Gestion de transactions

Définition

Exemples de programmes

Propriétés des transactions

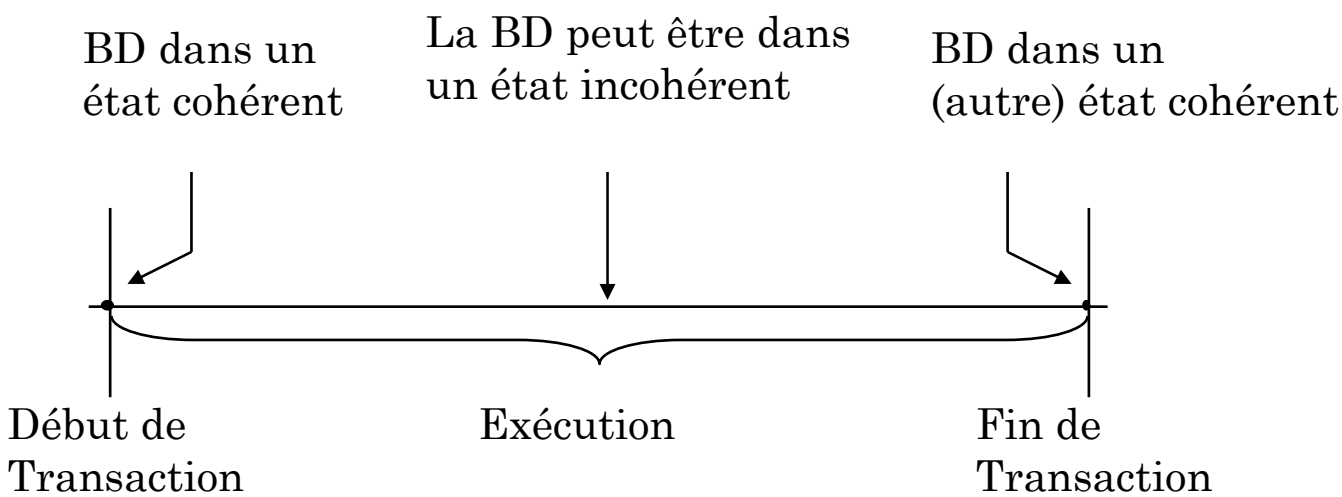
Fiabilité et tolérance aux pannes

- Journaux
- Protocoles de journalisation
- Points de reprise

Transactions

Transaction = séquence d'actions qui transforment une BD d'un état *cohérent* vers un autre état *cohérent*

- opérations de lecture et d'écriture de données de différentes granularités
- ***granules*** = tuples, tables, pages disque, etc…



Programmation

Une transaction est démarrée

- *implicitement* au début du programme ou après la fin d'une transaction (Oracle) ou
- explicitement par l'instruction begin_transaction.

Une transaction comporte :

- des opérations de lecture ou d'écriture de la BD
- des opérations de manipulation (calculs, tests, etc.)
- des *opérations transactionnelles* qui terminent la transaction :
  - Commit :
    - validation des modifications (explicite ou implicite à la fin)
  - Abort (ou Rollback):
    - annulation de la transaction : on revient à l'état cohérent initial avant le début de la transaction

Exemple de transaction simple

```
begin_transaction Budget-update
begin
  EXEC SQL UPDATE Project
     SET Budget = Budget * 1.1
     WHERE Pname = `CAD/CAM';
end . {Reservation}

/* validation (commit) implicite à la fin de
  la transaction */
```

Propriétés des transactions

ATOMICITE : Les opérations entre le début et la fin d'une transaction forment une *unité d'exécution.*

COHERENCE : Chaque transaction accède et retourne une base de données dans un état cohérent (pas de violation de contrainte d'intégrité).

ISOLATION : Le résultat d'un ensemble de *transactions concurrentes* et validées correspond au résultat d'une exécution *successive* des mêmes transactions.

DURABILITE: Les mises-à-jour des transactions validées *persistent*.

BD exemple

Considérons un système de réservation d'une compagnie aérienne avec les relations:

FLIGHT(FNO, DATE, SRC, DEST, STSOLD, CAP)

CUST(CNAME, ADDR, BAL)

FC(FNO, DATE, CNAME,SPECIAL)

Exemple de transaction de réservation

```
begin_transaction Reservation
begin
   input(flight_no, date, customer_name);
   EXEC SQL UPDATE    FLIGHT
            SET       STSOLD = STSOLD + 1
            WHERE     FNO = flight_no AND DATE =
        date;
   EXEC SQL INSERT
            INTO      FC(FNO, DATE, CNAME, SPECIAL);
            VALUES    (flight_no, date,
        customer_name, null);
   output("reservation completed")
  endif
end . {Reservation}
```

Terminaison de transaction

```
begin_transaction Reservation
begin
    input(flight_no, date, customer_name);
    EXEC SQL  SELECT     STSOLD, CAP
              INTO       temp1,temp2
              FROM       FLIGHT
              WHERE      FNO = flight_no AND DATE =  date;
    if temp1 = temp2 then
        output("no free seats");
        abort;
    else
        EXEC SQL  UPDATE  FLIGHT
                  SET     STSOLD = STSOLD + 1
                  WHERE   FNO = flight_no AND DATE = date;
        EXEC SQL  INSERT
                  INTO    FC(FNO, DATE, CNAME, SPECIAL);
                  VALUES  (flight_no, date, customer_name,
          null);
         commit;
         output("reservation completed")
     endif
end . {Reservation}
```

Transactions : Implantation

BD : a = 2; b=3;

T: x=read(a); y=read(b); write(a,y); write(b,x);

Les *transactions* fournissent des exécutions

- fiables : a=3 et b=2 après la validation d'une exécution de T (même en présence de pannes : coupure de courant, panne disque, …)
- correctes : une exécution de T échange a **et** b (commit) *ou ne fait rien* (abort)
- cohérentes : a=2 et b=3 après deux exécutions *concurrentes* et validées de T

T1: x=read(a); y=read(b); write(a,y); write(b,x);

T2: x=read(a); y=read(b); write(a,y); write(b,x);

Problème: Comment implanter/garantir ces propriétés ?

- **Gestion de pannes**
- Gestion de concurrence

Propriétés des transactions

ATOMICITE : Les opérations entre le début et la fin d'une transaction forment une *unité d'exécution*

DURABILITE: Les mises-à-jour des transactions validées *persistent*.

COHERENCE : Chaque transaction accède et retourne une base de données dans un état cohérent (pas de violation de contrainte d'intégrité).

ISOLATION : Le résultat d'un ensemble de *transactions concurrentes* et validées correspond au résultat d'une exécution *successive* des mêmes transactions.

1. Gestion de pannes
   - Cache
   - Journalisation
2. Gestion de cohérence
   - Sérialisibilité
   - Algorithmes de contrôle de concurrence

Gestion de pannes

Fiabilité : Types de pannes

Panne de transaction

- abandon normal (prévu dans le programme/incohérence logique) ou dû à un *deadlock* (conflit entre transactions)
- *pas de perte « physique » de contenu*

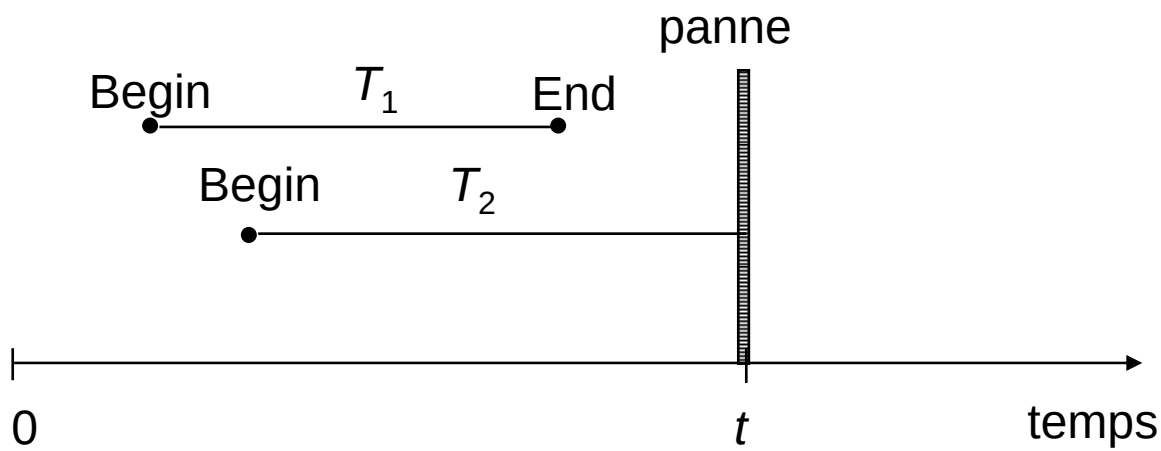
Panne processeur/mémoire

- panne de processeur, mémoire, alimentation, …
- *le contenu de la mémoire principale (programme et buffer) est perdu*

Panne disque

- panne de tête de lecture ou du contrôleur disque
- *des données de la BD sur disque sont perdues*

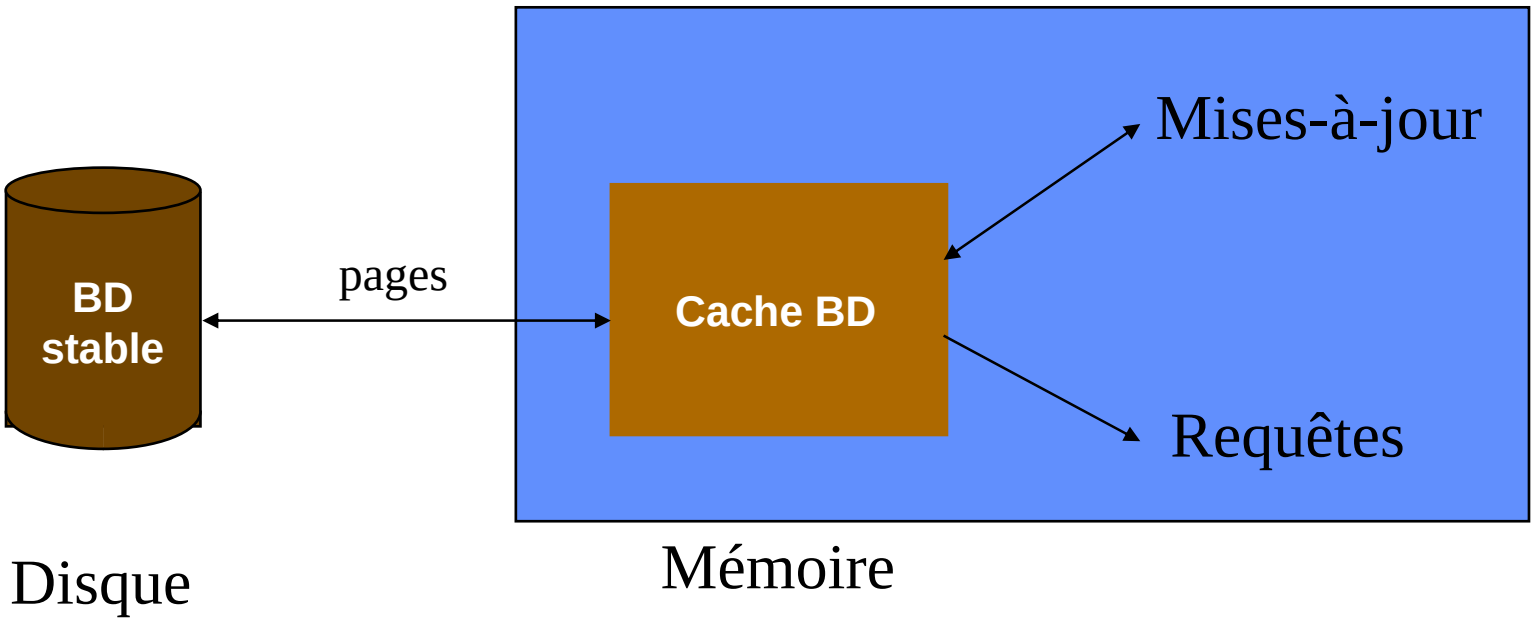
Gestion de pannes



Lors de la panne

- toutes les mises-à-jour de T_1 doivent perdurer (durabilité)
- aucune mise-à-jour de T_2 ne doit être faite dans la BD (atomicité)

Lecture/écriture BD



Cache BD : sert à augmenter la performance du système.

Mises-à-jour d'une base de données

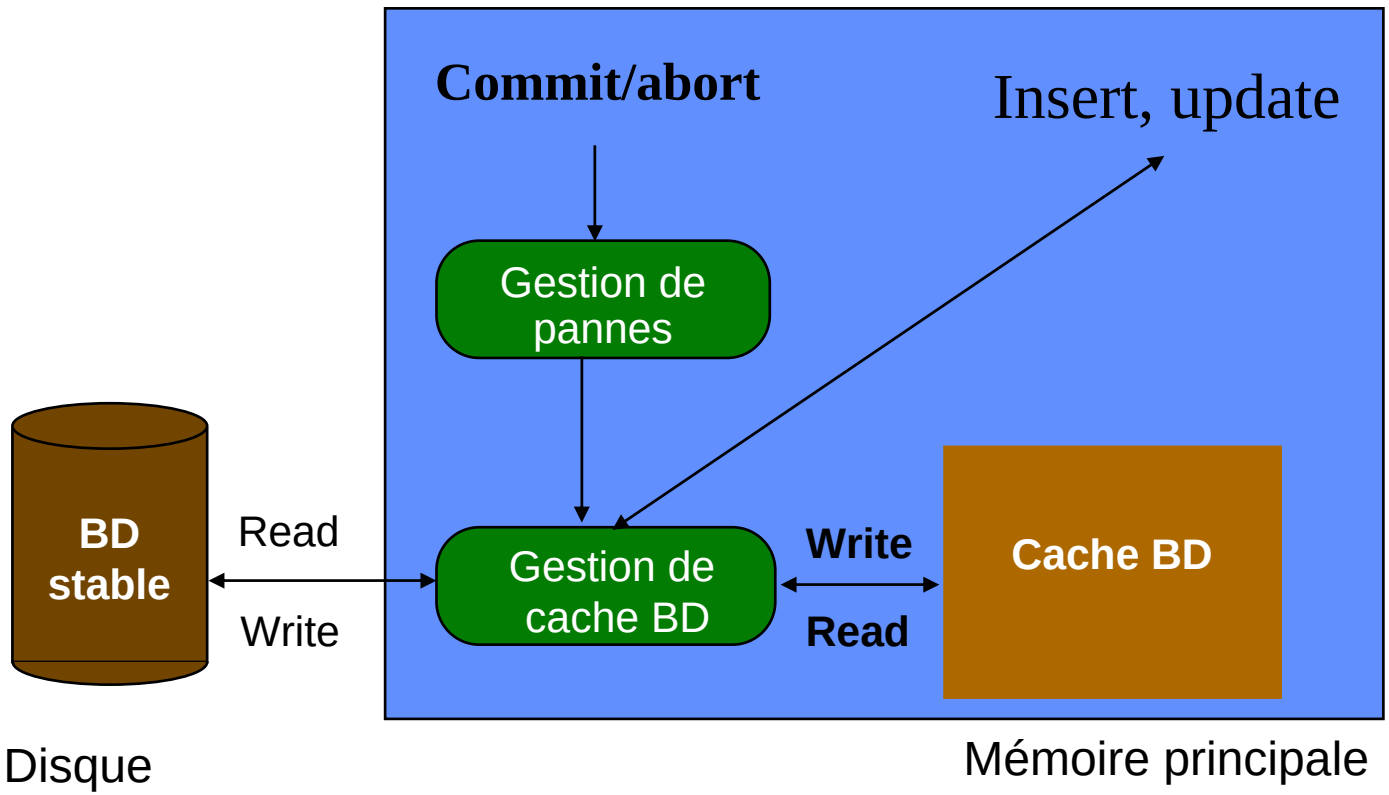
Mise-à-jour « *en place* » :

- chaque mise-à-jour cause la modification de données dans des pages du cache BD
- *l'ancienne valeur est écrasée* par la nouvelle

Mise-à-jour « *hors-place* » :

- les nouvelles valeurs de données sont écrites séparément des anciennes dans des pages ombres qui remplacent les pages d'origine au moment du commit.
- peu utilisée en pratique car *très coûteuse :*
  - *fragmentation des données sur disque*
  - nécessite la mise-à-jour d'index (même quand la clé ne change pas)

Gestion de pannes



Problème du cache BD

Cache BD :

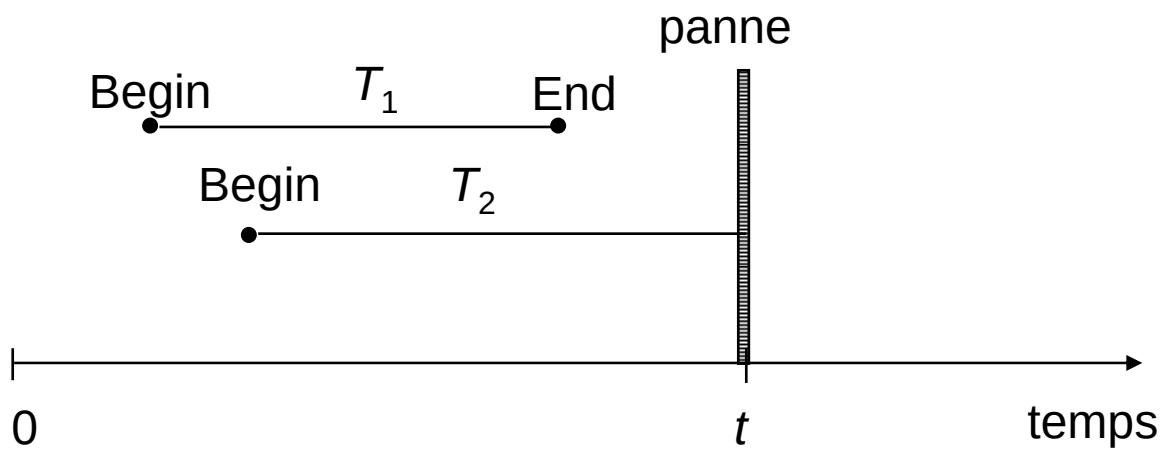
- sert à augmenter la performance du SGBD en évitant les lectures/écritures disque
- une page du cache peut contenir des données validées et non-validées

Problème : Comment garantir la *durabilité* et *l'atomicité*

- *sans forcer l'écriture* des données validées (commit) sur disque (non-force)
  - REDO : il faut garantir que les modifications de *transaction validées* seront prises en compte après une panne
- *sans empêcher d'écrire* des données non-validées sur disque (steal) :
  - UNDO : il faut être capable d'annuler des modifications de *transactions annulées*

Solution : maintenir un journal des mises-à-jour

Pourquoi journaliser?

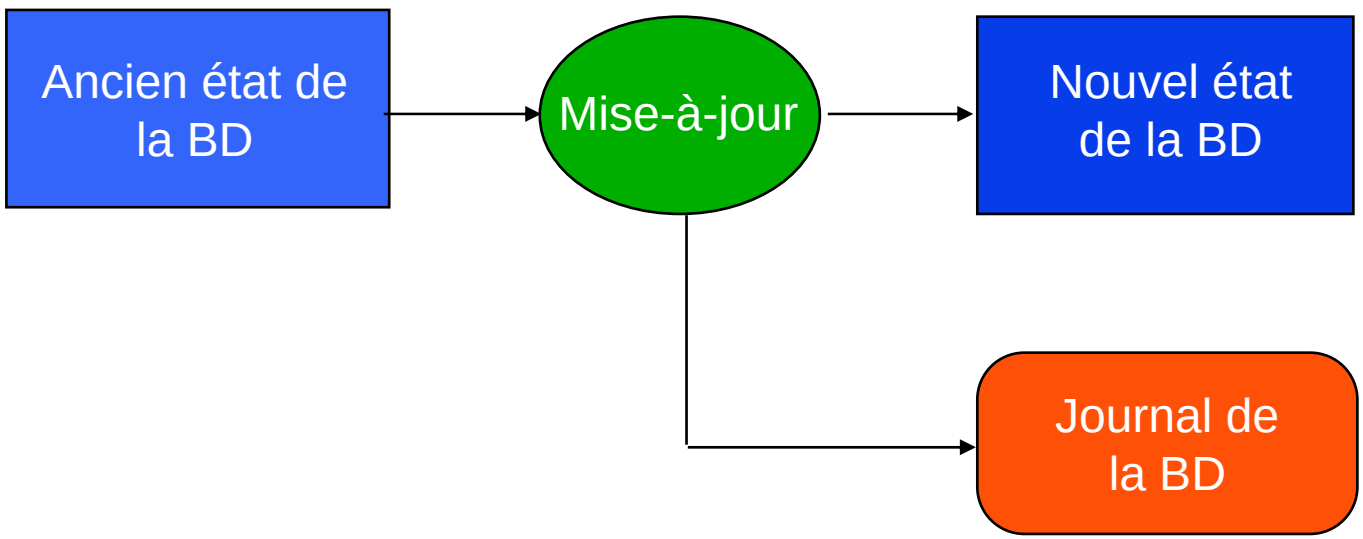


Après la reprise :

- toutes les mises-à-jour de T_1 doivent être faites dans la BD (REDO)
- aucune mise-à-jour de T_2 ne doit être faite dans la BD (UNDO)

Atomicité : Journal de la BD

Chaque action d'une transaction est enregistrée dans le journal qui est un fichier séquentiel répliqué sur des disques différents de la BD (un crash disque ne doit pas détruire le journal et les données!) :



Journalisation

Le journal contient les informations nécessaire à la restauration d'un état cohérent de la BD

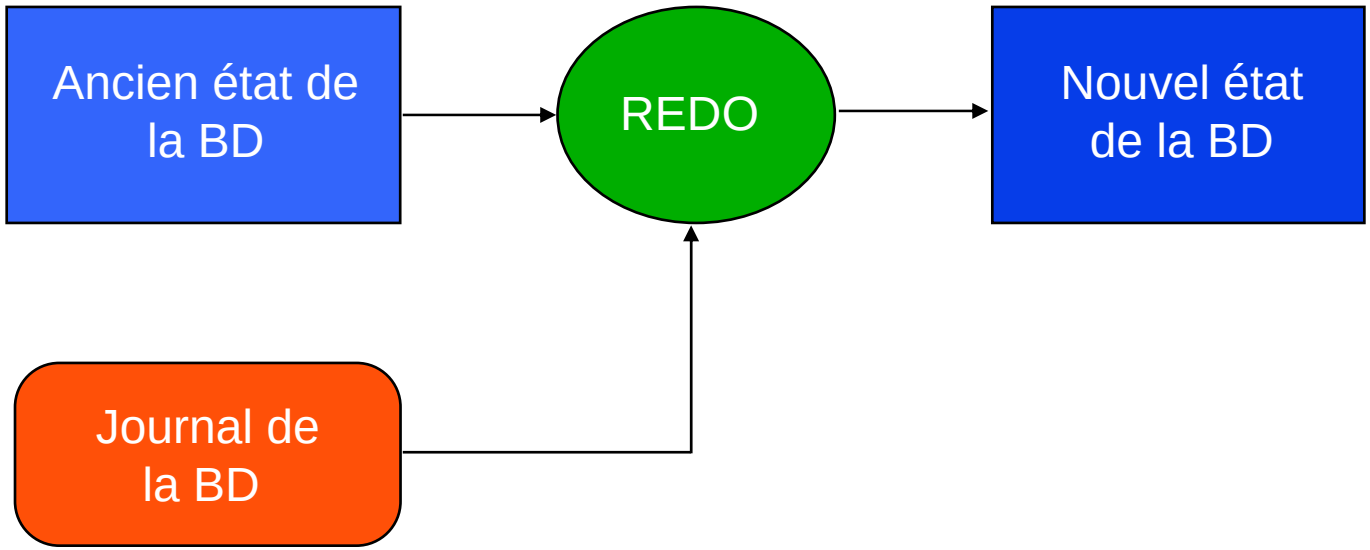
- identifiant de transaction
- type d'opération (action)
- granules accédés par la transaction pour réaliser l'action
- ancienne valeur de granule (image avant : UNDO)
- nouvelle valeur de granule (image après : REDO)
- …

Exemple de journal

Début du journal ⟶

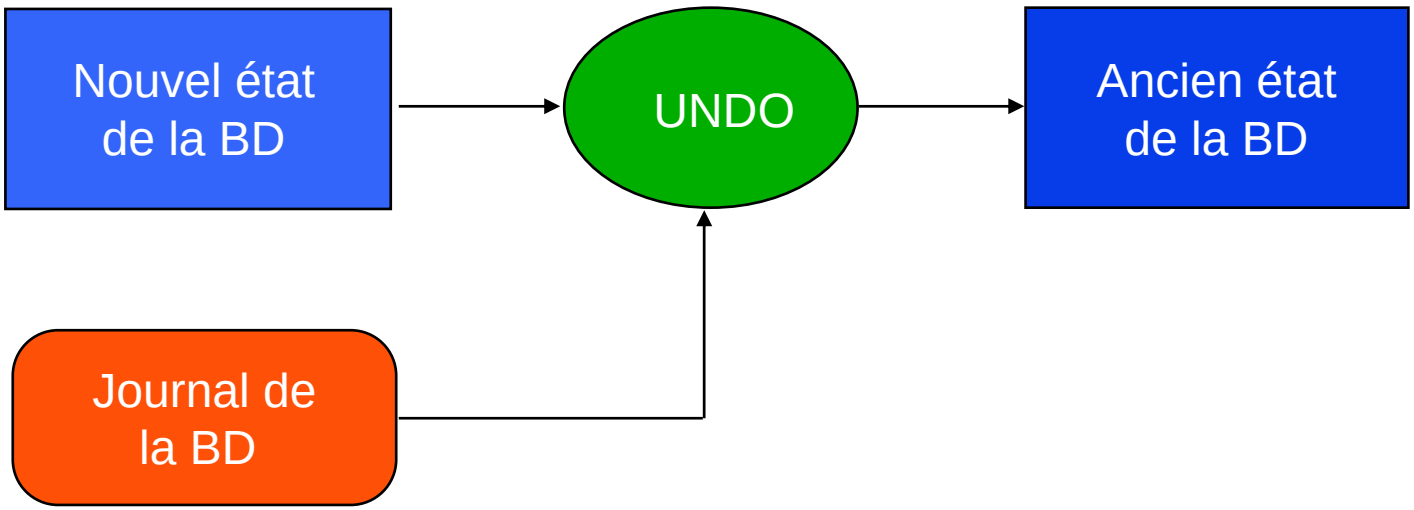
T_1, begin
T_1, x, 99, 100
T_2, begin
T_2, y, 199, 200
T_3, begin
T_3, z, 51, 50
T_2, w, 1000, 10
T_2, commit
T_4, begin
T_3, abort
T_4, y, 200, 50
T_5, begin
T_5, w, 10, 100
T_4, commit

Protocole REDO



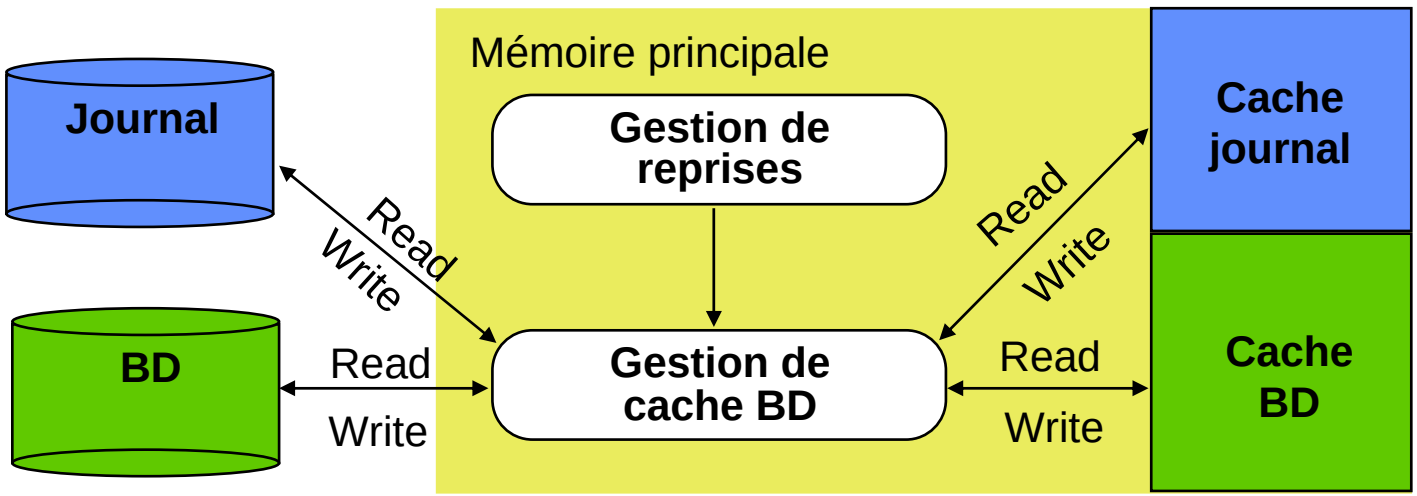
L'opération REDO utilise l'information du journal (image après) pour refaire les actions qui ont été exécutées ou interrompues.

Protocole UNDO



L'opération UNDO utilise l'information du journal (image avant) pour restaurer l'image avant du granule.

Interface du journal



Quand écrire le journal sur disque?

Supposons une transaction *T* qui modifie la page *P*

Cas chanceux :

- le système écrit *P* dans la BD sur disque
- le système écrit le journal sur disque pour cette opération
- PANNE!... (avant la validation de *T*)

Nous pouvons reprendre (undo) en restaurant *P* à son ancien état grâce au journal

Cas malchanceux :

- le système écrit *P* dans la BD sur disque
- PANNE!... (avant l'écriture du journal)

Nous ne pouvons pas récupérer car il n'y a pas d'enreg. avec l'ancienne valeur dans le journal

Solution: le protocole Write-Ahead Log (WAL)

Protocole WAL

Deux observations :

- si la panne précède la validation de transaction, alors toutes ses opérations doivent être défaites, en restaurant les images avant
- dès qu'une transaction a été validée, certaines de ses actions doivent pouvoir être refaites, en utilisant les images après

Protocole WAL:

- avant d'écrire une page « sale» P (qui peut contenir des modifications d'une transaction non terminée) dans le cache sur disque, la partie *undo* du journal concernant P doit être écrite sur disque.
- avant la validation d'une transaction T, toute la partie *redo* du journal des pages modifiées doit être écrite sur disque (force du log).

Points de reprise

Point de reprise : enregistrement de toutes les modifications d'une liste de transactions actives pour réduire la quantité de travail à refaire ou ou défaire lors d'une panne

Pose d'un point de reprise:

- écrire `begin_checkpoint` dans le journal
- écrire *les buffers du journal et de la BD sur disque*
- écrire `end_checkpoint` dans le journal

Procédures de reprise

Reprise à chaud :

- perte de données en mémoire, mais pas sur disque
- à partir du dernier point de reprise, déterminer les transactions
  - validées : REDO
  - non validée : UNDO
- Variante *ARIES* : refaire *toutes* les transactions et défaire les transactions non terminées au moment du crash

Reprise à froid :

- perte de données sur disque
- à partir de la dernière **sauvegarde** et du dernier point de reprise
  - REDO des transactions validées
  - UNDO inutile

Conclusion

- La gestion de pannes garantit la *durabilité* et *l'atomicité* des transactions.
- Elle ne doit pas trop pénaliser la performance du système (trop de lectures et écritures de disque).
- La gestion de concurrence entre transactions est *indépendante* de la gestion des pannes.