Transaction

Cours 11

Transaction

Transaction = séquence d'instructions SQL qui fait :

- passer la base de données **d'un état cohérent**
- **vers un autre état cohérent:**
- > But: **préserver la cohérence** de la base de données.

Transaction

Transaction = séquence d'instructions SQL qui fait :

- passer la base de données **d'un état cohérent**
- **vers un autre état cohérent:**
- > But: **préserver la cohérence** de la base de données.

Une transaction s'effectue :

- **complètement** : les mises à jour (update, …) effectuées sont **validées**.
- **incomplètement** (annulation ou panne) : les mises à jour effectuées depuis le début de la transaction sont **invalidées (annulées)**:
  - **Principe du « tout ou rien »**.

Transaction

- Sous-entend la notion de **concurrence d'accès : plusieurs utilisateurs accèdent simultanément aux tables**.

- Si **lecture** : ok.

- Si **écriture** : attention.

Exemple de GESTION DES TRANSACTIONS

Une **transaction** est une unité logique de travail, un **ensemble d'instructions** de mise à jour des données que SQL traite comme une seule entité : **soit toutes les instructions sont exécutées ensemble, soit aucune ne l'est.**

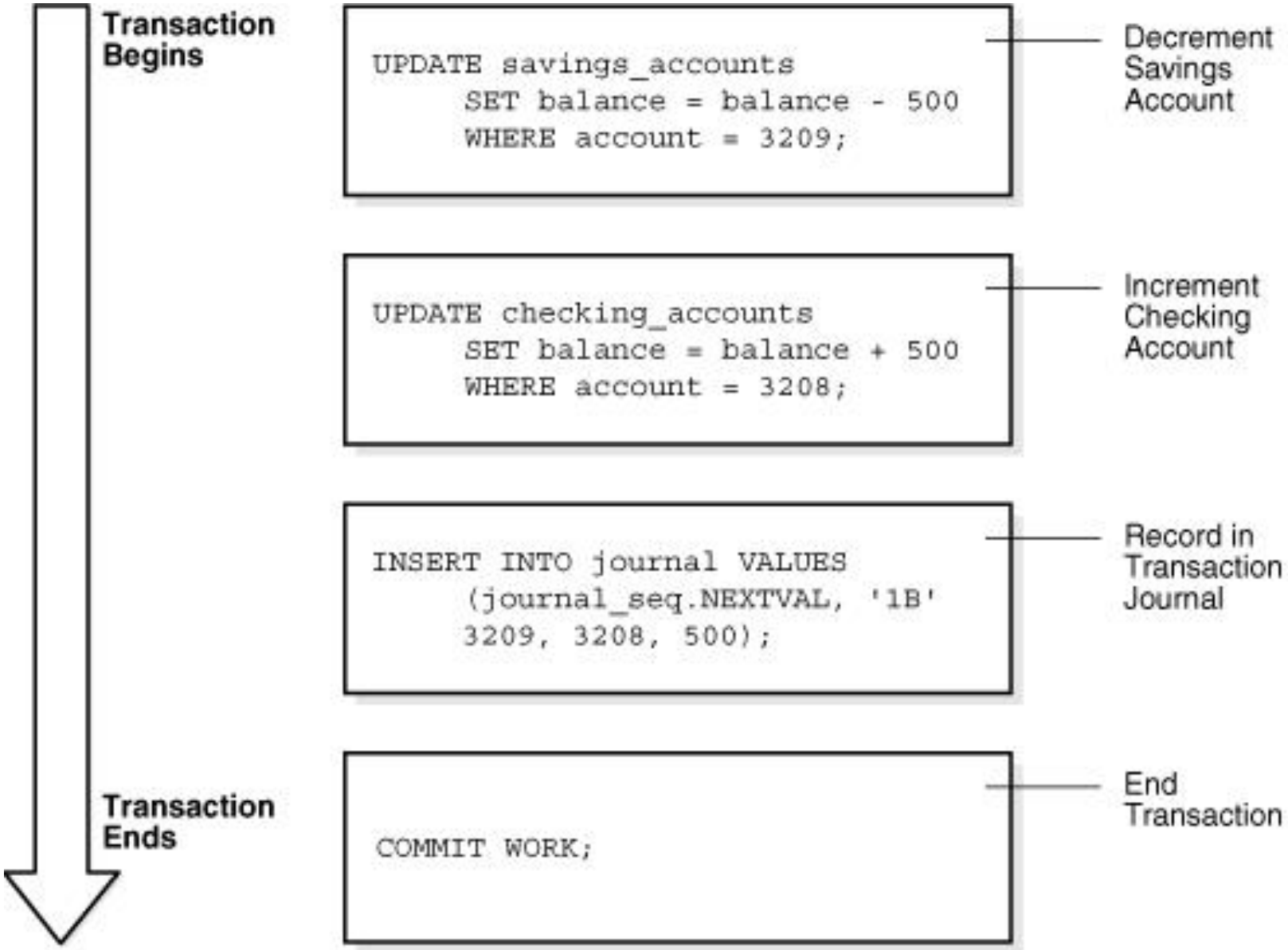
Par exemple, si une somme d'argent doit transiter d'un compte à un autre:

- les **deux opérations (crédit et débit)** de mise à jour doivent absolument **s'effectuer toutes les deux (ou ne pas s'effectuer du tout)**
- pour que la cohérence des données soit respectée.

Le mécanisme des transactions est nécessaire:

- parce que divers **événements** peuvent **interrompre l'exécution d'une séquence d'instructions :**
- défaillance du système, panne de courant, micro-coupure,
- le désir de revenir en arrière et abandonner les modifications faites jusqu'alors.

EXEMPLE:


Transaction Begins
UPDATE savings_accounts
SET balance = balance - 500
WHERE account = 3209;
Decrement Savings Account
UPDATE checking_accounts
SET balance = balance + 500
WHERE account = 3208;
Increment Checking Account
INSERT INTO journal VALUES
(journal_seq.NEXTVAL, '1B'
3209, 3208, 500);
Record in Transaction Journal
COMMIT WORK;
End Transaction
Transaction Ends

Les Problèmes

- **Problèmes de concurrence**
  - pertes d'opérations
  - introduction d'incohérences
- **Panne de transaction**
  - **erreur en cours d'exécution** du programme applicatif
  - nécessité de **défaire les mises à jour effectuées**
- **Panne système**
  - reprise avec perte de la mémoire centrale
  - toutes **les transactions en cours doivent être défaites**
- **Panne disque**
  - perte de données de la base

Propriétés des transactions

Le SGBD doit assurer que toute transaction possède les propriétés suivantes :

- **Atomicité:**
  - Unité de cohérence : **toutes les mises à jour doivent être effectuées ou aucune (tout ou rien).**
- **Cohérence:**
  - La transaction doit faire passer la base de donnée **d'un état cohérent à un autre.**
- **Isolation:**
  - **Les résultats** d'une transaction **ne sont visibles aux autres** transactions **qu'une fois la transaction validée**.
- **Durabilité:**
  - Les modifications d'une transaction **validée ne seront jamais perdue**

Commit et Abort

- INTRODUCTION D'ACTIONS ATOMIQUES
  - **Commit** (fin avec **succès**) et **Abort** (fin avec **échec**)
  - Ces actions s'effectuent **en fin de transaction**

- **COMMIT**
  - **Validation** de la transaction
  - Rend **effectives toutes les mises à jour** de la transaction

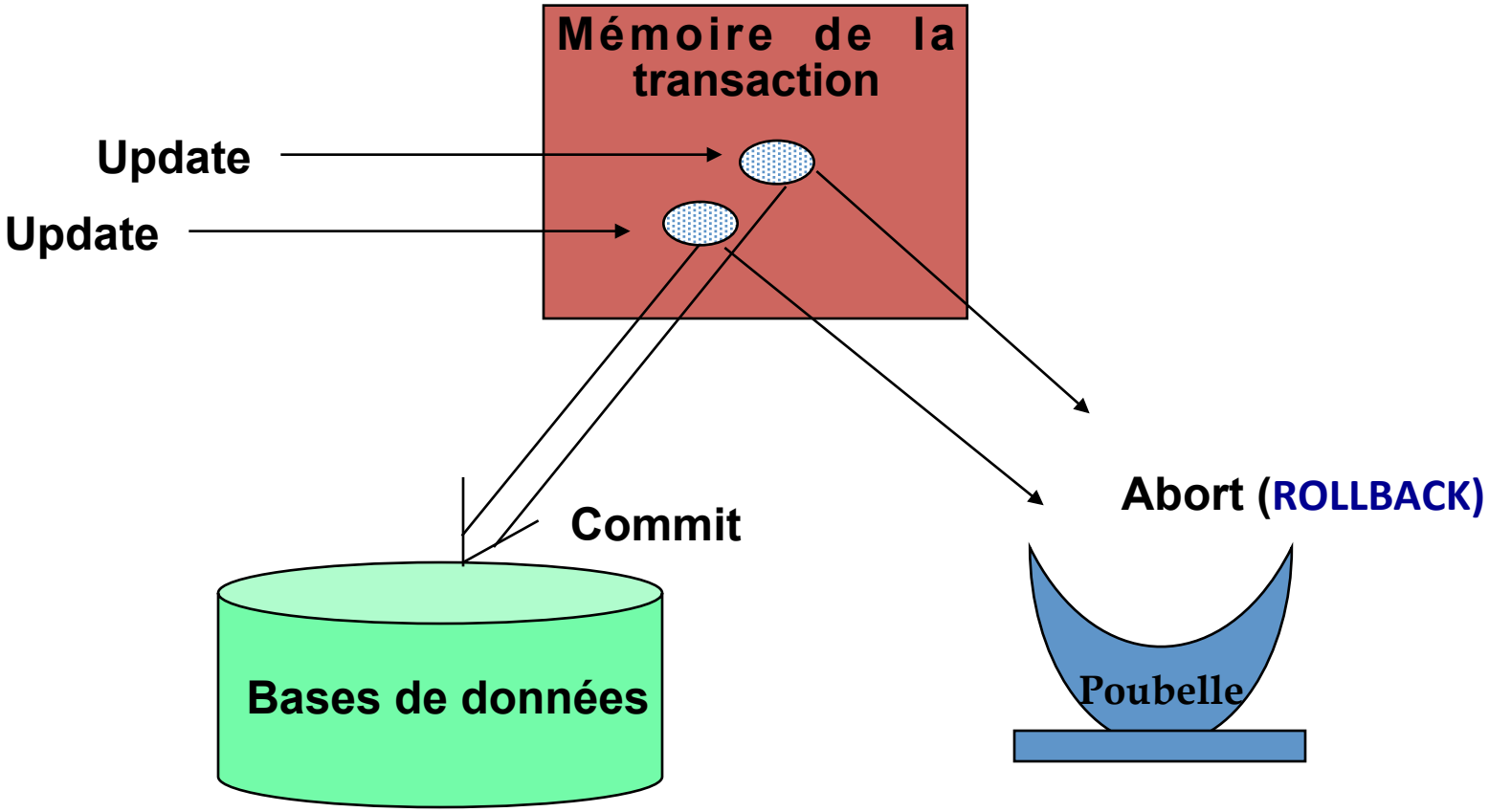
- **ABORT (ROLLBACK)**
  - **Annulation** de (toute) la transaction
  - **Défait toutes les mises à jour** de la transaction

Schéma de transaction simple

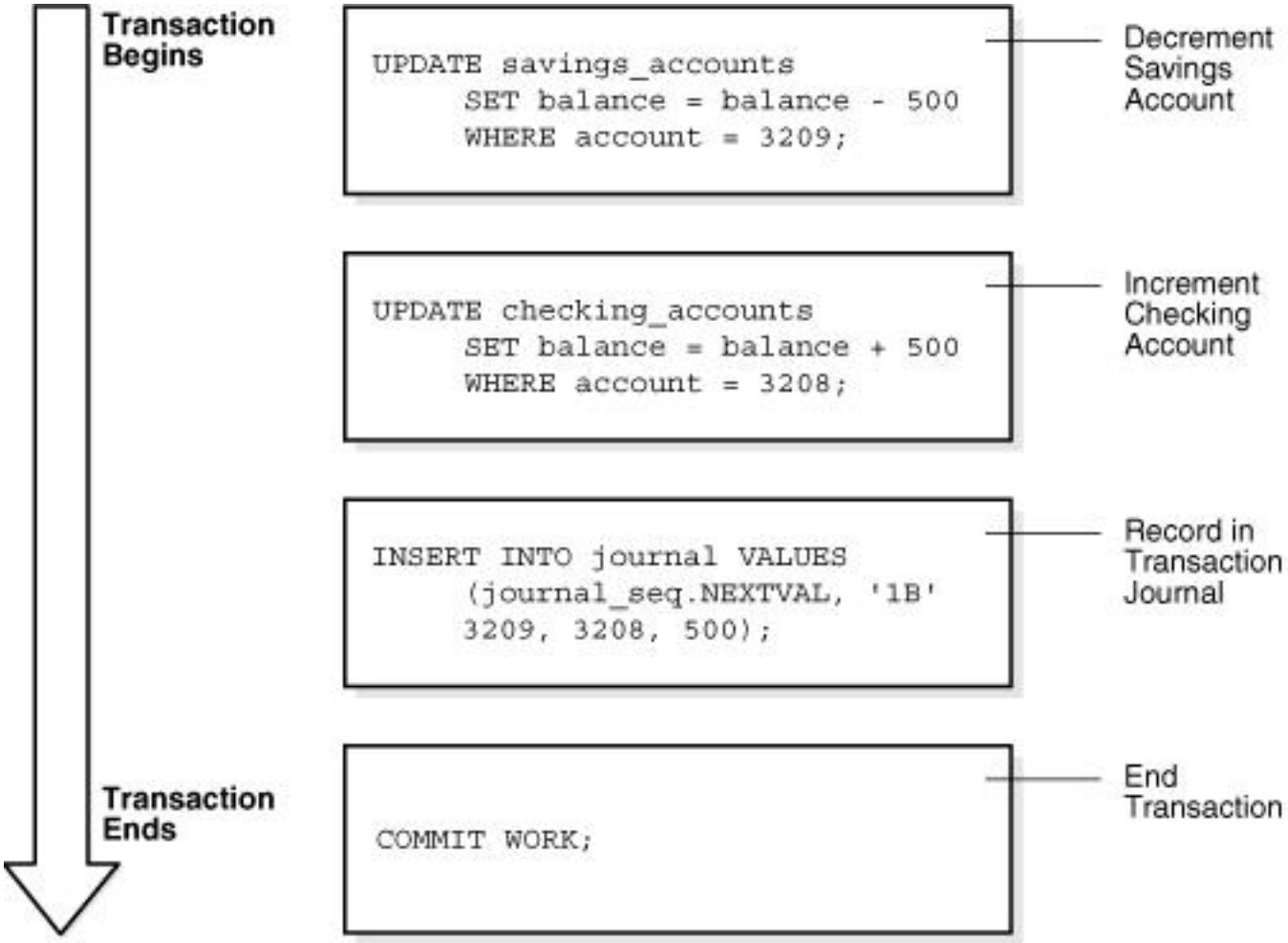
- Fin avec succès ou échec
- **Begin_Transaction**
  - update
  - update
  - .....
  - **Commit** ou Abort (**ROLLBACK**)

- Provoque l'intégration réelle des mises à jour dans la base
- Relâche les verrous

- Provoque l'annulation des mises à jour
- Relâche les verrous
- Reprend la transaction


Mémoire de la transaction
Update
Update
Commit
Abort (ROLLBACK)
Bases de données
Poubelle

EXEMPLE:


Transaction Begins
UPDATE savings_accounts
SET balance = balance - 500
WHERE account = 3209;
Decrement Savings Account
UPDATE checking_accounts
SET balance = balance + 500
WHERE account = 3208;
Increment Checking Account
INSERT INTO journal VALUES
(journal_seq.NEXTVAL, '1B'
3209, 3208, 500);
Record in Transaction Journal
Transaction Ends
COMMIT WORK;
End Transaction

Transaction: Commandes SQL

- **BEGIN TRANSACTION;**

- **COMMIT : Valide** définitivement les modifications de la transaction courante:
  - > écriture sur disque.

- **ROLLBACK : Annule** les modifications effectuées durant la transaction.

- **COMMIT : Valide** définitivement les modifications de la transaction courante:
  - > écriture sur disque.
- **ROLLBACK : Annule** les modifications effectuées durant la transaction.

Les modifications (INSERT, DELETE, UPDATE) d'une table **ne sont rendues permanentes** que si l'on exécute la commande **COMMIT**.

Tant que les changements à la base n'ont pas eu lieu de façon permanente :

- l'utilisateur peut **voir ces changements** par des ordres SELECT,

- les **autres utilisateurs ne voient pas** les modifications,
- l'utilisateur peut défaire les modifications par l'instruction ROLLBACK.

Un **ROLLBACK automatique** a lieu en cas de **panne** du système.

Transaction: Commandes SQL Exemple

```
BEGIN TRANSACTION;
    INSERT INTO ECOLE(IdEcole, NomEcole, Adresse, IdDirecteur)
                                VALUES (1, 'EPITA', '14-16 Rue Voltaire', 1);

    INSERT INTO PERSONNES(IdPersonne, Nom) VALUES (1, 'COURTOIS');
COMMIT;
```

Ce script s'exécute correctement: OUI ou NON ???

Transaction: Commandes SQL Exemple

```
BEGIN TRANSACTION;
    INSERT INTO ECOLE(IdEcole, NomEcole, Adresse, IdDirecteur)
                                VALUES (1, 'EPITA', '14-16 Rue Voltaire', 1);

    INSERT INTO PERSONNES(IdPersonne, Nom) VALUES (1, 'COURTOIS');
COMMIT;
```

Ce script s'exécute correctement ? **NON**

Transaction: Commandes SQL Exemple

```
BEGIN TRANSACTION;
    INSERT INTO PERSONNES(IdPersonne, Nom) VALUES (1, 'COURTOIS');

    INSERT INTO ECOLE(IdEcole, NomEcole, Adresse, IdDirecteur)
                                VALUES (1, 'EPITA', '14-16 Rue Voltaire', 1);
COMMIT;
```

Ce script s'exécute correctement ???

Transaction: Commandes SQL Exemple

```
BEGIN TRANSACTION;
    INSERT INTO PERSONNES(IdPersonne, Nom) VALUES (1, 'COURTOIS');

    INSERT INTO ECOLE(IdEcole, NomEcole, Adresse, IdDirecteur)
                                VALUES (1, 'EPITA', '14-16 Rue Voltaire', 1);
COMMIT;
```

Ce script s'exécute correctement ? **OUI**

Exemples :

```
CREATE TABLE T ( A NUMBER,
B NUMBER, PRIMARY KEY (A));

 INSERT INTO T VALUES ( 1, 1);
 INSERT INTO T VALUES ( 2, 2);
 SELECT * FROM T ;

Le résultat obtenu est :
       A          B
-------- ---------
       1          1
       2          2

 COMMIT ;

Validation effectuée.
```

```
INSERT INTO T VALUES ( 3, 3);
 INSERT INTO T VALUES ( 4, 4);
 SELECT * FROM T ;
Le résultat obtenu est :
         A          B
--------- ---------
         1          1
         2          2
         3          3
         4          4
  ROLLBACK ;
Annulation (ROLLBACK) effectuée.
 SELECT * FROM T ;
Le résultat obtenu est :
         A          B
--------- ---------
         1          1
         2          2
```

```
INSERT INTO T VALUES ( 3, 3);
SAVEPOINT A ;
1 ligne créée.
Point de sauvegarde (SAVEPOINT) créé.
INSERT INTO T VALUES ( 4, 4);
1 ligne créée.
 SELECT * FROM T ;
Le résultat obtenu est :
         A          B
--------- ---------
         1          1
         2          2
         3          3
         4          4
```

```
ROLLBACK TO SAVEPOINT A;
Annulation (ROLLBACK)
effectuée.
SELECT * FROM T ;
Le résultat obtenu est :
         A          B
--------- ---------
         1          1
         2          2
         3          3
```

- Blocage des tuples et des tables.

Transaction T1

```
SELECT * FROM T ;
Le résultat obtenu est :
        A          B
--------- ---------
        1          1
        2          2
        3          3
```

Transaction T2

```
SELECT * FROM T ;
Le résultat obtenu est :
        A          B
--------- ---------
        1          1
        2          2
        3          3
```

Transaction T1

```
update t set b = b - 2 ;

1 ligne mise à jour.

select * from t;

        A          B
--------- ---------
        1          1
        2          2
        3          1
```

Transaction T2

```
select * from t ;

        A          B
--------- ---------
        1          1
        2          2
        3          3
update t set b = b - 2 ;
-- blocage tant que T1 n'a pas
fait de COMMIT
```

Transaction T1 **COMMIT;** **Validation effectuée.**	**Transaction T2** **1 ligne mise à jour.** **select * from t;** **A B** **--------- ---------** **1 1** **2 2** **3 -1** **COMMIT;** **Validation effectuée.**
Transaction T1 **select * from t;** **A B** **--------- ---------** **1 1** **2 2** **3 -1**	**Transaction T2** **select * from t;** **A B** **--------- ---------** **1 1** **2 2** **3 -1**

Transaction T1 lock table t IN EXCLUSIVE MODE; Table(s) verrouillée(s).	Transaction T2
Mises à jour	En attente
Mises à jour	En attente
Commit ;	lock table t IN EXCLUSIVE MODE; etc…

Le Blocage de tuples et de tables se fait par la commande SQL :

LOCK TABLE **LeNomDeLObjet** IN **LeMode** MODE ;

(LeNomDeLObjet : Le nom d'une table ou d'une vue.)

Propriétés des transactions

Le SGBD doit assurer que toute transaction possède les propriétés suivantes :

- **Atomicité: ..**
- **Cohérence: …**
- **Isolation:**
  - **Les résultats** d'une transaction **ne sont visibles aux autres** transactions **qu'une fois la transaction validée**.
- **Durabilité: …**

Niveaux **d'isolation** de SQL

- Lecture impropre (dirty read) :

  cette anomalie se produit lorsqu'une transaction **lit des données qui sont en train d'être modifiées** par une autre transaction (non encore validée).

- Lecture non renouvelable (nonrepeatable read) :

  cette anomalie survient si une requête **ne renvoie pas les mêmes résultats lors d'exécutions successives.** C'est le cas si les données que vous lisez **sont modifiées par une autre transaction** (c'est un peu l'inverse de la lecture impropre).

- Lecture Fantôme (phantom read) :

  cette anomalie se produit **si des exécutions successives d'une même requête renvoient des données en plus ou en moins**. Cela peut être le cas si une autre transaction est **en train de supprimer ou d'ajouter** des données à la table.

Niveaux d'isolation de SQL

- Quel est le Problème ici ?

Transaction T1	**Transaction T2**
BEGIN	BEGIN
UPDATE Client	
SET nom = 'cours BDD'	
WHERE nom is NULL	
	UPDATE Client
	SET nom = UPPER(nom)
	Where nom LIKE 'cours%'
ROLLBACK	

Niveaux d'isolation de SQL

- Lecture impropre (dirty read) :

  cette anomalie se produit lorsqu'une transaction **lit des données qui sont en train d'être modifiées** par une autre transaction (non encore validée).

Transaction T1	**Transaction T2**
BEGIN	BEGIN
UPDATE Client	
SET nom = 'cours BDD'	
WHERE nom is NULL	
	UPDATE Client
	SET nom = UPPER(nom)
	Where nom LIKE 'cours%'
ROLLBACK	

Niveaux d'isolation de SQL

- Quel est le Problème ici ?

Transaction T1

```
BEGIN
SELECT *
FROM Client
WHERE nom Like 'cours%'




SELECT *
FROM Client
WHERE nom Like 'cours%'
```

Transaction T2

```
BEGIN




UPDATE Client
SET nom = 'cours BDD'
WHERE nom is NULL
```

Niveaux d'isolation de SQL

- Lecture non renouvelable (nonrepeatable read) :

  cette anomalie survient si une requête **ne renvoie pas les mêmes résultats lors d'exécutions successives.** C'est le cas si les données que vous lisez **sont modifiées par une autre transaction** (c'est un peu l'inverse de la lecture impropre).

Transaction T1	**Transaction T2**
BEGIN	BEGIN
SELECT *	
FROM Client	
WHERE nom Like 'cours%'	
	UPDATE Client
	SET nom = 'cours BDD'
	WHERE nom is NULL
SELECT *	
FROM Client	
WHERE nom Like 'cours%'	

Niveaux d'isolation de SQL

- Quel est le Problème ici ?

Transaction T1	**Transaction T2**
BEGIN SELECT * FROM Client WHERE nom Like 'cours%'	BEGIN
	DELETE FROM Client WHERE nom like 'cours%'
SELECT * FROM Client WHERE nom Like 'cours%'	

Niveaux d'isolation de SQL

- **Lecture Fantôme (phantom read) :**

  cette anomalie se produit **si des exécutions successives d'une même requête renvoient des données en plus ou en moins.**

  Cela peut être le cas si **une autre transaction est en train de supprimer ou d'ajouter** des données à la table.

Transaction T1	**Transaction T2**
BEGIN SELECT * FROM Client WHERE nom Like 'cours%'	BEGIN DELETE FROM Client WHERE nom like 'cours%'
SELECT * FROM Client WHERE nom Like 'cours%'	

Niveaux d'isolation de SQL: Syntaxe

- BEGIN TRANSACTION [< trans_mod >]
- trans_mod :

  ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED} READ WRITE | READ ONLY

Isolation	Dirty read	Nonrepeatable read	Phantom read
read uncommitted	Possible	Possible	Possible
read committed	Impossible	Possible	Possible
repeatable read	Impossible	Impossible	Possible
serializable	Impossible	Impossible	Impossible

Niveaux d'isolation de SQL: Syntaxe

BEGIN TRANSACTION **READ ONLY :**

- **Assure** la cohérence au niveau transaction : dans la transaction **les lectures sont reproductibles.**

- La transaction **ne voit que les changements qui ont été validés au moment où elle démarre** ;

- dans la transaction **on ne peut pas utiliser INSERT, DELETE, UPDATE.**

Niveaux d'isolation de SQL: Syntaxe

BEGIN TRANSACTION ISOLATION LEVEL **READ COMMITTED** :

- C'est l'option par défaut.
- Chaque requête (SELECT, clause WHERE) exécutée par la transaction **ne voit que les données qui ont été validées (COMMITTED) avant que la requête commence.**
- Cette option **ne résout pas les problèmes de lecture non reproductible** ni de lecture fantôme**.**

Niveaux d'isolation de SQL: Syntaxe

BEGIN TRANSACTION ISOLATION LEVEL **SERIALIZABLE :**

Avec cette option, une transaction sérialisable:

- **voit seulement les changements validés au moment où la transaction démarre,**
- ainsi que les changements effectués (INSERT, DELETE, UPDATE) **dans la transaction elle-même.**
- Elle **prévient des lectures non reproductible et fantôme**.
- ORACLE **signale une erreur** quand une transaction déclarée sérialisable tente **une mise à jour sur une ligne qui a été modifié ailleurs par une transaction non validé avant son démarrage**.