

Керування трансакціями

Відновлення

Трансакції

- Трансакція – логічна одиниця роботи БД
-

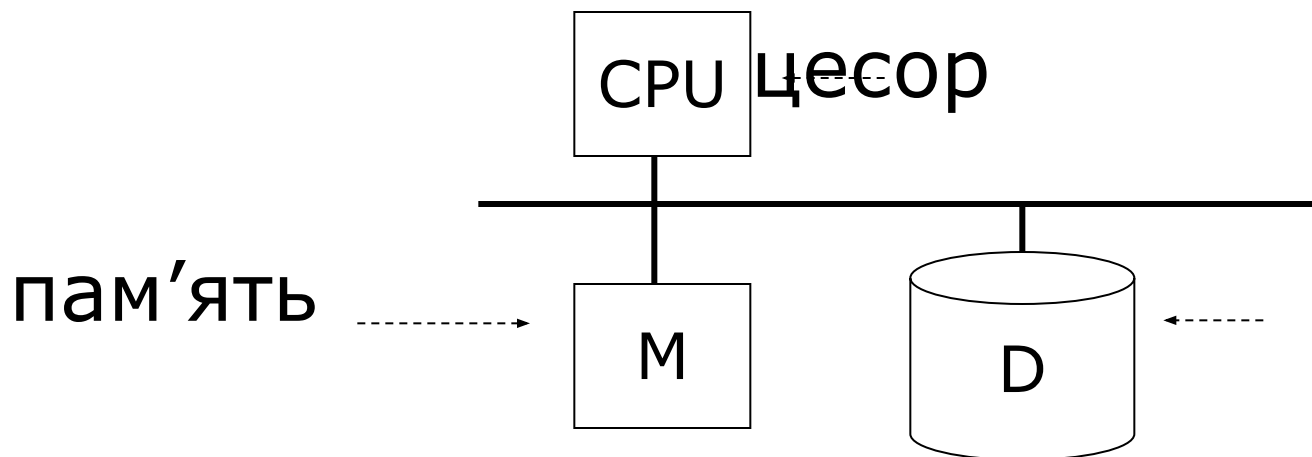
Властивості транзакцій (ACID)

- ❑ **Атомарність (Atomicity)**. Транзакції атомарні – виконується все або нічого.
 - ❑ **Узгодженість (Consistency)**. Транзакції переводять базу даних з одного узгодженого стану в інший узгоджений стан.
 - ❑ **Ізольованість (Isolation)**. Транзакції ізольовані одна від одної.
 - ❑ **Довговічність (Durability)**. Якщо транзакція успішно закінчена, виконані нею дії оновлень даних зберігаються в БД постійно, навіть якщо в наступний момент відбудеться збій.
-

Режими відмов

Події — Бажані
 Небажані Очікувані
 Неочікувані

Модель системи



-
- Бажані події: дивись документацію...
 - Небажані очікувані події:
 - Системний збій
 - Втрата вмісту пам'яті
 - Зависання -> перезавантаження
 - Пошкодження носія
 - Небажані неочікувані події
 - Все інше!
-

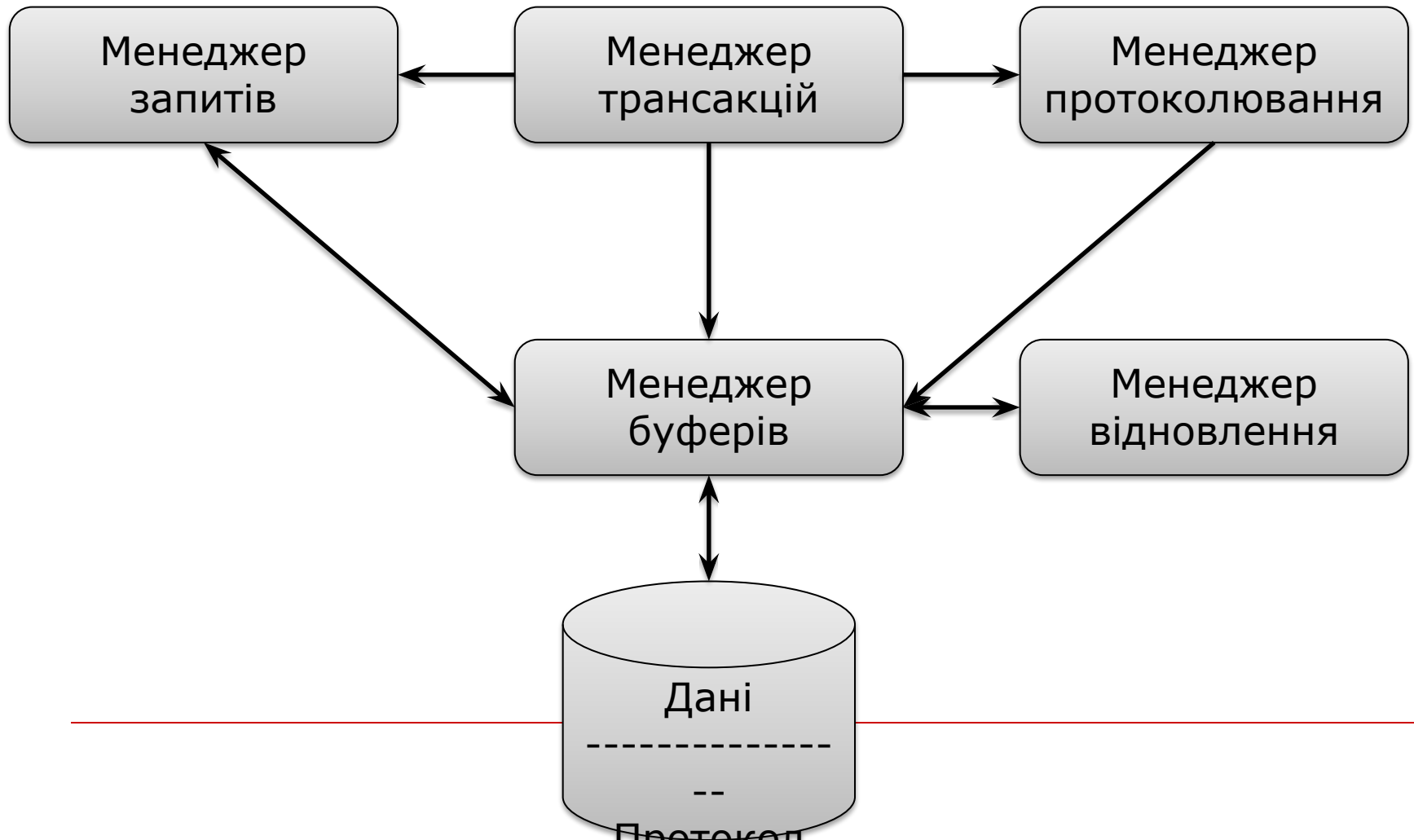
Режими відмов

- Помилкові елементи даних
 - Пошкодження носія
 - Катастрофа
 - Збій системи
-

Менеджер транзакцій

- Відповідальність за коректну поведінку транзакцій покладається на **менеджер транзакцій**.
-

Менеджер транзакцій



Базові операції трансакцій

- В процесі виконання трансакція має справу з трьома адресними просторами:
 - простір **дискових блоків**, що містять елементи даних БД;
 - простір **віртуальної або оперативної пам'яті**, що керується менеджером буферів;
 - власний **локальний адресний простір** трансакції.
-

Базові операції транзакцій

- **Input (X)**: дисковий блок, що містить значення X копіюється в буфер оперативної пам'яті.
 - **Output (X)**: копіювати блок з елементом даних X на диск
 - **Read (x,t)**: копіювати елемент X в локальну змінну t транзакції
 - **Write (x,t)**: копіювати X значення локальної змінної t в буфер пам'яті, що відповідає елементу X бази даних
-

Ключова проблема

- Незакінчені трансакції

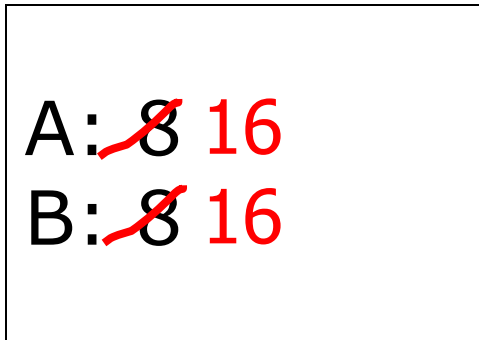
Наприклад Constraint: $A=B$

$$T_1: A \leftarrow A \times 2$$

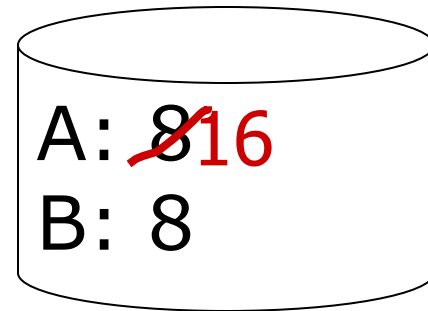
$$B \leftarrow B \times 2$$

```
T1: Read (A,t); t ← t×2
     Write (A,t);
     Read (B,t); t ← t×2
     Write (B,t);
     Output (A);
     Output (B);
```

збій!



пам'ять



ДИСК

Вирішення проблеми

- Потрібна **атомарність**
 - виконати всі дії транзакції або не виконати жодної

Рішення – протоколювання

- Протокол, чи журнал – це **послідовність** так званих **записів** протоколу, кожний із яких несе в собі інформацію про певну дію, що виконується трансакцією.
 - В протоколі можуть перемішуватись записи декількох трансакцій.
-

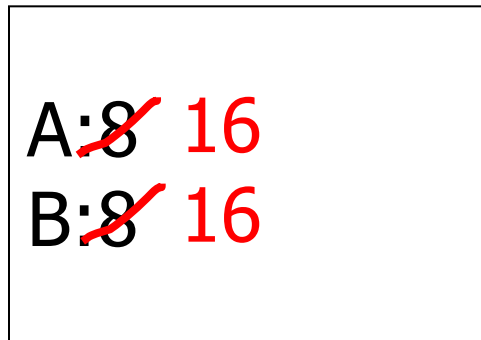
Відновлення з використання протоколу

- Для усунення згубних наслідків збою системи одні транзакції будуть виконані повторно, і зміни, виконані ними й раніше збережені в БД зафіксуються заново. Іншим транзакціям прийдеться відмінити свої дії, наче ці транзакції взагалі й не активувалися.
-

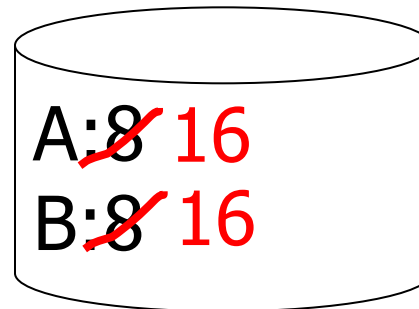
Undo logging

T₁: Read (A,t); t ← t×2
Write (A,t);
Read (B,t); t ← t×2
Write (B,t);
Output (A);
Output (B);

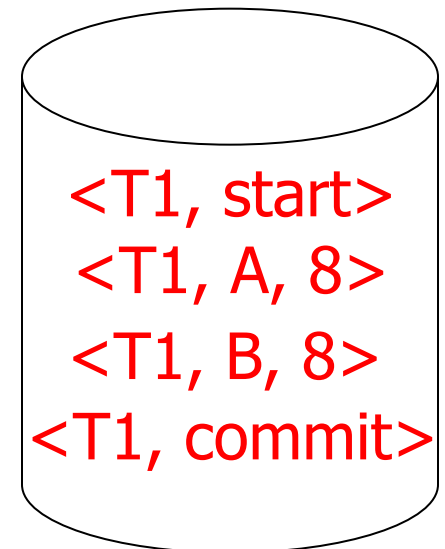
A=B



memory



disk



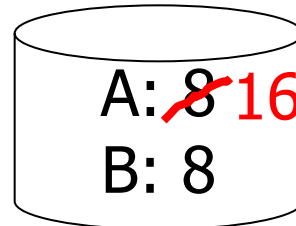
log

“Ускладнення”

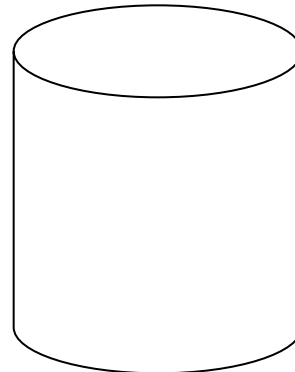
- Журнал пишеться спочатку в пам'ять
- Не пишеться на диск при кожній дії memory

A: ~~8~~ 16
B: ~~8~~ 16
Log:
<T₁, start>
<T₁, A, 8>
<T₁, B, 8>

DB



Log



BAD STATE
1

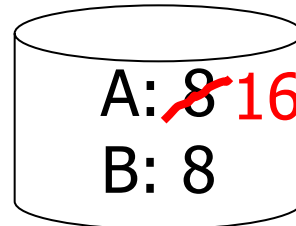
“Ускладнення”

- Журнал пишеться спочатку в пам'ять
- Не пишеться на диск при кожній дії

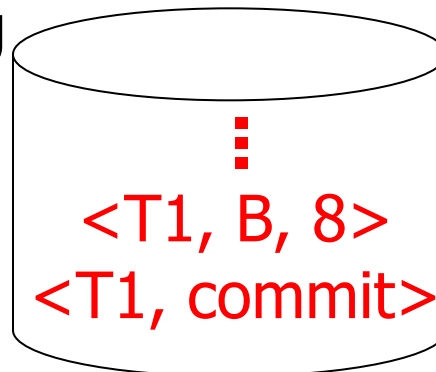
memory

```
A: 8 16  
B: 8 16  
Log:  
<T1,start>  
<T1, A, 8>  
<T1, B, 8>  
<T1,  
commit>
```

DB



Log



BAD STATE
2

Правила протоколювання в режимі Undo

1. Для кожної дії створювати запис журналу, що містить старе значення елементу даних
2. U_1 Якщо транзакція T змінює елемент x , запис журналу виду $\langle T, x, v \rangle$ повинен бути занесений в протокол **до** збереження нового значення на диску (write ahead logging:WAL)
3. U_2 При фіксації результатів транзакції T запис $\langle T, \text{commit} \rangle$ слід заносити в протокол тільки після «скидання» всіх змінених елементів БД на диск, причому інтервал між дисковими операціями і записом $\langle T, \text{commit} \rangle$ повинен бути як найменшим.

Відновлення: протокол Undo

- Перша задача менеджера відновлення полягає в розподілі всіх транзакцій, згаданих в журналі, по двом категоріям – зафіксовані й незафіксовані.
 - Менеджер відновлення повинен здійснювати сканування журналу, починаючи з кінця і рухатись до початку.
-

Відновлення: протокол Undo

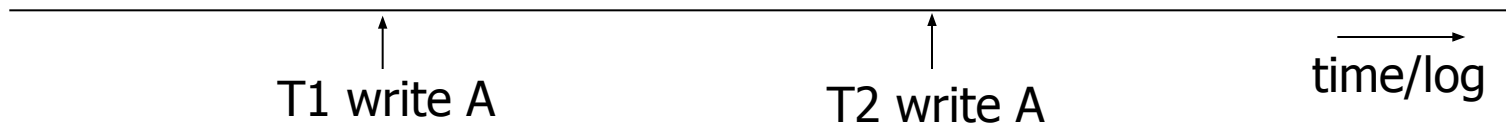
- Для кожної T_i із записом $\langle T_i, \text{start} \rangle$ в журналі:
 - If $\langle T_i, \text{commit} \rangle$ or $\langle T_i, \text{abort} \rangle$ in log, do nothing
 - Else For all $\langle T_i, X, v \rangle$ in log:
 - write (X, v)
 - output (X)
 - Write $\langle T_i, \text{abort} \rangle$ to log
-

Відновлення: протокол Undo

- (1) Дано $S =$ множина транзакцій в журналі з $\langle T_i, \text{start} \rangle$, але без $\langle T_i, \text{commit} \rangle$ (або $\langle T_i, \text{abort} \rangle$), тобто незакінчені
 - (2) Для кожної $\langle T_i, X, v \rangle$ в журналі, в зворотному порядку (останні \rightarrow ранні) виконати:
 - if $T_i \in S$ then $\left\{ \begin{array}{l} - \text{write } (X, v) \\ - \text{output } (X) \end{array} \right.$
 - (3) Для кожної $T_i \in S$ виконати
 - записати $\langle T_i, \text{abort} \rangle$ в журнал
-

Питання

- Чи можна зробити запис $\langle T_i, \text{abort} \rangle$ (крок 3) в будь-якій послідовності?
 - Наприклад: T1 і T2 обидві пишуть A
 - T1 виконується перед T2
 - T1 і T2 обидві rolled-back
 - $\langle T1, \text{abort} \rangle$ записаний, а $\langle T2, \text{abort} \rangle$ - ні



Питання

- Що буде коли відбудеться збій під час виконання відновлення?
-

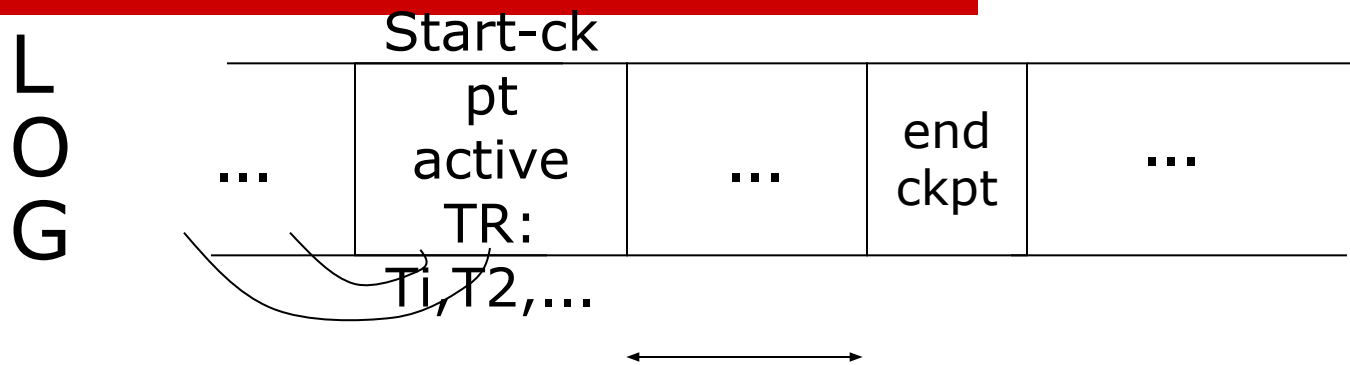
Введення контрольних точок

- Для введення контрольної точки системі потрібно:
 1. Призупинити прийом запитів на активізацію нових транзакцій.
 2. Дочекатися, поки всі діючі транзакції не виконають операції фіксації чи переривання і не занесуть в журнал записи `<T, commit>` або `<T, abort>`.
 3. Здійснити «скидання» журналу на диск командою `flush log`.
 4. «Скинути» всі буфери на диск
 5. Занести в протокол запис виду `<СКРТ>`.
 6. Відновити прийом запитів.
 - Недоліки?
-

Динамічне введення контрольних точок

1. Внести в протокол запис $\langle \text{START СКРТ } T_1 \dots T_n \rangle$ і через команду `flush log` «скинути» протокол на диск; $T_1 \dots T_n$ - всі активні транзакції на момент введення контрольної точки
 2. Дочекатися моменту фіксації або переривання всіх транзакцій з $T_1 \dots T_n$ не забороняючи можливості старту нових транзакцій.
 3. При завершенні всіх транзакцій з $T_1 \dots T_n$ зберегти в протоколі запис $\langle \text{END СКРТ} \rangle$ і виконати `flush log`.
-

Динамічні контрольні точки



чекаємо
завершення
транзакцій

Відновлення з використанням контрольних точок

1. Якщо спочатку менеджеру відновлення зустрівся запис виду $\langle \text{END СКРТ} \rangle$, це свідчить про те, що всі незавершені транзакції почалися після процедури введення контрольної точки, тобто знаходяться після запису $\langle \text{START СКРТ } T_1 \dots T_n \rangle$. Тому достатньо здійснити сканування до запису $\langle \text{START СКРТ } T_1 \dots T_n \rangle$.
-

Відновлення з використанням контрольних точок

2. Якщо першим зустрівся запис $\langle \text{START СКРТ } T_1 \dots T_n \rangle$, значить, збій відбувся в період виконання процедури введення контрольної точки. Незавершеними будуть тільки ті транзакції із числа $T_1 \dots T_n$, які не встигли здійснити фіксацію своїх результатів до виникнення ситуації відмови.
-

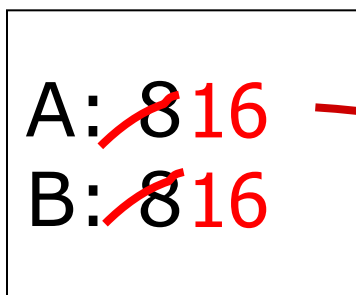
<T1, START>
<T1, A, 5>
<T2, START>
<T2, B, 10>
<START CKPT T1, T2>
<T3, START>
<T1, D, 20>
<T1, COMMIT>
<T3, E, 25>
<T2, COMMIT>
<END CKPT>
<T3, F, 30>

Протоколювання в режимі REDO

1. Для кожної дії створювати запис журналу, що містить нове значення елемента даних
 2. R_1 Перш ніж транзакція змінить X на дискові, необхідно внести в протокол всі записи оновлення X , включаючи запис оновлення $\langle T, x, v \rangle$ і запис фіксації $\langle T, commit \rangle$
-

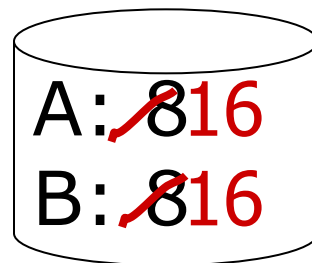
Протоколювання в режимі REDO

T₁: Read(A,t); t ← t×2; Write (A,t);
Read(B,t); t ← t×2; Write (B,t);
Output(A); Output(B)

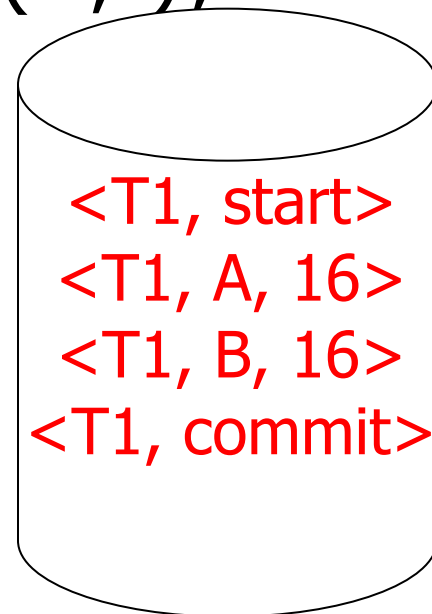


memory

output



DB



LOG

Відновлення: протокол REDO

1. Ідентифікувати всі завершені транзакції
 2. Сканувати протокол в напрямку від початку до кінця. Зустрівши запис $\langle T, x, v \rangle$:
 - ігнорувати його, якщо T – незавершена
 - зберегти на дискові значення v для елемента x , якщо транзакція T завершена.
 3. Для кожної незавершеної транзакції T зберегти в журналі запис $\langle T, abort \rangle$
 4. Виконати команду flush log
-

Відновлення: протокол REDO

- Для кожної T_i з $\langle T_i, \text{commit} \rangle$ в журналі:
 - Для всіх $\langle T_i, X, v \rangle$ в журналі:
 - Write(X, v)
 - Output(X)
-

Відновлення: протокол REDO

1. Дано $S =$ множина транзакцій з $\langle T_i, \text{commit} \rangle$ в журналі
 2. Для кожної $\langle T_i, X, v \rangle$ в журналі, в порядку появи (ранні \rightarrow останні) виконати:
 - if $T_i \in S$ then $\left\{ \begin{array}{l} \text{Write}(X, v) \\ \text{Output}(X) \end{array} \right.$
-

Введення контрольних точок

1. Внести в журнал запис виду `<START СКРТ T1...Tn>`, де - $T_1 \dots T_n$ всі активні (незафіксовані) транзакції і виконати «скидання» протоколу на диск командою `flush log`.
 2. Зберегти на дискові значення всіх елементів бази даних, змінених в буферах пам'яті, але не записаних на диск тими транзакціями, які на момент включення в журнал запису `<START СКРТ...>` виявилися завершеними.
 3. Зберегти в протоколі запис `<END СКРТ>` і виконати `flush log`
-

<T1, START>
<T1, A, 5>
<T2, START>
<T1, COMMIT>
<T2, B, 10>
<START CKPT T2>
<T2, C, 15>
<T3, START>
<T3, E, 25>
<END CKPT>
<T2, COMMIT>
<T3, COMMIT>

Відновлення: протокол REDO

1. Якщо спочатку менеджеру відновлення зустрівся запис виду `<END СКРТ>`, це свідчить про те, що всі зміни, внесені транзакціями, які закінчилися до відповідного запису `<START СКРТ $T_1 \dots T_n$ >` записані на диск і їх відновлювати не потрібно.
-

Відновлення: протокол REDO

- Але будь-яка транзакція зі списку $T_1 \dots T_n$, можливо ще не встигла записати дані на диск, навіть якщо в журналі є запис $\langle T, \text{commit} \rangle$. Тому для відновлення потрібно розглянути транзакції зі списку $T_1 \dots T_n$ або ті, які стартували після процедури введення контрольної точки.
-

Відновлення: протокол REDO

2. Якщо першим зустрівся $\langle \text{START СКРТ } T_1 \dots T_n \rangle$. В такому випадку неможливо гарантувати, що всі транзакції, завершені на момент внесення $\langle \text{START СКРТ } T_1 \dots T_n \rangle$, змогли зберегти свої оновлення даних на диск. Тому треба перейти до попереднього $\langle \text{END СКРТ} \rangle$, знайти відповідний йому запис $\langle \text{START СКРТ } S_1 \dots S_m \rangle$, а потім повторити збереження результатів оновлень всіх зафіксованих транзакцій S_j зі або тих, які стартували після внесення а протокол $\langle \text{START СКРТ } S_1 \dots S_m \rangle$
-

Протоколювання: undo/redo

- Режим undo вимагає, щоб змінені дані зберігалися на диску безпосередньо по закінченню транзакції.
 - Режим redo передбачає зберігання модифікованих блоків даних в буферах до моменту, поки транзакція не буде зафіксована і записи протоколу не будуть "скинуті" на диск.
-

Протоколювання: undo/redo

- Перш ніж транзакція T зможе модифікувати елемент X бази даних на диску, необхідно записати в журнал на диску запис оновлення **$\langle T_i, X_{id}, New\ X\ val, Old\ X\ val \rangle$**
-

Відновлення: undo/redo

1. Повторити операції всіх зафіксованих транзакцій в порядку від ранніх до останніх.
 2. Відмінити результати всіх незавершених транзакцій в зворотному порядку.
-

Введення контрольних точок

1. Внести в протокол запис виду <START СКРТ T1...Tn> і виконати "скидання" протоколу на диск командою flush log.
 2. Зберегти на диску інформацію всіх буферів, що містять "бруд", тобто змінені значення, не "скинуті" на диск.
 3. Зберегти в журналі запис <END СКРТ> і виконати команду flush log
-

<T1, START>
<T1, A, 4, 5>
<T2, START>
<T1, COMMIT>
<T2, B, 9, 10>
<START CKPT T2>
<T2, C, 14, 15>
<T3, START>
<T3, E, 20, 25>
<END CKPT>
<T2, COMMIT>
<T3, COMMIT>
