegarde de la base. On peut réduire le temps de la reprise en partant du dernier point de contrôle strict ou un peu avant en cas de point de contrôle flou…

Question IV.1 :

Selon l'arrivée des requêtes de chaque transaction au SGBD, il peut y avoir le problème de perte d'opération suivant :

(T2) **Lire** (X, v2);

(T1) **Lire** (X, v1);

(T2) v2 := v2 + 1;

(T2) **Ecrire** (X, v2) ;

(T1) v1 := v1 + 1;

(T1) **Ecrire** (X, v1);

La modification de X faite par T2 est perdue…

Question IV.2 :

Selon l'arrivée des requêtes de chaque transaction au SGBD, il peut y avoir les problèmes d'observation incohérente suivant :

(T3) **Lire** (A, v3);

(T4) **Lire** (A, v4);

(T3) v3 := v3 + 1;

(T3) **Ecrire** (A, v3);

(T4) imprimer v4;

(T3) **Lire** (B, v3);

(T3) v3 := v3 + 1;

(T4) **Lire** (B, v4);

(T3) **Ecrire** (B, v3);

(T4) imprimer v4;

La modification de X faite par T2 est perdue…

Question IV.3 :

La donnée A doit contenir une valeur positive (A>0), avant les opérations suivantes A = 100.

(T1) **Lire** (A, v1); → A=100

(T2) **Lire** (A, v2); → A=100

(T1) v1 := v1 - 90;

(T2) v2 := v2 - 90;

(T1) **Ecrire** (A, v1);

(T3) **Ecrire** (B, v3);

En fin d'exécution, A est négatif bien qu'aucune transaction n'ait eu l'impression de transgresser la contrainte…

Question IV.4 :

Solution 1 : On pose des verrous partagés en lecture et exclusifs en écriture (protocole de verrouillage deux phases)

Solution 2 : Nous définissons un ordre total sur les transactions (protocole d'estampillage). Cet ordre est obtenu par une estampille de transaction qui est un numéro associé à chaque transaction lors de son initialisation. Ce numéro peut être une valeur d'horloge ou un compteur uniformément croissant. Nous définissons un granule comme étant l'unité de données partageable, et un conflit entre deux transactions sur un même granule comme la suite de deux actions non permutables (L/E, E/L, E/E) de ces transactions sur ce granule. Pour assurer que les transactions qui entrent en conflit sur un granule y accèdent dans l'ordre des estampilles, nous associons deux estampilles à chaque granule :

- l'estampille d'écriture EE(A) qui mémorise le numéro de la dernière transaction en écriture
- l'estampille de lecture EL(A) qui mémorise le numéro de la dernière transaction en écriture