데이터베이스

동시성 제어

학습목표

- 다중 사용자 환경을 위한 동시성 제어의 필요성을 설명할 수 있다.
- 동시성 제어를 위한 로킹 기법을 설명할 수 있다.

학습내용

- 직렬 가능 스케줄
- 로킹

직렬 가능 스케줄

동시성 제어의 필요성

0 공유성과 동시 공유

- 공유성(Sharability)
  - 데이터베이스는 여러 사용자들과 공유
- 동시 공유의 장·단점

장점	단점
▶ 공유성 증가 ▶ 응답시간 단축 ▶ 시스템 활용도 증대	▶ 동시에 읽기 · 쓰기가 수행되면 비일관성을 유발 ➡ 동시성 제어(Concurrency Control)가 필요

직렬 가능 스케줄

동시성 제어의 필요성

0 동시성 제어 부재 시 발생하는 현상

01 갱신 분실(Lost Update)

- 다른 트랜잭션이 수정한 내용이 DB(Data Base)에 미반영

02 비일관성(Inconsistency)

- 트랜잭션(Transaction) 수행 결과의 일관성 결여

03 연쇄 복귀(Cascading Rollback)

- 이미 커밋(Commit)된 트랜잭션을 복귀 시킬 필요 발생

직렬 가능 스케줄

동시성 제어의 필요성

0 동시성 제어 부재 시 발생하는 현상

갱신 분실 (Lost Update) | 비일관성 (Inconsistency) | 연쇄복귀 (Cascading Rollback)

결과

- 트랜잭션(Transaction) A와 B가 수행됐으면, 16이 되어야 하는데 6이 저장

Time

TransactionA	Transaction B
read(C)(C= 5)	
	read (C)(C=5)
C:=C+5(C=10)	
	C:=C+1(C=6)
write(C)(C=10)	
	write(C)(C=6)

갱신 분실 (Lost Update) | **비일관성 (Inconsistency)** | 연쇄복귀 (Cascading Rollback)

결과

- 트랜잭션(Transaction) A와 B가 수행됐으면 C=50, D=50이 되어야 하는데, C=50, D=30이 저장

Time

Transaction A	Transaction B
read(C)(C= 5)	
C:=C+5(C=10)	
write(C)(C=10)	
	read(C)(C=10)
	C:=C*5(C=50)
	write(C)(C=50)
	read(D)(D=5)
	D:=D*5(D=25)
	write(D)(D=25)
read(D)(D=25)	
D:=D+5(D=30)	
write(D)(D=30)	

직렬 가능 스케줄

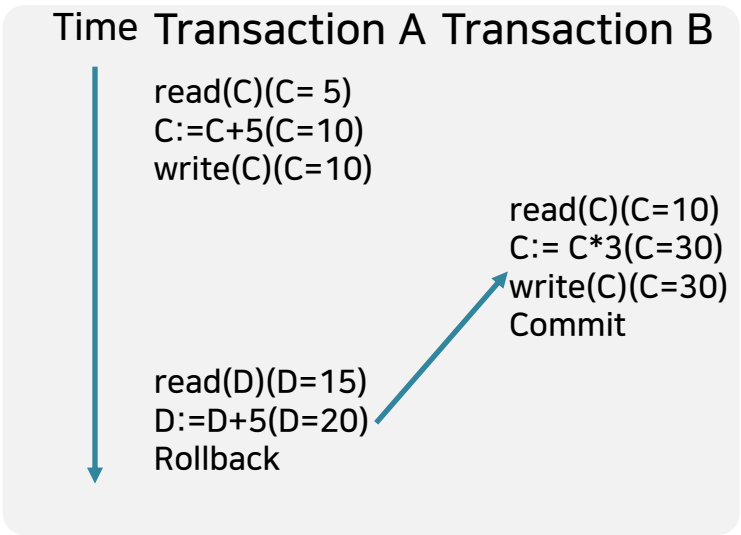
동시성 제어의 필요성

0 동시성 제어 부재 시 발생하는 현상

갱신 분실 (Lost Update) | 비일관성 (Inconsistency) | **연쇄 복귀 (Cascading Rollback)**

결과

- 트랜잭션(Transaction) A가 롤백(Rollback)하면, A가 중간에 만든 결과를 읽은 다른 트랜잭션(Transaction)(B)도 롤백(Rollback) 해야 함
- 트랜잭션(Transaction) B는 이미 커밋(Commit)
- 회복할 방법이 없음
- 커밋(Commit) 하지 않은 트랜잭션(Transaction)이 갱신한 데이터를 다른 트랜잭션(Transaction)이 접근할 때 문제 발생



직렬 가능 스케줄

동시성 제어의 필요성

0 동시성 문제의 원인

충돌(Conflict)

- 서로 다른 트랜잭션(Transaction)에서 동일한 데이터 아이템에 대하여 접근
- 접근하는 연산 중 하나는 쓰기 연산
- 여러 사람이 하나의 아이템을 보기만 할 때는 문제 미발생

동시성 제어

> 충돌 연산들 간의 실행 순서를 정해주는 것

직렬 가능 스케줄

스케줄

0 스케줄의 의미와 동시성

스케줄

[트랜잭션들의 연산 실행 순서]

동시성: 트랜잭션들의 동시 시행

- 하나의 CPU: 시간 분할(Time Slicing)
  - 인터리브드(Interleaved) 실행
  - 각 프로그램은 일정 시간 CPU 상에도 돌고, 이후 다른 프로그램이 CPU 상에서 도는 식으로 다중 프로그래밍을 지원
  - 사용자 입장에서는 각 프로그램이 병렬로 구동되고 있는 것으로 보임

직렬 가능 스케줄

스케줄

0 트랜잭션 스케줄

01 직렬 스케줄(Serial Schedule)

- 트랜잭션 $\{T_1, \ldots, T_n\}$의 순차적 실행
- 인터리브드(Interleaved) 되지 않은 스케줄
- 스케줄의 각 트랜잭션 T_i의 모든 연산 $<T_{i1},\ldots,T_{in}>$가 연속적으로 실행
- n! 가지의 방법
- 올바른 스케줄(Correct Schedule)

  ➡직렬 스케줄은 정확하다고 가정

02 비직렬 스케줄(Non-serial Schedule)

- 인터리브된 스케줄
- 트랜잭션 $\{T_1,\ldots,T_n\}$의 병렬 실행

03 직렬 가능 스케줄(Serializable Schedule)

- n개의 트랜잭션 $T_1,\ldots,T_n$에 대한 스케줄 S가 똑같은 n개의 트랜잭션에 대한 어떤 직렬 스케줄 S'와 동등하면 스케줄 S는 직렬 가능 스케줄

직렬 가능 스케줄

스케줄

0 트랜잭션 스케줄

» 예 | S_2가 직렬 스케줄 <T_1, T_2, T_3>과 동등하다면 S_2는 직렬 가능한 스케줄

- 직렬 스케줄 S_1: <T_1, T_2, T_3>

$$\underset{T_1}{S_1: <O_{11}, O_{12}, O_{13}, O_{14}>} \quad \underset{T_2}{< O_{21}, O_{22}, O_{23}>} \quad \underset{T_3}{< O_{31}, O_{32}>}$$

- 비직렬 스케줄 S_2

$$S_2: <O_{11}, O_{12}, O_{21}, O_{22}, O_{13}, O_{31}, O_{32}, O_{14}, O_{23}>$$

직렬 가능 스케줄

스케줄

0 트랜잭션 스케줄

동등 스케줄의 조건

- S_1과 S_2는 동일한 트랜잭션들을 포함한 경우
- 모든 데이터항목들에 대해 마지막 데이터베이스에 기록한 결과가 두 스케줄에서 동일한 경우

주의 사항

- 연산자의 유형이나 피연산자의 값에 따라 우연히 최종 결과가 같을 수도 있음
- 항상 동일한 결과를 생성하는 것을 보장하지 않음

예 | 초기값: A=1000, B=1000

T_0	T_1
read(A)	read(A)
A:=A-500	A:=A+500
write(A)	write(A)
read(B)	read(B)
B:=B-500	B:=B+500
write(B)	write(B)

직렬 스케줄

- $<T_0, T_1>$: A=1000, B=1000
- $<T_1, T_0>$: A=1000, B=1000

직렬 가능 스케줄

스케줄

0 트랜잭션 스케줄

» 예 | 동등 스케줄

T_0	T_1
read(A)	
A:=A-500	
write(A)	
	read(A)
	A:=A+500
	write(A)
	read(B)
	B:=B+500
	write(B)
read(B)	
B:=B-500	
write(B)	

- 마지막에 데이터베이스에 기록한 결과가 두 스케줄에서 동일
- 직렬 스케줄과 동등 스케줄

» 예 | 충돌 직렬 가능 스케줄

T_0	T_1
read(A)	
A:=A-500	
write(A)	
	read(A)
	A:=A+500
	write(A)
read(B)	
B:=B-500	
write(B)	
	read(B)
	B:=B+500
	write(B)

직렬 가능 스케줄

스케줄

0 트랜잭션 스케줄

» 예 | 비동등 스케줄

T_0	T_1
read(A)	
A:=A-500	
	read(A)
	A:=A+500
	write(A)
	read(B)
write(A)	
read(B)	
B:=B-500	
write(B)	
	B:=B+500
	write(B)

[결과]

- A=500
- B=1500

➡ 갱신 분실 현상

직렬 가능 스케줄

스케줄

0 트랜잭션 스케줄

충돌 동등

- 두 스케줄에서 어떠한 충돌 연산들의 순서가 동일한 경우

충돌 직렬 가능 스케줄

- 어떤 직렬 스케줄과 충돌 동등인 스케줄

» 예 | 어떤 직렬 스케줄과 충돌 동등인 스케줄

T_0	T_1
read(A)	
A:=A-500	
write(A)	
read(B)	
B:=B-500	
write(B)	
	read(A)
	A:=A+500
	write(A)
	read(B)
	B:=B+500
	write(B)

T_0	T_1
read(A)	
A:=A-500	
write(A)	
	read(A)
	A:=A+500
	write(A)
read(B)	
B:=B-500	
write(B)	
	read(B)
	B:=B+500
	write(B)

직렬 가능 스케줄

스케줄

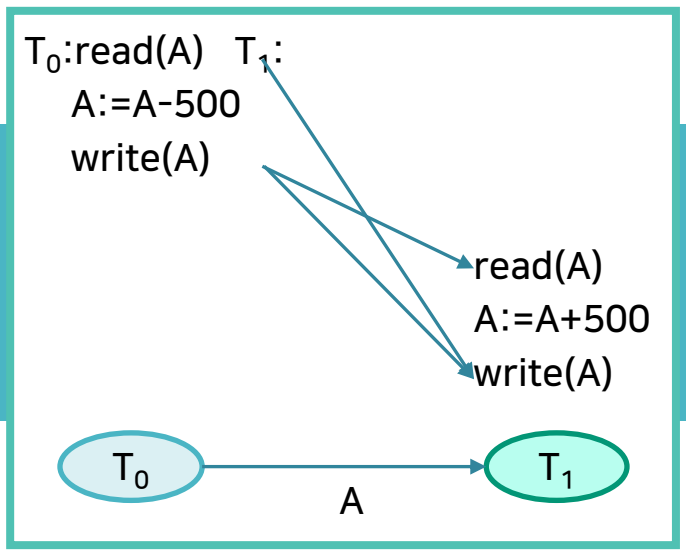
0 충돌 직렬 가능성 검사

선행 그래프(Precedence Graph)

- 방향 그래프: (N, E)
- 노드 T_i: 스케줄 S의 트랜잭션들 집합

간선: $T_i \rightarrow T_j$ 이 나타나는 경우

- T_j가 write(x)한 x의 값을 T_i가 read(x)를 수행하는 경우
- T_j가 read(x)한 뒤, T_i가 write(x)하는 경우
- T_j가 write(x)한 뒤, T_i가 write(x)하는 경우



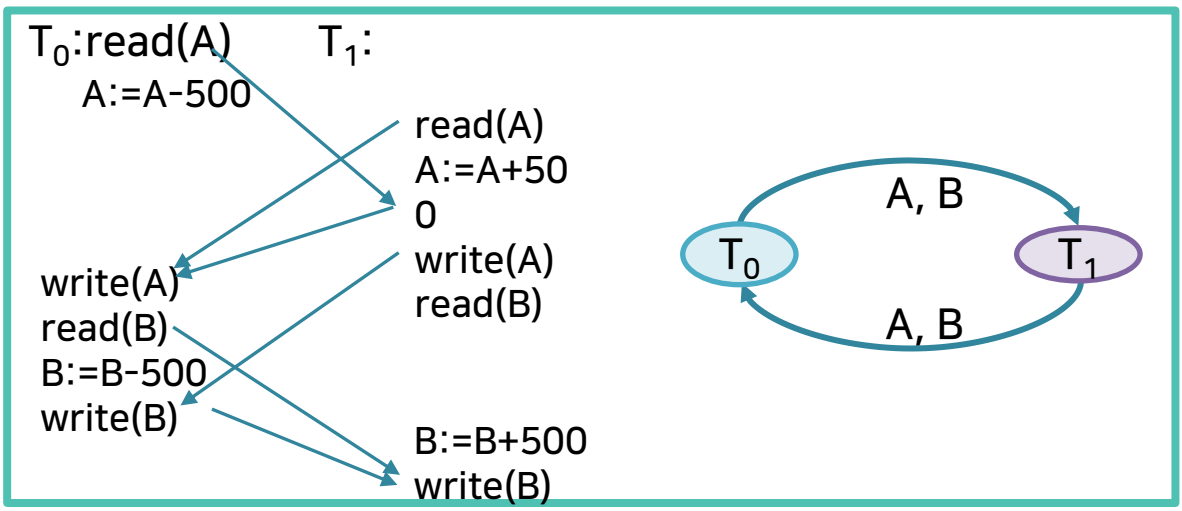
충돌 직렬 가능은
선행 그래프에
사이클이 없는 경우에 가능

직렬 가능 스케줄

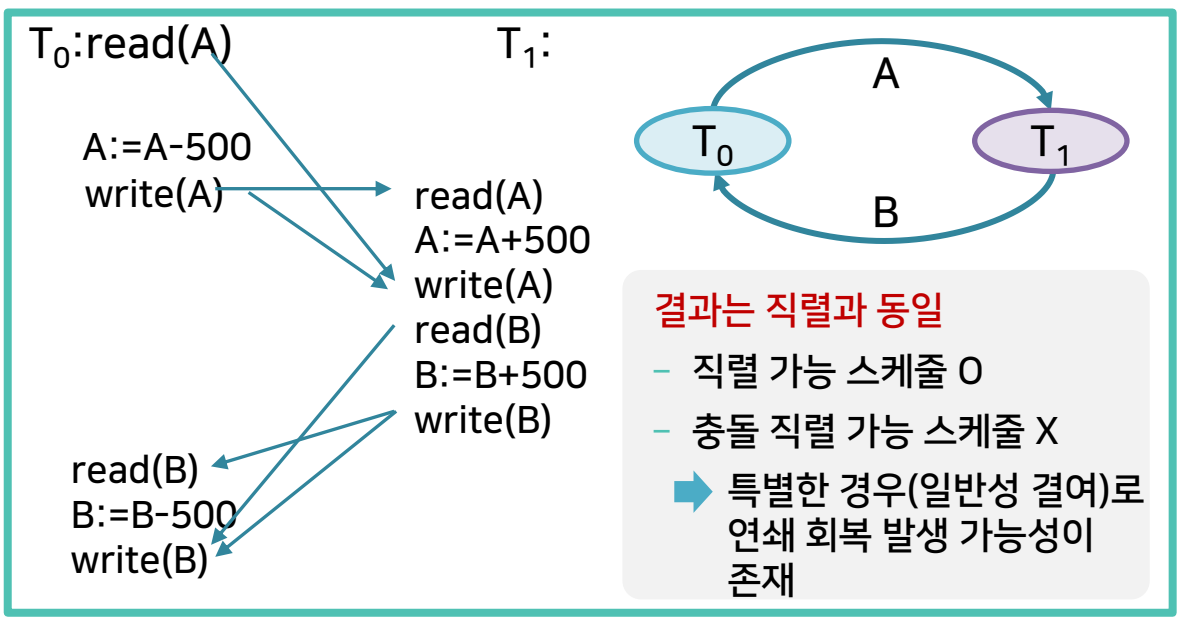
스케줄

0 충돌 직렬 가능성 검사

선행 그래프(Precedence Graph)



결과: 충돌 직렬 가능 스케줄이 아님



결과는 직렬과 동일

- 직렬 가능 스케줄 O
- 충돌 직렬 가능 스케줄 X

➡ 특별한 경우(일반성 결여)로 연쇄 회복 발생 가능성이 존재

직렬 가능 스케줄

스케줄

0 직렬 가능성 검사의 한계

- 트랜잭션을 임의로 수행시킨 뒤 직렬 가능성을 검사
  - 직렬 가능이 안 될 때 스케줄 취소
- 시스템에 트랜잭션들이 계속 들어올 경우
  - 어떤 스케줄이 언제 시작해서 언제 끝나는지 결정이 어려움
  - 문제가 복잡한 경우, 직렬 가능 검사는 불가능

0 직렬 가능성 검사의 활용

- 직렬 가능성 검사를 하지 않고 직렬 가능성을 보장하는 기법

로킹
(Locking)

타임스탬프
(Time Stamp)

직렬 가능 스케줄을 생성한다는 것을 이론적으로 검증할 때 활용

로킹

로킹 규약

로킹 규약의 정의

상호 배제(독점 제어)의 과정

- 잠금이 된 데이터 집합을 생성
- 데이터 객체에 배타적으로 할당
- 무간섭을 보장

로킹 규약의 성질

- 데이터 객체의 비공유·비중첩
- 부분 효과의 배제 (All or Nothing)
- 단일 소유자
- 로크한 트랜잭션만이 로크 해제 가능

로킹

로킹 규약

0 로킹의 단위

- 속성, 레코드, 릴레이션, 데이터베이스

단위가 큰 경우	단위가 작은 경우
동시성 감소	로킹 오버헤드의 증가

0 로킹 규약의 내용

01 트랜잭션 T가 read(x)나 write(x) 연산을 할 때 반드시 먼저 lock(x) 연산을 실행

02 트랜잭션 T가 실행한 lock(x)에 대해 T가 모든 실행을 종료하기 전, 반드시 unlock(x) 연산을 실행

03 트랜잭션 T는 다른 트랜잭션에 의해 이미 lock이 걸려 있는 x에 대해 다시 lock(x) 실행은 불가

04 트랜잭션 T는 x에 lock을 자기가 걸어 놓지 않았을 경우 unlock(x) 실행은 불가

→ 배타적, 공용성 저하

로킹

로킹 규약

0 로킹 모드의 확장

공용 로크 (Shared Lock)
- read 연산에만 허용
- 여러 트랜잭션의 동시 사용 가능

전용 로크 (Exclusive Lock)
- read 및 write 연산에 허용
- 오직 하나의 트랜잭션만 사용 가능

로킹

로킹 규약

0 공용 로킹 규약(Shared Locking Protocol)

트랜잭션 T가 데이터 아이템 x에 대해		
	read(x) 연산 실행 시	먼저 lock-S(x)나 lock-X(x) 연산을 실행
	write(x) 연산 실행 시	먼저 lock-X(x) 연산을 실행

01 x가 이미 다른 트랜잭션에 의해 양립될 수 없는 타입

02 lock이 걸려 있다면 그것이 모두 풀릴 때까지 대기

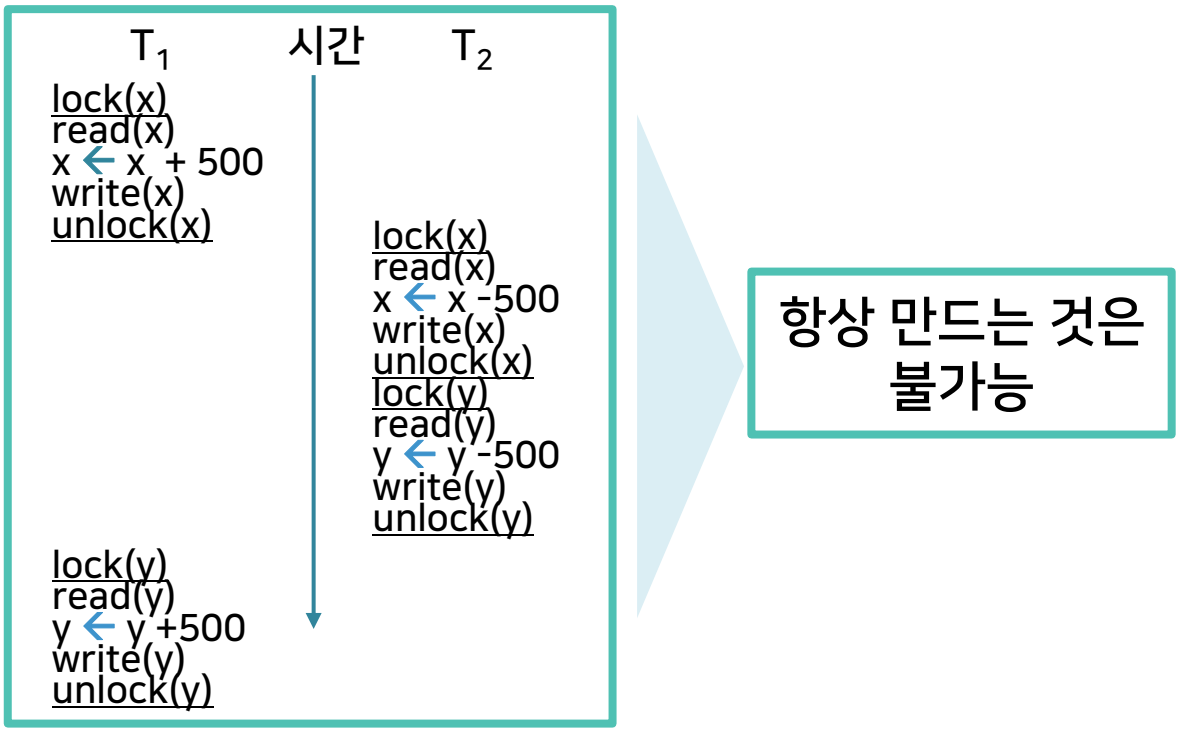
- 트랜잭션 T가 모든 실행을 종료하기 전일 경우
  - T가 실행한 모든 lock(x)에 대해 반드시 unlock(x)를 실행
- 트랜잭션 T가 lock을 걸지 않은 데이터 아이템의 경우
  - unlock(x) 실행 불가

로킹

로킹 규약

0 공용 로킹 규약(Shared Locking Protocol)

Q 로킹을 쓰면 충돌 직렬 가능 스케줄을 항상 만들 수 있을까?



로킹

2단계 로킹 규약(2 Phased Locking Protocol)

0 2단계 로킹 규약의 단계

확장 단계 (Growing Phase)	트랜잭션은 lock만 수행, unlock은 수행 불가
축소 단계 (Shrinking Phase)	트랜잭션은 unlock만 수행, lock은 수행 불가

unlock을 한번이라도 수행한 후에는 lock을 걸 수 없음

로킹

2단계 로킹 규약(2 Phased Locking Protocol)

0 2단계 로킹 규약의 필요성과 주의점

Q 2단계 로킹 규약을 사용하는 이유는?

- 스케줄 내의 모든 트랜잭션들이 2단계 로킹 규약을 준수한다면,
  그 스케줄은 직렬이 가능하기 때문

주의

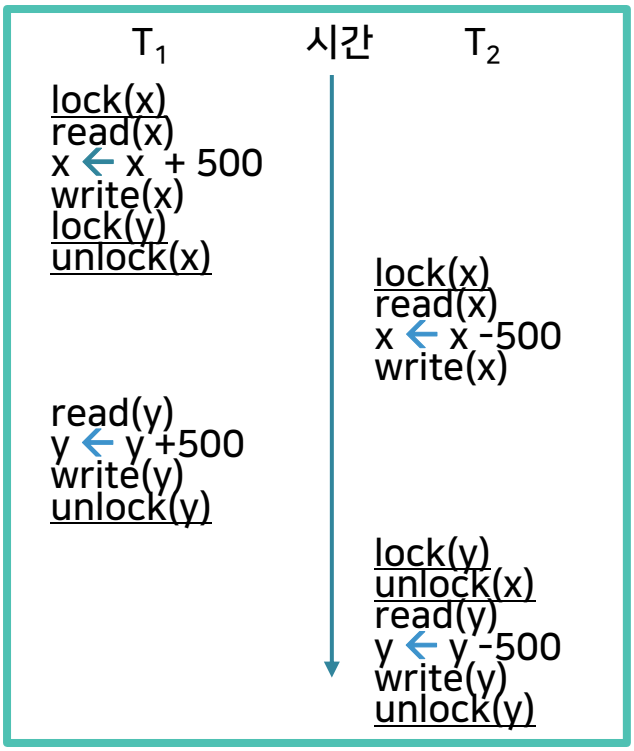
- 2단계 로킹 규약 ➡ 충돌 직렬 가능성을 보장
- 2단계는 직렬 가능성의 충분 조건이며, 필요 조건은 아님
  ➡ 2단계 로킹 규약을 하지 않아도 충돌 직렬 가능 스케줄 생성이 가능함

로킹

2단계 로킹 규약(2 Phased Locking Protocol)

0 2단계 로킹 규약의 예

» 예 | 충돌 직렬 가능성을 보장

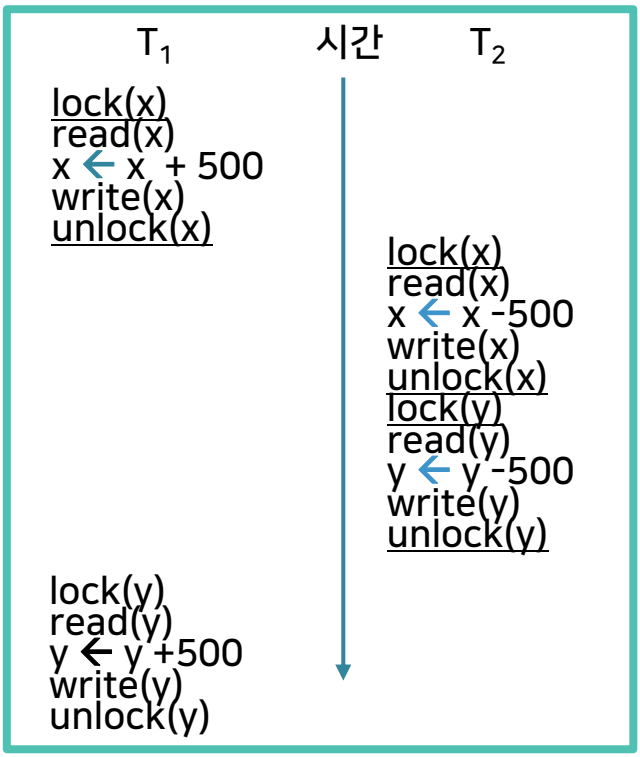


로킹

2단계 로킹 규약(2 Phased Locking Protocol)

0 2단계 로킹 규약이 아닌 예

» 예 | 충돌 직렬 가능 스케줄이 아닌 경우

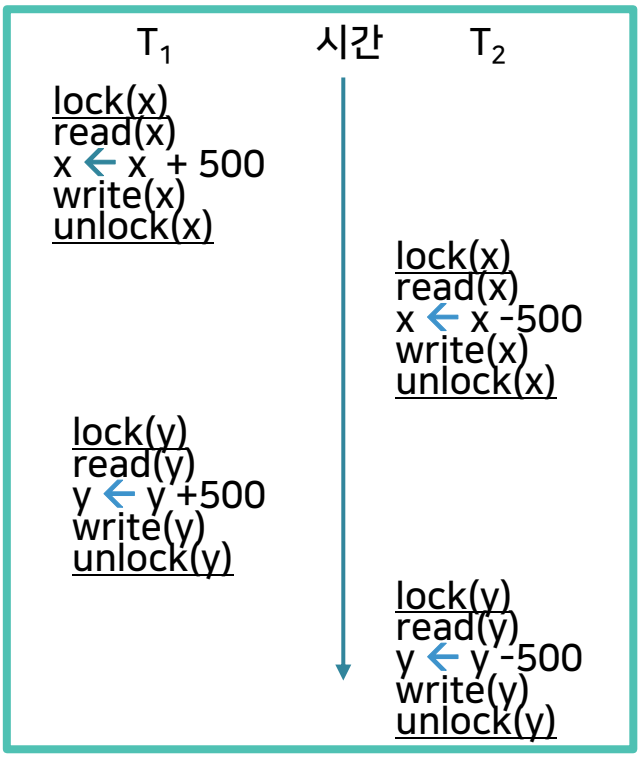


로킹

2단계 로킹 규약(2 Phased Locking Protocol)

0 2단계 로킹 규약이 아닌 예

» 예 | 충돌 직렬 가능 스케줄인 경우



- 2단계 로킹 규약은 충분 조건이고 필요 조건은 아님
- 2단계 로킹 규약으로는 생성할 수 없는 직렬 가능 스케줄

로킹

2단계 로킹 규약(2 Phased Locking Protocol)

0 엄밀 2단계 로킹 규약(Strict 2PLP)

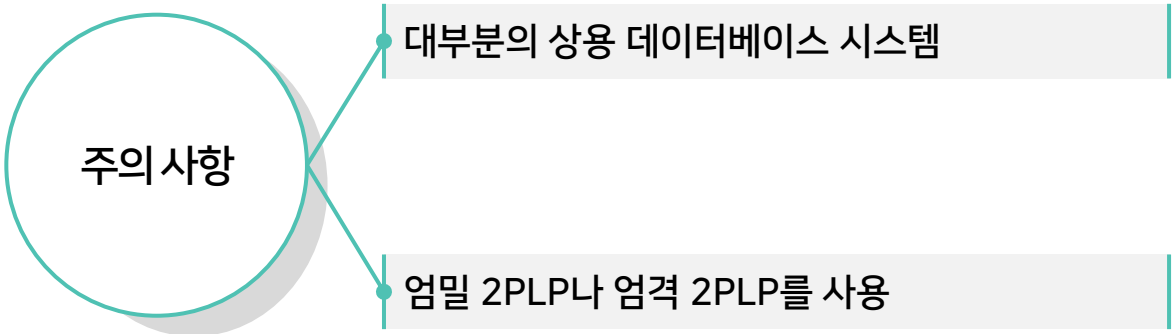
01 모든 독점 로크(lock-X)는 그 트랜잭션이 완료할 때까지 unlock하지 않고 그대로 유지

02 완료하지 않은 어떤 트랜잭션에 의해 기록된 모든 데이터는 그 트랜잭션이 완료될 때까지 독점 모드로 로킹

- 다른 트랜잭션은 그 데이터에 접근 불가
- 연쇄 복귀 문제 미발생

0 엄격 2단계 로킹 규약(Rigorous 2PLP)

- 엄밀 2PLP 보다 더 제한적
- 모든 로크는 그 트랜잭션이 완료할 때까지 unlock 되지 않고, 로크된 상태로 유지
- 트랜잭션들이 완료하는 순서로 직렬화 가능



1 직렬 가능 스케줄

✓ 동시성 문제
- 갱신 분실: 다른 트랜잭션이 수정한 내용이 DB에 미반영
- 비일관성: 트랜잭션 수행 결과의 일관성 결여
- 연쇄 복귀: 이미 커밋(Commit) 된 트랜잭션을 복귀시킬 필요성 발생

✓ 직렬 가능 스케줄
- 스케줄: 트랜잭션 연산의 실행 순서
- 직렬 스케줄: 트랜잭션의 순차적인 실행
- 병렬 스케줄: 트랜잭션의 시간분할 실행(Interleaved)
- 직렬 가능 스케줄: 비직렬 스케줄이며 직렬스케줄과 동등인 스케줄

2 로킹

✓ 상호 배제, 독점적 사용
✓ 잠금이 된 데이터 집합을 생성, 데이터 객체에 배타적 할당 □ 무간섭 보장
✓ 로크한 트랜잭션만이 로크 해제 가능
✓ 2단계 로킹 기법: 충돌 직렬 가능 스케줄 생성