트랜잭션

컴퓨터과학과 정재화

한국방송통신대학교
Korea National Open University

학습목차

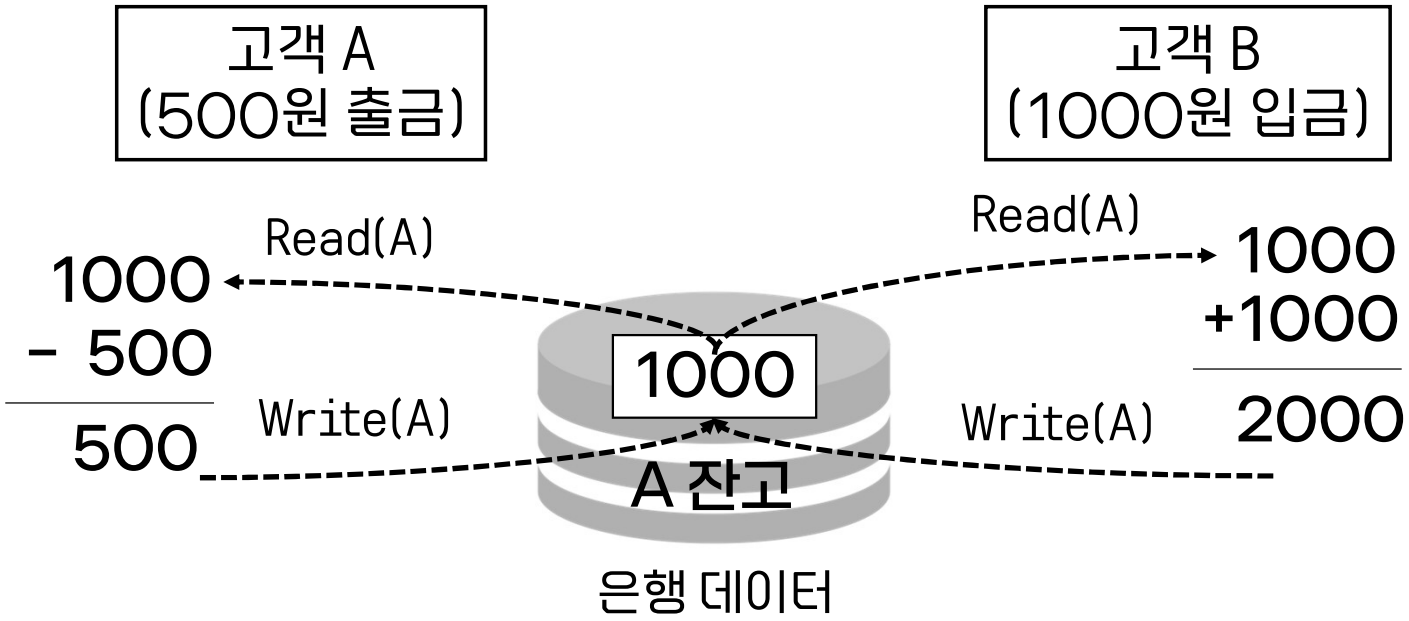
01

트랜잭션의 이해

- 트랜잭션의 개념
- 트랜잭션의 특징
- 트랜잭션 연산자

데이터 동시 접근의 문제

- 동일 데이터에 다수 사용자의 접근 허용 시 일관성이 훼손



트랜잭션의 개념

트랜잭션의 정의

- 데이터베이스를 조작하기 위한 하나의 논리적 단위를 이루는 일련의 데이터베이스 연산의 집합
- 예) 예금 인출
  - 작업 단위: 예금 1000원 인출
  - 일련의 연산: Read(A), A=A-1000, Write(A)

데이터베이스를 사용하여 처리하는 작업을 하나의 묶음으로 인식하여 묶음 단위로 실행된 것과 동일한 결과가 도출되도록 정의한 개념

데이터 읽기와 쓰기

데이터베이스의 두 연산

- Read(X): 데이터베이스에서 데이터 X를 읽고, 트랜잭션이 실행되는 메모리의 변수 X에 값을 저장하는 연산
- Write(X): 트랜잭션이 실행되는 메모리에 있는 변수 X의 값을 데이터베이스에 저장하는 연산

예) 계좌 A에서 B로 1,000원을 이체하는 트랜잭션

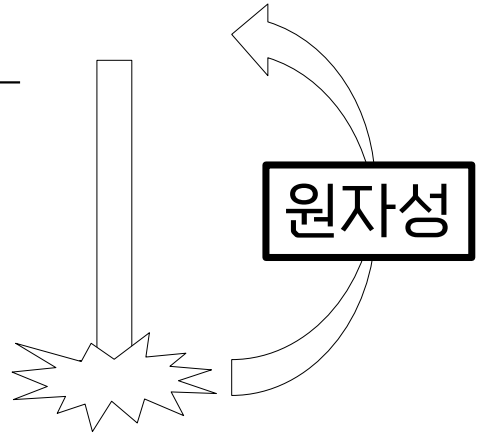
```
Read(A)
A := A – 1000
Write(A)
Read(B)
B := B + 1000
Write(B)
```

트랜잭션의 특징

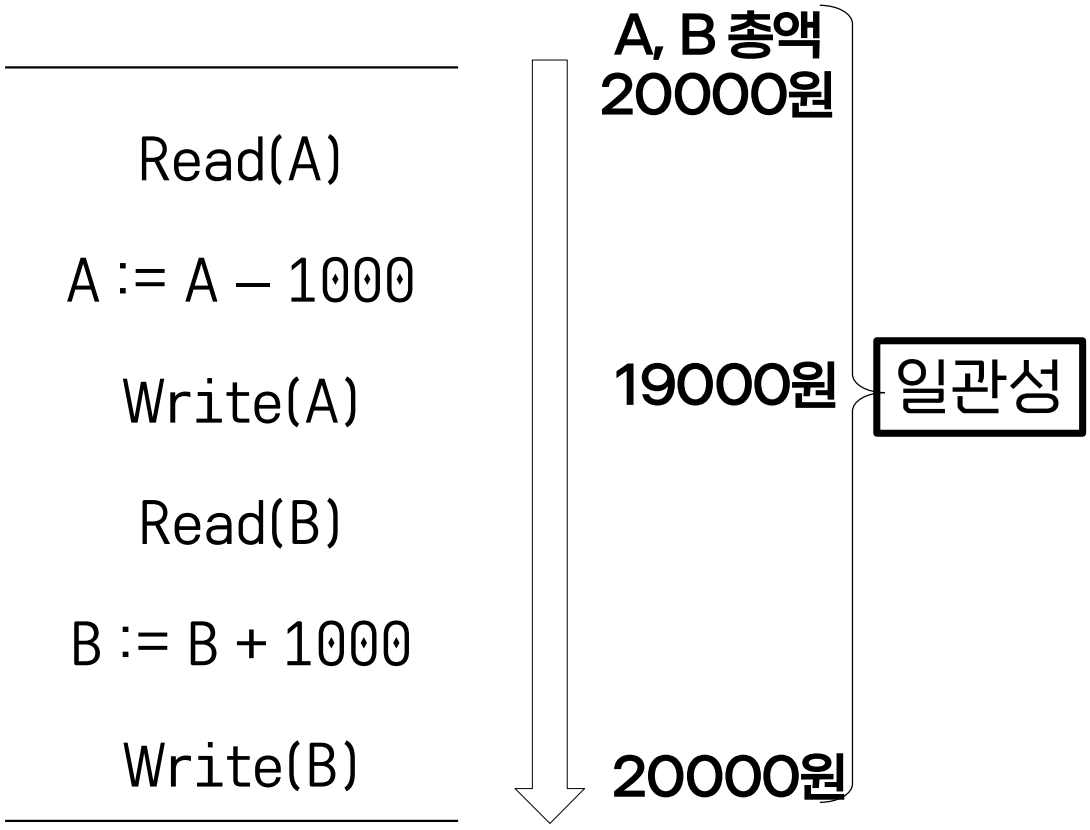
- 다수의 연산으로 구성된 트랜잭션이 사용자에게 단일작업처럼 다뤄지도록 ACID 특징을 준수
- ACID 특성
  - 원자성(Atomicity): 하나의 트랜잭션에 포함된 모든 연산은 완전히 수행되거나 전혀 수행되지 않음
  - 일관성(Consistency): 특정 트랜잭션이 수행되기 전과 후에 데이터베이스가 일관된 상태를 유지
  - 고립성(Isolation): 특정 트랜잭션이 데이터베이스를 갱신하는 동안 다른 트랜잭션에 의해 방해받지 않음
  - 지속성(Durability): 완료된 트랜잭션의 결과는 어떠한 시스템의 장애에도 데이터베이스에 반영되어야 함

ACID 특성 유지

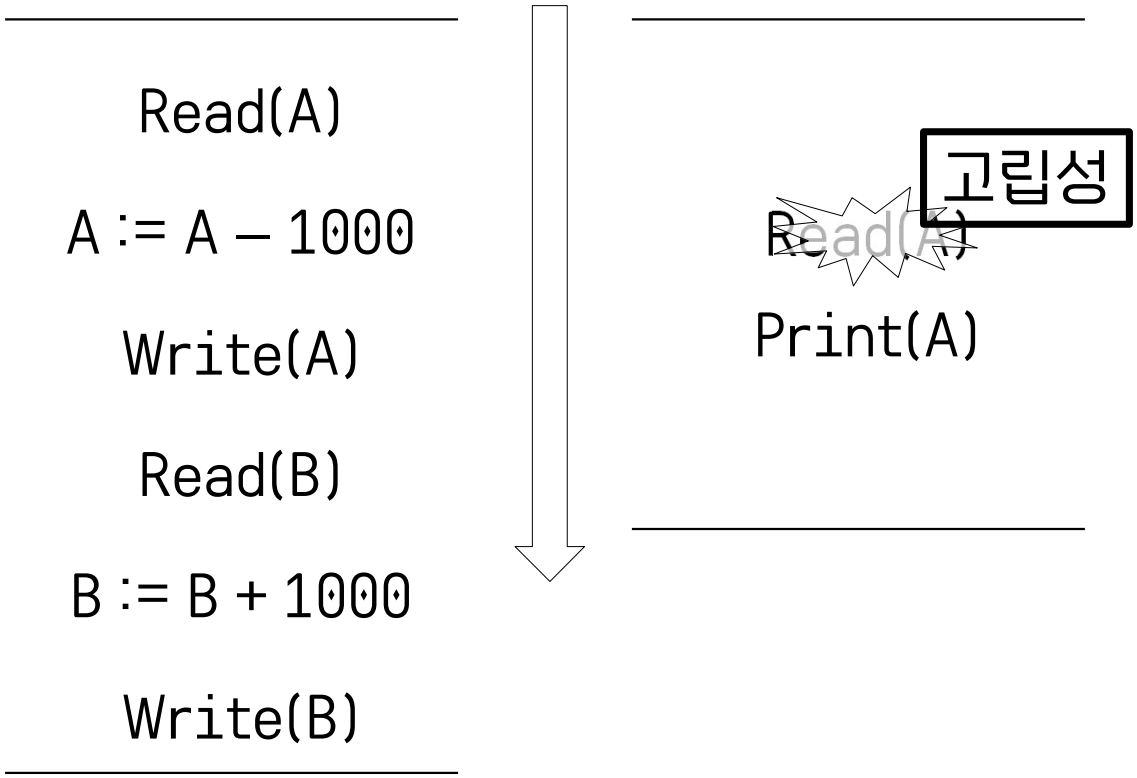
```
Read(A)
A := A – 1000
Write(A)
Read(B)
B := B + 1000
Write(B)
```



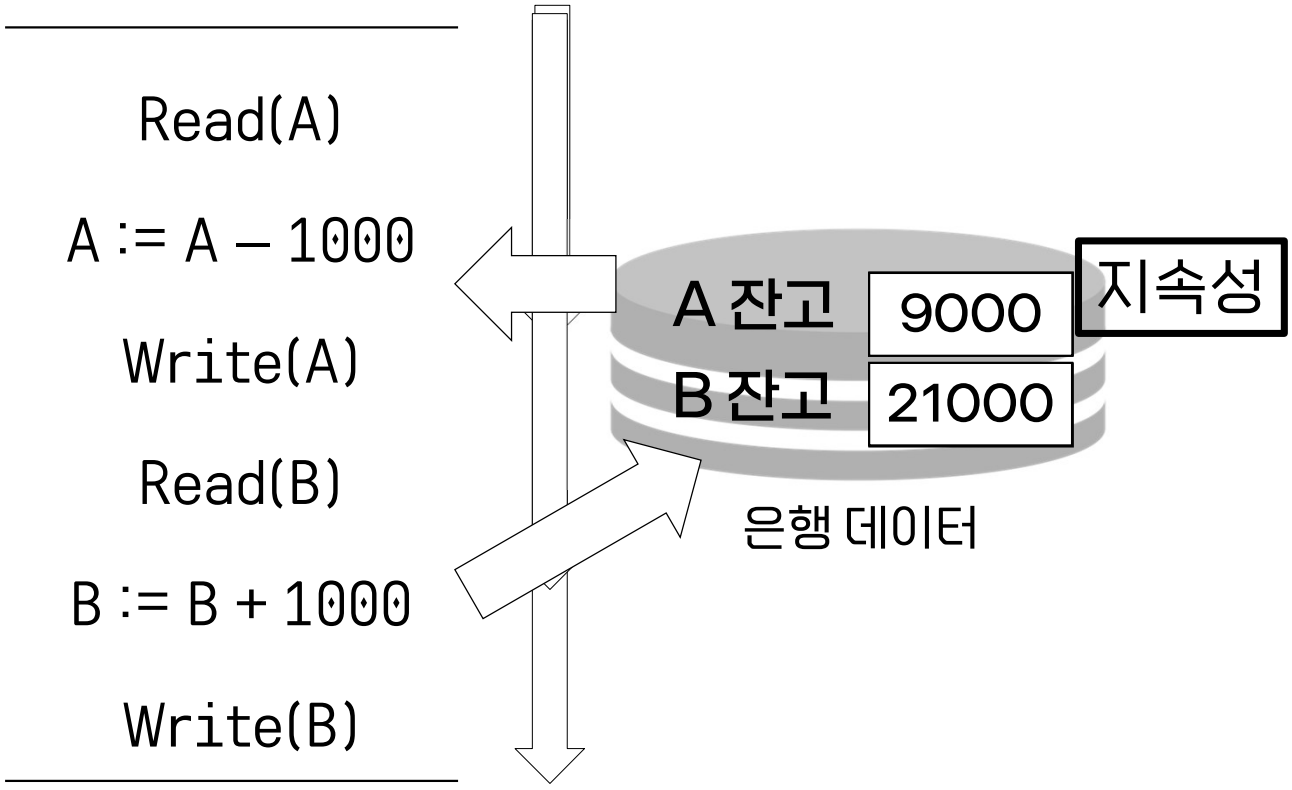
ACID 특성 유지



ACID 특성 유지



ACID 특성 유지



트랜잭션 실행 연산자

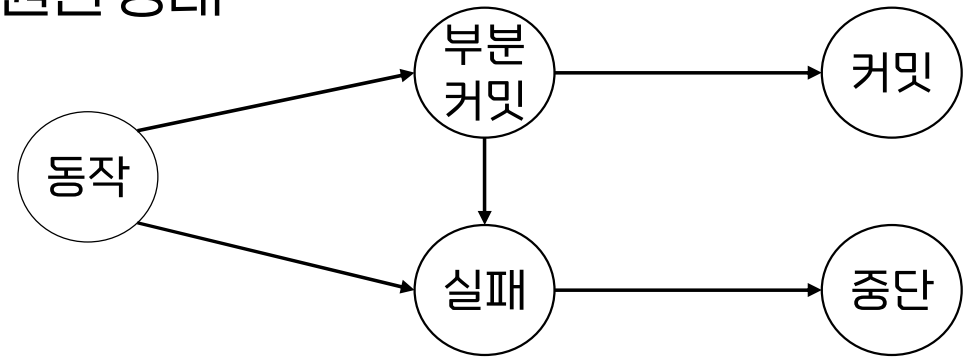
트랜잭션 실행의 연산

- 커밋(commit): 트랜잭션 연산에 의해 갱신된 데이터 항목의 값을 데이터베이스에 반영시키고 지속성을 확보
- 롤백(rollback): 트랜잭션이 중단되기 이전까지 수행한 연산에 의해 갱신된 모든 데이터 항목의 값을 무효화하여 일관성을 확보

트랜잭션의 5가지 상태 변화

트랜잭션의 상태

- 동작: 트랜잭션이 시작을 준비 또는 실행 중인 상태
- 부분 커밋: 마지막 연산을 실행한 직후의 상태
- 커밋: 모든 실행이 성공적으로 완료된 후의 상태
- 실패: 실행이 정상적으로 진행될 수 없는 상태
- 중단: 실행 실패로 롤백되고 시작 이전의 상태로 환원된 상태



02

트랜잭션의 동시성

- 동시성 고려
- 직렬 / 병렬 스케줄
- 충돌 동등 / 충돌 직렬성

동시성 고려

- DBMS는 다수의 사용자가 데이터베이스를 공용으로 사용하기 위한 목적으로 도입
- 다중 사용자 환경에서 트랜잭션의 동시 실행으로 데이터 갱신 시, 일관성 훼손 문제가 발생
- 트랜잭션 동시 실행의 이점
  - 트랜잭션 처리율과 자원 이용률을 향상
  - 트랜잭션의 대기 시간을 감소
- 동시성 제어(concurrency control)
  - 다수의 트랜잭션이 성공적으로 동시에 실행되어도 일관성을 유지할 수 있도록 지원하는 기법

스케줄의 개념

스케줄(schedule)

다수의 트랜잭션에 포함된
연산의 실행 순서를 명시한 것

예제 트랜잭션

- A와 B에 각각 10,000과 20,000원 잔액
- T_1: 1000원을 계좌 A에서 B로 이체
- T_2: 계좌 A의 잔고에서 20%를 B로 이체

	T_1	T_2	
Rea A := A Wri Rea B := B Wri	Read(A) A := A - 1000 Write(A) ⋮	Read(A) temp := A * 0.2 A := A - temp Write(A) ⋮	(A) A * 0.2 temp e(A) (B) temp e(B)

직렬 스케줄

직렬 스케줄

각 트랜잭션에 속한 모든 연산이
순차적으로 실행되는 스케줄

T_1과 T_2가 순차적으로 실행되는 경우

T_1	스케줄1	T_2
Read(A)		
A := A − 1000		
Write(A)		
Read(B)		
B := B + 1000		
Write(B)		
Commit		
		Read(A)
		temp := A * 0.2
		A := A − temp
		Write(A)
		Read(B)
		B := B + temp
		Write(B)
		Commit

T_1	스케줄2	T_2
		Read(A)
		temp := A * 0.2
		A := A − temp
		Write(A)
		Read(B)
		B := B + temp
		Write(B)
		Commit
Read(A)		
A := A − 1000		
Write(A)		
Read(B)		
B := B + 1000		
Write(B)		
Commit		

병렬 스케줄

병렬 스케줄

하나의 트랜잭션이 완료되기 전에
다른 트랜잭션이 실행되는 스케줄

병렬 스케줄의 순서로 연산을 수행할 경우
일관성 훼손 문제가 발생할 가능성 내포

T_0과 T_1의 비순차적 실행되는 스케줄

T_1 스케줄3 T_2

T_1	T_2
Read(A)	
A := A − 1000	
Write(A)	
	Read(A)
	temp := A * 0.2
	A := A − temp
	Write(A)
Read(B)	
B := B + 1000	
Write(B)	
Commit	
	Read(B)
	B := B + temp
	Write(B)
	Commit

T_1 스케줄4 T_2

T_1	T_2
Read(A)	
A := A − 1000	
	Read(A)
	temp := A * 0.2
	A := A − temp
	Write(A)
	Read(B)
Write(A)	
Read(B)	
B := B + 1000	
Write(B)	
Commit	
	B := B + temp
	Write(B)
	Commit

트랜잭션의 직렬화

직렬 가능 스케줄

복수개의 트랜잭션이 동시에 수행된 결과가 직렬 스케줄의 결과와 동일한 스케줄

T_3	스케줄5	T_4
Read(A)		
Write(A)		
		Read(A)
		Write(A)
Read(B)		
Write(B)		
		Read(B)
		Write(B)

직렬 가능 스케줄

- 트랜잭션 간 연산 순서를 교환하여 트랜잭션을 직렬 스케줄과 동등하게 변환이 가능한 스케줄
- 사용된 Read와 Write 연산 교환 시 상황에 따라 실행 결과에 일관성이 훼손되는 현상(충돌)이 발생
- 연산 순서의 교환(단, I_i는 T_i의 연산)

~~$I_i=Read(Q)$~~	~~$I_j=Read(Q)$~~
$I_i=Read(Q)$	$I_j=Write(Q)$
$I_i=Write(Q)$	$I_j=Read(Q)$
$I_i=Write(Q)$	$I_j=Write(Q)$

충돌 동등

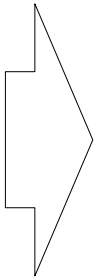
- 특정 스케줄 S에서 충돌이 일어나지 않는 연산의 순서를 바꿔 스케줄 S'으로 변환이 가능한 상태

T_3	스케줄6	T_4
Read(A)		
Write(A)		
		Read(A)
		Write(A)
Read(B)		
Write(B)		
		Read(B)
		Write(B)

충돌 직렬성

- 순서 교환이 가능한 연산을 교환하여
  직렬 스케줄의 연산과 동등하게 변환이 가능한 스케줄

T_3 스케줄5	T_4
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)
	Write(B)



T_3 스케줄6	T_4
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

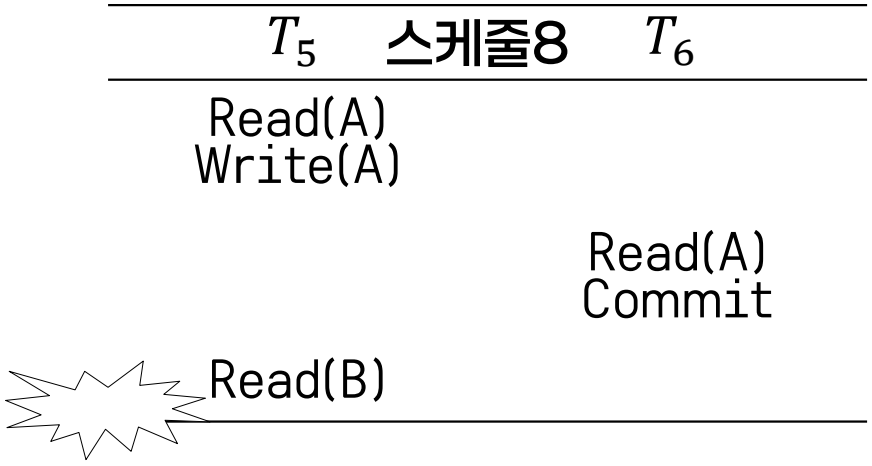
03

트랜잭션의 회복

- 회복의 개념
- 회복 가능한 스케줄
- 비연쇄적 스케줄

회복의 개념

- 트랜잭션 실패 시, 원자성을 보장하기 위해 실행된 모든 연산을 실행 이전 상태로 복원하는 기법
- 회복 불가능한 스케줄
  - T_6가 T_5가 기록한 A를 읽고 커밋한 상태
    - T_6는 T_5에 종속적(dependent)
  - 커밋한 T_5는 롤백 불가능



회복 가능한 스케줄

- T_i와 T_j에 대해, T_i가 기록한 데이터를 T_j가 읽을 때, T_i의 커밋이 T_j 보다 먼저 나타나는 스케줄
- 연쇄적 롤백 유발 가능
  - T_7의 롤백으로 인하여 연쇄적으로 다른 트랜잭션도 롤백 되는 현상

스케줄9

T_7	T_8	T_9
Read(A)		
Write(A)		
	Read(A)	
	Write(A)	
		Read(A)
Abort		

비연쇄적 스케줄

- 대량의 회복 연산이 발생하는 연쇄적 롤백을 방지하기 위해 연쇄적이지 않은 연산 순서로 구성된 스케줄
- T_i가 기록한 데이터를 읽을 때 T_i의 커밋이 T_j의 읽기 연산보다 먼저 나타나는 스케줄

T_7	T_8	T_9
Read(A)		
Write(A)		
Commit		
	Read(A)	
	Write(A)	
	Commit	
		Read(A)

다음 시간

동시성 제어