

Bases de données & Concurrency

Maggie LEKPA

Introduction

Une BD est amenée à être partagée entre plusieurs utilisateurs concurrents via des commandes SQL, Bloc PL/SQL, programme externe, ...

De nombreux problèmes se posent :

- ✓ En cas de panne que se passe t'il ? : **Fiabilité** du système (objet du dernier cours)
- ✓ Plusieurs utilisateurs sont amenés à effectuer des modifications simultanément sur la base de données : **Concurrence d'accès**

Solution: La gestion des transactions

Notion de Transaction

Une *transaction* est une unité « logique » d'opérations d'interrogation ou de modifications d'une base de données formant un tout :
l'ensemble doit être soit validé soit annulé.

Une transaction démarre par l'exécution de n'importe quel ordre SQL.

Elle se termine :

- soit de façon implicite (arrêt normal ou anormal de session etc...)
- soit de façon explicite par l'un des deux ordres **COMMIT** ou **ROLLBACK**.

Notion de Transaction – validation

COMMIT : permet de terminer explicitement une transaction par validation des données; libère la ressource modifiée et rend les modifications accessibles aux autres utilisateurs.

ROLLBACK : permet de terminer explicitement une transaction par annulation de toutes les modifications de données effectuées; les ressources sont libérées.

Les instructions DDL (Create, Alter, Drop...) valident implicitement les transactions.

Une déconnexion entraîne une validation implicite.

Notion de Transaction

EXEMPLE :

*UPDATE Client SET prenom='Martin' WHERE id=13;
COMMIT;*

Notion de Transaction

Il est possible de définir des points de sauvegarde dans une transaction. Cela permet de valider toutes les modifications ou une partie d'entre elles et en annuler d'autres.

On crée un point de sauvegarde avec le mot clé :

SAVEPOINT *<nom_repère>*

Pour annuler la partie d'une transaction à partir d'un point de sauvegarde X, on exécute : *ROLLBACK TO X*

Notion de Transaction

EXEMPLE :

UPDATE client set nom='Nom1' WHERE id=1;

SAVEPOINT A;

UPDATE client set nom='Nom2' WHERE id=1;

SAVEPOINT B;

UPDATE client set nom='Nom3' WHERE id=1;

SAVEPOINT C;

ROLLBACK TO B;

ID	PRENOM	NOM	EMAIL	VILLE
1	1 Flavie	Nom2	f.da.costa@example.com	Pomoy

Notion de Transaction

<pre>--USER1 UPDATE client set nom='NomA' WHERE id=1; SAVEPOINT A; SELECT * FROM USER1.CLIENT;</pre>				
Sortie de script x Résultat de requête x				
SQL Toutes les lignes extraites : 20 en 0,003 secondes				
ID	PRENOM	NOM	EMAIL	VILLE
1	1 Flavie	NomA	f.da.costa@example.com	Pomoy



<pre>-- USER2 SELECT * FROM USER1.CLIENT</pre>				
Sortie de script x Résultat de requête x				
SQL Toutes les lignes extraites : 20 en 0,002 secondes				
ID	PRENOM	NOM	EMAIL	VILLE
1	1 Flavie	Da costa	f.da.costa@example.com	Pomoy

Feuille de calcul Query Builder				
<pre>--USER1 UPDATE client set nom='NomA' WHERE id=1; SAVEPOINT A; COMMIT; SELECT * FROM USER1.CLIENT;</pre>				
Sortie de script x Résultat de requête x Résultat de requête 1 x				
SQL Toutes les lignes extraites : 20 en 0,002 secondes				
ID	PRENOM	NOM	EMAIL	VILLE
1	1 Flavie	NomA	f.da.costa@example.com	Pomoy



Feuille de calcul Query Builder				
<pre>-- USER2 SELECT * FROM USER1.CLIENT</pre>				
Sortie de script x Résultat de requête x				
SQL Toutes les lignes extraites : 20 en 0 secondes				
ID	PRENOM	NOM	EMAIL	VILLE
1	1 Flavie	NomA	f.da.costa@example.com	Pomoy

Propriété des transactions

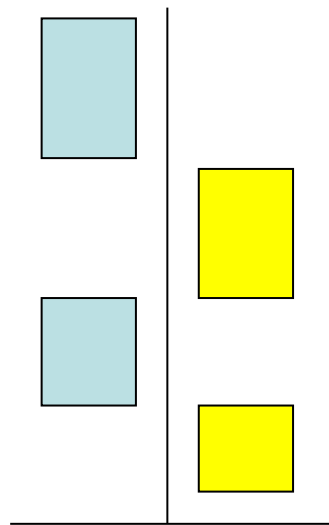
Une transaction est composée d'une suite de requêtes dépendantes à la base qui doivent vérifier les propriétés 'ACID':

- **A - Atomicité** : Une transaction doit effectuer toutes ses mises à jour ou ne rien faire du tout.
- **C - Cohérence** : La transaction doit faire passer la base de données d'un état cohérent à un autre.
- **I - Isolation** : une transaction se déroule sans être perturbée par d'autres transactions; tout se passe comme si elle s'exécutait seule.
- **D - Durabilité** : Dès qu'une transaction valide ses modifications, le système doit garantir qu'elles seront conservées en cas de panne.

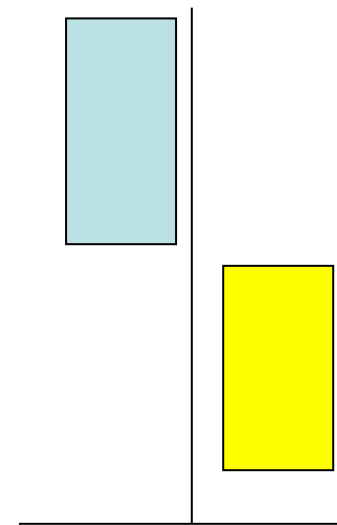
Concurrence d'accès

On parle **d'accès concurrent** lorsque plusieurs transactions (utilisateurs) accèdent en même temps à la même donnée dans la base de données.

Deux transactions T1 et T2 *interfèrent* : si le résultat de l'exécution en parallèle (*entrelacée*) de T1 et T2 est différent du résultat de l'exécution en série de T1 et T2 (T1 puis T2 ou T2 puis T1).



Exécutions entrelacées



Exécutions en série

Exécution de Transactions

Une exécution en parallèle de n transactions est sérialisable si et seulement si elle est équivalente à une exécution en série (les unes après les autres) de ces n transactions.

Une exécution est sérialisable si et seulement si son graphe de sérialisabilité ne présente aucun cycle. (cf. TD)

Concurrence d'accès

Le **but de la gestion des accès concurrents** est de s'assurer que le résultat de l'exécution en parallèle (*entrelacée*) de T1 et T2 est identique que le résultat de l'exécution en série de T1 et T2 (T1 puis T2 ou T2 puis T1).

Le contrôle de concurrence permet de s'assurer que l'exécution simultanée des transactions produit le même résultat que l'exécution séquentielle des mêmes transactions.

Concurrence d'accès – phénomène de lecture

Plusieurs phénomènes de lectures peuvent survenir lorsque des transactions interfèrent entre elle.

- Perte de mise à jour
- Lecture impropre (ou lecture de données incohérentes)
- Lecture non reproductible

Concurrence d'accès – phénomène de lecture

Perte de mise à jour :

T1	T2	BD
		X=20
READ X		
	READ X	
X=X*2		
WRITE X		X=40
	X=X/2	
	WRITE X	X=10

La mise à jour de T1 est perdue !

Concurrence d'accès – phénomène de lecture

Lecture impropre (dirty read): lecture de données non validées

T1	T2	BD
		$X+Y = 10$
		$X=2 ; Y=8$
READ X; READ Y		
$X=X+1$		$X=3 ; Y=8$
	READ X; READ Y	
$Y=Y-1$		$X=3; Y=7$
	$X=X-1 ; Y=Y+1$	$X=2 ; Y=9$

Les valeurs de X et Y lues par T2 sont impropres → incohérentes car la contrainte $X+Y=10$ n'est plus vérifiée.

Concurrence d'accès – phénomène de lecture

Non reproductibilité des lectures (non-repeatable read): lecture de la même ligne dans la même transaction donne des résultats différents

T1	T2	BD
		X=20
READ X		
	READ X	
	X=X*2	40
READ X		

Deux lectures d'une même donnée dans une même transaction (ici T1) conduit à des valeurs différentes.

Concurrence d'accès – phénomène de lecture

Lecture fantôme : une ligne nouvelle insérée est visible par une transaction alors qu'elle ne l'était pas avant (*Cas particulier de la Non reproductibilité des lectures*)

T1	T2	BD
		X
COUNT(*) FROM X		
	INSERT INTO X	
COUNT(*) FROM X		

Deux lectures avec la même condition retournent un nombre de lignes différent !

Concurrence d'accès – niveau d'isolation

Les phénomènes de lecture dépendent du niveau d'isolation.

4 niveaux d'isolation (Norme ANSI SQL92)

- **Lecture non commitée (read uncommitted)** : on peut accéder aux données non validées.
- **Lecture commités (read committed)** : par défaut sur Oracle. On ne lit que les données qui ont été validées.
- **Lecture reproductible (repeatable read)** : garantie que si une ligne est lue deux fois dans la même transaction, on obtiendra le même résultat.
- **Sérialisation (serialisable)** : la transaction s'exerce comme si elle était la seule sur la base.

Concurrence d'accès – niveau d'isolation

Tableau comparatif :

Niveau d'isolation	Phénomène de lecture		
	Lecture impropre	Lecture non reproductible	Lecture fantôme
Lecture non committée	OUI	OUI	OUI
Lecture committée(D)	NON	OUI	OUI
Lecture reproductible	NON	NON	OUI
Sérialisation	NON	NON	NON

Concurrence d'accès – résolution des interférences

Pour résoudre les problèmes dus à la concurrence d'accès, le mécanisme de verrouillage est la technique classique utilisée.

- Avant de lire ou écrire une donnée, la transaction demande un verrou sur la donnée pour empêcher les autres transactions de la modifier ou la lire.
- Si lors de la demande du verrou, une autre transaction a déjà verrouillé la donnée, la demandeuse de verrou est mise en attente jusqu'à ce que la donnée soit libérée.

Principe du verrouillage

Principe : Une transaction veut se réserver l'usage (exclusif ou partagé) d'une donnée aussi longtemps que nécessaire.

→ Elle va chercher à poser un verrou sur cette donnée.

On peut distinguer deux types de verrous:

- **LockS (A)** : verrou partagé en lecture sur A (avant lecture de A : *select*)
- **LockX (A)** : verrou exclusif sur A (avant écriture sur A: *insert, update, delete, ...*).

On suppose aussi que chaque transaction demande à poser un verrou correspondant avant de faire l'action et ne relâche celui-ci (par *UnLock*) qu'à la fin de la transaction.

Principe du verrouillage

Perte de mise à jour :

T1 (mise à jour →verrou exclusif)	T2 (mise à jour → verrou exclusif)	BD
		Y=20
LockX(Y)		
READ Y		
	LockX(Y)	
	En attente de T1	
Y=Y*2		
WRITE X; COMMIT		Y=40
	READ Y	
	Y=Y/2	
	WRITE Y ; COMMIT	Y=20

La mise à jour de T1 n'est pas perdue !

Le COMMIT libère de façon implicite le verrou.

Principe du verrouillage – Pessimiste (SQL Server)

Non reproductibilité des lectures (non-repeatable read):

T1 (lecture → verrou partagé)	T2 (mise à jour →verrou exclusif)	BD
		Y=20
LockS(Y)		
READ(Y)		Y=20
	LockX(Y)	
	En attente de T1	
READ(Y)		Y=20
COMMIT	READ(Y)	Y=20
	...	

Les deux lectures de T1 produisent le même résultat.

→ Le verrou partagé en lecture n'est pas compatible avec le verrou exclusif.

Principe du verrouillage – Multiversionning (Oracle)

ORACLE : les lecteurs ne pose jamais de VERROU

Non reproductibilité des lectures (non-repeatable read):

T1 (lecture → verrou partagé)	T2 (mise à jour →verrou exclusif)	BD
		Y=20
LockS(Y)		
READ(Y)		Y=20
	LockX(Y)	
	COMMIT	
READ(Y)		Y modifié
COMMIT		
	...	

Les deux lectures de T1 NE produisent PAS le même résultat.

Principe du verrouillage

Le verrou partagé : une transaction peut lire un objet mais non l'écrire. C'est un verrou en lecture. Aucune transaction ne peut mettre à jour la donnée mais des transactions concurrentes peuvent l'accéder en lecture.

Le mode exclusif : une transaction peut lire et écrire l'objet. C'est un verrou en écriture. Seulement la transaction T peut mettre à jour la donnée et tout autre accès est interdit.

Le mode partagé est incompatible avec le mode exclusif.

Incompatibilité → Attente de la transaction.

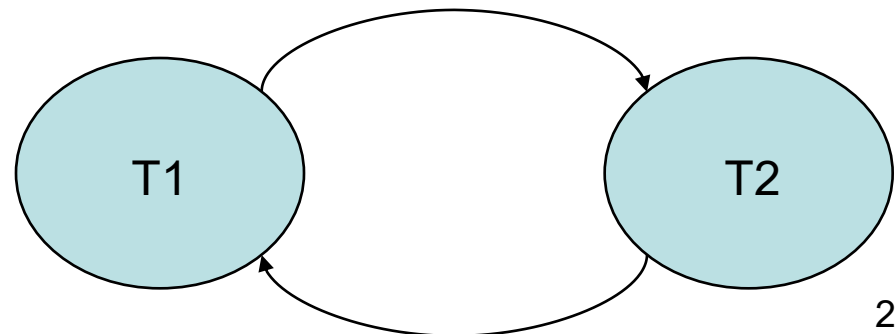
Demandé	P	E
Posé		
P		
E		

Problème du verrouillage : Interblocage

L'interblocage survient lorsqu'une transaction T_i demande le verrou sur une ressource verrouillée par T_j qui elle-même est en attente d'une ressource verrouillée par T_i ; on parle de situation de **deadlock**.

T1	T2
LockX(A)	
	LockX(B)
LockX(B) → en attente de T2	
	LockX(A) → en attente de T1

Graphe d'attente :



Interblocage – résolution

Deux approches :

1. Prévention :

Toutes les ressources utilisées par les transactions sont verrouillées au départ → certaines transactions ne se lanceront probablement jamais → peu utilisé aujourd'hui

2. Détection :

le graphe d'attente est inspecté à des intervalles réguliers pour détecter des interblocage. Si interblocage → une transaction ou plusieurs transactions bloquée(s) est défaite et relancé(s) plus tard.

Si le temps d'attente d'une transaction dépasse un seuil, cette dernière est défaite et relancée.

Niveaux de verrouillage

Afin de mieux gérer le verrouillage des ressources, le verrouillage se fait à plusieurs niveaux : on parle de **granularité de verrouillage**;

Ex : Il n'est pas nécessaire de verrouiller une relation pour modifier une ligne.

Au niveau « logique » : *base toute entière, relation, un groupe de tuples, un tuple, un champs d'un tuple....*

Au niveau « physique » : fichier, blocs ou groupes de blocs.

On peut tirer profit de cette **hiérarchie** pour affiner le mécanisme de verrouillage et le rendre plus efficace.

Verrouillage hiérarchisé

Verrou d'intention : verrous placés sur certains objets pour signaler que des objets plus petits (plus bas dans la hiérarchie) sont verrouillés et qui autorisent certaines formes d'accès aux autres objets plus petits.

On passe d'un système à deux verrous S (*Shared* = *partagé*) et X (*eXclusif*) à un mécanisme utilisant 5 types de verrous différents : **X, S, IX, IS, SIX**

Verrouillage hiérarchisé

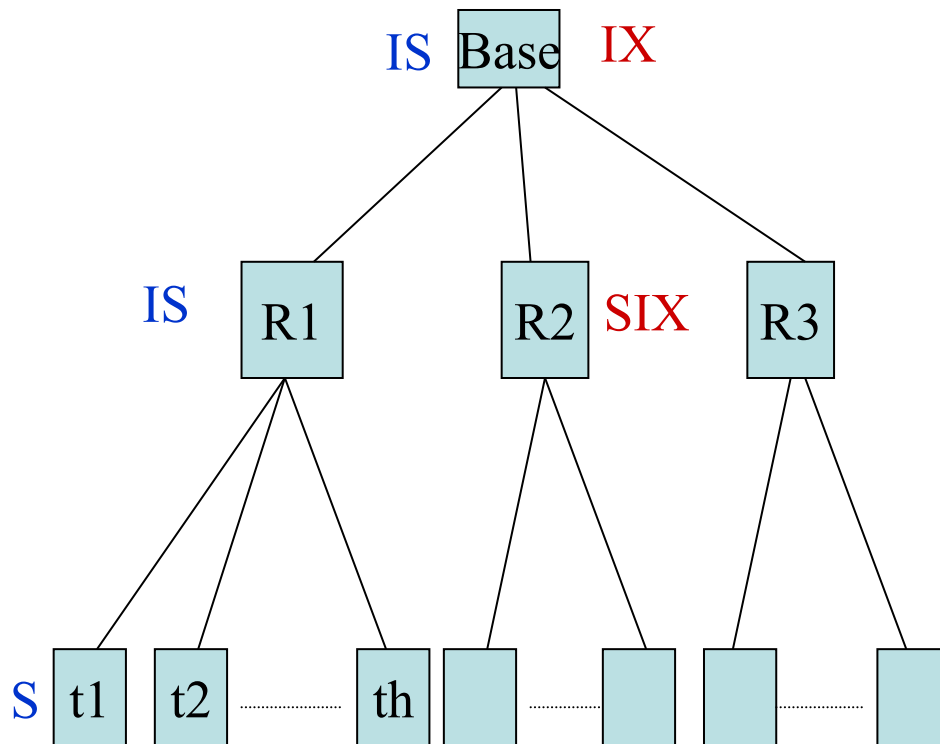
- X (*eXclusive*) : Verrouillage exclusif (écriture), verrouillage X implicite pour tous les fils.
 - S (*Shared*) : Verrouillage partagé (accès concurrent en lecture autorisé), verrouillage S implicite pour tous les fils.
 - IS (*Intention Shared*) : Verrouillage d'intention partagé, permet le verrouillage en mode S ou IS par la suite sur les fils.
 - IX (*Intention eXclusive*) : Verrouillage d'intention exclusif, permet le verrouillage en mode IX ou X sur ce nœud ou sur ses fils par la suite.
 - SIX (*S+IX*) : Verrou partagé et intention exclusif, de même que IX mais avec un verrouillage implicite des fils en mode S.
- A utiliser lorsque l'on souhaite lire un grand nombre de fils (évite de verrouiller explicitement chacun).

Verrouillage hiérarchisé

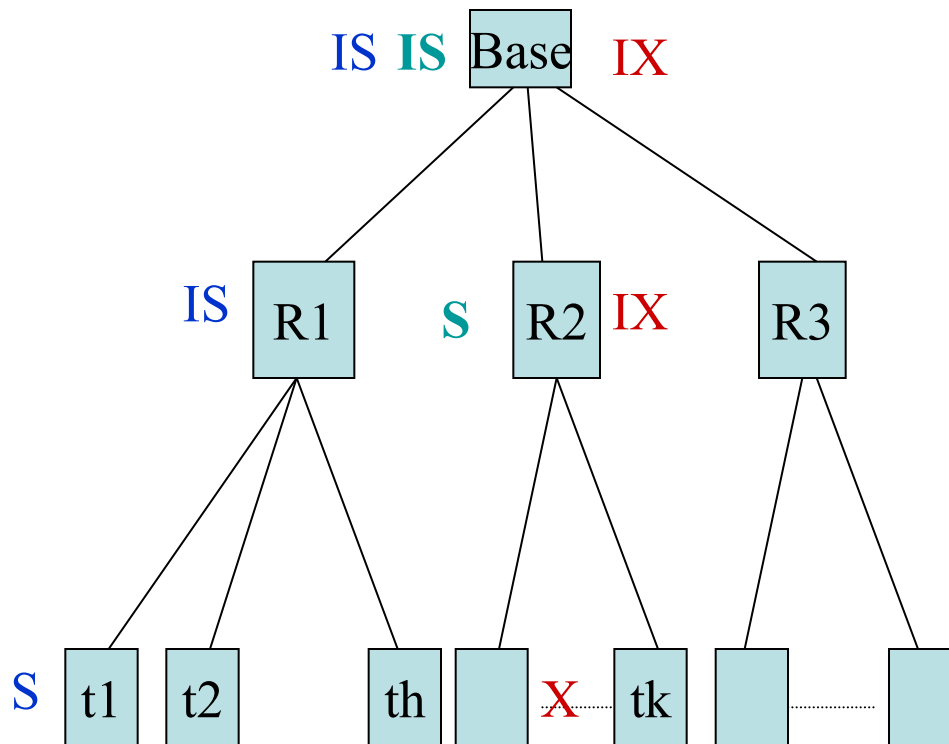
Transaction T: lire le tuple t1.

Transaction S: Lire tous les tuples de R2 et modification éventuelle de certains d'entre eux

Les deux transactions sont compatibles.



Pose de verrous hiérarchiques



Transaction T : lire le tuple t1.

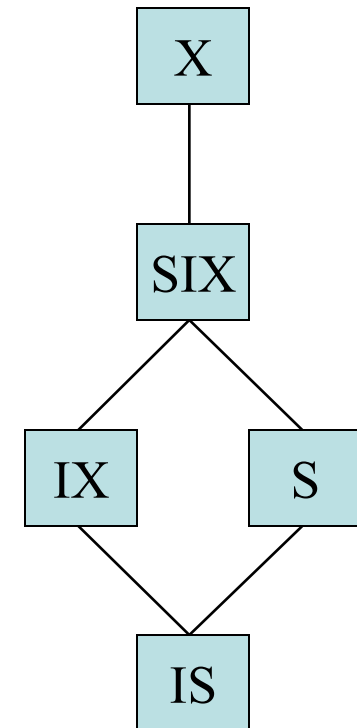
Transaction S : Ecrire sur le tuple tk

Transaction K : lire la relation R2.

Transaction K et transaction S sont incompatibles au niveau de la relation R2.

Verrouillage hiérarchisé

Demandé Posé	X	SIX	IX	S	IS
X	N	N	N	N	N
SIX	N	N	N	N	O
IX	N	N	O	N	O
S	N	N	N	O	O
IS	N	O	O	O	O



Verrouillage hiérarchisé

Protocole :

- Les verrous sont posés du haut vers le bas de la hiérarchie.
- Ils sont relâchés du bas vers le haut.
- Avant de pouvoir poser un verrou S ou IS sur un objet, une transaction doit avoir posé un verrou IS (ou supérieur) sur le nœud parent.
- Avant de pouvoir poser un verrou X, IX ou SIX sur un objet, une transaction doit avoir posé un verrou IX (ou supérieur) sur le nœud parent.
- Tout verrou X (resp. S et SIX) posé sur un objet implique un verrouillage implicite de type X (resp. de type S) de tous les enfants de cet objet.

Utilisateur bloqué

Scénario :

Utilisateur Scott crée la table emp.

Scott donne des privilèges à l'utilisateur David :

grant select, update, insert, delete on emp to david;

David pose un verrou exclusif sur la table emp qui appartient à Scott : (update)

Scott veut aussi poser un verrou exclusif sur sa table emp : (update) ... Il est mis en attente ; en attente que le verrou de David soit libéré.

David est parti en réunion pour au **moins 4 heures**

Que faire pour Scott ?

Utilisateur bloqué par un verrou

Le DBA doit :

1. identifier la session et le verrou posé par David
2. Identifier la session et le verrou posé par Scott
3. Supprimer la session de David → verrou sera libéré et Scott pourra travailler.

Gestion de la concurrence d'accès

Mécanisme de verrouillage → Interblocage mais est la plus utilisé.

Autre méthode pour gérer la concurrence **d'accès : le mécanisme d'estampillage** → Moins utilisé mais évite des interblocage

Le mécanisme d'estampillage

Il se base sur l'ordonnancement des transactions.

A chaque transaction, un numéro unique (appelé estampille) lui est attribué lui permettant d'être ordonnancer par rapport au autre transaction.

Soient T1 et T2 deux transactions concurrentes

$E(T1)$ et $E(T2)$ les estampilles des deux transactions, $E(T1) > E(T2)$

L'ordre de soumission des transactions est alors T1 puis T2

Le mécanisme d'estampillage

Soient T1 et T2 deux transactions concurrentes

$E(T1)$ et $E(T2)$ les estampilles des deux transactions, $E(T1) > E(T2)$

L'ordre de soumission des transactions est alors T1 puis T2

T1 lit un objet avant que T2 ne le modifie → OK

T1 lit un objet après une modification de T2 → KO, T1 est défait et resoumis au système avec un nouvel estampille.

Les transactions les plus récentes sont privilégiées dans cet algo.

Différents algos disponibles.

NIVEAUX D'ISOLATION

READ UNCOMMITTED : permet à une autre transaction de lire les données qui ont été chargées mais pas encore validées.

! Ce niveau d'isolation n'est pas disponible sur Oracle et pas recommandé.

READ COMMITTED : c'est le niveau d'isolation par défaut sur Oracle. Il assure que chaque requête dans une transaction lit seulement les données validées.

NIVEAUX D'ISOLATION

SERIALIZABLE : ce niveau d'isolation permet à une transaction de voir uniquement les modifications qui ont été validées au début de la transaction ainsi que celles effectuées dans la transaction elle-même via les instructions INSERT, DELETE et UPDATE

READ-ONLY : ce niveau d'isolation permet à une transaction de voir uniquement les modifications qui ont été validées au début de la transaction et n'autorise pas les instruction INSERT, DELETE ou UPDATE.

Fin