Chapitre 5 Les TRANSACTIONS

Djellali H

M1 RSI fev 2021

Sommaire

- Introduction
- Définition Transaction
- Propriétés ACID
- ACID: Atomicité- Cohérence-
- Isolation- Durabilite

Introduction

- Un modèle simplifié de SGBD se compose de programmes utilisateurs, d'un système et de mémoires secondaires.
- Les programmes accèdent aux données au moyen d'opérations du SGBD (Select, Insert, Update, Delete), généralement en SQL.
- Ces opérations déclenchent des actions de lecture et écriture sur disques (Read, Write), qui sont exécutées par le système, et qui conduisent à des entrées-sorties sur les mémoires secondaires.

Transaction (Transaction)

- Séquence d'opérations liées comportant des mises à jour ponctuelles d'une base de données devant vérifier **les propriétés d'atomicité, cohérence, isolation et durabilité** (ACID).
- C'est une séquence d'operations (lecture/ecriture) qui doit etre éxécutée dans son intégralité (de manière atomique) , amenant la BD d'un etat cohérent à un autre état cohérent.

Transaction exemple

- begin transaction

Try

Compte X=50000 DA
Compte y =20000 DA
Transfert de X ver Y de montantTransfert = 10000 DA
Update compte X = X- montantTransfert = 40000 DA
Update compte Y = Y + montantTransfert = 30000 DA
Enregistrer operations dans le journal
Save(date, CompteSource, compteDestination,montantTransfert)

Exception

cas d'erreur : Annuler toutes les opérations faite

End transaction

les propriétés ACID Atomicite- Cohérence-

- **Atomicité**
- Une transaction doit effectuer toutes ses mises à jour ou ne rien faire du tout(aucune).
- En cas d'échec, le système doit annuler toutes les modifications qu'elle a engagées (Rollback).
- L'atomicité est menacée par les pannes de programme, du système ou du matériel, et plus généralement par tout événement susceptible d'interrompre une transaction en cours.

Cohérence

- La transaction doit faire passer la base de données **d'un état cohérent** à un **autre**. En cas d'échec, l'état cohérent initial doit être **restauré.**
- La cohérence de la base peut être violée par un programme erroné ou un conflit d'accès concurrent entre transactions.

Isolation

- Les résultats d'une transaction ne doivent être visibles aux autres transactions qu'une fois la transaction validée, afin d'éviter les interférences avec les autres transactions. Les accès concurrents peuvent mettre en question l'isolation.
- Les accés concurrents peuvent mettre en question l'isolation.

Durabilité

- Dès qu'une transaction valide ses modifications, le système doit garantir qu'elles seront conservées en cas de panne. Le problème essentiel survient en cas de panne, notamment lors des pannes disques.

Les propriétés ACID

- Ces propriétés doivent être garanties dans le cadre d'un système SGBD centralisé, mais aussi dans les systèmes répartis. Elles nécessitent deux types de contrôle, qui sont intégrés dans un SGBD :
- contrôle de concurrence, résistance aux pannes avec validation et reprise.

THEORIE DE LA CONCURRENCE

- Cette section aborde les problèmes de gestion des accès concurrents. Les solutions proposées permettent de garantir la Cohérence et l'Isolation des mises à jour des transactions (le C et le I de ACID).
- Elles sont basées sur la théorie de la sérialisabilité des transactions.

- L’objectif général est de rendre invisible aux clients le partage simultané des données. Cette transparence nécessite des contrôles des accès concurrents au sein du SGBD. Ceux-ci s’effectuent au moyen de protocoles spéciaux permettant de synchroniser les mises à jour afin d’éviter les pertes de mises à jour et l'apparitions d’incohérences.
- Une perte de mise à jour survient lorsqu’une transaction T1 exécute une mise à jour calculée à partir d’une valeur périmée de donnée, c’est-à-dire d’une valeur modifiée par une autre transaction T2 depuis la lecture par la transaction T1. La mise à jour de T2 est donc écrasée par celle de T1.
- Une perte de mise à jour est illustrée figure .1. La mise à jour de la transaction T2 est perdue.

Exemple de perte de mise a jour

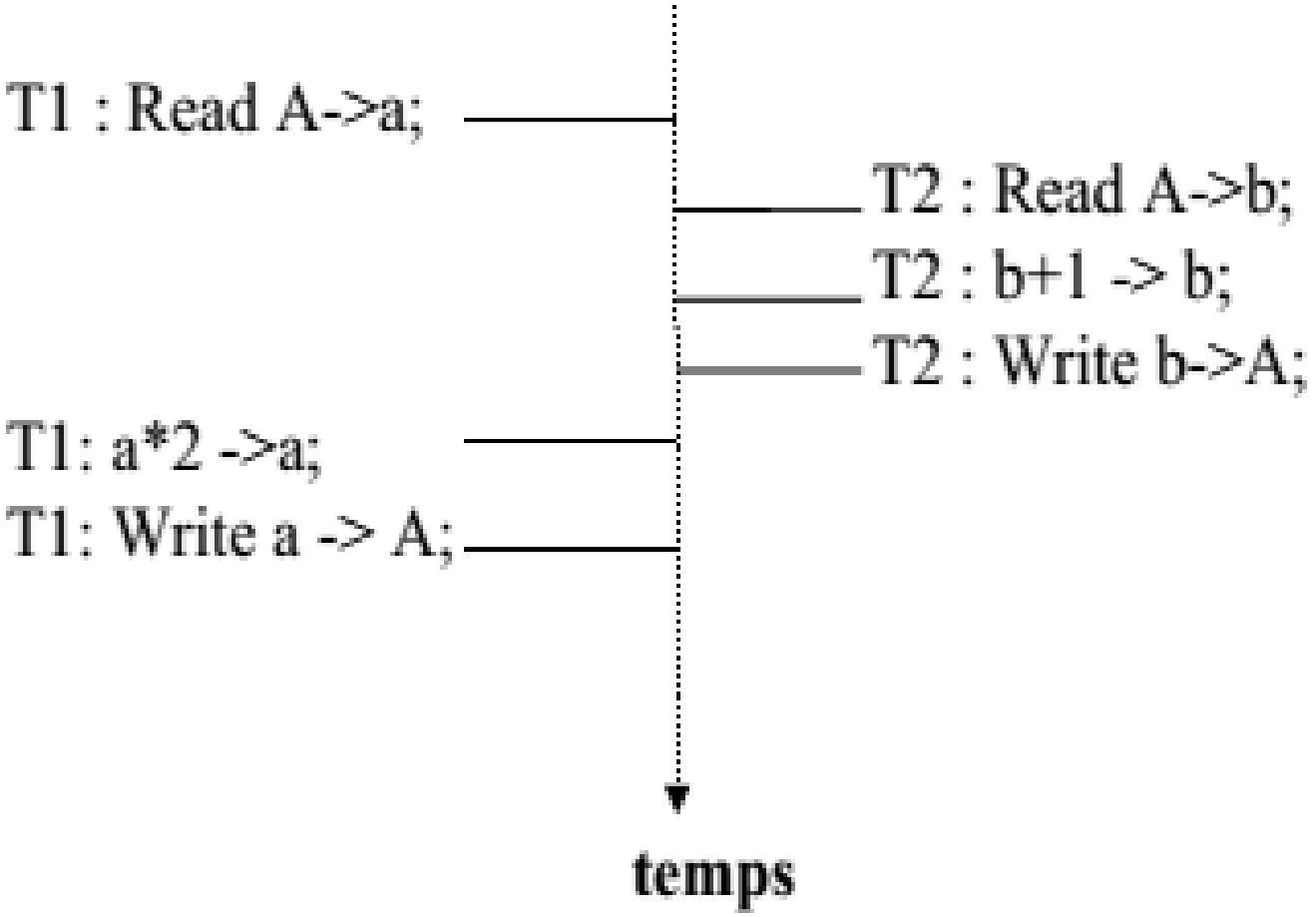


figure1,. Exemple de perte de mise a jour

Transaction et concurrence

Transactions et concurrence

1. **Introduction : les transactions**
2. **Les problèmes**
3. Sérialisabilité
4. Estampillage
5. Verrouillage à 2 phases

Les problèmes rencontres:

- Lecture non reproductible
- Lecture fantôme
- Lecture sale

Transactions bancaires

- Alice souhaite verser à Bob 100 euros.
- Le SGBD doit effectuer 2 opérations sur la table COMPTE :
  - Retirer 100 euros au compte d'Alice
  - Ajouter 100 euros au compte de Bob

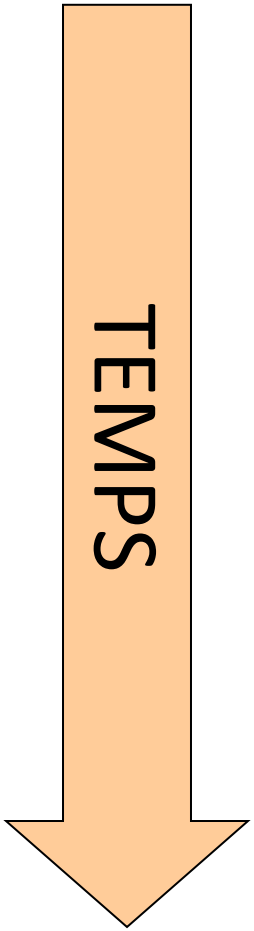
COMPTE

NC	CLIENT	SOLDE
0	Alice	1000
1	Bob	750
…	…	…

Transactions bancaires

t_1

UPDATE COMPTE
SET SOLDE = SOLDE – 100
WHERE NC = 0



COMPTE

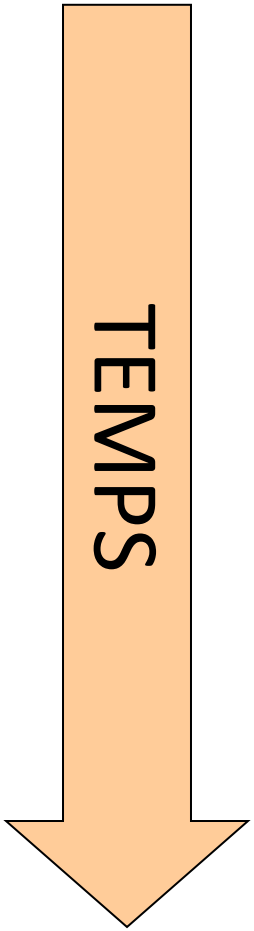
NC	CLIENT	SOLDE
0	Alice	1000
1	Bob	750
…	…	…

Somme : 1750

Transactions bancaires

t_1

UPDATE COMPTE
SET SOLDE = SOLDE – 100
WHERE NC = 0



COMPTE

NC	CLIENT	SOLDE
0	Alice	900
1	Bob	750
…	…	…

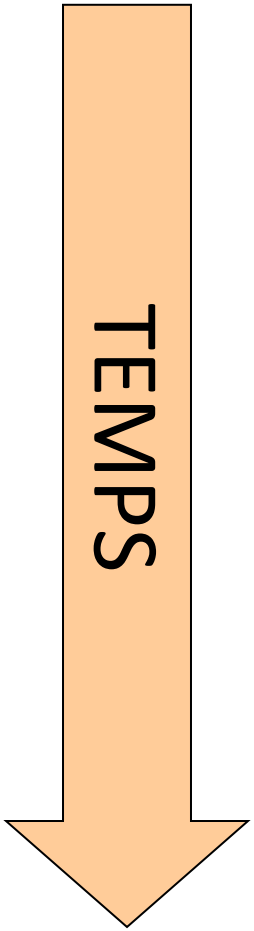
Somme : 1650

Transactions bancaires

t_1 UPDATE COMPTE
SET SOLDE = SOLDE – 100
WHERE NC = 0

t_2 UPDATE COMPTE
SET SOLDE = SOLDE + 100
WHERE NC = 1

t_3 **COMMIT** (valider) ou
ROLLBACK (avorter)



COMPTE

NC	CLIENT	SOLDE
0	Alice	900
1	Bob	**850**
…	…	…

Somme : 1750

Commit ou Rollback

- Les **deux** opérations (ou aucune)

doivent être validées pour maintenir

la cohérence des données : c'est **l'atomicité !**

- **COMMIT :** permet de valider **tous** les changements
- **ROLLBACK :** permet d'annuler **tous** les changements

COMPTE

NC	CLIENT	SOLDE
0	Alice	900
1	Bob	850
…	…	…

On appelle **transaction** un ensemble séquentiel d'opérations permettant de passer d'un état cohérent à un autre.

Concurrence

- Nombreuses transactions en parallèle
- Besoin pour le SGBD d'être capable de les gérer. Eviter :
  - Pertes d'opérations / introduction d'incohérences
  - Observation d'incohérences: Lectures non reproductibles / lectures fantômes

Des problèmes similaires apparaissent dans le cas des pannes.

Propriétés des transactions : ACIDité [Harder&Reuter, ACM CS 15(4), 1983]

- **Atomicité** : Toutes les MAJ sont exécutées ou aucune
- **Cohérence** : Passer d'un état cohérent à un autre (pas géré par le système)
- **Isolation** : La transaction s'effectue comme si elle était seule
- **Durabilité** : Une fois une transactions validée, son effet ne peut pas être perdu suite à une panne quelconque

SEMAINE 1 : Transactions et concurrence

1. **Introduction : les transactions**
2. Les problèmes
3. Sérialisabilité
4. Estampillage
5. Verrouillage à 2 phases
6. Degrés d'isolation dans les SGBD
7. Verrouillage hiérarchique

SEMAINE 1 : Transactions et concurrence

1. Introduction : les transactions
2. **Les problèmes**
3. Sérialisabilité
4. Estampillage
5. Verrouillage à 2 phases
6. Degrés d'isolation dans les SGBD
7. Verrouillage hiérarchique

Problématique

1. Une base de données n’est pas interrogée et modifiée par un seul utilisateur.
2. Des problèmes d’incohérences peuvent apparaître lorsque plusieurs utilisateurs effectuent des opérations conflictuelles, ce qui peut être dû à un défaut d’isolation.

Problématique

1. Une base de données n'est pas interrogée et modifiée par un seul utilisateur.
2. Des problèmes d'incohérences peuvent apparaître lorsque plusieurs utilisateurs effectuent des opérations conflictuelles, ce qui peut être dû à un défaut d'isolation.
3. Quels sont ces incohérences ?
4. Quelles sont ces opérations ?
5. Comment éviter de se placer dans des situations d'opérations conflictuelles ?

Problématique

1. Des problèmes peuvent survenir lors de l'accès (lecture / écriture) concurrent sur des opérations successives
2. On aimerait que les opérations puissent se dérouler en isolation

Nous allons étudier les aspects *isolation* des transactions.

3. Dans la suite : déterminer les opérations potentiellement conflictuelles

entre deux transactions.

Exemple de Base de Données

1. 1 table : EMP (NE, Nom, Sal)
2. 3 nuplets :

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

Exemple de Base de Données

1. 1 table : EMP (NE, Nom, Sal)
2. 3 tuples :

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

… et deux utilisateurs :

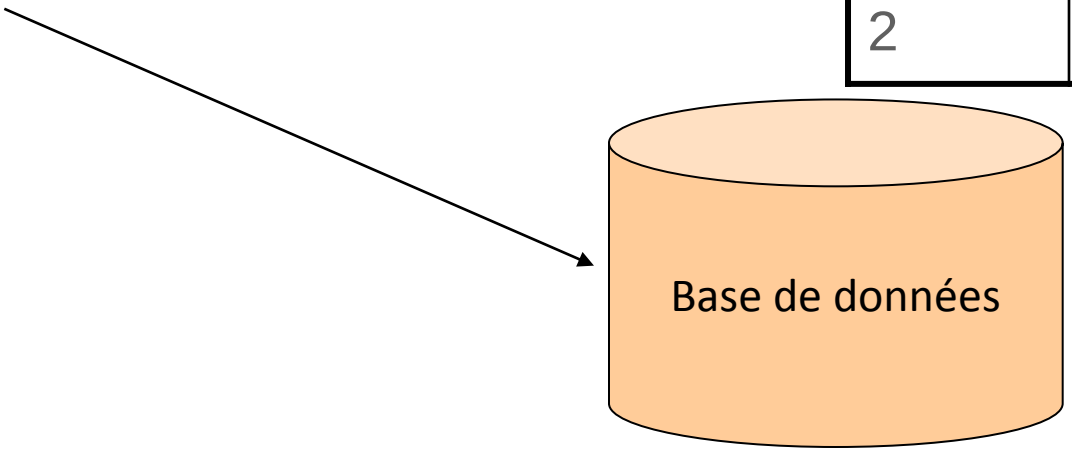
Alice et Bob

Lecture de nuplets distincts

Alice

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

Bob



NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

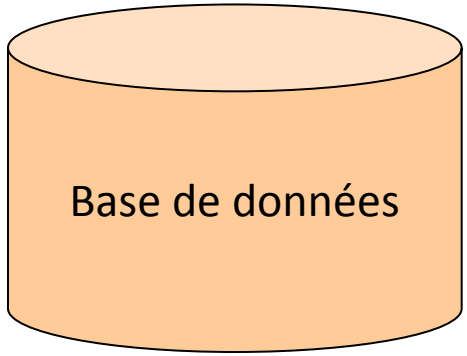
Lecture de nuplets distincts

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

Alice

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »



Bob

Lecture de nuplets distincts

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

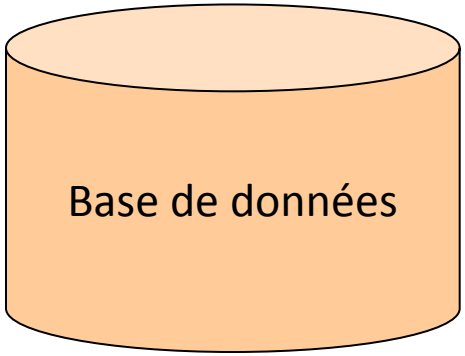
Alice

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Diana »



Lecture de nuplets distincts

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

Alice

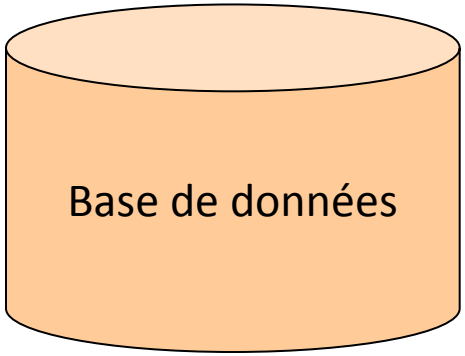
SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Diana »

« 2100 »



Lecture de nuplets distincts

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

Alice

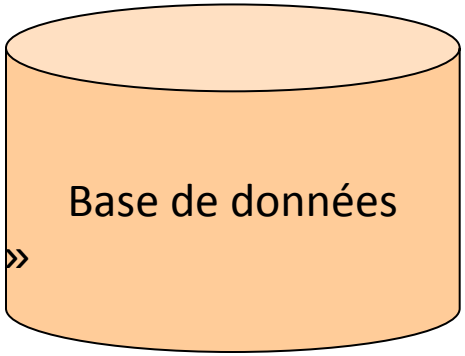
SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Diana »

« 2100 »



Lecture de nuplets distincts

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

Alice

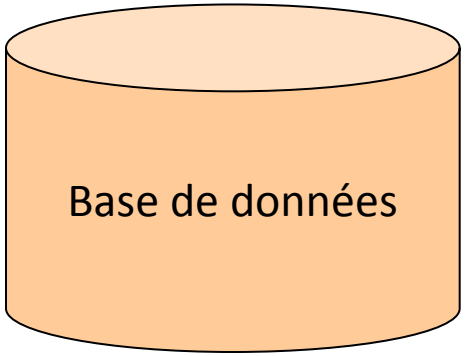
SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Diana »

« 2100 »



**PAS DE PROBLEME
SI A ET B LISENT DES
N-UPLETS DISTINCTS**

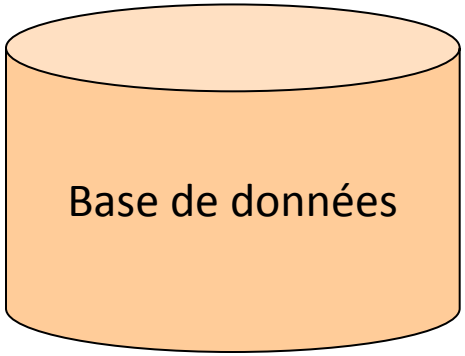
Lecture de nuplets identiques

Alice

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Bob



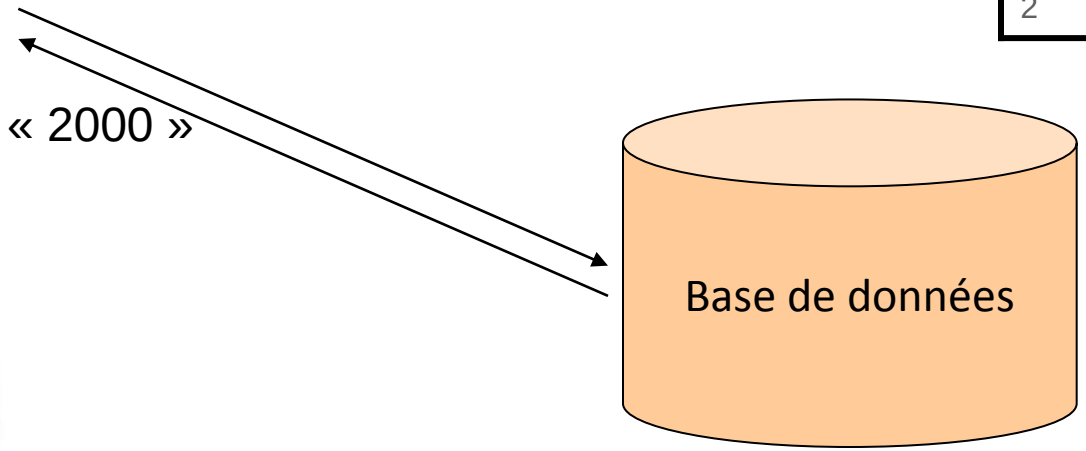
Lecture de nuplets identiques

Alice

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture de nuplets identiques

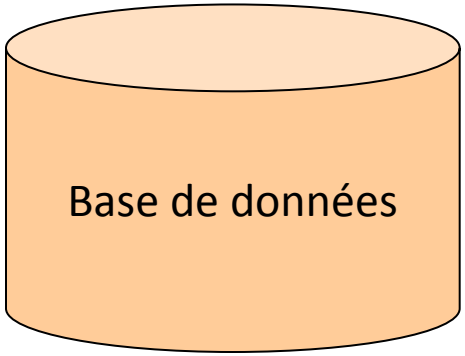
Alice

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Charlie »



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture de nuplets identiques

Alice

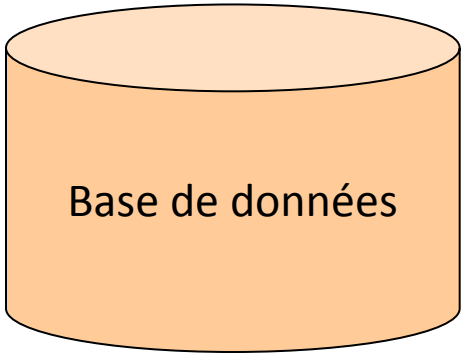
SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Charlie »

« 2000 »



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture de nuplets identiques

Alice

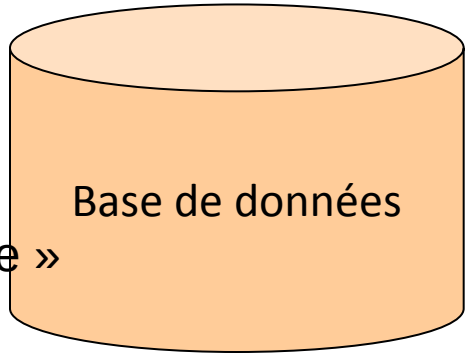
SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »



NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

Lecture de nuplets identiques

Alice

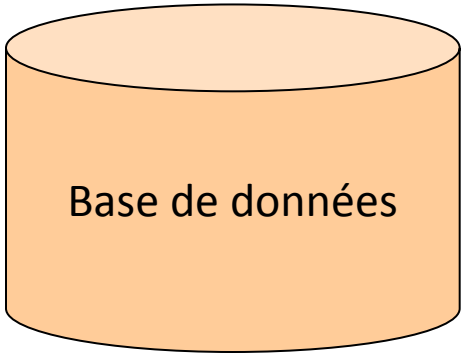
SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Charlie »

« 2000 »



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

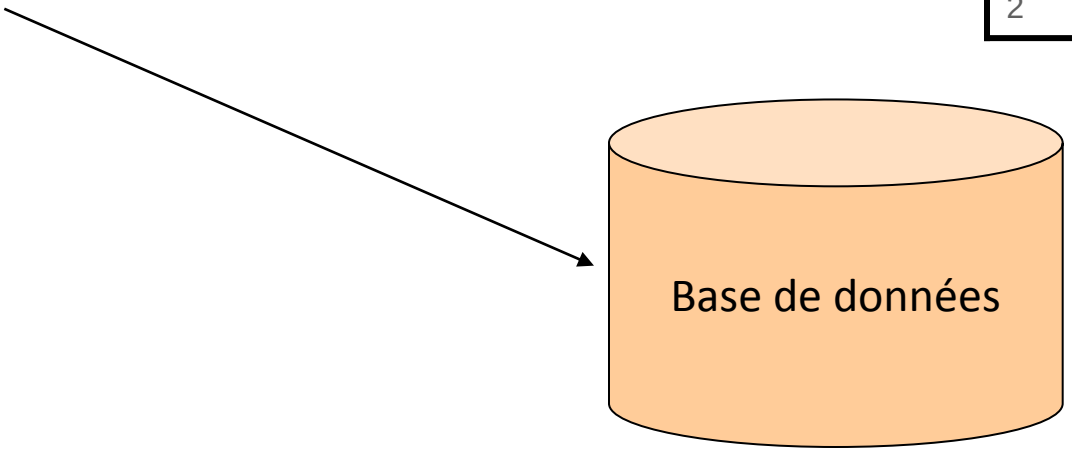
PAS DE PROBLEME SI A ET B LISENT LE MEME N-UPLET

Lecture et écriture de nuplets différents

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600



Bob

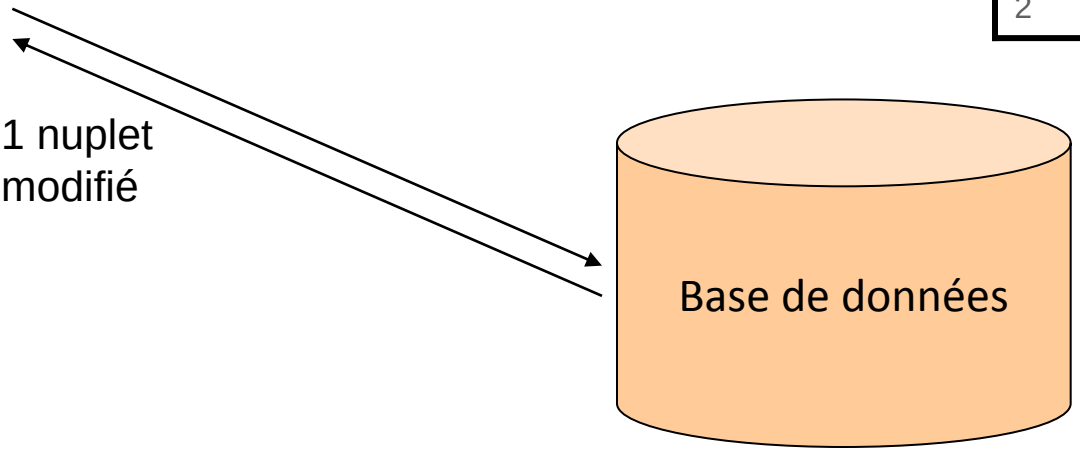
Lecture et écriture de nuplets différents

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet
modifié

Bob



NE	NOM	SAL
0	Charlie	**2050**
1	Diana	2100
2	Eric	1600

Lecture et écriture de nuplets différents

NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

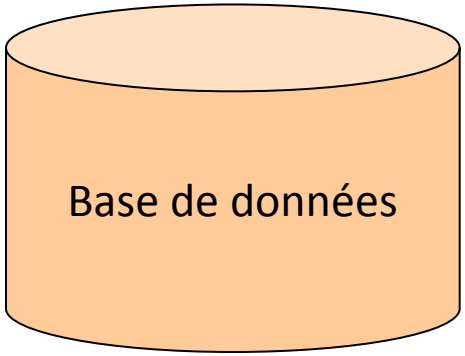
Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Diana »



Lecture et écriture de nuplets différents

Alice

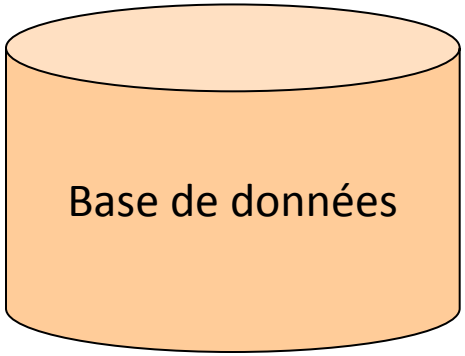
UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Diana »

« 2100 »



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture et écriture de nuplets différents

Alice

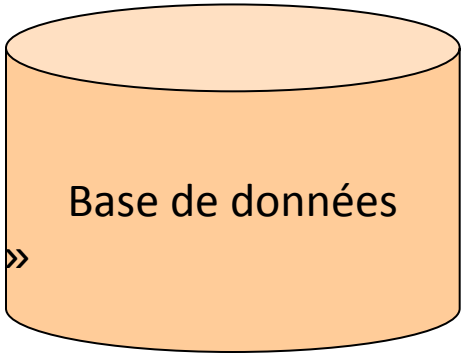
UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Diana »

« 2100 »



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture et écriture de nuplets différents

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

1 nuplet modifié

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Diana »

Base de données

« 2100 »

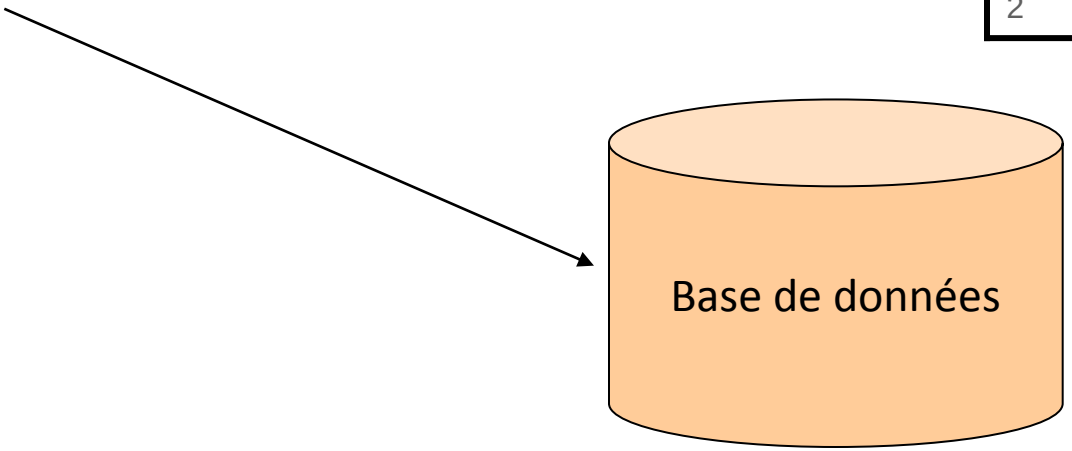
**PAS DE PROBLEME
SI A MODIFIE UN N-UPLET
QUE B NE LIT PAS
PAR LA SUITE**

Lecture et écriture de nuplets identiques

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600



Bob

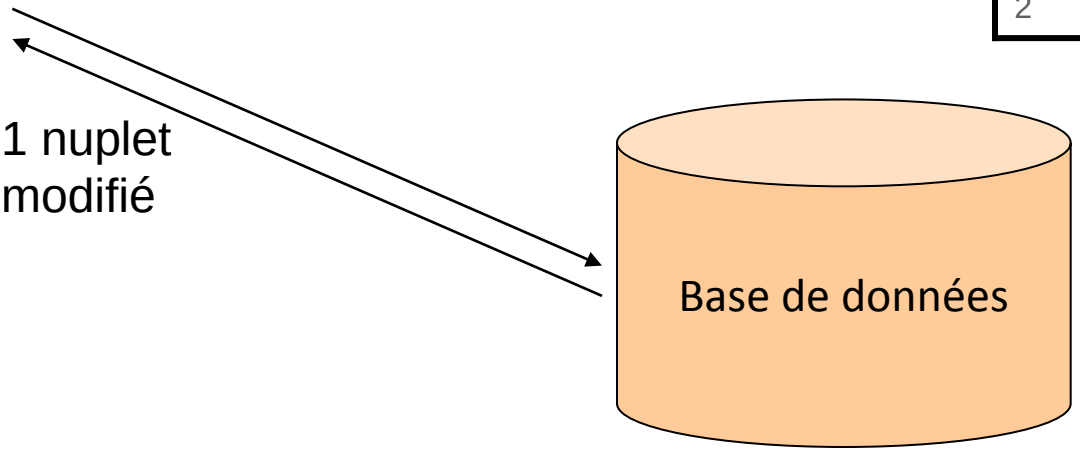
Lecture et écriture de nuplets identiques

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

Bob



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture et écriture de nuplets identiques

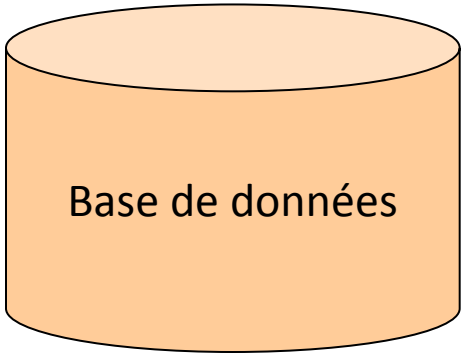
Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Charlie »



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture et écriture de nuplets identiques

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Charlie »

« 2050 »

Nouvelle valeur !

Base de données

NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture et écriture de nuplets identiques

Alice

UPDATE EMP
SET SAL = 2050
WHERE NOM
= « Charlie »

« 2000 »

1 nuplet modifié

Ancienne valeur !

NE	NOM	SAL
0	Charlie	**2000**
1	Diana	2100
2	Eric	1600

Bob

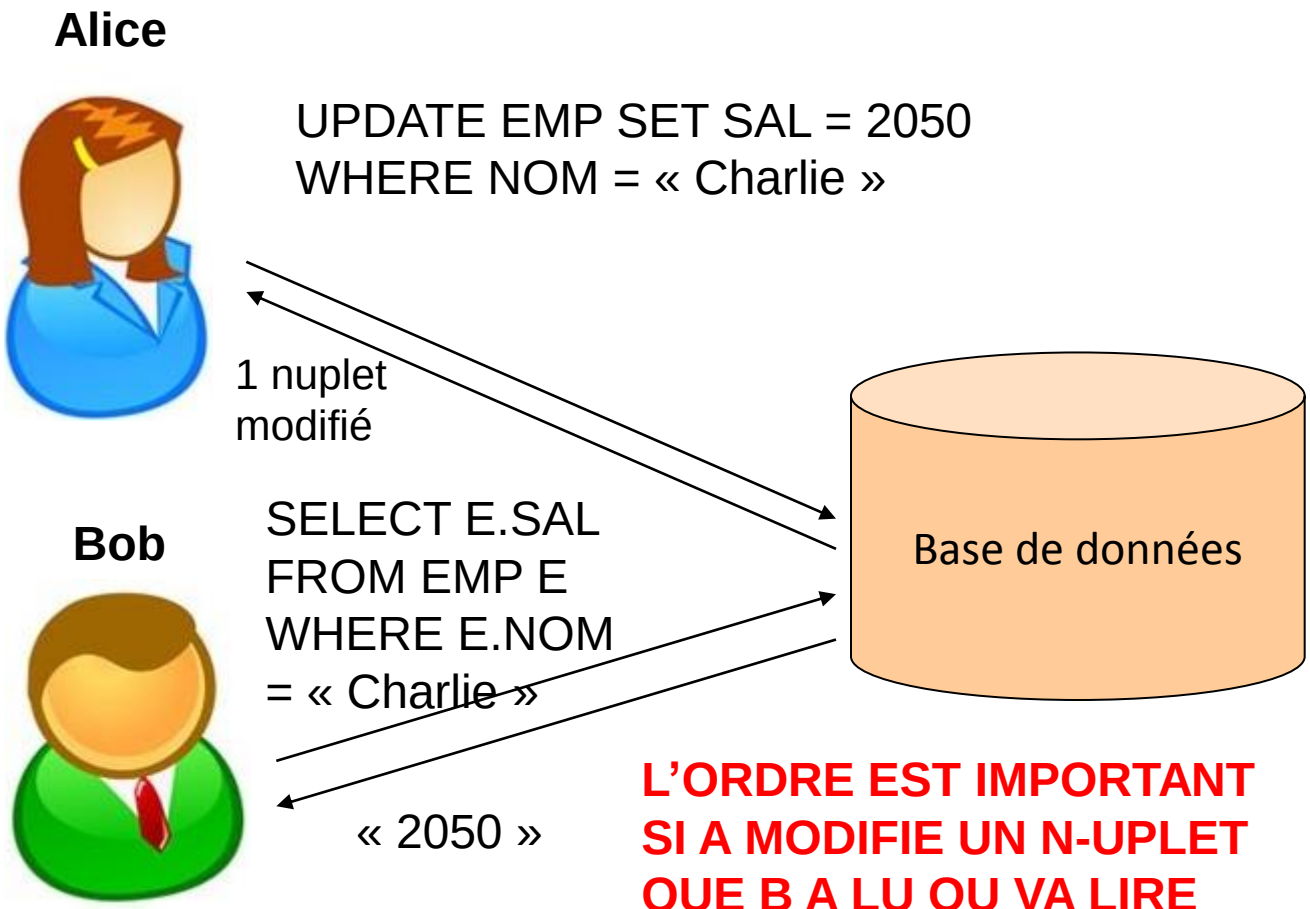
SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2050 »

Base de données

NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture et écriture de nuplets identiques



L'ORDRE EST IMPORTANT
SI A MODIFIE UN N-UPLET
QUE B A LU OU VA LIRE

Anomalie (problème) de la *lecture non reproductible*

Bob

Alice

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

T_1 SELECT E.SAL FROM EMP E
WHERE E.NOM = « Charlie »

2000 ←

TEMPS

Anomalie (problème) de la *lecture non reproductible*

Bob

Alice

NE	NOM	SAL
0	Charlie	**2050**
1	Diana	2100
2	Eric	1600

T_1 SELECT E.SAL FROM EMP E
WHERE E.NOM = « Charlie »

2000 ←

TEMPS

T_2 UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

Anomalie (problème) de la *lecture non reproductible*

Bob

Alice

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

T_1 SELECT E.SAL FROM EMP E
WHERE E.NOM = « Charlie »

2000 ←

TEMPS

T_2 UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

T_3 SELECT E.SAL FROM EMP E
WHERE E.NOM = « Charlie »

2050 ←

Anomalie (problème) de la *lecture non reproductible*

Bob

Alice

T_1 SELECT E.SAL FROM EMP E
WHERE E.NOM = « Charlie »

2000 ←

TEMPS

T_2 UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

T_3 SELECT E.SAL FROM EMP E
WHERE E.NOM = « Charlie »

2050 ←

B LIT DEUX FOIS LA MEME VALEUR ET OBTIENT DES RESULTATS DIFFERENTS !

CETTE ANOMALIE EST DITE *LECTURE NON REPRODUCTIBLE.*

Nuplet lu de manière explicite ou implicite

1. Un nuplet peut être produit par la requête (explicite).
2. Un nuplet peut être utilisé pour produire le résultat de la requête (implicite) e.g. requêtes d'agrégats.
3. Une modification de l'un des nuplets utilisé pour calculer la requête causera donc un problème.

Anomalie (problème) de la *lecture fantôme*

Bob

Alice

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

T_1 SELECT AVG(E.SAL)
FROM EMP E

1900 ←

TEMPS

Anomalie (problème) de la *lecture fantôme*

Bob

Alice

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600
3	**Flore**	**2300**

T_1 SELECT AVG(E.SAL)
FROM EMP E

1900 ←

TEMPS

T_2 INSERT INTO EMP VALUES
(3, « Flore», 2300)

Anomalie (problème) de la *lecture fantôme*

Bob

Alice

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600
3	Flore	2300

T_1 SELECT AVG(E.SAL)
FROM EMP E

1900 ←

TEMPS

T_2 INSERT INTO EMP VALUES
(3, « Flore», 2300)

T_3 SELECT AVG(E.SAL)
FROM EMP E

2000 ←

B EXECUTE DEUX FOIS LA MEME REQUETE ET OBTIENT DES RESULTATS DIFFERENTS !

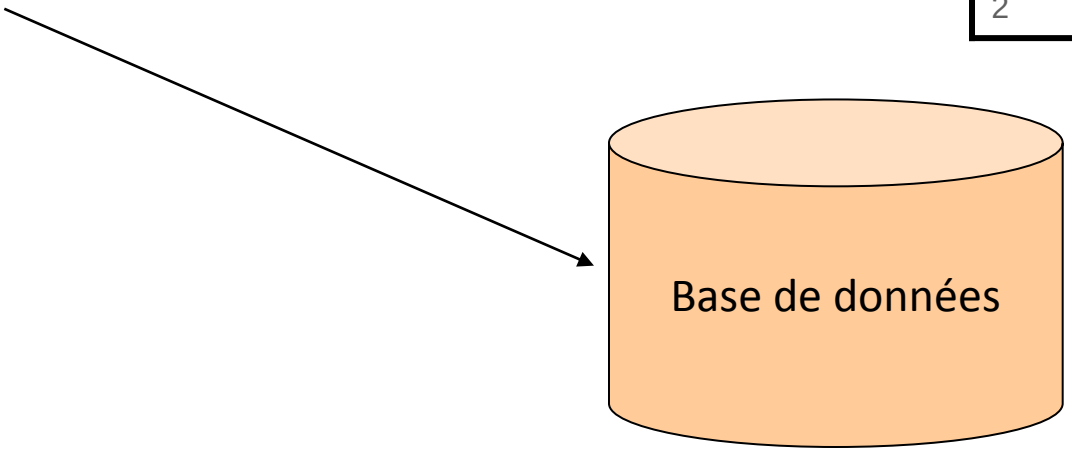
CETTE ANOMALIE EST DITE *LECTURE FANTOME.*

Ecriture de nuplets identiques

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600



Bob

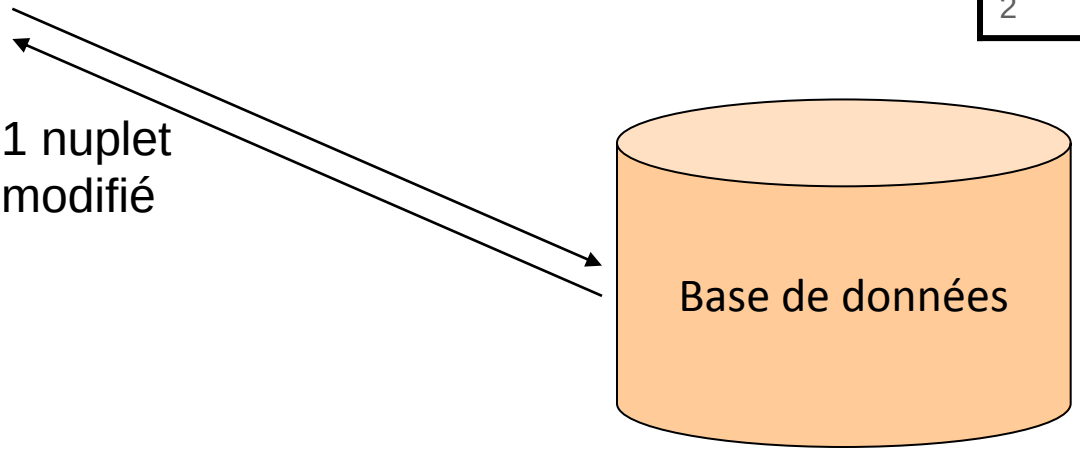
Ecriture de nuplets identiques

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

Bob



NE	NOM	SAL
0	Charlie	**2050**
1	Diana	2100
2	Eric	1600

Ecriture de nuplets identiques

Alice

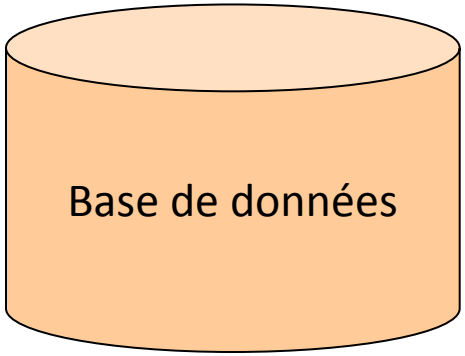
UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

1 nuplet modifié

UPDATE EMP
SET SAL = 3000
WHERE
NOM = « Charlie »

Bob



Ecriture de nuplets identiques

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

Bob

UPDATE EMP
SET SAL = 3000
WHERE
NOM = « Charlie »

1 nuplet modifié

Base de données

NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

NE	NOM	SAL
0	Charlie	**3000**
1	Diana	2100
2	Eric	1600

Ecriture de nuplets identiques

Alice

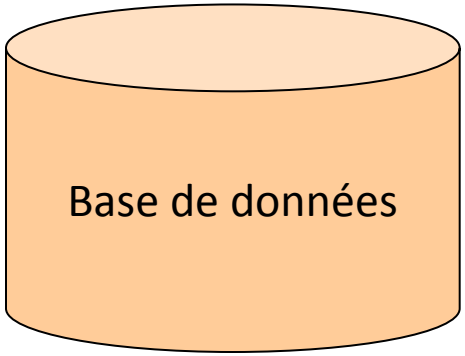
UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

NE	NOM	SAL
0	Charlie	**3000**
1	Diana	2100
2	Eric	1600

1 nuplet modifié

UPDATE EMP
SET SAL = 3000
WHERE
NOM = « Charlie »

Bob



1 nuplet modifié

SI A MODIFIE UN N-UPLET
PUIS B LE MODIFIE AUSSI
ALORS A PERD
SA MODIFICATION EN COURS
DE TRANSACTION

Tiré du diapositive de :

Du Mooc Bador

Serge Abiteboul, Benjamin Nguyen, Philippe Rigaux

Lecture impropre ou sale

T1	T2	Variale x=600
Read x 600 locale		
x=x+100= 700 locale		
Write x 700 pas encore committé		700
	Read x , 700 x=x+50=750	
	Write x	750
Fail rollback T1 est annulé		

La transaction T2 a lue la valeur de x=700 de la transaction T1 sans qu'elle ne soit validé (comit).

Perte de mise a jour

T1	T2	Variale x=100
Read x 100		
x=x+10= 110 locale		
	Read x , 100 x=x-20=80	
Write x 110		110
	Write x 80	80

1. Mise a jour faite par une transaction est écrase par une autre transaction

Sérialisation

La **capacité de sérialisation** (serializability) représente une aide à l'identification des transactions dont l'exécution est garantie pour assurer la cohérence.

Definition (Planification (schedule)) (ordonnancement)

Une séquence d'opérations d'un ensemble de n transactions T1; T2; : : : ; Tn qui préserve l'ordre des opérations dans chacune des transactions.

Definition (Planification sérielle)

Une planification ou les opérations de chaque transaction sont executées de maniére consécutive, sans aucune opération interfoliée d'autres transactions.

Planification sérielle / non sérielle

1. Définition (Planification non sérielle)

Une planification ou les opérations d'un ensemble de transactions sont exécutées de maniéré interfoliée.

1. Définition (Planification sérialisable)

Une planification non sérielle **qui produit les mêmes résultats**

qu'une exécution sérielle.

Conflit et Régles

1- Si deux transactions ne font que lire des données, elles n'entrent pas en conflit et leur ordre est sans importance.

R1(x) R2(x) R1(y)R2(Y);

2- Si deux transactions soit lisent, soit écrivent complétement des données différentes, leur ordre est sans importance.

T1 : R1(x) W1(y) et T2 : R2(z) W2(v)

3- **Si une transaction écrit dans des données et si une autre transaction lit ou écrit dans ces mêmes données, alors l'ordre de leur exécution importe.**

T1: W1(x) W1(y) T2 : R2(x) R2(y)

O1: w1(x) r2(x) R2(y) w1(y) probleme de conflit.

1. Deux opérations sont en conflits ssi

- **sont des transactions différentes**
- **sont sur le même objet ou granule (variable)**
- **au moins une des opérations est une écriture.**

- L'ordre des opérations non conflictuelles n'a pas d'effet sur l'état final de la base de données.

- Se concentrer sur l'ordre des opérations conflictuelles.

1. Deux ordonnancement sont conflictuels equivalent si et seulement si

    - il impliquent les mêmes actions des mêmes transactions et chaque pair d'actions conflictuels est ordonnée de la même maniére.

L'ordonnacement S est conflictuel serializable si:

- S est conflictuel equivalent a un ordonnancement sériel

- implique que S est serializable.

Test de conflits de la capacité de sérialisation

1. Pour une planification P, un **graphe de précédence** est un

graphe dirigé G = (N,F) construit comme suit :

1. Créer un nœud pour chaque transaction.
2. Créer une flèche dirigée **Ti → Tj** , Si Tj lit la valeur d'un élément écrit par Ti . Ex: W1 (x) R2(x) donc T1 → T2
3. Créer une flèche dirigée **Ti → Tj** , Si Tj écrit une valeur dans un élément après qu'il a été lu par Ti.
4. Créer une flèche dirigée **Ti → Tj** , si Tj écrit une valeur

dans un élément après qu'il a été écrit par Ti.

1. Si une flèche Ti → Tj existe dans le graphe de précédence pour P, alors dans toute planification sérielle S équivalente à P,

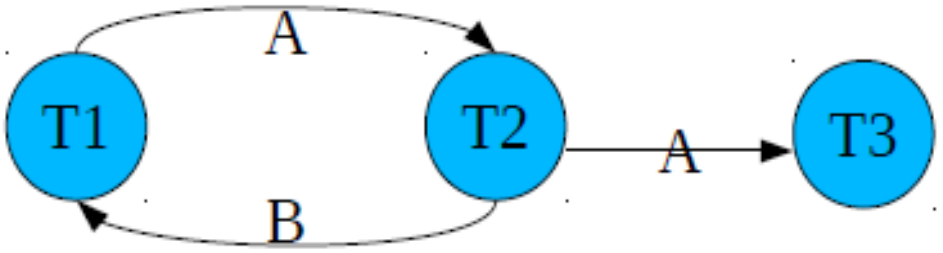
   Ti doit apparaître avant Tj .

- **Si le graphe de précédence contient un cycle, la planification n'est pas sérialisable en vue de résoudre les conflits.**

Graphe de précédence

- La notion de précédence de transactions peut être représentée par un graphe.
- Un graphe dont les noeuds représentent les transactions et dans lequel il existe un arc Ti vers Tj si Ti précède Tj dans l'exécution analysée.

{T1:Lire A;
T2: Ecrire A;
T2: Libre B;
T3: Lire A;
T1: Ecrire B}



Condition suffisante de sérialisabilité: le graphe de précédence est sans circuit.

Exemple

Un SGBD reçoit la séquence d'opérations suivantes :

O1: R1(x) R2(x) w2(y) r3(y) w3(z) w1(z) w2(x) r3(x) w3(x)

Construire le graphe de précédence de cet ordonnancement .

O1 est il sérializable?

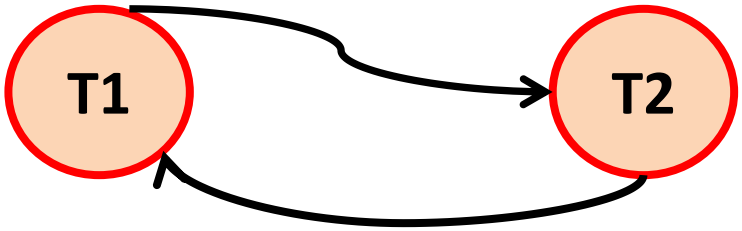
Exemple 2

O2 est un ordonnancement:

R1(x)R2(x)W1(X)R1(y)W2(x)W1(y)

O2 est non serialisable car

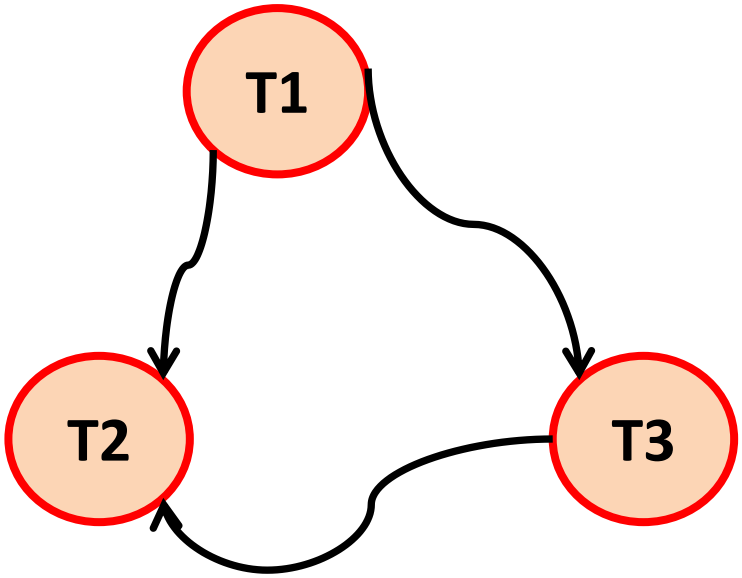
il y a un cycle



T1	T2
R(x)	
	R(x)
W(x)	
R(Y)	
	W(X)
W(y)	

Graphe de précedence

Le temps



Cet ordonnancement est serialisable car il n' y a pas de cycle

T1	T2	T3	temps
R(A)			t0
R(B)			t1
	R(A)		t2
	R(C)		t3
W(B)			t4
commit			t5
		R(B)	t6
		R(C)	t7
		W(B)	t8
		commit	t9
	W(A)		t10
	W(C)		t11
	commit		t12