SGBD - 2e

Chapitre 5 - Transactions et accès concurrents Langage de Contrôle des données (LCD)

Daniel Schreurs

14 février 2022

Haute École de Province de Liège

Table des matières du chapitre i

Introduction

Table des matières de la section : Introduction i

Introduction : Définition

Problèmes qui peuvent survenir lors de l'exploitation d'une base de données :

- Deux utilisateurs tentent de modifier la même donnée au même moment
- Une panne se produit pendant l'exécution d'une transaction

Introduction : Définition

Ces deux problèmes sont résolus grâce au concept de transaction.

Important

Une transaction est une unité de traitement cohérent et sûr.

Toute requête doit être exécutée dans le contexte d'une transaction.

Notions de base

Table des matières de la section : Notions de base i

Table des matières de la section : Notions de base ii

Notions de base : Définition

Important

Une base de données est dans un état cohérent si les valeurs contenues dans la base vérifient toutes les contraintes d'intégrité définies sur la base.

Les transactions de mise à jour font passer la base d'un état à un autre. Une **transaction est cohérente** si elle fait passer la base d'un état cohérent à un autre état cohérent. Cependant, lors des mises à jour par des transactions cohérentes, la base peut passer par des **états intermédiaires incohérents**.

Notions de base : Exemple

Exemple

```
1 -- Etat initial  : Base cohérente
2 UPDATE Compte
3 SET solde = solde - S
4 WHERE num_compte = A;
5 -- Etat intermédiaire  : Base incohérente
6 UPDATE Compte
7 SET solde = solde + S
8 WHERE num_compte = B;
9 -- Etat final  : Base cohérente
```

1

1. Si on a une panne entre les 2 updates, on ne veut pas perdre le montant S. Et donc les 2 opérations doivent obligatoirement être réalisées toutes les deux ou aucune.

Notions de base : Définition

Une transaction peut-être constituée d'un ensemble d'actions. Les actions sont des unités de traitement indivisibles. Une des grandes propriétés des transactions est l'**atomicité**.

Important

Le terme **atomicité** signifie qu'une transaction doit être traitée comme une seule opération. Autrement dit, le gestionnaire des transactions doit assurer que toutes les actions de la transaction sont exécutées, ou bien qu'aucune ne l'est.

Notions de base : Définition

Si l'exécution d'une transaction est interrompue à cause d'une panne, le SGBD doit être capable de décider des actions à entreprendre après la réparation. Il peut :

- **Compléter** la transaction en exécutant les actions restantes
- **Défaire** les actions qui avaient été exécutées avant la panne

Notions de base : Définition

Pour permettre au SGBD d'assurer l'atomicité des transactions, nous devons définir les actions qui indiquent le début et la fin d'une transaction. Le début d'une transaction indique au SGBD de démarrer une nouvelle transaction. Une transaction peut se terminer par l'action **valider ou annuler** dont les effets sur la base sont évidemment opposés.

Notions de base : Définition

Trois issues sont possibles pour une transaction :

- Une vie sans histoire ;
- Un arrêt interne ;
- Un arrêt externe.

Notions de base : Une vie sans histoire

La transaction T effectue successivement toutes ses actions lire et/ou écrire et exécute l'action valider. T **a atteint son point de confirmation.** Il s'agit d'un point de non-retour à partir duquel toutes les modifications faites sur la base par T sont considérées comme irrémédiablement faites.

Notions de base : Une vie sans histoire

La **durabilité** rend permanents les changements. Considérant aussi les pannes. La commande SQL qui valide et termine une transaction est : `COMMIT`

Notions de base : Un arrêt interne

Exemple : lors de son exécution, une transaction **détecte une erreur** de conversion de type de données ou une violation de contrainte d'intégrité. Dans ce cas, la transaction arrête son exécution en émettant l'**action annuler**. Afin d'assurer la propriété d'atomicité, le SGBD doit effacer toute trace du passage de la transaction annulée. La commande SQL qui défait et termine une transaction est :

```
ROLLBACK
```

Notions de base : Un arrêt externe

Un événement extérieur vient arrêter définitivement l'exécution de la transaction. Exemples :

- Action de l'utilisateur qui décide d'interrompre la transaction
- Action du SGBD qui a détecté un inter-blocage et qui, pour résoudre le problème, a choisi notre transaction comme victime
- Panne

Le SGBD doit effacer toute trace du passage de la transaction. Si l'arrêt a été provoqué par une panne, effacer les traces du passage de la transaction ne peut être réalisé qu'après le redémarrage du système : mécanisme (ou processus) de **reprise après panne** ou **recovery**.

Notions de base : Un arrêt externe

Le rôle du mécanisme de reprise après panne est double :

- Il doit assurer que les effets d'une transaction validée apparaîtront dans la base après la reprise
- Il doit assurer que les effets d'une transaction annulée n'apparaîtront pas dans la base après la panne

Transactions concurrentes

Table des matières de la section : Transactions concurrentes
i

Table des matières de la section : Transactions concurrentes ii

Transactions concurrentes : Définition

Si des transactions cohérentes sont exécutées séquentiellement, elles conduisent toujours à un état final cohérent. Pour des raisons d'efficacité, il est nécessaire d'autoriser l'exécution concurrente des transactions.

Important

On dit que deux transactions sont concurrentes si elles accèdent en même temps aux mêmes données.

Il faut gérer ces exécutions concurrentes de manière à ce qu'elles ne conduisent pas à une base incohérente

Transactions concurrentes : Définition

Les différentes transactions sont constituées d'actions lire et écrire et, si aucun mécanisme de contrôle de l'exécution n'est présent dans le SGBD, nous pouvons avoir les **3 types d'anomalies** :

- Pertes de mise à jour
- Lecture impropre
- Lecture non reproductible

Transactions concurrentes : Pertes de mise à jour

Soit les 2 transactions suivantes :

T1 : *Lire*(x)	T2 : *Lire*(x)
T1 : $x \longleftarrow x + 10$	T2 : $x \longleftarrow x + 20$
T1 : *écrire*(x)	T2 : *écrire*(x)

Si exécutées séquentiellement, si x vaut 50 au départ, après l'exécution des deux transactions, x vaudra 80.

Transactions concurrentes : Pertes de mise à jour

Si exécutées de manière concurrente, nous pourrions avoir la séquence suivante :

T1 : *Lire*(x)	(T1 lit 50 pour x)
T1 : $x \longleftarrow x + 10$	
T2 : *Lire*(x)	(T2 lit 50 pour x)
T1 : *écrire*(x)	(dans la base x vaut 60)
T2 : $x \longleftarrow x + 20$	
T2 : *écrire*(x)	(dans la base x vaut 70)

Dans cet exemple, la mise à jour effectuée par $T1$ a été écrasée par celle faite par $T2$.

Transactions concurrentes : Lecture impropre

T1	T2
DÉBUT TRANSACTION	DÉBUT TRANSACTiON
UPDATE comptes	
SET solde = 25000	
WHERE	
num_compte	
= ’007’ ;	
	SELECT solde
	FROM comptes
	WHERE
	num_compte
	= ’007’ ;
ROLLBACK ;	

Table 1 – Anomalie 2 : Lecture impropre

Transactions concurrentes : Lecture impropre

- À cause de la propriété d'atomicité, tout doit se passer comme si T1 n'avait jamais fait la modification dans la table comptes.
- La valeur lue par T2 est incorrecte, elle est impropre : T2 lit des données non confirmées ! Les données sont salies par T1.
- Effet domino : l'annulation de la transaction T1 doit entraîner l'annulation de toutes les transactions ayant lu des résultats intermédiaires de T1.

Transactions concurrentes : Lecture impropre

T1	T2
DÉBUT TRANSACTION	DÉBUT TRANSACTiON
UPDATE comptes	
SET solde = 25000	
WHERE num_compte= ’007’ ;	
	SELECT SUM(solde)
	FROM comptes
	WHERE
	num_compte IN (’007’, ’163’) ;
UPDATE comptes	
SET solde = solde + 200	
WHERE	
Num_compte = ’163’ ;	
COMMIT ;	

Table 2 – Anomalie 2 : Lecture impropre sans annulation

Transactions concurrentes : Lecture impropre

T2 puise ses valeurs dans la base alors qu'elle est dans un état intermédiaire incohérent.

Transactions concurrentes : Lecture impropre

- Anomalie 2 : Lecture impropre : **lecture ou référence fantôme**
- Une lecture ou référence fantôme **peut découler d'une lecture impropre.**

Transactions concurrentes : Lecture impropre

T1	T2
DÉBUT TRANSACTION	DÉBUT TRANSACTiON
SELECT *	
FROM eleves	
WHERE annee = 2 ;	
	INSERT INTO eleves
	VALUES (11, 'Bloche', ..., 2) ;
	COMMIT
SELECT *	
FROM eleves	
WHERE annee = 2 ;	

Table 3 – Anomalie 2 : Lecture impropre : lecture ou référence fantôme

Transactions concurrentes : Lecture impropre

La transaction **T1 ne retrouve pas le même ensemble de tuples lors de la deuxième lecture** : un tuple supplémentaire apparaît. Il s'agit d'un **tuple fantôme**.

Transactions concurrentes : Lecture impropre

T1	T2
DEBUT TRANSACTION	DEBUT TRANSACTiON
SELECT points	
FROM resultats WHERE	
num_cours = 5 AND num_eleve = 7 ;	
	UPDATE resultats
	SET points = points * 1.10
	WHERE num_cours = 5
	AND num_eleve = 7 ;
	COMMIT
SELECT points	
FROM resultats WHERE	
num_cours = 5 AND num_eleve = 7 ;	
COMMIT ;	

Table 4 – Anomalie 2 : Lecture non reproductible

Transactions concurrentes : Lecture impropre

Remarques :

- **Lecture non reproductible** : On ré-exécute une requête et le résultat ne donne pas les mêmes valeurs que lors de la première exécution.
- **Référence fantôme** : On ré-exécute une même requête et le résultat donne une ligne supplémentaire par rapport à la première exécution

Dans les deux cas, les données ont été modifiées par une autre transaction !

Transactions concurrentes : ACID

Jusqu'à présent, nous avons vu 3 propriétés des transactions :

- A comme Atomicité
- C comme Cohérence
- ? comme ?
- D comme Durabilité

Nous allons pouvoir ajouter **I** comme **ISOLATION**

Transactions concurrentes : isolation

Important

L'isolation est la propriété des transactions qui exige que chaque transaction perçoive à tout instant la base dans un état cohérent. En d'autres termes, une transaction en cours d'exécution ne peut pas dévoiler ses effets aux autres transactions concurrentes avant d'atteindre son point de confirmation.

Transactions concurrentes : isolation

La norme SQL2 distingue deux types de verrous

- Verrous courts : niveau instruction
- Verrous long : niveau transaction

Permet de résoudre :

- Perte de mise à jour
- Lecture impropre
- Lecture non reproductible

Transactions concurrentes : isolation

Important

4 niveaux d'isolation standardisés par SQL2 (Set Transaction Isolation Level) :

- Degré 0 : READ UNCOMMITTED
- Degré 1 : READ COMMITTED
- Degré 2 : REPEATABLE READ
- Degré 3 : SERIALIZABLE

Transactions concurrentes : isolation

Isolation degré 0 : READ UNCOMMITTED

- Verrou court exclusif en écriture
- Verrou exclusif libéré après l'écriture
- Pas de pose de verrous lors des lectures

Permet d'éviter les pertes de mises à jour.

Transactions concurrentes : isolation

Isolation degré 0 : **READ UNCOMMITTED**
Verrou court exclusif écriture

Transaction 1	Transaction 2
Verrou court exclusif écriture	
	pas possibilité de MAJ du nr 1234,
UPDATE employes	tant que l'instruction UPD transaction 1
SET bareme = bareme +10	n'est pas terminée. La transaction 1 a le
WHERE numsecu = '1234' ;	verrou exclusif surle 1234 pendant la
	durée de l'UPD.

Table 5 – Isolation degré 0 : READ UNCOMMITTED

Transactions concurrentes : isolation

Isolation degré 0 : **READ UNCOMMITTED**

- Pas de perte de mise à jour, verrou court exclusif en écriture
- Peut lire des données modifiées non validées : possibilité de lecture impropre
- Les données peuvent être modifiées par d'autres transactions entre deux instructions de la transaction active : possibilité de données non reproductibles
- D'autres transactions peuvent insérer des nouvelles lignes : possibilité de référence fantôme

Transactions concurrentes : isolation

Isolation degré 1 : **READ COMMITTED**
Une transaction de degré 1 satisfait le degré 0 et ne fait pas de lecture sale :

- Verrou court partagé (S) lors des lectures
- Verrou long exclusif (X) lors des écritures
- Verrou exclusif libéré en fin de transaction

Permet de résoudre le problème des pertes de mises à jour et des lectures impropres

Transactions concurrentes : isolation

Isolation degré 1 : **READ COMMITTED**

Transaction 1	Transaction 2
Verrou long exclusif en écriture UPDATE employes SET bareme = bareme +10 WHERE numsecu = ’1234’ ; COMMIT ;	Pas de MAJ du nr 1234 tant que COMMIT transaction 1 pas exécute Pas possibilité de lire la donnée modifiée

Table 6 – Isolation degré 1 : READ COMMITTED

pas de lecture impropre

Transactions concurrentes : isolation

Isolation degré 1 : **READ COMMITTED**

+ **Pas de perte de mise à jour**, verrou long exclusif en écriture
+ Pas de possibilité de lire les données modifiées non validées : **pas de lecture impropre**
- Les données peuvent être modifiées par d'autres transactions entre deux instructions de la transaction active : **possibilité de données non reproductibles**
- D'autres transactions peuvent insérer des nouvelles lignes : **possibilité de référence fantôme**

Transactions concurrentes : isolation

Isolation degré 2 : **REPEATABLE READ**
Une transaction de degré 2 satisfait le degré 1 et ne fait pas de lecture non reproductible :

- Verrou long partagé (S) lors des lectures
- Verrou libéré en fin de transaction

Permet de résoudre les problèmes

- des pertes de mises à jour
- les lectures impropres
- les lectures non reproductibles.

Transactions concurrentes : isolation

Isolation degré 2 : **REPEATABLE READ**

Transaction 1	Transaction 2
Verrou long partagé en lecture	Pas de modification des données
SELECT numsecu	lues par transaction active
FROM employes	car demande d'un verrou
WHERE codepostal LIKE '4%' ;	exclusif alors que la transaction 1
SELECT numsecu	a un verrou partagé sur
FROM employes	les données lues
WHERE codepostal LIKE '4%' ;	
COMMIT ;	

Table 7 – Isolation degré 2 : REPEATABLE READ

pas de lecture non reproductible.

Transactions concurrentes : isolation

Isolation degré 2 : **REPEATABLE READ**

+ Pas de perte de mise à jour, verrou long exclusif en écriture
+ Pas de possibilité de lire les données modifiées non validées : pas de lecture impropre
+ Pas de possibilité de modifier les données lues par la transaction active tant que celle-ci n'est pas terminée, pas de données non reproductibles
- D'autres transactions peuvent insérer des nouvelles lignes : possibilité de référence fantôme

Transactions concurrentes : isolation

Isolation degré 3 : **SERIALIZABLE**
Une transaction de degré 3 satisfait le degré 2 et ne fait pas de requête non reproductible :

- Verrou long exclusif lors des lectures
- Verrou libéré en fin de transaction

Permet de résoudre les problèmes

- De pertes de mises à jour ;
- Les lectures impropres ;
- Les lectures non reproductibles ;
- Et les lectures fantômes.

Transactions concurrentes : isolation

Isolation degré 3 : **SERIALIZABLE**

+ **Pas de perte de mise à jour**, verrou long exclusif en écriture
+ Pas de possibilité de lire les données modifiées non validées : **pas de lecture impropre**
+ Pas de possibilité de modifier les données lues par la transaction active tant que celle-ci n'est pas terminée, **pas de données non reproductibles**
+ Pas de possibilité d'insertion de nouvelles lignes dans l'objet lu par la transaction active tant que celle-ci n'est pas terminée : **pas de référence fantôme**

Transactions concurrentes : isolation

Isolation degré 3 : **SERIALIZABLE**
Dans la littérature, on dit que les transactions de degré 3 sont du niveau d'isolation sérialisable ou plus simplement sérialisables [2]

- exécution séquentielle
- donc cohérente

2. Serializable est le niveau recommandé par la norme. Cependant, ce niveau est fort restrictif : individuellement, chacun son tour !

Transactions concurrentes : Synthèse

Degré	Perte de mise à jour	Lecture impropre	Données non reprod	Référence fantôme
0 - Read Uncommitted V court excl en écriture Pas de V en lecture	NON	OUI	OUI	OUI
1 - Read Committed **V lg excl en écriture** **V court ptg en écriture**	**NON**	**NON**	**OUI**	**OUI**
2 - Repeatable Read V lg ptg en lecture	NON	NON	NON	OUI
3 - Serializable V lg excl en lecture	NON	NON	NON	NON

Important

Read Committed : niveau d'isolation par défaut d'Oracle

Spécifications de SQL2

Table des matières de la section : Spécifications de SQL2 i

Spécifications de SQL2 : Définitions

- **SQL-agent** : exécution d'un programme d'applications contenant une ou plusieurs requêtes SQL.
- **SQL-client** : l'agent démarre l'exécution sous le contrôle d'un client
- **SQL-connection** : pour pouvoir faire des accès à la base, l'agent doit forcer le client à établir une connexion avec un server
- **SQL-server** : a pour rôle d'effectuer les accès à la base de données
- **SQL-environnement** : le client et le serveur constituent les deux composantes d'un environnement

Spécifications de SQL2 : Définitions

- L'établissement d'une connexion entre le client et le serveur est réalisé au moyen de la commande **CONNECT**.
- Cette commande démarre également une session (**SQL-session**) au-dessus de la connexion.
- Quand la connexion est établie et la session initialisée, l'agent peut exécuter des transactions (**SQL-ransactions**) jusqu'à ce qu'il exécute l'instruction **DISCONNECT** qui arrête la connexion et met fin à la session.

Spécifications de SQL2 : Définitions

Important

- Une transaction est une unité logique de travail (suite atomique de commandes SQL)
- Deux transactions ne peuvent pas être imbriquées
- Une transaction est démarrée implicitement par un agent lorsqu'il exécute certaines commandes SQL appelées transaction-initiating statement. (on peut résumer en disant que toute commande du LDD et du LMD peut démarrer une transaction)
- Les transactions sont terminées explicitement par COMMIT ou ROLLBACK.

Spécifications de SQL2 : L'instruction SET TRANSACTION

L'instruction SET TRANSACTION est utilisée pour définir les caractéristiques de la prochaine transaction :

- Le mode d'accès (lecture seule (read only) ou lecture écriture (read write)),
- La taille de la zone de diagnostic (diagnotics area)

Le niveau d'isolation

```
(ISOLATION LEVEL
  READ UNCOMMITTED |
  READ COMMITTED |
  REPEATABLE READ |
  SERIALIZABLE )
```

3

3. Ne peut pas être exécutée s'il y a une transaction en cours. NE démarre PAS une transaction. Serializable : niveau par défaut

Spécifications de SQL2 : sérialisable

Pour la norme SQL2, toute transaction doit être sérialisable (serializable) [4]
L'exécution concurrente de transactions sérialisables doit produire le même effet que l'exécution séquentielle de ces transactions.
La norme définit 3 situations dans lesquelles la sérialisabilité peut être violée (aucune autre ne peut le faire) :

- Lecture impropre
- Lecture non reproductible
- Référence fantôme

4. La norme SQL2 précise également que si un SGDB offre un niveau d'isolation autre que sérialisable (le seul à être cohérent), il doit fournir des commandes de contrôle explicite de la concurrence (pose de verrous) afin qu'il soit possible de développer des applications cohérentes.

Études de cas dans Oracle 10

Table des matières de la section : Études de cas dans Oracle 10 i

Table des matières de la section : Études de cas dans Oracle 10 ii

Études de cas dans Oracle 10 : Terminaison

Dans Oracle, une transaction se termine quand :

- exécution d'un **COMMIT** ou d'un **ROLLBACK**
- **exécution d'une commande du LDD** : cela provoque d'abord la validation de la transaction en cours, l'exécution de la commande du LDD suivie immédiatement d'un nouveau COMMIT
- **déconnexion à Oracle** : la transaction courante est validée
- **fin anormale d'un processus utilisateur** : la transaction courante est annulée

Études de cas dans Oracle 10 : savepoint

Oracle implémente également

- La notion de point de sauvegarde intermédiaire (**savepoint**)[5].
- Un point de sauvegarde intermédiaire se définit par :
  ```
  SAVEPOINT point;
  ```
- L'annulation des opérations comprises entre le point courant et un point de sauvegarde intermédiaire donné se réalise par :
  ```
  ROLLBACK point;
  ```

5. Les savepoints permettent de diviser une transaction en plusieurs petites parties qu'il est possible d'annuler sans annuler toute la transaction.

Études de cas dans Oracle 10 : Comportements par défaut

- Par défaut, Oracle garantit la cohérence en lecture au niveau opération mais ne garantit pas une lecture cohérente au niveau transaction.
- Pour réaliser des lectures cohérentes au niveau transaction, il faut démarrer une transaction read only au moyen de la commande `set transaction read only` ou utiliser des verrous explicites.

Études de cas dans Oracle 10 : Comportements par défaut

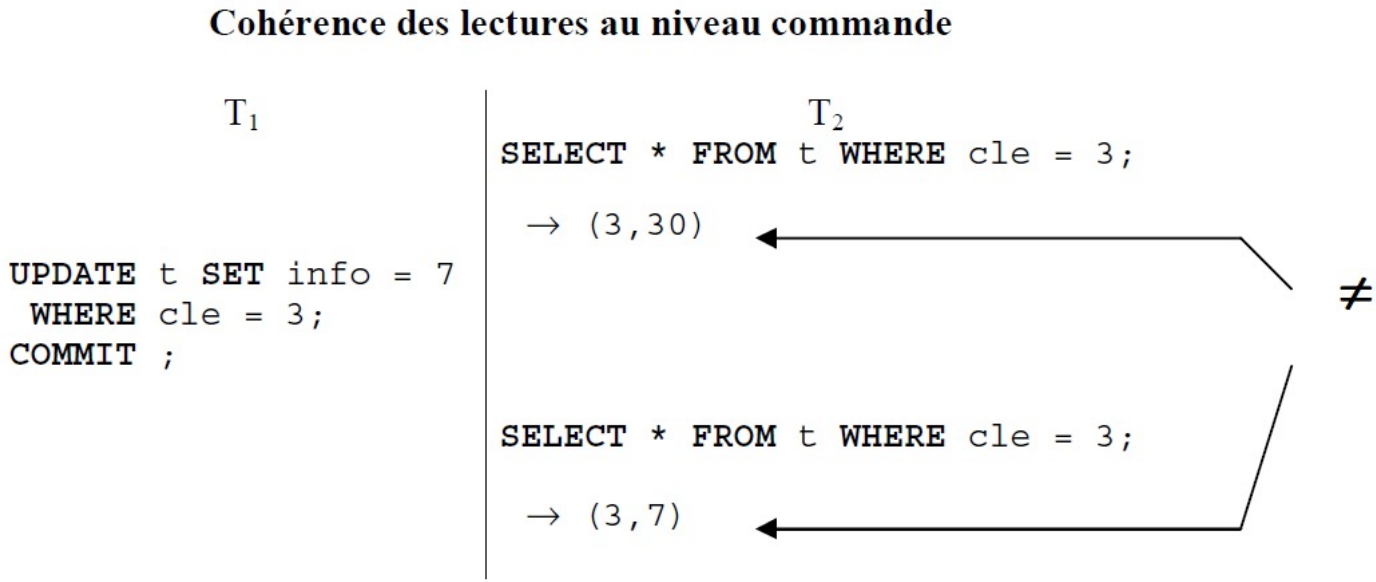

Figure 1 – Lecture non reproductible : niveau par défaut en Oracle.

Études de cas dans Oracle 10 : Comportements par défaut

En Oracle : **une lecture ne bloque pas une modification et une modification ne bloque pas une lecture.**

- Si une transaction T2 tente de lire un objet sur lequel une transaction T1 possède un accès en écriture, alors T2 a accès à une version antérieurement confirmée de l'objet ;
- Si une transaction T2 cherche à modifier un objet sur lequel une transaction T1 possède un accès en lecture, la transaction T2 obtient l'accès en écriture alors que T1 mémorise l'accès à sa version de l'objet (qui devient l'ancienne valeur dans le journal des images avant).

Études de cas dans Oracle 10 : Comportements par défaut

Comportements des accès concurrents par défaut

- Cette technique n'est utilisable que si le SGBD est capable de mémoriser une version antérieure confirmée.
- Rappel : par défaut, en Oracle, les lectures ne sont pas reproductibles (voir exemple ci-après).

Études de cas dans Oracle 10 : Comportements par défaut

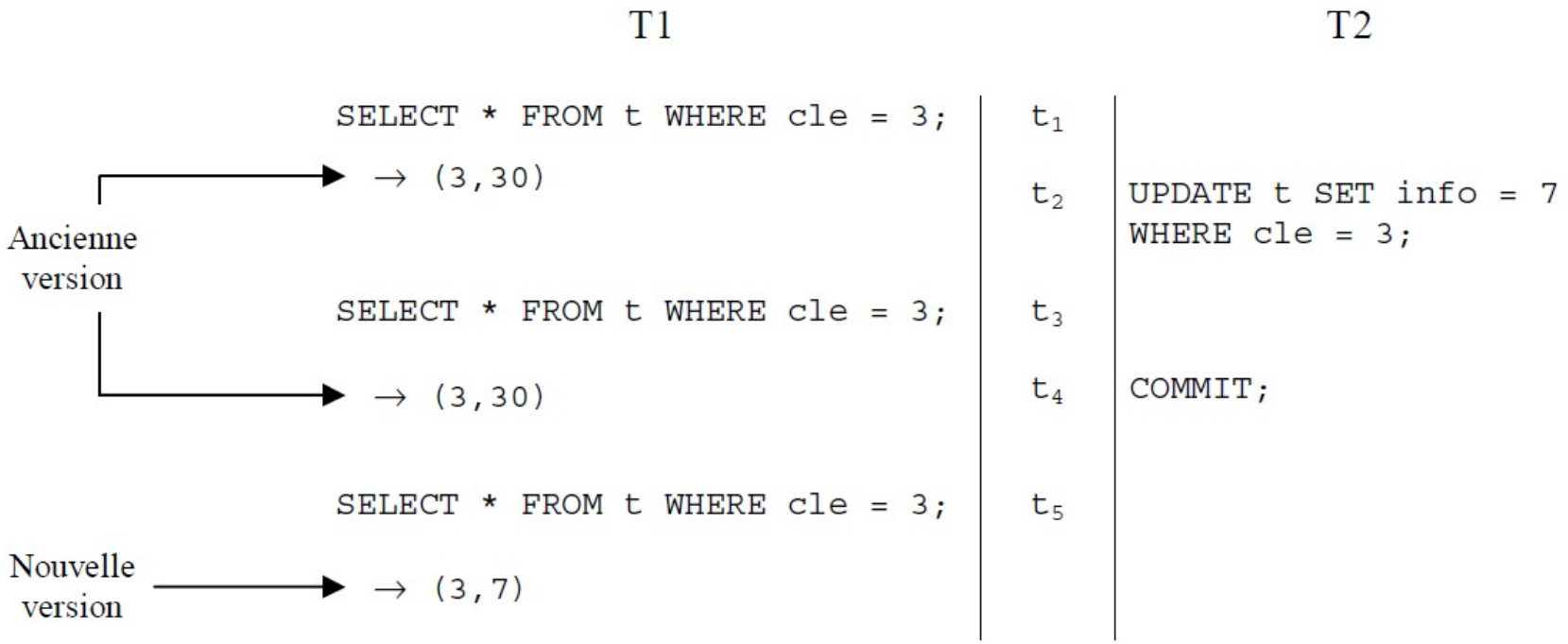

Figure 2 – Comportements des accès concurrents par défaut

Études de cas dans Oracle 10 : Comportements par défaut

Comportements des accès concurrents par défaut

En Oracle : **une écriture bloque une écriture.**

Études de cas dans Oracle 10 : Comportements par défaut

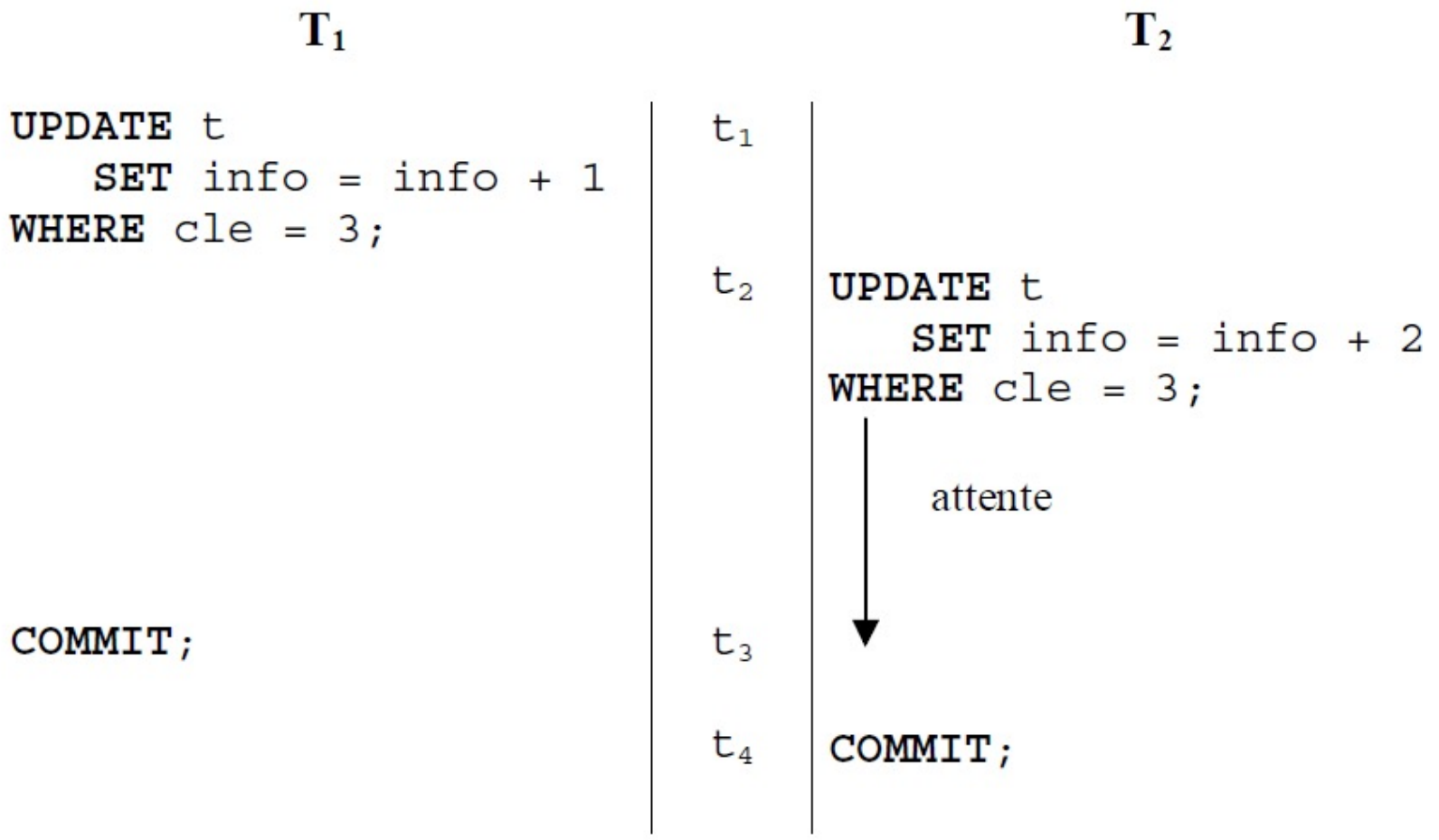

Figure 3 – Dès que l'on commence un update, on a un verrou long, jusqu'à la fin de la transaction : COMMIT

Études de cas dans Oracle 10 : Comportements par défaut

Comportements des accès concurrents par défaut
En résumé, dans Oracle, lorsque l'on travaille avec les options par défaut :

+ Il n'y a pas de lecture impropre
+ Il n'y a pas de pertes de mises à jour
- Il est possible de provoquer une référence fantôme
- Les lectures ne sont pas forcément reproductibles

Études de cas dans Oracle 10 : Comportements par défaut

Oracle et les niveaux d'isolation

- Oracle empêche toute lecture impropre.
- Le niveau `READ COMMITTED` est le niveau par défaut.
- Oracle ne supporte pas directement le niveau `REPEATABLE READ` (il ne le supporte qu'on mode `READ ONLY`)
- En écriture, ce niveau est inclus dans le niveau `SERIALIZABLE`.
- Le niveau d'isolation est spécifié au moyen de la clause `ISOLATION LEVEL` de la commande `SET TRANSACTION`

Études de cas dans Oracle 10 : Comportements par défaut

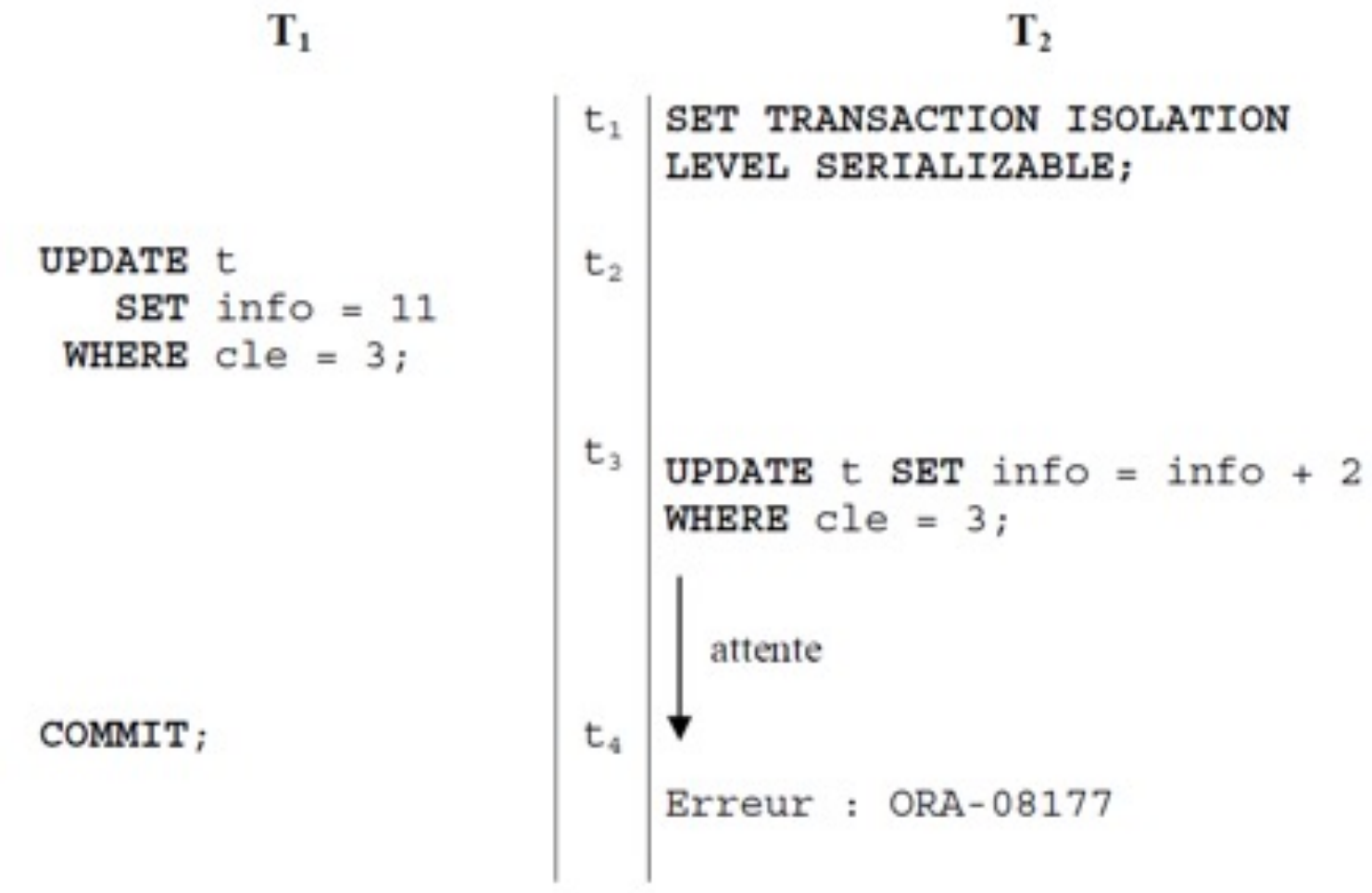

Figure 4 – Oracle et les niveaux d'isolation

Études de cas dans Oracle 10 : Comportements par défaut

- **t2**, T1 modifie le tuple 3, entraînant la pose d'un verrou en écriture sur ce tuple.
- **t3**, T2 veut, à son tour, modifier ce même tuple, T2 est placée en attente.
- **t4**, la transaction T1 libère les verrous et T2 est donc débloquée. Mais, à ce moment, T2 reçoit le code d'erreur ora-081177 indiquant qu'Oracle a détecté des données modifiées par une autre transaction depuis le début de la transaction sérialisable T2.

Études de cas dans Oracle 10 : Verrouillage explicite

- **t2**, T1 modifie le tuple 3, entraînant la pose d'un verrou en écriture sur ce tuple.
- **t3**, T2 veut, à son tour, modifier ce même tuple, T2 est placée en attente.
- **t4**, la transaction T1 libère les verrous et T2 est donc débloquée. Mais, à ce moment, T2 reçoit le code d'erreur ora-081177 indiquant qu'Oracle a détecté des données modifiées par une autre transaction depuis le début de la transaction sérialisable T2.

Études de cas dans Oracle 10 : Verrouillage explicite

Oracle permet au programmeur de placer des verrous explicitement.
Ceci se fait au moyen des commandes :

- `SELECT name FOR UPDATE [NOWAIT]`
- `LOCK TABLE`

Études de cas dans Oracle 10 : Verrouillage explicite

Verrouillage explicite dans Oracle

```
1 select *
2 from EMPLOYES
3 where upper(PRENOM) like 'JAMES'
4 FOR UPDATE;
```

Cette commande verrouille explicitement les tuples répondant à la condition de la clause `WHERE`. `verrou ROW SHARE` Les tuples sont verrouillés en vue d'être modifiés. Une autre transaction pourra lire ces tuples mais pas les modifier.

Études de cas dans Oracle 10 : Verrouillage explicite

Ici on ne souhaite pas attendre et donc risque d'erreur

```
1 SELECT *
2 FROM employes
3     FOR UPDATE NOWAIT;
```

Études de cas dans Oracle 10 : La double transaction

- La double transaction va être mise en place dans le cadre de la mise à jour de données de la base lorsque plusieurs utilisateurs sont susceptibles de modifier les mêmes tuples.
- Les programmes qui la mettent en œuvre utilisent deux transactions d'où son nom.
  - première transaction : recherche du tuple à modifier
  - deuxième transaction : mise à jour des données
- Entre les deux :
  - L'utilisateur courant peut faire des modifications (dans sa copie locale du tuple)
  - D'autres utilisateurs peuvent faire des mises à jour incompatibles

Études de cas dans Oracle 10 : La double transaction

Exemple : Un étudiant a changé d'adresse et va au service étudiant pour le faire savoir.

- Lecture de la donnée à modifier (recherche de l'étudiant à modifier)
- Encodage de la nouvelle valeur
- Mise à jour de cette valeur

Études de cas dans Oracle 10 : La double transaction

- Si on est tout seul à travailler sur la BD, pas de soucis
- S'il y a plusieurs utilisateurs susceptibles d'apporter des modifications aux données de la base, on peut lire la donnée à modifier en posant un verrou, on encode la nouvelle valeur, on met à jour et on libère le verrou.
- Le problème peut apparaître au niveau de l'encodage de la nouvelle valeur -> phénomène de la tasse de café

Études de cas dans Oracle 10 : La double transaction

- Lire le tuple à modifier (ancien tuple) et la lecture sera faite sans verrou !
- Encoder les nouvelles valeurs : nouveau tuple
- Relire le tuple à modifer en posant un verrou avec l'option `NOWAIT` dans l'idée de le remplacer par le tuple qui contient les nouvelles valeurs.

Études de cas dans Oracle 10 : La double transaction

Trois cas de figure peuvent se présenter lors de la lecture du tuple avec pose de verrou :

- La ressource est occupée (-54)
- La ressource n'existe plus (`NO_DATA_FOUND`)
- La ressource est disponible.

Études de cas dans Oracle 10 : La double transaction

La ressource est occupée (-54) Dans ce cas, on peut choisir d'attendre un moment et de refaire une tentative quelques instants plus tard. Si tout utilisateur qui accède à la BD le fait de manière correcte, la ressource ne devrait pas être bloquée longtemps. On peut répéter la séquence tentative/attente plusieurs fois (souvent on se limite à 3), si la ressource est toujours bloquée on abandonne la modification.

Études de cas dans Oracle 10 : La double transaction

La ressource n'existe plus (`NO_DATA_FOUND`) La ressource a été supprimée entretemps, la modification est alors interrompue

Études de cas dans Oracle 10 : La double transaction

La ressource est disponible Il faut vérifier qu'elle n'a pas été modifiée entretemps par un autre utilisateur On va alors comparer l'ancien tuple à celui que l'on vient de lire. La comparaison se fait champ par champ. Si les 2 tuples sont identiques, on va pouvoir insérer le tuple modifié et valider S'ils sont différents, le tuple a été modifié entretemps par un autre utilisateur, la modification courante est alors abandonnée. Ne pas oublier de libérer le verrou.