Cours Bases de données 2ème année IUT

Concurrence d'accès

Anne Vilnat

Plan

Plusieurs utilisateurs en même temps...

Le problème...

Quand plusieurs utilisateurs manipulent les mêmes données en même temps, il faut arbitrer entre :

- Disponibilité de l'information : ne pas bloquer tout le monde parce que une personne travaille
  $\rightarrow$*ne pas arrêter toutes les connexions à la SNCF quand une personne fait une réservation*,
- Cohérence de l'information : ne pas rendre les données incohérentes en tenant compte de plusieurs demandes en concurrence en même temps
  $\rightarrow$*donner la même place de train à 2 personnes...*

Plusieurs utilisateurs en même temps...

Définitions

- Transaction : unité logique de traitement = suite d'opérations interrogeant et/ou modifiant la BD et pour laquelle l'ensemble des opérations doit être soit validé, soit annulé. En Oracle :
  - début d'une transaction = ordre SQL ou fin de la précédente
  - fin d'une transaction = instruction COMMIT ou ROLLBACK.
- Accès concurrent : plusieurs utilisateurs accèdent en même temps à la même donnée dans la base.
- Base cohérente : contraintes d'intégrité respectées.Si lors d'une transaction une BD passe d'un état cohérent à un autre état cohérent : $\rightarrow$ l'intégrité des données est sauvegardée.
- Consistance des données: Si SGBD garantit que les données utilisées par une transaction ne sont pas modifiées par des requêtes d'autres transactions pendant cette transaction.

Verrous

Définition

Pour assurer la consistance : verrouiller ces données pendant la durée de la transaction.3 types de verrous :

- verrous partagés (*shared lock*), pour lire des données avec l'intention d'y faire des mises à jour.
- verrous exclusifs (*exclusive lock*), pour modifier des données.
- verrous globaux (*global lock*)pour bloquer un ensemble de données, généralement une table toute entière.

Collision: 2 transactions accèdent en même temps à la même donnée $\rightarrow$ perte de cohérence.

Un verrou est posé jusqu'à la fin de la transaction en cours.
COMMIT ou ROLLBACK = relâcher tous les verrous placés par une transaction.
CREATE TABLE,... modif des données $\rightarrow$ COMMIT implicite.

Collision de type perte de mise à jour : le problème, et ce qu'on voudrait

T_1 et T_2 modifient simultanément la quantité *qte*.

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		Lire qte	
t_2		qte←qte+3000	
t_3	qte=4000	Écrire qte `COMMIT`	
t_4			Lire qte
t_5			qte←qte+500
t_6			Écrire qte
	qte=4500		`COMMIT`

Collision de type perte de mise à jour : la vraie vie...

Or, ici :

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		Lire qte	
t_2			Lire qte
t_3		qte←qte+3000	
t_4	qte=4000	Écrire qte `COMMIT`	
t_5			qte←qte+500
t_6	**qte=???**		Écrire qte `COMMIT`

À la fin, *qte* : 1000 ? ou 1500 ? ou 4500 ?

Collision de type perte de mise à jour : la vraie vie...

Or, ici :

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		Lire qte	
t_2			Lire qte
t_3		qte←qte+3000	
t_4	qte=4000	Écrire qte `COMMIT`	
t_5			qte←qte+500
t_6	**qte=1500**		Écrire qte `COMMIT`

À la fin, *qte* : 1000 ? ou 1500 ? ou 4500 ?

Collision de type perte de mise à jour : solution

Temps	État base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		**Verrou exclusif** sur qte + Lire qte	
t_2			**Attente** pour poser **verrou exclusif** sur qte
t_3		qte←qte+3000	
t_4	qte=4000	Écrire qte `COMMIT`	
t_5			**Verrou exclusif** sur qte
t_6			Lire qte
t_7			qte←qte+500
t_8	**qte=4500**		Écrire qte `COMMIT`

Collision de type lecture impropre (ou inconsistante): ce qu'on voudrait

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		Lire qte	
t_2		qte←qte+3000	
t_3	qte=4000	Écrire qte	
t_4	qte=1000	Annuler T_1 (`ROLLBACK`)	
t_5			Lire qte
t_6			qte←qte+500
t_7	**qte=1500**		Écrire qte `COMMIT`

Collision de type lecture impropre (ou inconsistante): la vraie vie...

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		Lire qte	
t_2		qte←qte+3000	
t_3	qte=4000	Ecrire qte	
t_4			Lire qte
t_5			qte←qte+500
t_6	qte=4500		Écrire qte `COMMIT`
t_7	**qte=???**	Annuler T_1 (`ROLLBACK`)	

qte : 1000 ? ou 1500 ? ou 4500 ?

Collision de type lecture impropre (ou inconsistante): la vraie vie...

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		Lire qte	
t_2		qte←qte+3000	
t_3	qte=4000	Ecrire qte	
t_4			Lire qte
t_5			qte←qte+500
t_6	qte=4500		Écrire qte `COMMIT`
t_7	**qte=4500**	Annuler T_1 (`ROLLBACK`)	

qte : 1000 ? ou 1500 ? ou 4500 ?

Collision de type lecture impropre : solution

Temps	État base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		**Verrou exclusif** sur qte + Lire qte	
t_2		qte←qte+3000	
t_3	qte=4000	Écrire qte	
t_4			**Attente** d'un **verrou exclusif** sur qte
t_5	qte=1000	Annuler T_1 (`ROLLBACK`)	**Verrou exclusif** sur qte
t_6			Lire qte
t_7			qte←qte+500
t_8	**qte=1500**		Écrire qte `COMMIT`

Collision de type lecture non reproductible : ce qu'on voudrait

Temps	État base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		Lire qte	
t_2		Traitement impliquant qte	
t_3	**qte=1000**	Lire qte	
t_4	qte=1000		Lire qte qte←qte+1000
t_5	qte=2000		Écrire qte `COMMIT`

Collision de type lecture non reproductible : la vraie vie...

Temps	État base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		Lire qte	
t_2		Traitement impliquant qte	
t_3	qte=1000		**Verrou exclusif** sur qte Lire qte qte←qte+1000
t_4	qte=2000		Écrire qte `COMMIT`
t_5	**qte=???**	Lire qte	

Quelle valeur de qte est relue par T1 : 1000 ? ou 1500 ? ou 2000 ?

Collision de type lecture non reproductible : la vraie vie...

Temps	État base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		Lire qte	
t_2		Traitement impliquant qte	
t_3	qte=1000		**Verrou exclusif** sur qte Lire qte qte←qte+1000
t_4	qte=2000		Écrire qte `COMMIT`
t_5	**qte=2000**	Lire qte	

Collision de type lecture non reproductible : solution

Temps	État base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		**Verrou partagé** sur qte + Lire qte	
t_2			**Attente verrou exclusif** sur qte
t_3		Traitement impliquant qte	
t_4	**qte=1000**	Lire qte + Traitement impliquant qte `COMMIT`	
t_5	qte=1000		**Verrou exclusif** sur qte Lire qte qte←qte+1000
t_6	**qte=2000**		Ecrire qte + `COMMIT`

Niveaux de cohérence d'une transaction

Une donnée est dite salie si elle a été modifiée par une transaction non confirmée (par un COMMIT). Pour une transaction T, on peut exiger que T satisfasse une ou plusieurs des quatre propriétés suivantes :

1. T ne modifie pas des données salies par d'autres transactions.
2. T ne confirme pas ses changements avant la fin de la transaction.
3. T ne lit pas des données salies par d'autres transactions.
4. D'autres transactions ne salissent pas des données lues par T avant que T ne soit terminée.

Niveaux de cohérence d'une transaction

On dit que **T est cohérente de**:

- niveau 0 si T vérifie (1)→ pas de problème de perte de mise à jour.
- niveau 1 si T vérifie (1) et (2).
  Si une transaction est annulée, pas nécessaire de défaire explicitement les modifications antérieures à l'annulation.
- niveau 2 si T vérifie (1), (2) et (3) → pas de problème de perte de mise à jour et de lecture impropre.
- niveau 3 si T vérifie (1), (2), (3) et (4) → pas de problème de perte de mise à jour, de lecture impropre et permet d'assurer que les lectures sont reproductibles.
  → Isolation totale de la transaction.

Niveau 3 : Transactions sérialisables = l'exécution concurrente est équivalente à une exécution séquentielle

Blocage fatal (deadlock)

Verrous : solution pour les collisions MAIS des risques...
Point de vue de T1 : 40+50+30= ? Mais avec T2 :

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	somme=0 Enr_1=40 Enr_2=50 Enr_3=30		
t_1	somme=40	SELECT Enr_1 somme←somme+Enr_1	
t_2	somme=90	SELECT Enr_2 somme←somme+Enr_2	
t_3	Enr_3=20		SELECT Enr_3 UPDATE Enr_3
t_4	Enr_1=50		SELECT Enr_1 UPDATE Enr_1
t_5	**somme=110**	SELECT Enr_3 somme←somme+Enr_3	

Blocage fatal : verrous?

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	S=0 E_1=40 E_2=50 E_3=30		
t_1		**Verrou partagé** E_1 SELECT E_1	
	S=40	S←S+E_1	

Blocage fatal : verrous?

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	S=0 E_1=40 E_2=50 E_3=30		
t_1		**Verrou partagé** E_1 SELECT E_1	
	S=40	S←S+E_1	
t_2		**Verrou partagé** E_2 SELECT E_2	
	S=90	S←S+E_2	

Blocage fatal : verrous?

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	S=0 E_1=40 E_2=50 E_3=30		
t_1		**Verrou partagé** E_1 SELECT E_1	
	S=40	S←S+E_1	
t_2		**Verrou partagé** E_2 SELECT E_2	
	S=90	S←S+E_2	
t_3			**Verrou exclusif** E_3
			SELECT E_3
	E_3=20		UPDATE E_3

Blocage fatal : verrous?

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	S=0 E_1=40 E_2=50 E_3=30		
t_1	S=40	**Verrou partagé** E_1 SELECT E_1 S←S+E_1	
t_2	S=90	**Verrou partagé** E_2 SELECT E_2 S←S+E_2	
t_3			**Verrou exclusif** E_3
	E_3=20		SELECT E_3 UPDATE E_3
t_4			**Attente verrou exclusif** sur E_1

Blocage fatal : verrous?

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	S=0 E_1=40 E_2=50 E_3=30		
t_1	S=40	**Verrou partagé** E_1 SELECT E_1 S←S+E_1	
t_2	S=90	**Verrou partagé** E_2 SELECT E_2 S←S+E_2	
t_3			**Verrou exclusif** E_3
	E_3=20		SELECT E_3 UPDATE E_3
t_4			**Attente verrou exclusif** sur E_1
t_5		**Attente verrou**	

Blocage fatal : quand?

4 conditions nécessaires pour l'existence d'un blocage fatal :

1. Accès aux ressources en exclusion mutuelle.
2. Attente circulaire :
   - T_1 a vérouillé Enr_1 et demande à accéder à Enr_2,
   - T_2 a vérouillé Enr_2 et demande à accéder à Enr_3,
   - T_3 a vérouillé Enr_3 et demande à accéder à Enr_4,
   - $\vdots$
   - T_n a vérouillé Enr_n et demande à accéder à Enr_1.
3. Au moins une transaction détient des ressources et en demande d'autres.
4. Non-réquisition des ressources.

$\rightarrow$ régle de Havender pour éviter les attentes circulaires, chaque transaction demande et donc vérouille les ressources dans le même ordre

Règle de Havender

Temps	État base	Transaction T_1	Transaction T_2
t_0	S=0 E_1=40 E_2=50 E_3=30		
t_1	S=40	**Verrou partagé** sur E_1 SELECT E_1 S←S+E_1	
t_2	S=90	**Verrou partagé** sur E_2 SELECT E_2 S←S+E_2	
t_3			**Attente verrou exclusif** sur E_1

... →

Règle de Havender (fin)

$\ldots\rightarrow$

t_4	**S=120**	**Verrou partagé** sur E_3 SELECT E_3 $S \leftarrow S+E_3$ COMMIT	
t_5	E_1=50		**Verrou exclusif** sur E_1 SELECT E_1 UPDATE E_1
t_6	E_3=20		**Verrou exclusif** sur E_3 SELECT E_3 UPDATE E_1 COMMIT

Problème de la tasse de café...

Verrous libérés à la fin d'une transaction, soit par `COMMIT`, soit par `ROLLBACK`.
Problème si intervention de l'utilisateur au cours de la transaction.

Exemple en pseudo code

Connexion...
Exécution de "SELECT prix FROM article
    WHERE numeroArticle="+ numArticle; → verrou partagé sur la ligne
    ⋮
Affichage("Introduisez le prix de l article: ");
Saisie (nouveauPrix); → peut être long!
Exécution de "UPDATE article
                    SET prix=" +nouveauPrix+
                    "WHERE numeroArticle=" +numArticle;
Exécution de "COMMIT;"

Problème de la tasse de café...

Solution

Lorsque le programme requiert une intervention venant de l'utilisateur : travailler en deux phases distinctes.

- acquisition des données et donc intervention de l'utilisateur
- démarrer une transaction, réaliser les accès à la base et terminer la transaction.

Et l'utilisateur peut aller boire un café sans pénaliser tout le monde!

Visibilité des données

Où voit-on quoi?

Où voit-on les données non validées par un COMMIT :
à l'intérieur de la transaction en cours
Accessibles aux autres utilisateurs après la validation de la transaction.

Que voit-on?

Si SELECT sur une ou plusieurs tables :
$\rightarrow$ toutes les données telles qu'elles étaient au début de la requête, même si d'autres utilisateurs modifient la table et valident leurs modifications pendant ce temps.

Visibilité des données(2)

Elargir la règle

On peut élargir cette règle à une transaction ne comportant que des consultations (SELECT).
Débuter la transaction par :
SET TRANSACTION READ ONLY
et la terminer par ROLLBACK ou COMMIT.

Comment

- Enregistrement de la date de début de la transaction
- si une table est modifiée, le SGBD prend un cliché SNAPSHOT de la table
  - blocs modifiés : dans un rollback segment (avant modification) pour pouvoir reconstituer la version initiale
- ROLLBACK ou COMMIT libère les rollback segment

Verrous de ligne Oracle

Chaque fois qu'un utilisateur effectue une modification sur des données d'une table, par une commande :
UPDATE, INSERT, ou DELETE,
ou encore par une requête :
SELECT... FOR UPDATE OF :
Oracle pose automatiquement un *verrou de ligne* sur la ou les lignes concernées.
Un seul type de verrou de ligne : pour empêcher la ligne d'être modifiée par une autre transaction tant que le verrou n'est pas libéré par la commande COMMIT ou ROLLBACK (implicite ou explicite).
Chaque fois qu'il pose un verrou de ligne, Oracle pose aussi un *verrou de table* sur la table.

5 types de verrous de table Oracle

Verrous de table posés :

- implicitement si :
  UPDATE, DELETE, INSERT, ou
  SELECT... FOR UPDATE OF
- explicitement par LOCK TABLE (RARE)

But

Pour empêcher un autre utilisateur de modifier la structure de la table (ou de la supprimer!) pendant que les opérations sont en cours...

5 types de verrous de table Oracle

Opération SQL	Verrou ligne	Verrou table
`SELECT`	Non	Aucun
`INSERT`	Oui	RX
`UPDATE`	Oui	RX
`DELETE`	Oui	RX
`SELECT... FOR UPDATE`	Oui	RS
`LOCK TABLE ... IN :`		
`ROW SHARE MODE`	Non	RS
`ROW EXCLUSIVE MODE`	Non	RX
`SHARE MODE`	Non	S
`SHARE EXCLUSIVEMODE`	Non	SRX
`EXCLUSIVE MODE`	Non	X

Pas de verrou pour un SELECT,→ toujours possible, même si des verrous ont été posés (sur des lignes ou sur la table)

5 types de verrous de table Oracle

Les différents verrous :

- EXCLUSIVE : le verrou exclusif sur une table (X),
- SHARE : le verrou partagé sur une table (S),
- ROW EXCLUSIVE : le verrou exclusif pour une ligne (RX),
- ROW SHARE (ou SHARE UPDATE) : le verrou partagé pour une ligne (RS),
- SHARE ROW EXCLUSIVE : le verrou partagé exclusif pour une ligne (SRX).

Spécificités d'Oracle

- Partagé/Exclusif : idem cas général
- Spécificité Oracle : verrou de ligne / verrou de table

Si un verrou est posé sur une ligne : pas de conséquence sur les autres lignes de la table.
Si un verrou est posé sur une table, il concerne toutes les lignes de la table.

Verrous RS, RX et SRX

- Le verrou ROW SHARE (RS) est posé implicitement sur les lignes concernées par une instruction SELECT qui utilise la clause FOR UPDATE:

SELECT ... FOR UPDATE
[OF[[schema.]table.]attribut [,...]]

L'option OF attribut permet de spécifier les tables susceptibles d'être modifiées lorsque la requête fait intervenir plusieurs tables.

- Par défaut, toutes les tables sont verouillées.

Verrous RS, RX et SRX

- Le verrou ROW EXCLUSIVE (RX) est posé implicitement sur les lignes concernées par des instructions UPDATE, DELETE ou INSERT.
- Une seule transaction peut demander un verrou SHARE ROW EXCLUSIVE (SRX) sur une table. Les autres transactions ne peuvent demander aucun verrou exclusif ou partagé sur la table sauf un verrou de type RS.

Compatibilité entre types de verrous

Verrou posé par une transaction	Verrous impossibles pour les autres transactions	Verrous possibles pour les autres transactions
RS	X	RS, RX, S, SRX
RX	S, SRX, X	RS, RX
S	RX, SRX, X	RS, S
SRX	RX, S, SRX, X	RS
X	RS, RX, S, SRX, X	

Instruction LOCK TABLE

L'utilisateur peut poser ses propres verrous sur ses propres tables ou toutes les tables s'il est DBA.

Syntaxe

```
LOCK TABLE {nomTable | nomVue
                [,nomTable | ,nomVue[, ...]]} IN
  {ROW SHARE | ROW EXCLUSIVE
                     | SHARE UPDATE
                     | SHARE
                     | EXCLUSIVE
                     | SHARE ROW EXCLUSIVE} MODE
   [NOWAIT]
```

NOWAIT $\rightarrow$ si le verrou ne peut être posé à cause d'autres verrous, l'utilisateur ne souhaite pas attendre jusqu'à ce soit possible, mais qu'il souhaite reprendre le contrôle du processus.

Interblocage et visibilité des verrous

Où voir les verrous existants

La vue V$LOCK du dictionnaire des données repertorie les verrous actifs de la base de données.

Oracle = simplicité!

Sous Oracle : géré par le noyau.
Risque détecté: annulation de l'instruction à l'origine de l'interblocage

$\rightarrow$ Risque de surprises en employant d'autres SGBD!

Les points de repère : définition

Définition

Possible de subdiviser une transaction en plusieurs étapes en

- sauvant les infos modifiées à la fin de chaque étape,
- en gardant la possibilité soit
  - de valider l'ensemble des mises à jour,
  - d'annuler une partie des mises à jour à la fin de la transaction.

→ Insérer des *points de repère*, ou SAVEPOINT.

création

La création d'un point de repère se fait avec l'instruction :
SAVEPOINT pointRepere;

Usage pour annulation

Pour annuler la partie de la transaction depuis un point de repère :
ROLLBACK TO [SAVEPOINT] pointRepere;

Exemple

Exemple

```
UPDATE employe ...;
SAVEPOINT p1;
UPDATE employe ...;
SAVEPOINT p2;
INSERT INTO employe ...;
SAVEPOINT p3;
UPDATE employe ...;
IF vSal < 2000 THEN ROLLBACK TO p2; END IF;
```

Si vSal <2000 : on annule les mises à jour (un INSERT et un UPDATE) depuis le point de repère *p2*
on conserve les MAJ effectuées avant lui, → annulation des verrous placés depuis *p2* mais conservation de ceux placés auparavant.