Chapter 14: 트랜잭션

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use

Chapter 14: 트랜잭션

- 트랜잭션의 개념
- 트랜잭션의 상태
- 동시 실행
- 직렬성
- 직렬성 검사
- 회복성
- SQL에서의 트랜잭션의 정의

©Silberschatz, Korth and Sudarshan

트랜잭션의 개념

- 트랜잭션(**transaction**)이란 다양한 데이터 항목을 검색하고 갱신하는 프로그램 실행 단위이다.

Example:

```
CREATE TABLE accounts
( account_no INTEGER NOT NULL,
balance DECIMAL NOT NULL DEFAULT 0,
PRIMARY KEY(account_no),
CHECK(balance >= 0)
);
```

```
INSERT INTO accounts (account_no,balance)
VALUES (1,200);

INSERT INTO accounts (account_no,balance)
VALUES (2,100);
```

- E.g. 계좌 A에서 B로 50불을 이체하는 트랜잭션:

1. **read**(*A*)
2. *A* := *A* – 50
3. **write**(*A*)
4. **read**(*B*)
5. *B* := *B* + 50
6. **write**(*B*)

```
BEGIN TRANSACTION;

UPDATE accounts
 SET balance = balance - 50
WHERE account_no = 1;

UPDATE accounts
SET balance = balance + 50
WHERE account_no = 2;

COMMIT;
```


©Silberschatz, Korth and Sudarshan

트랜잭션의 개념

- 트랜잭션(**transaction**)이란 다양한 데이터 항목을 검색하고 갱신하는 프로그램 실행 단위이다. 한꺼번에 모두 실행 되어야 할 일련의 연산들을 의미한다. 만약 중간에 실행이 중단되었을 경우, 처음부터 다시 실행하는 **Rollback**을 수행하고, 오류없이 실행을 마치면 **commit**을 하는 실행 단위를 의미합니다. 즉, 한 번 질의가 실행되면 질의가 모두 수행되거나 모두 수행되지 않는 **작업수행의 논리적 단위**입니다.

- E.g. 계좌 A에서 B로 50불을 이체하는 트랜잭션:
  1. **read**(*A*)
  2. *A* := *A* – 50
  3. **write**(*A*)
  4. **read**(*B*)
  5. *B* := *B* + 50
  6. **write**(*B*)

- 다루어야 할 두 가지 주요 사항
  - 하드웨어 고장 및 시스템 이상과 같은 다양한 종류의 고장
  - 여러 트랜잭션의 동시 실행

자금 이체의 예제

- 계좌 A에서 B로 50불을 이체하는 트랜잭션:

1. **read**(*A*)
2. *A* := *A* – 50
3. **write**(*A*)
4. **read**(*B*)
5. *B* := *B* + 50
6. **write**(*B*)

- **트랜젝션의 성질: 데이터 무결성을 보존하기 위해 데이터베이스 시스템은 다음의 특성을 보장해야 한다**:

1. 원자성(**Atomicity**)
   - 트랜잭션의 작업이 부분적으로 실행되거나 중단되지 않는 것을 보장하는 것을 말한다. 즉, 작업 단위를 일부분만 실행하지 않는다는 것을 의미한다.
   - 3번 절차 이후 6번 절차 이전에 트랜잭션이 실패하면, 시스템은 갱신이 데이터베이스에 반영되지 않도록 해야 한다. 그렇지 않으면 불일치가 발생한다.
2. 지속성(**Durability**)
   - 성공적으로 수행된 트랜잭션은 영원히 반영이 되는 것을 말한다. commit을 하면 현재 상태는 영원히 보장됩니다.
   - 트랜잭션이 완료되었다(즉, 50불의 이체가 이루어짐)고 사용자가 통지를 받으면, 트랜잭션이 수행한 데이테베이스 갱신은 고장에도 불구하고 영속적이어야 한다.
3. 일관성(**Consistency**)
   - 트랜잭션은 일관성 있는 데이터베이스를 유지하여야 한다.
   - 트랜잭션 실행 중 데이터베이스는 불일치 상태에 있을 수 있다. 그러나, 트랜잭션이 완료되면, 데이터베이스는 일관성을 유지해야 한다.
   - A와 B의 합계는 트랜잭션의 실행으로 변하지 않는다.

자금 이체의 예제 (계속)

4. 고립성(**Isolation**)
   - 트랜잭션 수행시 다른 트랜잭션의 작업이 끼어들지 못하도록 보장하는 것을 말한다.
   - 즉, 트랜잭션끼리는 서로를 간섭할 수 없다.
   - 절차 3과 6사이에서 부분적으로 갱신된 데이터베이스에 다른 트랜잭션의 액세스를 허용하면, 데이터베이스에 불일치가 발생할 수 있다 (A+B가 원래보다 적게 될 것이다).
   - 트랜잭션을 순서대로 (하나 끝난 후 다음) 실행시키면 당연히 보장될 수 있다.
   - 그러나, 여러 트랜잭션을 동시에 실행시키면 상당한 이점을 가지게 된다.

T1	**T2**
1. **read**(*A*)	
2. *A* := *A* – 50	
3. **write**(*A*)	
	1. read(A), read(B), print(A+B)
4. **read**(*B*)	
5. *B* := *B* + 50	
6. **write**(*B*)	

ACID 속성

데이터 무결성을 보존하기 위해 데이터베이스 시스템은 다음을 보장해야 한다:

- **원자성**(**Atomicity**): 트랜잭션의 모든 연산이 데이터베이스에 적절히 반영되든지 또는 반영이 되지 않든지 해야 한다.

- **일관성**(**Consistency**): 고립 상태의 트랜잭션 실행은 데이터베이스의 일관성을 보존한다.

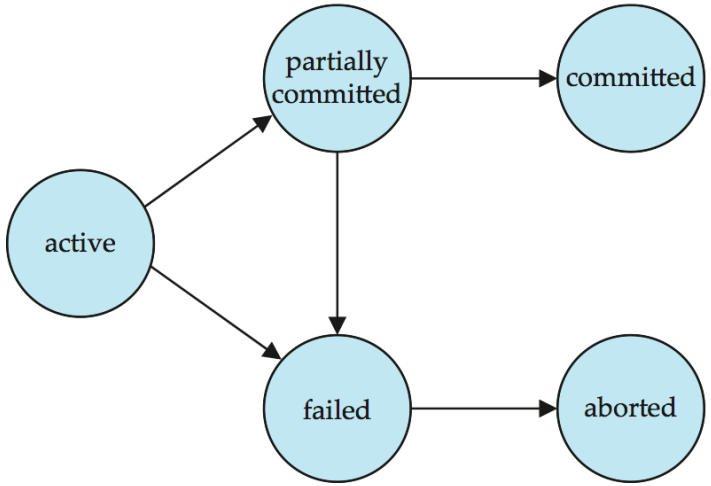
- **고립성** (**Isolation**) **:** 여러 트랜잭션이 동시에 실행되더라도 각 트랜잭션은 다른 트랜잭션들이 동시에 실행되고 있다는 사실을 인지해서는 안된다. 트랜잭션의 중간 결과는 동시에 실행되고 있는 다른 트랜잭션에게는 감추어져야 한다. 즉, 각 트랜잭션의 쌍 T_i와 T_j에 대해 T_i가 시작하기 전에 T_j가 끝나든지 T_i가 끝난 후에 T_j가 시작하는 것처럼 보인다.

- **지속성** (**Durability**) **:** 트랜잭션이 성공적으로 완료한 후에는 시스템 고장이 발생하더라도 데이터베이스에 이루어진 변경은 영속적이다.

©Silberschatz, Korth and Sudarshan

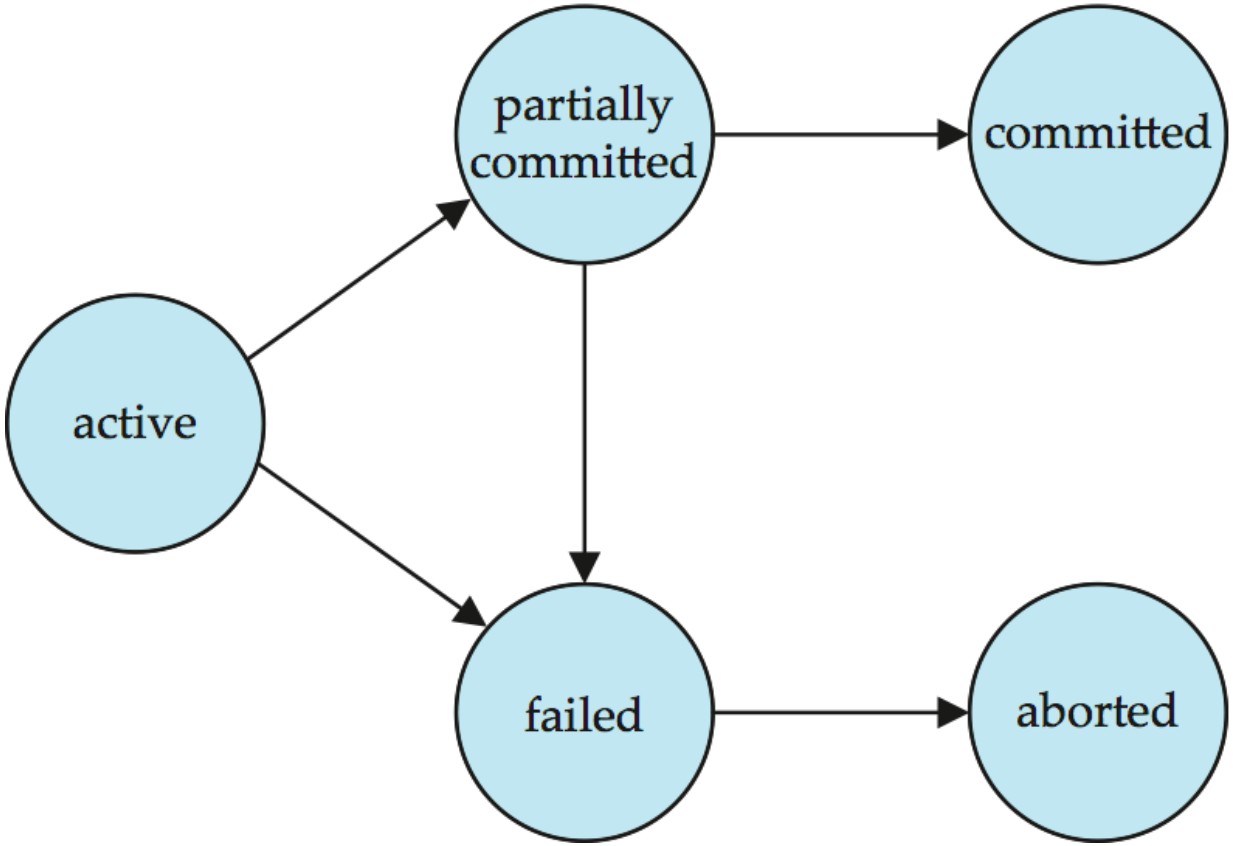
트랜잭션의 상태

- **동작 (Active) -** 초기 상태이다. 트랜잭션이 실행 중인 동안은 이 상태에 머문다.
- **부분완료 (Partially committed) -** 마지막 문장을 실행한 후.
- **실패 (Failed) -** 정상적인 실행을 더 이상 진행할 수 없음을 발견한 후.
- **중단 (Aborted) -** 트랜잭션이 복귀되고 트랜잭션이 시작하기 이전의 데이터베이스 상태로 복구한 후. 중단된 후 두 가지 선택이 가능하다.
  - 트랜잭션의 재시작 – 내부 논리 오류가 없는 경우
  - 트랜잭션을 kill 시킴
- **완료 (Committed) -** 트랜잭션이 성공적으로 종료한 후.



©Silberschatz, Korth and Sudarshan

트랜잭션의 상태 (계속)



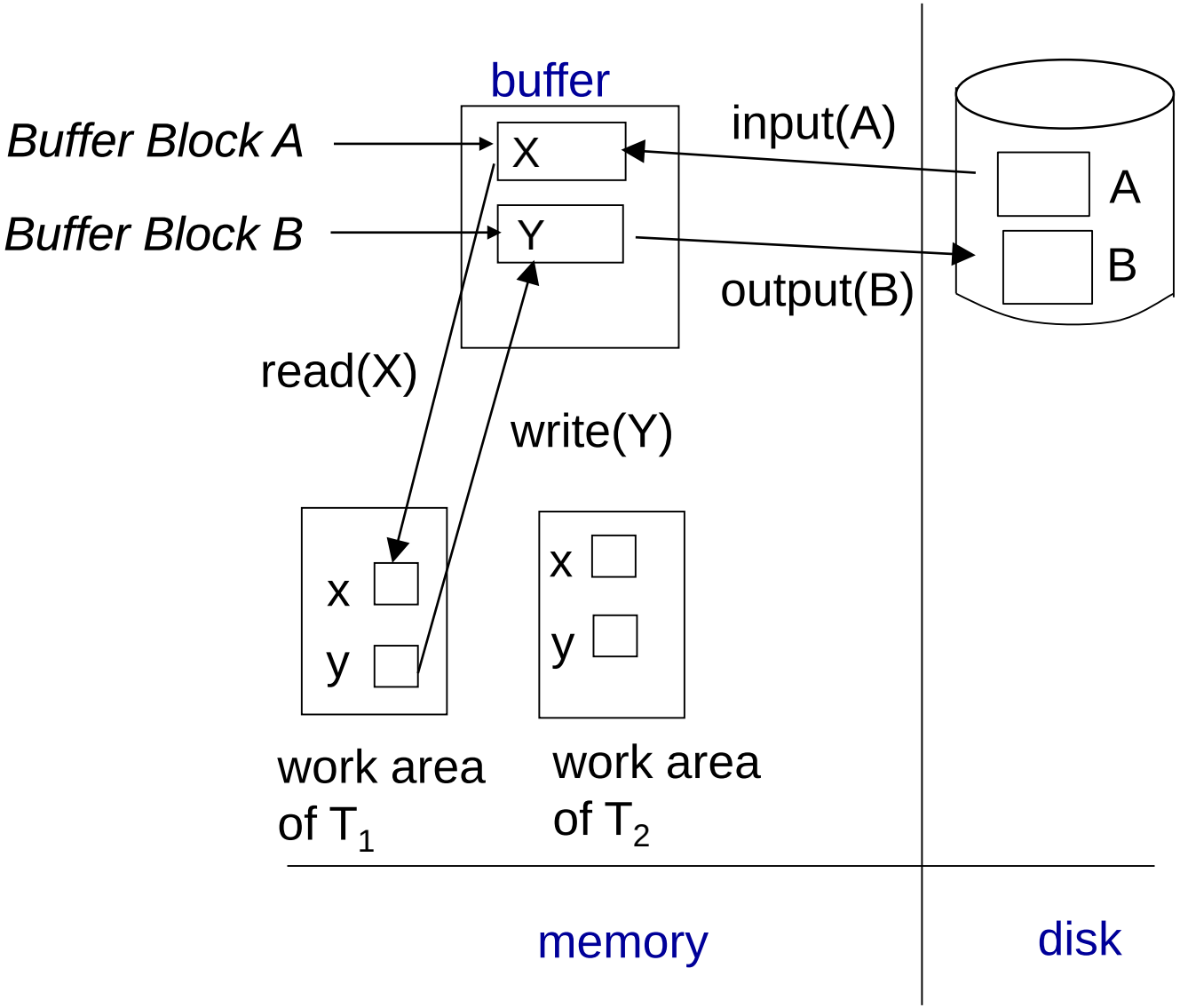
동시 실행

- 여러 트랜잭션이 시스템 내에서 동시에 실행되도록 한다.
- 장점은 다음과 같다:
  - 처리기와 디스크의 이용율을 증가시켜 보다 좋은 트랜잭션 처리율(throughput)을 얻는다: 한 트랜잭션이 디스크 입출력을 수행하는 동안 다른 트랜잭션은 CPU를 사용할 수 있다.
  - 트랜잭션에 대해 평균 응답 시간(**response time**)을 줄인다: 짧은 트랜잭션들이 긴 트랜잭션들 뒤에 기다릴 필요가 없다.
- 동시성 제어 기법(**Concurrency control schemes**) — 데이터베이스의 일관성을 파괴하지 못하도록 동시 실행 트랜잭션들 간의 상호 작용을 제어하는 기법

스케쥴 (Schedules)

- 스케쥴 — 동시 실행 트랜잭션들의 명령들이 어떤 순서로 실행되는가를 나타내는 순서
  - 트랜잭션들의 집합에 대한 스케쥴에는 그들 트랜잭션의 모든 명령을 포함해야 한다.
  - 각 트랜잭션에 나타나는 명령들의 순서를 보존해야 한다.

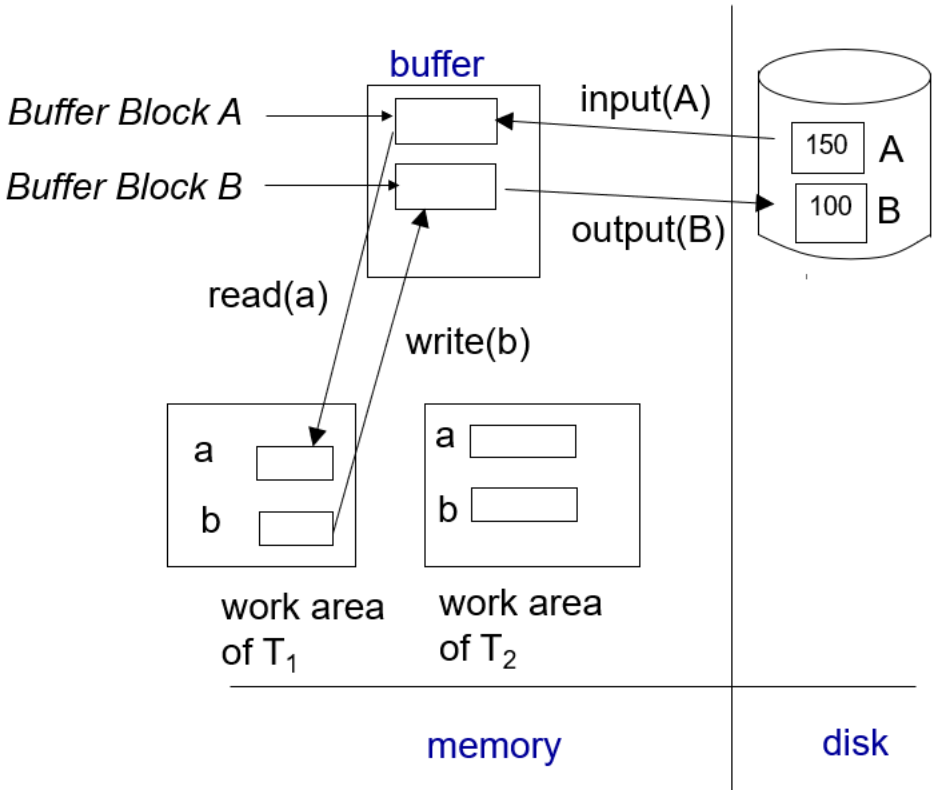
데이터 액세스의 예


buffer
Buffer Block A
Buffer Block B
X
Y
input(A)
output(B)
A
B
read(X)
write(Y)
x
y
x
y
work area
of T_1
work area
of T_2
memory
disk

Schedule 1

- T_1은 계좌 A에서 B로 50불을 이체하고, T_2는 계좌 A의 잔고의 10%를 B로 이체한다 하자.
- 다음은 T_1이 끝난 후 T_2가 시작하는 직렬 스케쥴 (스케쥴1)이다.

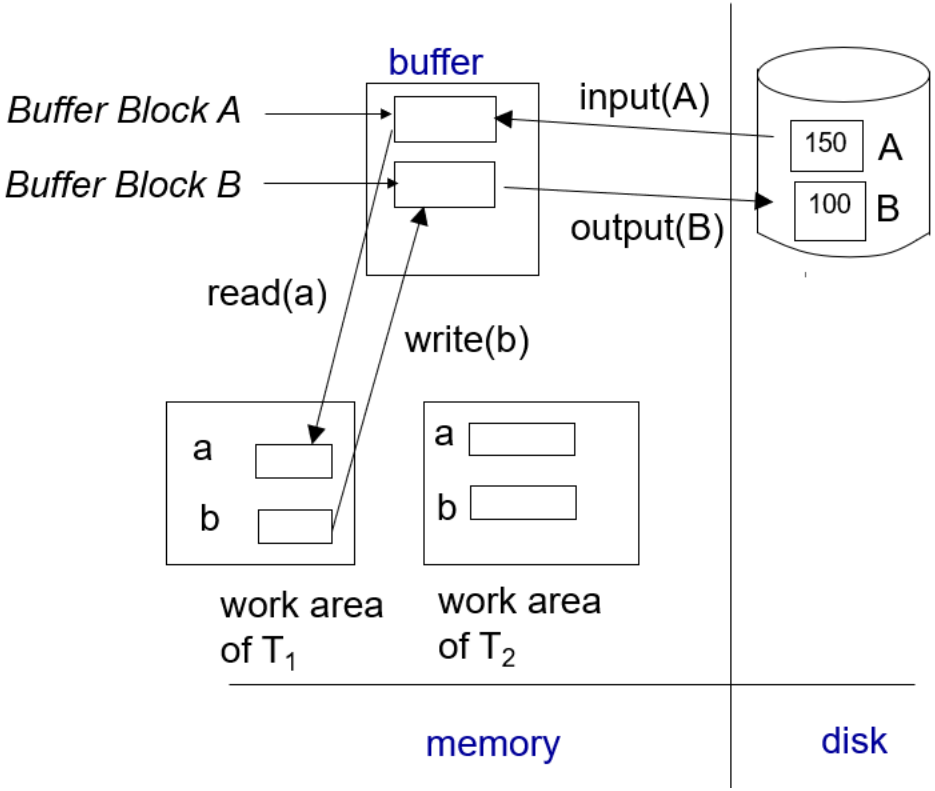
T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit



Schedule 2

- 다음은 T_2가 끝난 후 T_1이 시작하는 직렬 스케쥴 (스케쥴 2)

T_1	T_2
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	

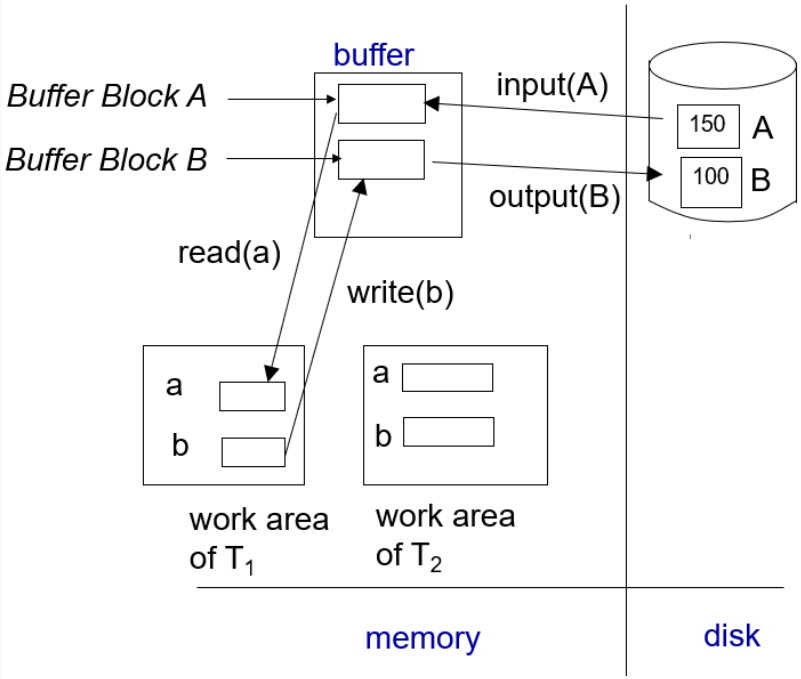


 ©Silberschatz, Korth and Sudarshan

Schedule 3

- T_1과 T_2는 앞의 예와 같다 하자. 다음 스케쥴(스케쥴 3)은 직렬 스케쥴은 아니지만 스케쥴1과 동등하다(*equivalent*).

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (B)
	$B := B + temp$
	write (B)
	commit



스케쥴 1

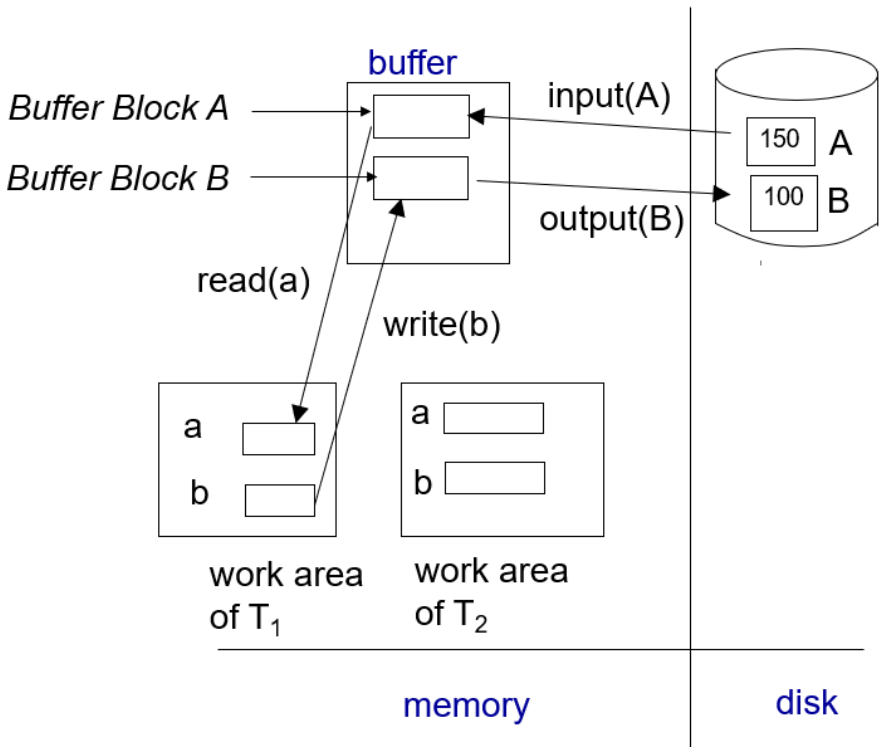
T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

스케쥴1, 2, 3에서 모두 A + B의 합계가 보존된다.

Schedule 4

- 다음 동시 스케쥴(스케쥴 4)은 A + B 합계의 값을 보존하지 않는다.

T_1	T_2
read (A)	
$A := A - 50$	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	$B := B + temp$
	write (B)
	commit



직렬성 (Serializability)

- 기본 가정 — 각 트랜잭션은 데이터베이스의 일관성을 보존한다.
- 따라서, 트랜잭션 집합의 직렬 실행은 데이터베이스의 일관성을 보존한다.
- 직렬 스케쥴과 동등한 스케쥴(동시 실행 가능)을 직렬 가능하다고 한다. 서로 다른 유형의 스케쥴 동등성 (schedule equivalence)이 다음과 같은 개념으로 발생한다:
  1. 충돌 직렬성 (**conflict serializability**)
  2. 뷰 직렬성 (**view serializability**)

©Silberschatz, Korth and Sudarshan

트랜잭션의 간략한 표기법

- read와 write 명령 이외의 연산은 무시한다.
- 트랜잭션은 읽고 쓰는 사이에 지역 버퍼내의 데이터에 임의의 계산을 수행한다고 가정한다.
- 예제의 단순한 스케쥴은 read와 write 명령만으로 구성된다.

©Silberschatz, Korth and Sudarshan

충돌하는 명령어들

- 트랜잭션 T_i와 T_j 각각의 명령 I_i와 I_j가 충돌하는 필요 충분 조건은 I_i와 I_j 가 모두 액세스하는 어떤 항목 Q가 존재하고 이들 명령 중 적어도 하나가 Q를 쓰는 경우이다.
- 1. I_i = read(Q), I_j = read(Q). I_i 와 I_j는 충돌이 아니다.
- 2. I_i = read(Q), I_j = write(Q). I_i 와 I_j는 충돌이다.
- 3. I_i = write(Q), I_j = read(Q). I_i 와 I_j는 충돌이다.
- 4. I_i = write(Q), I_j = write(Q). I_i 와 I_j는 충돌이다.
- 직관적으로, I_i와 I_j 간의 충돌은 그들간의 (논리적) 임시 순서를 강요한다.
  - I_i와 I_j가 스케쥴 내에 연속해 있고 충돌이 아니면, 스케쥴 내에서 그들을 교환한다 하더라도 결과는 같게 된다.

©Silberschatz, Korth and Sudarshan

충돌 직렬성 (Conflict Serializability)

- 스케쥴 S가 비충돌 명령들의 일련의 교환으로 스케쥴 S′로 변환될 수 있으면 S와 S′는 충돌 동등하다(**conflict equivalent**)고 말한다.

- 스케쥴 S가 직렬 스케쥴과 충돌 동등이면 충돌 직렬 가능하다 (**conflict serializable)고** 말한다.

충돌 직렬성(계속)

- 스케쥴 3은 일련의 비충돌 명령들의 맞바꿈으로 *T1* 다음에 *T2*가 수행되는 직렬 스케쥴 6으로 변환될 수 있다. 그러므로, 스케쥴 3은 충돌 직렬 가능하다.

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

Schedule 3

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

Schedule 6

T1	T2
read(A)	
write(A)	
read(B)	
	read(A)
	write(A)
write(B)	
	read(B)
	write(B)

©Silberschatz, Korth and Sudarshan

충돌 직렬성(계속)

- 충돌 직렬 가능하지 않은 스케줄의 예:

T_3	T_4
read (Q)	
	write (Q)
write (Q)	

- 상기 스케줄내 명령들을 맞바꾸어 다음과 같은 <T_3, T_4> 또는 <T_4, T_3>의 직렬 스케줄을 얻을 수 없다.

T3	T4
read(Q)	
write(Q)	
	write(Q)

T3	T4
	write(Q)
read(Q)	
write(Q)	

뷰 직렬성

- S와 S′을 동일한 트랜잭션의 집합을 가진 스케쥴이라 하자. S와 S′은 다음 세가지 조건을 만족하면 뷰 동등(**view equivalent**)하다.
  1. 각 데이터 항목 Q에 대해 트랜잭션 T_i가 스케쥴 S내에서 Q의 초기값을 읽으면, 트랜잭션 T_i는 스케쥴 S′ 내에서 또한 Q의 초기값을 읽어야 한다.
  2. 각 데이터 항목 Q에 대해, 스케쥴 S내어서 T_i가 read(Q)를 실행하고 그 값은 트랜잭션 T_j가 생성한 것(있다면) 이라면, 스케쥴 S′내에서 트랜잭션 T_i 또한 트랜잭션 T_j가 생성한 Q의 값을 읽어야 한다.
  3. 각 데이터 항목 Q에 대해, 스케쥴 S내에서 마지막 write(Q) 연산을 수행한 트랜잭션(있다면)은 스케쥴 S′ 내에서도 마지막 write(Q) 연산을 수행해야 한다.
- 뷰 동등성(view equivalence)도 순수히 read와 write 연산에만 근거하여 정의된다.

©Silberschatz, Korth and Sudarshan

뷰 직렬성 (계속)

- 스케쥴 S가 직렬 스케쥴과 뷰 동등하면 뷰 직렬 가능(**view serializable**)하다.
- 모든 충돌 직렬 가능 스케쥴은 또한 뷰 직렬 가능하다.
- 뷰 직렬 가능하지만 충돌 직렬 가능하지 않은 스케쥴의 예:

T_{27}	T_{28}	T_{29}
read (Q)		
	write (Q)	
write (Q)		
		write (Q)

S: 뷰직렬가능 스케쥴

T_{27}	T_{28}	T_{29}
read (Q)		
write (Q)		
	write (Q)	
		write (Q)

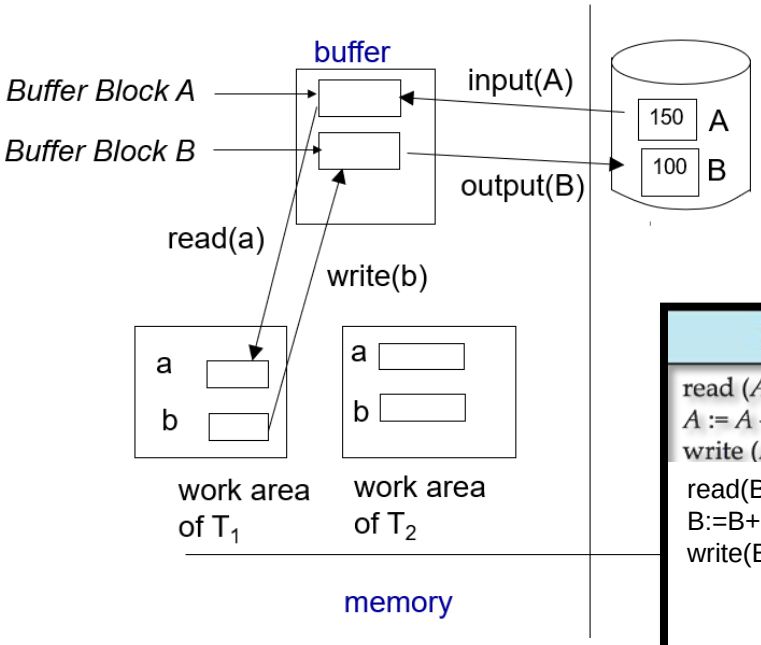
직렬 스케쥴 S'

- 동등한 직렬 스케쥴은 ?
- 충돌 직렬 가능하지 않은 모든 뷰 직렬 가능 스케쥴은 맹목적 출력(**blind writes**)을 갖는다.

직렬성의 기타 개념

- 아래의 스케쥴 은 직렬 스케쥴 <T_1, T_5> 와 같은 결과를 생성하지만, 충돌 동등이거나 뷰 동등의 조건을 만족시키지 않는다.

T_1	T_5
read (A)	
$A := A - 50$	
write (A)	
	read (B)
	$B := B - 10$
	write (B)
read (B)	
$B := B + 50$	
write (B)	
	read (A)
	$A := A + 10$
	write (A)



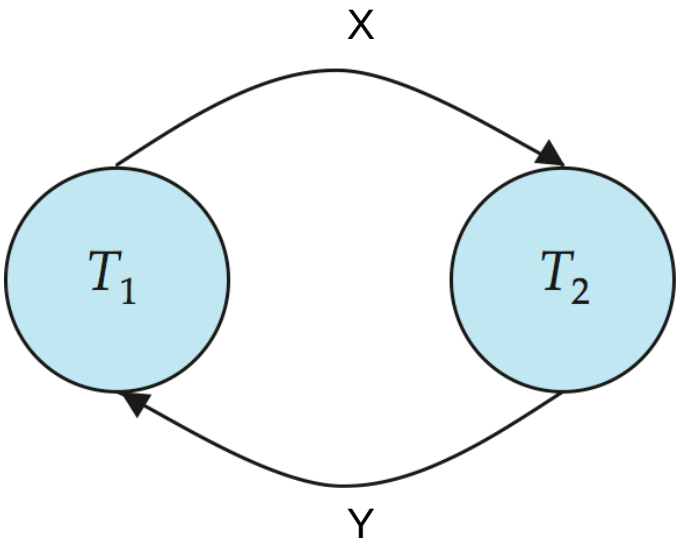
T_1	T_5
read (A) $A := A - 50$ write (A)	
read(B) B:=B+50 write(B)	
	read(B) B:=B-10 write(B)
	read (A) $A := A + 10$ write (A)

- 이러한 동등성을 결정할 때는 read와 write 이외의 연산을 분석할 필요가 있다.

©Silberschatz, Korth and Sudarshan

직렬성 검사 방법

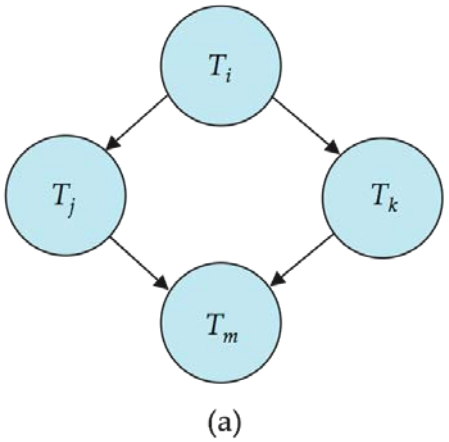
- 트랜잭션의 집합 $T_1, T_2, ..., T_n$의 어떤 스케쥴을 고려해 보자.
- 선행 그래프(**Precedence graph**) — 각 정점이 트랜잭션(명)인 방향성 그래프
- 두 트랜잭션이 충돌이고 충돌이 앞에서 발생한 데이터 항목에 T_i가 액세스하면 T_i에서 T_j로 호를 그린다.
- 호 위에 액세스한 항목 이름을 붙일 수 있다.
- **Example 1**



©Silberschatz, Korth and Sudarshan

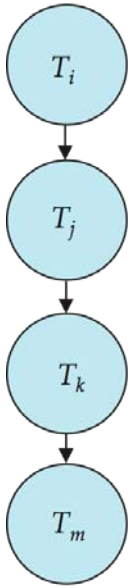
충돌 직렬성 검사

- 스케쥴이 충돌 직렬 가능인 필요 충분 조건은 그의 선행 그래프가 비순환적일 때이다.
- n^2 차수의 시간이 걸리는 순환 탐지 알고리즘(n은 그래프 내 정점의 수)이 존재한다
  - 보다 나은 알고리즘은 n + e 시간이 걸린다. e는 간선의 수.

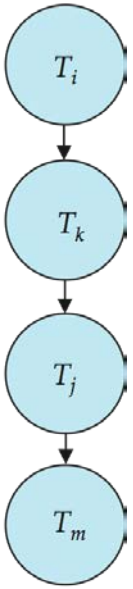


(a)

- 선행 그래프가 비순환이면, 그래프의 위상 정렬(*topological sorting*)로 직렬성 순서를 얻을 수 있다.
  - 이것은 그래프의 부분 순서와 일치하는 선형 순서이다.
  - 예를 들어, 스케쥴 (a)에 대한 직렬성 순서는 $T_i \rightarrow T_j \rightarrow T_k \rightarrow T_m$ 이 된다.



(b)



(c)

©Silberschatz, Korth and Sudarshan

뷰 직렬성 검사 (생략)

- 충돌 직렬성 검사를 위한 선행 그래프는 뷰 직렬성 검사를 위해 수정해야 한다.
- 라벨 선행 그래프를 구축한다. 0이 아닌 같은 라벨을 가진 모든 간선의 쌍으로부터 하나의 간선을 선택해 라벨 선행 그래프에서 추출한 비순환 그래프를 찾는다. 스케쥴이 뷰 직렬 가능한 필요 충분 조건은 그러한 비순환 그래프가 있는 경우이다.
- 그러한 비순환 그래프를 찾는 문제는 NP-complete 문제에 속한다. 따라서, 효율적인 알고리즘은 존재하지 않는다.
- 그러나, 뷰 직렬성을 위한 어떤 충분 조건만을 검사하는 실질적인 알고리즘은 여전히 사용될 수 있다.

©Silberschatz, Korth and Sudarshan

회복 가능 스케쥴 (Recoverable Schedules)

동시에 실행되는 트랜잭션들에서 트랜잭션 실패의 영향을 언급할 필요가 있다.

- 회복가능스케쥴: 트랜잭션 T_i가 먼저 쓴 데이터 항목을 트랜잭션 T_j가 읽는다면, T_i의 완료 연산이 T_j의 완료 연산 전에 나타난다.
- 다음 스케쥴은 T_9가 읽은 직후 완료하면 회복 가능하지 않다.
- T_8이 중단(abort) 되면, T_9는 불일치 상태의 데이터베이스를 읽게 된 것이다.
- 그러므로, 데이터베이스는 스케쥴이 회복 가능하도록 보장해야 한다.

T_8	T_9
read (A)	
write (A)	
	read (A)
	commit
read (B)	

©Silberschatz, Korth and Sudarshan

연쇄 복귀 (Cascading Rollbacks)

- 연쇄 복귀: 하나의 트랜잭션의 실패(failure)가 연속적으로 여러 트랜잭션의 복귀(rollback)을 유발시키는 현상
- 한 트랜잭션의 실패가 일련의 트랜잭션 복귀를 야기한다. 어떤 트랜잭션도 현재 완료되지 않은 다음 스케쥴을 고려해 보자
- (따라서 이 스케쥴은 회복 가능하다).
- T_{10}이 실패(failure) 하면, T_{11}과 T_{12} 또한 복귀(rollback) 되어야 한다.
- 막대한 작업량의 취소를 야기할 수 있다.

T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

연쇄 복귀 없는 스케쥴 (Cascadeless Schedules)

- 연쇄 복귀 없는 스케쥴 — 연쇄 복귀는 발생할 수 없다;
- T_i가 먼저 쓴 데이터 항목을 T_j가 읽는 그러한 각 트랜잭션의 쌍 T_i와 T_j에 대해, T_i의 완료 연산은 T_j의 읽기 연산 전에 나타난다.
- 모든 연쇄 복귀없는 스케쥴은 또한 회복 가능하다.
- 연쇄 복귀없는 스케쥴로 제한하는 것이 바람직하다.

©Silberschatz, Korth and Sudarshan

동시성 제어 (Concurrency Control)

- 데이터베이스 시스템은 모든 스케쥴이 다음 조건을 만족시키도록 관리하여야 한다.
  - 충돌 혹은 뷰 직렬 가능
  - 회복 가능하며, 가능하다면 연쇄 복귀 없는 스케쥴

- 스케쥴이 실행된 후 직렬성을 검사하는 것은 너무 늦다.
- 목표 — 직렬성을 보장할 동시성 제어 규약을 개발하는 것.
  - 규약은 일반적으로 선행 그래프가 생성될 때 검사하지 않고, 비직렬 가능 스케쥴을 회피하는 원칙을 부과한다. 그러한 규약에 대해서는 14장에서 다룬다.
- 직렬성 검사는 동시성 제어 규약이 왜 정확한지의 이해를 돕는다.

SQL에서의 트랜잭션 정의

- 데이터 조작어에는 트랜잭션을 구성하는 행위 집합을 지정하기 위한 구조체를 내포해야 한다.
- SQL에서 트랜잭션은 묵시적으로 시작한다.
- SQL에서 트랜잭션은 다음 중 하나로 끝난다:
  - **Commit work** 현재 트랜잭션을 완료하고 새로운 트랜잭션을 시작한다.
  - **Rollback work** 현재 트랜잭션이 중단되도록 한다.

©Silberschatz, Korth and Sudarshan

End of Chapter 14

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use