Transactions et Bases de Données

Faculté des Sciences - Université Montpellier

(Référence : cours d'Isabelle Mougenot)

BD : partage de données

Garantir la cohérence des données lors de manipulations simultanées par différents utilisateurs - Gros volumes de données - Accès distribués

Les notions importantes :

1. *transaction*
2. *accès concurrent* : accès simultanés à une ressource (bd, table, tuple) qui peuvent aboutir à des conflits
3. *session* : période délimitée dans le temps pendant laquelle un client entre en communication avec un serveur (de données) afin de satisfaire ses demandes - Une session est vue ici comme une collection de transactions
4. *connexion* : gestion de l'ouverture de la session - souvent associée à un mécanisme d'identification

Schéma illustratif

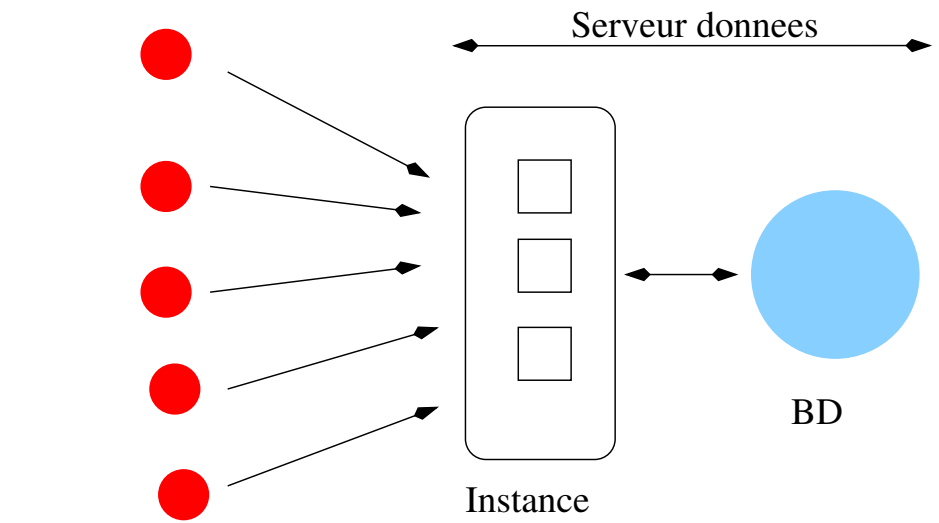

Figure: Vue d'ensemble

Domaines cibles

Exemples de domaines pour lesquels ces notions vont revêtir une importance capitale

1. *Systèmes bancaires : transferts monétaires*
2. *Systèmes de réservation : train, avion, hôtel, . . .*
3. *Centrales d'achats*
4. *Systèmes de santé*
5. *. . .*

Concept de transaction

Unité de traitement séquentiel (séquence cohérente d'actions), exécutée pour le compte d'un usager, qui appliquée à une bd cohérente, restitue une bd cohérente

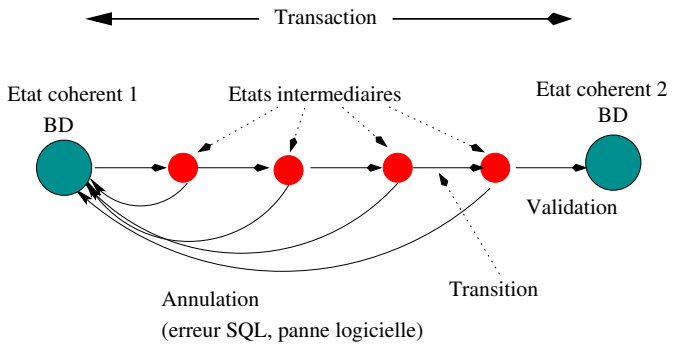

Figure: Illustration transaction

Transaction $T_i :< a_i^1, a_i^2, a_i^3, a_i^4, a_i^5, ...a_i^n >$

Voir une transaction comme un objet

Les opérations de la transaction doivent être soit exécutées en bloc, soit annulées en bloc (tout ou rien) $\Longrightarrow$ se doter d'un début et d'une fin de transaction

Marquer le début (implicite ou explicite)

- au premier ordre SQL (ouverture de la session)
- après la fin d'une transaction (validation ou annulation)
- ordre : `start transaction`, `begin transaction` ou `set transaction` ....

Fin de transaction

Marquer la fin (implicite ou explicite)

- fin explicite d'une transaction à l'aide des commandes COMMIT (validation des actions) et ROLLBACK (annulation des actions)
- fin implicite d'une transaction
  - exécution d'une commande LDD (CREATE, ALTER, RENAME et DROP) : actions exécutées depuis le début de la transaction sont validées
  - fin normale d'une session ou d'un programme avec déconnexion : la transaction est validée
  - fin anormale d'un programme ou d'une session (sortie sans déconnexion) : la transaction est annulée

Exemples d'ordres SQL

```
CREATE TABLE Compte (numC integer primary key,
typeC varchar(10),  solde float);

INSERT INTO Compte VALUES (2, 'courant', -200);
INSERT INTO Compte VALUES (3, 'courant', 500);
INSERT INTO Compte VALUES (4, 'courant', 100.50);
COMMIT;

UPDATE Compte SET solde = solde + 100 WHERE numC=3;
ROLLBACK;

DELETE FROM Compte WHERE numC=3;
ALTER TABLE Compte ADD numAg integer ;
```

Détails sur l'illustration

Actions : read (select), write (update, insert, delete), commit, roll-back

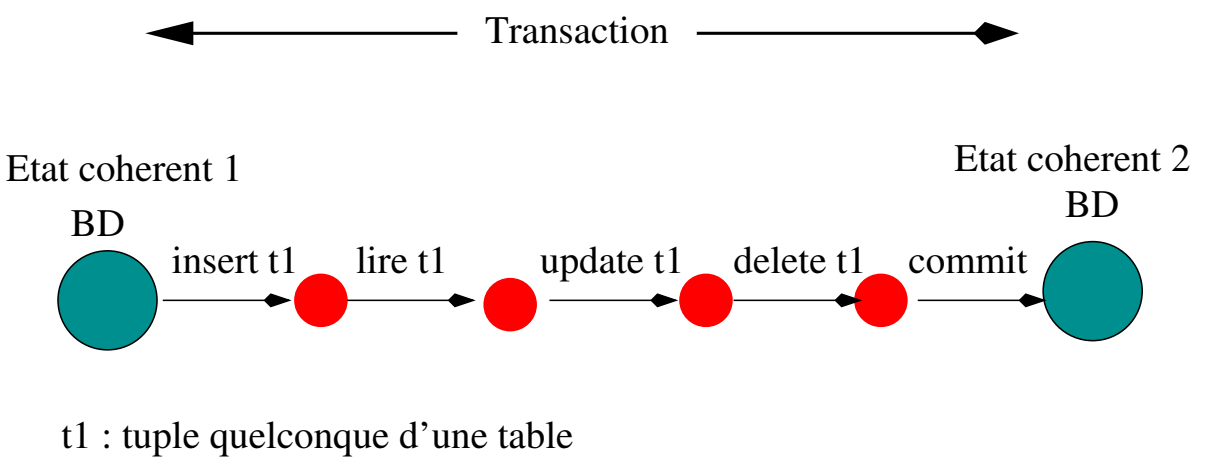

Figure: Illustration transaction

Propriétés d'une transaction

Gérer les transactions : s'assurer qu'elles possèdent les propriétés *ACID*

- **Atomicité** : lors d'une exécution d'une transaction, toutes ses actions sont exécutées ou bien aucune ne l'est.
- **Cohérence** : les modifications apportées à la b.d lors d'une transaction doivent être valides cad respecter les contraintes d'intégrité.
- **Isolation** : chaque transaction est isolée, pour s'affranchir des incohérences lors d'exécutions concurrentes
- **Durabilité ou Permanence** : les effets d'une transaction, qui s'est exécutée correctement, doivent survivre à une panne

Atomicité

Assurer que les actions même les plus complexes, englobées au sein d'une transaction, soient perçues comme une opération unique. Les usagers doivent connaître, en toute circonstance, l'état des données

Modèle général

- Tdébut
- Actions isolation, atomicité (panne $\Longrightarrow$ défaire)
- Tfin
- Validation : calcul de la validité de la transaction - certification
- Point de validation (commit)
- Permanence (panne $\Longrightarrow$ refaire éventuellement)
- Vrai fin de transaction

Validation à deux phases

La **validation à deux phases** suppose l'existence d'une mémoire stable, dans laquelle au point de validation, les nouvelles valeurs seront enregistrées.

- importance du point de validation : calculs pour accepter ou rejeter la transaction finissante (certification). segments d'annulation, mémoire redo-log
- après le point de validation, la transaction sera visible (au bout d'un certain temps) par les autres.
- transaction vivante : avant le point de validation
- transaction validée : après le point de validation

Isolation d'une transaction

H : histoire des transactions qui respecte l'ordre chronologique des actions des transactions qui s'exécutent en simultané

Exemple : pour un ensemble de transactions concurrentes notées : $\{ T_1, T_2, T_3, T_4, T_5 \}$
Ordonnancement d'un ensemble de transactions : trace chronologique des opérations a_i^j des transactions T_i
Exemple H $< a_1^1, a_2^1, a_2^2, a_1^2, a_5^1, a_1^3, a_5^2 ... a_i^n >$

Pb d'accès concurrent aux données et d'entrelacement des transactions : incohérences globales même si les transactions sont cohérentes

Accès concurrent et entrelacement des transactions

Différents problèmes liés aux accès concurrents

- Perte de mise à jour
- Lecture impropre (dirty read)
  - Lecture de données non validées
  - Lecture de donnés incohérentes
- Lecture non reproductible (non repeatable read)
- Lecture de données fantômes (phantom)

Problème de perte de mise à jour

Deux transactions en parallèle : r1(X) r2(X) w1(X) w2(X)

Temps	T_1	Etat de la base	T_2
t1	lire(X)	(X=100)	——
t2	——		lire(X)
t3	X := X+100		——
t4	——		X := X+200
t5	écrire(X)	(X=200)	——
t6	——	(X=300)	écrire(X)

Figure: Perte de mise à jour

Avec transactions séquentielles : valeur de X = 400

Problème de lecture impropre (incohérence et violation de contrainte)

Contrainte d'intégrité (CI) posée sur la base : Y=2X ; T. locales : respect CI, T. globale : violation de la CI

Temps	T_1	Etat de la base	T_2
t1	X :=10	(X=5, Y=10)	——
t2	écrire(X)	(X=10, Y=10)	——
t3	——		X := 30
t4	——	(X=30, Y=10)	écrire(X)
t5	——		Y :=60
t6	Y :=20	(X=30, Y=60)	écrire(Y)
t7	écrire(Y)	(X=30, Y=20)	——

Figure: Violation contrainte d'intégrité

Problème de lecture impropre (incohérence et violation de contrainte)

CI Y=2X ; bd cohérente mais la lecture est faussée par des écritures qui viennent s'intercaler : lecture impropre

Temps	T_1	Etat de la base	T_2
t1	lire(X)	(X=5, Y=10)	——
t2	——	(X=5, Y=10)	——
t3	——		X := 30
t4	——	(X=30, Y=10)	écrire(X)
t5	——		Y :=60
t6	——	(X=30, Y=60)	écrire(Y)
t7	lire(Y)		——

Figure: Problème sur la lecture

Problème de lecture impropre (données non validées)

Données non validées

Temps	T_1	Etat de la base	T_2
t1	——	(X=100)	——
t2	——		X := X+20
t3	——	(X=120)	écrire(X)
t4	lire(X)		——
t5	——	(X=100)	rollback

Figure: Ex. de *dirty read*

Problème de lecture non reproductible

Lecture de la même donnée qui diffère

Temps	T_1	Etat de la base	T_2
t1	lire(X)	(X=100)	——
t2	——		X := X+20
t3	——	(X=120)	écrire(X)
t4	lire(X)		——

Figure: Ex. de *non-repeatable read*

Problème de lecture de fantômes

Apparition de nouvelles données

Temps	T_1	Etat de la base	T_2
t1	lire(Table)		——
t2	——		écrire(tuple t1)
t4	lire(Table)		——

Figure: Ex. de *phantom*

Transactions sérialisables

Une exécution d'un ensemble de transactions est sérialisable ssi elle est équivalente à une exécution séquentielle (ou en série) de transactions. Quand les transactions sont arbitraires, la sériabilité est la seule à pouvoir assurer un entrelacement correct.

$< T_p(1), , T_p(2), T_p(3), T_p(4), ..., T_p(n) >$ avec p une permutation de 1, 2, 3, ..., n

Actions conflictuelles : portent sur le même objet, et une action au moins, sur les deux, est en écriture. Actions commutables A et B si l'exécution de A suivie de B est identique à l'exécution de B suivie de A

Transactions sérialisables

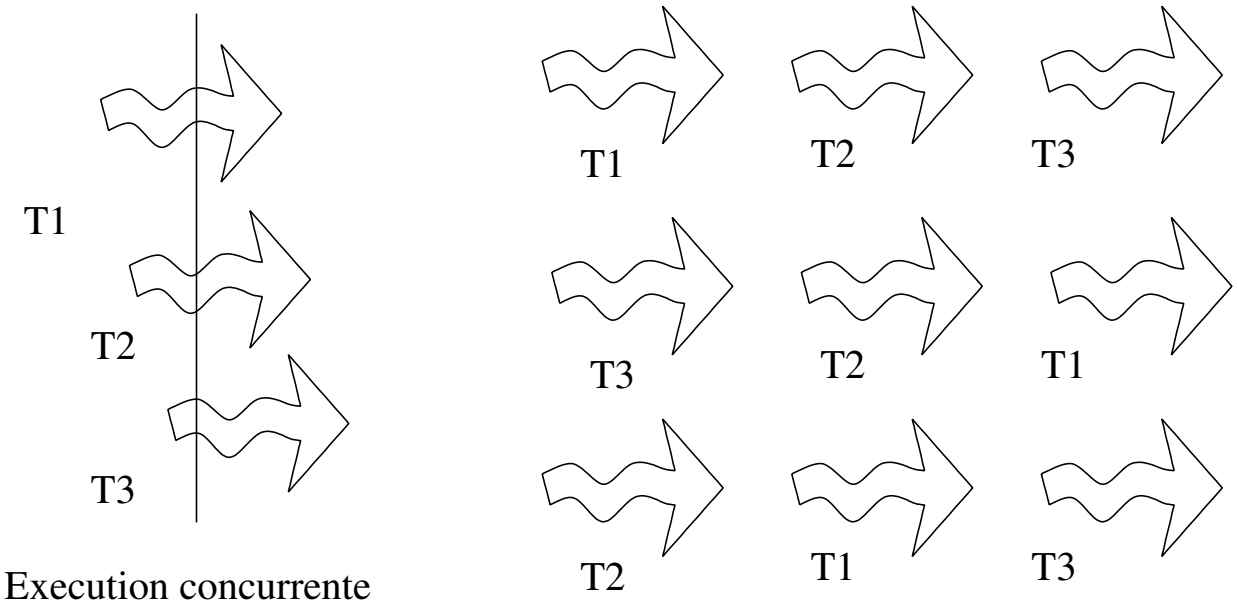

Figure: Illustration transaction

Retour sur le premier exemple

Vision sérielle des transactions

Temps	T_1	Etat de la base	T_2
t1	lire(X)	(X=100)	——
t2	X := X+100		——
t3	écrire(X)	(X=200)	——
t4	——		lire(X)
t5	——		X := X+200
t6	——	(X=400)	écrire(X)

Figure: Aucune perte de mise à jour

T2 puis T1 donnerait un résultat identique : ordonnancements équivalents

Isoler au moyen de verrous

Isoler un élément dans une transaction en le verrouillant (lock). Les verrous sont définis par deux opérations : Verrouillage à deux phases (2PL)

- verrouiller(A) (lock(A)): cette opération oblige toute transaction à attendre le déverrouillage de l'élément A si elle a besoin de cet élément
- déverrouiller(A) (unlock(A)): la transaction effectuant cette opération libère le verrou qu'elle avait obtenu sur A et permet à une autre transaction candidate, en attente, de poser, à son tour, un verrou.

Inconvénients des verrous

Situation d'attente pour d'autres transactions

- **la famine** : lorqu'un verrou est relâché sur A, le système choisit parmi les transactions candidates en attente : ordre d'entrée dans une file d'attente
- **l'interblocage (deadlock)** : Cette situation se présente lorsqu'un ensemble de transactions attendent mutuellement le déverrouillage d'éléments actuellement verrouillés par des transactions de cet ensemble.

Illustration interblocage

Deux transactions qui se bloquent mutuellement

TEMPS	T_1	T_2
t1	update(X := X+100)	——
t2	——-	update(Y := Y+300)
t3	update(Y:=800)	——
t4	——	update(X:=X-200)

Figure: Exemple interblocage

Nécessité d'un système de détection des interblocages pout lever les blocages

Exemple Interblocage en SQL

Transactions T_1 et T_2

UPDATE Compte SET solde = solde + 100 WHERE numC=3;
UPDATE Compte SET solde = solde + 100 WHERE numC=4;
UPDATE Compte SET solde = solde - 100 WHERE numC=4;
UPDATE Compte SET solde = solde - 100 WHERE numC=3;

w1(C3),w2(C4),w1(C4),w2(C3)

Verrou et granularité du verrou

La pose de verrous dégrade les performances du système et impose des temps d'attente $\Longrightarrow$: limiter les impacts des effets en donnant le choix sur l'objet à verrouiller $\Longrightarrow$ notion de granularité du verrou

- bd : collection de tables
- table : collection de tuples ou d'attributs
- page : collection d'enregistrements
- tuple : collection de couples attribut-valeur
- attribut : collection de valeurs

Granularité des objets verrouillés

La BD est découpée en granules. Les verrous peuvent porter sur ces granules en fonction de la configuration du système fixée par l'administrateur

.

Poser des verrous en anticipation : exemples de verrouillages explicites qui peuvent être exploités pour synchroniser des transactions

- SELECT * FROM Compte where num_compte = 12345 FOR UPDATE;
- LOCK TABLE Compte IN EXCLUSIVE MODE NOWAIT;
- LOCK TABLE Compte IN SHARE MODE;
- LOCK TABLE Compte IN ROW SHARE MODE;

Transactions sérialisables et verrous

Réalisation : la sériabilité impose aux transactions que tous les verrouillages précèdent tous les déverrouilages. Les transactions sont dites à deux phases : une phase d'acquisition des verrous puis une phase de libération (à la validation ou à l'annulation). Aucun granule ne reste verrouillé après la fin d'une transaction.

En complément au protocole à deux phases, deux types de verrous sont distingués :

- verrou partagé ou lâche (lecture) noté S (Shared)
- verrou exclusif ou bloquant (écriture) noté X (eXclusive)

Matrice de compatibilité

Selon les verrous imposés par les transactions, une matrice de compatibilité peut se dégager selon le type de verrou S (Verrou partagé) X (verrou exclusif)

	S	X
S	Y	N
X	N	N

Figure: Matrice de compatibilité

Isoler au moyen de niveaux d'isolation de la transaction

Quatre niveaux qui définissent le degré d'isolement de la transaction

De + en + performant mais de + en + contraignant (en terme de verrous posés)

- niveau 0 - read-uncommitted : la transaction T peut lire des objets modifiés par une autre transaction
- niveau 1 - read-committed : la transaction T lit uniquement les mises-à-jour des transactions validées (à chaque requête de T)
- niveau 2 - repeatable-read : 1 + aucun objet lu par la transaction ne peut être modifié par une autre transaction
- niveau 3 - serializable : 3 + transaction T isolée dès la 1ère requête (Si modification d'un enregistrement par transactions concurrentes, et T essaie ensuite de modifier cet enregistrement, alors erreur)

Niveaux d'isolation

Ils se distinguent par la possibilité ou l'impossibilité d'obtenir les effets non désirés des accès concurrents précédents

NIVEAU	DIRTY READ	NON-REPEATABLE	FANTOM
read-uncommitted	Y	Y	Y
read-committed	N	Y	Y
repeatable-read	N	N	Y
serializable	N	N	N

Figure: Niveaux et effets indésirables

Isoler au moyen du mode d'accès aux données

Deux possibilités

- read only : lecture autorisée, mise à jour interdite (SELECT autorisé mais INSERT, UPDATE, DELETE, SELECT FOR UPDATE interdits).
- read write : lecture et écriture autorisées (SELECT, INSERT, UPDATE, DELETE autorisés)

Syntaxe SQL pour définir une transaction

```
set transaction <option> ;
<option> ::= read only | isolation level
      <niveau-d-isolation> | read write
   | use rollback segment <rollback_segment>
<niveau-d-isolation> ::= serializable |
      repeatable read | read committed |
    read uncommitted
```

Listing 1: Exemples syntaxe

Exemples SQL pour définir une transaction

```
SET TRANSACTION READ ONLY NAME 'TransactionUn';
SET TRANSACTION READ WRITE;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET TRANSACTION USE ROLLBACK SEGMENT
    some_rollback_segment;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE;
(session level)
```

Listing 2: Transaction

Exemple read committed

T_1 READ COMMITTED	T_2 READ COMMITTED
	select solde from Compte where numC=2 ; (valeur 500)
update Compte set solde = 300 where numC=2; commit ;	
	select solde from Compte where numC=2; (valeur 300)

Figure: Exemple transaction read committed

Exemple 1 serializable

T_1 READ COMMITTED	T_2 SERIALIZABLE
	select solde from Compte where numC=2 ; (valeur 500)
update Compte set solde = 300 where numC=2; commit ;	
	select solde from Compte where numC=2; (valeur 500)

Figure: Exemple transaction serializable

Exemple 2 serializable

T_1 READ COMMITTED	T_2 SERIALIZABLE
	select solde from Compte where numC=2 ; (valeur 500)
insert into Compte (numC, solde) values (5, 300); commit ;	
	select solde from Compte where numC=5; (résultat vide)
update Compte set solde=400 where numC=2; commit ;	
	update Compte set solde=100 where numC=2 ; échec rollback

Figure: Exemple transaction serializable

Point de reprise

Découper les transactions *longues* et introduire des points de reprise (marqueurs) à partir desquels il est possible de remonter en cas de problème

```
update Compte set solde = 100 where numC = 2 ;
savepoint Compte_2 ;

update Compte set solde = -1000 where numC = 4 ;
savepoint Compte_4 ;
-- non le compte 4 mais le compte 8 dans le rouge :
rollback to savepoint Compte_2 ;
update Compte set solde = -1000 where numC = 8 ;

commit ;
```

Listing 3: Point de reprise

Permanence ou durabilité

Au point de validation, les effets d'une transaction, doivent être conservés sur la base en toute circonstance $\Longrightarrow$ fichiers journaux qui conservent la trace des transactions successives.

Synthèse des problèmes pouvant survenir

- Pb logique : erreur de syntaxe, violation de contrainte, confusion sur les objets du schéma
- Choix d'annuler la transaction
- Panne logicielle (moteur SGBD)
- Pb physique : panne machine, crash disque (mémoire secondaire), coupure courant, ...

Détails sur la répercussion des changements

Lors d'une transaction qui effectue une mise à jour sur la base : la base passe d'un ancien état à un nouvel état et le journal conserve l'identification des éléments modifiés, leur ancienne valeur et leur nouvelle valeur

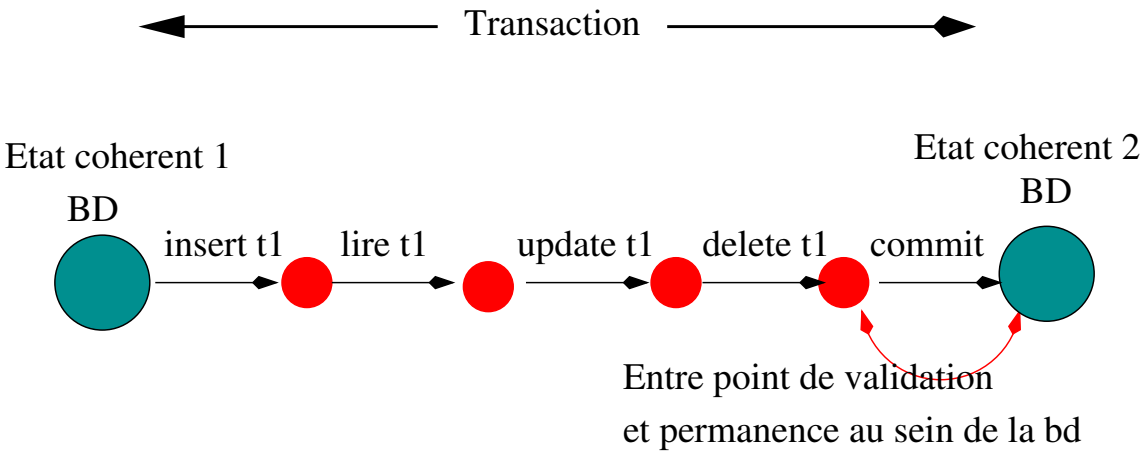

Figure: Point délicat

Vue globale

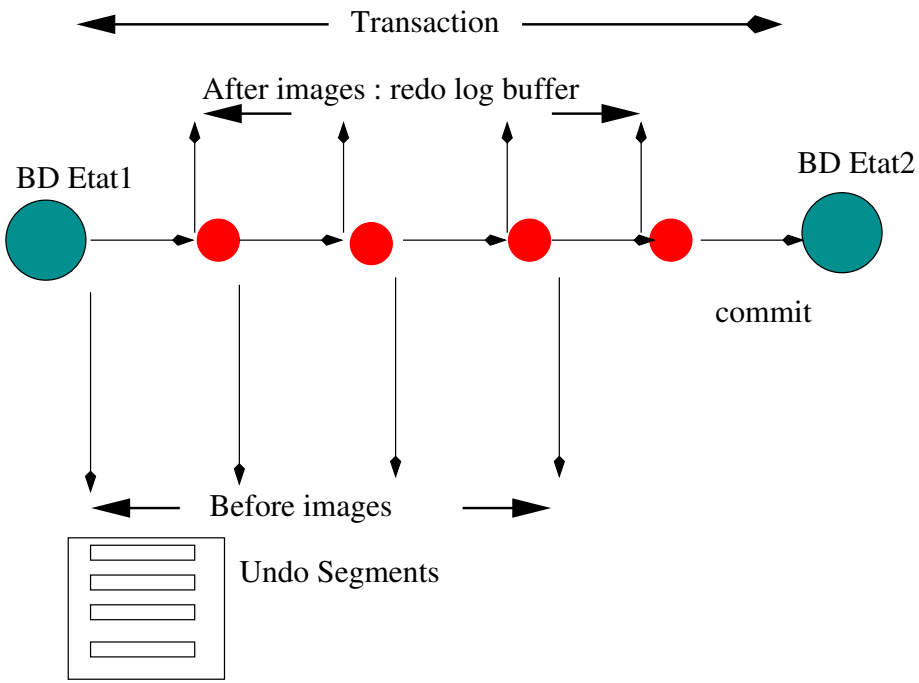

Figure: Approche générale

Architecture Oracle

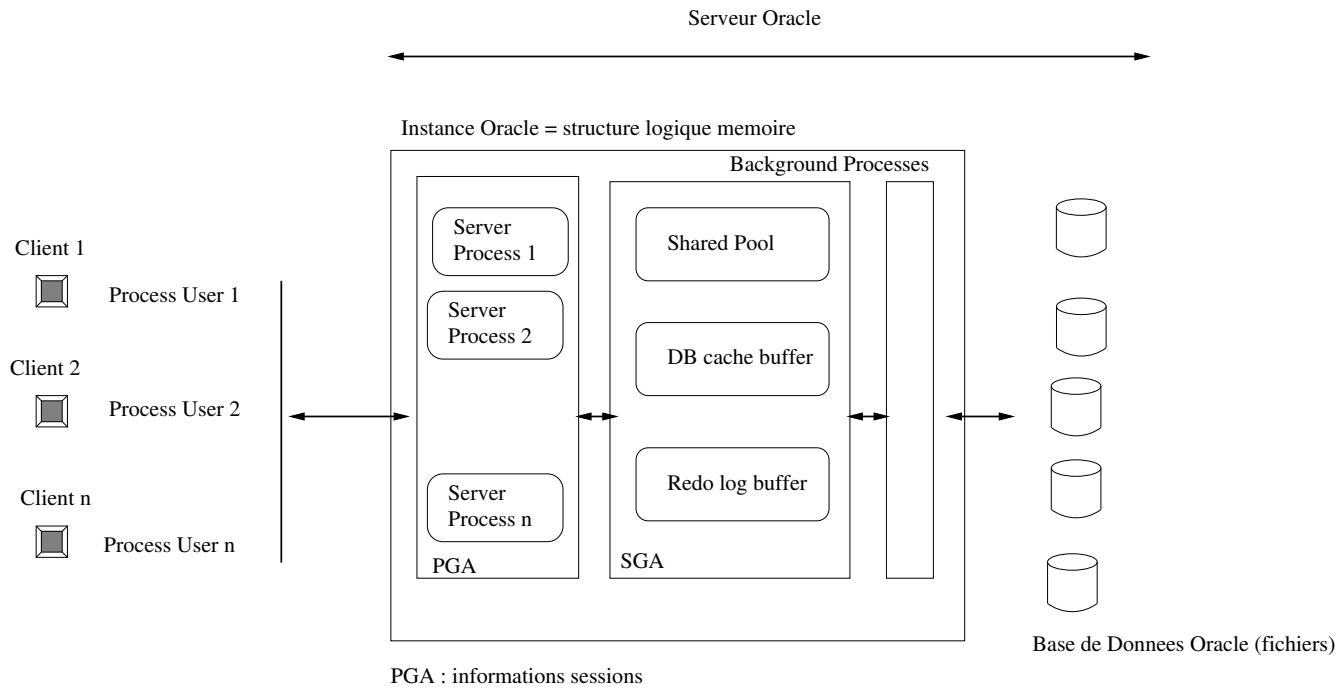

Figure: Rappel Architecture Oracle

Architecture Oracle

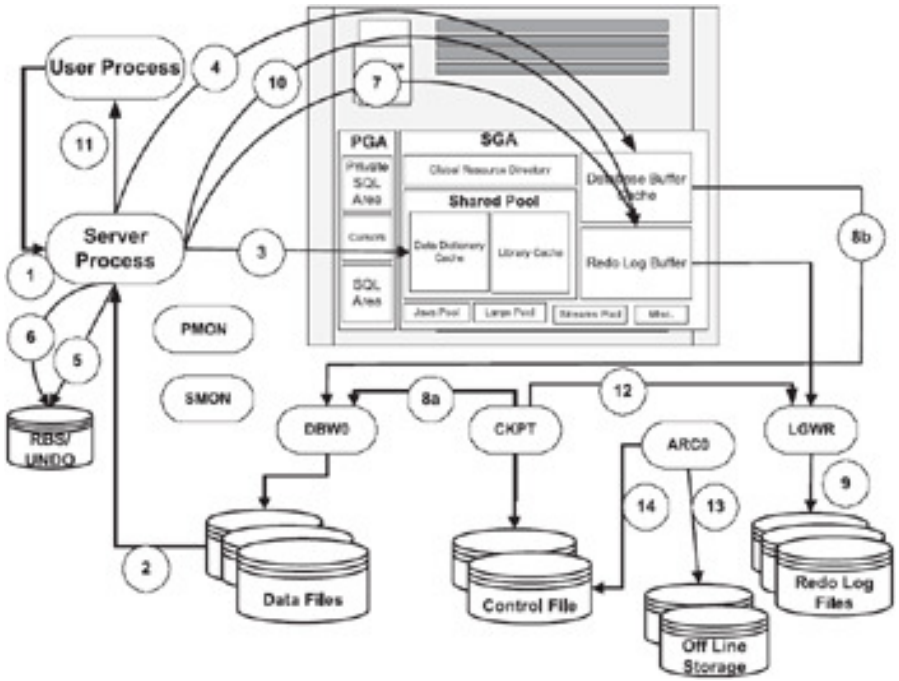

Figure: Flux d'actions au niveau d'un serveur Oracle (extrait du Web)

Détails

Action : mise à jour du solde d'un compte de 500 à 1000

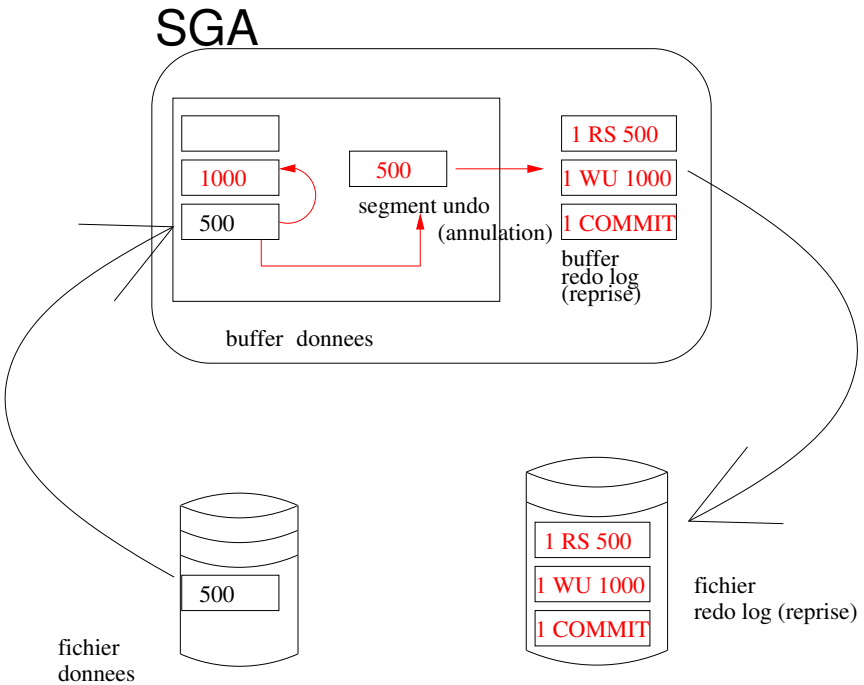

Figure: Détails de l'exécution d'une transaction

Refaire - Défaire - Corriger une panne

Stratégie Refaire - Défaire

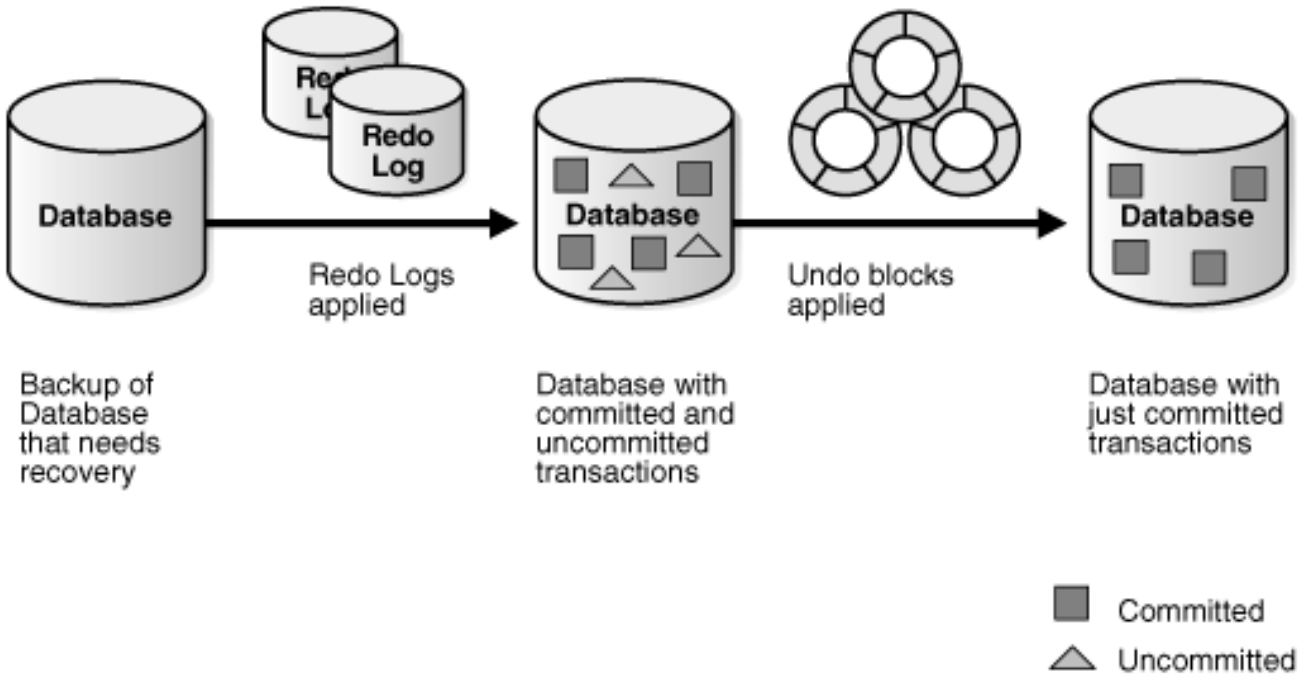

Figure: Situation de panne (Extrait doc Oracle)

Accorder des privilèges

L'ordre GRANT

```
Syntaxe :
GRANT [privilege|privilege list|ALL|EXECUTE] ON [object]
TO [schema] [WITH GRANT OPTION];

Exemples :
GRANT INSERT, UPDATE ON Compte TO user1;
GRANT ALL ON Compte TO public;
GRANT EXECUTE ON f_tranfert TO public;
```

Listing 4: GRANT

Accorder des privilèges

L'ordre GRANT

```
Grant Column Privileges
Syntaxe:
GRANT [privilege] ([column]) ON [table]
TO [schema];

Exemple:
GRANT UPDATE (solde) ON Compte TO user3;
```

Listing 5: GRANT

Retirer des privilèges

L'ordre REVOKE

```
Syntaxe :
REVOKE [privilege|privilege list|ALL|EXECUTE] ON
    [object]
FROM [schema];

Exemples :
REVOKE INSERT, UPDATE ON Compte FROM user1;
REVOKE ALL ON Compte FROM public;
REVOKE EXECUTE ON f_transfert FROM public;
```

Listing 6: REVOKE

Retirer des privilèges

L'ordre REVOKE

```
Revoke Column Privileges

Syntaxe :
REVOKE [privilege] ([column]) ON [table]
FROM [schema] [CASCADE CONSTRAINTS];

Exemples :
REVOKE UPDATE (solde) ON Compte FROM user3;
```

Listing 7: REVOKE

Vues du dictionnaire de données

Renseignements sur les privilèges accordés

```
Vue : user_tab_privs
     user_tab_privs_made
     user_tab_privs_recd

desc user_tab_privs

select grantee, table_name, grantor privilege
   from user_tab_privs;
```

Listing 8: Vues dictionnaire

Illustration

```
SQL> col grantee for a10
SQL> col grantor for a10
SQL> col privilege for a10
SQL> select grantee, grantor, table_name, privilege from user_tab_privs;

GRANTEE    GRANTOR    TABLE_NAME                     PRIVILEGE
---------- ---------- ------------------------------ ----------
THE        ISA        COMPTE                         SELECT
THE        ISA        COMPTE                         UPDATE
ISA        THE        CLIENT                         UPDATE
ISA        THE        CLIENT                         SELECT
ISA        THE        COMPTE                         UPDATE
ISA        THE        COMPTE                         SELECT

6 rows selected.
```

Figure: Illustration consultation vue méta-schéma

Privilèges systèmes

Renseignements sur les privilèges systèmes accordés

```
Vues : user_sys_privs : privileges accordes usager
       user_role_privs : roles accordes usager
       dba_roles : roles definis au niveau de la bd

 col username for a5
 col privilege for a10
 col admin_option for a5
 select * from user_sys_privs;
```

Listing 9: Vues dictionnaire

Notion de rôle

Factoriser la gestion des privilèges : définir des types d'usagers - Un usager peut endosser plusieurs rôles

```
create role M1_IC;
grant create public database link to M1_IC ;
grant create materialized view to M1_IC ;
grant M1_IC to user1 ;
alter user user1 default role M1_IC ;
```

Listing 10: ROLE