CHAPTER 10

회복과 병행 제어

Recovery & Concurrency Control

데이터베이스 시스템 | Database Systems

박상돈 교수
컴퓨터공학과

CONTENTS

이번 장에서 다루는 세 가지 핵심 주제

⇄

01

트랜잭션

Transaction

트랜잭션의 개념

Transaction

- 하나의 작업을 수행하는 데 필요한 데이터베이스 연산을 모아놓은 것
- DB에서 처리되는 논리적인 작업의 단위
- 장애 발생 시 데이터 복구 작업의 단위
- 작업 수행에 필요한 SQL 문들의 모임

트랜잭션 관리 (Transaction Management)

데이터베이스의 일관성(consistency)을 유지하는 것

왜 묶어서 관리할까?

연산을 따로따로 처리하면 중간에 장애가 났을 때 데이터가 어긋난 상태로 남습니다. 하나의 단위로 묶어야 '전부 성공' 또는 '전부 취소'를 보장할 수 있습니다.

트랜잭션의 예시 ① 계좌이체

성호 계좌에서 은경 계좌로 5,000원을 이체하는 과정 — 출금 ①과 입금 ②는 반드시 함께 일어나야 한다.

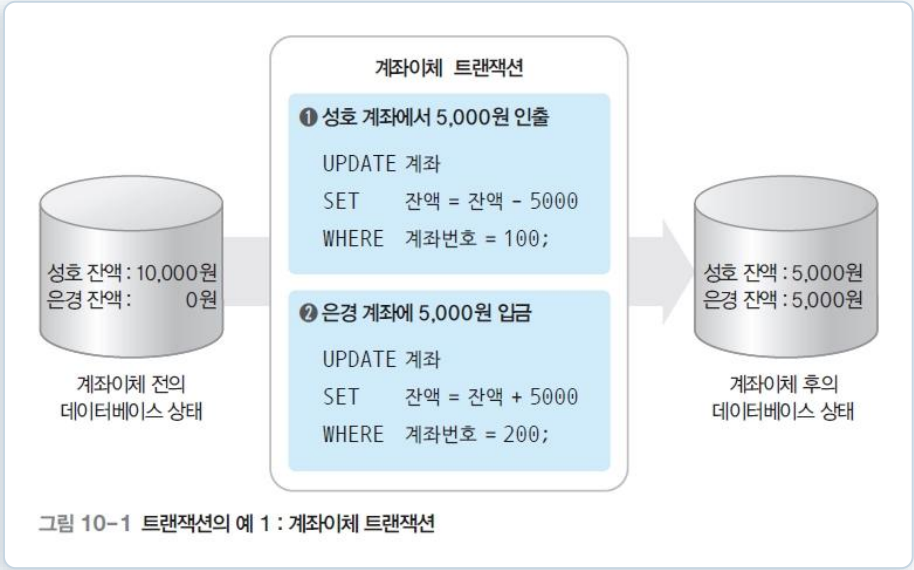


그림 10-1 트랜잭션의 예 1 : 계좌이체 트랜잭션

트랜잭션의 예시 ② 상품주문

쇼핑몰에서 제품 10개를 주문하면 주문 등록 ①과 재고량 감소 ②가 함께 발생해야 한다.

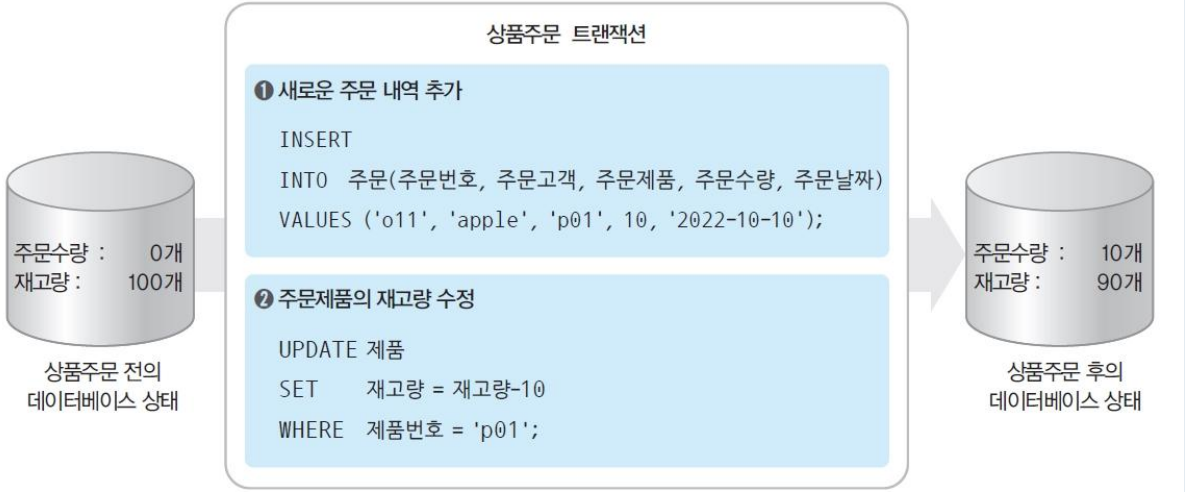


그림 10-2 트랜잭션의 예 2 : 상품주문 트랜잭션

트랜잭션 관리 SQL문

트랜잭션은 데이터를 변경하는 다음 SQL 문들로 구성된다.

INSERT

새로운 데이터(행)를 테이블에 삽입

```
INSERT INTO 주문 VALUES(...)
```

UPDATE

기존 데이터의 값을 수정

```
UPDATE 계좌 SET 잔액=잔액-5000
```

DELETE

기존 데이터(행)를 테이블에서 삭제

```
DELETE FROM 주문 WHERE ...
```

트랜잭션의 특성 — ACID

트랜잭션이 안전하게 처리되기 위해 반드시 보장되어야 할 네 가지 특성

A

원자성

Atomicity

모두 실행되거나
전혀 실행되지 않거나

C

일관성

Consistency

수행 후에도 DB가
일관된 상태 유지

I

격리성

Isolation

수행 중 중간 결과에
다른 트랜잭션 접근 불가

D

지속성

Durability

완료된 결과는
영구적으로 보존

원자성 Atomicity — All or Nothing

트랜잭션의 연산들이 모두 정상 실행되거나, 하나도 실행되지 않아야 한다. 성공 시 두 연산이 모두 반영된다.

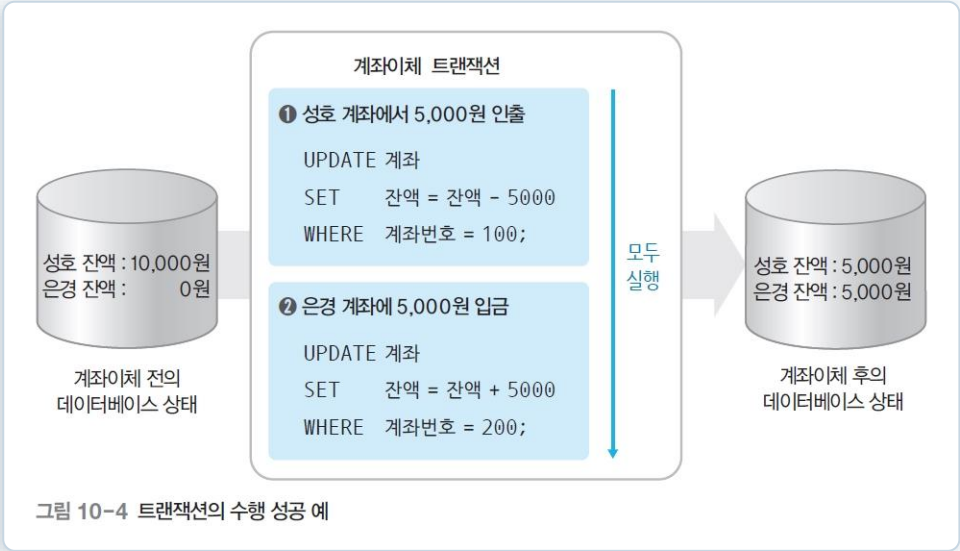


그림 10-4 트랜잭션의 수행 성공 예

원자성 — 장애 발생 시 작업 취소(rollback)

수행 중 장애로 트랜잭션을 완료하지 못하면, 지금까지 실행한 연산을 모두 취소하고 작업 이전 상태로 되돌린다.

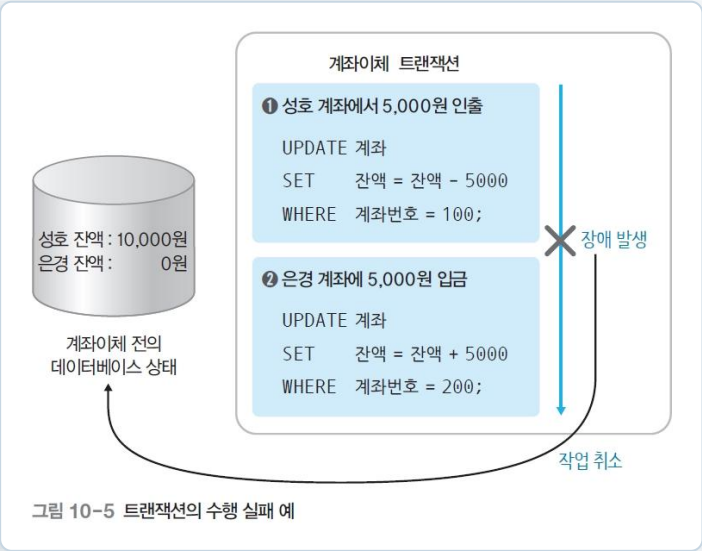


그림 10-5 트랜잭션의 수행 실패 예

일관성 Consistency

트랜잭션 수행 후에도 DB가 일관된 상태를 유지해야 한다. (두 계좌 잔액 합계는 항상 10,000원 보존)

일관성 만족 (10,000 = 5,000+5,000)

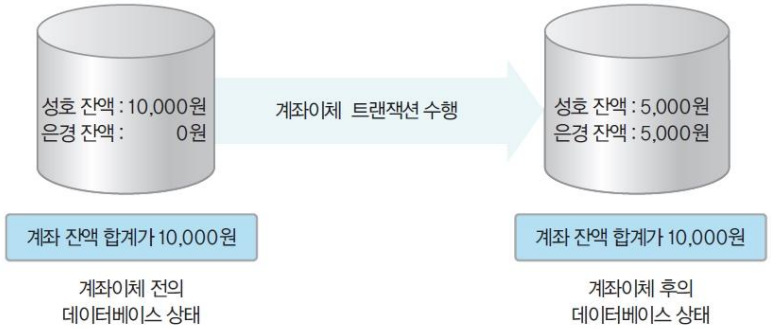


그림 10-6 트랜잭션의 일관성을 만족하는 예

일관성 위반 (10,000 ≠ 5,000+0)

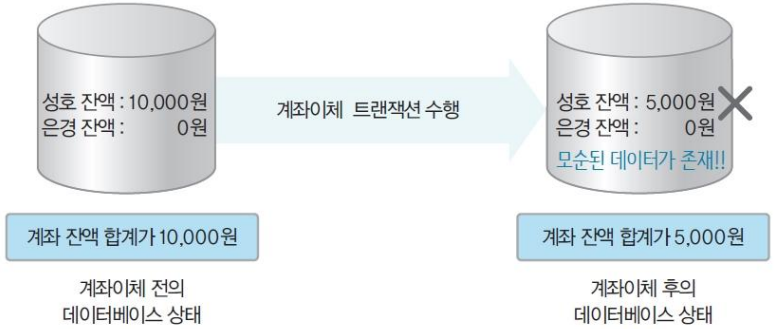


그림 10-7 트랜잭션의 일관성을 만족하지 않는 예

격리성 Isolation (고립성)

수행 중인 트랜잭션이 완료되기 전에는, 그 트랜잭션이 만든 중간 연산 결과에 다른 트랜잭션이 접근할 수 없어야 한다.

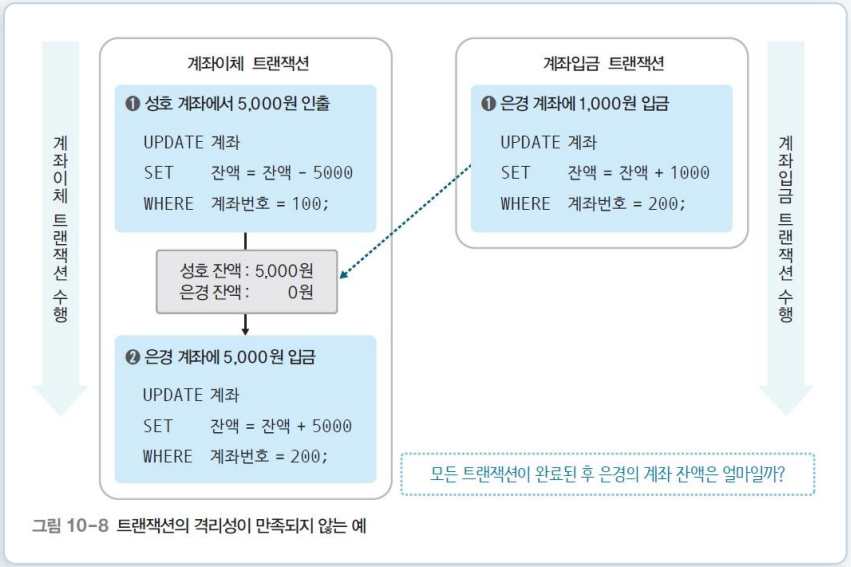


그림 10-8 트랜잭션의 격리성이 만족되지 않는 예

격리성 — 순차 수행으로 해결

격리성 문제는 트랜잭션들을 순서대로 하나씩 수행하는 것처럼 처리하면 해결된다. T1이 끝난 뒤 T2가 실행된다.

트랜잭션 T1 먼저 완료

이후 트랜잭션 T2 수행

그림 10-9 트랜잭션의 격리성이 만족되는 예

지속성 Durability & ACID–DBMS 기능

지속성 (Durability)

- 완료된 트랜잭션의 결과는 어떤 경우에도 손실되지 않고 영구적이어야 함
  - 시스템 장애가 발생해도 결과가 보존되어야 함
- 이를 위해 DB 회복·복구 기능이 반드시 필요함

트랜잭션 특성과 DBMS 기능의 연결

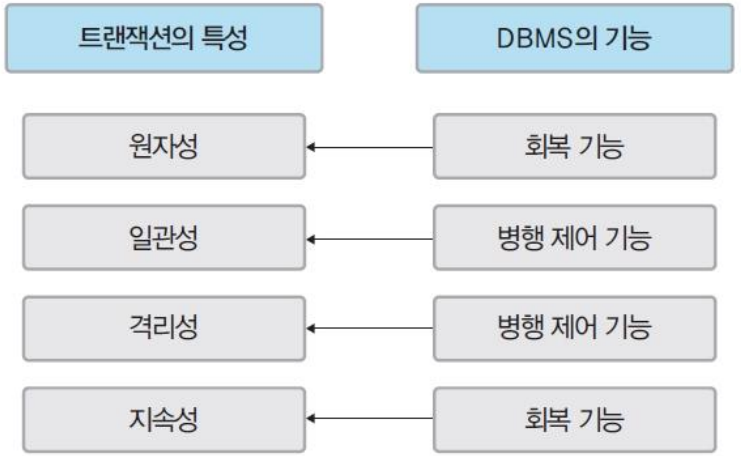


그림 10-10 트랜잭션의 특성과 DBMS의 기능

트랜잭션의 연산 — commit & rollback

commit (작업 완료)

트랜잭션이 성공적으로 수행되었음을 선언 → 최종 DB에 결과를 저장한다.

rollback (작업 취소)

수행을 취소하고 트랜잭션 이전 상태로 되돌린다. 장애가 없어도 사용자가 취소(예: 이체 취소)할 수 있다.

그림 10-11 트랜잭션의 연산

commit 연산을 실행한 예

두 연산이 모두 성공한 뒤 commit을 실행하면, 최종 DB에 이체 결과(성호 5,000원 · 은경 5,000원)가 저장된다.

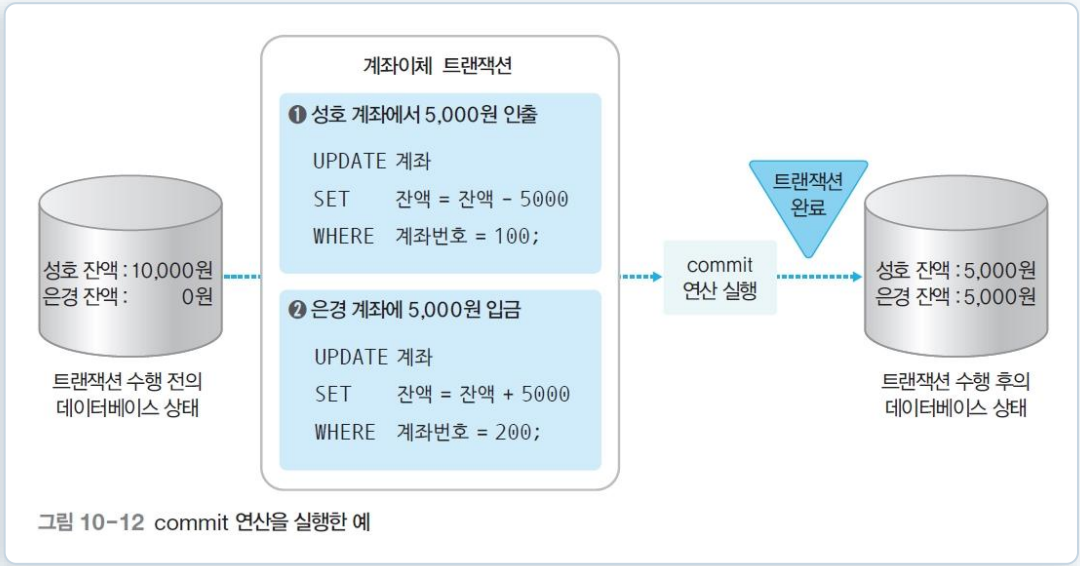


그림 10-12 commit 연산을 실행한 예

rollback 연산을 실행한 예

수행 도중 장애가 발생하거나 사용자가 취소하면, rollback으로 지금까지의 변경을 모두 취소하고 이전 상태로 복귀한다.

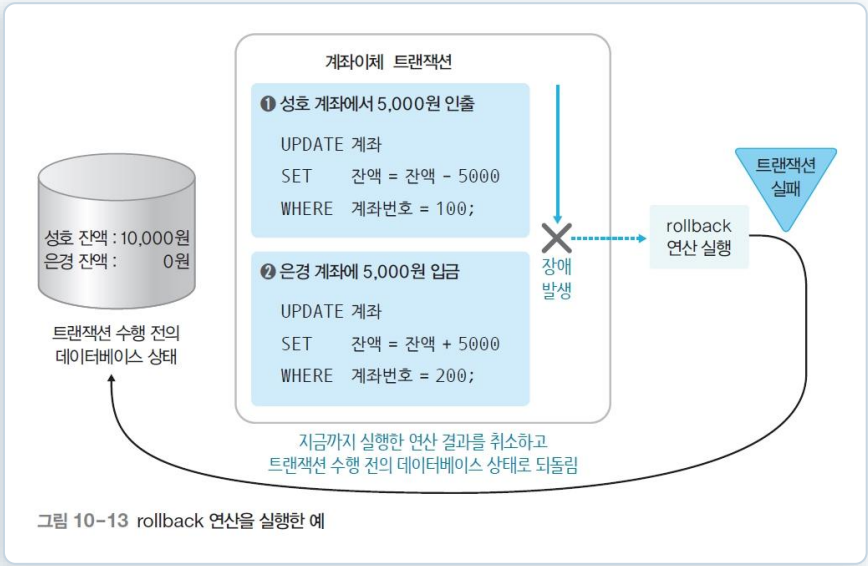


그림 10-13 rollback 연산을 실행한 예

트랜잭션의 다섯 가지 상태

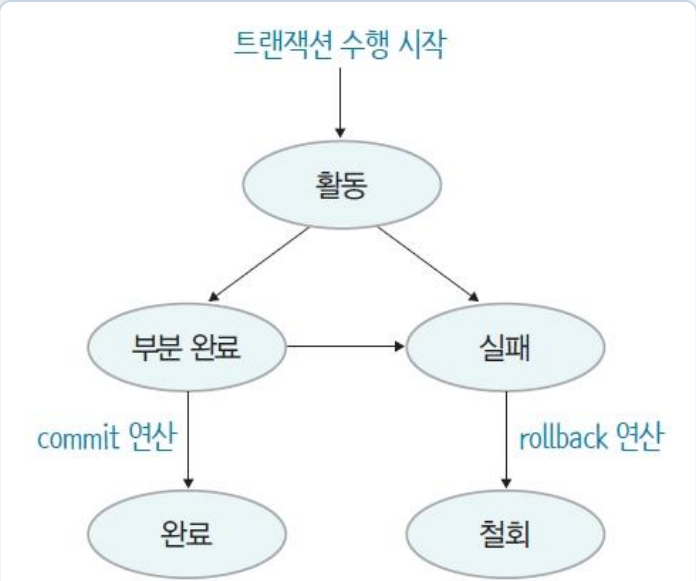


그림 10-14 트랜잭션의 다섯 가지 상태

활동 (active)
트랜잭션이 수행 중인 상태

부분 완료 (partially committed)
연산은 끝났으나 DB에 미반영

완료 (committed)
결과를 DB에 반영하고 종료

실패 (failed)
수행이 중단된 상태

철회 (aborted)
rollback을 실행해 이전 상태로 복귀

02

장애와 회복

Failure & Recovery

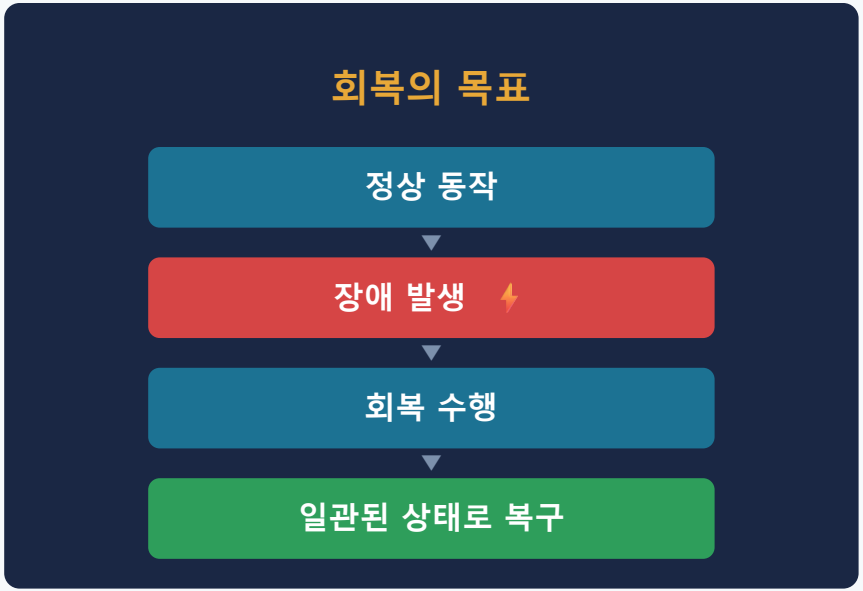
회복(recovery)과 장애(failure)

회복 (Recovery)

장애가 발생했을 때 데이터베이스를 장애 직전의 일관된 상태로 복구시키는 것

장애 (Failure)

시스템이 제대로 동작하지 않는 상태. 원인은 사용자 실수, 정전·H/W 고장, S/W 논리 오류 등 다양하다.



장애(failure)의 유형

장애는 크게 트랜잭션 장애, 시스템 장애, 미디어 장애의 세 가지로 나뉜다.

표 10-1 장애의 유형

유형	설명	
트랜잭션 장애	의미	트랜잭션 수행 중 오류가 발생하여 정상적으로 수행을 계속할 수 없는 상태
	원인	트랜잭션의 논리적 오류, 잘못된 데이터 입력, 시스템 자원의 과다 사용 요구, 처리 대상 데이터의 부재 등
시스템 장애	의미	하드웨어의 결함으로 정상적으로 수행을 계속할 수 없는 상태
	원인	하드웨어 이상으로 메인 메모리에 저장된 정보가 손실되거나 교착 상태가 발생한 경우 등
미디어 장애	의미	디스크 장치의 결함으로 디스크에 저장된 데이터베이스의 일부 혹은 전체가 손상된 상태
	원인	디스크 헤드의 손상이나 고장 등

데이터베이스의 저장 연산 — 저장 장치의 종류

저장 장치는 장애 시 데이터 손실 여부에 따라 휘발성·비휘발성·안정 저장 장치로 구분된다.

표 10-2 저장 장치의 종류

저장 장치	설명	
휘발성volatile 저장 장치 (소멸성)	의미	장애가 발생하면 저장된 데이터가 손실됨
	예	메인 메모리 등
비휘발성nonvolatile 저장 장치 (비소멸성)	의미	장애가 발생해도 저장된 데이터가 손실되지 않음. 단, 디스크 헤더 손상 같은 저장 장치 자체에 이상이 발생하면 데이터가 손실될 수 있음
	예	디스크, 자기 테이프, CD/DVD 등
안정stable 저장 장치	의미	비휘발성 저장 장치를 이용해 데이터 복사본 여러 개를 만드는 방법으로, 어떤 장애가 발생해도 데이터가 손실되지 않고 데이터를 영구적으로 저장할 수 있음

디스크와 메인 메모리 간의 데이터 이동

DB는 디스크에 상주한다. 처리하려면 메모리로 가져와야 한다. input(X): 디스크→메모리, output(X): 메모리→디스크.

블록 단위 이동 (버퍼 ↔ 디스크 블록)

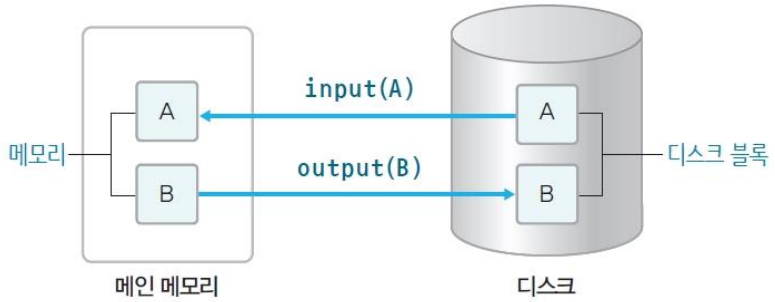


그림 10-15 디스크와 메인 메모리 간의 데이터 이동

input(X) / output(X) 연산

input(X)	디스크 블록에 저장되어 있는 데이터 X를 메인 메모리 버퍼 블록으로 이동시키는 연산
output(X)	메인 메모리 버퍼 블록에 있는 데이터 X를 디스크 블록으로 이동시키는 연산

그림 10-16 디스크와 메인 메모리 간의 데이터 이동 연산

버퍼 블록과 응용 프로그램 변수 간의 이동

read(X): 처리할 데이터를 프로그램 변수로 읽어옴, write(X): 변수 값을 메모리 버퍼 블록의 데이터에 기록.

read(X) / write(X) 연산

read(X)	메인 메모리 버퍼 블록에 저장되어 있는 데이터 X를 프로그램의 변수로 읽어오는 연산
write(X)	프로그램의 변수 값을 메인 메모리 버퍼 블록에 있는 데이터 X에 기록하는 연산

그림 10-17 메인 메모리의 버퍼 블록과 프로그램 변수 간의 데이터 이동 연산

트랜잭션 수행을 위한 전체 데이터 이동

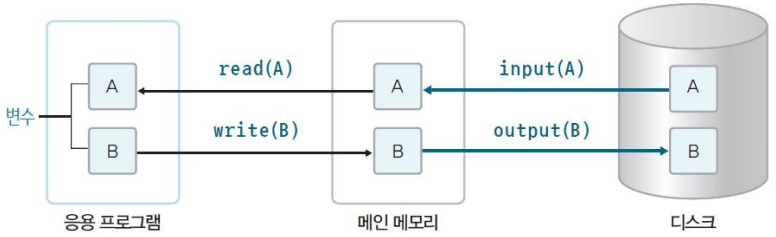


그림 10-18 응용 프로그램이 실행한 트랜잭션의 수행을 위해 필요한 데이터 이동 연산

회복 기법 — 회복 관리자와 복사본

회복 관리자 (recovery manager)

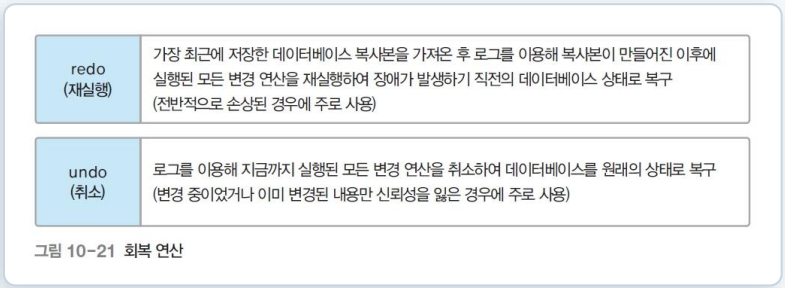
- 장애 발생을 탐지하고 DB 복구 기능을 제공
- 데이터베이스를 빠르게 회복시키는 작업 수행
- 복구의 핵심 원리 = 데이터 중복(복사본)
  - 데이터를 별도 장소에 미리 복사해두고, 장애 시 이를 이용해 원래 상태로 복원

복사본을 만드는 방법

덤프 (dump)	데이터베이스 전체를 다른 저장 장치에 주기적으로 복사하는 방법
로그 (log)	데이터베이스에서 변경 연산이 실행될 때마다 데이터를 변경하기 이전 값과 변경한 이후의 값을 별도의 파일에 기록하는 방법

그림 10-20 데이터베이스 회복을 위해 복사본을 만드는 방법

회복 연산 — redo & undo



redo (재실행)	가장 최근에 저장한 데이터베이스 복사본을 가져온 후 로그를 이용해 복사본이 만들어진 이후에 실행된 모든 변경 연산을 재실행하여 장애가 발생하기 직전의 데이터베이스 상태로 복구 (전반적으로 손상된 경우에 주로 사용)
undo (취소)	로그를 이용해 지금까지 실행된 모든 변경 연산을 취소하여 데이터베이스를 원래의 상태로 복구 (변경 중이었거나 이미 변경된 내용만 신뢰성을 잃은 경우에 주로 사용)

그림 10-21 회복 연산

redo (재실행)

최근 DB 복사본을 가져온 뒤 로그를 이용해 복사본 이후의 변경 연산을 다시 실행 (반영된 결과를 재반영)

undo (취소)

로그를 이용해 트랜잭션이 변경한 연산을 취소하여 변경 이전 값으로 되돌림 (잘못 반영된 결과를 되돌림)

로그(log)와 로그 레코드의 종류

로그 = 데이터 변경 이전 값과 이후 값을 기록한 것. 레코드 단위로, 손실 없는 안정 저장 장치에 기록한다.

표 10-3 로그 레코드의 종류

로그 레코드	설명	
⟨T_i, start⟩	의미	트랜잭션 T_i가 수행을 시작했음을 기록
	예	⟨T_1, start⟩
⟨T_i, X, old_value, new_value⟩	의미	트랜잭션 T_i가 데이터 X를 이전 값 old_value에서 새로운 값new_value으로 변경하는 연산을 실행했음을 기록
	예	⟨T_1, X, 10000, 5000⟩
⟨T_i, commit⟩	의미	트랜잭션 T_i가 성공적으로 완료되었음을 기록
	예	⟨T_1, commit⟩
⟨T_i, abort⟩	의미	트랜잭션 T_i가 철회되었음을 기록
	예	⟨T_1, abort⟩

계좌이체 트랜잭션의 로그 예

트랜잭션 T1이 수행되며 start → 변경(X) → 변경(Y) → commit 순서로 로그 레코드가 차례대로 기록된다.

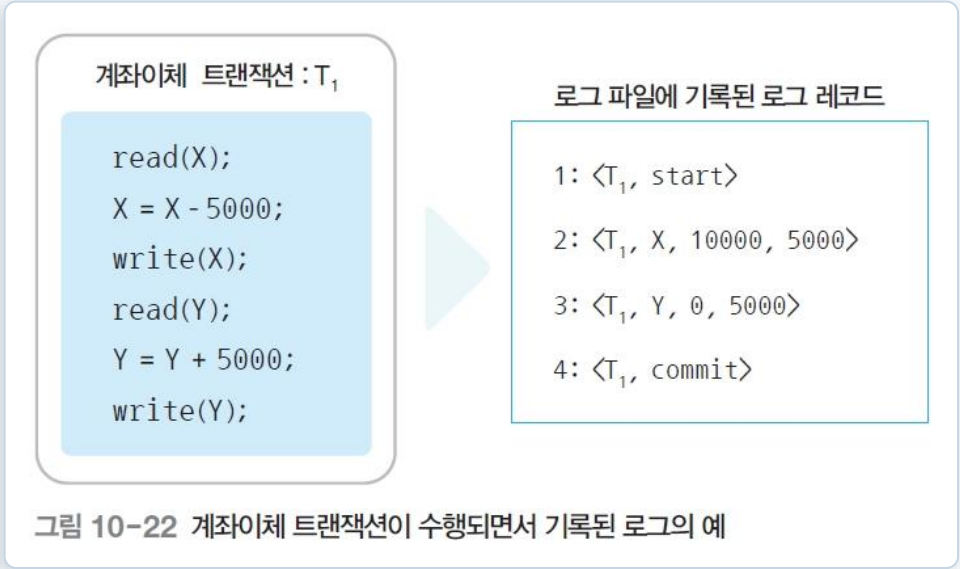


그림 10-22 계좌이체 트랜잭션이 수행되면서 기록된 로그의 예

로그 회복 기법 ① 즉시 갱신(immediate update)

트랜잭션 수행 중 데이터 변경 결과를 DB에 즉시 반영한다. (undo·redo 모두 사용, 기준은 commit 로그 레코드)

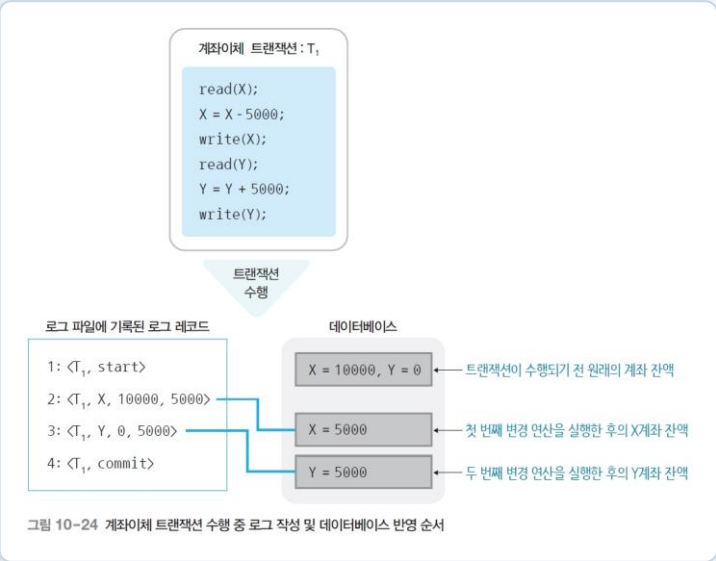


그림 10-24 계좌이체 트랜잭션 수행 중 로그 작성 및 데이터베이스 반영 순서

즉시 갱신 — 데이터베이스 회복 전략

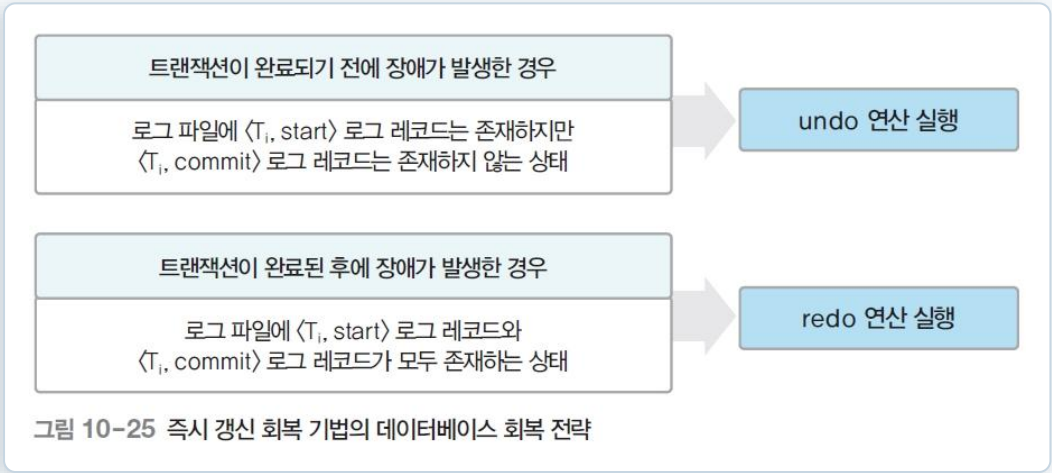


그림 10-25 즉시 갱신 회복 기법의 데이터베이스 회복 전략

undo 실행 (취소)

commit 로그 레코드가 없는 경우 → 트랜잭션이 완료 전에 장애가 났으므로, 변경을 모두 취소한다.

redo 실행 (재실행)

start와 commit 로그가 모두 있는 경우 → 완료된 트랜잭션이므로, 변경을 다시 반영한다.

즉시 갱신 — 두 트랜잭션의 로그와 DB 반영

순차 수행되는 두 트랜잭션

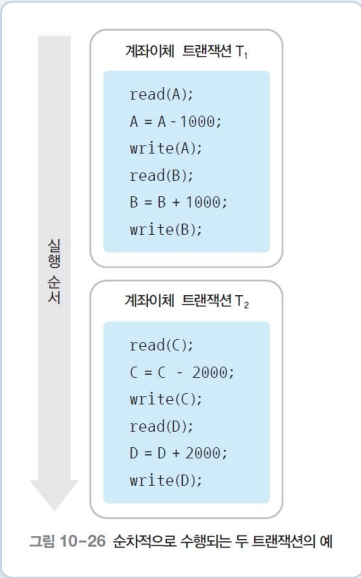


그림 10-26 순차적으로 수행되는 두 트랜잭션의 예

로그 내용과 DB 반영 결과

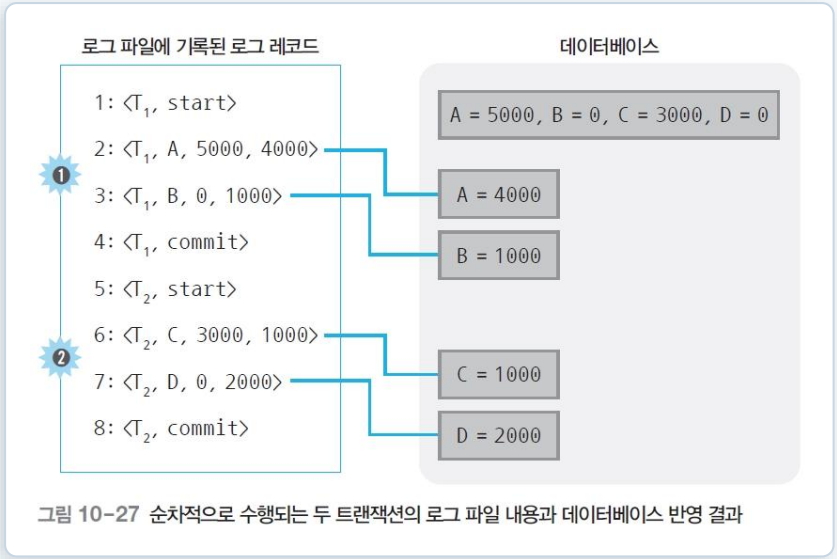


그림 10-27 순차적으로 수행되는 두 트랜잭션의 로그 파일 내용과 데이터베이스 반영 결과

로그 회복 기법 ② 지연 갱신(deferred update)

수행 중에는 변경을 DB에 반영하지 않고 로그에만 기록했다가, 부분 완료 후 한 번에 반영한다. (redo만 사용)

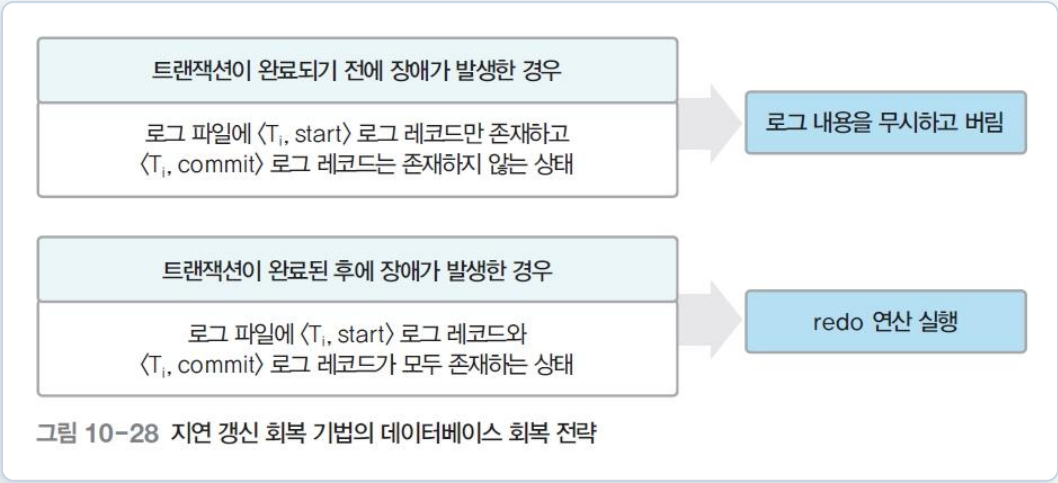


그림 10-28 지연 갱신 회복 기법의 데이터베이스 회복 전략

commit 전 장애

무시 — 아직 DB에 아무것도 반영되지 않았으므로 할 일이 없다.

commit 후 장애

redo 실행 — 로그를 보고 변경을 DB에 다시 반영한다.

지연 갱신 — 로그와 DB 반영 결과

각 변경은 로그에만 먼저 기록되고, commit 이후에 데이터베이스에 반영된다. (undo 불필요, redo만 사용)

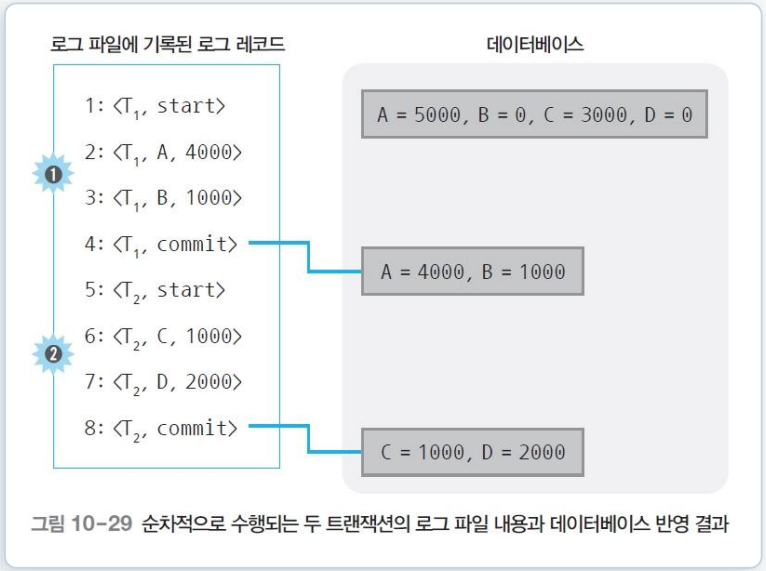


그림 10-29 순차적으로 수행되는 두 트랜잭션의 로그 파일 내용과 데이터베이스 반영 결과

로그 회복 기법 ③ 검사 시점 & 미디어 회복

검사 시점(checkpoint) 회복 기법

- 로그 전체를 분석하는 대신, 일정 간격으로 검사 시점을 만든다
- 검사 시점 이후의 트랜잭션만 대상으로 redo/undo 회복 수행
  - 검사 시점에 메모리의 모든 로그 레코드를 안정 저장 장치에 기록 <checkpoint L>
- **DB 회복 시간을 크게 단축**

미디어(media) 회복 기법

- 디스크 등 비휘발성 저장 장치 장애에 대한 회복
- DB 전체를 주기적으로 다른 안전한 장치에 복사(덤프)
- 덤프로 복구 후, 로그 기반으로 redo 연산 실행
- 단점: 복사 비용·CPU 낭비 / 그럼에도 가장 확실한 대책

03

병행 제어

Concurrency Control

병행 수행과 병행 제어

병행 수행 (concurrency)

- 인터리빙(interleaving): 여러 트랜잭션이 번갈아 가며 수행되는 방식
- 서로 다른 데이터를 쓰면 문제없음
- 같은 데이터에 동시 접근하면 예상치 못한 결과 발생 가능

병행 제어 (concurrency control)

- 여러 트랜잭션이 병행 수행되며 같은 데이터에 접근해도
- **문제없이 정확한 결과를 얻도록 트랜잭션을 제어하는 것**
- 동시 처리 성능과 데이터 정확성을 동시에 보장하는 것이 목표

병행 수행의 문제 ① 갱신 분실(lost update)

기존 트랜잭션이 변경한 결과를 다른 트랜잭션이 덮어써, 기존 변경이 무효화되는 것. 순차 수행하면 올바른 결과가 나온다.

병행 수행 — 갱신 분실 발생

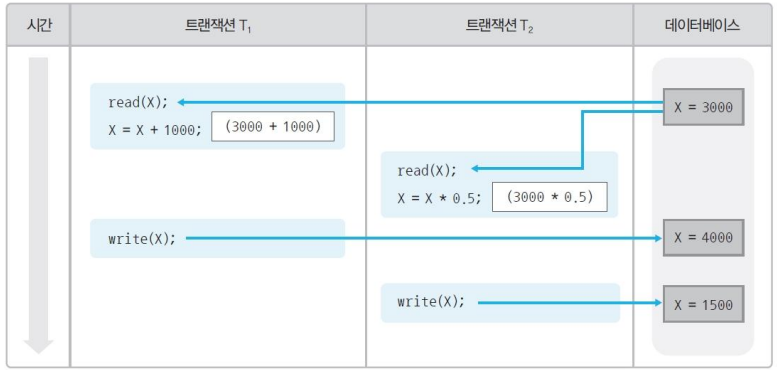


그림 10-30 두 트랜잭션의 병행 수행으로 발생한 갱신 분실의 예

T1 → T2 순차 수행 (정상)

그림 10-31 트랜잭션 T_1을 수행한 후 트랜잭션 T_2를 수행한 결과

병행 수행의 문제 ② 모순성(inconsistency)

하나의 트랜잭션이 여러 변경 연산을 실행할 때, 일관성 없는 중간 상태의 데이터를 읽어 모순된 결과를 내는 것.

병행 수행 — 모순성 발생

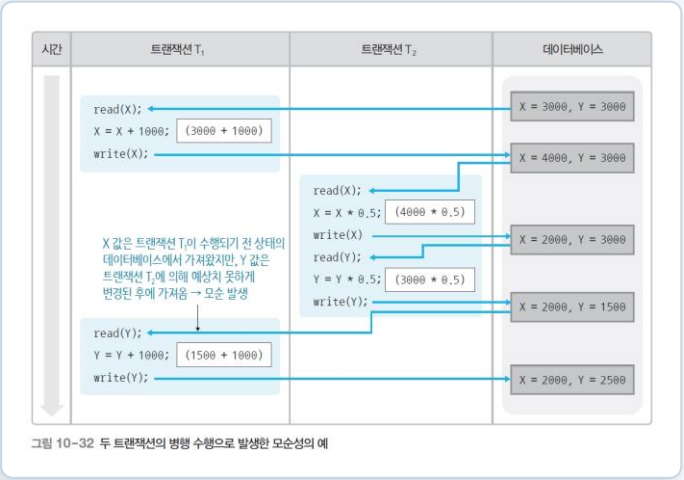


그림 10-32 두 트랜잭션의 병행 수행으로 발생한 모순성의 예

T1 → T2 순차 수행 (정상)

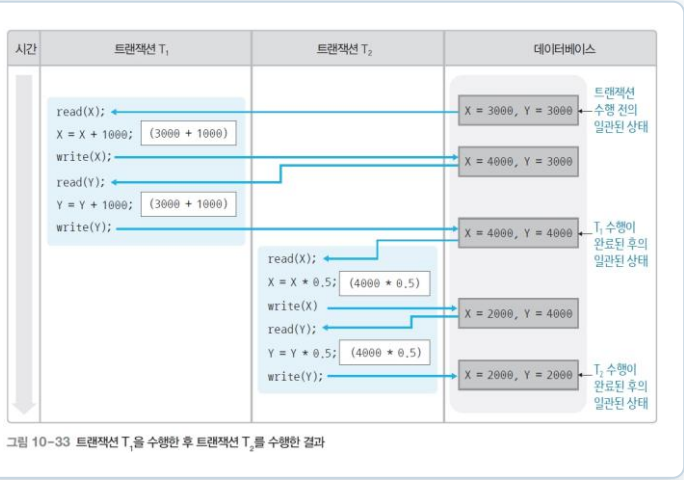


그림 10-33 트랜잭션 T_1을 수행한 후 트랜잭션 T_2를 수행한 결과

병행 수행의 문제 ③ 연쇄 복귀(cascading rollback)

트랜잭션이 rollback할 때, 그 데이터를 가져다 쓴 다른 트랜잭션도 함께 rollback해야 하는 것.

병행 수행 — 연쇄 복귀 발생

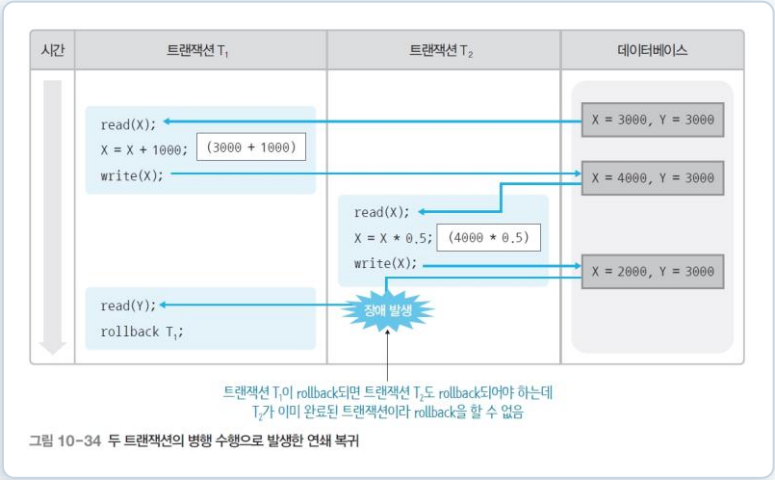


그림 10-34 두 트랜잭션의 병행 수행으로 발생한 연쇄 복귀

T1 → T2 순차 수행 (정상)

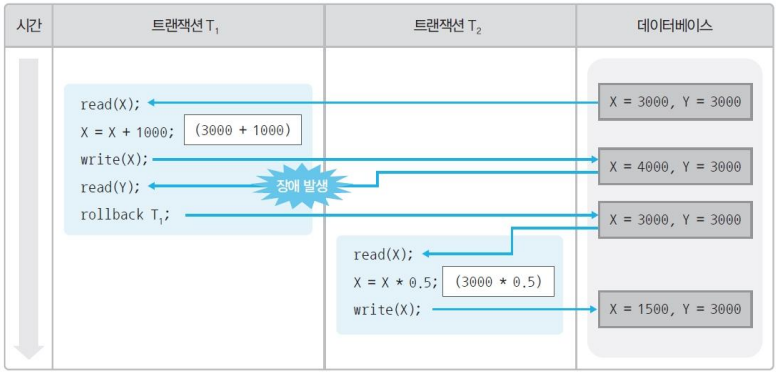


그림 10-35 트랜잭션 T_1을 수행한 후 트랜잭션 T_2를 수행한 결과

트랜잭션 스케줄의 유형

스케줄 = 여러 트랜잭션의 연산을 실행하는 순서. 직렬·비직렬·직렬 가능 스케줄로 나뉜다.

표 10-4 트랜잭션 스케줄의 유형

트랜잭션 스케줄	의미
직렬 스케줄	인터리빙 방식을 이용하지 않고 트랜잭션별로 연산들을 순차적으로 실행시키는 것
비직렬 스케줄	인터리빙 방식을 이용하여 트랜잭션들을 병행해서 수행시키는 것
직렬 가능 스케줄	직렬 스케줄과 같이 정확한 결과를 생성하는 비직렬 스케줄

직렬 스케줄(serial schedule)

인터리빙 없이 트랜잭션별로 연산을 순차 실행. 실행 순서에 따라 결과가 달라지며, 일반적으로 잘 쓰지 않는다.

T1 → T2 순서로 수행

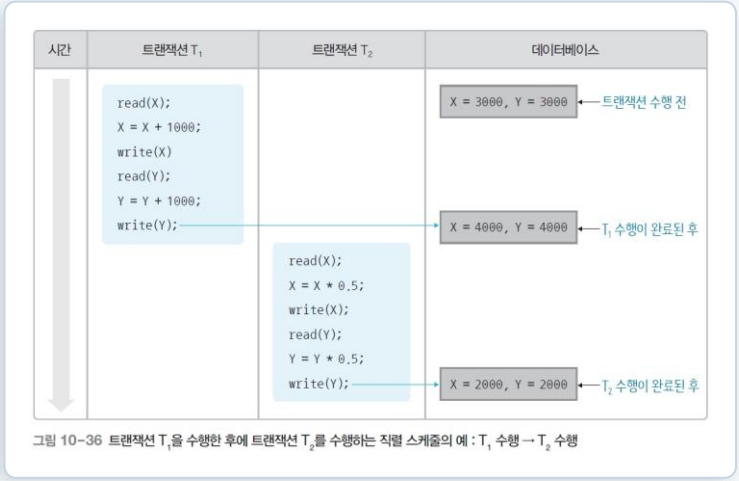


그림 10-36 트랜잭션 T_1을 수행한 후에 트랜잭션 T_2를 수행하는 직렬 스케줄의 예 : T_1 수행 → T_2 수행

T2 → T1 순서로 수행

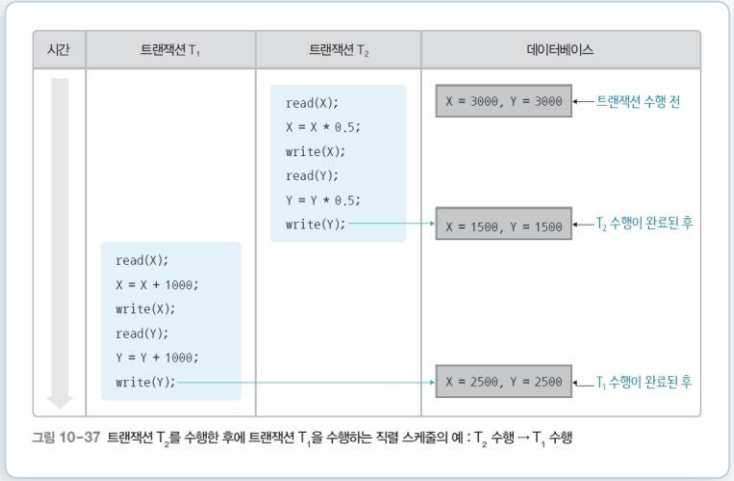


그림 10-37 트랜잭션 T_2를 수행한 후에 트랜잭션 T_1을 수행하는 직렬 스케줄의 예 : T_2 수행 → T_1 수행

비직렬 스케줄(nonserial schedule)

인터리빙으로 트랜잭션을 병행 수행. 순서에 따라 갱신 분실·모순성 등이 생길 수 있어 정확성을 보장할 수 없다.

정확한 결과를 내는 예

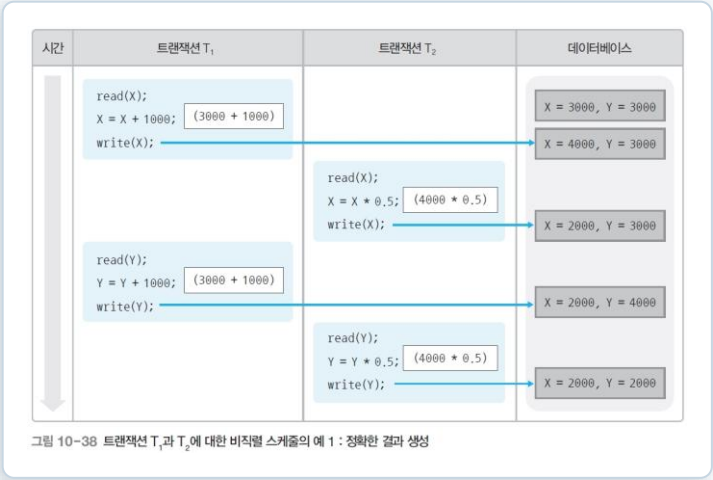


그림 10-38 트랜잭션 T_1과 T_2에 대한 비직렬 스케줄의 예 1 : 정확한 결과 생성

잘못된 결과를 내는 예

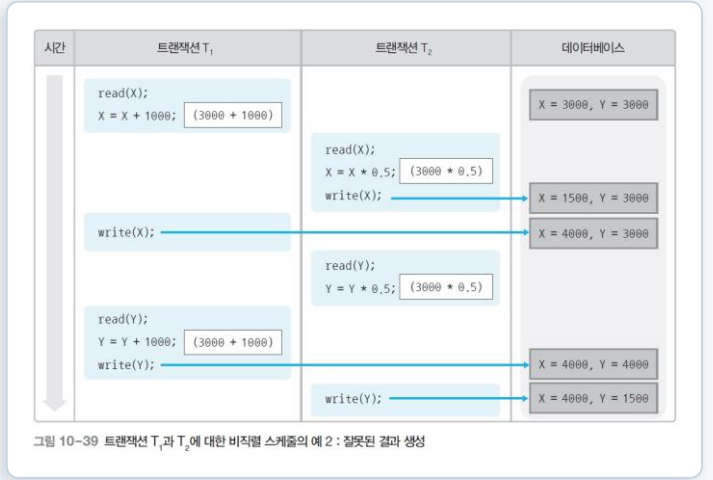


그림 10-39 트랜잭션 T_1과 T_2에 대한 비직렬 스케줄의 예 2 : 잘못된 결과 생성

직렬 가능 스케줄(serializable schedule)

직렬 스케줄을 수행한 것처럼 정확한 결과를 내는 비직렬 스케줄

병행 수행(성능)과 직렬 수행(정확성)의 장점을 모두 가진다.

왜 어려운가?

가능한 스케줄이 매우 많아, 직렬 가능 스케줄인지 일일이 찾아내는 것은 쉽지 않다.

그래서 필요한 것

직렬 가능성을 자동으로 보장해 주는 실용적 규칙이 필요 → 로킹(locking) 기법.

병행 제어 기법 — 로킹(locking)

로킹 (locking) 기법

- 병행 트랜잭션들이 같은 데이터에 동시에 접근하지 못하게 함
- 상호 배제(mutual exclusion)로 직렬 가능성을 보장
- lock / unlock 연산으로 제어하며, 데이터에 대한 독점권을 가짐
- **두 종류: 공용 lock (읽기) · 전용 lock (쓰기)**

lock 연산의 종류 (표 10-5)

표 10-5 lock 연산

연산	설명
공용shared lock	트랜잭션이 데이터에 대해 공용 lock 연산을 실행하면, 해당 데이터에 read 연산을 실행할 수 있지만 write 연산은 실행할 수 없다. 그리고 해당 데이터에 다른 트랜잭션도 공용 lock 연산을 동시에 실행할 수 있다. (데이터에 대한 사용권을 여러 트랜잭션이 함께 가질 수 있음)
전용exclusive lock	트랜잭션이 데이터에 전용 lock 연산을 실행하면 해당 데이터에 read 연산과 write 연산을 모두 실행할 수 있다. 그러나 해당 데이터에 다른 트랜잭션은 공용이든 전용이든 어떤 lock 연산도 실행할 수 없다. (전용 lock 연산을 실행한 트랜잭션만 해당 데이터에 대한 독점권을 가질 수 있음)

로킹의 규약과 적용 범위

로킹의 규약

- read/write 연산 전에 반드시 lock 연산을 실행
- lock으로 데이터 독점권을 얻어 다른 트랜잭션의 접근 차단
- 모든 연산 후 unlock으로 독점권을 반납
  - 공용 lock끼리는 양립 가능, 전용 lock은 unlock 후에만 접근 가능

로킹 단위(적용 범위)의 선택

크게 잡으면 (전체 DB)

- 제어가 간단함
- 하나의 트랜잭션만 허용 → 병행성 낮음

작게 잡으면 (튜플·속성)

- 많은 트랜잭션의 병행 수행 가능
- 제어가 복잡함

lock 연산의 양립성

표 10-6 lock 연산의 양립성

	공용 lock	전용 lock
공용 lock	가능	불가능
전용 lock	불가능	불가능

핵심: 읽기끼리만 공유, 쓰기가 끼면 독점

공용(S) lock — 양립 가능

서로 다른 트랜잭션이 같은 데이터에 공용 lock을 동시에 걸 수 있다. (읽기는 충돌하지 않음)

전용(X) lock — 양립 불가

전용 lock이 걸린 데이터에는 공용·전용 어떤 lock도 걸 수 없다. 반드시 unlock 후에 접근 가능하다.

기본 로킹 규약으로 직렬 가능성이 보장되지 않는 경우

⚠ lock·unlock만 지켜도, 너무 일찍 unlock하면 다른 트랜잭션이 일관성 없는 데이터에 접근해 문제가 생긴다.

그림 10-40 기본 로킹 규약으로 직렬 가능성이 보장되지 않는 스케줄의 예

2단계 로킹(2PL) 규약

2PL의 두 단계 전략

확장 단계	트랜잭션이 lock 연산만 실행할 수 있고, unlock 연산은 실행할 수 없는 단계
축소 단계	트랜잭션이 unlock 연산만 실행할 수 있고, lock 연산은 실행할 수 없는 단계

그림 10-41 2단계 로킹 규약의 전략

2PL로 직렬 가능성이 보장된 예

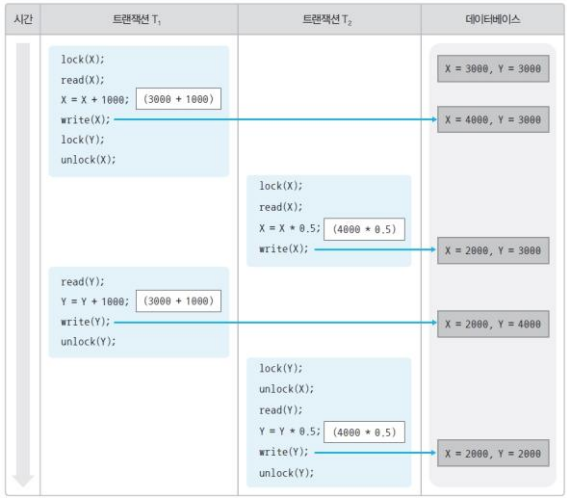


그림 10-42 2단계 로킹 규약으로 직렬 가능성이 보장된 스케줄의 예

교착 상태(deadlock)

2PL로 직렬 가능성이 보장되더라도, 서로가 가진 lock을 기다리며 영원히 멈추는 교착 상태가 발생할 수 있다.

트랜잭션 T1
데이터 X를 기다림
(T2가 lock 보유)

↔ 서로 무한 대기 ↔

트랜잭션 T2
데이터 Y를 기다림
(T1이 lock 보유)

대책: 교착 상태가 생기지 않게 미리 예방하거나, 빨리 탐지하여 한쪽을 rollback시켜 푸는 것이 최선이다.

WRAP-UP

배운 것들을 정리해 봅시다

트랜잭션	DB의 논리적 작업 단위. ACID(원자성·일관성·격리성·지속성)를 보장해야 한다.
장애와 회복	로그를 이용해 장애 직전 상태로 복구. 즉시 갱신(undo·redo), 지연 갱신(redo), 검사 시점, 미디어 회복.
병행 제어	병행 수행의 문제(갱신 분실·모순성·연쇄 복귀)를 로킹으로 해결. 2PL이 직렬 가능성을 보장하나 교착 상태에 유의.

NEXT CLASS

다음 시간 예고

11장	데이터베이스 보안과 권한 관리
복습	ACID · 로그 회복 · 2단계 로킹 다시 정리
과제	계좌이체 트랜잭션의 로그를 직접 작성해 보기

감사합니다

Thank You

데이터베이스 시스템 · 10장 회복과 병행 제어