

## 10. 회복과 병행제어

### 03. 병행제어

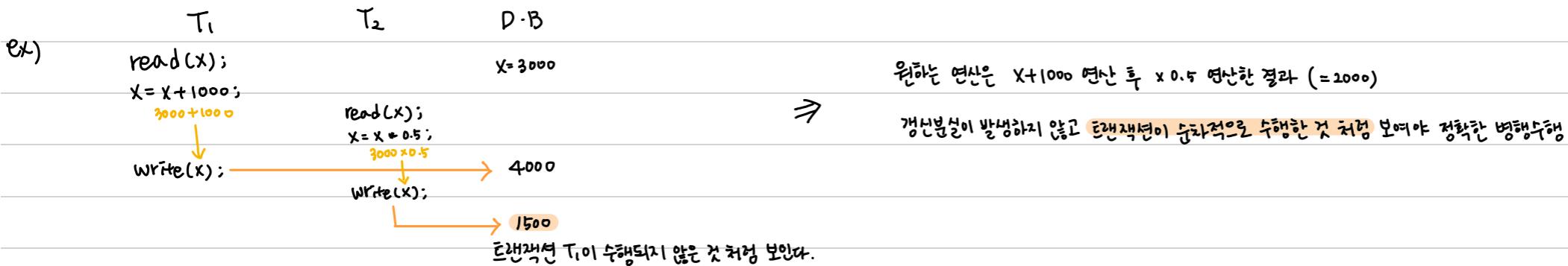
DBMS는 여러 사용자가 데이터베이스를 동시에 공유할 수 있도록 여러 트랜잭션이 동시에 수행되는 병행수행 지원  $\Rightarrow$  차례로 번갈아 수행되는 인터리빙 방식

하지만 병행 수행되는 트랜잭션이 동시에 같은 데이터에 접근, 변경 연산 수행하면 문제!

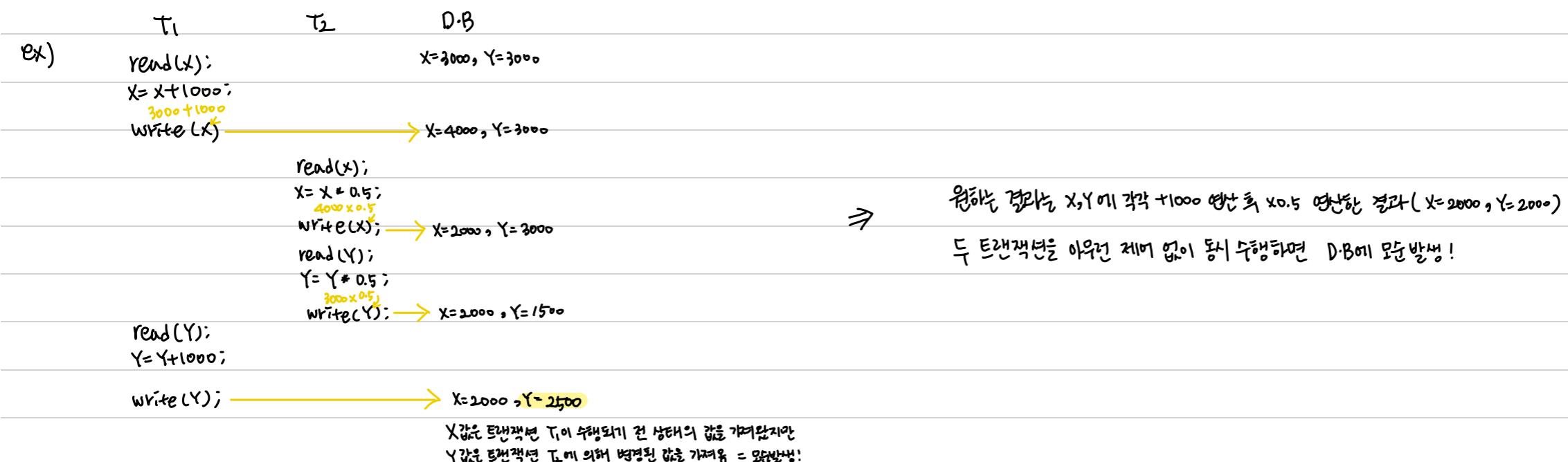
$\rightarrow$  해결! 같은 데이터에 접근하여 연산을 실행하더라도 정확한 결과를 얻을 수 있도록 트랜잭션 수행을 제어 = 병행제어 or 동시성 제어

#### ⓧ 병행수행의 문제

① 깨신분실 : 하나의 트랜잭션이 수행한 데이터 변경 연산의 결과를 다른 트랜잭션이 덮어쓰면 변경 연산이 유효화 되는 것



② 모순성 : 하나의 트랜잭션이 여러 개의 데이터 변경 연산을 실행할 때 일관성 없는 데이터베이스에서 데이터를 가져와 연산을 실행함으로써 모순된 결과가 발생하는 것



③ 연쇄복구 : 트랜잭션이 완료되기 전 장애가 발생하여 rollback 연산을 수행하면, 변경 연산을 실행한 다른 트랜잭션에도 rollback 연산을 연쇄적으로 실행해야 한다.

ex) 트랜잭션  $T_1$ 이 rollback 되면 트랜잭션  $T_2$ 도 rollback 되어야 하는데  $T_2$ 가 이미 완료된 트랜잭션이라면 rollback 불가능

### ⓧ 트랜잭션 스케줄

여러 트랜잭션을 병행 수행할 때 트랜잭션의 연산 순서 중요  $\rightarrow$  트랜잭션 스케줄은 트랜잭션에 포함되어 있는 연산들을 수행하는 순서

#### ① 직렬 스케줄

인터리빙 방식을 사용하지 않고 트랜잭션 별로 연산들을 순차적으로 실행시키는 것 = 독립적으로 수행, 병행 수행이라 할 수 없다.

Ex)  $T_1$        $T_2$       D.B

read(X);  
 $X = X + 1000;$   
write(X)

read(Y);  
 $Y = Y + 1000;$   
write(Y);

$X = 3000, Y = 3000$

$X = 4000, Y = 4000$   $T_1$ 이 수행된 후

read(X);  
 $X = X * 0.5;$   
write(X);  
read(Y);  
 $Y = Y * 0.5;$   
write(Y);

$X = 4000 * 0.5$

$Y = 4000 * 0.5$

$X = 2000, Y = 2000$   $T_2$ 가 수행된 후

#### ② 비직렬 스케줄

인터리빙 방식 사용, 하나의 트랜잭션이 완료되기 전에 다른 트랜잭션 연산이 실행될 수 있다.

Ex)  $T_1$        $T_2$       D.B

read(X);  
 $X = X + 1000;$   
 $3000 + 4000$

$X = 3000, Y = 3000$

read(X);  
 $X = X * 0.5;$   
 $3000 * 0.5$

write(X);  
 $X = 1500, Y = 3000$

read(Y);  
 $Y = Y + 1000;$   
 $3000 + 1000$

write(Y);  
 $X = 1500, Y = 4000$

read(Y);  
 $Y = Y + 1000;$   
 $3000 + 1000$

write(Y);  
 $X = 1500, Y = 5000$

$\Rightarrow$  모두가 있는 비직렬 스케줄도 가능하지만 열의 예외처럼 변경된 값이 언제 D.B에 반영되는가 (write 연산)에 따른 오류 발생 가능  
즉, 정확한 결과를 보장할 수 없다.

write(Y);  
 $X = 1500, Y = 5000$

$T_2$  수행 후

### ③ 직렬 가능 스케줄

- 직렬 스케줄에 따라 수행한 것과 같이 정확한 결과를 생성하는 비직렬 스케줄 ≠ 직렬 스케줄
- 인터러빙, 병행 제어 기법 사용

### ⓧ 병행제어 기법

트랜잭션을 병행 수행하면서도 정확한 결과를 얻을 수 있는 직렬 가능성성을 보장받기 위해 사용

#### ① 로깅기법

- 트랜잭션들이 동일한 데이터에 동시 접근할 수 없도록 lock, unlock 연산을 사용해 제어
- 한 트랜잭션이 연산을 모두 마칠 때 까지 해당 데이터에 다른 트랜잭션이 접근할 수 없도록 상호배제하여 직렬 가능성 보장

lock 연산 : 사용할 데이터에 대한 독점권 획득

unlock 연산 : 데이터에 대한 독점권 반납

- 순서: lock 연산 수행(독점권 획득) → read/write 연산 → unlock 연산  
↳ read/write 연산 전 반드시 수행  
↳ lock 연산을 수행한 트랜잭션만 가능

• 로깅 단위가 커질수록 병행성 ↓, 제어 쉽다 ↔ 로깅 단위가 작아질수록 병행성 ↑, 제어 어렵다.

• 데이터를 변경하는 write 연산 전에는 반드시 독점권을 가져야 하지만, 데이터를 읽어오기만 하는 read 연산은 동시에 실행되어도 문제 없다.

공용 lock : 해당 데이터에 read 연산 가능, write 연산 불가능

ex) T<sub>1</sub>      T<sub>2</sub>      D.B

전용 lock : 해당 데이터에 read, write 연산 불가능 → 양립불가

lock(X);  
read(X);  
 $X = X + 1000;$

X=3000, Y=3000

write(X); → X=4000, Y=3000

lock(X);  
read(X);  
 $X = X * 0.5;$

1) + 먼저 → X ⇒ X=2000, Y=2000

2) X 먼저 → + ⇒ X=2500, Y=2500

write(X); → X=2000, Y=3000

unlock(X);

lock(Y);

read(Y);

$Y = Y * 0.5;$

스테이터 결과가 같은 직렬 스케줄에 업으로 직렬 가능성 X

⇒ 이유! T<sub>1</sub>이 데이터 X에 빨리 unlock 연산 수행 → T<sub>2</sub>가 일관성 없는 데이터에 접근

lock(Y); → X=2000, Y=1500

read(Y);

$Y = Y + 1000;$

write(Y); → X=2000, Y=2500

unlock(Y);

## ② 2단계 로깅 규약

[ 확장단계 ] : 트랜잭션이 lock 연산만 실행 가능, unlock 실행 불가

[ 축소단계 ] : 트랜잭션이 unlock 연산만 실행 가능, lock 실행 불가

ex)  $T_1$        $T_2$       DB

**확장단계** `lock(X);`  
 $\downarrow$   
**LOCK연산만 가능**  
`read(X);`  
 $X = X + 1000;$

$X=3000, Y=3000$

**축소단계** `unlock(X);`  
 $\downarrow$   
**unlock 연산만 가능**  
`write(X);`  $\rightarrow X=4000, Y=3000$   
`lock(Y);`  $\quad \quad \quad$  **X에 대한 풍질권 뺏기**  
`unlock(X);`  $\quad \quad \quad$  **Y에 대한 풍질권 획득**

$X$  테이터 처리

**확장단계** `lock(X);`  
 $\downarrow$   
**LOCK연산만 가능**  
`read(X);`  
 $X = X * 0.5;$   
`write(X);`  $\rightarrow X=2000, Y=3000$   
`read(Y);`  
 $Y = Y + 1000;$

$\Rightarrow$  상대가 특정하고 있는 테이터에 unlock 연산이 실행되기를 서로 기다리면서 트랜잭션 수행을 중단하고 있는 "교착상태"에 빠질수 있다.  
 미리 예방 or 빠르게 탈지

**축소단계** `lock(Y);`  
 $\downarrow$   
**unlock 연산만 가능**  
`unlock(X);`  
`read(Y);`  
 $Y = Y * 0.5;$   
`write(Y);`  $\rightarrow X=2000, Y=4000$   
`unlock(Y);`

$Y$  테이터 처리